ഹയർ സെക്കന്ററി കോഴ്സ്

XI

കമ്പ്യൂട്ടർ സയൻസ്
ഭാഗം - II

കേരളസർക്കാർ
പൊതുവിദ്യാഭ്യാസ വകുപ്പ്

സംസ്ഥാന വിദ്യാഭ്യാസ ഗവേഷണ പരിശീലന സമിതി (SCERT); കേരളം
2019

ദേശീയഗാനം

ജനഗണമന അധിനായക ജയഹേ
ഭാരത ഭാഗ്യവിധാതാ,
പഞ്ചാബസിന്ധു ഗുജറാത്ത മറാഠാ
ദ്രാവിഡ ഉത്ക്കല ബംഗാ,
വിന്ധ്യഹിമാചല യമുനാഗംഗാ,
ഉച്ഛ്വല ജലധിതരംഗാ,
തവശുഭനാമേ ജാഗേ,
തവശുഭ ആശിഷ മാഗേ,
ഗാഹേ തവ ജയ ഗാഥാ
ജനഗണമംഗലദായക ജയഹേ
ഭാരത ഭാഗ്യവിധാതാ
ജയഹേ, ജയഹേ, ജയഹേ,
ജയ ജയ ജയ ജയഹേ!

പ്രതിജ്ഞ

ഇന്ത്യ എന്റെ രാജ്യമാണ്. എല്ലാ ഇന്ത്യക്കാരും എന്റെ സഹോദരീ സഹോദരന്മാരാണ്.

ഞാൻ എന്റെ രാജ്യത്തെ സ്നേഹിക്കുന്നു; സമ്പൂർണവും വൈവിധ്യപൂർണവുമായ അതിന്റെ പാരമ്പര്യത്തിൽ ഞാൻ അഭിമാനം കൊള്ളുന്നു.

ഞാൻ എന്റെ മാതാപിതാക്കളെയും ഗുരുക്കന്മാരെയും മുതിർന്നവരെയും ബഹുമാനിക്കും.

ഞാൻ എന്റെ രാജ്യത്തിന്റെയും എന്റെ നാട്ടുകാരുടെയും ക്ഷേമത്തിനും ഐശ്വര്യത്തിനും വേണ്ടി പ്രയത്നിക്കും.

Prepared by:
State Council of Educational Research and Training (SCERT)
Poojappura, Thiruvananthapuram 695012, Kerala
Website : www.scertkerala.gov.in *e-mail* : scertkerala@gmail.com
Phone : 0471 - 2341883, Fax : 0471 - 2341869
Typesetting and Layout : SCERT
© Department of Education, Government of Kerala

ആമുഖം

ഏതു വിജ്ഞാനവും മാതൃഭാഷയിൽ പഠിക്കാനും പ്രകാശനം ചെയ്യാനും സാധിക്കും. അതിനുള്ള അവസരം പഠിതാക്കൾക്ക് ഒരുക്കേണ്ടത്, ഏതൊരു പഠന സമ്പ്രദായത്തിന്റെയും അനിവാര്യതയാണ്. അതിന്റെ തുടക്കമെന്ന നിലയ്ക്കാണ് ഹയർസെക്കന്ററി തലത്തിൽ ഭാഷേതര വിഷയങ്ങളിലെ പാഠപുസ്തകങ്ങൾ മലയാളത്തിൽ പ്രസിദ്ധീകരിക്കുന്നത്.

മാതൃഭാഷയിലൂടെയുള്ള വിദ്യാഭ്യാസം, ജ്ഞാനസമ്പാദനത്തിനുള്ള സുഗമമാർഗം എന്നതിനോടൊപ്പം സാംസ്കാരികത്തനിമയുടെ തിരിച്ചറിയൽ കൂടിയാണ്. അതുകൊണ്ടാണ് വികസിതരാജ്യങ്ങൾ മാതൃഭാഷയെ മുഖ്യബോധന മാധ്യമമായി സ്വീകരിച്ചിരിക്കുന്നത്. ഇന്ത്യയിലാകട്ടെ, ദേശീയതലത്തിലുള്ള പ്രധാന പരീക്ഷകളെല്ലാം പ്രാദേശിക ഭാഷകളിൽക്കൂടി നടത്തുന്നതിനുള്ള സംവിധാനവും ഉണ്ടായി വരികയാണ്. ഈയൊരു സാഹചര്യത്തിൽ നമ്മുടെ കുട്ടികളും മാതൃഭാഷയുടെ ശക്തിസൗന്ദര്യങ്ങൾ തിരിച്ചറിഞ്ഞ് വിവിധ വിഷയങ്ങളിൽ ജ്ഞാനനിർമിതിയിൽ ഏർപ്പെടേണ്ടതുണ്ട്. അതിന് അവരെ സജ്ജരാക്കുകയാണ് ഈ പാഠപുസ്തകങ്ങളുടെ മുഖ്യ ലക്ഷ്യം.

പരിഭാഷപ്പെടുത്തിയ പുസ്തകങ്ങളിൽ അതത് വിഷയങ്ങളിലെ സാങ്കേതിക പദങ്ങൾ പരമാവധി മലയാളത്തിലാക്കിയിട്ടുണ്ട്. നമ്മുടെ ഭാഷയിൽ ചിരപരിചിതമായ ഇംഗ്ലീഷ് പദങ്ങളെ അതേപടി സ്വീകരിച്ചിട്ടുമുണ്ട്. വിവർത്തനത്തിന് തീർത്തും വഴങ്ങാത്ത പദങ്ങളെ അതേരീതിയിൽ തന്നെ ഉപയോഗിച്ചിരിക്കുന്നു. മാതൃഭാഷയിൽ പഠിക്കുന്നവർക്ക് ആശയഗ്രഹണം സുഗമമാക്കുന്ന വിധത്തിലാണ് പാഠപുസ്തകരചന നടത്തിയിരിക്കുന്നത്. അതോടൊപ്പം മലയാളഭാഷയുടെ വളർച്ചയ്ക്കും ഈ പ്രവർത്തനം സഹായകമാകുമെന്ന് കരുതുന്നു.

പാഠപുസ്തകവിവർത്തന രംഗത്ത് നമ്മുടെ രാജ്യത്ത് നടന്ന വലിയൊരു കാൽവെപ്പാണ് ഇത്. പ്രഥമ സംരംഭമെന്നനിലയിൽ പല പരിമിതികളും പരിഭാഷയിൽ വന്നിട്ടുണ്ടാകാം. ക്ലാസ്മുറിയിൽ പ്രയോഗത്തിൽ വരുമ്പോഴാണ് അവയെല്ലാം കൂടുതൽ ബോധ്യപ്പെടുക. തുടർന്ന് വരുന്ന ഘട്ടങ്ങളിൽ അവയൊക്കെ പരിഹരിക്കുന്നതിന് എല്ലാ അഭ്യുദയകാംക്ഷികളിൽ നിന്നും വിശിഷ്യ അധ്യാപകർ, വിദ്യാർത്ഥികൾ എന്നിവരിൽ നിന്നും അഭിപ്രായങ്ങളും നിർദേശങ്ങളും പ്രതീക്ഷിക്കുന്നു.

ഡോ.ജെ. പ്രസാദ്

ഡയറക്ടർ,

എസ്.സി.ഇ.ആർ.ടി. കേരളം

പാഠപുസ്തക നിർമാണ സമിതി

കമ്പ്യൂട്ടർ സയൻസ്

ശ്രീ. ജോയി ജോൺ
എച്ച്.എസ്.എസ്.ടി, സെന്റ്. ജോസഫ്
എച്ച്.എസ്.എസ്. തിരുവനന്തപുരം

ശ്രീ. അസീസ് വി.
എച്ച്.എസ്.എസ്.ടി, ജി.എച്ച്.എസ്.എസ്
വെള്ളിയോട്, കോഴിക്കോട്

ശ്രീ. റോയ് ജോൺ
എച്ച്.എസ്.എസ്.ടി, അലോഷ്യസ് എച്ച്.എസ്.എസ്.
എൽതുരുത്ത്, തൃശൂർ

ശ്രീ. അബൂബക്കർ പി.
എച്ച്.എസ്.എസ്.ടി, ഗവ. ജി.എച്ച്.എസ്.എസ്.
ചാലപ്പുറം, കോഴിക്കോട്

ശ്രീ. ഷാജൻ ജോസ് എൻ.
എച്ച്.എസ്.എസ്.ടി, സെന്റ്. ജോസഫ്
എച്ച്.എസ്.എസ്. പാവറട്ടി, തൃശൂർ

ശ്രീ. അഫ്സൽ കെ.എ.
എച്ച്.എസ്.എസ്.ടി, ജി. എച്ച്.എസ്.എസ്. ശിവപുരം,
കരിയാത്തൻകരേ പി.ഒ., കോഴിക്കോട്

ശ്രീ. പ്രശാന്ത് പി.എം.
എച്ച്.എസ്.എസ്.ടി, സെന്റ്. ജോസഫ് ബോയ്സ്
എച്ച്.എസ്.എസ്. കോഴിക്കോട്

ശ്രീ. വിനോദ് വി.
എച്ച്.എസ്.എസ്.ടി, എൻ.എസ്.എസ്.
എച്ച്.എസ്.എസ്., പ്രാക്കുളം, കൊല്ലം

ശ്രീ. രാജമോഹൻ സി.
എച്ച്.എസ്.എസ്.ടി, നവമുകുന്ദ എച്ച്.എസ്.എസ്.
തിരുനാവായ, മലപ്പുറം

ശ്രീ. എ.എസ്. ഇസ്മൈൽ
എച്ച്.എസ്.എസ്.ടി, ഗവ. എച്ച്.എസ്.എസ്.
പ്ലാപ്പെട്ടി, മലപ്പുറം

ശ്രീ. സുനിൽ കാര്യത്തൻ
എച്ച്.എസ്.എസ്.ടി, ഗവ. ബ്രണ്ണൻ എച്ച്.എസ്.എസ്.,
തലശ്ശേരി

ശ്രീ. സായ് പ്രകാശ് എസ്.
എച്ച്.എസ്.എസ്.ടി, സെന്റ്. തോമസ് എച്ച്.എസ്.എസ്.
പൂന്തുറ, തിരുവനന്തപുരം

വിദഗ്ധർ

ഡോ. ലജീഷ് വി.എൽ.
അസിസ്റ്റന്റ് പ്രൊഫസർ, ഡിപ്പാർട്ട്മെന്റ് ഓഫ്
കമ്പ്യൂട്ടർ സയൻസ്, കാലിക്കറ്റ് യൂണിവേഴ്സിറ്റി

ഡോ. മധു എസ്. നായർ
അസിസ്റ്റന്റ് പ്രൊഫസർ, ഡിപ്പാർട്ട്മെന്റ് ഓഫ്
കമ്പ്യൂട്ടർ സയൻസ്, കേരള സർവകലാശാല

ശ്രീ. മധു വി.ടി.
ഡയറക്ടർ, കമ്പ്യൂട്ടർ സെന്റർ,
കാലിക്കറ്റ് യൂണിവേഴ്സിറ്റി

ഡോ. ബിനു പി. ചാക്കോ
അസോസിയേറ്റ് പ്രൊഫസർ, ഡിപ്പാർട്ട്മെന്റ് ഓഫ്
കമ്പ്യൂട്ടർ സയൻസ്, പ്രജ്യോതി നികേതൻ കോളേജ്,
പുതുക്കാട്

ഡോ. സുശീൽ കുമാർ ആർ.
അസോസിയേറ്റ് പ്രൊഫസർ, ഡിപ്പാർട്ട്മെന്റ് ഓഫ്
ഇംഗ്ലീഷ്, ഡി.ബി. കോളേജ്, ശാസ്താംകോട്ട

ഡോ. വിനീത് കെ. പലേരി
പ്രൊഫസർ, ഡിപ്പാർട്ട്മെന്റ് ഓഫ് സയൻസ് ആന്റ്
എഞ്ചിനീയറിംഗ്, എൻ.ഐ.ടി., കോഴിക്കോട്

മഹേശ്വരൻ നായർ. വി.
സബ് ഡിവിഷണൽ എഞ്ചിനീയർ, റീജിയണൽ
ടെലികോം ട്രെയിനിങ് സെന്റർ, തിരുവനന്തപുരം

ആർട്ടിസ്റ്റ്

സുധീർ വൈ   വിനീത് വി

അക്കാദമിക് കോർഡിനേറ്റർ

ഡോ. മീന എസ്.
റിസർച്ച് ഓഫീസർ, എസ്.സി.ഇ.ആർ.ടി

ശ്രീമതി. ജാൻസി റാണി എ.കെ.
റിസർച്ച് ഓഫീസർ, എസ്.സി.ഇ.ആർ.ടി

പാഠപുസ്തക പരിഭാഷ സമിതി
(മലയാളം)

ഡോ. ബിനു പി ചാക്കോ
അസോസിയേറ്റ് പ്രൊഫസർ, ഡിപ്പാർട്ട്മെന്റ് ഓഫ് കമ്പ്യൂട്ടർ സയൻസ്
പ്രജോതി നികേതൻ കോളേജ്, പുതുക്കാട്

ഡോ. ഗ്ലാഡ്സ്റ്റൺ എസ്.രാജ്
അസിസ്റ്റന്റ് പ്രൊഫസർ & ഹെഡ് ഡിപ്പാർട്ട്മെന്റ് ഓഫ് കമ്പ്യൂട്ടർ സയൻസ്
ഗവ. കോളേജ്, നെടുമങ്ങാട്

ശ്രീ. എ.എസ്. ഇസ്മൈൽ
എച്ച്.എസ്.എസ്.ടി, എച്ച്.എസ്.എസ്.
പ്ലാപ്പെട്ടി, മലപ്പുറം

ശ്രീ. മുകേഷ് വി.കെ
ജി.എച്ച്.എസ്.എസ്.ടി, കോക്കലൂർ

ശ്രീ. വിനയചന്ദ്രൻ സി
എച്ച്.എസ്.എസ്.ടി, വി.ജി.എച്ച്.എസ്.എസ്.
അംമ്പികോദയം എച്ച്.എസ്.എസ്
നെടിയവിള, കൊല്ലം

ശ്രീമതി. വിദ്യ പി.എസ്.
എച്ച്.എസ്.എസ്.ടി, ഗവ. നളന്ദ എച്ച്.എസ്.എസ്.
കിഴുപില്ലിക്കര, തൃശൂർ

ശ്രീ. ശ്രീജിത്ത് പി.
സി.ജെ.എച്ച്.എസ്.എസ്., ചെമ്മനാട്, കാസർഗോഡ്

ശ്രീ. ഹരി കെ
ഗവ. വി ആന്റ് എച്ച്.എസ്.എസ്. കടപ്പുറം
ചാവക്കാട്, തൃശൂർ

അക്കാദമിക് കോർഡിനേറ്റർ

ശ്രീമതി റിയാന അൻസാരി
റിസർച്ച് ഓഫീസർ, എസ്.സി.ഇ.ആർ.ടി

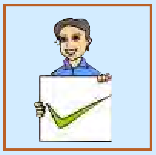
ഉള്ളടക്കം

യൂണിറ്റ് 7	നിയന്ത്രണ പ്രസ്താവനകൾ	219
യൂണിറ്റ് 8	അറേകൾ	269
യൂണിറ്റ് 9	സ്ട്രിങ് കൈകാര്യം ചെയ്യലും ഇൻപുട്ട്/ ഔട്ട്പുട്ട് ഫങ്ഷനുകളും	299
യൂണിറ്റ് 10	ഫങ്ഷനുകൾ	315
യൂണിറ്റ് 11	കമ്പ്യൂട്ടർ ശൃംഖലകൾ	357
യൂണിറ്റ് 12	ഇന്റർനെറ്റും മൊബൈൽ കമ്പ്യൂട്ടിങ്ങും	397

A6E8V4

പാഠപുസ്തകത്തിൽ ഉപയോഗിച്ചിരിക്കുന്ന സൂചനകൾ

നമുക്ക് ചെയ്യാം



നിങ്ങളുടെ പുരോഗതി അറിയുക

ഇൻഫർമേഷൻ ബോക്സ്

നമുക്ക് പരിശീലിക്കാം

നമുക്ക് സംഗ്രഹിക്കാം

7

നിയന്ത്രണ പ്രസ്താവനകൾ

പ്രധാന ആശയങ്ങൾ

- തീരുമാനം എടുക്കുന്നതിനുള്ള പ്രസ്താവനകൾ
  - `if` പ്രസ്താവന
  - `if .. else` പ്രസ്താവന
  - നെസ്റ്റഡ് `if`
  - `else if` ലാഡർ
  - `switch` പ്രസ്താവന
  - കൺഡീഷണൽ ഓപ്പറേറ്റർ
- ആവർത്തന പ്രസ്താവനകൾ
  - `while` പ്രസ്താവന
  - `for` പ്രസ്താവന
  - `do .. while` പ്രസ്താവന
  - ലൂപ്പുകളുടെ നെസ്റ്റിംഗ്
- ജമ്പ് പ്രസ്താവന
  - `go to`
  - `breack`
  - `continue`

H2D7S6

ഇൻപുട്ട്, ഔട്ട്പുട്ട്, വില നൽകൽ എന്നിവ ചെയ്യുന്നതതിനുള്ള C++-ലെ നിർവഹണ പ്രസ്താവനകൾ കഴിഞ്ഞ അധ്യായങ്ങളിൽ നാം ചർച്ച ചെയ്തു. ഇതുപയോഗിച്ച് ലളിതമായ പ്രോഗ്രാമുകൾ എങ്ങനെ എഴുതുവാൻ കഴിയുമെന്ന് നമുക്കറിയാം. ഈ പ്രോഗ്രാമുകളുടെ നിർവഹണം അനുക്രമമാണ്. അതായത്, പ്രോഗ്രാമിലെ ഓരോ നിർദ്ദേശവും ഒന്നിന് പുറകെ ഒന്നായി പ്രവർത്തിക്കുന്നു. ഈ അധ്യായത്തിൽ C++-ലെ പ്രോഗ്രാമിന്റെ തനത് പ്രവർത്തനക്രമത്തിന് മാറ്റം വരുത്തുന്ന പ്രസ്താവനകളെ കുറിച്ചാണ് നാം ചർച്ച ചെയ്യുന്നത്. അധ്യായം 3 ൽ നാം ചർച്ച ചെയ്ത തെരഞ്ഞെടുക്കൽ, ആവർത്തിക്കൽ, നീക്കംചെയ്യൽ എന്നീ പ്രസ്താവനകൾ പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യാൻ ആവശ്യമായേക്കാം. സാധാരണയായി ഇത്തരം തീരുമാനങ്ങൾ കൈക്കൊള്ളുന്നത് ചില നിബന്ധനകളെ അടിസ്ഥാനമാക്കിയാണ്. C++ ഈ ആവശ്യം നിറവേറ്റുന്നത് നിയന്ത്രണ പ്രസ്താവനകളുടെ സഹായത്തോടെയാണ്. ഈ പ്രസ്താവനകൾ പ്രോഗ്രാം നിർവഹണത്തിലെ സാധാരണ രീതിക്ക് മാറ്റം വരുത്തുന്നതിനായി ഉപയോഗിക്കുന്നു. നിയന്ത്രണ പ്രസ്താവനകളെ രണ്ടായി തരംതിരിക്കാം. (1) തീരുമാനമെടുക്കൽ/തിരഞ്ഞെടുക്കൽ (Decision making/Selection statement) (2) ആവർത്തന പ്രസ്താവനകൾ (Iteration statement). ഈ പ്രസ്താവനകളും ഇവയുടെ വാക്യഘടനയും നിർവഹണ രീതികളും നമുക്ക് ചർച്ച ചെയ്യാം.

7.1 തീരുമാനങ്ങൾ എടുക്കുന്നതിനുള്ള പ്രസ്താവനകൾ (Decision making statements)

പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യുമ്പോൾ കമ്പ്യൂട്ടറുകളിൽ എല്ലാ പ്രസ്താവനകളും എല്ലാ സന്ദർഭങ്ങളിലും ഒരുപോലെ പ്രവർത്തിക്കണമെന്നില്ല. ചില പ്രസ്താവനകൾ ഒരു സന്ദർഭത്തിൽ പ്രവർത്തിക്കുമെങ്കിലും മറ്റു ചില സന്ദർഭങ്ങളിൽ പ്രവർത്തിക്കണമെന്നില്ല. ഇത്തരം സന്ദർഭ

ങ്ങളിൽ കമ്പ്യൂട്ടറിന് ആവശ്യമായ തീരുമാനങ്ങൾ എടുക്കേണ്ടതുണ്ട്. ഇതിനായി നാം ഇവിടെ അനുയോജ്യമായ നിബന്ധനകൾ നൽകുകയും അവയെ കമ്പ്യൂട്ടർ വിലയിരുത്തുകയും വേണം. ഈ ഫലത്തിന്റെ അടിസ്ഥാനത്തിൽ അത് ഒരു തീരുമാനം എടുക്കുന്നു. ഈ തീരുമാനങ്ങൾ ഒന്നുകിൽ ഒരു പ്രത്യേക പ്രസ്താവനയെ പ്രവർത്തിപ്പിക്കുന്നതിനായി തിരഞ്ഞെടുക്കുന്നതിനോ അല്ലെങ്കിൽ ചില പ്രസ്താവനകളെ പ്രവർത്തിപ്പിക്കുന്നതിൽ നിന്നും ഒഴിവാക്കുന്നതിനോ ആയിരിക്കും. ഇപ്രകാരം ചില പ്രസ്താവനകൾ മാത്രം നിർവഹണം നടത്തുന്നതിനായി C++ ൽ തീരുമാനമെടുക്കൽ പ്രസ്താവനകൾ അല്ലെങ്കിൽ തെരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ ഉപയോഗിക്കുന്നു. `if`, `switch` എന്നിവയാണ് C++ ലെ രണ്ടുതരം തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ.

7.1.1 if പ്രസ്താവന (if statement)

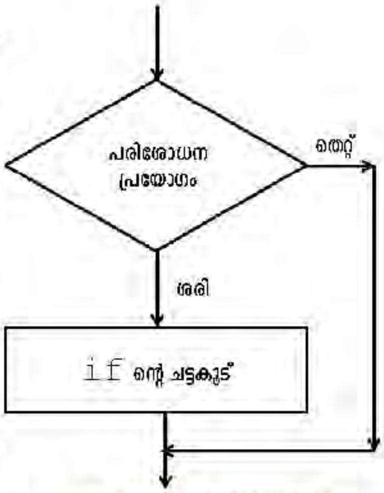
ഒരു നിബന്ധനയുടെ (condition) അടിസ്ഥാനത്തിൽ ഒരു കൂട്ടം പ്രസ്താവനകളെ പ്രവർത്തിപ്പിക്കുന്നതിനായി `if` പ്രസ്താവന ഉപയോഗിക്കുന്നു. C++ ൽ നിബന്ധനകൾ (പരിശോധനാ പ്രയോഗങ്ങൾ എന്നും അറിയപ്പെടുന്നു) നൽകുന്നത് റിലേഷണൽ അല്ലെങ്കിൽ ലോജിക്കൽ പ്രയോഗങ്ങൾ ഉപയോഗിച്ചാണ്. `if` പ്രസ്താവനയുടെ വാക്യഘടന (`Syntax`) താഴെ കൊടുത്തിരിക്കുന്നു.

```
if (പരിശോധന പ്രയോഗം)
{
     പ്രസ്താവനകൾ;
}
```

if ചട്ടകൂടിലെ നിബന്ധനകൾ ശരിയാവുകയാണെങ്കിൽ പ്രവർത്തിക്കേണ്ട പ്രസ്താവന കൂട്ടം;

```
if (test expression)
{
     statement block;
}
```

ഇവിടെ പരിശോധനാ പ്രയോഗം എന്നത് ഒന്നുകിൽ റിലേഷണൽ പ്രയോഗം അല്ലെങ്കിൽ ലോജിക്കൽ പ്രയോഗമായ ഒരു നിബന്ധനയെയാണ് സൂചിപ്പിക്കുന്നത്. പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ (True-പൂജ്യം അല്ലാത്ത വില) `if` -നോടു ചേർന്നുള്ള പ്രസ്താവനയോ അല്ലെങ്കിൽ ഒരു കൂട്ടം പ്രസ്താവനകളോ പ്രവർത്തിക്കും. അല്ലെങ്കിൽ `if` -നു ശേഷമുള്ള പ്രസ്താവനയിലേക്ക് നിയന്ത്രണം കൈമാറുന്നു. ചിത്രം 7.1 `if` പ്രസ്താവനയുടെ പ്രവർത്തന രീതി കാണിക്കുന്നു. `if` ഉപയോഗിക്കുമ്പോൾ താഴെ പറയുന്ന ചില കാര്യങ്ങൾ ഓർത്തിരിക്കേണ്ടതുണ്ട്.



ചിത്രം 7.1: if പ്രസ്താവനയുടെ പ്രവർത്തനം

- പരിശോധനാ പ്രയോഗം എപ്പോഴും ആവരണ ചിഹ്നത്തിന് അകത്തായിരിക്കണം.
- നിബന്ധനയിലുള്ള പ്രയോഗം റിലേഷണൽ പ്രയോഗങ്ങൾ ഉപയോഗിച്ചുള്ള ലളിതമായ പ്രയോഗങ്ങളോ, ലോജിക്കൽ പ്രയോഗങ്ങൾ ഉപയോഗിച്ചുള്ള സംയുക്ത പ്രയോഗങ്ങളോ ആകാം.
- `if` പ്രസ്താവനയോടുകൂടി ഒന്നോ അതിലധികമോ

പ്രസ്താവനകൾ ഉണ്ടാകാം. ഒരു പ്രസ്താവന മാത്രമാണെങ്കിൽ {,} എന്നീ ബ്രാക്കറ്റുകൾ നിർബന്ധമില്ല. ഒന്നിൽ കൂടുതൽ പ്രസ്താവനകൾ ഉണ്ടെങ്കിൽ ഈ ബ്രാക്കറ്റുകൾ നിർബന്ധമാണ്.

പ്രോഗ്രാം 7.1 ഒരു വിദ്യാർഥിയുടെ സ്കോർ സ്വീകരിക്കുകയും അത് 18 ഓ അതിലധികമോ ആണെങ്കിൽ "You have Passed" എന്ന് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. (പാസാവാൻ മിനിമം 18 സ്കോർ വേണമെന്ന് വിചാരിക്കുക)

പ്രോഗ്രാം 7.1: സ്കോർ 18 ഓ അതിലധികമോ ആണെങ്കിൽ "You have Passed" എന്ന പ്രദർശിപ്പിക്കുന്നതിന്

```
#include<iostream>
using namespace std;

{
    int score ;
    cout << "Enter your score: ";
    cin >> score;
    if (score >= 18)
         cout << "You have passed";
    return 0;
}
```

if ന്റെ ചട്ടക്കൂട്

പ്രോഗ്രാം 7.1 - ന്റെ മാതൃകാ ഔട്ട്പുട്ട് താഴെകൊടുത്തിരിക്കുന്നു.

```
Enter your score: 25
You have passed
```

പ്രോഗ്രാം 7.1 -ൽ ഒരു വിദ്യാർത്ഥിയുടെ സ്കോർ നൽകുകയും അത് `score` എന്ന വേരിയബിളിൽ സംഭരിക്കുകയും ചെയ്യുന്നു. പരിശോധനാപ്രയോഗം സ്കോർ എന്ന വേരിയബിളിലെ വില 18-ഓ അതിൽ അധികമോ ആണോ എന്നു നോക്കുന്നു. പരിശോധനാപ്രയോഗം ശരിയാണെങ്കിൽ `if` -ന്റെ ഭാഗം പ്രവർത്തിക്കുന്നു. അതായത് സ്കോർ 18-ഓ അതിലധികമോ ആണെങ്കിൽ `"You have Passed"` എന്ന സന്ദേശം സ്ക്രീനിൽ പ്രദർശിപ്പിക്കപ്പെടുന്നു. അല്ലാത്തപക്ഷം ഒരു ഔട്ട്പുട്ടും ലഭിക്കുന്നില്ല.

`if`-നോടു കൂടിയുള്ള ഭാഗം ഒരു ടാബ് ദൂരത്തിന് ശേഷമാണ് എഴുതിയിട്ടുള്ളത് എന്നത് ശ്രദ്ധിക്കുക. നാം അതിനെ ഇൻഡന്റേഷൻ എന്നു വിളിക്കുന്നു. ഇത് പ്രോഗ്രാമിന്റെ വായന ക്ഷമത വർദ്ധിപ്പിക്കുകയും തെറ്റുകൾ കണ്ടുപിടിക്കാൻ സഹായിക്കുകയും ചെയ്യുന്നു. എന്നാൽ ഇവ പ്രോഗ്രാമിന്റെ പ്രവർത്തനത്തിൽ ഒരു സ്വാധീനവും ചെലുത്തുന്നില്ല.

താഴെ കൊടുത്തിരിക്കുന്ന C++ പ്രോഗ്രാം ശകലം ശ്രദ്ധിക്കുക. തന്നിരിക്കുന്ന ഇൻപുട്ട് ഒരു അക്ഷരമാണോ അല്ലെങ്കിൽ ഒരു അക്കമാണോ എന്ന് ഇത് പരിശോധിക്കുന്നു.

```
char ch;
cin >> ch;
if (ch >= 'a' && ch <= 'z')
```

ലോജിക്കൽ പ്രസ്താവന നിർദ്ധാരണം ചെയ്തിരിക്കുന്നു.

```
        cout << "You entered an alphabet";
    if (ch >= '0' && ch <= '9')
    {
        cout << "You entered a digit\n";
        cout << "It is a decimal number ";
    }
```

ഒരു പ്രസ്താവന മാത്രമേയുള്ളൂ അതുകൊണ്ട് {, } ആവശ്യമില്ല

ഒന്നിൽകൂടുതൽ പ്രസ്താവനകൾ ഉള്ളതു കൊണ്ട് അവ നിർബന്ധമായും {,} ന് അകത്തായിരിക്കണം

7.1.2 if... else പ്രസ്താവന (if... else statement)

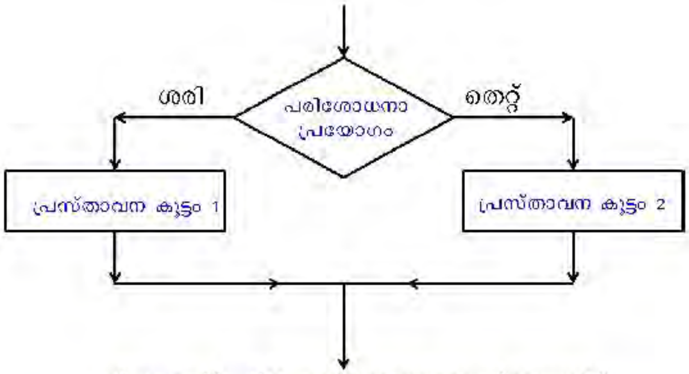
പ്രോഗ്രാം 7.1-ലെ if പ്രസ്താവന പരിഗണിക്കുക.

```
if (score >= 18)
    cout << "You have passed";
```

ഇവിടെ സ്കോർ പതിനെട്ടോ അതിൽ കൂടുതലോ ആണെങ്കിൽ മാത്രമേ ഔട്ട്പുട്ട് ലഭിക്കുന്നുള്ളൂ. നൽകിയ സ്കോർ 18-ൽ കുറവാണെങ്കിൽ എന്ത് സംഭവിക്കും? ഒരു ഔട്ട്പുട്ടും ലഭിക്കില്ല എന്ന് വ്യക്തമാണ്. പരിശോധന പ്രയോഗം വിലയിരുത്തുമ്പോൾ തെറ്റ് (false) ലഭിക്കുകയാണെങ്കിൽ മറ്റൊരു കൂട്ടം പ്രസ്താവനകൾ തിരഞ്ഞെടുക്കുന്നതിനുള്ള അവസരം നമുക്ക് ലഭിക്കാതെ വരുന്നു. നിബന്ധന തെറ്റാവുന്ന അവസരത്തിൽ ചില പ്രവർത്തനങ്ങൾ ചെയ്യണമെങ്കിൽ if പ്രസ്താവനയുടെ മറ്റൊരു രൂപമായ **if.. else** നമുക്ക് ഉപയോഗിക്കാം. ഇതിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
if    (പരിശോധനാ പ്രയോഗം)
{
      പ്രസ്താവനകൾ 1 ;
}
else
{
      പ്രസ്താവനകൾ 2;
}
if    (test expression)
{
      statement block 1;
}
else
{
      statement
block 2;
}
```

പരിശോധനാപ്രയോഗം ശരിയാണെങ്കിൽ പ്രസ്താവനകൾ 1 ഉം തെറ്റാണെങ്കിൽ പ്രസ്താവനകൾ 2 ഉം പ്രവർത്തിക്കുന്നു. if...else പ്രസ്താവനയുടെ പ്രവർത്തനം ചിത്രം 7.2-ൽ കാണിച്ചിരിക്കുന്നു.



ചിത്രം 7.2 if....else പ്രസ്താവനയുടെ ഫ്ലോചാർട്ട്

താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം `if...else` പ്രസ്താവനയുടെ പ്രവർത്തനം വിവരിക്കുന്നു.

```
if (score >= 18)
    cout << "Passed";
else
    cout << "Failed";
```

18 ഓ അതിലധികമോ ആണെങ്കിൽ മാത്രം ഈ പ്രസ്താവന പ്രവർത്തിക്കുന്നു. (അതായത് പരിശോധനാ പ്രയോഗം ശരിയാകുമ്പോൾ)

18 ൽ കുറവാണെങ്കിൽ ഈ പ്രസ്താവന പ്രവർത്തിക്കുന്നു. (അതായത് പരിശോധനാ പ്രയോഗം തെറ്റാകുമ്പോൾ.)

രണ്ട് കുട്ടികളുടെ ഉയരം ഇൻപുട്ടായി സ്വീകരിച്ച് അവരിൽ ഉയരമുള്ള കുട്ടിയെ കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്കെഴുതാം.

പ്രോഗ്രാം 7.2: വിദ്യാർത്ഥികളുടെ ഉയരം താരതമ്യം ചെയ്ത് അവരിൽ ഉയരം കൂടുതലുള്ള ആളെ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int ht1, ht2;
    cout << "Enter heights of the two students: ";
    cin >> ht1 >> ht2;
    if (ht1 > ht2)   //decision making based on condition
        cout<<"Student with height "<<ht1<<" is taller";
    else
        cout<<"Student with height "<<ht2<<" is taller";
    return 0;
}
```

പ്രോഗ്രാം 7.2 പ്രവർത്തിക്കുമ്പോൾ `ht1>ht2` എന്ന റിലേഷണൽ പ്രയോഗത്തിന്റെ ഫലത്തെ ആശ്രയിച്ച് ഏതെങ്കിലും ഒരു ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കപ്പെടും. മാതൃകാ ഔട്ട്പുട്ടുകൾ താഴെ കൊടുത്തിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1:
```
Enter the height of two students: 170   165
Student with height 170 is taller
```

ഔട്ട്പുട്ട് 2:
```
Enter the height of two students: 160   171
Student with height 171 is taller
```

ഔട്ട്പുട്ട് 1-ൽ `ht1` -ന് 170-ഉം `ht2` -ന് 165-ഉം ഇൻപുട്ടായി നൽകിയിരിക്കുന്നു. അതുകൊണ്ട് `(ht1>ht2)` എന്ന പരിശോധനാപ്രയോഗം ശരിയാവുകയും തൽഫലമായി `if` ബ്ലോക്ക് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ഔട്ട്പുട്ട് 2-ൽ `ht1` -ന് 160-ഉം `ht2`-ന് 171-ഉം വിലകൾ നൽകുമ്പോൾ `ht1>ht2` എന്ന പരിശോധനാ പ്രയോഗം തെറ്റാവുകയും തൽഫലമായി `else` ബ്ലോക്ക് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. `if .. else` പ്രസ്താവനയിൽ ഒന്നുകിൽ `if` നോട് അനുബന്ധിച്ചുള്ള കോഡും

(പ്രസ്താവനകൾ 1) അല്ലെങ്കിൽ `else` നോട് അനുബന്ധിച്ചുള്ള കോഡും (പ്രസ്താവനകൾ 2) പ്രവർത്തിക്കുന്നു.

പരിശോധനാ പ്രയോഗത്തിൽ ഒരു ഓപറൻഡ് ആയി അരിത്തമെറ്റിക് പ്രയോഗം ഉപയോഗിച്ചിരിക്കുന്ന മറ്റൊരു പ്രോഗ്രാം നമുക്ക് നോക്കാം. പ്രോഗ്രാം 7.3 ഈ ആശയം ഉപയോഗിച്ച് ഒരു സംഖ്യ ഒറ്റസംഖ്യയാണോ ഇരട്ടസംഖ്യയാണോ എന്ന് പരിശോധിക്കുന്നു.

പ്രോഗ്രാം 7.3: തന്നിരിക്കുന്ന സംഖ്യ ഒറ്റസംഖ്യയാണോ ഇരട്ടസംഖ്യയാണോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout << "Enter the number: ";
    cin >> num;
    if (num%2 == 0)
        cout << "The given number is Even";
    else
        cout << "The given number is Odd";
    return 0;
}
```

പ്രോഗ്രാം 7.3 ന്റെ ചില മാതൃക ഔട്ട്പുട്ടുകൾ താഴെ കാണിച്ചിരിക്കുന്നു

ഔട്ട്പുട്ട് 1:

```
Enter the number: 7
The given number is Odd
```

ഔട്ട്പുട്ട് 2:

```
Enter the number: 10
The given number is Even
```

ഈ പ്രോഗ്രാമിൽ `(num%2)` എന്ന പ്രയോഗം `num` ലെ വിലയെ 2 കൊണ്ട് ഹരിച്ച് കിട്ടുന്ന ശിഷ്ടത്തെ പൂജ്യമായി താരതമ്യം ചെയ്യുന്നു. അത് തുല്യമാണെങ്കിൽ `if` പ്രസ്താവനകൾ പ്രവർത്തിക്കും അല്ലെങ്കിൽ `else` പ്രസ്താവനകൾ പ്രവർത്തിക്കും.

നമുക്ക് ചെയ്യാം

1. തന്നിരിക്കുന്ന സംഖ്യ പോസിറ്റീവ് ആണോ നെഗറ്റീവ് ആണോ എന്ന് പരിശോധിക്കാനുള്ള പ്രോഗ്രാം എഴുതുക. (പൂജ്യം ഒഴികെയുള്ള സംഖ്യ മാത്രമേ ഇൻപുട്ട് ചെയ്യുന്നുള്ളൂ.)
2. ഒരു അക്ഷരം സ്വീകരിച്ച് 'M' ആണെങ്കിൽ Male എന്നും 'F' ആണെങ്കിൽ Female എന്നും ഔട്ട്പുട്ട് കാണിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക.
3. നിങ്ങളുടെ പ്രായം ഇൻപുട്ടായി നൽകി അത് 18 വയസ്സിനു മുകളിലാണെങ്കിൽ വോട്ടു ചെയ്യാൻ യോഗ്യതയുണ്ടെന്നും അല്ലെങ്കിൽ യോഗ്യതയില്ലെന്നും പ്രദർശിപ്പിക്കാനുള്ള പ്രോഗ്രാം എഴുതുക.

7.1.3 നെസ്റ്റഡ് if (Nested if)

ചില സാഹചര്യങ്ങളിൽ `if` പ്രസ്താവനയുടെ അകത്തുനിന്നുകൊണ്ട് തീരുമാനങ്ങൾ എടുക്കേണ്ട ആവശ്യം വരും. ഒരു `if` ബ്ലോക്കിനകത്ത് മറ്റൊരു `if` ബ്ലോക്ക് എഴുതുന്നതിനെ നെസ്റ്റിങ്ങ് എന്നു പറയുന്നു. നെസ്റ്റഡ് എന്നാൽ ഒന്നിനകത്ത് മറ്റൊന്ന് എന്നാണ് അർഥം. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലം പരിഗണിക്കുക.

```
if (score >= 60)
{
    if(age >= 18)
        cout<<"You are selected for the course!";
}
```

(പുറമെയുള്ള if — `if (score >= 60)`)
(അകത്തുള്ള if — `if(age >= 18)`)

ഈ കോഡ് ശകലത്തിൽ Scoreന്റെ വില 60 ഓ അതിൽ കൂടുതലോ ആണെങ്കിൽ പ്രോഗ്രാമിന്റെ നിയന്ത്രണം പുറത്തെ **`if`** ബ്ലോക്കിനുള്ളിലേക്ക് പ്രവേശിക്കുന്നു. അതിനുശേഷം അകത്തുള്ള **`if`** ന്റെ പരിശോധനാ പ്രസ്താവന വിലയിരുത്തുന്നു. (അതായത് age ന്റെ വില പതിനെട്ടോ അതിൽ കൂടുതലോ എന്ന്). ഇത് ശരിയാണെങ്കിൽ `"You are selected for the course!"` എന്ന സന്ദേശം പ്രദർശിപ്പിക്കും. തുടർന്ന് പുറത്തെ **`if`** പ്രസ്താവനകൾക്ക് ശേഷമുള്ള പ്രസ്താവനകൾ പ്രവർത്തിക്കുന്നു.

ഒരു `if` പ്രസ്താവനക്കുള്ളിലെ മറ്റൊരു `if` പ്രസ്താവനയെ നെസ്റ്റഡ് `if` (nested **`if`**) എന്നു വിളിക്കുന്നു. നെസ്റ്റഡ് `if`-ന്റെ വിപുലീകരിച്ച വാക്യഘടന താഴെ കൊടുത്തിരിക്കുന്നു.

```
if  (test expression1)
{
    if (test expression 2)
        statement 1;
    else
        statement 2;
}
else
{
    body of else;
}
```

(statement 1 — പരിശോധനാ പ്രയോഗം രണ്ടും ശരിയാകുമ്പോൾ മാത്രം പ്രവർത്തിക്കുന്നു.)
(statement 2 — പരിശോധനാ പ്രയോഗം 1 ശരിയാവുകയും പരിശോധനാ പ്രയോഗം 2 തെറ്റാവുകയും ചെയ്യുമ്പോൾ പ്രവർത്തിക്കുന്നു.)
(body of else — പരിശോധനാ പ്രയോഗം 1 തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കും. പരിശോധനാ പ്രയോഗം 2 നിർദ്ധാരണം ചെയ്യുന്നില്ല.)

നെസ്റ്റഡ് if മായി ബന്ധപ്പെട്ട് ശ്രദ്ധിക്കേണ്ട പ്രധാന കാര്യം ഒരു else എപ്പോഴും അതേ ബ്ലോക്കിൽ തന്നെയുള്ള ഏറ്റവും അടുത്ത if മായി ബന്ധപ്പെട്ടിരിക്കുന്നു എന്നതാണ്. ഒരു ഉദാഹരണത്തിലൂടെ നമുക്ക് ഇത് ചർച്ച ചെയ്യാം. താഴെ പറയുന്ന പ്രോഗ്രാം ശകലം പരിഗണിക്കുക.

```
cout<<"Enter your score in Computer Science exam: ";
  cin>>score;
  if (score >= 18)
       cout<<"You have passed";
       if(score >= 54)
            cout<<"with A+ grade !";
  else
       cout<<"\nYou have failed";
```

Score ന് 45 എന്ന വില നൽകുകയാണെങ്കിൽ ഔട്ട്പുട്ട് താഴെ ഉള്ളതുപോലെയായിരിക്കും.

```
You have passed
You have failed
```

യുക്തിപരമായി ഇത് ശരിയല്ല എന്ന് നമുക്ക് അറിയാം. കോഡിന്റെ ഇൻഡന്റേഷൻ ശരിയാണെങ്കിലും പ്രവർത്തനത്തിൽ പ്രതിഫലിച്ചിട്ടില്ല. ഇതിൽ രണ്ടാമത്തെ if പ്രസ്താവന നെസ്റ്റഡ് if ആയി പരിഗണിച്ചിട്ടില്ല. പകരം അതിനെ else ബ്ലോക്കോടു കൂടിയ സ്വതന്ത്രമായ ഒരു if ആയിട്ടാണ് കണക്കാക്കിയിട്ടുള്ളത്. അതുകൊണ്ട് ആദ്യത്തെ if പ്രസ്താവനയിലെ പരിശോധനാ പ്രയോഗം ശരിയായതിനാൽ അതിലെ if ബ്ലോക്ക് പ്രവർത്തനത്തിനായി തിരഞ്ഞെടുക്കുന്നു. ഇത് ഔട്ട്പുട്ടിലെ ഒന്നാമത്തെ വരിക്ക് കാരണമാകുന്നു. അതിനുശേഷം രണ്ടാമത്തെ if പ്രസ്താവനയിലെ പരിശോധനാ പ്രയോഗം തെറ്റായതിനാൽ ഔട്ട്പുട്ടിലെ രണ്ടാമത്തെ വരി ലഭിക്കുന്നു. അതുകൊണ്ട് ശരിയായ ഔട്ട്പുട്ട് ലഭിക്കുന്നതിനായി കോഡ് താഴെയുള്ളതുപോലെ പരിഷ്കരിക്കണം.

```
cout<<"Enter your score in Computer Science exam: ";
cin>>score;
if (score >= 18)
{
     cout<<"You have passed";
     if(score >= 54)
          cout<<" with A+ grade !";
}
else
     cout<<"\nYou have failed";
```

ഒരു ജോഡി ബ്രാക്കറ്റുകളുടെ സഹായത്തോടെ നെസ്റ്റിങ് നടപ്പിലാക്കിയിരിക്കുന്നു.

else പ്രസ്താവന ഇപ്പോൾ പുറത്തെ if നോട് ചേർന്നിരിക്കുന്നു.

മുൻ ഉദാഹരണത്തിലെ ഇൻപുട്ട് ആയ 45 ഇവിടെയും നൽകിയാൽ താഴെ പറയുന്ന ഔട്ട്പുട്ട് ലഭിക്കും.

```
You have passed
```

തന്നിരിക്കുന്ന മൂന്നു സംഖ്യകളിൽ വലുത് കണ്ടെത്തുന്നതിനായി പ്രോഗ്രാം 7.4 ൽ നെസ്റ്റഡ് `if` ഉപയോഗിച്ചിരിക്കുന്നു. ഈ പ്രോഗ്രാമിൽ `if` പ്രസ്താവന `if` ബ്ലോക്കിനകത്തും `else` ബ്ലോക്കിനകത്തും ഉപയോഗിച്ചിരിക്കുന്നു.

പ്രോഗ്രാം 7.4: മൂന്ന് സംഖ്യകളിൽ നിന്നും വലുത് കണ്ടുപിടിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int x, y, z;
    cout << "Enter three different numbers: ";
    cin >> x >> y >> z ;
    if (x > y)
    {
        if (x > z)
            cout << "The largest number is: " << x;
        else
            cout << "The largest number is: " << z;
    }
    else
    {
        if (y > z)
            cout << "The largest number is: " << y;
        else
            cout << "The largest number is: " << z;
    }
    return 0;
}
```

പ്രോഗ്രാം 7.4-ന്റെ ഒരു മാതൃകാ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
Enter three different numbers: 6   2   7
The largest number is: 7
```

ഇൻപുട്ട് നൽകിയ പ്രകാരം പുറമെയുള്ള if ലെ പരിശോധനാപ്രയോഗം `(x>y)` ശരിയായതിനാൽ അതിലെ അകത്തെ `if` ലേക്ക് പ്രവേശിക്കുന്നു. ഇവിടെ `(x>z)` എന്ന പരിശോധനാപ്രയോഗം തെറ്റായതിനാൽ അതിന്റെ `else` ബ്ലോക്ക് പ്രവർത്തിക്കുന്നു. അതുകൊണ്ട് z-ന്റെ വില ഔട്ട്പുട്ടായി പ്രദർശിപ്പിക്കുന്നു.

സ്വയം പരിശോധിക്കാം

1. ഒരു പൂർണ്ണ സംഖ്യ ഇൻപുട്ടായി സ്വീകരിച്ച്, അത് പോസറ്റീവാണോ നെഗറ്റീവാണോ പൂജ്യം ആണോ എന്ന് പരിശോധിക്കുന്നതിനുള്ള ഒരു പ്രോഗാം എഴുതുക
2. മൂന്ന് സംഖ്യകളെ ഇൻപുട്ടായി സ്വീകരിച്ച് അതിലെ ചെറുത് പ്രിന്റ് ചെയ്യാനുള്ള ഒരു പ്രോഗാം എഴുതുക.

7.1.4 else if ലാഡർ (The else if ladder)

ഒരു else ബ്ലോക്കിനുള്ളിൽ ഒരു if പ്രസ്താവന ഉപയോഗിക്കേണ്ട സാഹചര്യം ഉണ്ടായേക്കാം. അനേകം നിബന്ധനകൾ (condition) ആവശ്യമുള്ള പ്രോഗ്രാമുകളിൽ അത് ഉപയോഗിക്കുന്നു. പ്രവർത്തനത്തിനായി ഏത് പ്രസ്താവന തിരഞ്ഞെടുക്കണമെന്ന് അതാത് നിബന്ധന തീരുമാനിക്കുന്നു. if പ്രസ്താവനയെ അടിസ്ഥാനമാക്കിയുള്ള ഒരു സാധാരണ പ്രോഗ്രാമിംഗ് രൂപകൽപനയാണ് else if ലാഡർ. അതിന്റെ രൂപഘടനയിലുള്ള പ്രത്യേകത കാരണം അതിനെ else if സ്റ്റെയർ കെയ്സ് എന്നും പറയുന്നു. ഇത് if..else if പ്രസ്താവന എന്നും അറിയപ്പെടുന്നു. else if ലാഡറിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
if (പരിശോധനാ പ്രയോഗം 1)
      പ്രസ്താവനകൾ 1;
         else if (പരിശോധനാ പ്രയോഗം 2)
            പ്രസ്താവനകൾ 2;
               else if (പരിശോധനാ പ്രയോഗം 3)
                  പ്രസ്താവനകൾ 3;
                     ...............
                        else
                           പ്രസ്താവനകൾ n;
```

```
if (test expression 1)
      statement block 1;
         else if (test expression 2)
            statement block 2;
               else if (test expression 3)
                  statement block 3;
                     ...............
                        else
                           statement block n;
```

ആദ്യം പരിശോധനാ പ്രയോഗം 1 വിലയിരുത്തുമ്പോൾ അത് ശരിയാണെങ്കിൽ പ്രസ്താവനകൾ 1 പ്രവർത്തിച്ചതിനുശേഷം ലാഡറിൽ നിന്ന് പുറത്തേക്ക് വരുന്നു. അതായത് ലാഡറിന്റെ ബാക്കി ഭാഗം ഒഴിവാക്കപ്പെടുന്നു. പരിശോധനാപ്രയോഗം 1 വിലയിരുത്തുമ്പോൾ അത് തെറ്റാണെങ്കിൽ പരിശോധനാ പ്രയോഗം 2 വിലയിരുത്തുന്നു. ഈ പ്രക്രിയ അങ്ങിനെ തുടരുന്നു. ഏതെങ്കിലും ഒരു പരിശോധനാപ്രയോഗം ശരിയാണെങ്കിൽ അതിനനുസൃതമായ പ്രസ്താവനകൾ പ്രവർത്തിച്ചതിനുശേഷം നിയന്ത്രണം ലാഡറിന്റെ പുറത്തേക്ക് വരുന്നു. എല്ലാ പരിശോധനാ പ്രയോഗങ്ങളും വിലയിരുത്തുമ്പോൾ തെറ്റാണെങ്കിൽ അവസാന else-നുശേഷമുള്ള പ്രസ്താവനകൾ n പ്രവർത്തിക്കുന്നു. വാക്യഘടനയിൽ കൊടുത്തിരിക്കുന്ന ഇൻഡന്റേഷൻ നിരീക്ഷിക്കുകയും else if ലാഡർ ഉപയോഗിക്കുന്നതിന് ഈ രീതി പിന്തുടരുകയും ചെയ്യുക.

ഒരു വിദ്യാർത്ഥിക്ക് ഒരു വിഷയത്തിൽ 100 ൽ ലഭ്യമായ സ്കോറിന്റെ അടിസ്ഥാനത്തിൽ ഗ്രേഡ് കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാം else if ലാഡർ ഉപയോഗിച്ച് നമുക്ക് വിവരിക്കാം.

താഴെയുള്ള പട്ടികയിൽ കൊടുത്തിരിക്കുന്ന മാനദണ്ഡമനുസരിച്ചാണ് ഗ്രേഡ് കണ്ടുപിടിക്കേണ്ടത്.

സ്കോർ	ഗ്രേഡ്
80 ഓ അതിൽ കൂടുതലോ	A
60 മുതൽ 79 വരെ	B
40 മുതൽ 59 വരെ	C
30 മുതൽ 39 വരെ	D
30 ൽ താഴെ	E

പ്രോഗ്രാം 7.5: തന്നിരിക്കുന്ന സ്കോറിന്റെ അടിസ്ഥാനത്തിൽ വിദ്യാർത്ഥിയുടെ ഗ്രേഡ് കണ്ടുപിടിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int score;
    cout << "Enter your score: ";
    cin >> score;
    if (score >= 80)
        cout << "A Grade";
      else if (score >= 60)
          cout << "B Grade ";
        else if (score >= 40)
            cout << "C grade";
          else if (score >= 30)
              cout << "D grade";
            else
                cout << "E Grade";
    return 0;
}
```

പ്രോഗ്രാം 7.5 ന്റെ മാതൃക ഔട്ട്പുട്ടുകളാണ് താഴെയുള്ളത്.

ഔട്ട്പുട്ട് 1:

```
Enter your score: 73
B Grade
```

ഔട്ട്പുട്ട് 2:

```
Enter your score: 25
E Grade
```

പ്രോഗ്രാം 7.5 ൽ ആദ്യം പരിശോധനാ പ്രയോഗം `score>=80` വിലയിരുത്തുന്നു. ഔട്ട്പുട്ട് 1-ൽ ഇൻപുട്ട് ചെയ്ത വില 73 ആയതിനാൽ പരിശോധനാ പ്രയോഗം തെറ്റ് ആകുകയും `score>=60`

എന്ന അടുത്ത പരിശോധനാ പ്രയോഗം വിലയിരുത്തുകയും ചെയ്യുന്നു. ഇവിടെ ഇത് ശരിയായതിനാൽ "B Grade" എന്ന് പ്രദർശിപ്പിക്കുകയും `else if` ലാഡറിന്റെ ബാക്കി ഭാഗം ഒഴിവാക്കുകയും ചെയ്യുന്നു. എന്നാൽ ഔട്ട്പുട്ട് 2 - ൽ എല്ലാ പരിശോധനപ്രയോഗങ്ങളും തെറ്റാണെന്ന് വിലയിരുത്തിയതിനാൽ അവസാനത്തെ `else` ബ്ലോക്ക് പ്രസ്താവന പ്രവർത്തിക്കുകയും "E grade" എന്ന ഔട്ട്പുട്ട് ലഭിക്കുകയും ചെയ്യുന്നു.

തന്നിരിക്കുന്ന വർഷം അധിവർഷം (leap year) ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്ക് എഴുതാം. ഇൻപുട്ട് സംഖ്യ ശതാബ്ദമാണോ എന്ന് പരിശോധിക്കേണ്ടതുണ്ട് (നൂറുകൊണ്ട് ഹരിക്കാൻ സാധിക്കുന്ന വർഷമാണോ എന്ന്). അത് ഒരു ശതാബ്ദ വർഷമാണെങ്കിൽ അതിനെ 400 കൊണ്ട് കൂടി ഹരിക്കാമെങ്കിലേ അത് അധിവർഷമാകുന്നുള്ളൂ. ഇൻപുട്ട് സംഖ്യ ശതാബ്ദ വർഷമല്ലെങ്കിൽ അതിനെ 4 കൊണ്ട് ഹരിക്കുവാൻ സാധിക്കുമോ എന്ന് നാം പരിശോധിക്കേണ്ടതുണ്ട്. അതിനെ ഹരിക്കാൻ സാധിക്കുമെങ്കിൽ തന്നിരിക്കുന്ന വർഷം അധിവർഷം ആണ്, അല്ലെങ്കിൽ അത് ഒരു അധിവർഷമല്ല.

പ്രോഗ്രാം 7.6 തന്നിരിക്കുന്ന വർഷം അധിവർഷമാണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
void main()
{
  int year ;
  cout << "Enter the year (in 4-digits): ";
  cin >> year;
  if (year%100 == 0)// Checks for century year
  {
        if (year%400 == 0)
            cout << "Leap year\n";
        else
            cout<< "Not a leap year\n";
  }
  else if (year%4 == 0)
        cout << "Leap year\n";
  else
        cout<< "Not a leap year\n";
  return 0;
}
```

> ശതാബ്ദ വർഷമല്ലാത്തവ അധിവർഷം ആകണമെങ്കിൽ അവയെ 4കൊണ്ട് ഹരിക്കുവാൻ കഴിയണം.

പ്രോഗ്രാം 7.6 ന്റെ ചില മാതൃക ഔട്ട്പുട്ടുകൾ നമുക്ക് നോക്കാം.

ഔട്ട്പുട്ട് 1:

```
Enter the year (in 4-digits): 2000
Leap year
```

ഔട്ട്പുട്ട് 2:

```
Enter the year (in 4-digits): 2014
Not a leap year
```

ഔട്ട്പുട്ട് 3:

```
Enter the year (in 4-digits): 2100
Not a leap year
```

ഔട്ട്പുട്ട് 4:

```
Enter the year (in 4-digits): 2004
Leap year
```

`else if` ലാഡറിന്റെ ഉപയോഗം വിവരിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം കൂടി നമുക്ക് എഴുതാം. പ്രോഗ്രാം 7.7-ൽ ആഴ്ചയിലെ ദിവസത്തെ സൂചിപ്പിക്കുന്നതിനായി 1 മുതൽ 7 വരെയുള്ള സംഖ്യ ഇൻപുട്ടു ചെയ്യുന്നതിന് അനുവദിക്കുകയും അതിനനുസൃതമായ ദിവസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. ഇൻപുട്ട് 1 ആണെങ്കിൽ "Sunday" എന്നും 2 ആണെങ്കിൽ "Monday" എന്നും ഔട്ട്പുട്ടുകൾ പ്രദർശിപ്പിക്കപ്പെടുന്നു. ഇതുപോലെ മറ്റു ദിവസങ്ങളും. 1 മുതൽ 7 വരെയുള്ള പരിധിക്ക് പുറത്താണ് ഇൻപുട്ട് എങ്കിൽ "Wrong input" എന്നായിരിക്കും ഔട്ട്പുട്ട്.

പ്രോഗ്രാം 7.7: തന്നിരിക്കുന്ന ദിവസത്തെ സൂചിപ്പിക്കുന്ന സംഖ്യയ്ക്ക് അനുസൃതമായ ദിവസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int day;
    cout << "Enter the day number (1-7): ";
    cin >> day;
    if (day == 1)
      cout << "Sunday";
        else if (day == 2)
          cout << "Monday";
            else if (day == 3)
              cout << "Tuesday";
                else if (day == 4)
                  cout << "Wednesday";
                    else if (day == 5)
                      cout << "Thursday";
                        else if (day == 6)
                          cout << "Friday";
                            else if (day == 7)
                              cout << "Saturday";
                                else
                                  cout << "Wrong input";
    return 0;
}
```

പ്രോഗാം 7.7 ന്റെ ചില മാതൃക ഔട്ട്പുട്ടുകളാണ് താഴെയുള്ളത്.

ഔട്ട്പുട്ട് 1:

```
Enter the day number (1-7): 5
Thursday
```

ഔട്ട്പുട്ട് 2:

```
Enter day number (1-7): 9
Wrong input
```

സ്വയം പരിശോധിക്കാം

1. ഒരു പൂർണ്ണ സംഖ്യ ഇൻപുട്ടായി സ്വീകരിച്ച് അത് പോസറ്റീവാണോ നെഗറ്റീവാണോ പൂജ്യമാണോ എന്ന് പരിശോധിക്കുവാനുള്ള പ്രോഗ്രാം if else if പ്രസ്താവന ഉപയോഗിച്ച് എഴുതുക.
2. ഒരു അക്ഷരം (a, b, c അല്ലെങ്കിൽ d) ഇൻപുട്ട് ചെയ്യുന്നതിനും താഴെപറയുന്ന രീതിയിൽ ഔട്ട് പ്രദർശിപ്പിക്കുന്നതിനുമുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.
   "a - abacus", "b - boolean", "c - computer", "d - debugging"
3. ഒരു അക്ഷരം ഇൻപുട്ട് ചെയ്യുന്നതിനും അത് ആൽഫബറ്റാണോ, സംഖ്യയാണോ അതോ മറ്റേതെങ്കിലും ക്യാരക്ടർ ആണോ എന്ന് പ്രിന്റ് ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗാം എഴുതുക.

7.1.5. switch പ്രസ്താവന (switch statement)

else if ലാഡറിന്റെ സഹായത്തോടെ ബഹുശാഖീകരണം (Multiple branching) എന്ന ആശയം നാം കണ്ടു കഴിഞ്ഞു. C++-ലെ മറ്റൊരു രൂപകൽപ്പനയായ switch പ്രസ്താവന ഉപയോഗിച്ച് ഇവയിൽ ചില പ്രോഗ്രാമുകൾ എഴുതുവാൻ സാധിക്കും. ഈ തിരഞ്ഞെടുക്കൽ പ്രസ്താവന ഒരു വേരിയബിളിന്റേയോ ഒരു പ്രയോഗത്തിന്റേയോ (expression) വിലയെ ഒരു കൂട്ടം പൂർണ സംഖ്യകളുമായോ അക്ഷര സ്ഥിരാങ്കങ്ങളുമായോ തുടർച്ചയായി പരിശോധിക്കുന്നു. switch പ്രസ്താവനയുടെ വാക്യഘടന ചുവടെ ചേർത്തിരിക്കുന്നു.

```
switch (പ്രയോഗം)
        case സ്ഥിരാംഗം_1        : പ്രസ്താവനകൾ 1;
                                  break;
        case സ്ഥിരാംഗം_2        : പ്രസ്താവനകൾ 2;
                                  break;
        case സ്ഥിരാംഗം_3        : പ്രസ്താവനകൾ 3;
                                  break;
                                :
                                :
        case സ്ഥിരാംഗം_n-1      : പ്രസ്താവനകൾ n-1;
                                  break;
        default                 : പ്രസ്താവനകൾ n;
    }
```

```
switch (expression)
        'case' constant_1   : statement block 1;
                              break;
        'case' constant_2   : statement block 2;
                              break;
        'case' constant_3   : statement block 3;
                              break;
                            :
                            :
        'case' constant_n-1   :  statement block n-1;
                              break;
        default             : statement block n;
    }
```

വാക്യഘടനയിൽ switch, case, break, default എന്നിവ കീ വേർഡുകളാണ് (Keyword). ഒരു പൂർണ്ണസംഖ്യയോ ഒരു ക്യാരക്ടർ കോൺസ്റ്റന്റോ കിട്ടാവുന്ന രീതിയിൽ പ്രയോഗത്തെ വിലയിരുത്തുകയും അത് case പ്രസ്താവനകളിൽ കൊടുത്തിരിക്കുന്ന സ്ഥിരാംഗങ്ങൾക്ക് തുല്യമാണോ എന്ന് നോക്കുകയും ചെയ്യുന്നു. ഒരു തുല്യത കണ്ടെത്തിയാൽ ആ case-നോട് അനുബന്ധിച്ചുള്ള പ്രസ്താവനകൾ പ്രവർത്തിക്കും (break പ്രസ്താവന വരെയോ അല്ലെങ്കിൽ switch പ്രസ്താവനയുടെ അവസാനം വരെയോ). തുല്യത കണ്ടെത്തിയില്ലെങ്കിൽ default ബ്ലോക്കിലെ പ്രസ്താവന കൂട്ടം പ്രവർത്തിക്കും. default പ്രസ്താവന നിർബന്ധമല്ല. അത് ഉപയോഗിച്ചിട്ടില്ലെങ്കിൽ തുല്യത കണ്ടെത്താനാവാത്ത സന്ദർഭങ്ങളിൽ മറ്റൊന്നും പ്രവർത്തിക്കുകയില്ല.

switch-നകത്ത് ഉപയോഗിച്ചിരിക്കുന്ന break പ്രസ്താവന C++-ലെ ഒരു ജമ്പ് പ്രസ്താവനയാണ്. break പ്രസ്താവനയിൽ എത്തുമ്പോൾ പ്രോഗ്രാം നിയന്ത്രണം switch പ്രസ്താവനയ്ക്ക് ശേഷമുള്ള പ്രസ്താവനകളിലേക്ക് പോകുന്നു.

ഭാഗം 7.3.2 ൽ break സ്റ്റേറ്റ്മെന്റിനെക്കുറിച്ച് വിശദമായി നമുക്ക് ചർച്ച ചെയ്യാം. പ്രോഗ്രാം 7.7നെ switch പ്രസ്താവന ഉപയോഗിച്ച് എഴുതാവുന്നതാണ്. ഇത് കോഡിന്റെ വായനാ സുഖവും ഫലപ്രാപ്തിയും വർദ്ധിപ്പിക്കുന്നു. പ്രോഗ്രാം 7.8 ൽ വരുത്തിയ ഭേദഗതികൾ ശ്രദ്ധിക്കുക.

പ്രോഗ്രാം 7.8: switch പ്രസ്താവന ഉപയോഗിച്ച് ആഴ്ചയിലെ ദിവസം പ്രദർശിപ്പിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{     int day ;
      cout << "Enter a number between 1 and 7: ";
      cin >> day ;
      switch (day)
      {
         case 1: cout << "Sunday";
                 break;
```

```
        case 2: cout << "Monday";
                break;
        case 3: cout << "Tuesday";
                break;
        case 4: cout << "Wednesday";
                break;
        case 5: cout << "Thursday";
                break;
        case 6: cout << "Friday";
                break;
        case 7: cout << "Saturday";
                break;
        default: cout << "Wrong input";
    }
    return 0;
}
```

പ്രോഗ്രാം 7.7-ന്റെ ഔട്ട്പുട്ട് തന്നെയായിരിക്കും പ്രോഗ്രാം 7.8-നും. ചില മാതൃകകൾ താഴെ കൊടുത്തിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1:

```
Enter a number between 1 and 7: 5
Thursday
```

ഔട്ട്പുട്ട് 2:

```
Enter a number between 1 and 7: 8
Wrong input
```

പ്രോഗ്രാം 7.8-ൽ day വേരിയബിന്റെ വില case പ്രസ്താവനയിലെ സ്ഥിരാങ്കങ്ങളുമായി താരതമ്യം ചെയ്തിരിക്കുന്നു. ഒരു തുല്യത കണ്ടെത്തുമ്പോൾ ആ case-നോട് അനുബന്ധിച്ചുള്ള ഔട്ട്പുട്ട് പ്രസ്താവന പ്രവർത്തിക്കുന്നു. വേരിയബിൾ day യ്ക്ക് 5 എന്ന വില നാം ഇൻപുട്ടായി കൊടുത്താൽ അഞ്ചാമത്തെ case പ്രസ്താവന തുല്യമാവുകയും cout << "Thursday"; എന്ന പ്രസ്താവന പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ഇൻപുട്ട് 8 ആണെങ്കിൽ തുല്യത കണ്ടെത്തുവാൻ ആകില്ല. ആയതിനാൽ default ബ്ലോക്ക് പ്രവർത്തിക്കും.

എല്ലാ break പ്രസ്താവനകളെയും ഒഴിവാക്കുകയാണെങ്കിൽ പ്രോഗ്രാം 7.8-ന്റെ ഔട്ട്പുട്ട് നിങ്ങൾക്ക് പ്രവചിക്കാമോ? case സ്ഥിരാങ്കങ്ങളുമായി day യുടെ വില താരതമ്യം ചെയ്യുന്നു. ആദ്യത്തെ തുല്യത കണ്ടെത്തുമ്പോൾ അനുബന്ധ പ്രസ്താവനകളും തുടർന്നുള്ള പ്രസ്താവനകളും പ്രവർത്തിക്കുന്നു (അവശേഷിക്കുന്ന സ്ഥിരാങ്കങ്ങൾ പരിശോധിക്കാതെ). ചില സാഹചര്യങ്ങളിൽ break പ്രസ്താവന മനഃപൂർവ്വം ഒഴിവാക്കാറുണ്ട്. ഒരു switch ലെ തന്നിട്ടുള്ള എല്ലാ case നോടും അനുബന്ധിച്ചുള്ള പ്രസ്താവനകൾ ഒരുപോലെയാണെങ്കിൽ അത്തരം പ്രസ്താവനകൾ അവസാനത്തെ case ൽ നാം എഴുതിയാൽ മതിയാകും. പ്രോഗ്രാം 7.9 ഈ ആശയം വിവരിക്കുന്നു.

പ്രോഗ്രാം 7.9: തന്നിരിക്കുന്ന അക്ഷരം സ്വരാക്ഷരം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout<<"Enter the character to check: ";
    cin>>ch;
    switch(ch)
    {
        case 'A' :
        case 'a' :
        case 'E' :
        case 'e' :
        case 'I' :
        case 'i' :
        case 'O' :
        case 'o' :
        case 'U' :
        case 'u' : cout<<"The given character is a vowel";
                   break;
        default  : cout<<"The given character is not a vowel";
    }
    return 0;
}
```

പ്രോഗ്രാം. 7.9 നൽകുന്ന ചില ഔട്ട്പുട്ടുകൾ താഴെ കാണിച്ചിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1:

```
Enter a character to check: E
The given character is a vowel
```

ഔട്ട്പുട്ട് 2:

```
Enter a character to check: k
The given character is not a vowel
```

Switch ഉപയോഗിക്കുന്നതിന്റെ അനുയോജ്യതയും ആവശ്യകതയും

Switch പ്രസ്താവന else if ലാഡറിന്റെ അനേകം ശാഖകളുള്ള പ്രസ്താവനകൾക്ക് പകരമായാണ് ഉപയോഗിക്കുന്നതെങ്കിലും ഇവ രണ്ടും ഒരേ രീതിയിലല്ല പ്രവർത്തിക്കുന്നത്. C++-ൽ എല്ലാ Switch പ്രസ്താവനകളേയും else if ലാഡർ ഉപയോഗിച്ച് മാറ്റിയെഴുതാം. എന്നാൽ എല്ലാ else if ലാഡറുകളും switch ഉപയോഗിച്ച് മാറ്റിയെഴുതാൻ സാധിക്കില്ല. switch പ്രസ്താവന ഉപയോഗിച്ച് അനേകം ശാഖകൾ നടപ്പിൽ വരുത്തുന്നതിന് താഴെ പറയുന്നവ ആവശ്യമാണ്.

- നിബന്ധനകളിൽ തുല്യത പരിശോധന മാത്രമെ ഉള്ളൂ. മറ്റ് അവസരങ്ങളിൽ അതിനെ തുല്യത പ്രയോഗങ്ങളാക്കി മാറ്റിയെഴുതണം.
- എല്ലാ തുല്യതാ പ്രയോഗങ്ങളിലേയും ആദ്യത്തെ ഓപ്പറൻഡ് (operand) ഒരേ വേരിയബിളോ പ്രയോഗമോ ആയിരിക്കണം.
- ഈ പ്രയോഗങ്ങളിലെ രണ്ടാമത്തെ ഓപ്പറൻഡ് (operand) പൂർണസംഖ്യ (integer) അല്ലെങ്കിൽ ക്യാരക്ടർ കോൺസ്റ്റന്റ് ആയിരിക്കണം.

ഈ അധ്യായത്തിൽ ഇതുവരെ ചർച്ച ചെയ്ത പ്രോഗ്രാം 7.3, പ്രോഗ്രാം 7.7 എന്നിവയിലെ ശാഖകൾ മാത്രമേ `switch` ഉപയോഗിച്ച് മാറ്റിയെഴുതുവാൻ സാധിക്കുകയുള്ളൂ. പ്രോഗ്രാം 7.5-ൽ പരിശോധനാ പ്രയോഗങ്ങൾ `score/10==10`, `score/10==9`, `score/10==8` എന്നിങ്ങനെ മാറ്റം വരുത്തിയാൽ `switch` ഉപയോഗിക്കാം. ഇതുപോലെ മറ്റു സ്കോറുകൾ മാറ്റി എഴുതുക. താഴെകൊടുത്തിരിക്കുന്ന പ്രോഗ്രാംശകലം `else if` ഗോവണിക്കു പകരമായി ഉപയോഗിക്കാം.

```
switch(score/10)
{
    case 10:
    case 9: case 8: cout<< "A Grade"; break;
    case 7: case 6: cout<< "B Grade"; break;
    case 5: case 4: cout<< "C Grade"; break;
    case 3:          cout<< "D Grade"; break;
    default: cout<< "E Grade";
}
```

> സ്കോർ `int` ടൈപ്പ് ആയതിനാൽ പ്രയോഗം പൂർണ സംഖ്യകൾ മാത്രമേ നൽകുന്നുള്ളൂ.

`switch` ഉം `else if` ലാഡറും തമ്മിലുള്ള ഒരു താരതമ്യം പട്ടിക 7.1-ൽ കൊടുത്തിരിക്കുന്നു.

`switch` പ്രസ്താവന	`else if` ലാഡർ
1. അനേകം ശാഖകൾ (ബ്രാഞ്ച്) അനുവദിക്കുന്നു.	1. അനേകം ശാഖകൾ (ബ്രാഞ്ച്) അനുവദിക്കുന്നു.
2. തുല്യത (equality) ഓപ്പറേറ്റർ ഉള്ള നിബന്ധനകൾ മാത്രം വിലയിരുത്തുന്നു.	2. ഏതൊരു റിലേഷണൽ/ലോജിക്കൽ പ്രയോഗങ്ങളും വിലയിരുത്തുന്നു.
3. `case` സ്ഥിരാങ്കം എപ്പോഴും പൂർണ സംഖ്യയോ അക്ഷരമോ ആയിരിക്കണം.	3. ഫ്ളോട്ടിങ്ങ് പോയിന്റ് സ്ഥിരാങ്കങ്ങളോ ഒരു പരിധിയിലുള്ള വിലകളോ നിബന്ധനകളിലുൾപ്പെടുത്താം.
4. ഒരു തുല്യതയും ലഭിക്കാത്തപ്പോൾ default പ്രസ്താവന പ്രവർത്തിക്കുന്നു.	4. ഒരു പ്രയോഗവും ശരിയായില്ലെങ്കിൽ else ബ്ലോക്ക് പ്രവർത്തിക്കുന്നു.
5. `switch` പ്രസ്താവനയിൽ നിന്നും പുറത്തു കടക്കുന്നതിന് `break` പ്രസ്താവന ആവശ്യമാണ്.	5. ഒരു ബ്ലോക്ക് പൂർത്തീകരിച്ചതിനുശേഷം പ്രോഗ്രാമിന്റെ നിയന്ത്രണം സ്വയം ബ്ലോക്കിന് പുറത്തു പോകുന്നു.
6. ഒരേ വേരിയബിളോ പ്രയോഗമോ ഒരു കൂട്ടം വിലകളുമായി തുല്യത പരിശോധിക്കുന്നതിന് കൂടുതൽ ഫലപ്രദമാണ്.	6. `switch`-നെക്കാൾ വഴക്കമുള്ളതും എളുപ്പത്തിൽ ഉപയോഗിക്കുവാൻ സാധിക്കുന്നതുമാണ്.

പട്ടിക 7.1: `switch` ഉം `else if` ലാഡറും തമ്മിലുള്ള താരതമ്യം

7.1.6 കണ്ടിഷണൽ ഓപ്പറേറ്റർ (Conditional operator (?:))

അധ്യായം 6- ൽ സൂചിപ്പിച്ചതുപോലെ C++ ൽ ഒരു ടെർണറി ഓപ്പറേറ്റർ ഉണ്ട്. ? ഉം : ഉം എന്നീ ചിഹ്നങ്ങൾ (ചോദ്യചിഹ്നവും കോളനും) ഉൾപ്പെടുന്ന കണ്ടിഷണൽ ഓപ്പറേറ്റർ (Conditional operator) ആണ് അത്. ഇത് ഉപയോഗിക്കുന്നതിന് മൂന്ന് ഓപ്പറന്റുകൾ ആവശ്യമാണ്. `if...else` പ്രസ്താവനക്ക് പകരമായി ഇതിനെ ഉപയോഗിക്കാം. ഇതിന്റെ വാക്യഘടന താഴെ കൊടുത്തിരിക്കുന്നു.

പരിശോധനാ പ്രയോഗം ? പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ് : പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ് ;

```
Test expression ? True_case code : False_case code;
```

പരിശോധനാ പ്രയോഗം ഏതെങ്കിലും റിലേഷണലോ ലോജിക്കലോ ആയ പ്രയോഗം ആകാം. പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ്, പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ് എന്നിവ സ്ഥിരവിലയോ, വേരിയബിളോ, പ്രയോഗമോ അല്ലെങ്കിൽ പ്രസ്താവനയോ ആകാം. ഇതിന്റെ പ്രവർത്തനം `if else` പ്രസ്താവനയുടെ സഹായത്തോടു കൂടി താഴെ കാണിച്ചിരിക്കുന്നു.

```
if Test experssion (പരിശോധനാ പ്രയോഗം)
     {
     True_case code: (പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ്:)
     }
     else
     {
     False_case code: (പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ്;)
     }
```

```
if   (Test expression)
     {
          True_case code;
     }
     else
     {
          False_case code;
     }
```

`if...else` പ്രവർത്തിക്കുന്നതുപോലെയാണ് കണ്ടിഷണൽ ഓപ്പറേറ്ററും പ്രവർത്തിക്കുന്നത്. പരിശോധനാ പ്രയോഗം വിലയിരുത്തി അത് ശരിയാണെങ്കിൽ 'പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡും' (`true_case code`) തെറ്റാണെങ്കിൽ 'പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡും' (`False_case code`) തിരഞ്ഞെടുക്കുന്നു. കണ്ടീഷണൽ ഓപ്പറേറ്ററിന്റെ പ്രവർത്തനം പ്രോഗ്രാം 7.10 ൽ വിവരിച്ചിരിക്കുന്നു.

പ്രോഗ്രാം 7.10: കണ്ടീഷണൽ ഓപ്പറേറ്റർ ഉപയോഗിച്ച് ഏറ്റവും വലിയ സംഖ്യ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
int num1, num2;
cout << "Enter two numbers: ";
cin>> num1 >> num2 ;
(num1>num2)? cout<<num1<<" is larger" : cout<<num2<<" is larger";
return 0;
}
```

ഈ പ്രോഗ്രാമിലെ `return 0` പ്രസ്താവനയ്ക്ക് മുമ്പുള്ള പ്രസ്താവനയിൽ കണ്ടീഷണൽ ഓപ്പറേറ്റർ ഉപയോഗിക്കുന്നതുകൊണ്ട് അതിനെ കണ്ടീഷണൽ പ്രസ്താവന എന്നു വിളിക്കുന്നു. ഈ പ്രസ്താവനയെ താഴെയുള്ള കോഡ് ശകലം ഉപയോഗിച്ച് മാറ്റി എഴുതാവുന്നതാണ്.

```
int big = (num1>num2)? num1 : num2;
cout<< big << "is larger";
```

പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ `num1`-ന്റെ വിലയും തെറ്റാണെങ്കിൽ `num2` ന്റെ വിലയും ആയിരിക്കും `big` ലേക്ക് ശേഖരിക്കുക. ഇവിടെ കണ്ടീഷണൽ ഓപ്പറേറ്റർ ഉപയോഗിച്ചാണ് ഒരു കണ്ടീഷണൽ പ്രയോഗം ഉണ്ടാക്കിയിരിക്കുന്നത്. ഈ പ്രയോഗത്തിൽ നിന്നും ലഭിക്കുന്ന വില `big` ലേക്ക് ശേഖരിക്കുന്നു.

കണ്ടീഷണൽ പ്രയോഗത്തിന്റെ ഒരു സങ്കീർണ രൂപം താഴെ കൊടുത്തിരിക്കുന്നു. ഇത് മൂന്ന് സംഖ്യകളിൽ ഏറ്റവും വലുത് നൽകുന്നു. `n1`, `n2`, `n3`, `big` എന്നിവ പൂർണ സംഖ്യ വേരിയബിളുകളാണെങ്കിൽ,

```
big = (n1>n2) ? ( (n1>n3)?n1:n3 ) : ( (n2>n3)?n2:n3);
```

പ്രോഗ്രാം 7.4 പരിശോധിച്ച് മുകളിൽ കൊടുത്തിരിക്കുന്ന കണ്ടീഷണൽ പ്രയോഗം എങ്ങനെയാണ് നെസ്റ്റഡ് `if` നുപകരമായി ഉപയോഗിച്ചിരിക്കുന്നത് എന്ന് നോക്കുക.

സ്വയം പരിശോധിക്കാം

1. 1 മുതൽ 12 വരെയുള്ള സംഖ്യകൾ ഇൻപുട്ട് ചെയ്ത് അതിനനുസൃതമായ മാസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക. (1 ആണെങ്കിൽ January, 2 ആണെങ്കിൽ February എന്നിങ്ങനെ)
2. `switch` പ്രസ്താവന ഉപയോഗിച്ച് അരിത്തമറ്റിക് ഓപ്പറേഷനുകൾ ചെയ്യുവാനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. ഇതിനുവേണ്ടി 2 ഓപ്പറന്റുകളും ഒരു ഓപ്പറേറ്ററും ഇൻപുട്ടായി സ്വീകരിക്കുക.
3. `switch` പ്രസ്താവനയ്ക്കകത്തുള്ള `break` പ്രസ്താവനയുടെ പ്രാധാന്യം എന്താണ്?
4. 0 മുതൽ 9 വരെയുള്ള ഏതെങ്കിലുമൊരു സംഖ്യ ഇൻപുട്ട് ചെയ്ത് അതിനെ അക്ഷരത്തിലെഴുതുവാനുള്ള ഒരു പ്രോഗ്രാം. `switch` പ്രസ്താവന ഉപയോഗിച്ച് എഴുതുക.

5. ഒരു സംഖ്യ ഇൻപുട്ടായി സ്വീകരിച്ച് ആ സംഖ്യ 5 ന്റെ ഗുണിതമാണ് എന്ന് പരിശോധിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം തിരഞ്ഞെടുക്കൽ പ്രസ്താവനയും കണ്ടീഷണൽ ഓപ്പറേറ്ററും ഉപയോഗിച്ച് എഴുതുക.
6. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവന `if..else` ഉപയോഗിച്ച് മാറ്റിയെഴുതുക.

```
result = (mark>30)? 'P' : 'F';
```

7.2. ആവർത്തന പ്രസ്താവനകൾ (Iteration statements)

അധ്യായം 4-ൽ നാം ചർച്ച ചെയ്ത ചില പ്രശ്നങ്ങളുടെ ഉത്തരങ്ങളിൽ ആവർത്തന സ്വാഭാവമുള്ള പ്രവർത്തികൾ അടങ്ങിയിട്ടുണ്ട്. പ്രോഗ്രാമുകൾ എഴുതുമ്പോൾ ഒന്നോ അതിലധികമോ പ്രസ്താവനകളെ പല തവണ പ്രവർത്തിപ്പിക്കുന്നതിനായി ഭാഷയുടെ പ്രത്യേക രൂപകൽപ്പനകൾ നാം ഉപയോഗിക്കുന്നു. ഇത്തരം രൂപകൽപ്പനകളെ **ആവർത്തന പ്രസ്താവനകൾ** (Iteration statements) അല്ലെങ്കിൽ **ലൂപ്പിങ് പ്രസ്താവനകൾ** എന്നു വിളിക്കുന്നു. C++-ൽ മൂന്ന് തരം ആവർത്തന പ്രസ്താവനകൾ ഉണ്ട്. ഒരു നിബന്ധന ശരിയാകുമ്പോൾ ഒരു കൂട്ടം പ്രസ്താവനകൾ ആവർത്തിച്ച് പ്രവർത്തിപ്പിക്കുവാൻ ഇവ അനുവദിക്കുന്നു.

ലൂപ്പ് എന്ന ആശയം നിത്യജീവിതത്തിൽ നാം പ്രയോഗിക്കാറുണ്ട്. നമുക്ക് ഒരു സാഹചര്യം പരിഗണിക്കാം. പരീക്ഷയിൽ A+ ഗ്രേഡ് ലഭിക്കുന്ന എല്ലാ വിദ്യാർത്ഥികൾക്കും നിങ്ങളുടെ ക്ലാസ് ടീച്ചർ ഒരു സമ്മാനം തരുമെന്ന് പ്രഖ്യാപിച്ചു എന്ന് വിചാരിക്കുക. സമ്മാനം പൊതിയാനുള്ള ചുമതല നിങ്ങളെ ഏൽപ്പിക്കുന്നു. സമ്മാനം പൊതിയേണ്ടതെങ്ങനെയെന്ന് താഴെ കൊടുത്ത രീതിയിൽ ടീച്ചർ വിശദീകരിക്കുന്നു.

ഘട്ടം 1 : സമ്മാനം എടുക്കുക.

ഘട്ടം 2 : പൊതിയാനുള്ള പേപ്പർ മുറിക്കുക.

ഘട്ടം 3 : സമ്മാനം പൊതിയുക.

ഘട്ടം 4 : റിബൺ ഉപയോഗിച്ച് കവർ കെട്ടുക.

ഘട്ടം 5 : കാർഡിൽ പേരെഴുതി സമ്മാനത്തിന് മുകളിൽ ഒട്ടിക്കുക.

പരീക്ഷയിൽ 30 വിദ്യാർഥികൾക്ക് A+ ഗ്രേഡ് ഉണ്ടെങ്കിൽ ഇതേ പ്രവർത്തി 30 തവണ നിങ്ങൾ ആവർത്തിക്കേണ്ടതുണ്ട്. സമ്മാനം പൊതിയുന്ന ഈ പ്രവർത്തി 30 തവണ ആവർത്തിക്കുന്നതിന് താഴെ കൊടുത്ത രീതിയിൽ നിർദ്ദേശങ്ങൾ പുനഃക്രമീകരിക്കാം.

താഴെ കൊടുത്ത ഘട്ടങ്ങൾ 30 തവണ ആവർത്തിക്കുക

{

അടുത്ത സമ്മാനം എടുക്കുക.

പൊതിയാനുള്ള പേപ്പർ മുറിക്കുക.

സമ്മാനം പൊതിയുക.

റിബൺ ഉപയോഗിച്ച് കവർ കെട്ടുക.

കാർഡിൽ പേരെഴുതി സമ്മാനത്തിന് മുകളിൽ ഒട്ടിക്കുക.

}

ഇനി വേറൊരു ഉദാഹരണമെടുക്കാം. കമ്പ്യൂട്ടർ ആപ്ലിക്കേഷൻ വിഷയത്തിൽ ലഭിച്ച സ്കോറുകളുടെ ക്ലാസ് ശരാശരി നമുക്ക് കണ്ടുപിടിക്കണമെന്ന് കരുതുക. അതിനായി താഴെ പറയുന്ന ഘട്ടങ്ങളിലൂടെ കടന്നു പോകണം.

ആകെ-സ്കോറിന് പ്രാരംഭ വിലയായി പൂജ്യം കൊടുക്കുക.

താഴെ പറയുന്ന ഘട്ടങ്ങൾ ആദ്യത്തെ വിദ്യാർഥിക്ക് മുതൽ അവസാനത്തെ ആൾ വരെ ആവർത്തിക്കുക.

{

വിദ്യാർത്ഥിയുടെ സ്കോർ ആകെ-സ്കോറിനോട് കൂട്ടുക.

അടുത്ത വിദ്യാർത്ഥിയുടെ സ്കോർ എടുക്കുക.

}

ശരാശരി = ആകെ- സ്കോർ/ക്ലാസ്സിലെ ആകെ വിദ്യാർത്ഥികളുടെ എണ്ണം

ഈ രണ്ടു ഉദാഹരണങ്ങളിലും ചില ഘട്ടങ്ങൾ നാം പല തവണ ചെയ്യുന്നു. പ്രക്രിയ എത്ര തവണ ആവർത്തിച്ചു എന്നറിയുന്നതിന് നാം ഒരു കൗണ്ടർ (counter) ഉപയോഗിക്കുന്നു. പ്രവർത്തനം തുടരണമോ വേണ്ടയോ എന്ന് കൗണ്ടറിന്റെ വില തീരുമാനിക്കുന്നു. നിബന്ധനയ്ക്ക് വിധേയമായി ലൂപ്പുകൾ പ്രവർത്തിക്കുന്നതിനാൽ കൗണ്ടർ പോലുള്ള വേരിയബിൾ ലൂപ്പ് നിർമ്മിക്കുന്നതിന് ഉപയോഗിക്കുന്നു. ഈ വേരിയബിൾ പൊതുവെ ലൂപ്പ് നിയന്ത്രണവേരിയബിൾ (Loop control variable) എന്നറിയപ്പെടുന്നു. എന്തുകൊണ്ടെന്നാൽ യഥാർത്ഥത്തിൽ ഇതാണ് ലൂപ്പിന്റെ പ്രവർത്തനത്തെ നിയന്ത്രിക്കുന്നത്. അധ്യായം 4-ൽ ഒരു ലൂപ്പിന്റെ 4 ഘടകങ്ങളെക്കുറിച്ച് നാം ചർച്ച ചെയ്തു. നമുക്ക് അതൊന്ന് ഓർത്തെടുക്കാം.

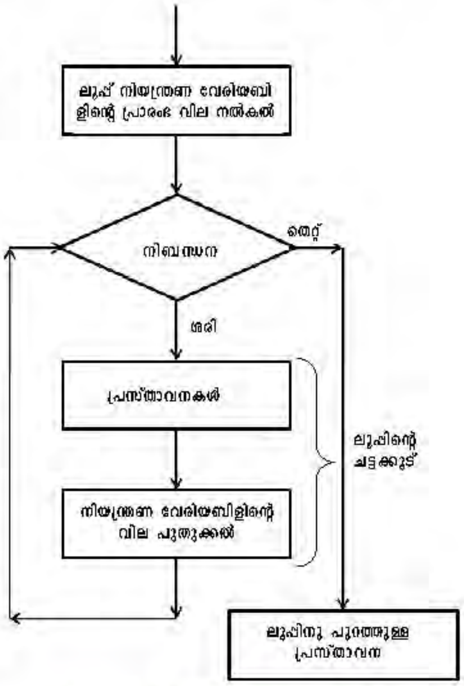
1. **പ്രാരംഭ വില നൽകൽ (Initialisation)** : ലൂപ്പിലേക്ക് പ്രവേശിക്കുന്നതിനു മുമ്പ് അതിന്റെ നിയന്ത്രണ വേരിയബിളിന് പ്രാരംഭ വില നൽകണം. അങ്ങനെ ലൂപ്പ് നിയന്ത്ര വേരിയബിളിന് അതിന്റെ ആദ്യത്തെ വില ലഭിക്കും. പ്രാരംഭ വില നൽകുന്ന പ്രസ്താവന ലൂപ്പിന്റെ തുടക്കത്തിൽ മാത്രമേ പ്രവർത്തിക്കുന്നുള്ളു.
2. **പരിശോധനാ പ്രയോഗം (Test Expression)** : ഇത് ഒരു റിലേഷണൽ അല്ലെങ്കിൽ ലോജിക്കൽ പ്രയോഗമാണ്. ഇതിന്റെ വില ശരി അല്ലെങ്കിൽ തെറ്റ് ആയിരിക്കും. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കണോ വേണ്ടയോ എന്ന് ഇത് തീരുമാനിക്കുന്നു. പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ ലൂപ്പ് പ്രവർത്തിക്കുന്നു. അല്ലെങ്കിൽ അത് പ്രവർത്തിക്കില്ല.
3. **പരിഷ്കരിക്കൽ പ്രസ്താവന (Updation Statement)** : പരിഷ്കരിക്കൽ പ്രസ്താവന ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വിലയിൽ മാറ്റം വരുത്തുന്നു. ഈ പ്രസ്താവന അടുത്ത ആവർത്തനത്തിന് മുന്നേ പ്രവർത്തിക്കുന്നു.
4. **ലൂപ്പിന്റെ ചട്ടക്കൂട് (Body of loop)** : ആവർത്തിക്കപ്പെടേണ്ട പ്രസ്താവനകൾ ഉപയോഗിച്ച് ലൂപ്പിന്റെ ചട്ടക്കൂട് രൂപപ്പെടുത്തുന്നു. ഇതിൽ ഒന്നോ അതിലധികമോ പ്രസ്താവനകൾ ഉണ്ടായിരിക്കും. ലൂപ്പുകളെ പൊതുവെ ആഗമന നിയന്ത്രണ ലൂപ്പുകൾ (Entry controlled loop) എന്നും ബഹിർഗമന നിയന്ത്രണ ലൂപ്പുകൾ (Exit controlled loop) എന്നും തരംതിരിച്ചിരിക്കുന്നു എന്ന് അധ്യായം 4-ൽ നാം പഠിച്ചു. C++ ൽ മൂന്നുതരം ലൂപ്പ് പ്രസ്താവനകൾ ഉണ്ട്: `while loop`, `for loop`, `do-while loop`. ഓരോന്നിന്റേയും പ്രവർത്തനം വിശദമായി നമുക്ക് ചർച്ച ചെയ്യാം.

7.2.1 while പ്രസ്താവന (while statement)

while ലൂപ്പ് ഒരു ആഗമന നിയന്ത്രണ ലൂപ്പ് ആണ്. നിബന്ധന (Condition) ആദ്യം പരിശോധിക്കുകയും അത് ശരിയാണെങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. അതായത് നിബന്ധന ശരിയാകുന്നിടത്തോളം ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കും. while ലൂപ്പിന്റെ വാക്യഘടന ഇതാണ്.

```
നിയന്ത്രണ വേരിയബിളിന്റെ പ്രാരംഭ വില നൽകൽ;
    while (പരിശോധനാ പ്രയോഗം)
    {
        ലൂപ്പിന്റെ ചട്ടക്കൂട്;
        ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിനെ പുതുക്കൽ;
    }
intialisation of loop control variable;
while (test expression)
{
    body of the loop;
    updation of loop control variable;
}
```

ഇവിടെ പരിശോധനാ പ്രയോഗം നിബന്ധന നിർവചിക്കുകയും അത് ലൂപ്പിനെ നിയന്ത്രിക്കുകയും ചെയ്യുന്നു. ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രസ്താവനയോ ഒന്നിലധികം പ്രസ്താവനകളോ അല്ലെങ്കിൽ പ്രസ്താവനകളില്ലാതെയോ ആകാം. ആവർത്തിച്ചു പ്രവർത്തിക്കുന്നതിനുള്ള ഒരു കൂട്ടം പ്രസ്താവനകളാണ് ലൂപ്പിന്റെ ചട്ടക്കൂട്. ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വില വ്യത്യാസപ്പെടുത്തുന്ന പ്രസ്താവനയാണ് പരിഷ്കരിക്കൽ പ്രസ്താവന. ഒരു while ലൂപ്പിൽ ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന് ലൂപ്പ് തുടങ്ങുന്നതിനുമുമ്പ് പ്രാരംഭവില നൽകുകയും ലൂപ്പ് ചട്ടക്കൂടിനുള്ളിൽ വച്ച് അതു പുതുക്കുകയും ചെയ്യുന്നു. ഒരു while ലൂപ്പിന്റെ പ്രവർത്തന ചിത്രം 7.3-ലെ ഫ്ളോചാർട്ടിൽ വിവരിച്ചിരിക്കുന്നു.



ചിത്രം 7.3. while ലൂപ്പിന്റെ പ്രവർത്തനം

നിയന്ത്രണ വേരിയബിളിന് പ്രാരംഭ വില നൽകുകയാണ് ആദ്യം ചെയ്യുന്നത്. പിന്നീട് പരിശോധനാ പ്രയോഗം വിലയിരുത്തുന്നു. അത് ശരിയാണ് എങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. അതുകൊണ്ടാണ് while ലൂപ്പിനെ ആഗമന നിയന്ത്രണ ലൂപ്പ് എന്ന് വിളിക്കുന്നത്. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നതിനൊപ്പം ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വിലയും പുതുക്കുന്നു. ലൂപ്പ് ചട്ടക്കൂട്ടിന്റെ പ്രവർത്തനം കഴിഞ്ഞതിനു ശേഷം പരിശോധനാപ്രയോഗം വീണ്ടും വിലയിരുത്തു

ന്നു. നിബന്ധന ശരിയായിരിക്കുന്നിടത്തോളം ഈ പ്രക്രിയ തുടരുന്നു. while ലൂപ്പിന്റെ പ്രവർത്തനം വിവരിക്കുന്നതിനുള്ള ഒരു കോഡ് ശകലം നമുക്ക് ഇപ്പോൾ പരിഗണിക്കാം.

```
int k=1;
while(k<=3)
{
      cout << k << '\t';
      ++k;
}
cout << "\n Program Ends";
```

ലൂപ്പിന് മുമ്പേ പ്രാരംഭ വില നൽകൽ

പരിശോധനാ പ്രയോഗം

ലൂപ്പിന്റെ ചട്ടക്കൂട്

ലൂപ്പിന്റെ ചട്ടക്കൂടിനകത്തെ പുതുക്കൽ പ്രസ്താവന

ഈ കോഡ് ശകലത്തിൽ k എന്ന ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന് 1 എന്ന വില ആദ്യം നൽകിയിരിക്കുന്നു. പിന്നീട് k<=3 എന്ന പരിശോധനാ പ്രയോഗമായ വിലയിരുത്തുന്നു. ഇത് ശരിയായതു കൊണ്ട് ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. അതായത് k-യുടെ വിലയായ 1 സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. അതിനുശേഷം പരിഷ്കരിക്കൽ പ്രസ്താവനയായ (update statement) ++k പ്രവർത്തിച്ച് k യുടെ വില 2 ആയി മാറുകയും ചെയ്യുന്നു. നിബന്ധന (k<=3) ഒന്നുകൂടി പരിശോധിച്ച് ശരിയാണെന്ന് കണ്ടെത്തുകയും ചെയ്തു. പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിനകത്ത് പ്രവേശിച്ച് k യുടെ വില 2 എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. വീണ്ടും പരിഷ്കരിക്കൽ പ്രസ്താവന ആവർത്തിക്കുകയും k യുടെ വില 3 ആകുകയും ചെയ്യുന്നു. നിബന്ധന ഇപ്പോഴും ശരിയായതിനാൽ ലൂപ്പ് പ്രവർത്തിച്ച് 3 എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. k യുടെ വില വീണ്ടും പരിഷ്കരിച്ച 4 ആവുകയും ഇപ്പോൾ പരിശോധന പ്രയോഗത്തിനന്റെ ഫലം തെറ്റാവുകയും ചെയ്യുന്നു. നിയന്ത്രണം ലൂപ്പിന് പുറത്തേക്ക് വരുകയും while ലൂപ്പിന് പുറത്തുള്ള അടുത്ത പ്രസ്താവന പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ചുരുക്കത്തിൽ കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നത് പോലെയായിരിക്കും.

```
1     2     3
Program ends
```

k-യുടെ പ്രാരംഭ വില 5 ആണെങ്കിൽ എന്ത് സംഭവിക്കുമെന്ന് സങ്കൽപ്പിക്കുക? ആദ്യം വിലയിരുത്തുമ്പോൾ തന്നെ പരിശോധന പ്രയോഗം തെറ്റായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയില്ല. ലൂപ്പിന്റെ ചട്ടക്കൂടിലെക്കുള്ള പ്രവേശനം while loop നിയന്ത്രിക്കുന്നുവെന്ന് ഇത് വ്യക്തമായി കാണിക്കുന്നു.

ആദ്യത്തെ 10 എണ്ണൽ സംഖ്യകളെ while loop ഉപയോഗിച്ച് പ്രിന്റ് ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്ക് നോക്കാം.

പ്രോഗ്രാം 7.11: ആദ്യത്തെ 10 എണ്ണൽ സംഖ്യകൾ പ്രിന്റ് ചെയ്യുന്നതിന്

```
#include<iostream>
using namespace std;
int main()
{
   int n = 1;
```

ലൂപ്പ് നിയന്ത്രണ വേരിയബളിന് പ്രാരംഭ വില നൽകൽ.

```
    while(n <= 10)
    {
        cout<< n << "  ";
        ++n;
    }
    return 0;
}
```

പരിശോധനാ പ്രയോഗം

ലൂപ്പിന്റെ ചട്ടക്കൂട്

ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വില പുതുക്കൽ

പ്രോഗ്രാം 7.11 ന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്ന പോലെ ആയിരിക്കും.

```
1  2  3  4  5  6  7  8  9  10
```

20 വരെയുള്ള ഇരട്ട സംഖ്യകളുടെ തുക കണ്ട്പിടിക്കുന്നതിന് പ്രോഗ്രാം 7.12 `while` ലൂപ്പ് ഉപയോഗിക്കുന്നു. ലൂപ്പ് വേരിയബിളിന്റെ വില ഏത് ഓപ്പറേഷനുപയോഗിച്ചും പരിഷ്കരിക്കാമെന്ന് ഈ പ്രോഗ്രാം കാണിക്കുന്നു.

പ്രോഗ്രാം 7.12: 20 വരെയുള്ള ഇരട്ടസംഖ്യകളുടെ തുക കണ്ടുപിടിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
int main()
{
    int i, sum = 0;
    i = 2;
    while( i<= 20)
    {
        sum = sum + i;
        i = i + 2;
    }
    cout<<"\nThe sum of even numbers up to 20 is: "<<sum;
    return 0;
}
```

നിലവിലുള്ള വിലയോട് രണ്ട് കൂട്ടി ച്ചേർത്തു കൊണ്ട് ലൂപ്പ് നിയന്ത്രണ വേരിയബിന്റെ വില പുതുക്കുന്നു.

പ്രോഗ്രാം 7.12 ന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
The sum of even numbers up to 20 is: 110
```

നമുക്ക് ചെയ്യാം

1. *100 - നും 200 - നും ഇടയിലുള്ള എല്ലാ ഒറ്റ സംഖ്യകളും പ്രദർശിപ്പിക്കുവാനായി പ്രോഗ്രാം 7.11 പരിഷ്കരിക്കുക.*
2. *പ്രോഗ്രാം 7.12 പരിഷ്കരിച്ച് ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ ശരാശരി കണ്ടുപിടിക്കുക.*

while പ്രസ്താവനയിലെ പരിശോധനാ പ്രയോഗത്തിന് ശേഷം നാം ഒരു അർധവിരാമം (;) ഇട്ടാൽ വാക്യഘടനയിൽ തെറ്റൊന്നുമില്ല. എന്നാൽ അതിനുശേഷമുള്ള ബ്രാക്കറ്റുകളിലെ പ്രസ്താവനകളെ ലൂപ്പ് ചട്ടക്കൂടായി പരിഗണിക്കുന്നില്ല. പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ while ലൂപ്പിനു ശേഷമുള്ള കോഡ് പ്രവർത്തിക്കുകയുമില്ല പ്രോഗ്രാം അവസാനിക്കുകയുമില്ല എന്നതാണ് ഏറ്റവും പരിതാപകരമായ അവസ്ഥ. ഇത് ഒരു അനന്തമായ ലൂപ്പിന് കാരണമാകുന്നു

7.2.2 for പ്രസ്താവന (for statement)

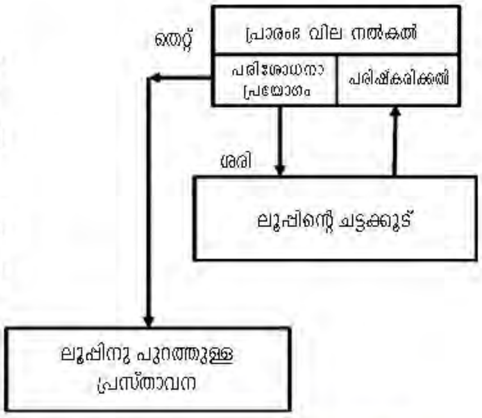
for ലൂപ്പും C++-ലെ ഒരു ആഗമന നിയന്ത്രണ ലൂപ്പ് ആണ്. ലൂപ്പിലെ ഘടകങ്ങളായ പ്രാരംഭ വില നൽകൽ, പരിശോധന പ്രയോഗം, പരിഷ്കരിക്കൽ പ്രസ്താവന എന്നിവ ഒരുമിച്ചാണ് for പ്രസ്താവനയിൽ നൽകിയിരിക്കുന്നത്. അതുകൊണ്ട് പ്രോഗ്രാം ഒതുക്കമുള്ളതായി തീരുന്നു. വാക്യ ഘടന ഇതാണ്:

```
for (പ്രാരംഭവില നൽകൽ; പരിശോധന പ്രയോഗം; പരിഷ്കരിക്കൽ പ്രസ്താവന)
{
     ലൂപ്പിന്റെ ചട്ടക്കൂട്;
}
for (initialisation; test expression; update statement)
{
     body-of-the-loop;
{
```

for ലൂപ്പിന്റെ പ്രവർത്തനം while ലൂപ്പിന്റേതുപോലെയാണ്. while ലൂപ്പിന്റെ ഫ്ളോചാർട്ട് for ലൂപ്പിന്റെ പ്രവർത്തനം വിശദമാക്കുന്നതിന് ഉപയോഗിക്കാവുന്നതാണ്.

for ലൂപ്പിൽ മൂന്നു ഘടകങ്ങളും ഒരുമിച്ചു വന്നതിനാൽ എണ്ണുന്ന (Counting) സാഹചര്യങ്ങളിൽ ഈ പ്രസ്താവന ഉപയോഗിക്കുന്നത് അഭികാമ്യമാണ്.

ചിത്രം 7.4 ൽ കൊടുത്തിരിക്കുന്ന ഫ്ളോചാർട്ട് സാധാരണയായി for പ്രസ്താവനയുടെ പ്രവർത്തനം കാണിക്കുന്നതിന് ഉപയോഗിക്കുന്നു.



ചിത്രം 7.4. ഫോർലൂപ്പിന്റെ പ്രവർത്തനം.

തുടക്കത്തിൽ, പ്രാരംഭ വില നൽകൽ നടക്കുന്നു, തുടർന്ന് പരിശോധനാ പ്രയോഗം വിലയിരുത്തുന്നു. ഇതിന്റെ ഫലം ശരിയാണെങ്കിൽ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. അല്ലെങ്കിൽ പ്രോഗ്രാം നിയന്ത്രണം ലൂപ്പിനു പുറത്തേക്കു പോകുന്നു. ലൂപ്പ് ചട്ടക്കൂടിന്റെ പ്രവർത്തനത്തിനുശേഷം പരിഷ്കരിക്കൽ പ്രയോഗം പ്രവർത്തിക്കുകയും പരിശോധനാ പ്രയോഗം വീണ്ടും വിലയിരുത്തുകയും ചെയ്യുന്നു. പരിശോധനാ പ്രയോഗം തെറ്റാവുന്നതു വരെ ഈ മൂന്നു ഘട്ടങ്ങളും (പരിശോധന, ചട്ടക്കൂട്, പരിഷ്കരിക്കൽ) തുടർന്നു കൊണ്ടേയിരിക്കും.

പ്രോഗ്രാം 7.11 ൽ ഉപയോഗിച്ചിരിക്കുന്ന ലൂപ്പ് ശകലത്തെ `for` ലൂപ്പ് ഉപയോഗിച്ച് താഴെ കാണും വിധം മാറ്റി എഴുതാം.

```
for (n=1; n<=10; ++n)
    cout << n << "  ";
```

`while` ലൂപ്പിലേതുപോലെ തന്നെ ഈ കോഡ് പ്രവർത്തിക്കുന്നു.

നമുക്ക് ചെയ്യാം

തൊട്ട് മുമ്പ് സൂചിപ്പിച്ച for ലൂപ്പിന്റെ പ്രവർത്തനക്രമത്തിലെ ഒന്നും രണ്ടും ഘട്ടങ്ങൾ താഴെ കൊടുത്തിരിക്കുന്നു. ബാക്കി ഘട്ടങ്ങൾ എഴുതുക.

ഘട്ടം 1 : n= 1, നിബന്ധന ശരിയാണ്, ഒന്ന് പ്രദർശിപ്പിക്കുന്നു, n ന്റെ വില 2 ആകുന്നു.

ഘട്ടം 2 : നിബന്ധനശരിയാണ് , 2 പ്രദർശിപ്പിക്കുന്നു, n ന്റെ വില 3 ആകുന്നു.

ഘട്ടം 3 : ....

`for` ലൂപ്പ് ഉപയോഗിച്ച് ഒരു സംഖ്യയുടെ ഫാക്റ്റോറിയൽ കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാം നമുക്കു എഴുതാം. N എന്ന സംഖ്യയുടെ ഫാക്റ്റോറിയൽ എന്നത് N! എന്ന് സൂചിപ്പിക്കുന്നു. ഇത് ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ ഗുണനഫലമാണ്. ഉദാഹരണത്തിന് 5 ന്റെ ഫാക്റ്റോറിയൽ (5!)കണക്കാക്കുന്നത് $1 \times 2 \times 3 \times 4 \times 5 = 120$ എന്നാണ്.

പ്രോഗ്രാം 7.13: for ലൂപ് ഉപയോഗിച്ച് ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{   int n, i;
    long fact=1;
    cout<<"Enter the number: ";
    cin>>n;
    for (i=1; i<=n; ++i)
        fact = fact * i;
    cout << "Factorial of " << n << " is " << fact;
    return 0;
}
```

പ്രാരംഭ വില നൽകൽ, പരിശോധനാ പ്രയോഗം, പുതുക്കൽ പ്രസ്താവന

ലൂപ് ചട്ടക്കൂട്

പ്രോഗ്രാം 7.13 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
Enter the number:  6
Factorial of 6 is 720
```

കമ്പ്യൂട്ടർ ആപ്ലിക്കേഷൻസ് എന്ന വിഷയത്തിലെ സ്കോറുകളുടെ ശരാശരി കാണുന്നതിനുള്ള മറ്റൊരു പ്രോഗ്രാമാണ് താഴെ കൊടുത്തിരിക്കുന്നത് പ്രോഗ്രാം 7.14-ൽ n നു(കുട്ടികളുടെ എണ്ണം) വില സ്വീകരിക്കുകയും പിന്നീട് ഓരോ വിദ്യാർഥികളേയും സ്കോർ ഇൻപുട്ടായി സ്വീകരിച്ച് ശരാശരി സ്കോർ പ്രിന്റ് ചെയ്യുന്നു.

പ്രോഗ്രാം 7.14 ■ വിദ്യാർഥികളുടെ ശരാശരി സ്കോർ കണ്ടുപിടിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
int main()
{
    int i, sum, score, n;
    float avg;
    cout << "How many students? ";
    cin >> n ;
    for( i=1, sum=0;  i<=n; ++i)
    {
        cout << "Enter the score of student " << i << ": ";
        cin >> score;
        sum = sum + score;
    }
    avg = (float)sum / n;
    cout << "Class Average: " << avg;
    return 0;
}
```

(i=1, sum=0): രണ്ട് വേരിയബിളുകൾക്ക് പ്രാരംഭ വില നൽകുന്നു

(float)sum: എക്സ്പ്ലിസിറ്റ് ടൈപ്പ് കൺവേർഷൻ

പ്രോഗ്രാം 7.14 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
How many students? 5
Enter the score of student 1: 45
Enter the score of student 2: 50
Enter the score of student 3: 52
Enter the score of student 4: 34
Enter the score of student 5: 55
Class Average: 47.2
```

പ്രോഗ്രാം 7.14-ൽ പ്രാരംഭ വില നൽകുന്ന പ്രസ്താവനയിൽ ഒരു കോമ ഉപയോഗിച്ച് വേർതിരിച്ച രണ്ട് പ്രയോഗങ്ങൾ (`i=1, sum=0`) അടങ്ങിയിരിക്കുന്നു. `i, sum` എന്നീ വേരിയബിളുകൾക്ക് അവയുടെ ആദ്യ വിലയായ `0,1` യഥാക്രമം കിട്ടുന്നു. `i<=n` എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും അത് ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ചതിനുശേഷം പരിഷ്കരിക്കൽ പ്രസ്താവനയായ `++i` പ്രവർത്തിക്കുന്നു. വീണ്ടും `i<=n` എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും നിബന്ധന ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. പരിശോധന പ്രയോഗം തെറ്റായ വില തിരിച്ചു തരുന്നതുവരെ ഈ പ്രക്രിയ തുടരുന്നു. മാതൃക ഔട്ട്പുട്ടിൽ ഇത് സംഭവിക്കുന്നത് `i`-യുടെ വില 6 ആകുമ്പോഴാണ്.

നമുക്ക് ചെയ്യാം

തന്നിരിക്കുന്ന സംഖ്യയുടെ ഗുണനപട്ടിക പ്രദർശിപ്പിക്കുവാനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. സംഖ്യ ഇൻപുട്ട് ചെയ്യുന്നത് n എന്ന വേരിയബിളിലാണെന്ന് കരുതുക. ലൂപ്പിന്റെ ചട്ടക്കൂട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
cout<<i<<" x " <<n<<" = "<< i * n << "\n";
```

ഔട്ട്പുട്ട് കൂടി കാണിക്കുക.

for ലൂപ്പ് ഉപയോഗിക്കുമ്പോൾ ചില കാര്യങ്ങൾ ശ്രദ്ധിക്കേണ്ടതുണ്ട്. തന്നിരിക്കുന്ന നാലു കോഡ് ശകലങ്ങൾ ഈ പ്രത്യേക സാഹചര്യത്തിൽ വിശദീകരിക്കുന്നു. കോഡിൽ ഉപയോഗിച്ചിട്ടുള്ള എല്ലാ വേരിയബിളുകളും int ഡാറ്റ ഇനത്തിലുള്ളതാണ് എന്ന് കരുതുക.

കോഡ് ശകലം 1:

```
for (n=1; n<5; n++);
    cout<<n;
```

for പ്രസ്താവനയുടെ ബ്രായ്ക്കറ്റ് കഴിഞ്ഞ് ഒരു അർധവിരാമം കാണപ്പെടുന്നു. ഇത് വാക്യഘടനയിലെ തെറ്റ് (syntax error) അല്ല. ഇതിന്റെ ഔട്ട്പുട്ട് നിങ്ങൾക്ക് പ്രവചിക്കാൻ കഴിയുമോ? 5 ആണെങ്കിൽ നിങ്ങൾ പറഞ്ഞത് ശരിയാണ്. ഈ ലൂപ്പിന് ചട്ടക്കൂട് ഇല്ല. പക്ഷേ ഇതിന്റെ പ്രവർത്തനം സാധാരണപോലെ പൂർത്തീകരിക്കുന്നു. പ്രാരംഭവില നൽകുന്ന പ്രസ്താവന n ന് 1 എന്ന വില നൽകുകയും നിബന്ധന വിലയിരുത്തുമ്പോൾ ശരിയാവുകയും ചെയ്യുന്നു. അവിടെ ലൂപ്പ് ചട്ടക്കൂട് ഇല്ലാത്തതിനാൽ പരിഷ്കരിക്കൽ പ്രസ്താവന പ്രവർത്തിക്കുകയും n ന്റെ വില 5 ആകുന്നതുവരെ ഈ പ്രവർത്തനം തുടരുകയും ചെയ്യുന്നു. ഈ സന്ദർഭത്തിൽ നിബന്ധന വിലയിരുത്തി തെറ്റാവുകയും പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിൽ നിന്നും പുറത്തു വരികയും ചെയ്യുന്നു. ഔട്ട്പുട്ട് പ്രസ്താവന പ്രവർത്തിക്കുമ്പോൾ സ്ക്രീനിൽ 5 എന്ന് പ്രദർശിപ്പിക്കുന്നു.

കോഡ് ശകലം 2:

```
for (n=1; n<5; )
    cout<<n;
```

ഈ കോഡിൽ പരിഷ്കരിക്കൽ പ്രസ്താവന (update expression) ഇല്ല. ഇത് കോഡിന്റെ വാക്യഘടനയിൽ തെറ്റ് ഉണ്ടാക്കുന്നില്ല. പക്ഷെ ലൂപ്പ് പ്രവർത്തിക്കുമ്പോൾ ഒരിക്കലും അവസാനിക്കുന്നില്ല. 1 എന്ന സംഖ്യ അനന്തമായി പ്രദർശിപ്പിക്കുന്നു. ഇതിനെ നമുക്ക് അനന്തമായ ലൂപ്പ് (Infinite loop) എന്നു വിളിക്കാം.

കോഡ് ശകലം 3:

```
for ( ; n<5; n++)
    cout<<n;
```

ഈ കോഡിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുവാൻ സാധ്യമല്ല. കാരണം നിയന്ത്രണവേരിയബിളിന് (Control Variable) പ്രാരംഭ വില നൽകിയിട്ടില്ല. അതിനാൽ നിയന്ത്രണ വേരിയബിൾ n -ന് ചില പൂർണസംഖ്യകൾ കിട്ടുന്നു. ചിലപ്പോൾ അത് 5-നെക്കാൾ കുറവാണെങ്കിൽ നിബന്ധന (condition) തെറ്റാവുന്നതുവരെ ചട്ടക്കൂട് പ്രവർത്തിക്കും. n ന്റെ തനത് വില 5-ഓ അതിൽ കൂടുതലോ ആണെങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കാതെ തന്നെ ലൂപ്പ് അവസാനിക്കുന്നു.

കോഡ് ശകലം 4:

```
for (n=1; ; n++)
    cout<<n;
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന കോഡിൽ പരിശോധന പ്രയോഗം (test expression) നൽകിയിട്ടില്ല. ഇത്തരം ഘട്ടത്തിൽ പരിശോധനാ പ്രയോഗത്തിന്റെ ഫലം ശരിയായി എടുക്കുകയും ലൂപ്പ് അനന്തമായി മാറുകയും ചെയ്യുന്നു.

മുകളിൽ കൊടുത്തിരിക്കുന്ന നാലു കോഡ് ശകലങ്ങളും സൂചിപ്പിക്കുന്നത് **for** ലൂപ്പിലെ എല്ലാ ഘടകങ്ങളും നിർബന്ധമില്ല എന്നാണ്. എന്നാൽ **while, do...while** പ്രസ്താവനകളുടെ കാര്യം ഇങ്ങനെയല്ല. ഈ രണ്ടു ലൂപ്പുകൾക്കും പരിശോധന പ്രയോഗങ്ങൾ നിർബന്ധമാണ്. എന്നാൽ മറ്റു ഘടകങ്ങൾ നിർബന്ധമില്ല. എന്നാൽ ഔട്ട്പുട്ട് സംബന്ധിച്ച് ജാഗ്രത പുലർത്തണം.

മറ്റൊരു വസ്തുത ശ്രദ്ധിക്കേണ്ടത് പരിശോധന പ്രയോഗത്തിനു പകരമായി നമുക്കു ഒരു സംഖ്യ നൽകുവാൻ സാധിക്കുമെന്നതാണ്. ഈ സംഖ്യ പൂജ്യമാണെങ്കിൽ പരിശോധന പ്രയോഗം തെറ്റായായും അല്ലെങ്കിൽ ശരിയായും ലൂപ്പ് പരിഗണിക്കും.

നമുക്ക് പരിശോധിക്കാം

1. 1 നും 49 നും ഇടയ്ക്കും എല്ലാ ഇരട്ടസംഖ്യകളുടെയും തുകയും ശരാശരിയും കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാം എഴുതുക.

2. 3 കൊണ്ടും 5 കൊണ്ടും ഹരിക്കാവുന്ന 10 നും 50 നും ഇടയ്ക്കുള്ള സംഖ്യകൾ പ്രദർശിപ്പിക്കുവാനുള്ള പ്രോഗ്രാം എഴുതുക.

3. താഴെ കൊടുത്തിരിക്കുന്ന കോഡിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുക.

```
for (i=1; i<=10; ++i);
cout<<i+2;
```

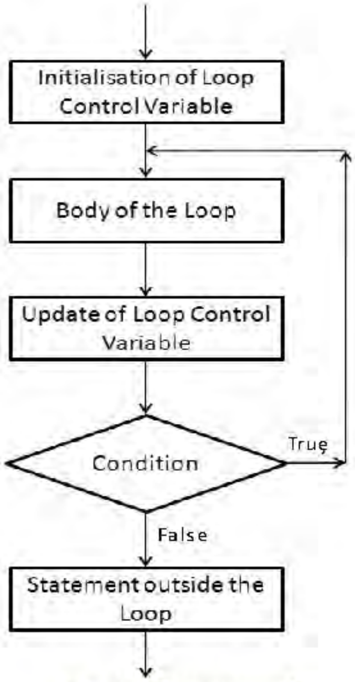
7.2.3 do...while പ്രസ്താവന (do...while statement)

for ലൂപ്പിന്റെയും , while ലൂപ്പിന്റെയും കാര്യത്തിൽ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നതിന് മുമ്പ് പരിശോധന പ്രയോഗം വിലയിരുത്തുന്നു. ആദ്യ തവണ തന്നെ പരിശോധന പ്രയോഗം തെറ്റാണെങ്കിൽ ലൂപ്പ് പ്രവർത്തിക്കില്ല. എന്നാൽ ചില സാഹചര്യങ്ങളിൽ പരിശോധന പ്രയോഗത്തിന്റെ ഫലം പരിഗണിക്കാതെ തന്നെ ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യമെങ്കിലും പ്രവർത്തിപ്പിക്കേണ്ടത് ആവശ്യമായി വരും. അത്തരം സാഹചര്യത്തിൽ **do...while** ലൂപ്പ് ഉപയോഗിക്കുന്നതാണ് നല്ലത്. do...while ലൂപ്പിന്റെ വാക്യഘടന (syntax) ഇതാണ്.

നിയന്ത്രണവേരിയബിളിന്റെ പ്രാരംഭ വില നൽകൽ;

```
do
{
ലൂപ്പിന്റെ ചട്ടക്കൂട്;
ലൂപ്പ് നിയന്ത്രണവേരിയബിളിന്റെ വില പുതുക്കൽ;
} while (പരിശോധന പ്രയോഗം) ;
```

```
initialisation of loop control variable;
do
{
      body of the loop;
      updation of loop control variable;
} while(test expression);
```



ചിത്രം 7.5: Execution of `do..while` loop

ചിത്രം 7.5-ൽ ഈ ലൂപ്പിന്റെ പ്രവർത്തന ക്രമം കാണിച്ചിരിക്കുന്നു. ഇവിടെ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിച്ചതിനുശേഷം മാത്രമാണ് പരിശോധന പ്രസ്താവന വിലയിരുത്തുന്നത്. അതിനാൽ `do...while` ലൂപ്പ് ഒരു ബഹിർഗമന നിയന്ത്രണ ലൂപ്പ് (`Exit controlled loop`) ആകുന്നു. പരിശോധന പ്രയോഗം തെറ്റാണെങ്കിൽ ലൂപ്പിന്റെ പ്രവർത്തനം അവസാനിക്കുന്നു. ഇത് അർഥമാക്കുന്നത് പരിശോധന പ്രയോഗത്തിന്റെ ഫലം പരിഗണിക്കാതെ തന്നെ ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യം പ്രവർത്തിക്കുന്നു എന്നാണ്.

`do...while` ലൂപ്പിന്റെ പ്രവർത്തനം വിശദീകരിക്കുന്നതിനായി താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലം നമുക്ക് പരിശോധിക്കാം.

```
int k=1;
do
{
    cout << k << '\t';
    ++k;
} while(k<=3);
 cout << "\n Program Ends";
```

ലൂപ് തുടങ്ങുന്നതിനുമുമ്പ് പ്രാരംഭ വില നൽകുന്നു.

ലൂപ്പിന്റെ ചട്ടക്കൂട്

ലൂപ്പിന്റെ ചട്ടക്കൂടിനകത്ത് വില പുതുക്കുന്നു

പരിശോധനാ പ്രയോഗം

ആദ്യം വേരിയബിൾ **k**-യുടെ വിലയായി 1 നൽകുന്നു. അതിനുശേഷം ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും **k** യുടെ വിലയായ 1 എന്ന് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. തുടർന്ന് **k**-യുടെ വില 1 വർദ്ധിപ്പിക്കുന്നു (ഇപ്പോൾ **k=2**). അതിനുശേഷം **k<=3** എന്ന വ്യവസ്ഥ പരിശോധിക്കുന്നു. ആ വ്യവസ്ഥ ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ച് **k**-യുടെ വില 2 എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. പുതുക്കൽ പ്രക്രിയ വീണ്ടും നടത്തി **k**-യുടെ വില 3 ആക്കുകയും **k<=3** എന്ന നിബന്ധന വീണ്ടും പരിശോധിക്കുകയും ചെയ്യുന്നു. നിബന്ധന ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിപ്പിച്ച് **k**-യുടെ വിലയായ 3 പ്രദർശിപ്പിക്കുന്നു. **k**-യുടെ വില വീണ്ടും പരിഷ്കരിച്ച് 4 ആകുന്നു. ഇത് പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന് പുറത്ത് വരുന്നതിനും തുടർന്നുള്ള പ്രസ്താവന പ്രവർത്തിക്കുന്നതിനും കാരണമാകുന്നു. ആയതിനാൽ കോഡിന്റെ ഔട്ട്പുട്ട് ഇങ്ങനെയായിരിക്കും.

```
1      2      3
```

ഈ ലൂപ്പ് മറ്റു രണ്ടു ലൂപ്പിൽ നിന്നും എങ്ങനെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു എന്ന് ഇപ്പോൾ നമുക്കു നോക്കാം. **k**-യുടെ പ്രാരംഭവില 5 ആണെന്ന് സങ്കൽപിക്കുക. എന്ത് സംഭവിക്കും? ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ച് **k**-യുടെ വിലയായ 5 സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. അതിനുശേഷം **k**-യുടെ വില ഒന്ന് വർദ്ധിപ്പിച്ച് 6 ആയി തീരുന്നു. **k<=3** എന്ന നിബന്ധന പരിശോധിച്ചപ്പോൾ പരിശോധന പ്രയോഗം തെറ്റാവുകയും നിയന്ത്രണം ലൂപ്പിന് പുറത്തേയ്ക്കു വരുകയും ചെയ്യുന്നു. `do...while` ലൂപ്പിന്റെ ചട്ടക്കൂടിലേക്ക് ആദ്യത്തെ പ്രാവശ്യം പ്രവേശിക്കുന്നതിന് യാതൊരു നിയന്ത്രണവും ഇല്ലെന്നാണ് ഇത് കാണിക്കുന്നത്. അതുകൊണ്ടു നിബന്ധനയുടെ ശരി (`True`) വില മാത്രം അനുസരിച്ചാണ് ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കേണ്ടതെങ്കിൽ `while` ലൂപ്പോ, `for` ലൂപ്പോ ഉപയോഗിക്കുക.

ഉപയോക്താവിന്റെ ആവശ്യത്തിനനുസരിച്ച് പ്രവർത്തിക്കുന്ന ഒരു പ്രോഗ്രാം നമുക്കു നോക്കാം. ഇത്തരം പ്രോഗ്രാമുകൾ ഉപയോക്താവിന്റെ പ്രതികരണം സ്വീകരിച്ചുകൊണ്ട് കോഡ് ശകലം ആവർത്തിച്ചു പ്രവർത്തിപ്പിക്കുന്നു.

ഉപഭോക്താവിൽ നിന്നും ഓരോ ചതുരത്തിന്റേയും നീളവും വീതിയും ഇൻപുട്ടായി സ്വീകരിച്ച് ചതുരങ്ങളുടെ വിസ്തീർണം കണ്ട്പിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം `do...while` ലൂപ്പ് ഉപയോഗിച്ച് എഴുതിയിരിക്കുന്നു. (പ്രോഗ്രാം 7.15)

പ്രോഗ്രാം 7.15 ചതുരത്തിന്റെ വിസ്തീർണം കാണുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    float length, breadth, area;
    char ch;
    do
    {
        cout << "Enter length and breadth: ";
        cin >> length >> breadth;
        area = length * breadth;
        cout << "Area = " << area;
        cout << "Any more rectangle (Y/N)? ";
        cin >> ch;
    } while (ch == 'Y' || ch == 'y');
    return 0;
}
```

പ്രോഗ്രാം 7.15 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുക്കുന്നു.

```
Enter length and breadth: 3.5     7
      Area = 24.5
      Any more rectangle (Y/N)? Y
      Enter length and breadth: 6     4.5
      Area = 27
```

ഉപയോക്താവ് ഇൻപുട്ട് നൽകുന്നു

ഉപയോക്താവ് ഇൻപുട്ട് നൽകുന്നു

```
Any more rectangle (Y/N)? N
```
ഉപയോക്താവ് ഇൻപുട്ട് നൽകുന്നു

C++ ലെ മൂന്ന് ലൂപ്പിങ്ങ് പ്രസ്താവനകളെക്കുറിച്ചും നാം ചർച്ച ചെയ്തു. പട്ടിക 7.2 ൽ ഈ പ്രസ്താവനകൾ താരതമ്യം ചെയ്തിരിക്കുന്നു.

for ലൂപ്പ്	while ലൂപ്പ്	do...while ലൂപ്പ്
ആഗമന നിയന്ത്രണ ലൂപ്പ് (Entry controlled loop)	ആഗമന നിയന്ത്രണ ലൂപ്പ് (Entry controlled loop)	ബഹിർഗമന നിയന്ത്രണ ലൂപ്പ് (Exit controlled loop)
ലൂപ്പിന്റെ നിർവചനത്തോടൊപ്പം തന്നെ പ്രാരംഭ വിലയും നൽകുന്നു.	ലൂപ്പ് നിർവചനത്തിനു മുമ്പ് പ്രാരംഭവില നൽകുന്നു.	ലൂപ്പ് നിർവചനത്തിനു മുമ്പ് പ്രാരംഭവില നൽകുന്നു.
ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യമെങ്കിലും പ്രവർത്തിക്കുമെന്ന് ഉറപ്പില്ല	ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യമെങ്കിലും പ്രവർത്തിക്കുമെന്ന് ഉറപ്പില്ല.	നിബന്ധന തെറ്റാണെങ്കിലും ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യം പ്രവർത്തിക്കും.

പട്ടിക 7.2: C++ ലൂപ്പ് പ്രസ്താവനകളുടെ താരതമ്യം

7.2.4 ലൂപ്പുകളുടെ നെസ്റ്റിങ്ങ് (Nesting of loops)

ഒരു ലൂപ്പിനകത്ത് മറ്റൊരു ലൂപ്പ് ഉൾപ്പെടുത്തുന്നതിനെ ലൂപ്പുകളുടെ നെസ്റ്റിങ്ങ് എന്നു പറയുന്നു. രണ്ട് ലൂപ്പുകൾ നാം നെസ്റ്റ് ചെയ്യുമ്പോൾ പുറത്തുള്ള ലൂപ്പ് (Outer loop) അകത്തുള്ള ലൂപ്പ് എത്ര തവണ പ്രവർത്തിച്ചു എന്ന് തിട്ടപ്പെടുത്തുന്നു. ഇവിടെ രണ്ടു ലൂപ്പുകളുടെയും ലൂപ്പ് നിയന്ത്രണ വേരിയബിളുകൾ (Loop control variable) വ്യത്യസ്തമായിരിക്കണം.

നെസ്റ്റഡ് ലൂപ്പ് എങ്ങനെ പ്രവർത്തിക്കുന്നു എന്ന് നമുക്കു നോക്കാം. ഒരു ക്ലോക്കിലെ മിനുട്ട് സൂചിയുടെയും, സെക്കന്റ് സൂചിയുടെയും കാര്യം എടുക്കുക. നിങ്ങൾ ക്ലോക്കിന്റെ പ്രവർത്തനം ശ്രദ്ധിച്ചിട്ടുണ്ടോ? മിനുട്ട് സൂചി ഏതെങ്കിലും ഒരു സ്ഥാനത്ത് നിൽക്കുമ്പോൾ സെക്കന്റ് സൂചി ഒരു ഭ്രമണം പൂർത്തിയാക്കുന്നു (1 മുതൽ 60 വരെ). സെക്കന്റ് സൂചി ഒരു ഭ്രമണം പൂർത്തിയാക്കിയതിനു ശേഷം മിനുട്ട് സൂചി അടുത്ത സ്ഥാനത്തേക്ക് മാറുന്നു. മിനുട്ട് സൂചിയുടെ ഓരോ സ്ഥാനത്തിനും അനുസൃതമായി സെക്കന്റ് സൂചി കറക്കം പൂർത്തിയാക്കുന്നു. ഈ പ്രക്രിയ തുടർന്നു കൊണ്ടേയിരിക്കുന്നു. ഇവിടെ സെക്കന്റ് സൂചിയുടെ ചലനം ഉള്ളിലെ ലൂപ്പിന്റെ പ്രവർത്തനമായും മിനുട്ട് സൂചിയുടെ ചലനം ബാഹ്യലൂപ്പിന്റെ പ്രവർത്തനമായും കരുതാവുന്നതാണ്. C++ ലെ എല്ലാ ലൂപ്പുകളും നെസ്റ്റിങ്ങ് അനുവദിക്കുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം for ലൂപ്പിന്റെ നെസ്റ്റിങ്ങ് പ്രവർത്തനം കാണിച്ചുതരുന്നു.

```
for( i=1; i<=2; ++i)        ← ബാഹ്യ ലൂപ്പ്
{
     for(j=1; j<=3; ++j)     ← ആന്തരിക ലൂപ്പ്
     {
          cout<< "\n" << i << " and " << j;
     }
}
```

ബാഹ്യലൂപ്പിലെ വേരിയബിളായ `i` ക്ക് പ്രാരംഭ വിലയായി 1 നൽകുന്നു. അതിന്റെ പരിശോധന പ്രയോഗം വിലയിരുത്തി ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. ചട്ടക്കൂടിൽ അടങ്ങിയിരിക്കുന്നത് നിയന്ത്രണ വേരിയബിൾ `j` യോടുകൂടിയ ആന്തരിക ലൂപ്പാണ്. `j` ക്ക് പ്രാരംഭ വിലയായ 1 നൽകി അതിന്റെ പ്രവർത്തനം ആരംഭിക്കുന്നു. `j` =1, `j`=2, `j`=3 ആയി ആന്തരിക ലൂപ്പ് 3 തവണ പ്രവർത്തിക്കുന്നു. ഓരോ തവണയും **`j<=3`** എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും ശരിയായതിനാൽ ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു.

```
1 and 1
1 and 2
1 and 3
```

ആദ്യത്തെ 1, i യുടെ വിലയും രണ്ടാമത്തെ 1, j യുടെ വിലയുമാണ്

പരിശോധന പ്രയോഗം **`j<=3`** തെറ്റാവുമ്പോൾ പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ആന്തരിക ലൂപ്പിൽ നിന്നും പുറത്തു കടക്കുന്നു. ഇപ്പോൾ ബാഹ്യ ലൂപ്പിന്റെ പുതുക്കൽ പ്രസ്താവന പ്രവർത്തിച്ച് `i`=2 ആക്കുന്നു. പരിശോധന പ്രയോഗമായ **`i<=2`** പരിശോധിച്ച് ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒന്നുകൂടി പ്രവർത്തിക്കുന്നു. `j` =1, `j`=2, `j`=3 ആയി ആന്തരിക ലൂപ്പ് വീണ്ടും മൂന്നു തവണ പ്രവർത്തിച്ച് ഔട്ട് പുട്ട് പ്രദർശിപ്പിക്കുന്നു.

```
2 and 1
2 and 2
2 and 3
```

ആന്തരിക ലൂപ്പിന്റെ പ്രവർത്തനം പൂർത്തിയാക്കിയതിനുശേഷം നിയന്ത്രണം പുറത്തെ ലൂപ്പിന്റെ വില പുതുക്കൽ പ്രയോഗത്തിൽ തിരിച്ചെത്തുന്നു. `i` യുടെ വില 1 വെച്ച് വർദ്ധിപ്പിക്കുന്നു. (ഇപ്പോൾ `i`=3) പരിശോധന പ്രയോഗം **`i<=2`** വിലയിരുത്തുമ്പോൾ തെറ്റാവുന്നു. ആയതിനാൽ ലൂപ്പ് അതിന്റെ പ്രവർത്തനം അവസാനിപ്പിക്കുന്നു. പട്ടിക 7.3 മുകളിൽ കൊടുത്ത പ്രോഗ്രാം ശകലത്തിന്റെ പ്രവർത്തനം വിവരിക്കുന്നു.

ആവർത്തനം	ബാഹ്യ ലൂപ്പ്	ആന്തരികലൂപ്പ്	ഔട്ട്പുട്ട്
1	1	1	1 and 1
2	1	2	1 and 2
3	1	3	1 and 3
4	2	1	2 and 1
5	2	2	2 and 2
6	2	3	2 and 3

പട്ടിക 7.3: നെസ്റ്റഡ് ലൂപ്പിന്റെ പ്രവർത്തനം

നെസ്റ്റഡ് ലൂപ്പുകളിൽ പ്രവർത്തിക്കുന്ന സമയത്ത് ബാഹ്യലൂപ്പിലെ നിയന്ത്രണ വേരിയബിളുകളിൽ അവയുടെ വിലയിൽ മാറ്റം വരുന്നത് ആന്തരികലൂപ്പ് പൂർത്തികരിച്ചതിനുശേഷം മാത്രമാണ്.

ഇനി താഴെ കൊടുത്തിരിക്കുന്ന രീതിയിലുള്ള ത്രികോണം പ്രദർശിപ്പിക്കാനുള്ള ഒരു പ്രോഗ്രാം നമുക്കു എഴുതാം.

```
*
* *
* * *
* * * *
* * * * *
```

പ്രോഗ്രാം 7.16: ത്രികോണാകൃതിയിൽ നക്ഷത്രചിഹ്നം പ്രദർശിപ്പിക്കുന്നതിന്.

```
#include<iostream>
using namespace std;
int main()
{   int i, j;
    char ch = '*';
    for(i=1; i<=5; ++i)          //ബാഹ്യ ലൂപ്പ്
    {
        cout<< "\n" ;
        for(j=1; j<=i; ++j)      // ആന്തരിക ലൂപ്പ്
            cout<<ch;
    }
    return 0;
}
```

നമുക്ക് ചെയ്യാം

1. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലത്തിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുക.

```
sum = 0;
for( i=1; i<3; ++i)
{
   for(j=1; j<3; ++j)
   {
      sum = sum + i * j;
   }
 cout<<sum ;
```

2. താഴെ കൊടുത്തിരിക്കുന്ന ത്രികോണങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക.

```
1              1
2  2           1  2
3  3  3        1  2  3
4  4  4        1  2  3  4
5  5  5  5     1  2  3  4  5
```

7.2.5 നിയന്ത്രണ പ്രസ്താവനകളുടെ നെസ്റ്റിങ്ങ് (Nesting of Control Statements)

നാം ലൂപ്പുകളുടെയും പ്രസ്താവനകളുടെയും നെസ്റ്റിങ്ങിനെകുറിച്ച് ചർച്ച ചെയ്തു. നിയന്ത്രണ പ്രസ്താവന മറ്റൊരു നിയന്ത്രണ പ്രസ്താവന ഉപയോഗിച്ച് നെസ്റ്റ് ചെയ്യാം. ഒരു ലൂപ്പിൽ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകളായ `if`, `switch` എന്നിവ അടങ്ങിയിരിക്കാം. അതുപോലെ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകളിൽ ലൂപ്പു പ്രസ്താവനകളായ `while`, `for`, `do...while` എന്നിവയും അടങ്ങിയിരിക്കാം. പ്രോഗ്രാം 7.17 ൽ ലൂപ്പും അതിന്റെ ചട്ടക്കൂടും ഒരു `switch` പ്രസ്താവനയും ഉൾപ്പെട്ടിരിക്കുന്നു. ഇത് സാധാരണയായിട്ടുള്ള ഒരു മെനു നിയന്ത്രിത ശൈലിയുള്ള പ്രോഗ്രാമാണ്.

പ്രോഗ്രാം 7.17 രണ്ട് സംഖ്യകൾ സ്വീകരിച്ച് ഉപയോക്താവിന്റെ താല്പര്യത്തിന് അടിസ്ഥാനമായി ഗണിത ക്രിയകൾ ചെയ്യൽ.

```
#include<iostream>
using namespace std;
int main()
  {
    char ch;
    float n1, n2;
    cout<<"Enter two numbers: ";
    cin>>n1>>n2;
    do
    {
         cout<<"\nNumber 1: "<<n1<<"\tNumber 2: "<<n2;
         cout<<"\n\t\tOperator Menu";
         cout<<"\n\t1. Addition (+)";
         cout<<"\n\t2. Subtraction (-)";
         cout<<"\n\t3. Multiplication (*)";
         cout<<"\n\t4. Division (/)";
         cout<<"\n\t5. Exit (E)";
         cout<<"\nEnter Option number or operator: ";
         cin>>ch;
         switch(ch)
         {
            case '1' :
            case '+' : cout<<n1<<" + "<<n2<<" = "<<n1+n2;
                       break;
            case '2' :
            case '-' : cout<<n1<<" - "<<n2<<" = "<<n1-n2;
                       break;
            case '3' :
            case '*' : cout<<n1<<" * "<<n2<<" = "<<n1*n2;
                       break;
```

```
            case '4' :
            case '/' : cout<<n1<<" / "<<n2<<" = "<<n1/n2;
                       break;
            case '5' :
            case 'E' :
            case 'e' : cout<<"Thank You for using the program";
                       break;
            default  : cout<<"Invalid Choice!!";
        }
    } while (ch!='5' && ch!='E' && ch!='e');
    return 0;
}
```

പ്രോഗ്രാം 7.17 ന്റെ മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുക്കുന്നു:

```
Enter two numbers: 25    4
Number 1: 25          Number 2: 4
              Operator Menu
      1. Addition (+)
      2. Subtraction (-)
      3. Multiplication (*)
      4. Division (/)
      5. Exit (E)
Enter Option number or operator: 1
25 + 4 = 29

Number 1: 25          Number 2: 4
              Operator Menu
      1. Addition (+)
      2. Subtraction (-)
      3. Multiplication (*)
      4. Division (/)
      5. Exit (E)
Enter Option number or operator: /
25 / 4 = 6.25

Number 1: 25          Number 2: 4
              Operator Menu
      1. Addition (+)
      2. Subtraction (-)
      3. Multiplication (*)
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

```
      4. Division (/)
      5. Exit (E)
   Enter Option number or operator: 5
   Thank You for using the program
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

കൺട്രോൾ പ്രസ്താവനകളുടെ നെസ്റ്റിങ്ങിന്റെ വിവിധ സംയോഗങ്ങൾ ഉപയോഗിക്കുന്ന കൂടുതൽ പ്രോഗ്രാമുകൾ പ്രോഗ്രാം ഗ്യാലറി വിഭാഗത്തിൽ നാം ചർച്ച ചെയ്യും.

7.3 ജമ്പ് പ്രസ്താവനകൾ (Jump Statements)

പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ഒരു ഭാഗത്തുനിന്നും മറ്റൊരു ഭാഗത്തേക്ക് മാറ്റാൻ ഉപയോഗിക്കുന്ന പ്രസ്താവനകളെ ജമ്പ് പ്രസ്താവനകൾ (Jump statements) എന്നു പറയുന്നു. C++ ൽ പ്രത്യേക നിബന്ധനകളില്ലാതെ പ്രവർത്തിക്കുന്ന നാലുതരം ജമ്പ് പ്രസ്താവനകൾ ഉണ്ട്. അവ **return**, **goto**, **break**, **continue** എന്നിവയാണ് ഇതിനുപുറമെ, C++ ലെ **exit()** എന്ന സ്റ്റാൻഡേർഡ് ലൈബ്രറി ഫങ്ഷൻ പ്രോഗ്രാമിന്റെ പ്രവർത്തനം അവസാനിപ്പിക്കുന്നതിനും ഉപയോഗിക്കുന്നുണ്ട്.

return പ്രസ്താവന ഫങ്ഷനിൽ നിന്ന് പുറത്ത് വരുന്നതിനും നിയന്ത്രണം, വിളിച്ച പ്രോഗ്രാമിലേക്ക് തിരിച്ചു കൊണ്ടു പോകുന്നതിനും ഉപയോഗിക്കുന്നു. അധ്യായം 10 ൽ ഇതിനെക്കുറിച്ച് പിന്നീട് വിശദീകരിച്ചിട്ടുണ്ട്. ഇനി നമുക്ക് മറ്റു ജമ്പ് പ്രസ്താവനകളെക്കുറിച്ച് ചർച്ച ചെയ്യാം.

7.3.1 goto പ്രസ്താവന

goto പ്രസ്താവന ഉപയോഗിച്ച് പ്രോഗ്രാം നിയന്ത്രണത്തെ ഫങ്ഷനിലെ ഏതു സ്ഥലത്തേക്കും മാറ്റാൻ സാധിക്കും. ഒരു **goto** പ്രസ്താവനയുടെ ലക്ഷ്യസ്ഥാനം ലേബൽ (ഒരു ഐഡന്റിഫയർ) ഉപയോഗിച്ച് അടയാളപ്പെടുത്തുന്നു.

goto പ്രസ്താവനയുടെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
         goto ലേബൽ ;
         .............;
         .............;
ലേബൽ:   ...........;
         .............;


         goto label;
         .............;
         .............;
label:   ...........;
         .............;
```

goto പ്രസ്താവനക്ക് മുമ്പോ പിൻപോ ഒരു പ്രോഗ്രാമിൽ കാണപ്പെടുന്നു. ലേബലിനുശേഷം ഒരു അപൂർണവിരാമം (**:**) ചിഹ്നം ആവശ്യമാണ്. ഉദാഹരണത്തിന് 1 മുതൽ 50 വരെ പ്രിന്റ് ചെയ്യാനുള്ള കോഡ് ശകലം പരിഗണിക്കുക.

```
      int i=1;
start:                ലേബൽ
      cout<<i;
      ++i;
      if (i<=50)
           goto start;
```

ഇവിടെ cout, പ്രസ്താവന 1 എന്ന വില പ്രിന്റ് ചെയ്യുന്നു. അതിനുശേഷം i യുടെ വില 1 വർദ്ധിപ്പിക്കുന്നു. (ഇപ്പോൾ i=2), ഇപ്പോൾ പരിശോധന പ്രയോഗം i<=50 വിലയിരുത്തുന്നു. നിബന്ധന ശരിയായതിനാൽ start എന്ന ലേബലിലേക്ക് പ്രോഗ്രാം നിയന്ത്രണത്തെ മാറ്റുന്നു. നിബന്ധന തെറ്റാവുമ്പോൾ പ്രവർത്തനം അവസാനിപ്പിച്ച് പ്രോഗ്രാം നിയന്ത്രണം if പ്രസ്താവനക്കു ശേഷം എത്തുന്നു.

നമുക്കു മറ്റൊരു ഉദാഹരണം നോക്കാം. ഇവിടെ ഒരു സംഖ്യ സ്വീകരിക്കുകയും മുൻകൂട്ടി നിശ്ചയിച്ച വിലയുമായി താരതമ്യം ചെയ്യുകയും ചെയ്യുന്നു. തുല്യമാണെങ്കിൽ പ്രോഗ്രാം തുടരും അല്ലെങ്കിൽ അത് അവസാനിക്കുന്നു.

```
  int p;
  cout<<"Enter the Code: ";
  cin>>p;
  if(p!=7755)
     goto end;
  cout<<"Enter the details";
  ........................;
  ........................;              ലേബൽ
end:
  cout<<"Sorry, the code number is wrong. Try again!";
```

ഇവിടെ ഉപയോക്താവ് ഇൻപുട്ടിന്റെ സാധുത പരിശോധിക്കുന്നു. കോഡ് സാധുവായതാണെങ്കിൽ പ്രോഗ്രാം മറ്റു വിശദാംശങ്ങൾ സ്വീകരിക്കുന്നു. ഇല്ലെങ്കിൽ നിയന്ത്രണം end എന്ന ലേബലിലേക്ക് പോകുന്നു. സ്ട്രക്ചേർഡ് പ്രോഗ്രാമിങ് goto ന്റെ ഉപയോഗം പ്രോത്സാഹിപ്പിക്കുന്നില്ല.

7.3.2 break (ബ്രേക്ക്) പ്രസ്താവന

ഒരു പ്രോഗ്രാമിൽ break പ്രസ്താവന കാണപ്പെട്ടാൽ പ്രോഗ്രാമിന്റെ നിയന്ത്രണം തൊട്ടടുത്ത ലൂപ്പിനോ (for, while, do...while), switch പ്രസ്താവനയ്ക്കോ പുറത്തേക്ക് മാറ്റുന്നു. കൺട്രോൾ ചട്ടക്കൂടിന് ശേഷമുള്ള പ്രസ്താവന മുതൽ പ്രവർത്തനം തുടരുന്നു. switch പ്രസ്താവനയിൽ break ന്റെ പ്രവാഹത്തെ കുറിച്ച് നാം ഇതിനോടകം ചർച്ച ചെയ്തു കഴിഞ്ഞു. ഇത് ലൂപ്പുകളുടെ പ്രവർത്തനത്തെ എങ്ങനെ സ്വാധീനിക്കുന്നു എന്ന് നമുക്ക് നോക്കാം. താഴെ കൊടുത്തിരിക്കുന്ന രണ്ട് പ്രോഗ്രാം ശകലങ്ങൾ പരിഗണിക്കുക.

കോഡ് ശകലം 1

```
i=1;
while(i<=10)
{
    cin>>num;
    if (num==0)
        break;
    cout<<"Entered number is: "<<num;
    cout<<"\nInside the loop";
    ++i;
}
cout<<"\nComes out of the loop";
```

മുകളിലെ പ്രോഗ്രാം 10 സംഖ്യകളെ ഇൻപുട്ട് ചെയ്യാൻ അനുവദിക്കുന്നു. ഇൻപുട്ട് ചെയ്യുമ്പോൾ ഏതെങ്കിലും ഒരു സംഖ്യ 0 ആണെങ്കിൽ ലൂപ്പ് ചട്ടക്കൂടിലെ ബാക്കി പ്രസ്താവനകളെ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിനു പുറത്ത് വരികയും "Comes out of the loop" എന്ന സന്ദേശം സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. ഒരു നെസ്റ്റഡ് ലൂപ്പിൽ break പ്രസ്താവനഉപയോഗിക്കുന്ന മറ്റൊരു കോഡ് ശകലം നമുക്കു പരിഗണിക്കാം.

കോഡ് ശകലം 2

```
for(i=1; i<=5; ++i)    //outer loop
{
    cout<<"\n";
    for(j=1; j<=i; ++j)     //inner loop
    {
        cout<<"* ";
        if (j==3)
            break;
    }
}
```

ഈ കോഡ് ശകലം താഴെ കൊടുത്തിരിക്കുന്ന മാതൃക പ്രദർശിപ്പിക്കുന്നു.

```
*
*  *
*  *  *
*  *  *
*  *  *
```

j യുടെ വില എപ്പോൾ മൂന്ന് ആകുന്നുവോ അപ്പോൾ ആന്തരിക ലൂപ്പിന്റെ പ്രവർത്തനം അവസാനിക്കുന്നു.

നെസ്റ്റഡ് ലൂപ്പ് സാധാരണയായി i=1, i=2, i=3 എന്നീ വിലകൾക്ക് അനുസരിച്ച് പ്രവർത്തിക്കുന്നു. i യുടെ ഓരോ വിലക്കനുസരിച്ച് j, 1 മുതൽ i വരെയുള്ള വിലകൾ സ്വീകരിക്കും. i യുടെ വില 4 ഓ 5 ഓ ആകുമ്പോൾ ഉള്ളിലുള്ള ലൂപ്പ് j = 1, j=2, j=3 എന്നീ വിലകൾക്കനുസരിച്ച് പ്രവർത്തിച്ച് break നു ശേഷം ലൂപ്പിൽ നിന്നും പുറത്ത് പോകുന്നു.

7.3.3 continue (കൺഡിന്യൂ) പ്രസ്താവന

continue പ്രസ്താവന മറ്റൊരു ജംപ് പ്രസ്താവനയാണ് അത് ലൂപ്പ് ചട്ടക്കൂടിന്റെ ഒരു ഭാഗം ഒഴിവാക്കി അടുത്ത ആവർത്തനത്തിലേക്ക് എത്തിക്കുന്നതിനു വേണ്ടി ഉപയോഗിക്കുന്നു. `break` പ്രസ്താവന ലൂപ്പിന്റെ പ്രവർത്തനം നിർത്തി വെയ്ക്കുമ്പോൾ `continue` പ്രസ്താവന ചില ഭാഗങ്ങൾ ഒഴിവാക്കി അടുത്ത ആവർത്തനം നടത്താൻ നിർബന്ധിക്കുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലം `continue` പ്രസ്താവനയുടെ പ്രവർത്തനം വിവരിക്കുന്നു.

```
for (i=1; i<=10; ++i)
    {
        if (i==6)
            continue;
        cout<<i<<"\t";
    }
```

ഈ കോഡ് താഴെ പറയുന്ന ഔട്ട്പുട്ട് തരുന്നു.

```
1    2    3    4    5    7    8    9    10
```

6 ലിസ്റ്റിൽ ഇല്ല എന്നത് ശ്രദ്ധിക്കുക. `i` യുടെ വില 6 ആകുമ്പോഴാണ് continue പ്രസ്താവന പ്രവർത്തിക്കുന്നത്. അതിന്റെ ഫലമായി ഔട്ട്പുട്ട് പ്രസ്താവന ഒഴിവാക്കി പ്രോഗ്രാം നിയന്ത്രണം അടുത്ത ആവർത്തനത്തിലെ പുതുക്കൽ പ്രസ്താവനയിൽ എത്തിച്ചേരുന്നു.

ഒരു ലൂപ്പിനകത്തെ `break` പ്രസ്താവന ലൂപ്പിനെ അവസാനിപ്പിക്കുകയും ലൂപ്പിനു ശേഷമുള്ള പ്രസ്താവനകളിലേക്ക് പ്രോഗ്രാം നിയന്ത്രണത്തെ എത്തിക്കുകയും ചെയ്യുന്നു. `continue` പ്രസ്താവന നിലവിലുള്ള ആവർത്തനത്തിലെ ശേഷിച്ച ഭാഗം ഉപേക്ഷിച്ച് ലൂപ്പിന്റെ അടുത്ത ആവർത്തനം ആരംഭിക്കുന്നു. `While` ലൂപ്പിലും, `do...while` ലൂപ്പിലും `continue` പ്രസ്താവന ഉപയോഗിക്കുമ്പോൾ ലൂപ്പ് അനന്തമാകുന്നത് ഒഴിവാക്കണം എന്നത് ശ്രദ്ധിക്കണം. പട്ടിക 7.4 `break`, `continue` എന്നീ പ്രസ്താവനകൾ തമ്മിലുള്ള താരതമ്യം കാണിക്കുന്നു.

break പ്രസ്താവന	continue പ്രസ്താവന
switch ന്റെയും ലൂപ്പിന്റെയും കൂടെ ഉപയോഗിക്കാം.	ലൂപ്പിന്റെ കൂടെ മാത്രം ഉപയോഗിക്കുന്നു.
ബ്ലോക്കിലെ അവശേഷിക്കുന്ന പ്രസ്താവനകൾ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം സ്വിച്ചിനോ ലൂപ്പിനോ പുറത്തേക്കു കൊണ്ടു വരുന്നു.	ബ്ലോക്കിലെ അവശേഷിക്കുന്ന പ്രസ്താവനകൾ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന്റെ ആരംഭത്തിലേക്ക് കൊണ്ടു വരുന്നു.
പരിശോധന പ്രസ്താവന ശരിയാണങ്കിലും പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിനു പുറത്തു പോകുന്നു.	പരിശോധന പ്രസ്താവനയുടെ വില തെറ്റാകുമ്പോൾ മാത്രം പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന് പുറത്തു കൊണ്ടു പോകുന്നു.

പട്ടിക 7.4 `break`, `continue` എന്നീ പ്രസ്താവനകൾ തമ്മിലുള്ള താരതമ്യം

ഒരു പ്രോഗ്രാം അതിന്റെ തന്നെ പ്രവർത്തനം നിർത്തുന്നതിന് **`exit()`** എന്ന ഒരു ബിൽട്ട്-ഇൻ ഫങ്ഷൻ C++ ൽ ഉപയോഗിക്കുന്നു. **`cstdlib`** എന്ന ഹെഡർ ഫയൽ (ടർബോ C++ ൽ process .h) പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തിയാൽ മാത്രമേ **`exit ()`** എന്ന ഫങ്ഷൻ ഉപയോഗിക്കാൻ കഴിയും. പ്രോഗ്രാം 7.18 ഈ ഫങ്ഷന്റെ പ്രവർത്തനം വിശദീകരിക്കുന്നു.

പ്രോഗ്രാം 7.18: തന്നിരിക്കുന്ന സംഖ്യ അഭാജ്യ സംഖ്യയാണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    int i, num;
    cout<<"Enter the number: ";
    cin>>num;
    for(i=2; i<=num/2; ++i)
    {
        if(num%i == 0)
        {
            cout<<"Not a Prime Number";
            exit(0);
        }
    }
    cout<<"Prime Number";
    return 0;
}
```

പ്രോഗ്രാം 7.18 ലെ `for` ലൂപ്പിലെ പരിശോധന പ്രസ്താവന `i<=sqrt(num)` ഉപയോഗിച്ച് മാറ്റി എഴുതാം. ഇവിടെ `sqrt()` എന്നത് തന്നിരിക്കുന്ന സംഖ്യയുടെ വർഗ്ഗമൂലം കണ്ടുപിടിക്കുനന്നതിനുള്ള ഒരു ഫങ്ഷനാണ്. ഒരു സംഖ്യക്ക് 2 മുതൽ അതിന്റെ വർഗ്ഗമൂലം വരെ ഘടകങ്ങൾ ഇല്ലെങ്കിൽ അത് ഒരു അവിഭാജ്യ (prime) സംഖ്യയാണ്. `sqrt()` ഉപയോഗിക്കുന്നതിന് `#include<cmath>` എന്ന പ്രസ്താവന ഉൾപ്പെടുത്തണം.

പ്രോഗ്രാം 7.18 ന്റെ മാതൃക ഔട്ട്പുട്ടുകൾ താഴെ കാണിച്ചിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1

```
Enter the number: 17
Prime Number
```

ഔട്ട്പുട്ട് 2

```
Enter the number: 18
Not a Prime Number
```

നമുക്ക് പരിശോധിക്കാം

1. goto പ്രസ്താവന താഴെ പറയുന്ന ഏതിലേക്ക് നിയന്ത്രണം പോകുന്നതിന് കാരണമാകുന്നു.
   (a) ഒരു ഓപ്പറേറ്റർ (b) ഒരു ലേബൽ (c) ഒരു വേരിയബിൾ
   (d) ഒരു ഫങ്ഷൻ
2. break പ്രസ്താവന ഏതിൽ നിന്ന് പുറത്തുപോകാൻ കാരണമാകുന്നു.
   (a) ഏറ്റവും അകത്തുള്ള ലൂപ്പിൽ നിന്ന് മാത്രം
   (b) ഏറ്റവും ഉള്ളിലുള്ള switch ൽ നിന്ന് മാത്രം
   (c) എല്ലാ ലൂപ്പിൽ നിന്നും, switch ൽ നിന്നും
   (d) ഏറ്റവും അകത്തുള്ള ലൂപ്പിൽ നിന്നോ സ്വിച്ചിൽ നിന്നോ
3. exit( ) ഫങ്ഷൻ ഏതിൽ നിന്ന് പുറത്തേക്ക് എത്തിക്കുന്നു.
   (a) അത് കാണപ്പെടുന്ന ഫങ്ഷനിൽ നിന്നും
   (b) അത് കാണപ്പെടുന്ന ലൂപ്പിൽ നിന്നും
   (c) അത് കാണപ്പെടുന്ന ബ്ലോക്കിൽ നിന്നും
   (d) അത് കാണപ്പെടുന്ന പ്രോഗ്രാമിൽ നിന്നും
4. exit( ) ഫങ്ഷൻ ഉപയോഗിക്കുന്നതിന് ഉൾപ്പെടുത്തത്തേണ്ട ഹെഡർ ഫയലിന്റെ പേരെഴുതുക.

പ്രോഗ്രാം ഗാലറി

ഈ പാഠഭാഗത്ത് പ്രശ്ന പരിഹാരത്തിനായി വിവിധ നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിച്ചുള്ള പ്രോഗ്രാമുകളുടെ ശേഖരമാണ് ഉള്ളത്. പ്രോഗ്രാമുകളുടെ മാതൃക ഔട്ട്പുട്ടുകൾ ഓരോ പ്രോഗ്രാമിനും ശേഷവും കൊടുത്തിട്ടുണ്ട്.

പ്രോഗ്രാം 7.19 $ax^2 + bx + c = 0$ എന്ന രൂപത്തിലുള്ള ഒരു ദ്വിമാന സമവാക്യത്തിന്റെ മൂന്ന് കോയിഫിഷന്റുകൾ സ്വീകരിക്കുകയും അതിന്റെ മൂല്യസംഖ്യ കണക്കാക്കുകയും ചെയ്യുന്നു. a യുടെ വില 0 (പൂജ്യം) ആകരുത്. ഈ പ്രശ്നം നിർദ്ധാരണം ചെയ്യുന്നതിന് $(b^2 - 4ac)$ എന്ന സൂത്രവാക്യം ഉപയോഗിച്ച് ദ്വിമാന സമവാക്യത്തിന്റെ ഡിസ്ക്രിമിനിനെന്റ് മൂല്യം കണ്ടുപിടിക്കണം. മൂല്യസംഖ്യയുടെ തരം കണ്ടുപിടിക്കുന്നതിന് വർഗമൂലം കണ്ടുപിടിക്കുന്നതിനുള്ള സൂത്രവാക്യവും ലഭ്യമാണ്. ഈ പ്രോഗ്രാമിൽ sqrt () എന്ന ഫങ്ഷനാണ് സംഖ്യയുടെ വർഗമൂലം കണ്ടുപിടിക്കാൻ ഉപയോഗിക്കുന്നത്. ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നതിന് cmath (ടർബോ C++ ൽ math.h) എന്ന ഹെഡർ ഫയൽ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ടതാണ്.

Program 7.19: To find the roots of a quadratic equation

```
#include <iostream>
#include <cmath> // to use sqrt()function
using namespace std;
int main()
```

```
{
    float a, b, c, root1, root2, d;
    cout<< "Enter the three coefficients: ";
    cin >> a >> b >> c ;
    if (!a)  // equivalent to if (a == 0)
        cout<<"Value of \'a \' should not be zero\n"
            <<"Aborting!!!!!\n";
    else
    {
        d =b*b-4*a*c;      //beginning of else block
        if (d > 0)
        {
            root1 = (-b + sqrt(d))/(2*a);
            root2 = (-b - sqrt(d))/(2*a);
            cout<<"Roots are REAL and UNEQUAL\n";
            cout<<"Root1 = "<<root1<<"\tRoot2 = "<<root2;
        }
        else if (d == 0)
        {
            root1 = -b/(2*a);
            cout<<"Roots are REAL and EQUAL\n";
            cout<<"Root1 =" <<root1;
        }
        else
            cout<<"Roots are COMPLEX and IMAGINARY";
    }// end of else block of outer if
    return 0;
}
```

ഔട്ട്പുട്ട് 1:

```
Enter the three coefficients:   2    3    4
Roots are COMPLEX and IMAGINARY
```

ഔട്ട്പുട്ട് 2:

```
Enter the three coefficients:   3    5    1
Roots are REAL and UNEQUAL
Root1 =  -0.232408   Root2 = -1.434259
```

പ്രോഗ്രാം 7.20 ഒരു ഫിബിനോസി ശ്രേണിയിലെ N പദങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനുള്ളതാണ്. ശ്രേണി തുടങ്ങുന്നത് 0, 1 എന്ന പദങ്ങളിൽ നിന്നാണ് അടുത്ത പദം വരുന്നത് തൊട്ടു മുമ്പുള്ള രണ്ടു പദങ്ങളുടെ തുകയായാണ്. ശ്രേണി ഇപ്രകാരമാണ് 0, 1, 1, 2, 3, 5, 8, 13, ...

പ്രോഗ്രാം 7.20: ഫിബിനോസി ശ്രേണിയിലെ N പദങ്ങൾ പ്രദർശിപ്പിക്കൽ

```
#include <iostream>
using namespace std;
int main()
{
    int first=0, second=1, third, n;
    cout<<"\nEnter number of terms in the series: ";
    cin>>n;
    cout<<first<<"\t"<<second;
    for(int i=3; i<=n; ++i)
    {
        third = first + second;
        cout<<"\t"<<third;
        first = second;
        second = third;
    }
    return 0;
}
```

first, second എന്നീ വേരിയബിളുകളുടെ പ്രാരംഭ വിലകൾ യഥാക്രമം -1, +1 എന്ന് നൽകിയാൽ നമുക്ക് ശ്രേണിയിലെ ആദ്യത്തെ രണ്ട് പദങ്ങൾ പ്രദർശിപ്പിക്കാനുള്ള cout പ്രസ്താവന ഒഴിവാക്കാം.

ഔട്ട് പുട്ട്

```
Enter number of terms in the series: 10
0    1    1    2    3    5    8    13    21    34
```

പ്രോഗ്രാം 7.21 ഒരു സംഖ്യ സ്വീകരിക്കുകയും അത് പാലിൻഡ്രോം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുകയും ചെയ്യുന്നു. ഒരു സംഖ്യ പാലിൻഡ്രോം എന്ന് പറയണമെങ്കിൽ ആ സംഖ്യയും അതിന്റെ പ്രിതിബിംബവും തുല്യമായിരിക്കണം. പ്രതിബിംബം എന്നത് കൊണ്ട് അർത്ഥമാക്കിയത് ഒരു സംഖ്യ തിരിച്ചെഴുതിയാലും അതേ സംഖ്യ കിട്ടുന്നു എന്നതാണ്.
അതായത് 163 ന്റെ പ്രതിബിംബം 361 ആണ്. ഈ രണ്ടു സംഖ്യകളും തുല്യമല്ലാത്തതിനാൽ 163 എന്ന സംഖ്യ പാലിൻഡ്രോം അല്ല. എന്നാൽ 232 എന്നത് ഒരു പാലിഡ്രോം സംഖ്യയാണ്.

പ്രോഗ്രാം 7.21: തന്നിരിക്കുന്ന സംഖ്യ പാലിഡ്രോം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int num, copy, digit, rev=0;
    cout<<"Enter the number: ";
    cin>>num;
    copy=num;
    while(num != 0)
```

ലൂപ്പിന്റെ പ്രവർത്തനം പൂർത്തിയാകുമ്പോൾ num എന്ന വേരിയബിളിന്റെ വില പൂജ്യമാകുന്നു. അതുകൊണ്ടാണ് അതിന്റെ ആദ്യത്തെ വില മറ്റൊരു വേരിയബിളിലേക്ക് പകർത്തിയത്

```
    {
        digit = num % 10;
        rev = (rev * 10)+ digit;
        num = num/10;
    }
    cout<<"The reverse of the number is: "<<rev;
    if (rev == copy)
        cout<<"\nThe given number is a palindrome.";
    else
        cout<<"\nThe given number is not a palindrome.";
    return 0;
}
```

ഔട്ട്പുട്ട് 1:

```
Enter the number: 363
The reverse of the number is: 363
The given number is a palindrome.
```

ഔട്ട്പുട്ട് 2:

```
Enter the number: 257
The reverse of the number is: 752
The given number is not a palindrome.
```

പ്രോഗ്രാം 7.22: N പൂർണ സംഖ്യകൾ സ്വീകരിച്ച് അതിലെ ഏറ്റവും വലിയ സംഖ്യ പ്രിന്റ് ചെയ്യുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int num, big, count;
    cout<<"How many Numbers in the list? ";
    cin >> count;
    cout<<"\nEnter first number: ";
    cin >> num;
    big = num;
    for(int i=2; i<=count; i++)
    {
        cout<<"\nEnter next number: ";
        cin >> num;
        if(num > big) big = num;
    }
    cout<<"\nThe largest number is " << big;
    return 0;
}
```

ഔട്ട്പുട്ട്:

```
How many Numbers in the list? 5
Enter first number: 23
Enter next number: 12
Enter next number: -18
Enter next number: 35
Enter next number: 18
The largest number is 35
```

നമുക്ക് സംഗ്രഹിക്കാം

തീരുമാനങ്ങൾ എടുക്കുന്നതിനോ ആവർത്തന പ്രവർത്തനങ്ങൾ നടപ്പാക്കുന്നതിനോ ഉള്ള സൗകര്യങ്ങൾ ഉള്ള പ്രസ്താവനകളെ നിയന്ത്രണ പ്രസ്താവനകൾ എന്ന് അറിയപ്പെടുന്നു. നിയന്ത്രണ പ്രസ്താവനകൾ ഒരു കമ്പ്യൂട്ടർ പ്രോഗ്രാമിന്റെ നട്ടെല്ലാണ്. ഈ അധ്യായത്തിൽ വിവിധ തരം നിയന്ത്രണ പ്രസ്താവനകളായ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ (`if`, `if…else`, `if…else if`, `switch`), ആവർത്തന പ്രസ്താവനകൾ (`for`, `while`, `do…while`) ജമ്പ് പ്രസ്താവനകൾ (`goto`, `break`, `continue`, `exit`) എന്നിവ നാം പഠിച്ചു. ഈ നിയന്ത്രണ പ്രസ്താവനകൾ കാര്യക്ഷമമായ C++ പ്രോഗ്രാമുകൾ എഴുതുന്നതിന് നമ്മെ സഹായിക്കും.

പഠനനേട്ടങ്ങൾ

ഈ അധ്യായം പൂർത്തിയാകുമ്പോൾ പഠിതാവ്

- പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യുന്നതിന് C++ ലെ നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിക്കുന്നു.
- നിയന്ത്രണ പ്രസ്താവനകൾ ഒരു പ്രോഗ്രാമിൽ ഏതു സാഹചര്യത്തിലാണ് ഉപയോഗിക്കേണ്ടത് എന്ന് തിരിച്ചറിയുന്നു.
- സാഹചര്യത്തിന് അനുയോജ്യമായ ശരിയായ നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിക്കുന്നു.
- വിവിധ തരം നിയന്ത്രണ പ്രസ്താവനകളെ തരം തിരിക്കുന്നു.
- C++ ലെ വിവിധ തരം ജമ്പ് പ്രസ്താവനകളെ തിരിച്ചറിയുന്നു.
- നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിച്ച് C++ പ്രോഗ്രാം എഴുതുന്നു.

മാതൃകാ ചോദ്യങ്ങൾ

ഹ്രസ്വോത്തര ചോദ്യങ്ങൾ

1. switch പ്രസ്താവനയിൽ break- പ്രസ്താവനയുടെ പ്രാധാന്യം എഴുതുക. switch പ്രസ്താവനയിൽ break-ന്റെ അഭാവം എന്ത് ഫലം ഉളവാക്കും?
2. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലത്തിന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും?

```
for(i=1;i<=10;++i) ;
cout<<i+5;
```

3. താഴെ പറയുന്ന പ്രസ്താവനയെ **while, do while** ലൂപ്പുകൾ ഉപയോഗിച്ച് മാറ്റി എഴുതുക.

```
for (i=1; i<=10; i++) cout<<i;
```

4. താഴെ കൊടുത്തിരിക്കുന്ന ലൂപ്പ് എത്ര തവണ പ്രവർത്തിക്കും.

```
int  s=0, i=0;
while(i++<5)
      s+=i;
```

5. exit( ) ഫങ്ഷൻ അടങ്ങിയിരിക്കുന്ന ഹെഡർ ഫയലിന്റെ പേരെഴുതുക.
6. പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ഒരു ലേബലിലേക്ക് കൈമാറാൻ സാധിക്കുന്ന C++ ലെ പ്രസ്താവന ഏത്?
7. switch പ്രസ്താവനയിൽ default പ്രസ്താവനയുടെ ആവശ്യകത എന്ത്?

ലഘു ഉപന്യാസ ചോദ്യങ്ങൾ

1. താഴെ കൊടുത്തിരിക്കുന്ന രണ്ട് കോഡ് ശകലങ്ങൾ പരിഗണിക്കുക.

```
// version 1
cin>>mark;
if (mark > = 90)
cout<<" A+";
if (mark > = 80 && mark <90)
cout<<" A";
if (mark > = 70 && mark <80)
cout<<" B+";
if (mark > = 60 && mark <70)
cout<<" B";
```

```
//version 2
cin>>mark;
if (mark>=90)
cout<<" A+";
else if (mark>=80 && mark <90)
cout<<" A";
else if (mark>=70 && mark <80)
cout<<" B+";
else if (mark>=60 && mark <70)
cout<<" B";
```

വേർഷൻ 2 ന് വേർഷൻ 1 നെ അപേക്ഷിച്ചുള്ള മേന്മകൾ ചർച്ച ചെയ്യുക.

2. ഒരു for ലൂപ്പിന്റെ പ്രവർത്തനം അതിന്റെ വാക്യഘടന (Syntax) യോടുകൂടി ചുരുക്കി വിവരിക്കുക. നിങ്ങളുടെ ഉത്തരം സാധൂകരിക്കുന്നതിന് for ലൂപ്പിന്റെ ഒരു ഉദാഹരണം നൽകുക.

3. വിവിധ സാഹചര്യങ്ങളിൽ മൂന്നു ലൂപ്പുകളുടെ അനുയോജ്യത താരതമ്യം ചെയ്ത് ചർച്ച ചെയ്യുക.

4. താഴെ കൊടുത്തിരിക്കുന്ന `if.. else if` പ്രസ്താവന പരിഗണിക്കുക. `switch` പ്രസ്താവന കൊണ്ട് അത് മാറ്റി എഴുതുക.

```
if (a==1)
      cout << "One";
else if (a==0)
      cout << "Zero";
else
      cout << "Not a binary digit";
```

5. z = 3 ആണെങ്കിൽ താഴെ കൊടുത്തിരിക്കുന്ന `while` പ്രസ്താവനയിലെ തെറ്റ് എന്താണ്?

```
while(z>=0)
      sum+=z;
```

6. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലത്തിന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും?

```
for (outer=10; outer > 5; --outer)
{
      for (inner=1; inner<4; ++inner)
            cout<<outer <<"\t"<<inner <<endl;
}
```

7. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലത്തിന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും? വിശദീകരിക്കുക.

```
for (n = 1; n <= 10; ++n)
{
      for ( m=1; m <= 5 ; ++m)
            num = n*m;
      cout<<num <<endl;
}
```

8. ഒരു ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ പ്രാധാന്യം എഴുതുക. ഒരു ലൂപ്പിന്റെ വിവിധ ഭാഗങ്ങളെക്കുറിച്ച് ചുരുക്കി വിവരിക്കുക.

ഉപന്യസിക്കുക

1. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഔട്ട്പുട്ട് എന്ത്?

```
int val, res, n=1000;
cin>>val;
res = n+val > 1750 ? 400 : 200;
```

(a) ഇൻപുട്ട് 2000 ആണെങ്കിൽ

(b) ഇൻപുട്ട് 500 ആണെങ്കിൽ

2. താഴെ പറയുന്നവ ഉപയോഗിച്ച് ഒരു സംഖ്യയിലെ അക്കങ്ങളുടെ തുക കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.

(a) ആഗമന നിയന്ത്രണ ലൂപ്പ്

(b) ബഹിർഗമന നിയന്ത്രണ ലൂപ്പ്

3. 1000 ൽ താഴെയുള്ള ആംസ്ട്രോങ്ങ് സംഖ്യ പ്രിന്റ് ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. (ഒരു ആംസ്ട്രോങ്ങ് സംഖ്യ എന്നാൽ അതിലെ ഓരോ അക്കത്തിന്റെയും ക്യൂബുകളുടെ തുകയ്ക്ക് തുല്യമായിരിക്കും. ഉദാഹരണത്തിന് $153 = 1^3+5^3+3^3$)

4. C++ ലെ ലഭ്യമായ വിവിധ ജംപ് പ്രസ്താവനകൾ വിശദീകരിക്കുക.

5. നെസ്റ്റഡ് ലൂപ്പ് ഉപയോഗിച്ച് താഴെ കൊടുത്തിരിക്കുന്ന ഔട്ട്പുട്ട് സൃഷ്ടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.

```
A
A   B
A   B   C
A   B   C   D
A   B   C   D   E
```

6. `if...else` പ്രസ്താവനയിൽ `else` എന്ന വാക്ക് നിങ്ങൾ എഴുതാൻ മറന്നുപോയി എന്ന് വിചാരിക്കുക. നിങ്ങളുടെ പ്രോഗ്രാമിന്റെ ഔട്ട്പുട്ടിനെ ഇത് എങ്ങനെ ബാധിക്കുമെന്ന് ചർച്ച ചെയ്യുക.

8

പ്രധാന ആശയങ്ങൾ

- അറെയും ആവശ്യകതയും
  - അറെ പ്രഖ്യാപനം
  - അറെയുടെ മെമ്മറി നീക്കി വയ്ക്കൽ
  - അറെയും പ്രാരംഭ വില നൽകലും
  - അറെയിലെ അംഗങ്ങളെ ഉപയോഗിക്കൽ
- അറെയുടെ പ്രവർത്തനങ്ങൾ
  - കടന്നുപോകൽ
  - ക്രമപ്പെടുത്തൽ
    - സെലക്ഷൻ സോർട്ട്
    - ബബിൾ സോർട്ട്
  - തിരയൽ
    - രേഖീയ തിരയൽ
    - ബൈനറി തിരയൽ
- ദ്വിമാന അറെകൾ
  - ദ്വിമാന അറെ പ്രഖ്യാപനം
  - മെട്രിക്സായി ദ്വിമാന അറെകൾ
- ബഹുമുഖ അറെകൾ

Y6X6M2

അറെകൾ

പ്രോഗ്രാമുകളിൽ ഡാറ്റ സംഭരിക്കുന്നതിനായി നാം വേരിയബളുകൾ ഉപയോഗിക്കുന്നു. എന്നാൽ ഡാറ്റയുടെ എണ്ണം കൂടുതലാണെങ്കിൽ കൂടുതൽ വേരിയബളുകൾ ഉപയോഗിക്കേണ്ടതായി വരും. ഈ സാഹചര്യത്തിൽ ഡാറ്റ ഉപയോഗിക്കുന്ന രീതി വളരെ ബുദ്ധിമുട്ടുള്ളതായി അനുഭവപ്പെടും. ഇതു മറികടക്കാൻ ഈ അദ്ധ്യായത്തിൽ അറെ (Array) എന്ന പേരിലുള്ള C++ ൽ നിന്നും ഉരുത്തിരിഞ്ഞ ഡാറ്റ ഇനം പരിചയപ്പെടുത്തുന്നു. അറെ എന്നത് കേവലമൊരു ഡാറ്റ ഇനത്തിന്റെ നാമം മാത്രമല്ല, മറിച്ച് ഇത് വളരെ കൂടുതൽ ഡാറ്റ എളുപ്പത്തിൽ കൈകാര്യം ചെയ്യുന്നതിന് വേണ്ടി അടിസ്ഥാനപരമായ ഡാറ്റ ഇനങ്ങളിൽ നിന്നും നിർമിച്ചെടുത്ത മറ്റൊരു തരം ഡാറ്റ ഇനമാണ്. അറെയുടെ പ്രഖ്യാപനം പ്രാഥമിക വിലയിരുത്തൽ (Initialization), കടന്നുപോകൽ (Traversal), ക്രമപ്പെടുത്തൽ (Sorting), തിരയൽ (Searching) പോലുള്ള പ്രവർത്തനങ്ങളെപ്പറ്റി നമുക്ക് ചർച്ച ചെയ്യാം.

8.1 അറെയും അവയുടെ ആവശ്യകതയും (Array and its need)

അറെ എന്നാൽ തുടർച്ചയായ മെമ്മറി സ്ഥാനങ്ങളിൽ ശേഖരിച്ചു വച്ചിട്ടുള്ള ഒരേ ഇനത്തിലുള്ള ഡാറ്റകളുടെ സമൂഹമാണ്. ഒരു പേരിൽ ഒരേ ഇനത്തിലുള്ള ഒരു കൂട്ടം വിലകൾ ശേഖരിക്കുന്നതിനായി അറെകൾ ഉപയോഗിക്കുന്നു. ഒരു അറെയിലെ ഓരോ അംഗങ്ങളേയും അതിന്റേതായ സൂചിക വ്യക്തമാക്കിക്കൊണ്ട് ഉപയോഗിക്കുവാൻ സാധിക്കും.

എന്തുകൊണ്ടാണ് പ്രോഗ്രാമുകളിൽ അറെ ആവശ്യമായിവരുന്നത്. ഒരു ഉദാഹരണത്തിന്റെ സഹായത്തോടെ ഇത് നമുക്ക് പരിശോധിക്കാം. ഒരു ക്ലാസിലെ 20 വിദ്യാർത്ഥികളുടെ മാർക്കുകളുടെ ശരാശരിയെ കണ്ടെത്തണം എന്ന് കരുതുക. ഈ സാഹചര്യത്തിൽ സാധാരണ വേരിയബിളുകൾ ഉപയോഗിച്ചാൽ 20 വിദ്യാർത്ഥികളുടെ മാർക്കുകൾ ശേഖരിക്കുവാൻ 20 വേരിയബിളുകൾ ആവശ്യമായി വരും.

```
int  a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t;
float avg;
cin>>a>>b>>c>>d>>e>>f>>g>>h>>i>>j>>k>>l>>m>>n>>o>>p>>q>>r>>s>>t;
avg  =  (a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t)/20.0;
```

ഒരു പരിധിവരെ മുകളിൽ കൊടുത്തിരിക്കുന്ന കോഡ് ഉപയോഗിച്ച് 20 കുട്ടികളുടെ മാർക്കുകളുടെ ശരാശരി കണ്ടുപിടിക്കുവാൻ കഴിയും. എന്നാൽ 1000 കുട്ടികളുടെ ശരാശരി മാർക്ക് കണ്ടുപിടിക്കേണ്ട ഒരു സാഹചര്യം ഉണ്ടായാൽ ഈ രീതിയിലുള്ള പ്രവർത്തനം സാധ്യമല്ല. അതായത് ഒരു പ്രോഗ്രാമിൽ 1000 വേരിയബിളുകൾ ഉപയോഗിക്കുന്നതും അവ ഉപയോഗിച്ച് പ്രോഗ്രാം ചെയ്യുന്നതും എളുപ്പമുള്ള കാര്യമല്ല, മാത്രമല്ല ഇങ്ങനെ നിർമിക്കുന്ന പ്രോഗ്രാം വളരെ സങ്കീർണവും മനസ്സിലാക്കുന്നതിന് ബുദ്ധിമുട്ടുള്ളതും ആയിരിക്കും. ഇത്തരം സാഹചര്യങ്ങളിൽ അറെ എന്ന ആശയം നമുക്ക് ഉപകരിക്കും. അറെയിലെ ഓരോ അംഗങ്ങൾക്കും മെമ്മറി സ്ഥാനങ്ങൾ അനുവദിക്കേണ്ടതുണ്ട്. മെമ്മറി നീക്കിവെയ്ക്കുന്നതിന് പ്രഖ്യാപന പ്രസ്താവനകൾ ആവശ്യമാണെന്നും നമുക്കറിയാം. എങ്ങനെയാണ് അറെകൾ പ്രഖ്യാപനം നടത്തി അവ ഉപയോഗിക്കുന്നത് എന്ന് നമുക്ക് നോക്കാം.

8.1.1 അറെകളുടെ പ്രഖ്യാപനം (Array Declaration)

സാധാരണ വേരിയബിളിനെ പോലെ അറെ ഉപയോഗിക്കുന്നതിന് മുമ്പായി പ്രഖ്യാപനം നടത്തേണ്ടതുണ്ട്. C++ൽ അറെ പ്രഖ്യാപനം ചെയ്യുന്നതിനുള്ള വാക്യഘടന താഴെ പറഞ്ഞിരിക്കുന്നു.

```
datatype  array_name[size];
```

വാക്യഘടനയിൽ datatype എന്നത് അറെയിലെ അംഗങ്ങളുടെ ഡേറ്റയുടെ ഇനമാണ് സൂചിപ്പിക്കുന്നത്. array_name എന്നത് അറെയുടെ പേരും size എന്നത് അറെയിലെ ആകെ അംഗങ്ങളുടെ എണ്ണം വ്യക്തമാക്കുന്ന ഒരു പോസിറ്റീവ് സംഖ്യയും ആകുന്നു. താഴെ പറയുന്നത് ഒരു അറെ നിർമാണത്തിന്റെ ഉദാഹരണമാണ്.

```
int  num[10];
```

മുകളിലുള്ള പ്രസ്താവന num എന്ന് വിളിക്കുന്ന 10 പൂർണ്ണസംഖ്യകൾ സൂക്ഷിക്കാവുന്ന ഒരു അറെയെ നിർമിക്കുന്നു. ചിത്രം 8.1 കാണിച്ചിരിക്കുന്നതു പോലെ അറെയിലെ അംഗങ്ങൾ മെമ്മറിയിൽ തുടർച്ചയായി സൂക്ഷിക്കുന്നു.

	num[0]	num[1]	num[2]	num[3]	num[4]	num[5]	num[6]	num[7]	num[8]	num[9]
Index→	0	1	2	3	4	5	6	7	8	9

ചിത്രം 8.1 ഒരു അറെയിലെ അംഗങ്ങളുടെ ക്രമീകരണം

അറെയിലെ അംഗങ്ങൾ ക്രമാനുഗതമായി സൂക്ഷിക്കുന്നതുകൊണ്ട്, ഏത് അംഗത്തേയും അറെയുടെ പേരും അംഗത്തിന്റെ സ്ഥാനവും നൽകി ഉപയോഗിക്കുവാൻ കഴിയും. ഓരോ അംഗത്തെയും സൂചിപ്പിക്കുന്ന സ്ഥാനത്തിന് സൂചിക (index or subscript) എന്നു പറയുന്നു.

C++ൽ അറെയുടെ സൂചിക പൂജ്യത്തിൽ ആരംഭിക്കുന്നു. `int num[10]` എന്ന് ഒരു അറെ നിർമ്മിച്ചാൽ അതിൽ സാധ്യമായ സൂചിക വിലകൾ 0 മുതൽ 9 വരെയാകും. ഈ അറെയിലെ ഒന്നാമത്തെ അംഗം `num [0]` ഉം അവസാനത്തെ അംഗം `num [9]` ഉം ആകുന്നു. `num [0]` എന്നത് 'നം ഓഫ് സീറോ' എന്ന് വായിക്കുന്നു. ആയിരം വിദ്യാർത്ഥികളുടെ മാർക്കുകൾ സംഭരിക്കുന്ന പ്രശ്നം താഴെപ്പറയുന്ന പ്രസ്താവന ഉപയോഗിച്ച് പരിഹരിക്കാനാകും.

```
int score[1000];
```

`score` എന്നു പേരുള്ള അറെയിൽ 1000 വിദ്യാർഥികളുടെ മാർക്കുകൾ സംഭരിക്കാം. ആദ്യ വിദ്യാർത്ഥിയുടെ മാർക്ക് `score[0]` ലും അവസാനത്തെ വിദ്യാർത്ഥിയുടെ മാർക്ക് `score[999]` ലും സംഭരിക്കും.

8.1.2 അറെയുടെ മെമ്മറി നീക്കിവെയ്ക്കൽ (Memory Allocation for Arrays)

ഒരു അറെയിൽ അംഗങ്ങളെ സംഭരിക്കുന്നതിന് ആവശ്യമായ മെമ്മറിയുടെ അളവ് അതിന്റെ ഇനവും അംഗങ്ങളുടെ എണ്ണവുമായി ബന്ധപ്പെട്ടിരിക്കുന്നു. ചിത്രം 8.2ൽ `num` എന്ന ഒരു അറെയുടെ മെമ്മറി നീക്കിവെയ്ക്കൽ കാണിച്ചിരിക്കുന്നു, ഇതിൽ ആദ്യ അംഗത്തിന്റെ വിലാസമായി 1000 എന്ന് കാണിച്ചിരിക്കുന്നു. `num` ഒരു പൂർണ്ണസംഖ്യകളുടെ അറെ ആയ തിനാൽ, ഓരോ അംഗത്തിന്റെയും വ്യാപ്തി 4 ബൈറ്റുകൾ ആണ് (16 ബിറ്റ് പ്രതിനിധീകരിക്കുന്ന ഒരു സിസ്റ്റത്തിൽ). താഴെക്കൊടുത്തിരിക്കുന്ന ചിത്രം 8.2ൽ `num[0]`ന്റെ വിലാസം `1000`, `num[1]`ന്റെ വിലാസം `1004`, അവസാന അംഗമായ `num[4]` വിലാസം വിലാസം `1016` എന്നിങ്ങനെ ആയിരിക്കും.

ചിത്രം 8.2 ഒരു പൂർണ്ണ സംഖ്യ അറെയുടെ മെമ്മറി അലോക്കേഷൻ

ഒരു ഏകമാന അറെക്ക് (single dimensional array) ആവശ്യമായ മെമ്മറിയുടെ അളവ് താഴെ പറയുന്ന സൂത്രവാക്യം ഉപയോഗിച്ച് കണ്ടുപിടിക്കാം.

ആകെ ബൈറ്റുകൾ = size_of (അറെയുടെ ഇനം) × അറെയിലെ അംഗങ്ങളുടെ എണ്ണം

ഉദാഹരണത്തിന്, `int num [10];` `num` അറെക്കായി നീക്കിവെച്ചിട്ടുള്ള ആകെ ബൈറ്റുകൾ $4 \times 10 = 40$ ബൈറ്റുകൾ ആയിരിക്കും.

8.1.3 അറെയുടെ പ്രാരംഭ വില നൽകൽ (Array Initialization)

സാധാരണ വേരിയബിൾ പോലെ തന്നെ അറെയുടെ പ്രഖ്യാപന പ്രസ്താവനകളോടൊപ്പം അവയുടെ പ്രാരംഭ വിലകൾ നൽകുവാൻ കഴിയും. താഴെപ്പറയുന്ന ഉദാഹരണങ്ങളിൽ കാണിച്ചിരിക്കുന്നതുപോലെ അറെയിലെ അംഗങ്ങളെ ബ്രാക്കറ്റിനുള്ളിൽ എഴുതണം.

```
int score[5] = {98, 87, 92, 79, 85};
char code[6] = {'s', 'a', 'm', 'p', 'l', 'e'};
float wgpa[7] = {9.60, 6.43, 8.50, 8.65, 5.89, 7.56, 8.22};
```

അറെയിലെ അംഗങ്ങളെ അവ എഴുതപ്പെട്ട ക്രമത്തിൽ സൂക്ഷിക്കുന്നു. ഒന്നാമത്തെ അംഗം സൂചിക 0ലും, രണ്ടാമത്തെ അംഗം സൂചിക 1ലും പ്രാരംഭ വിലകളായി സൂക്ഷിക്കുന്നു. ആദ്യത്തെ ഉദാഹരണത്തിൽ, `score[0]`ലേക്ക് 98, `score[1]` ലേക്ക് 87, `score[2]` ലേക്ക് 92, `score[3]` ലേക്ക് 79, `score[4]` ലേക്ക് 85 ഉം പ്രാരംഭ വിലകളായി സൂക്ഷിക്കുന്നു. ഒരു അറെയ്ക്ക് അനുവദിക്കപ്പെട്ട അംഗങ്ങളുടെ എണ്ണത്തെക്കാൾ പ്രാരംഭ മൂല്യങ്ങളുടെ എണ്ണം കുറവാണെങ്കിൽ, ആദ്യ സ്ഥാനങ്ങളിൽ അംഗങ്ങൾ സംഭരിക്കും, ശേഷിക്കുന്ന സ്ഥാനങ്ങൾ സംഖ്യാ ഡാറ്റകളുടെ കാര്യത്തിൽ പൂജ്യവും അക്ഷരഡാറ്റകളുടെ കാര്യത്തിൽ ' ' (സ്പെയിസും) സംഭരിക്കും. ഒരു അറെയിലെ അംഗങ്ങളുടെ പ്രാരംഭ വിലകൾ നൽകുമ്പോൾ അംഗങ്ങളുടെ എണ്ണം ഒഴിവാക്കാവുന്നതാണ്. ഉദാഹരണത്തിന്, താഴെ പറയുന്ന പ്രാരംഭ വില നൽകൽ പ്രസ്താവന അഞ്ച് അംഗങ്ങളുള്ള ഒരു അറെ നിർമ്മിക്കുന്നു.

```
int num[] = {16, 12, 10, 14, 11};
```

8.1.4 അറെയിലെ അംഗങ്ങളെ ഉപയോഗിക്കൽ (Accessing elements of arrays)

ഒരു അറെയിലെ അംഗങ്ങളെ പ്രോഗ്രാമിൽ എവിടെയും ഉപയോഗിക്കാം. ഒരു സമയം ഒരു അംഗത്തിനെ മാത്രമേ ഉപയോഗിക്കാൻ കഴിയു. ഓരോ അംഗത്തേയും അറെയുടെ പേരും അവയുടെ സൂചികയും നല്കി ഉപയോഗിക്കുന്നു. `score` എന്ന അറെയിലെ അംഗങ്ങളെ ഉപയോഗിക്കുന്ന ചില ഉദാഹരണങ്ങൾ താഴെ കൊടുത്തിരിക്കുന്നു.

```
score[0] = 95;
score[1] = score[0] - 11;
cin >> score[2];
score[3] = 79;
cout << score[2];
sum = score[0] + score[1] + score[2] + score[3] + score[4];
```

ബ്രാക്കറ്റിനുള്ളിലെ സൂചിക ഒരു വേരിയബിളോ, ഒരു പൂർണ്ണസംഖ്യയോ, പൂർണ്ണസംഖ്യ നിർദ്ധാരണം ചെയ്യുന്ന ഒരു പ്രസ്താവനയോ ആകാം. ഓരോ സന്ദർഭത്തിലും പ്രസ്താവനയുടെ മൂല്യം അറെയുടെ സൂചികയുടെ സാധുവായ പരിധിക്കുള്ളിൽ ആയിരിക്കണം. ഈ രീതിയിൽ വേരിയബിളോ പ്രസ്താവനയോ ഉപയോഗിക്കുന്നതിന്റെ ഗുണം, അറെയിലുള്ള അംഗങ്ങളെ ഉപയോഗിക്കുന്നതിന് വേണ്ടി ലൂപ്പിന്റെ നിയന്ത്രണ വേരിയബിള് ഉപയോഗിക്കാം എന്നുള്ളതാണ്. ഇത് പ്രസ്താവനകളെ താഴെപ്പറയുന്ന രീതിയിൽ അനുചിതമായി ഉപയോഗിക്കുന്നതിൽ നിന്നും നമ്മെ പിന്തിരിപ്പിക്കുന്നു.

```
sum = score[0] + score[1] + score[2] + score[3] + score[4];
```

മുകളിലുള്ള പ്രസ്താവനയിലെ സൂചികയുടെ മൂല്യങ്ങൾക്കു പകരം ലൂപ്പിന്റെ നിയന്ത്രണ വേരിയബിൾ ഉപയോഗിച്ചുകൊണ്ട് അറെയിലെ അംഗങ്ങളെ ഉപയോഗിക്കാം. താഴെപ്പറയുന്ന പ്രസ്താവനകൾ ഈ ആശയം വിശദമാക്കുന്നു.

```
sum = 0;
for (i=0; i<5; i++)
sum = sum + score[i];
```

താഴെ കാണിച്ചിരിക്കുന്നത് പോലെ ഒരു ഇൻപുട്ട് പ്രസ്താവന ഉപയോഗിച്ചുകൊണ്ടും അറെയിലെ അംഗത്തിന് മൂല്യം നൽകാം.

```
    for(int i=0; i<5; i++)
        cin>>score[i];
```

ഈ ലൂപ്പ് പ്രവർത്തിച്ചു കഴിയുമ്പോൾ ആദ്യം സ്വീകരിക്കുന്ന വില അറെയുടെ ഒന്നാമത്തെ അംഗമായ `score [0]` ലും, രണ്ടാമത്തെ വില `score [1]` ലും, അവസാന വില `score[4]` ലും സൂക്ഷിക്കുന്നു.

പ്രോഗ്രാം 8.1 ഒരു അറെയിൽ എങ്ങനെ അഞ്ച് വിലകൾ സ്വീകരിക്കാമെന്നും അവയെ വിപരീത ക്രമത്തിൽ പ്രദർശിപ്പിക്കാമെന്നും കാണിക്കുന്നു. ഈ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തിയിട്ടുള്ള രണ്ട് ലൂപ്പുകളിൽ ആദ്യത്തേത് അറെയുടെ അംഗങ്ങളുടെ വിലകൾ സ്വീകരിക്കുന്നു. അഞ്ച് വിലകൾ സ്വീകരിച്ച് കഴിഞ്ഞാൽ രണ്ടാമത്തെ ലൂപ്പ് സംഭരിച്ച വിലകളെ അവസാനം മുതൽ ആദ്യം വരെ പ്രദർശിപ്പിക്കുന്നു.

പ്രോഗ്രാം 8.1: 5 കുട്ടികളുടെ സ്കോറുകൾ ഇൻപുട്ട് ചെയ്ത്, അവയെ നേർവിപരീത ക്രമത്തിൽ പ്രദർശിപ്പിക്കുക.

```
#include <iostream>
using namespace std;
int main()
{
  int i, score[5];
  for(i=0; i<5; i++)  // Reads the scores
  {
    cout<<"Enter a score: ";
    cin>>score[i];
  }
  for(i=4; i>=0; i--)  // Prints the scores
    cout<<"score[" << i << "] is " << score[i]<<endl;
  return 0;
}
```

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
Enter a score: 55
Enter a score: 80
Enter a score: 78
Enter a score: 75
Enter a score: 92
score[4] is 92
score[3] is 75
```

```
score[2] is 78
score[1] is 80
score[0] is 55
```

നമുക്ക് ചെയ്യാം

1. താഴെ പറയുന്നവ സംഭരിക്കുന്നതിനുള്ള അറേ പ്രഖ്യാപന പ്രസ്താവനകൾ എഴുതുക
   i. 100 വിദ്യാർത്ഥികളുടെ മാർക്ക്
   ii. ഇംഗ്ലീഷ് അക്ഷരമാല
   iii. 10 വർഷങ്ങളുടെ പട്ടിക
   iv. 30 ദശാംശ സംഖ്യകളുടെ പട്ടിക
2. താഴെ പറയുന്ന അറേയിൽ പ്രാരംഭ വിലകൾ നല്കുന്നതിനുള്ള പ്രസ്താവനകൾ എഴുതുക
   i. 10 സ്കോറുകളുടെ പട്ടിക 89, 75, 82, 93, 78, 95, 81, 88, 77, 82
   ii. അഞ്ച് അളവുകളുടെ പട്ടിക: 10.62, 13.98, 18.45, 12.68, 14.76 എന്നിവ
   iii. 100 പലിശ നിരക്കുകളുടെ പട്ടിക, ആദ്യ ആറ് പലിശ നിരക്കുകൾ 6.29, 6.95, 7.25, 7.35, 7.40, 7.42.
   iv. മൂല്യം 0 ഉപയോഗിച്ച് 10 മാർക്കിനുള്ള ഒരു അറെ.
   v. VIBGYOR അക്ഷരങ്ങളുള്ള ഒരു അറെ.
   vi. ഓരോ മാസത്തിലുമുള്ള ദിവസങ്ങളുള്ള ഒരു അറെ.
3. `int ar[50];` എന്ന അറെയിലേക്ക് വിലകൾ ഇൻപുട്ട് ചെയ്യുന്നതിനുള്ള C++ കോഡ് ശകലങ്ങൾ എഴുതുക.
4. `float val [100];` `val` അറെയുടെ ഇരട്ട സ്ഥാനങ്ങളിലുള്ള അംഗങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിന് C++ കോഡ് ശകലം എഴുതുക:

8.2 അറെയുടെ പ്രവർത്തനങ്ങൾ (Array Operations)

കടന്നുപോകൽ (Traversal), ക്രമപ്പെടുത്തൽ (Sorting), തിരയൽ (Searching), ഇടയിൽ ചേർക്കൽ (Insertion), നീക്കം ചെയ്യൽ (Deletion), ലയിപ്പിക്കൽ (Merging) തുടങ്ങിയവ അറെയുടെ പ്രവർത്തനങ്ങളിൽ ഉൾപ്പെടുന്നു. ഈ പ്രവർത്തനങ്ങൾ നടത്താൻ വ്യത്യസ്ത യുക്തികൾ പ്രയോഗിക്കുന്നു. അവയിൽ ചിലത് നമുക്ക് ചർച്ചചെയ്യാം.

8.2.1 കടന്നുപോകൽ (Traversal)

കുറഞ്ഞത് ഒരിക്കലെങ്കിലും അറെയിലെ ഓരോ അംഗത്തേയും ഉപയോഗിക്കുക എന്നതാണ് കടന്നുപോകൽ എന്നതുകൊണ്ട് ഉദ്ദേശിക്കുന്നത്. ഇടയിൽ ചേർക്കൽ, നീക്കം ചെയ്യൽ തുടങ്ങിയ പ്രവർത്തനങ്ങളുടെ കൃത്യത പരിശോധിക്കുവാൻ കടന്നുപോകൽ പ്രവർത്തനം നമുക്ക് ഉപയോഗിക്കാം. അറെയിലെ എല്ലാ അംഗങ്ങളേയും പ്രദർശിപ്പിക്കുന്നത് കടന്നുപോകലിന് ഒരു ഉദാഹരണമാണ്. ഏതെങ്കിലുമൊരു പ്രവർത്തനം അറെയിലെ എല്ലാ അംഗങ്ങളിലും നടക്കുന്നു എങ്കിൽ അതിനെ കടന്നുപോകൽ എന്നു പറയുന്നു .

ഒരു പ്രോഗ്രാമിൽ എങ്ങനെയാണ് കടന്നുപോകൽ നടത്തുന്നത് എന്നത് താഴെ കൊടുത്തിരിക്കുന്നു

പ്രോഗ്രാം 8.2: അറെയിലെ കടന്നുപോകൽ

```
#include <iostream>
using namespace std;
int main()
{
int a[10], i;
cout<<"Enter the elements of the array :";
for(i=0; i<10; i++)
     cin >> a[i];                  കടന്നുപോകൽ
for(i=0; i<10; i++)
     a[i] = a[i] + 1;              കടന്നുപോകൽ
cout<<"\nEntered elements of the array are...\n";
for(i=0; i<10; i++)
    cout<< a[i]<< "\t";            കടന്നുപോകൽ
return 0;
}
```

8.2.2 ക്രമപ്പെടുത്തൽ (Sorting)

ചില യുക്തിപരമായ ക്രമത്തിൽ അറെയിലെ അംഗങ്ങൾ ക്രമീകരിക്കുന്ന പ്രവർത്തനമാണ് ക്രമപ്പെടുത്തൽ. വാക്കുകളുടെ കാര്യത്തിൽ നിഘണ്ടു ക്രമവും സംഖ്യകളുടെ കാര്യത്തിൽ അവയുടെ മൂല്യങ്ങളുടെ ആരോഹണ ക്രമമോ അവരോഹണ ക്രമമോ ആകാം യുക്തി പരമായ ക്രമം. ഈ പ്രവർത്തനം ഫലപ്രദമായി ചെയ്യാൻ പല അൽഗോരിതങ്ങളുമുണ്ട് ഇവയിൽ സെലക്ഷൻ സോർട്ട്, ബബിൾ സോർട്ട് എന്നീ അൽഗോരിതങ്ങൾ നമുക്ക് ഇവിടെ ചർച്ചചെയ്യാം.

a. സെലക്ഷൻ സോർട്ട് (Selection sort)

ഏറ്റവും ലളിതമായ ക്രമപ്പെടുത്തൽ രീതികളിലൊന്നാണ് സെലക്ഷൻ സോർട്ട്. ആരോഹണക്രമത്തിൽ ഒരു അറെ ക്രമീകരിക്കുന്നതിനായി അറെയിലെ ഏറ്റവും കുറഞ്ഞ മൂല്യമുള്ള അംഗത്തെ കണ്ടെത്തി ആദ്യ സ്ഥാനത്തേക്ക് മാറ്റിക്കൊണ്ടാണ് സെലക്ഷൻ സോർട്ട് ആരംഭിക്കുന്നത്. അതേസമയം ആദ്യ സ്ഥാനത്തെ അംഗത്തെ ചെറിയ മൂല്യമുള്ള അംഗത്തിന്റെ സ്ഥാനത്തേക്കും മാറ്റുന്നു. അതിനുശേഷം രണ്ടാമത്തെ കുറഞ്ഞ മൂല്യമുള്ള അംഗത്തെ കണ്ടെത്തി രണ്ടാം സ്ഥാനത്തുള്ള അംഗവുമായി പരസ്പരം മാറ്റം ചെയ്യുന്നു. അങ്ങനെ എല്ലാ അംഗങ്ങളെയും ക്രമീകരിക്കപ്പെടുന്നതുവരെ ഈ പ്രവർത്തനം തുടരുന്നു. ഒരു അറെയിലെ കുറഞ്ഞ മൂല്യമുള്ള അംഗത്തെ കണ്ടെത്തി അനുയോജ്യമായ സ്ഥാനത്തെ അംഗവുമായി മാറ്റം ചെയ്യുന്ന പ്രക്രിയയെ സ്ഥാനമാറ്റം എന്ന് പറയുന്നു. N അംഗങ്ങളുള്ള ഒരു സെലക്ഷൻ സോർട്ടിൽ N-1 സ്ഥാനമാറ്റങ്ങൾ ഉണ്ടായിരിക്കും. ഉദാഹരണത്തിന് താഴെ നൽകിയിരിക്കുന്ന സംഖ്യകളുടെ പട്ടിക നോക്കുക.

പ്രാരംഭ പട്ടിക

32	23	10	2	30

സ്ഥാനമാറ്റം 1

32	23	10	2	30

സ്ഥാനമാറ്റം 1: പട്ടികയിൽ നിന്നും ഏറ്റവും ചെറിയ അംഗമായ 2 തിരഞ്ഞെടുക്കുന്നു, അത് ആദ്യത്തെ അംഗവുമായി സ്ഥാനമാറ്റം ചെയ്യപ്പെടുന്നു.

സ്ഥാനമാറ്റം 1 നുശേഷം

2	23	10	32	30

സ്ഥാനമാറ്റം 2

2	23	10	32	30

സ്ഥാനമാറ്റം 2: പട്ടികയിൽ നിന്നും 2 ഒഴികെയുള്ള തിൽ ചെറിയ അംഗമായ 10 തിരഞ്ഞെടുക്കുന്നു, അത് രണ്ടാമത്തെ അംഗവുമായി സ്ഥാനമാറ്റം ചെയ്യപ്പെടുന്നു.

സ്ഥാനമാറ്റം 2 നുശേഷം

2	10	23	32	30

സ്ഥാനമാറ്റം 3

2	10	23	32	30

സ്ഥാനമാറ്റം 3: പട്ടികയിൽ നിന്നും 2, 10 എന്നിവ ഒഴികെയുള്ളതിൽ ചെറിയ അംഗമായ 23 തിരഞ്ഞെടുക്കുന്നു, അത് മൂന്നാമത്തെ അംഗവുമായി സ്ഥാനമാറ്റം ചെയ്യപ്പെടുന്നു.

സ്ഥാനമാറ്റം 3 നുശേഷം

2	10	23	32	30

സ്ഥാനമാറ്റം 4

2	10	23	32	30

സ്ഥാനമാറ്റം 4: പട്ടികയിൽ നിന്നും 2, 10, 23 എന്നിവ ഒഴികെയുള്ളതിൽ ചെറിയ അംഗമായ 30 തിരഞ്ഞെടുക്കുന്നു, അത് നാലാമത്തെ അംഗവുമായി സ്ഥാനമാറ്റം ചെയ്യപ്പെടുന്നു.

സ്ഥാനമാറ്റം 4 നുശേഷം

2	10	23	30	32

ഓരോ തവണയും ഒരു സ്ഥാനമാറ്റം ഉദ്ദേശിച്ചിട്ടുണ്ടെങ്കിലും, ഏറ്റവും കുറഞ്ഞ മൂല്യം ശരിയായ സ്ഥലത്ത് ആണെങ്കിൽ സ്ഥാനമാറ്റം സംഭവിക്കുന്നില്ല. സ്ഥാനമാറ്റം മൂന്നിൽ ഇത് നിങ്ങൾക്ക് കാണുവാൻ സാധിക്കും

സെലക്ഷൻ സോർട്ടിന്റെ അൽഗോരിതം

1. ആരംഭിക്കുക
2. N ന്റെ വില സ്വീകരിക്കുക
3. I ← 0
4. ഘട്ടങ്ങൾ 5, 6 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
5. AR[I] ലേയ്ക്ക് ഡാറ്റ സ്വീകരിക്കുക
6. I ← I +1
7. I ← 0
8. ഘട്ടങ്ങൾ 9 മുതൽ 14 വരെ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
9. MIN ← AR[I], POS ← I
10. J ← I +1 മുതൽ N-1 ആകുന്നതുവരെ ഘട്ടം 11, 12 ആവർത്തിക്കുക
11. IF AR[J] < MIN ആണെങ്കിൽ MIN ← AR[J], POS ← J
12. J ←J +1
13. IF POS <> I ആണെങ്കിൽ AR[POS] ← AR[I], AR[I] ← MIN
14. I ← I +1
15. I ← 0
16. ഘട്ടങ്ങൾ 15, 16 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
17. AR[I] പ്രദർശിപ്പിക്കുക.
18. I ← I +1
19. അവസാനിപ്പിക്കുക

പ്രോഗ്രാം 8.3: ആരോഹണ ക്രമത്തിൽ അംഗങ്ങളെ ക്രമീകരിക്കുന്നതിനുള്ള സെലക്ഷൻ സോർട്ട്

```
#include <iostream>
using namespace std;
int main()
{   int AR[25], N, I, J, MIN, POS;
    cout<<"How many elements? ";
    cin>>N;
    cout<<"Enter the array elements: ";
    for(I=0; I<N; I++)
    cin>>AR[I];
    for(I=0; I < N-1; I++)
    {
        MIN=AR[I];
        POS=I;
```

```
        for(J = I+1; J < N; J++)
            if(AR[J]<MIN)
            {
                MIN=AR[J];
                POS=J;
            }
        if(POS != I)
        {
            AR[POS]=AR[I];
            AR[I]=MIN;
        }
    }
    cout<<"Sorted array is: ";
    for(I=0; I<N; I++)
        cout<<AR[I]<<"\t";
    return 0;
}
```

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
How many elements? 5
Enter the array elements: 12 3   6   1   8
Sorted array is: 1   3   6   8   12
```

b. ബബിൾ സോർട്ട് (Bubble sort)

ബബിൾ സോർട്ട് അൽഗോരിതം ക്രമീകരിക്കേണ്ട അറെയിലെ അടുത്തടുത്ത ഓരോ ജോഡി അംഗങ്ങളെ താരതമ്യം ചെയ്യുകയും അവ തെറ്റായ ക്രമത്തിലാണെങ്കിൽ പരസ്പരം സ്ഥാനമാറ്റം നടത്തുകയും ചെയ്യുന്നു. ഒരു സ്ഥാനമാറ്റവും ആവശ്യമില്ലാത്തതു വരെ ഈ പ്രക്രിയ ആവർത്തിക്കപ്പെടുന്നു, ഈ അവസ്ഥയിൽ അറെ ക്രമീകരിക്കപ്പെട്ടതായി കരുതാം. ഒരു ഉദാഹരണത്തിന്റെ സഹായത്തോടെ ഈ പ്രക്രിയ പരിശോധിക്കാം.

പ്രാരംഭ പട്ടിക

23	12	7	11	4

സ്ഥാനമാറ്റം 1

23	12	7	11	4

ആദ്യത്തെ ആദ്യ രണ്ട് അംഗങ്ങളായ 23, 12 എന്നിവ താരതമ്യം ചെയ്ത ശേഷം പരസ്പരം സ്ഥാനമാറ്റം ചെയ്യുന്നു.

12	(23)	(7)	11	4

പരിഷ്കരിച്ച പട്ടികയിലെ രണ്ടാമത്തെയും മൂന്നാമത്തെയും അംഗങ്ങളായ 23, 7 എന്നിവ താരതമ്യം ചെയ്ത ശേഷം പരസ്പരം സ്ഥാനമാറ്റം ചെയ്യുന്നു.

12	7	(23)	(11)	4

പരിഷ്കരിച്ച പട്ടികയിലെ മൂന്നാമത്തെയും നാലാമത്തെയും അംഗങ്ങളായ 23, 11 എന്നിവ താരതമ്യം ചെയ്ത ശേഷം പരസ്പരം സ്ഥാനമാറ്റം ചെയ്യുന്നു.

12	7	11	(23)	(4)

പരിഷ്കരിച്ച പട്ടികയിലെ നാലാമത്തെയും അഞ്ചാമത്തെയും അംഗങ്ങളായ 23, 4 എന്നിവ താരതമ്യം ചെയ്ത ശേഷം പരസ്പരം സ്ഥാനമാറ്റം ചെയ്യുന്നു.

12	7	11	4	23

ആദ്യത്തെ സ്ഥാനമാറ്റം കഴിയുമ്പോൾ അറെയിലെ ഏറ്റവും വലിയ അംഗമായ 23 അറെയുടെ അവസാന സ്ഥാനത്ത് എത്തുന്നു.

സ്ഥാനമാറ്റം 2

(12)	(7)	11	4	23

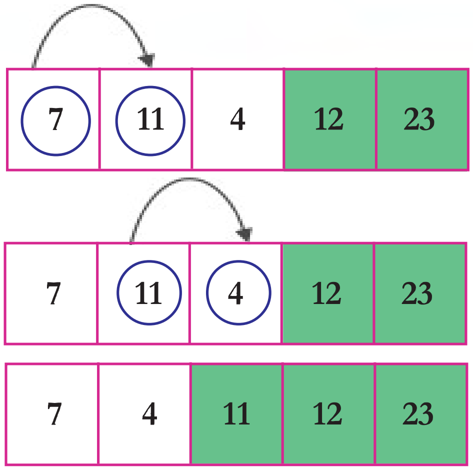
7	(12)	(11)	4	23

7	11	(12)	(4)	23

7	11	4	12	23

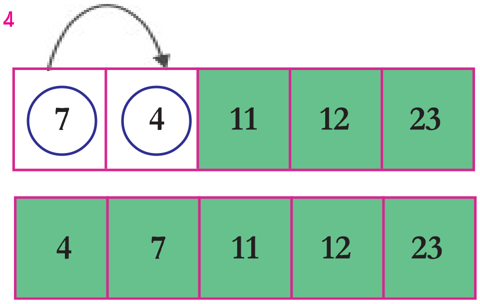
രണ്ടാമത്തെ സ്ഥാനമാറ്റത്തിൽ ആദ്യത്തെ നാല് അംഗങ്ങളെ മാത്രം പരിഗണിക്കുന്നു. ഒന്നാമത്തെ സ്ഥാനമാറ്റത്തിലെ അതേ പ്രക്രിയ തുടരുന്നു, അതിന്റെ ഫലമായി രണ്ടാമത്തെ സ്ഥാനമാറ്റത്തിൻറെ അവസാനം രണ്ടാമത്തെ വലിയ സംഖ്യയായ 12 അറെയുടെ നാലാം സ്ഥാനത്ത് എത്തുന്നു.

സ്ഥാനമാറ്റം 3



മൂന്നാമത്തെ സ്ഥാനമാറ്റത്തിൽ, 23 ഉം 12 ഉം ഒഴികെ പട്ടികയിലെ മൂന്ന് അംഗങ്ങളെ മാത്രമേ പരിഗണിക്കുന്നുള്ളൂ. മേൽപ്പറഞ്ഞ സ്ഥാനമാറ്റത്തിലെ അതേ പ്രക്രിയ തുടരുന്നു, അതിന്റെ ഫലമായി മൂന്നാമത്തെ സ്ഥാനമാറ്റത്തിന്റെ അവസാനം 11 എന്ന അംഗം അറെയുടെ മൂന്നാം സ്ഥാനത്ത് എത്തുന്നു.

സ്ഥാനമാറ്റം 4



നാലാമത്തെ സ്ഥാനമാറ്റത്തിൽ, 23, 12, 11 എന്നിവ ഒഴികെ പട്ടികയിലെ രണ്ട് അംഗങ്ങളെ മാത്രമേ പരിഗണിക്കുന്നുള്ളൂ. മേൽപ്പറഞ്ഞ സ്ഥാനമാറ്റത്തിലെ അതേ പ്രക്രിയ തുടരുന്നു, അതിന്റെ ഫലമായി നാലാമത്തെ സ്ഥാനമാറ്റത്തിന്റെ A അവസാനം 7 എന്ന അംഗം അറെയുടെ രണ്ടാം സ്ഥാനത്തെത്തുന്നു. 4 ഒന്നാം സ്ഥാനത്തും എത്തുന്നു

N അംഗങ്ങളുള്ള ഒരു ബബിൾ സോർട്ടിൽ N-1 സ്ഥാനമാറ്റങ്ങൾ ഉണ്ടായിരിക്കും. ഓരോ സ്ഥാനമാറ്റത്തിലും പരിഷ്കരിച്ച അറെയിലെ പരിഗണിക്കപ്പെടുന്ന അംഗങ്ങളുടെ എണ്ണം ഒന്നു വീതം കുറയും.

ബബിൾ സോർട്ടിന്റെ അൽഗോരിതം

1. ആരംഭിക്കുക
2. N ന്റെ വില സ്വീകരിക്കുക
3. I ← 0
4. ഘട്ടങ്ങൾ 5, 6 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
5. AR[I] ലേയ്ക്ക് ഡാറ്റ സ്വീകരിക്കുക
6. I ← I +1
7. I ← 1
8. ഘട്ടങ്ങൾ 9, 12 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
9. J ← 0 മുതൽ N-2 ആകുന്നതുവരെ ഘട്ടം 10, 11 എന്നിവ ആവർത്തിക്കുക
10. IF AR[J] > AR[J+1] ആണെങ്കിൽ TEMP←AR[J], AR[J] ←AR[J+1], AR[J+1] ← MIN

11. J ← J +1
12. I ← I +1
13. I ← 0
14. ഘട്ടങ്ങൾ 14, 15 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
15. AR[I] പ്രദർശിപ്പിക്കുക.
16. I ← I +1
17. അവസാനിപ്പിക്കുക

പ്രോഗ്രാം 8.4: ആരോഹണ ക്രമത്തിൽ അംഗങ്ങളെ ക്രമീകരിക്കുന്നതിനുള്ള ബബിൾ സോർട്ട്

```
#include <iostream>
using namespace std;
int main()
{   int AR[25],N;
    int I, J, TEMP;
    cout<<"How many elements? ";
    cin>>N;
    cout<<"Enter the array elements: ";
    for(I=0; I<N; I++)
        cin>>AR[I];
    for(I=1; I<N; I++)
        for(J=0; J<N-I; J++)
            if(AR[J] > AR[J+1])
            {
                TEMP = AR[J];
                AR[J] = AR[J+1];
                AR[J+1] = TEMP;
            }
    cout<<"Sorted array is: ";
    for(I=0; I<N; I++)
        cout<<AR[I]<<"\t";
}
```

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
How many elements? 5
Enter the array elements: 23  10  -3  7   11
Sorted array is: -3    7    10    11    23
```

8.2.3 തിരയൽ (Searching)

അറെയിലെ ഒരു അംഗത്തിന്റെ സ്ഥാനം കണ്ടുപിടിക്കുന്ന പ്രക്രിയയെ തിരയൽ (Searching) എന്നു പറയുന്നു. തന്നിരിക്കുന്ന ഡാറ്റയെ അറെയിൽ കണ്ടെത്തിയാൽ ആ തിരയൽ വിജയിച്ചു എന്നു പറയാം, അതായത് തന്നിരിക്കുന്ന ഡാറ്റ അറെയിലെ ഒരു അംഗമാണ്. അല്ലെങ്കിൽ തിരയൽ പരാജയപ്പെട്ടു. തിരയലിന് രേഖീയ തിരയൽ, ബൈനറി തിരയൽ

എന്നിങ്ങനെ രണ്ട് സമീപന രീതികൾ ഉണ്ട്. തിരയലിന് തിരഞ്ഞെടുക്കുന്ന അൽഗോരിതം അറെയിലെ അംഗങ്ങളുടെ ക്രമീകരണത്തെ ആശ്രയിച്ചിരിക്കുന്നു ക്രമരഹിതമായി അംഗങ്ങളെ ക്രമീകരിച്ചിരിക്കുന്ന അറെയിൽ രേഖീയ തിരയൽ രീതി ഉപയോഗിക്കുന്നു, എന്നാൽ അംഗങ്ങളെ ആരോഹണ ക്രമത്തിലോ അവരോഹണ ക്രമത്തിലോ വിന്യസിച്ചിരിക്കുന്ന അറെയിൽ ബൈനറി തിരയൽ രീതി ഉപയോഗിക്കുന്നതാണ് അഭികാമ്യം. ഈ തിരയൽ രീതികൾ താഴെ വിവരിക്കുന്നു.

a. രേഖീയ തിരയൽ (Linear search)

അറെയിൽ ഒരു പ്രത്യേക ഡാറ്റ കണ്ടെത്തുവാനുള്ള ഒരു രീതിയാണ് രേഖീയ തിരയൽ. രേഖീയ തിരയൽ പട്ടികയിലെ ഒന്നാമത്തെ അംഗത്തിൽ നിന്നും ആരംഭിച്ച് ക്രമമനുസരിച്ച്, ഓരോ അംഗത്തേയും പരിശോധിക്കുന്നു. ഈ പരിശോധന ഒന്നുകിൽ അംഗത്തെ കണ്ടെത്തുന്നതു വരെ അല്ലെങ്കിൽ പട്ടികയുടെ അവസാനം വരെ തുടരുന്നു.

50, 18, 48, 35, 45, 26, 12 എന്നീ അംഗങ്ങളുള്ള ഒരു അറെയിൽ നിന്ന് '45' എന്ന അംഗത്തെ തിരയണമെന്ന് കരുതുക. ആദ്യ അംഗമായ 50 ൽ നിന്നും രേഖീയ തിരയൽ ആരംഭിക്കുന്നു, അത് ഓരോ അംഗത്തെയും താരതമ്യം ചെയ്യുന്നു അഞ്ചാം സ്ഥാനത്ത് എത്തുമ്പോൾ ചിത്രം 8.3 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ 45 കണ്ടെത്തുന്നു.

ഇൻഡക്സ്	പട്ടിക	താരതമ്യം
0	50	50 == 45 : തെറ്റ്
1	18	18 == 45 : തെറ്റ്
2	48	48 == 45 : തെറ്റ്
3	35	35 ==45 : തെറ്റ്
4	45	45 ==45 : ശരി
5	26	
6	12	

ചിത്രം 8.3 രേഖീയ തിരയൽ

രേഖീയ തിരയലിന്റെ അൽഗോരിതം

1. ആരംഭിക്കുക
2. N ← ന്റെ വില സ്വീകരിക്കുക
3. I ← 0
4. ഘട്ടങ്ങൾ 5, 6 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
5. AR[I] ലേയ്ക്ക് ഡാറ്റ സ്വീകരിക്കുക
6. I ← I +1
7. ITEM ന്റെ വില സ്വീകരിക്കുക
8. I ← 0, LOC ← 0

9. ഘട്ടങ്ങൾ 10, 11 എന്നിവ I ← N-1 ആകുന്നതുവരെ ആവർത്തിക്കുക
10. IF AR[I] = ITEM ആണെങ്കിൽ LOC ←1, ഘട്ടം 12 ൽ പോകുക
11. I ← I +1
12. IF LOC = 1 ആണെങ്കിൽ തിരയൽ വിജയിച്ചു എന്ന് പ്രദർശിപ്പിക്കുക അല്ലെങ്കിൽ തിരയൽ പരാജയപ്പെട്ടു എന്നും പ്രദർശിപ്പിക്കുക.
13. അവസാനിപ്പിക്കുക

പ്രോഗ്രാം 8.5: അറെയിലുള്ള ഒരു അംഗത്തെ കണ്ടെത്തുന്നതിനുള്ള രേഖീയ തിരയൽ

```
#include <iostream>
using namespace std;
int main()
{
    int AR[25], N;
    int I, ITEM, LOC=-1;
    cout<<"How many elements? ";
    cin>>N;
    cout<<"Enter the array elements: ";
    for(I=0; I<n; I++)
        cin>>AR[I];
    cout<<"Enter the item you are searching for: ";
    cin>>ITEM;
    for(I=0; I<N; I++)
        if(AR[I] == ITEM)
        {
            LOC=I;
            break;
        }
    if(LOC!=-1)
        cout<<"The item is found at position "<<LOC+1;
    else
        cout<<"The item is not found in the array";
    return 0;
}
```

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
How many Elements?  7
Enter the array elements: 12 18 26 35 45 48 50
Enter the item you are searching for: 35
The item is found at position 4
```

```
How many Elements? 7
Enter the array elements: 12 18 26 35 45 48 50
Enter the item you are searching for: 25
The item is not found in the array
```

പട്ടികയുടെ മുൻപന്തിയിലാണ് തിരയേണ്ട അംഗമെങ്കിൽ ഏതാനും താരതമ്യങ്ങൾ കൊണ്ട് രേഖീയ തിരയൽ പ്രക്രിയ അവസാനിക്കും. പട്ടികയുടെ അവസാന ഭാഗത്താണ് തിരയേണ്ട അംഗമെങ്കിൽ തിരയൽ പ്രക്രിയയിലെ താരതമ്യങ്ങളുടെ എണ്ണം വളരെ വലുതായിരിക്കും, ഉദാഹരണത്തിന് 10000 അംഗങ്ങളുള്ള ഒരു പട്ടികയിൽ പരമാവധി താരതമ്യങ്ങളുടെ എണ്ണം 10000 ആയിരിക്കും.

b. ബൈനറി തിരയൽ (Binary Search)

നമ്മൾ വിശകലനം ചെയ്ത രേഖീയ തിരയൽ അൽഗോരിതം ലളിതവും കുറച്ച് അംഗങ്ങളുള്ള അറേകൾക്ക് യോജിച്ചതുമാണ്. എന്നാൽ അറെയിൽ നിരവധി അംഗങ്ങൾ ഉണ്ടെങ്കിൽ ധാരാളം തിരയലുകൾ ആവശ്യമായി വരും. ഈ സാഹചര്യത്തിൽ കൂടുതൽ കാര്യക്ഷമമായ ഒരു അൽഗോരിതം ഉപയോഗിക്കേണ്ടതുണ്ട്. അറെയിലെ അംഗങ്ങളെ ആരോഹണ ക്രമത്തിലോ അവരോഹണ ക്രമത്തിലോ ക്രമീകരിച്ചിട്ടുണ്ടെങ്കിൽ തിരച്ചിൽ സമയം കുറയ്ക്കാൻ കഴിയുന്ന കൂടുതൽ മെച്ചപ്പെട്ട ബൈനറി തിരയൽ അൽഗോരിതം നമുക്ക് ഉപയോഗിക്കാൻ കഴിയും

ഉദാഹരണത്തിന്, ഒരു നിഘണ്ടുവിൽ 'മോഡം' എന്ന പദത്തിന്റെ അർത്ഥം കണ്ടെത്തണമെന്ന് കരുതുക. തീർച്ചയായും നമ്മൾ ഒന്നാമത്തെ പേജിന്റെ ആദ്യ വാക്കു മുതൽ തിരച്ചിൽ ആരംഭിക്കുകയില്ല, മറിച്ച് നമ്മൾ തിരയുന്ന വാക്ക് ഏതാണ് ഉദ്ദേശം വെച്ച് നിഘണ്ടു തുറക്കുന്നു. നമുക്ക് തിരയേണ്ട വാക്ക് ആ പുറത്ത് ഇല്ലെങ്കിൽ പിന്നീടുള്ള തിരച്ചിൽ ഒരു പകുതി അവഗണിച്ച് മറ്റെ പകുതിയിൽ തിരയുന്നു. തിരച്ചിൽ നടത്തി ആവശ്യമായ പദം കണ്ടെത്തുവരെ അല്ലെങ്കിൽ നിഘണ്ടുവിൽ ഈ പദം ഇല്ല എന്ന് ഉറപ്പാക്കുക്കുന്നതു വരെ ഈ പ്രക്രിയ തുടർന്നുകൊണ്ടേയിരിക്കും. ഈ തിരച്ചിൽ രീതി ഒരു നിഘണ്ടുവിൽ സാധ്യമാണ്, കാരണം വാക്കുകൾ അക്ഷരമാലയുടെ ആരോഹണക്രമത്തിലാണ് അവിടെ ക്രമീകരിച്ചിരിക്കുന്നത്.

ബൈനറി തിരയൽ അൽഗോരിതം കുറഞ്ഞ തിരച്ചിലുകൾ കൊണ്ട് പട്ടികയിൽനിന്നും ഒരു അംഗത്തിൻറെ സ്ഥാനം കണ്ടെത്തുന്നു.

ബൈനറി തിരയലിന്റെ അൽഗോരിതം

1. ആരംഭിക്കുക
2. MAX ന്റെ വില സ്വീകരിക്കുക
3. I ← 0
4. ഘട്ടങ്ങൾ 5, 6 എന്നിവ I = MAX -1 ആകുന്നതുവരെ ആവർത്തിക്കുക
5. LIST [I] ലേയ്ക്ക് ഡാറ്റ സ്വീകരിക്കുക
6. I I +1
7. ITEM ന്റെ വില സ്വീകരിക്കുക

8. FIRST ← 0, LAST ← MAX -1
9. FIRST = LAST ആകുന്നതുവരെ ഘട്ടങ്ങൾ 10 മുതൽ 12 വരെ ആവർത്തിക്കുക
10. MIDDLE ← (FIRST + LAST) / 2
11. IF LIST [MIDDLE] = ITEM ആണെങ്കിൽ LOC ← 1, ഘട്ടം 13 ൽ പോകുക
12. IF ITEM < LIST [MIDDLE] ആണെങ്കിൽ LAST ← MIDDLE-1 അല്ലെങ്കിൽ FIRST ← MIDDLE+1
13. IF LOC = 1 ആണെങ്കിൽ തിരയൽ വിജയിച്ചു എന്ന് പ്രദർശിപ്പിക്കുക അല്ലെങ്കിൽ തിരയൽ പരാജയപ്പെട്ടു എന്നും പ്രദർശിപ്പിക്കുക.
14. അവസാനിപ്പിക്കുക

പ്രോഗ്രാം 8.6: അറെയിലുള്ള ഒരു അംഗം കണ്ടെത്തുന്നതിനുള്ള ബൈനറി തിരയൽ

```
#include <iostream>
using namespace std;
int main()
{   int LIST[25],MAX;
    int FIRST, LAST, MIDDLE, I, ITEM, LOC=-1;
    cout<<"How many elements? ";
    cin>>MAX;
    cout<<"Enter array elements in ascending order: ";
    for(I=0; I<MAX; I++)
        cin>>LIST[I];
    cout<<"Enter the item to be searched: ";
    cin>>ITEM;
    FIRST=0;
    LAST=MAX-1;
    while(FIRST<=LAST)
    {
        MIDDLE=(FIRST+LAST)/2;
        if(ITEM == LIST[MIDDLE])
        {
            LOC = MIDDLE;
            break;
        }
        if(ITEM < LIST[MIDDLE])
            LAST = MIDDLE-1;
        else
            FIRST = MIDDLE+1;
    }
    if(LOC != -1)
```

```
        cout<<"The item is found at position "<<LOC+1;
    else
        cout<<"The item is not found in the array";
    return 0;
}
```

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
How many elements? 7
Enter array elements in ascending order: 21 28 33 35 45 58 61
Enter the item to be searched: 35
The item is found at position 4
```

ബൈനറി തിരയൽ പ്രവർത്തനം വ്യക്തമാക്കുന്നതിന് താഴെപ്പറയുന്ന 7 (MAX=7) അംഗങ്ങളുള്ള അറെ പരിഗണിക്കാം, തിരയേണ്ടുന്ന അംഗം 45 ആണെന്നും കരുതുക.

0	1	2	3	4	5	6
21	28	33	35	45	58	61

MAX = 7

FIRST = 0

LAST = 6

FIRST<=LAST,

ആയതിനാൽ നമുക്ക് പ്രവർത്തനം ആരംഭിക്കാം

0	1	2	3	4	5	6
21	28	33	35	45	58	61

MIDDLE = (**FIRST**+**LAST**)/2 = (0+6)/2 = 3
ഇവിടെ **LIST[MIDDLE]** അതായത്, **LIST[3]** ന്റെ വിലയും **45** ഉം തുല്യമല്ല, എന്നാൽ **LIST[3]** ന്റെ വില തിരയൽ വിലയേക്കാൾ കുറവാണ്. അതിനാൽ
FIRST = **MIDDLE** + 1 = 3 + 1 = 4, **LAST** = 6

FIRST<=LAST,

ആയതിനാൽ അടുത്ത തിരച്ചിൽ ആരംഭിക്കാം

0	1	2	3	4	5	6
21	28	33	35	45	58	61

MIDDLE = (**FIRST**+**LAST**)/2 = (4+6)/2= 5
ഇവിടെ **LIST[MIDDLE]** അതായത്, **LIST[5]** ന്റെ വിലയും **45** ഉം തുല്യമല്ല, എന്നാൽ **LIST[5]** ന്റെ വില തിരയൽ വിലയേക്കാൾ വലുതാണ്. അതിനാൽ
FIRST = 4, **LAST** = **MIDDLE** - 1=5 -1 = 4,

FIRST<=LAST,

ആയതിനാൽ അടുത്ത തിരച്ചിൽ ആരംഭിക്കാം

0	1	2	3	4	5	6
21	28	33	35	45	58	61

MIDDLE = (**FIRST**+**LAST**)/2 = (4+4)/2 = 4
ഇവിടെ **LIST[MIDDLE]** അതായത്, **LIST[4]** ന്റെ വിലയും **45** ഉം തുല്യമാണ്, കൂടാതെ തിരയൽ വിജയകരമായി അവസാനിച്ചിരിക്കുന്നു.

ബൈനറി സെർച്ചിൽ, 100,00,00,000 (100 കോടി) അംഗങ്ങളുള്ള ഒരു അറെയിൽ ഒരു അംഗം തിരയാൻ പരമാവധി 30 താരതമ്യങ്ങൾ ആവശ്യമാണ്. അറെയിലെ അംഗങ്ങളുടെ എണ്ണം ഇരട്ടിയായെങ്കിൽ, ഒരു താരതമ്യം മാത്രമേ കൂടുതലായി ആവശ്യമുള്ളു.

പട്ടിക 8.1 ബൈനറി തിരയലും രേഖീയ തിരയലും തമ്മിലുള്ള വ്യത്യാസം കാണിച്ചിരിക്കുന്നു:

രേഖീയ തിരയൽ	ബൈനറി തിരയൽ
• അംഗങ്ങൾ ഏതെങ്കിലും രീതിയിൽ ക്രമീകരിക്കേണ്ടതില്ല • തിരയൽ പ്രവർത്തനത്തിന് കൂടുതൽ സമയം എടുക്കുന്നു • എല്ലാ അംഗങ്ങളേയും സന്ദർശിക്കേണ്ടതായി വരാം • കുറച്ച് അംഗങ്ങളുള്ള അറെകൾക്ക് അനുയോജ്യം.	• അംഗങ്ങളെ ആരോഹണ ക്രമത്തിലോ അവരോഹണ ക്രമത്തിലോ ക്രമീകരിച്ചിരിക്കണം • തിരയൽ പ്രവർത്തനത്തിന് വളരെ കുറച്ച് സമയമേ ആവശ്യമുള്ളു • എല്ലാ അംഗങ്ങളേയും സന്ദർശിക്കേണ്ടതില്ല • കൂടുതൽ അംഗങ്ങളുള്ള അറെകൾക്ക് അനുയോജ്യം

പട്ടിക 8.1 ബൈനറി സെർച്ചും രേഖീയ സെർച്ചും തമ്മിലുള്ള താരതമ്യം

8.3 ദ്വിമാന അറെകൾ (Two dimensional Arrays)

50 വിദ്യാർത്ഥികളുടെ ആറു വ്യത്യസ്ത വിഷയങ്ങളിലെ മാർക്കുകൾ നമുക്ക് സൂക്ഷിക്കേണ്ടതുണ്ട് എന്ന് കരുതുക. ഇവിടെ നമുക്ക് 50 അംഗങ്ങൾ ഉള്ള 6 ഏകമാന അറെകൾ ഉപയോഗിക്കാം. എന്നാൽ ഈ ക്രമീകരണം കൈകാര്യം ചെയ്യുന്നത് എളുപ്പമുള്ള കാര്യമല്ല. ഈ സാഹചര്യത്തിൽ നമുക്ക് അറെകളുടെ അറെ അല്ലെങ്കിൽ ദ്വിമാന അറെ ഉപയോഗിക്കാം.

ഒരു ദ്വിമാന അറെയിലെ ഓരോ അംഗവും ഒരു അറെയാണ്. ഉദാഹരണമായി, AR[m][n] എന്ന ദ്വിമാന അറെയിൽ n അംഗങ്ങളുള്ള m അറെകൾ ഉണ്ട്. അല്ലെങ്കിൽ m വരികളും n നിരകളും അടങ്ങുന്ന ഒരു പട്ടികയാണ് AR [m][n] എന്ന ദ്വിമാന അറെ.

8.3.1 ദ്വിമാന അറെകളുടെ പ്രഖ്യാപനം (Declaring 2D Array)

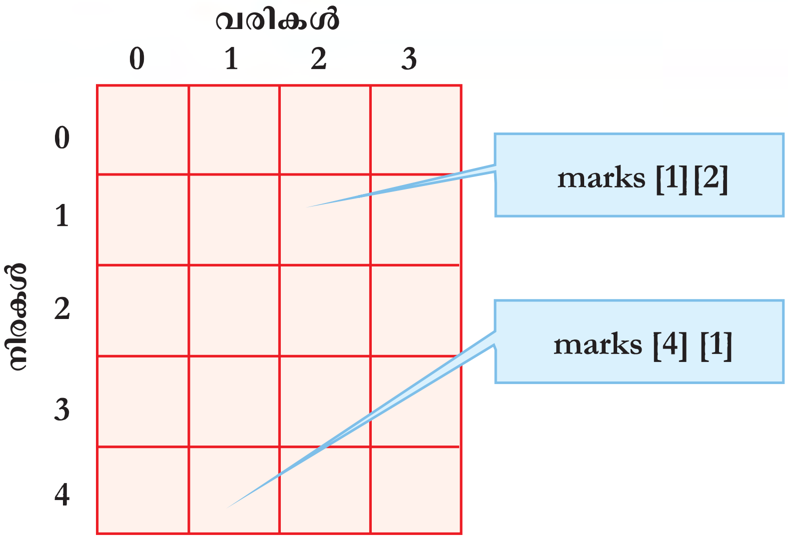
C ++ ലെ ദ്വിമാന അറെ നീക്കിവെയ്ക്കലിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു

```
data_type  array_name[rows][columns];
```

വാക്യഘടനയിൽ data_type എന്നത് അറെയിലെ അംഗങ്ങളുടെ ഡേറ്റയുടെ ഇനമാണ് സൂചിപ്പിക്കുന്നത്. array_name എന്നത് അറെയുടെ പേരും rows എന്നത് ദ്വിമാന അറെയിലെ വരികളുടെ എണ്ണവും columns എന്നത് ദ്വിമാന അറെയിലെ നിരകളുടെ എണ്ണവും സൂചിപ്പിക്കുന്നു. വരികളുടെയും നിരകളുടെയും സൂചിക 0-ൽ ആരംഭിച്ച് യഥാക്രമം വരികൾ rows -1 ലും നിരകൾ coulmns - 1 ലും അവസാനിക്കും. 5 വരികളും 4 നിരകളും ഉള്ള ഒരു ദ്വിമാന അറെ പ്രഖ്യാപനത്തിന്റെ ഉദാഹരണം താഴെ കൊടുക്കുന്നു.

```
int  marks[5][4];
```

ചിത്രം 8.4 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ. ഈ അറെയിലെ അംഗങ്ങൾ marks[0][0], marks[0][1], marks[0][2], marks[0][3], marks[1][0], marks[1][1], ..., marks[4][3] എന്നിവയാകുന്നു.



ചിത്രം 8.4 ഒരു ദ്വിമാന അറെയുടെ ഘടന

ഒരു ദ്വിമാന അറേക്ക് ആവശ്യമായ മെമ്മറിയുടെ അളവ് അതിന്റെ ഡാറ്റ ഇനം, വരികളുടെ എണ്ണം, നിരകളുടെ എണ്ണം എന്നിവയുടെ അടിസ്ഥാനത്തിലാണ് കണക്കാക്കുന്നത്. ദ്വിമാന അറേക്ക് ആവശ്യമായ ആകെ ബൈറ്റുകളുടെ എണ്ണം കണക്കുകൂട്ടുന്നതിനുള്ള സൂത്രവാക്യം താഴെ കൊടുത്തിരിക്കുന്നു.

ആകെ ബൈറ്റുകൾ = `sizeof`(ഡാറ്റ ഇനം) × വരികളുടെ എണ്ണം × നിരകളുടെ എണ്ണം

ഉദാഹരണത്തിന്, മുകളിൽ പറഞ്ഞിരിക്കുന്ന `marks[5][4]` ന് $4 \times 5 \times 4 = 80$ ബൈറ്റ് മെമ്മറി ആവശ്യമാണ്.

8.3.2 മെട്രിക്സായി ദ്വിമാന അറെകൾ (Matrices as 2D arrays)

ഗണിതശാസ്ത്രത്തിലെ ഉപയോഗപ്രദമായ ഒരു ആശയമാണ് മെട്രിക്സ്. ഒരു മെട്രിക്സ് എന്നത് m വരികളിലും n നിരകളിലുമായി ഒരു പട്ടികയുടെ രൂപത്തിൽ ക്രമീകരിച്ചിരിക്കുന്ന m × n സംഖ്യകളുടെ ഒരു ഗണമാണെന്ന് നമുക്കറിയാം. ദ്വിമാന അറെയുടെ സഹായത്തോടെ മെട്രിക്സ് പ്രതിനിധീകരിക്കാനാകും. ഒരു ദ്വിമാന അറെ പ്രോസസ് ചെയ്യുന്നതിന് നിങ്ങൾ നെസ്റ്റഡ് ലൂപ്പ് ഉപയോഗിക്കണം. ഒരു ലൂപ്പ് വരികളേയും അടുത്തത് നിരകളേയും പ്രോസസ്സുചെയ്യുന്നു. സാധാരണയായി പുറത്തെ ലൂപ്പ് വരികൾക്കും അകത്തെ ലൂപ്പ് നിരകൾക്കും വേണ്ടിയുള്ളതാണ്. പ്രോഗ്രാം 8.7 ഉപയോഗിച്ച് m വരികളും n നിരകളും ഉള്ള ഒരു മെട്രിക്സ് പ്രോസസ്സ് ചെയ്യുന്നു.

പ്രോഗ്രാം 8.7. m വരികളും n നിരകളും ഉള്ള ഒരു മെട്രിക്സ് നിർമിക്കുന്നു

```
#include <iostream>
using namespace std;
int main()
{   int m, n, row, col, mat[10][10];
```

```
    cout<< "Enter the order of matrix: ";
    cin>> m >> n;
    cout<<"Enter the elements of matrix\n";
    for (row=0; row<m; row++)
        for (col=0; col<n; col++)
            cin>>mat[row][col];
    cout<<"The given matrix is:";
    for (row=0; row<m; row++)
    {
        cout<<endl;
        for (col=0; col<n; col++)
            cout<<mat[row][col]<<"\t";
    }
    return 0;
}
```

മെട്രിക്സ് നിർമ്മാണം

അംഗങ്ങളെ മെട്രിക്സ് മാതൃകയിൽ പ്രദർശിപ്പിക്കൽ

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
Enter the order of matrix: 3  4
Enter the elements of matrix
1  2  3  4  2  3  4   5   3  4   5  6
The given matrix is:
1    2    3    4
2    3    4    5
3    4    5    6
```

[redacted]

രണ്ടു മെട്രിക്സുകളുടെ ക്രമവും അംഗങ്ങളേയും സ്വീകരിച്ച് അവയുടെ തുക കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്ക് ചർച്ച ചെയ്യാം.

പ്രോഗ്രാം 8. 8 രണ്ട് മെട്രിക്സുകളുടെ തുക കണ്ടെത്തുന്നതിന്

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{   int m1, n1, m2, n2, row, col;
    int A[10][10], B[10][10], C[10][10];
    cout<<"Enter the order of first matrix: ";
    cin>>m1>>n1;
    cout<<"Enter the order of second matrix: ";
```

exit() ഫംഗ്ഷൻ ഉപയോഗിക്കുന്നതിനായി

```
    cin>>m2>>n2;
    if(m1!=m2 || n1!=n2)
    {
        cout<<"Addition is not possible";
        exit(0);
    }
    cout<<"Enter the elements of first matrix\n";
    for (row=0; row<m1; row++)
        for (col=0; col<n1; col++)
            cin>>A[row][col];
    cout<<"Enter the elements of second matrix\n";
    for (row=0; row<m2; row++)
        for (col=0; col<n2; col++)
            cin>>B[row][col];
    for (row=0; row<m1; row++)
        for (col=0; col<n1; col++)
            C[row][col] = A[row][col] + B[row][col];
    cout<<"Sum of the matrices:\n";
    for(row=0; row<m1; row++)
    {
        cout<<endl;
        for (col=0; col<n1; col++)
            cout<<C[row][col]<<"\t";
    }
}
```

> 2 മെട്രിക്സുകളുടെ ഓർഡർ വ്യത്യാസമാണെങ്കിൽ പ്രോഗ്രാം അവസാനിപ്പിക്കുന്നു.

> ആദ്യ മെട്രിക്സിന്റെ നിർമ്മാണം

> രണ്ടാം മെട്രിക്സിന്റെ നിർമ്മാണം

> മെട്രിക്സിന്റെ തുക കണ്ടുപിടിക്കുന്നു

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
Enter the order of first matrix: 3   4
Enter the order of second matrix: 3   4
Enter the elements of first matrix
2     5     -3     7
5     12    4      9
-3    0     6      -5
Enter the elements of second matrix
1     4     3      5
4     -5    7      13
3     -4    7      9
Sum of the matrices:
3     9     0      12
9     7     11     22
0     -4    13     4
```

> ഇവിടെ അംഗങ്ങളെ നൽകിയിരിക്കുന്നത് മെട്രിക്സ് മാതൃകയിലാണ്, പക്ഷെ ഇങ്ങനെ നൽകൽ നിർബന്ധമില്ല

പ്രോഗ്രാം 8.8 ൽ C[i][j] = A[i][j] + B[i][j] എന്നതിനു പകരം C[i][j] = A[i][j] – B[i][j] ഉപയോഗിച്ചാൽ രണ്ട് മെട്രിക്സുകൾ തമ്മിലുള്ള വ്യത്യാസം കണ്ടുപിടിക്കാൻ കഴിയും.

ഇനി നമുക്ക് സമചതുര മെട്രിക്സിന്റെ വികർണ്ണ അംഗങ്ങളുടെ തുക കണ്ടുപിടിക്കാൻ ഒരു പ്രോഗ്രാം എഴുതാം. ഒരു മെട്രിക്സിലെ വരികളുടേയും നിരകളുടേയും എണ്ണം ഒരേ പോലെയാണെങ്കിൽ അത്തരം മെട്രിക്സ് ഒരു സമചതുര മെട്രിക്സ് ആയിരിക്കും. ഒരു സമചതുര മെട്രിക്സിന് രണ്ടു വികർണ്ണങ്ങൾ ഉണ്ട്. `mat[0][0]`, `mat[1][1]`, `mat[2][2]`, `...`, `mat[n-1][n-1]`, അംഗങ്ങളെ മുൻനിര അല്ലെങ്കിൽ മുഖ്യ വികർണ്ണ അംഗങ്ങൾ എന്ന് വിളിക്കുന്നു. മുഖ്യ വികർണ അംഗങ്ങളുടെ തുക കണ്ടുപിടിക്കുന്നതിന് പ്രോഗ്രാം 8.9 ഉപയോഗിക്കാം.

പ്രോഗ്രാം 8.9: ഒരു മെട്രിക്സിന്റെ മുഖ്യ വികർണ്ണ അംഗങ്ങളുടെ തുക കണ്ടെത്തുക.

```
#include <iostream>
using namespace std;
int main()
{   int mat[10][10], n, i, j, s=0;
    cout<<"Enter the rows/columns of square matrix: ";
    cin>>n;
    cout<<"Enter the elements\n";
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            cin>>mat[i][j];
    cout<<"Major diagonal elements are\n";
    for(i=0; i<n; i++)
    {
        cout<<mat[i][i]<<"\t";
        s = s + mat[i][i];
    }
    cout<<"\nSum of major diagonal elements is: ";
    cout<<s;
    return 0;
}
```

തുക കാണുന്നതിന് വികിർണ അംഗങ്ങളെ മാത്രം ഉപയോഗിക്കുന്നു.

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
Enter the rows/columns of square matrix: 3
Enter the elements
3    5    -2
7    4    0
2    8    -1
Major diagonal elements are
3    4    -1
Sum of major diagonal elements is: 6
```

ഓരോ മെട്രിക്സിനും ഒരു ട്രാൻസ്പോസ് ഉണ്ട്. വരിയിലെ അംഗങ്ങൾ നിരയായും അല്ലെങ്കിൽ തിരിച്ചും മാറ്റം വരുത്തിയാണ് ഇത് ലഭിക്കുന്നത്. പ്രോഗ്രാം 8.10 ഈ പ്രക്രിയ വ്യക്തമാക്കുന്നു.

പ്രോഗ്രാം 8.10: ഒരു മെട്രിക്സിന്റെ ട്രാൻസ്പോസ് കണ്ടുപിടിക്കുക.

```
#include <iostream>
using namespace std;
int main()
{   int ar[10][10], m, n, row, col;
    cout<<"Enter the order of matrix: ";
    cin>>m>>n;
    cout<<"Enter the elements\n";
    for(row=0; row<m; row++)
        for(col=0; col<n; col++)
            cin>>ar[row][col];
    cout<<"Original matrix is\n";
    for(row=0; row<m; row++)
    {
        cout<<"\n";
        for(col=0; col<n; col++)
            cout<<ar[row][col]<<"\t";
    }
    cout<<"\nTranspose of the entered matrix is\n";
    for(row=0; row<n; row++)
    {
        cout<<"\n";
        for(col=0; col<m; col++)
            cout<<ar[col][row]<<"\t";
    }
    return 0;
}
```

വരികളുടെയും നിരകളുടെയും എണ്ണം പരസ്പരം മാറ്റിയത് ശ്രദ്ധിക്കുക

സൂചികകളും പരസ്പരം മാറിയത് ശ്രദ്ധിക്കുക.

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
Enter the order of matrix: 4    3
Enter the elements
3     5     -1
2     12    0
6     8     4
7     -5    6
Original matrix is
3     5     -1
```

ഈ അംഗങ്ങളെ ഒരു വരിയിലായും നൽകാവുന്നതാണ്.

```
2     12    0
6     8     4
7     -5    6
Transpose of the entered matrix is
3     2     6     7
5     12    8     -5
-1    0     4     6
```

ഡാറ്റ പട്ടിക രൂപത്തിൽ ക്രമീകരിക്കപ്പെടുമ്പോൾ, ചില സാഹചര്യങ്ങളിൽ നമുക്ക് ഓരോ വരിയിലേയും, നിരയിലേയും അംഗങ്ങളുടെ ആകെത്തുക ആവശ്യമായി വരും. പ്രോഗ്രാം 8.11 ഈ ചുമതല നിർവ്വഹിക്കാൻ കമ്പ്യൂട്ടറിനെ സഹായിക്കുന്നു.

പ്രോഗ്രാം 8.11

```
#include <iostream>
using namespace std;
int main()
{
int ar[10][10], rsum[10]={0}, csum[10]={0};
int m, n, row, col;
cout<<"Enter the number of rows & columns in the array: ";
cin>>m>>n;
cout<<"Enter the elements\n";
for(row=0; row<m; row++)
    for(col=0; col<n; col++)
        cin>>ar[row][col];
for(row=0; row<m; row++)
    for(col=0; col<n; col++)
    {
        rsum[row] += ar[row][col];
        csum[col] += ar[row][col];
    }
cout<<"Row sum of the 2D array is\n";
for(row=0; row<m; row++)
    cout<<rsum[row]<<"\t";
cout<<"\nColumn sum of the 2D array is\n";
for(col=0; col<n; col++)
    cout<<csum[col]<<"\t";
return 0;
}
```

വരിയിലെയും നിരയിലെയും അംഗങ്ങളെ പ്രത്യേകം കൂട്ടുകയും, ഓരോ തുകയും പ്രസ്തുത അറെയിലെ അനുസ്യൂതമായ സ്ഥാനങ്ങളിൽ സൂക്ഷിക്കുകയും ചെയ്യുന്നു.

ഔട്ട്പുട്ടിന്റെ മാതൃക:

```
Enter the number of rows & columns in the array: 3    4
Enter the elements
3     12    5     0
4     -6    2     1
5     7     -6    2
Row sum of the 2D array is
20    1     8
Column sum of 2D array is
12    13    1     3
```

8.4 ബഹുമുഖ അറെകൾ (Multi dimensional arrays)

ഒരു ദ്വിമാന അറെയുടെ ഓരോ അംഗവും മറ്റൊരു അറെയായിരിക്കാം. അത്തരത്തിലുള്ള ഒരു അറെയെ 3D (ത്രിമാന അറെ) അറെ എന്ന് പറയുന്നു.

```
data_type  array_name[size_1][size_2][size_3];
```

എന്ന പ്രസ്താവന ഉപയോഗിച്ച് ഒരു ത്രിമാന അറെയുടെ പ്രഖ്യാപനം നടത്താം. മൂന്നു സൂചിക ഉപയോഗിച്ച് ഒരു 3D അറെയുടെ അംഗങ്ങളെ ഉപയോഗിക്കുകകയും ചെയ്യാം. Ar[10][5][3] ഒരു 3D അറെ ആണെങ്കിൽ ആദ്യത്തെ അംഗം Ar [0][0][0] അവസാന അംഗം Ar [9][4][2] ആയിരിക്കും. ഈ അറെയിൽ 150 (10×5×3) അംഗങ്ങൾ അടങ്ങിയിരിക്കാം.

നമുക്ക് സംഗ്രഹിക്കാം

അറെ എന്നാൽ തുടർച്ചയായ മെമ്മറി സ്ഥാനങ്ങളിൽ ശേഖരിച്ചു വച്ചിട്ടുള്ള ഒരേ തരത്തിലുള്ള ഡാറ്റകളുടെ സമൂഹമാണ്. ഒരു പേരിൽ ഒരേ തരത്തിലുള്ള ഒരു കൂട്ടം വിലകൾ ശേഖരിക്കുന്നതിനായി അറെകൾ ഉപയോഗിക്കുന്നു. ഒരു അറെയിലെ എല്ലാ അംഗങ്ങളേയും സൂചികയുടെ സഹായത്താൽ ഉപയോഗിക്കുവാൻ കഴിയും. for ലൂപ്പിന്റെ സഹായത്തോടെ അറെയിലെ അംഗങ്ങളെ എളുപ്പത്തിൽ ഉപയോഗിക്കുന്നു. കടന്നുപോകൽ, ക്രമപ്പെടുത്തൽ, തിരയൽ തുടങ്ങിയ പ്രവർത്തനങ്ങൾ അറെകളിൽ നടത്തപ്പെടുന്നു. അറെയിലെ അംഗങ്ങളെ ക്രമീകരിക്കുന്നതിന് ബബിൾ സോർട്ട്, സെലക്ഷൻ സോർട്ട് എന്നീ രീതികൾ ഉപയോഗിക്കുന്നു. ഒരു അറെയിൽ ഒരു അംഗത്തെ തിരയാൻ രേഖീയ തിരയൽ, ബൈനറി തിരയൽ എന്നീ വിദ്യകൾ ഉപയോഗിക്കുന്നു. മെട്രിക്സ് സംബന്ധമായ കാര്യങ്ങൾ ചെയ്യുന്നതിന് ദ്വിമാന അറെകൾ ഉപയോഗിക്കുന്നു. ദ്വിമാന അറെയിലുള്ള ഒരു അംഗത്തെ പരാമർശിക്കുന്നതിന് നമുക്ക് രണ്ട് സൂചികകൾ ഉണ്ടാകും. ദ്വിമാന അറെകൾ കൂടാതെ, ബഹുമുഖ അറെകളും സൃഷ്ടിക്കാൻ കഴിയും.

പഠനനേട്ടങ്ങൾ

ഈ അധ്യായത്തിന്റെ പൂർത്തീകരണത്തിനുശേഷം പഠിതാവിന് താഴെ പറയുന്നവ ആർജിക്കാൻ കഴിയും.

- അറെ ഉപയോഗിക്കേണ്ട സാഹചര്യങ്ങളുടെ തിരിച്ചറിവ്
- ഏകമാന, ദ്വിമാന അറെകളുടെ പ്രഖ്യാപനവും പ്രാരംഭവില നൽകലും
- തിരയൽ, ക്രമപ്പെടുത്തൽ തുടങ്ങി വിവിധങ്ങളായ അറെ പ്രവർത്തനങ്ങളുടെ യുക്തി നിർമ്മാണം
- ദ്വിമാന അറെയുടെ സഹായത്തോടെ മെട്രിക്സുമായി ബന്ധപ്പെട്ട പ്രശ്ന പരിഹാരങ്ങൾ

ലാബ് പ്രവർത്തനങ്ങൾ

1. 12 മാസത്തെ വിൽപ്പനയുടെ തുക SalesAmt എന്ന അറെയിലേക്ക് ഇൻപുട്ട് ചെയ്തതിനുശേഷം വിൽപനയുടെ ആകെത്തുകയും ശരാശരിയും കണ്ടെത്തുന്നതിനുള്ള ഒരു C++ പ്രോഗ്രാം എഴുതുക.

2. N സംഖ്യകളുടെ ഒരു അറെ നിർമ്മിച്ചതിന് ശേഷം സംഖ്യകളുടെ ശരാശരി കണ്ടെത്തുകയും ശരാശരിക്ക് മുകളിൽ ഉള്ള സംഖ്യകൾ പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നതിന് ഒരു C++ പ്രോഗ്രാം എഴുതുക.

3. വില, അളവ്, തൂക്കം എന്നിവ ശേഖരിക്കുന്നതിനായി price, quanttiy, amount എന്നിങ്ങനെ 3 അറെകൾ നിർമ്മിക്കുക. ഓരോ അറെയും 10 അംഗങ്ങളെ ഉൾക്കൊള്ളാൻ കഴിയുന്നവയായിരിക്കണം. price, quanttiy എന്നീ അറകളിലേക്ക് വിലകൾ നൽകുക. amount അറെയുടെ മൂല്യം amount[i] = price[i] × quanttiy[i] എന്നായിരിക്കണം. എല്ലാ ഡാറ്റയും നൽകിയ ശേഷം, താഴെ കൊടുത്തിരിക്കുന്ന രീതിയിൽ ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കുന്നതിനു വേണ്ടിയുള്ള ഒരു C++ പ്രോഗ്രാം എഴുതുക

Price	Quantity	Amount
______	______	______
______	______	______

4. ഒരു അറെയിലേക്ക് 10 സംഖ്യകൾ നൽകിയതിന് ശേഷം, അവയിലെ ഏറ്റവും വലിയ സംഖ്യയും ചെറിയ സംഖ്യയും കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു C++ പ്രോഗ്രാം എഴുതുക.

5. ഓർഡർ n ആയിട്ടുള്ള ഒരു സമചതുര മെട്രിക്സിന്റെ വികർണ്ണത്തിനു മുകളിലുള്ള അംഗങ്ങളെ പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക. ഉദാഹരണത്തിന് താഴെ കാണുന്ന മെട്രിക്സ് പരിഗണിക്കുക.

```
2   3   1
7   1   5
2   5   1
```

ഉത്തരം ഇവിടെ കാണുന്ന വിധമായിരിക്കും

```
2   3   1
    1   5
        1
```

5. ഓർഡർ n ആയിട്ടുള്ള ഒരു സമചതുര മെട്രിക്സിന്റെ വികർണ്ണത്തിനു താഴെയുള്ള അംഗങ്ങളെ പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക. ഉദാഹരണത്തിന് താഴെ കാണുന്ന മെട്രിക്സ് പരിഗണിക്കുക.

```
2   3   1
7   1   5
2   5   1
```

ഉത്തരം ഇവിടെ കാണുന്ന വിധമായിരിക്കും

```
2
7   1
2   5   7
```

7. താഴെ കാണിച്ചിരിക്കുന്നതുപോലെ പാസ്കൽസ് ത്രികോണം (Pascal's Triangle) പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഒരു C++ പ്രോഗ്രാം എഴുതുക

```
1
1   2   1
1   3   3   1
1   4   6   4   1
```

മാതൃകാ ചോദ്യങ്ങൾ

1. ഒരു അറെയിലെ എല്ലാ അംഗങ്ങളും ________ ഡാറ്റ ഇനം ആയിരിക്കണം.
2. പത്തു അംഗങ്ങളുള്ള ഒരു അറെയുടെ സൂചിക ________ മുതൽ ________ വരെയുള്ള സംഖ്യകൾ ആയിരിക്കും.
3. ഒരു അറെയിലെ അംഗത്തെ ________ ഉപയോഗിച്ച് ഉപയോഗിക്കാം.
4. AR ഒരു അറെയാണെങ്കിൽ, AR[7] എത് അംഗത്തെ പ്രതിനിധാനം ചെയ്യുന്നു?
5. int a[3]={2,3,4}; എന്ന അറെയിൽ a[1] ന്റെ വില എന്ത്?
6. int a[]={1,2,3,4}; എന്ന അറെയിൽ a[2] ന്റെ വില എന്ത്?
7. int a[5]={1,2,3,4}; എന്ന അറെയിൽ a[4] ന്റെ വില എന്ത്?
8. 89, 75, 82, 93, 78, 95. എന്നീ സ്കോറുകൾ score എന്ന അറെയിലേക്ക് പ്രാരംഭ വിലയായി നല്കുന്നതിനുള്ള പ്രസ്താവന എഴുതുക
9. ഒരു അറെയിലെ എല്ലാ അംഗങ്ങളേയും പ്രദർശിപ്പിക്കുന്നത് ________ പ്രവർത്തനത്തിന് ഒരു ഉദാഹരണമാണ്.

10. int a[2][3]; എന്ന അറെ നിർമിക്കുന്നതിന് എത്ര ബൈറ്റുകൾ ആവശ്യമാണ്.

11. m അംഗങ്ങളുള്ള അറെയിൽ, ബൈനറി തിരയലിൽ ഒരു അംഗത്തെ കണ്ടെത്തുന്നതിന് പരമാവധി n തിരയൽ ആവശ്യമാണ്. എന്നാൽ അംഗങ്ങളുടെ എണ്ണം ഇരട്ടിയെങ്കിൽ എത്ര തിരയൽ ആവശ്യമായി വരും?

12. Mark എന്ന അറെയിലേക്ക് പൂജ്യം പ്രാരംഭ വിലയായി നല്കുന്നതിനുള്ള പ്രസ്താവന എഴുതുക.

13. പ്രസ്താവന ശരിയോ തെറ്റോ: 10 അംഗങ്ങളുള്ള അറെയിലെ പതിനാറാമത്തെ അംഗത്തെ ഉപയോഗിക്കുവാൻ നിങ്ങൾ ശ്രമിക്കുകയാണെങ്കിൽ കമ്പൈലർ തെറ്റ് രേഖപ്പെടുത്തും.

ലഘു ഉപന്യാസ ചോദ്യങ്ങൾ

1. അറെ നിർവ്വചിക്കുക.
2. int studlist[1000]; എന്ന പ്രഖ്യാപന പ്രസ്താവന അർത്ഥമാക്കുന്നത് എന്ത്?
3. ഒരു ഏകമാന അറെയ്ക്കായി മെമ്മറി അനുവദിക്കുന്നത് എങ്ങനെ?
4. 10 അംഗങ്ങളുള്ള അറെയിലേക്ക് സംഖ്യകളെ സ്വീകരിച്ച് അവയിലെ ഒറ്റ സംഖ്യയുടെയും ഇരട്ട സംഖ്യയുടെയും എണ്ണം പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ കോഡ് ശകലങ്ങൾ എഴുതുക.
5. 2, 3, 4, 5 എന്നീ സംഖ്യകൾ ഒരു അറെയിൽ ശേഖരിക്കുന്നതിനുള്ള പ്രാരംഭ വിലനൽകൽ പ്രസ്താവന എഴുതുക.
6. കടന്നുപോകൽ എന്നാൽ എന്ത്?
7. ക്രമപ്പെടുത്തൽ നിർവ്വചിക്കുക.
8. തിരയൽഎന്നാൽ എന്ത്?
9. ബബിൾ സോർട്ട് എന്നാൽ എന്ത്?
10. ബൈനറി തിരയൽ എന്നാൽ എന്ത്?
11. ഒരു ദ്വിമാന അറെ നിർവ്വചിക്കുക
12. ഒരു ദ്വിമാന അറെയ്ക്കായി മെമ്മറി അനുവദിക്കുന്നത് എങ്ങനെ?

ഉപന്യാസ ചോദ്യങ്ങൾ

1. അറെ AR 25, 81, 36, 15, 45, 58, 70 എന്നീ അംഗങ്ങൾ ഉൾക്കൊള്ളുന്നു. 45 എന്ന വില തിരയുന്നതിനായുള്ള ബൈനറി തിരയൽ രീതിയുടെ പ്രവർത്തനം വിശദമാക്കുക.
2. തുല്യ വലിപ്പത്തിലുള്ള രണ്ടു അറെയിൽ അംഗങ്ങളെ സ്വീകരിച്ച് അതത് സ്ഥാനങ്ങളിലെ അംഗങ്ങൾ തമ്മിലുള്ള വ്യത്യാസം കണ്ടെത്തുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.

3. 3, 32, 25, 44, 16, 37, 12 എന്നീ അംഗങ്ങളെ ക്രമീകരിക്കുന്നതിനുള്ള ബബിൾ സോർട്ടിന്റെ പ്രവർത്തനരീതി വിശദീകരിക്കുക.

4. 24, 45, 98, 56, 76, 24, 15 എന്നീ അംഗങ്ങളെ ക്രമീകരിക്കുന്നതിനുള്ള രേഖീയ തിരയലിന്റെ പ്രവർത്തനരീതി വിശദീകരിക്കുക.

5. രണ്ട് മെട്രിക്സുകൾ തമ്മിൽ വ്യവകലനം ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.

6. ഒരു ദ്വിമാന അറെയിൽ അംഗങ്ങളുടെ തുകയും ശരാശരിയും കണ്ടെത്താനായി പ്രോഗ്രാം എഴുതുക.

7. ഒരു ദ്വിമാന അറെയിലെ ഏറ്റവും വലിയ സംഖ്യ കണ്ടെത്തുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക.

****

9

സ്ട്രിങ് കൈകാര്യം ചെയ്യലും ഇൻപുട്ട് /ഔട്ട്പുട്ട് ഫങ്ഷനുകളും

പ്രധാന ആശയങ്ങൾ

- അറെ ഉപയോഗിച്ചുള്ള സ്ട്രിങ് കൈകാര്യം ചെയ്യൽ
- സ്ട്രിങ്ങിനു വേണ്ടിയുള്ള മെമ്മറി നീക്കിവെയ്ക്കൽ
- സ്ട്രിങ്ങിനു മേലുള്ള ഇൻപുട്ട് /ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങൾ
- കാരക്റ്റർ ഇൻപുട്ട് /ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങൾക്ക് വേണ്ടിയുള്ള കൺസോൾ ഫങ്ഷനുകൾ
  - getchar()
  - putchar()
- ഇൻപുട്ട് /ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങൾക്കുള്ള സ്ട്രീം ഫങ്ഷനുകൾ
  - ഇൻപുട്ട് ഫങ്ഷനുകൾ get(), getline()
  - ഔട്ട്പുട്ട് ഫങ്ഷനുകൾ put(), write()

V9V1U9

ഒരേ തരത്തിലുള്ള അനേകം ഡാറ്റയെ കൈകാര്യം ചെയ്യുന്നതിനുള്ള ഫലപ്രദമായ ഉപാധിയാണ് അറെകൾ (Arrays) എന്ന് നാം പഠിച്ചു കഴിഞ്ഞു. ഇതിനു മുമ്പ് ചർച്ച ചെയ്ത മിക്കവാറും പ്രോഗ്രാമുകളിലും അറേകൾ ഉപയോഗിച്ചിരിക്കുന്നത് ന്യൂമെറിക് ഡാറ്റ ഇനങ്ങളെ പ്രോസസ്സ് ചെയ്യുന്നതിനാണ്. എന്നാൽ സ്ട്രിങ് ടൈപ്പ് ഡാറ്റയും ഉണ്ടെന്നത് നമുക്കറിയാവുന്ന ഒരു വസ്തുതയുമാണ്. അത്തരം ഡാറ്റയെ മെമ്മറിയിൽ ശേഖരിക്കുന്നതും പ്രോസസ്സ് ചെയ്യുന്നതും നാം ഇവിടെ ചർച്ചചെയ്യുന്നു. കൂടാതെ സ്ട്രിങ്ങുകളെയും കാരക്റ്ററുകളെയും കൈകാര്യം ചെയ്യുന്നതിനുള്ള ചില ഇൻപുട്ട്/ഔട്ട്പുട്ട് അന്തർനിർമിത ഫങ്ഷനുകളും (Built in functions) ഇവിടെ പ്രതിപാദിക്കപ്പെടുന്നുണ്ട്.

9.1. അറെ ഉപയോഗിച്ചുള്ള സ്ട്രിങ് കൈകാര്യം ചെയ്യൽ (String handling using arrays)

C++ ലെ ഒരുതരം ലിറ്ററലാണ് സ്ട്രിങ്. പ്രോഗ്രാമുകളിൽ ഇവ കാണപ്പെടുന്നത് ഉദ്ധരണിക്കുള്ളിൽ (Double quotes) തുടർച്ചയായുള്ള കാരക്റ്ററുകളായാണ്. നിങ്ങളോട് പേര് ശേഖരിക്കുവാനും പ്രദർശിപ്പിക്കുന്നതിനുമുള്ള ഒരു പ്രോഗ്രാം എഴുതുവാൻ ആവശ്യപ്പെട്ടുവെന്നിരിക്കട്ടെ. ഡാറ്റ ശേഖരിക്കുവാൻ വേരിയബിൾ ആവശ്യമാണെന്ന് ഇതിനു മുമ്പ് നാം പഠിച്ചിട്ടുണ്ട്. my_name എന്ന വേരിയബിൾ ഒരു ഐഡന്റിഫയർ ആയി ഇവിടെ നമുക്ക് ഉപയോഗിക്കാം. ഒരു വേരിയബിൾ ഉപയോഗിക്കുന്നതിനു മുമ്പ് അത് പ്രഖ്യാപിക്കണമെന്നുള്ളത് ഈ അവസരത്തിൽ തീർച്ചയായും ഓർമിക്കേണ്ടതാണ്. സ്ട്രിങ് ഡാറ്റയെ സൂചിപ്പിക്കാനുള്ള അടിസ്ഥാന ഡാറ്റ ഇനം നിലവിലില്ലാത്തതിനാൽ ഏതു തരം ഡാറ്റയാണ് സ്ട്രിങ് ഡാറ്റ ശേഖരിക്കുന്ന വേരിയബിൾ പ്രഖ്യാപനത്തിന് ഉപയോഗിക്കാനാവുക എന്ന് പറയാൻ സാധിക്കില്ല? അതു

കൊണ്ട് നമുക്ക് char ഡാറ്റ ഇനത്തെക്കുക്കുറിച്ച് ആലോചിക്കാം. എന്നാൽ അവിടെയും ഒരു പ്രശ്നമുണ്ട്. char ഡാറ്റ ഇനത്തിന് ഒരു കാരക്റ്റർ മാത്രമേ ശേഖരിക്കുവാൻ കഴിയുകയുള്ളു. അതുകൊണ്ടുതന്നെയാണ് സ്ട്രിങ് എന്നത് തുടർച്ചയായ കാരക്റ്ററുകളുടെ ഇൻപുട്ട് ആയി സ്വീകരിക്കേണ്ടി വരുന്നത് .

"Niketh" എന്ന പേര് പരിഗണിക്കുക. ഇത് ആറ് കാരക്റ്ററുകൾ ഉൾക്കൊള്ളുന്ന ഒരു സ്ട്രിങ് ആണ്. എന്നാൽ ഒരു കാരക്റ്റർ അറെയ്ക്ക് ഒന്നിലധികം കാരക്റ്ററുകളെ ശേഖരിക്കുവാൻ കഴിയുമെന്ന് നമുക്കറിയാം. അതുകൊണ്ടു ഒരു അറേയെ താഴെ കാണുന്നവിധം പ്രഖ്യാപിക്കാവുന്നതാണ്.

```
char my_name[10];
```

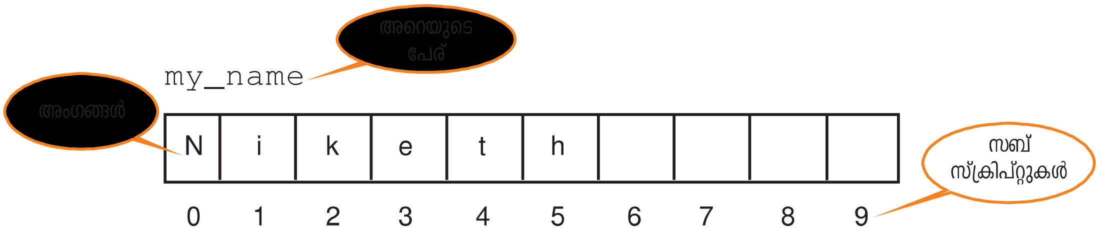
my_name എന്ന് പേരുള്ള അറേയിൽ ഒരു ബൈറ്റ് വീതം വലിപ്പമുള്ള തുടർച്ചയായ പത്ത് മെമ്മറി സ്ഥാനങ്ങൾ നീക്കിവച്ചിട്ടുണ്ട്. ഈ അറേയിലേക്ക് താഴെ കാണുന്നത് പോലെ പ്രാരംഭ വിലകൾ നൽകാവുന്നതാണ്.

```
char my_name[10] = { 'N','i','k','e','t','h'};
```

ചിത്രം 9.1ൽ മേൽ സൂചിപ്പിച്ച കാരക്റ്റർ അറേയുടെ മെമ്മറി നീക്കിവെയ്പ് ചിത്രീകരിച്ചിട്ടുണ്ട്. സ്ട്രിങ്ങിലെ കാരക്റ്ററുകൾ കോമായുപയോഗിച്ച് വേർതിരിച്ചാണ് ശേഖരിക്കുന്നത് എന്ന് പ്രത്യേകം ശ്രദ്ധിക്കേണ്ടതാണ് . ഇതേ ഡാറ്റ ഇൻപുട്ട് ചെയ്യണമെങ്കിൽ താഴെ പറഞ്ഞിരിക്കുന്ന C++ പ്രസ്താവന ഉപയോഗിക്കാം .

```
for (int i=0;i<6;i++)
    cin>>my_name[i];
```

ഈ കോഡ് പ്രവർത്തിക്കുന്ന സമയത്ത് നാം "Niketh" എന്ന സ്ട്രിങ്ങിനകത്തെ ആറ് കാരക്റ്ററുകൾ ഒന്നിന് പുറകെ ഒന്നായി സ്പേസ് ബാർ, ടാബ് കീ അല്ലെങ്കിൽ എന്റർ കീ എന്നിവയിൽ ഏതെങ്കിലും ഒന്നുപയോഗിച്ച് വേർതിരിച്ച് വേണം ഇൻപുട്ട് ചെയ്യേണ്ടത്. മേൽ സൂചിപ്പിച്ച രണ്ടു രീതിയിലുമുള്ള മെമ്മറി നീക്കിവയ്ക്കലുകൾ താഴെ തന്നിരിക്കുന്ന വിധത്തിലാണ്.



ചിത്രം 9.1 കാരക്റ്റർ അറെയുടെ മെമ്മറി നീക്കിവെയ്പ്

സ്ട്രിങ്ങുകൾ തുടർച്ചയായുള്ള കാരക്റ്ററുകൾ ആയതിനാൽ കാരക്റ്റർ അറേയെ സ്ട്രിങ്ങുകൾ ശേഖരിക്കുന്നതിന് ഉപയോഗിക്കാവുന്നതാണ്. എന്നിരുന്നാലും ഒരു സ്ട്രിങ് നേരിട്ട് ഇൻപുട്ട് ചെയ്യുന്നതായി നമുക്ക് തോന്നുകയേ ഇല്ല എന്നത് ഒരു വസ്തുതയാണ്. പകരം നാം ഒന്നിന് പുറകെ ഒന്നായി കാരക്റ്ററുകൾ ഇൻപുട്ട് ചെയ്ത് അതിനെ ഒരു സ്ട്രിങ് ആക്കി മാറ്റുകയാണ് ചെയ്യേണ്ടത്.

C++ ൽ കാരക്റ്റർ അറേകൾക്ക് ചില പ്രത്യേക സവിശേഷതകൾ ഉണ്ട്. ഒരിക്കൽ ഒരു കാരക്റ്റർ അറെ പ്രഖ്യാപിച്ചാൽ, അറെയുടെ പേര് സ്ട്രിങ് ഡാറ്റ സൂക്ഷിക്കാനുള്ള സാധാരണ വേരിയബിളായിത്തന്നെ പരിഗണിക്കപ്പെടുന്നു. അതുകൊണ്ടു തന്നെ കാരക്റ്റർ അറേയുടെ പേര് സ്ട്രിങ് വേരിയബിളിന് സമാനമാണ് എന്ന് പറയാം. അതിനാൽ നിങ്ങളുടെ പേര് my_name (അറെയുടെ പേര്) ൽ താഴെ കൊടുത്തിട്ടുള്ള പ്രസ്താവന ഉപയോഗിച്ച് സംഭരിക്കാവുന്നതാണ്.

```
cin>>my_name;
```

മറ്റുള്ള ഡാറ്റ ഇനങ്ങളുടെ കാര്യത്തിൽ മേൽ സൂചിപ്പിക്കപ്പെട്ട പ്രയോഗം തെറ്റാണെന്ന് പ്രത്യേകം ശ്രദ്ധിക്കേണ്ടതാണ്. ഇനി നമുക്ക് ഒരു സ്ട്രിങ് ഇൻപുട്ട് ചെയ്തു പ്രദർശിപ്പിക്കുന്നതിനുള്ള പ്രോഗ്രാം പൂർത്തിയാക്കാം. പ്രോഗ്രാം 9.1 ൽ പറഞ്ഞിരിക്കുന്നത് പോലെ ഇത് ചെയ്യാവുന്നതാണ് .

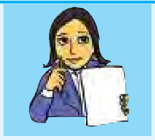
പ്രോഗ്രാം 9.1: ഒരു സ്ട്രിങ് ഇൻപുട്ട് ചെയ്ത് പ്രദർശിപ്പിക്കുക.

```
     #include<iostream>
using namespace std;
int main()
{
     char my_name[10];
     cout << "Enter your name: ";
     cin >> my_name;
     cout << "Hello " << my_name;
}
```

ഈ പ്രോഗ്രാം പ്രവർത്തിപ്പിക്കുമ്പോൾ താഴെ കാണുന്നവിധം ഔട്ട്പുട്ട് ലഭിക്കുന്നതാണ്.

```
Enter your name: Niketh
Hello Niketh
```

പ്രത്യേകം ശ്രദ്ധിക്കേണ്ടത് ഇവിടെ സ്ട്രിങ് കോൺസ്റ്റന്റ് "Hello" അല്ല "Hello " ആണ് എന്നുള്ളത്. ('o' എന്ന അക്ഷരത്തിനു ശേഷം ഒരു സ്പേസ് നൽകിയിട്ടുണ്ട്).



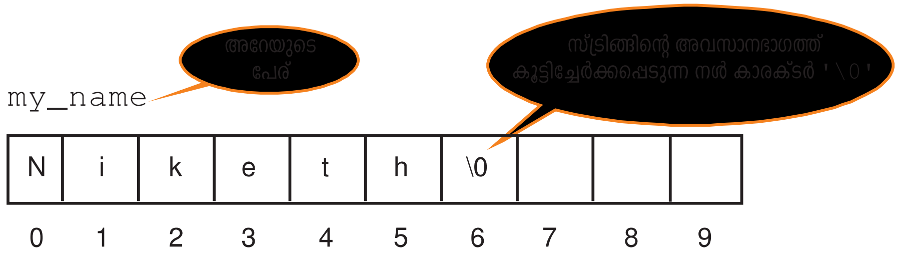
നമുക്ക് ചെയ്യാം

പ്രോഗ്രാം 9.1 പ്രവർത്തിപ്പിച്ച് നിങ്ങളുടെ പേരിന്റെ കൂടെ ഇനിഷ്യലും ഇൻപുട്ട് ചെയ്ത് ഔട്ട്പുട്ട് ശരിയോ തെറ്റോ എന്ന് പരിശോധിക്കുക. പേരിൽ 10 കാരക്റ്ററുകളിലും കൂടുതൽ ഉണ്ടെങ്കിൽ അറെയുടെ വലിപ്പം ആവശ്യത്തിനനുസരിച്ച് വർദ്ധിപ്പിക്കുക .

9.2 സ്ട്രിങ്ങിനു വേണ്ടിയുള്ള മെമ്മറി നീക്കിവെയ്പ് (Memory allocation for strings)

ഒരു അറെയിലുള്ള കാരക്റ്ററുകൾക്ക് എങ്ങനെയാണ് മെമ്മറി അനുവദിക്കുന്നതെന്നു നാം കണ്ടു കഴിഞ്ഞു. ചിത്രം 9.1 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ മെമ്മറി ആവശ്യകത കണക്കാക്കുന്നത് ഇൻപുട്ട് ചെയ്ത കാരക്റ്ററുകളുടെ എണ്ണമനുസരിച്ചാണ്. എന്നാൽ ഒരു

കാരക്റ്റർ അറെയിൽ സ്ട്രിങ് ഇൻപുട്ട് ചെയ്യുമ്പോൾ ചിത്രം മറ്റൊന്നാകുന്നു. നമ്മൾ പ്രോഗ്രാം 9.1 പ്രവർത്തിപ്പിച്ച് "Niketh" എന്ന സ്ട്രിങ് ഇൻപുട്ട് ചെയ്താൽ മെമ്മറി നീക്കിവെയ്പ് താഴെ കാണുന്ന വിധമായിരിക്കും.



ചിത്ര 9.2 : കാരക്റ്റർ അറേയുടെ മെമ്മറി നീക്കിവെയ്പ്.

ഇവിടെ നൾ കാരക്റ്റർ ('\0') സ്ട്രിങ്ങിന്റെ അവസാനഭാഗത്ത് കൂട്ടിച്ചേർക്കപ്പെടുന്നു. ഇത് കാരക്റ്റർ സ്ട്രിങ്ങിന്റെ ടെർമിനേറ്റർ ആയി ഉപയോഗിക്കുന്നു. അതിനാൽ ഒരു സ്ട്രിങ് സംഭരിക്കാനാവശ്യമായ മെമ്മറി എന്നത് സ്ട്രിങ്ങിലെ ആകെ കാരക്റ്ററുകളുടെ എണ്ണവും നൾ കാരക്റ്ററിനു വേണ്ട ഒരു ബൈറ്റും ചേർന്നതാണ്. മേല്പറഞ്ഞ "Niketh" എന്ന സ്ട്രിങ് ശേഖരിക്കുവാൻ ഏഴ് ബൈറ്റ് ആവശ്യമാണ്. (അതായത് 6 കാരക്റ്ററുകൾക്കുള്ള 6 ബൈറ്റ് + നൾ കാരക്റ്ററിനുള്ള 1 ബൈറ്റ്).

താഴെ കാണിച്ചിരിക്കുന്നവിധത്തിൽ നമുക്ക് കാരക്റ്റർ അറെയ്ക്ക് പ്രാരംഭവില നൽകാം.

```
char my_name[10] = "Niketh";
char str[] ="Hello world";
```

ആദ്യത്തെ പ്രസ്താവനയിൽ പത്ത് മെമ്മറി സ്ഥാനങ്ങൾ നീക്കി വെക്കുകയും അതിൽ പ്രാരംഭ വിലയും നൾ കാരക്റ്ററും സംഭരിക്കുകയും ചെയ്യുന്നു. ഇവിടെ അവസാന മൂന്ന് ബൈറ്റുകൾ ഉപയോഗിക്കുന്നില്ല. എന്നാൽ രണ്ടാമത്തെ സ്റ്റേറ്റ്മെന്റിൽ അറെയുടെ വലിപ്പം ഉൾപ്പെടുത്തിയിട്ടില്ല. അതുകൊണ്ട് 11 ബൈറ്റ് സ്ട്രിങ്ങിനും 1 ബൈറ്റ് '\0' നും അടക്കം ആകെ 12 ബൈറ്റ് നീക്കിവെയ്ക്കപ്പെടുന്നു .

9.3 സ്ട്രിങ്ങിനു മേലുള്ള ഇൻപുട്ട്/ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങൾ (Input/Output operations on strings)

പ്രോഗ്രാം 9.1ൽ സ്ട്രിങ് ഡാറ്റ ഇൻപുട്ട്/ഔട്ട്പുട്ട് ചെയ്യുന്നതിനുള്ള പ്രസ്താവനകൾ ഉൾപെടുത്തിയിട്ടുണ്ട്. അറെയുടെ വലിപ്പം 20 ആക്കി പ്രഖ്യാപന പ്രസ്താവനയിൽ ഒരു ചെറിയ മാറ്റം വരുത്തുക. "Maya Mohan" എന്ന പേര് ഇൻപുട്ട് ചെയ്ത് പ്രോഗ്രാം പ്രവർത്തിപ്പിക്കുകയാണെങ്കിൽ താഴെ കാണുന്ന വിധത്തിലുള്ള ഔട്ട്പുട്ട് ലഭിക്കുന്നതാണ്.

```
Enter your name: Maya Mohan
Hello Maya
```

സ്ട്രിങ് ശേഖരിക്കുന്നതിന് ആവശ്യമായ വലിപ്പം അറെയ്ക് ഉണ്ടെങ്കിലും നമുക്ക് ഔട്ട്പുട്ടായി "Maya" എന്ന് മാത്രമാണ് ലഭിക്കുന്നത്. ഇതെന്തുകൊണ്ട് സംഭവിച്ചു?

നമുക്ക് `cin>>my_name;` എന്ന പ്രസ്താവന സൂക്ഷ്മമായൊന്നു പരിശോധിക്കാം. ഒരു ഡാറ്റ ഇനത്തെ മാത്രമേ ഈ പ്രസ്താവന ഉപയോഗിച്ചു ഇൻപുട്ട് ചെയ്യാൻ കഴിയൂ എന്ന്

നമുക്കറിയാം. ഒരു ഡാറ്റയെ മറ്റൊന്നിൽ നിന്ന് വേർതിരിക്കുവാൻ ഉപയോഗിക്കുന്നതാണ് വൈറ്റ് സ്പേസ്. അതുകൊണ്ട് "Maya Mohan" എന്നത് രണ്ട് ഡാറ്റയായി പരിഗണിക്കപ്പെടുന്നു. (Maya, Mohan എന്നിവയ്ക്കിടയ്ക്ക് വൈറ്റ് സ്പേസ് ഉള്ളതുകൊണ്ട്). my_nameന് മുമ്പ് ഒരു ഇൻപുട്ട് ഓപ്പറേറ്റർ (>>) മാത്രമേയുള്ളൂ. അതിനാൽ ആദ്യത്തെ ഡാറ്റയായ "Maya" മാത്രം സംഭരിക്കപ്പെടുന്നു. അതിന് ശേഷമുള്ള വൈറ്റ് സ്പേസ് ഡിലിമിറ്റർ ആയി വർത്തിക്കുകയും ചെയ്യുന്നു. അതിനാൽ ഈ പ്രസ്താവന സംവിധാനം ഉപയോഗിച്ച് വൈറ്റ് സ്പേസ് അടങ്ങിയ സ്ട്രിങ്ങുകൾ മുഴുവനായും ഇൻപുട്ട് ചെയ്യുവാൻ കഴിയുകയില്ല. ഇതിനു പരിഹാരമായി gets() എന്ന ഫങ്ഷൻ ഉപയോഗിക്കാവുന്നതാണ്. സ്റ്റാൻഡേർഡ് ഇൻപുട്ട് ഉപകരണങ്ങളിൽ (keyboard) നിന്ന് വൈറ്റ് സ്പേസ് അടങ്ങിയ സ്ട്രിങ്ങുകളെ സ്വീകരിക്കുകയും അതിനെ ഒരു കാരക്റ്റർ അറെയിൽ സംഭരിക്കുന്നതിനുമുള്ള കൺസോൾ ഇൻപുട്ട് ഫങ്ഷനാണ് gets(). ഈ ഫങ്ഷനിലേക്ക് സ്ട്രിങ് വേരിയബിൾ (കാരക്റ്റർ അറെയുടെ പേര്) താഴെ കാണുന്നവിധത്തിൽ നൽകാവുന്നതാണ്.

```
gets(character_array_name);
```

ഈ ഫങ്ഷൻ ഉപയോഗിക്കുമ്പോൾ cstdio(stdio.h എന്നത് Turbo C++ൽ) എന്ന ലൈബ്രറി ഹെഡർ ഫയൽ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ടതാണ്. പ്രോഗ്രാം 9.1 ൽ include <cstdio> ഉൾപ്പെടുത്തുകയും കൂടാതെ cin>>my_name; എന്ന പ്രസ്താവനയ്ക്ക് പകരം gets(my_name); ഉപയോഗിച്ച് പ്രോഗ്രാം വീണ്ടും പ്രവർത്തിപ്പിച്ചാൽ താഴെ കാണുന്ന ഔട്ട്പുട്ട് ലഭിക്കുന്നതാണ്.

```
Enter your name : Maya Mohan
Hello Maya Mohan
```

ഇപ്പോൾ നാം ഇൻപുട്ട് ചെയ്ത മുഴുവൻ സ്ട്രിങ്ങും ഔട്ട്പുട്ട് ആയി കാണപ്പെടുന്നുണ്ട്. ഇനി നമുക്ക് gets() ഫങ്ഷനും cin ഉം തമ്മിലുള്ള വ്യത്യാസം എന്താണെന്നു നോക്കാം.

സ്ട്രിങ്ങിന്റെ ഇൻപുട്ട്/ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങളിൽ സബ്സ്ക്രിപ്റ്റഡ് വേരിയബിൾ എന്ന ആശയം ഉപയോഗിക്കുന്നില്ലെങ്കിലും, അറെയിലെ ഏതൊരു അംഗത്തെയും അറെയുടെ പേരും സബ്സ്ക്രിപ്റ്റും ഉപയോഗിച്ചു വേർതിരിച്ചുപയോഗിക്കാവുന്നതാണ്. സ്ട്രിങ്ങിലെ ആദ്യത്തെ കാരക്റ്ററിനെ ഉപയോഗിക്കണമെങ്കിൽ my_name[0] എന്നും, അഞ്ചാമത്തെ കാരക്റ്റർ എടുത്തുപയോഗിക്കണമെങ്കിൽ my_name[4] എന്നിങ്ങനെ എന്നും പ്രയോഗിക്കാവുന്നതാണ്. നൾ കാരക്റ്ററും ('\0') നമുക്ക് സബ്സ്ക്രിപ്റ്റ് ഉപയോഗിച്ചു തെരഞ്ഞെടുക്കാം. താഴെ കൊടുത്തിട്ടുള്ള പ്രോഗ്രാം ഈ ആശയം വ്യക്തമാക്കുന്നതാണ്.

പ്രോഗ്രാം 9.2 തന്നിരിക്കുന്ന സ്ട്രിങ്ങിലെ സ്വരാക്ഷരങ്ങളുടെ (Vowels) എണ്ണം കണ്ടുപിടിക്കുക

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
 {
  char str[20];
```

gets() ഫങ്ഷന് വേണ്ടിയുള്ള ഹെഡർ ഫയൽ

```
int vow=0;
cout<<"Enter a string: ";
gets(str);
for(int i=0; str[i]!='\0'; i++)
   switch(str[i])
   {
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u': vow++;
   }
cout<<"No. of vowels in the string "<<str<<" is "<<vow;
return 0;
}
```

നൾ കാരക്ടർ എത്തുന്നതുവരെ തുടർന്നു കൊണ്ടിരിക്കുന്നു.

അറേയിലെ ഓരോ കാരക്ടറും കാരക്ടർ കോൺസ്റ്റന്റുമായി താരതമ്യം ചെയ്യുന്നു

"Hello guys" എന്ന സ്ട്രിങ് ഇൻപുട്ട് ചെയ്ത് പ്രോഗ്രാം 9.2 പ്രവർത്തിപ്പിക്കുകയാണെങ്കിൽ ചുവടെ കൊടുത്തിരിക്കുന്ന ഔട്ട്പുട്ട് കാണാവുന്നതാണ് .

```
Enter a string : Hello guys
No.of vowels in the string Hello guys is 3
```

ഈ പ്രോഗ്രാം പ്രവർത്തിച്ച് ഫലം ലഭ്യമാകുന്നത് എങ്ങനെയെന്ന് നമുക്ക് വിശകലനം ചെയ്യാം.

- തുടക്കത്തിൽ തന്നെ `gets()` ഫങ്ഷൻ ഉപയോഗിച്ച് `"Hello guys"` എന്ന സ്ട്രിങ് ഇൻപുട്ട് ചെയ്യുന്നു .
- 'i' എന്ന സബ്സ്ക്രിപ്റ്റ് ഉപയോഗിച്ചു സൂചിപ്പിക്കുന്ന അറെയിലെ ഓരോ കാരക്റ്ററും, നൾ കാരക്റ്റർ ('\0') അല്ലാത്തിടത്തോളം ഫോർ ലൂപ്പിന്റെ ചട്ടക്കൂട് തുടർച്ചയായി പ്രവർത്തിച്ചുകൊണ്ടിരിക്കുന്നു. അതായത് നൾ കാരക്റ്റർ എത്തുന്നതുവരെ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ചുകൊണ്ടിരിക്കും.
- ലൂപ്പ് ചട്ടക്കൂടിനകത്ത് ഒരേയൊരു സ്വിച്ച് പ്രസ്താവന (switch statement) മാത്രമേ ഉള്ളൂ. ആദ്യത്തെ നാലു കേസുകളിലും ഒരു പ്രസ്താവന പോലും നൽകിയിട്ടില്ല. അവസാനത്തെ കേസിന് vow എന്ന വേരിയബിളിന്റെ വില ഒന്ന് വർദ്ധിക്കുന്നു (vow++). എല്ലാ കേസുകൾക്കും ഇതാവശ്യമാണെന്നു ഒരു പക്ഷെ നിങ്ങൾ ചിന്തിക്കുന്നുണ്ടാവും. അത് തികച്ചും ശരിയാണ്. എന്നാൽ അങ്ങനെയാണെങ്കിൽ ഓരോ കേസിനും വെവ്വേറെ ബ്രേക്ക് പ്രസ്താവനകൾ ഉപയോഗിക്കേണ്ടതായി വരും. ഈ പ്രോഗ്രാമിൽ എല്ലാ കേസുകളുടെയും പ്രവർത്തനം ഒരേ പോലെയായതിനാലാണ് ഈ രീതിയിലുള്ള പ്രസ്താവന ഉപയോഗിച്ചിരിക്കുന്നത്.
- ഫോർ ലൂപ്പ് തുടർച്ചയായി പ്രവർത്തിക്കുമ്പോൾ ഓരോ കാരക്റ്ററും ഒന്നിന് പുറകെ ഒന്നായി ലഭ്യമാകുന്നു. അവയെ കേസിലെ ഓരോ കാരക്റ്റർ കോൺസ്റ്റന്റുമായി താരതമ്യം ചെയ്യുന്നു. ഏതെങ്കിലും ഒരു തവണ സമാനത കൈവരിച്ചാൽ vow എന്ന വേരിയബിളിന്റെ വില ഒന്ന് കൂടുന്നു (vow ++).

- നൽകിയിട്ടുള്ള ഇൻപുട്ട് സ്ട്രിങ്ങിന്റെ കാര്യത്തിൽ സമാനത കൈവരിക്കുന്നത് " i " യുടെ വില 1, 4, 7 എന്നിങ്ങനെ ആകുമ്പോഴാണ്. അതുകൊണ്ട് തന്നെ vow ന്റെ വില മൂന്നു തവണ ഓരോന്ന് വെച്ച് വർധിക്കുകയും നമുക്ക് ശരിയായ ഉത്തരം ലഭിക്കുകയും ചെയ്യുന്നു.

സ്ട്രിങ്ങുകൾ ഇൻപുട്ട് ചെയ്യുന്നതിനു `gets()` ഫങ്ഷൻ എങ്ങനെ ഉപയോഗിക്കുന്നുവെന്ന് നാം മനസ്സിലാക്കി. അതുപോലെ സ്ട്രിങ് ഔട്ട്പുട്ട് ചെയ്യുന്നതിന് C++ ൽ `puts()` എന്ന ഫങ്ഷൻ ലഭ്യമാണ്. സ്ട്രിങ് ഡാറ്റയെ സ്റ്റാൻഡേർഡ് ഔട്ട്പുട്ട് ഉപകരണ (മോണിറ്റർ) ത്തിൽ പ്രദർശിപ്പിക്കുവാൻ വേണ്ടിയുള്ള കൺസോൾ ഔട്ട്പുട്ട് ഫങ്ഷനാണ് `put()`. ഇതിന്റെ വാക്യഘടന (syntax) താഴെ കൊടുത്തിരിക്കുന്നു .

```
puts(string data);
```

ഈ ഫങ്ഷനിലേക്ക് സ്ട്രിങ് കോൺസ്റ്റന്റ് അഥവാ വേരിയബിൾ (കാരക്റ്റർ അറെയുടെ പേര്) ആണ് നൽകേണ്ടത്. താഴെ കാണുന്ന C++ കോഡ് നിരീക്ഷിക്കുക .

```
char str[10] ="friends";
puts("hello");
puts(str);
```

മേൽ സൂചിപ്പിച്ച കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കാണും വിധത്തിലാണ് .

```
hello
friends
```

കാരക്റ്റർ അറെ str[10] ലെ "friends" എന്ന സ്ട്രിങ് അടുത്ത ലൈനിലാണ് പ്രദർശിപ്പിച്ചിരിക്കുന്നത്. `puts()` ഫങ്ഷനുകൾക്ക് പകരം `cout<<"hello";,cout<<str;` എന്നീ പ്രസ്താവനകൾ ഉപയോഗിക്കുമ്പോഴുള്ള വ്യത്യാസം ശ്രദ്ധിക്കുക. cout ഉപയോഗിക്കുമ്പോൾ സ്ട്രിങ്ങുകൾക്കിടയിൽ ഒരു സ്പേസ് പോലും ഇല്ലാതെ ഔട്ട്പുട്ട് അതേ വരിയിൽ തന്നെ പ്രദർശിപ്പിക്കപ്പെടുന്നു.

നമുക്ക് ചെയ്യാം

പ്രോഗ്രാം 9.2 ൽ "HELLO GUYS" എന്ന ഇൻപുട്ട് നൽകി ഔട്ട്പുട്ട് പ്രവചിക്കുക. പ്രോഗ്രാം പ്രവർത്തിപ്പിച്ചു ഈ ഇൻപുട്ടിന് ശരിയായ ഔട്ട്പുട്ട് ലഭിക്കുന്നുണ്ടോ എന്ന് പരിശോധിക്കുക. ഔട്പുട്ടിലുണ്ടായിരിക്കുന്ന വ്യത്യാസത്തിന് കാരണം കണ്ടെത്തുക. തന്നിരിക്കുന്ന ഏതൊരു സ്ട്രിങ്ങിനും അനുസരിച്ച് കൃത്യമായ ഔട്ട്പുട്ട് ലഭിക്കുന്നതിന് പ്രോഗ്രാമിൽ ആവശ്യമായ മാറ്റങ്ങൾ വരുത്തുക.

9.4 കാരക്റ്റർ ഇൻപുട്ട്/ഔട്ട്പുട്ട് നുവേണ്ടിയുള്ള കൺസോൾ ഫങ്ഷനുകൾ (More console functions)

സ്ട്രിങ്ങിന് മേലുള്ള ഇൻപുട്ട്/ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങൾ നാം ചർച്ച ചെയ്ത് കഴിഞ്ഞു. കാരക്റ്ററുകൾക്ക് മേൽ പ്രയോഗിക്കുവാനുള്ള ചില ഇൻപുട്ട്/ഔട്ട്പുട്ട് ഫങ്ഷനുകളും C++ ൽ ഉൾപ്പെടുത്തിയിട്ടുണ്ട്. ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് **cstdio**

(stdio.h എന്നത് Turbo C++ ൽ) എന്ന ഹെഡർ ഫയൽ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ടത് അത്യാവശ്യമാണ് .

getchar()

ഈ ഫങ്ഷൻ കീബോർഡിലൂടെ ഇൻപുട്ട് ചെയ്ത കാരക്റ്ററിനെ തിരികെ തരികയാണ് ചെയ്യുന്നത്. താഴെ കൊടുത്തിട്ടുള്ള ഉദാഹരണത്തിൽ കാണുന്ന പോലെ ഒരു കാരക്റ്ററിനെ വേരിയബിലിളിലേക്ക് സംഭരിക്കാവുന്നതാണ്.

```
char ch=getchar();
```

സ്ട്രിങ് ഔട്ട്പുട്ടിൽ puts() ഫങ്ഷന്റെ മേന്മകൾ നാം കണ്ടു കഴിഞ്ഞു. ഇനി നമുക്ക് കാരക്റ്റർ ഡാറ്റ ഔട്ട്പുട്ടായി ലഭിക്കുവാനുള്ള ഫങ്ഷനെക്കുറിച്ച് പഠിക്കാം.

putchar()

തന്നിരിക്കുന്ന കാരക്റ്റർ ആർഗ്യുമെന്റിനെ സ്റ്റാൻഡേർഡ് ഔട്ട്പുട്ട് ഉപകരണ (മോണിറ്റർ) ത്തിൽ പ്രദർശിപ്പിക്കുകയാണ് ഈ ഫങ്ഷൻ ചെയ്യുന്നത്. ഇവിടെ ആർഗ്യുമെന്റ് ഒരു കാരക്റ്റർ കോൺസ്റ്റന്റോ അല്ലെങ്കിൽ ഒരു വേരിയബിളോ ആവാം. ആർഗ്യുമെന്റായി ഒരു പൂർണ സംഖ്യയാണ് (integer) നൽകുന്നതെങ്കിൽ അതിനെ ഒരു ASCII വിലയായി പരിഗണിക്കുകയും അതിനുനസൃതമായ കാരക്റ്റർ പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. താഴെ കൊടുത്തിട്ടുള്ള കോഡ് putchar() ഫങ്ഷന്റെ ഉപയോഗം വ്യക്തമാക്കുന്നു.

```
char ch='B';     // വേരിയബിൾ ch നകത്ത് 'B' ശേഖരിക്കപ്പെടുന്നു
putchar(ch);     //'B' സ്ക്രീനിൽ പ്രദർശിപ്പിക്കപ്പെടുന്നു
ptchar('c');     // 'c' സ്ക്രീനിൽ പ്രദർശിപ്പിക്കപ്പെടുന്നു
putchar(97);     // 97 എന്ന ASCII വിലയ്ക്കനുസൃതമായ 'a' സ്ക്രീനിൽ
                 പ്രദർശിപ്പിക്കപ്പെടുന്നു.
```

പോഗ്രാം 9 .3 ഈ ഫങ്ഷനുകളുടെ പ്രവർത്തനം വ്യക്തമാക്കുന്നതാണ്. ഒരു സ്ട്രിങ് ഇൻപുട്ട് ചെയ്ത് ഒരു കാരക്റ്റർ കണ്ടെത്തുവാൻ ഈ പ്രോഗ്രാമിലൂടെ സാധിക്കുന്നു. ഒരു കാരക്റ്റർ എത്ര തവണ ആവർത്തിക്കുന്നുവെന്നു പ്രദർശിപ്പിക്കുകയാണ് ഈ പ്രോഗ്രാം ചെയ്യുന്നത് .

പ്രോഗ്രാം 9.3 തന്നിരിക്കുന്ന കാരക്റ്റർ ഒരു സ്ട്രിങ്ങിനകത്ത് എത്ര തവണ ഉണ്ടെന്നു കൺസോൾ ഫങ്ഷൻ ഉപയോഗിച്ച് കണ്ടെത്തുക

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
   char str[20], ch;
   int i, num=0;
   puts("Enter a string:"); //To print '\n' after the string
```

```
    gets(str);//To accept a string with white spaces
    cout<<"Enter the character to be searched: ";
    ch=getchar(); //To input the character to be searched
    /* A loop to search for the character and count its
       occurrences in the string. Search will be
       terminated when a null character is found   */
    for(i=0; str[i]!='\0'; i++)
       if (str[i]==ch)
          num++;
    cout<<"The string \'"<<str<<"\' uses the character \'";
    putchar(ch);
    cout<<"\' ")<<num<<" times";
    return 0;
}
```

ഇതുവരെ ചർച്ച ചെയ്ത എല്ലാ കൺസോൾ ഫങ്ഷനുകളും പ്രോഗ്രാം 9.3 യിൽ ഉപയോഗിച്ചിട്ടുണ്ട്. ഈ പ്രോഗ്രാമിന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

Enter the string :

I have a dream

Enter the character to be searched : a

The string "I have a dream" uses the character 'a' 3 times

സ്വയം പരിശോധിക്കാം

1. ഒരു സ്ട്രിങ്ങിന്റെ അവസാനം സൂചിപ്പിക്കാൻ മെമ്മറിയിൽ ഉപയോഗിക്കുന്ന കാരക്റ്റർ ഏത്?
2. 'Save earth' എന്ന സ്ട്രിങ്ങ് ശേഖരിക്കുന്നതിനുള്ള വേരിയബിൾ പ്രഖ്യാപന പ്രസ്താവന എഴുതുക?
3. കൺസോൾ ഇൻപുട്ട് / ഔട്ട്പുട്ട് ഉപയോഗിക്കുന്നതിനാവശ്യമായ ഹെഡർ ഫയലിന്റെ പേരെഴുത്തുക?
4. 'Be Positive' എന്ന സ്ട്രിങ്ങ് ശേഖരിക്കുന്നതിനു എത്ര ബൈറ്റുകൾ ആവശ്യമാണ്?
5. `puts ("hello"); cout<<"hello";` എന്നിവ എങ്ങനെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു?

9.5 ഇൻപുട്ട് / ഔട്ട്പുട്ട് പ്രക്രിയകൾക്ക് വേണ്ടിയുള്ള സ്ട്രീം ഫങ്ഷനുകൾ (Stream functions for I/O operations)

കാരക്റ്ററുകളിലും സ്ട്രിങ്ങുകളിലും ഇൻപുട്ട്/ഔട്ട്പുട്ട് പ്രക്രിയകൾ ചെയ്യുവാനുള്ള മറ്റൊരു സൗകര്യം C++ ൽ ലഭ്യമാക്കിയിട്ടുണ്ട്. iostream എന്ന ഹെഡർ ഫയലിൽ ഉൾപ്പെടുത്തിയിട്ടുള്ള ഫങ്ഷനുകളാണിവ. മെമ്മറിക്കും ഒബ്ജക്റ്റുകൾക്കുമിടയിൽ പ്രവഹിക്കുവാൻ ബൈറ്റുകളെ (ഡാറ്റ) (stream of bytes) യെ കൈകാര്യം ചെയ്യുന്നതിനാൽ ഇവയെ പൊതുവെ സ്ട്രീം ഫങ്ഷനുകൾ എന്നാണ് വിളിക്കുന്നത്. C++ൽ കീബോർഡ്, മോണിറ്റർ

എന്നിവയെയാണ് സാധാരണയായി ഒബ്ജക്റ്റുകളായി സൂചിപ്പിച്ചിരിക്കുന്നത്. ഇവയിൽ ഏതാനും ചില ഫങ്ഷനുകൾ നമുക്ക് പരിശോധിക്കാം .

A. ഇൻപുട്ട് ഫങ്ഷനുകൾ

കാരക്റ്റർ /സ്ട്രിങ് ഡാറ്റയെ ഇൻപുട്ട് ചെയ്യുന്നതിന് ഉപയോഗിക്കുന്ന ഫങ്ഷനുകളാണിവ. ഒബ്ജക്റ്റുകൾക്കും മെമ്മറിക്കുമിടയിൽ ബൈറ്റുകളെ പ്രവഹിക്കുവാൻ സഹായിക്കുന്ന ഫങ്ഷനുകളാണ് `get()`, `getline()` എന്നിവ. കീ ബോർഡ് ഉപയോഗിച്ച് ഡാറ്റ ഇൻപുട്ട് ചെയ്യുമ്പോൾ കീബോർഡിനെ സൂചിപ്പിക്കാൻ cin എന്ന ഓബ്ജക്ട് ഉപയോഗിക്കുകയും മേൽപ്പറഞ്ഞ ഫങ്ഷനുകൾ `cin.get()`, `cin.getline()` എന്നീ രീതികളിൽ വിളിക്കുകയോ പ്രയോഗക്ഷമമാക്കുകയോ ചെയ്യുന്നു. ഇവിടെ ഡോട്ട് ഓപ്പറേറ്റർ എന്ന് വിളിക്കുന്ന പീരിയേഡ് (period) ചിഹ്നം (.) ആണ് `cin` എന്ന ഒബ്ജെക്ടിനും ഫങ്ഷനുമിടയിൽ ഉപയോഗിച്ചിരിക്കുന്നത്.

i.get()

കീബോർഡിലൂടെ ഒരു കാരക്റ്ററിനെയോ ഒന്നിലധികം കാരക്റ്ററുകളെയോ സ്വീകരിക്കുവാൻ ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഒരു സ്ട്രിങ്ങിനെ സ്വീകരിക്കുന്നതിന് ഫങ്ഷന്റെ ആർഗ്യുമെന്റായി അറേയുടെ പേരും വലിപ്പവും നൽകേണ്ടതാണ്. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ഈ ഫങ്ഷന്റെ ഉപയോഗം വ്യക്തമാക്കുന്നതാണ്.

```
char ch,str[10];
ch = cin.get(ch);// ഒരു കാരക്റ്റർ സ്വീകരിച്ച് 'ch' ൽ ശേഖരിക്കുന്നു.
cin.get(ch);      // മേൽ സൂചിപ്പിച്ച പ്രസ്താവനയ്ക് സമാനം.
cin.get(str,10); // പരമാവധി 10 കാരക്റ്ററുകളുള്ള സ്ട്രിങ്ങിനെ സ്വീകരിക്കുന്നു.
```

ii. getline()

കീബോർഡിലൂടെ ഒരു സ്ട്രിങ്ങിനെ സ്വീകരിക്കുവാനുള്ള ഫങ്ഷനാണിത്. എന്റർ കീ, കാരക്റ്ററുകളുടെ എണ്ണം അല്ലെങ്കിൽ ഏതെങ്കിലും പ്രത്യേക കാരക്റ്റർ, ഇവയിൽ ഏതെങ്കിലും ഉപയോഗിച്ചാണ് സ്ട്രിങ്ങിന്റെ അവസാനം സൂചിപ്പിക്കുന്നത്. ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നതിനുള്ള രണ്ടുതരം വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
char ch,str[10];
int len;
cin.getline(str,len);      // 2 ആർഗ്യുമെന്റുകൾ സഹിതം.
cin.getline(str,len,ch); // 3 ആർഗ്യുമെന്റുകൾ സഹിതം .
```

ആദ്യത്തേതിൽ `getline()` ഫങ്ഷന് രണ്ട് ആർഗ്യുമെന്റുകളായ കാരക്റ്റർ അറേയും (ഇവിടെ `str`) കൂടാതെ ആകെ എത്ര കാരക്റ്ററുകൾ ശേഖരിക്കാമെന്നു സൂചിപ്പിക്കുന്ന ഇന്റിജർ വിലയും (len) ഉണ്ട്. രണ്ടാമത്തേതിൽ സ്ട്രിങ്ങിന്റെ അവസാനം (Delimiter) സൂചിപ്പിക്കുന്ന കാരക്റ്ററും (`ch` വേരിയബിളിന്റെ വില) കൂടി ആകെ കാരക്റ്ററുകളുടെ എണ്ണത്തിനൊപ്പം നൽകിയിരിക്കുന്നു. സ്ട്രിങ് ഇൻപുട്ട് ചെയ്യുമ്പോൾ ഒന്നുകിൽ കാരക്റ്ററുകൾ മാത്രം (len-1) അല്ലെങ്കിൽ സ്ട്രിങ്ങിന്റെ അവസാനം സൂചിപ്പിക്കുന്ന കാരക്റ്റർ വരെ, ഇവയിലേതാണോ ആദ്യം സംഭവിക്കുന്നത് എന്നതിനെ ആശ്രയിച്ചായിരിക്കും സ്ട്രിങ് സംഭരിക്കപ്പെടുന്നത്.

B. ഔട്ട്പുട്ട് ഫങ്ഷനുകൾ

മെമ്മറിയ്ക്കും ഒബ്ജെക്റ്റിനുമിടയിൽ ഡാറ്റ ബൈറ്റുകൾ തുടർച്ചയായി പ്രവഹിക്കുവാൻ സഹായിക്കുന്ന ഔട്ട്പുട്ട് ഫങ്ഷനുകളാണ് put(), write() എന്നിവ. ഔട്ട്പുട്ട് ലഭിക്കുവാൻ വേണ്ടി മോണിറ്റർ ഉപയോഗിക്കുന്നതിനാൽ `cout` എന്ന ഒബ്ജക്റ്റ് ആണ് ഈ ഫങ്ഷനുകളുടെ കൂടെ ഉപയോഗിക്കുന്നത് .

i. put()

ഒരു കാരക്റ്റർ കോൺസ്റ്റന്റോ അല്ലെങ്കിൽ വേരിയബിളോ ആർഗ്യുമെന്റായി സ്വീകരിച്ചു പ്രദർശിപ്പിക്കുവാൻ ഉപയോഗിക്കുന്ന ഫങ്ഷനാണിത്.

```
char ch='c';
cout.put(ch);  // 'c' പ്രദർശിപ്പിക്കപ്പെടുന്നു.
cout.put('B'); // 'B' പ്രദർശിപ്പിക്കപ്പെടുന്നു.
cout.put(65);  // ASCII വില 65 നു പകരമായി 'A' പ്രദർശിപ്പിക്കപ്പെടുന്നു.
```

ii. write()

ആർഗ്യുമെന്റായി നൽകിയിട്ടുള്ള സ്ട്രിങ്ങിനെ പ്രദർശിപ്പിക്കുവാനാണ് ഇത് ഉപയോഗിക്കുന്നത്. വ്യക്തതയ്ക്ക് വേണ്ടി താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം കാണുക.

```
char str[10] ="hello";
cout.write(str,10);
```

മേൽപ്പറഞ്ഞ കോഡ് പ്രവർത്തിപ്പിക്കുമ്പോൾ "hello" എന്ന സ്ട്രിങ്ങിന് ശേഷം 5 വൈറ്റ് സ്പേസോടു കൂടിയാണ് പ്രദർശിപ്പിക്കപ്പെടുന്നത്. കാരണം രണ്ടാമത്തെ ആർഗ്യുമെന്റിന്റെ വില 10 ഉം കൂടാതെ സ്ട്രിങ്ങിലെ ആകെ കാരക്റ്ററുകളുടെ എണ്ണം 5 ഉം ആയതിനാലാണ്.

പ്രോഗ്രാം 9.4 .സ്ട്രീം ഇൻപുട്ട്/ഔട്ട്പുട്ട് ഫങ്ഷനുകളുടെ പ്രവർത്തനം വിശദമാക്കുന്നതിന്

```
#include <iostream>
#include <cstring> //To use strlen() function
using namespace std;
int main()
{
    char ch, str[20];
    cout<<"Enter a character: ";
    cin.get(ch); //To input a character to the variable ch
    cout<<"Enter a string: ";
    cin.getline(str,10,'.');     //To input the string
    cout<<"Entered character is:\t";
    cout.put(ch);            //To display the character
    cout.write("\nEntered string is:",20);
    cout.write(str,strlen(str));
    return 0;
}
```

പ്രോഗ്രാം 9.4 പ്രവർത്തിപ്പിക്കുമ്പോൾ താഴെ കാണുന്ന തരത്തിലുള്ള ഔട്ട്പുട്ട് ലഭ്യമാവുന്നതാണ്.

```
Enter a character: p
Enter a string: hello world
Entered character is:     p
Entered string is:
hello wo
```

ഈ പ്രോഗ്രാം പ്രവർത്തിപ്പിക്കുമ്പോൾ എന്താണ് സംഭവിക്കുന്നത് എന്ന് നോക്കാം. തുടക്കത്തിൽ തന്നെ get() ഫങ്ഷൻ 'p' എന്ന കാരക്റ്ററിനെ സ്വീകരിക്കുന്നു. തുടർന്ന് getline() ഫങ്ഷൻ ഉപയോഗിച്ച് "hello world" എന്ന സ്ട്രിങ് ഇൻപുട്ട് ചെയ്യുന്നു. അതിന് ശേഷം put() ഫങ്ഷനുപയോഗിച്ച് 'p' എന്ന കാരക്റ്റർ പ്രദർശിപ്പിക്കുന്നു. write() ഫങ്ഷൻ "hello wo" എന്ന് മാത്രമാണ് പ്രദർശിപ്പിക്കുന്നത്. str എന്ന അറേയിൽ ശേഖരിക്കാവുന്ന പാമാവധി കാരക്റ്ററുകളുടെ എണ്ണം 10 ആണെന്ന് ഫങ്ഷനിൽ സൂചിപ്പിച്ചിട്ടുമുണ്ട്. ഒരു ബൈറ്റ് നൾ കാരക്റ്ററിന് ('\0'– സ്ട്രിങ്ങിന്റെ അവസാന കാരക്റ്റർ) വേണ്ടി മാറ്റിവെക്കപ്പെട്ടതിനാൽ സാധാരണ 9 കാരക്റ്ററുകൾ മാത്രമേ ശേഖരിക്കുവാൻ കഴിയുകയുള്ളൂ. എന്നാൽ ഇവിടെ ഔട്ട്പുട്ട് ആയി വൈറ്റ് സ്പേസ് ഉൾപ്പെടെ 8 കാരക്റ്ററുകൾ മാത്രമാണ് കാണപ്പെടുന്നത്. ഇതിനു കാരണം "p" എന്ന കാരക്റ്ററിന് ശേഷം എന്റർ കീ ഉപയോഗിക്കുമ്പോൾ '\n', str എന്ന അറേയുടെ ആദ്യത്തെ അംഗമായി ശേഖരിക്കപ്പെടുന്നതിനാലാണ്. അതുകൊണ്ട് "hello wo" എന്ന സ്ട്രിങ് പുതിയ വരിയിലാണ് പ്രദർശിപ്പിക്കപ്പെടുന്നത്.

ഈ പ്രോഗ്രാമിൽ "hello.world" എന്ന് ഇൻപുട്ട് ചെയ്ത് പ്രവർത്തിപ്പിച്ചാൽ താഴെ കാണുന്ന വിധത്തിലുള്ള ഔട്ട്പുട്ട് ലഭിക്കുന്നതാണ്.

```
Enter a character : a
Enter a string : hello.world
Entered character string is : a
Entered string is :
hello
```

(.) ഡോട്ട് കാരക്റ്റർ ശ്രദ്ധിക്കുക.

ഈ മാറ്റം ഔട്പുട്ടിൽ ഉണ്ടായതിന് കാരണം getline() എന്ന ഫങ്ഷൻ ഡോട്ട് ചിഹ്നത്തിന് (dot operator) മുമ്പുള്ള കാരക്റ്ററുകളെ മാത്രം സ്വീകരിച്ചതിനാലാണ്.

കീബോർഡിലെ കീകൾ ഉപയോഗിച്ച് ഇൻപുട്ട് ചെയ്യുമ്പോൾ ഒരുപാടു കാര്യങ്ങൾ സംഭവിക്കുന്നതിനാൽ ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുമ്പോൾ ജാഗ്രത പുലർത്തേണ്ടതാണ്. അതുകൊണ്ടു തന്നെ നിങ്ങൾ ആഗ്രഹിക്കുന്ന ഔട്ട്പുട്ട് തന്നെ ലഭിക്കണമെന്നില്ല. മറ്റൊരു കാര്യം ശ്രദ്ധിക്കേണ്ടത് write() ഫങ്ഷനകത്ത് strlen() എന്ന ഫങ്ഷൻ ഉപയോഗിച്ചിരി

ക്കുന്നുവെന്നതാണ്. ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നതിനു പകരം നേരിട്ട് 10 അല്ലെങ്കിൽ 20 എന്നീ സംഖ്യകൾ നൽകാവുന്നതാണ്. കൊടുത്തിരിക്കുന്ന സംഖ്യയെക്കാളും കുറവാണ് കാരക്റ്ററുകളുടെ എണ്ണമെങ്കിൽ ഇൻപുട്ട് ചെയ്യപ്പെട്ട സ്ട്രിങ്ങിന്റെ തുടർച്ചയായി ചില ASCII കാരക്റ്ററുകളും ചേർന്നാണ് ഔട്ട്പുട്ട് ലഭിക്കുക. നിങ്ങൾ `strlen()` ഉപയോഗിക്കുമ്പോൾ സ്ട്രിങ്ങിനകത്തെ കാരക്റ്ററുകളുടെ കൃത്യമായ എണ്ണം സൂചിപ്പിക്കപ്പെടുന്നുണ്ട്. ഈ ഫങ്ഷനുകളെ കുറിച്ചുള്ള കൂടുതൽ വിവരങ്ങൾ പത്താമത്തെ അധ്യായത്തിൽ ചർച്ച ചെയ്യാം. string.h എന്ന ഹെഡർ ഫയൽ ഉൾപ്പെടുത്തിയാൽ മാത്രമേ ഈ ഫങ്ഷൻ ഉപയോഗിക്കുവാൻ കഴിയുകയുള്ളു.

നമുക്ക് ചെയ്യാം

താഴെ കൊടുത്തിരിക്കുന്ന പട്ടികയിൽ വിട്ട ഭാഗം പൂരിപ്പിച്ച് സ്ട്രീം ഫങ്ഷനുകളെ താരതമ്യം ചെയ്യുക.

താരതമ്യ സൂചനകൾ	കൺസോൾ ഫങ്ഷനുകൾ	സ്ട്രീം ഫങ്ഷനുകൾ
ആവശ്യമായ ഹെഡർ ഫയലുകൾ	....	....
ഉപയോഗ രീതി	ബ്രാക്കറ്റിനകത്ത് ആവശ്യമായ ഡാറ്റയോ, വേരിയബിളിന്റെ പേരോ ചേർത്ത് ഫങ്ഷന്റെ പേര് സൂചിപ്പിക്കുക	ഒബ്ജക്റ്റിന്റെ തുടർച്ചയായി ഡോട്ട് ഓപ്പറേറ്ററും, ആവശ്യമായ ഡാറ്റയോ വേരിയബിളിന്റെ പേരോ ചേർത്തിട്ടുള്ള ഫങ്ഷൻ ഉപയോഗിക്കുക.
ഉപകരണങ്ങൾ	കീബോർഡോ, മോണിറ്ററോ എന്ന് സൂചിപ്പിക്കപ്പെട്ടില്ലെങ്കിൽ	
ഉദാഹരണങ്ങൾ	....	....

സ്വയം പരിശോധിക്കാം

1. കാരക്റ്റർ ഡാറ്റ ഇൻപുട്ട് ചെയ്യുന്നതിനുള്ള ഒരു സ്ട്രീം ഫങ്ക്ഷന്റെ പേരെഴുതുക?
2. `write()` ഫങ്ഷൻ ഉപയോഗിച്ച് "Smoking is injurious to health"എന്ന സ്ട്രിങ്ങ് പ്രദർശിപ്പിക്കുവാനുള്ള C++ പ്രസ്താവന എഴുതുക?
3. സ്ട്രീം ഇൻപുട്ട്/ഔട്ട്പുട്ട് ഫങ്ക്ഷനുകൾ ഉപയോഗിക്കുന്നതിനാവശ്യമായ ഹെഡർ ഫയലിന്റെ പേരെഴുതുക?
4. `getline()` ഫങ്ഷന്റെ വാക്യഘടന (syntax) എഴുതുക?

നമുക്ക് സംഗ്രഹിക്കാം

സ്ട്രിങ്ങുകളെ കൈകാര്യം ചെയ്യുന്നതിനാണ് C++ പ്രോഗ്രാമുകളിൽ കാരക്റ്റർ അറേ ഉപയോഗിക്കുന്നത്. ഒരു സ്ട്രിങ്ങിനു മെമ്മറി അനുവദിക്കപ്പെടുമ്പോൾ സ്ട്രിങ്ങിന്റെ അവസാന ഭാഗത്താണ് നൾ കാരക്റ്റർ ('\0') കൂട്ടിച്ചേർക്കപ്പെടുന്നത്. സ്ട്രിങ്ങുകൾ ഇൻപുട്ട്/ഔട്ട്പുട്ട് ചെയ്യുന്നതിന് വിവിധ തരം കൺസോൾ ഫങ്ഷനുകൾ ലഭ്യമാണ്. ഇവ cstdio, എന്ന ഹെഡർ ഫയലിലാണ് ഉൾപ്പെട്ടിട്ടുള്ളത്. iostream എന്ന ഹെഡർ ഫയലും സ്ട്രിങ്ങുകളുടെ ഇൻപുട്ട്/ഔട്ട്പുട്ട് പ്രയോഗങ്ങൾക്കുള്ള ചില സ്ട്രീം ഫങ്ഷനുകൾ ലഭ്യമാക്കുന്നുണ്ട് .

പഠന നേട്ടങ്ങൾ

ഈ പാഠഭാഗത്തിന്റെ പൂർത്തീകരണത്തിനു ശേഷം പഠിതാവ്

- സ്ട്രിങ് കൈകാര്യം ചെയ്യുന്നതിന് കാരക്റ്റർ അറെ ഉപയോഗിക്കുന്നതിനുള്ള ശേഷി ആർജിക്കുന്നു.
- കാരക്റ്റർ, സ്ട്രിങ് എന്നിവ ഇൻപുട്ട്/ഔട്ട്പുട്ട് ചെയ്യുന്നതിനുള്ള വിവിധ തരം ബിൽറ്റ് ഇൻ ഫങ്ഷനുകൾ (Built in function) ഉപയോഗിക്കുന്നതിനുള്ള കഴിവ് നേടുന്നു.
- കൺസോൾ ഫങ്ഷനുകളും സ്ട്രീം ഫങ്ഷനുകളും താരതമ്യം ചെയ്യാനുള്ള പ്രാവീണ്യം നേടുന്നു.

ലാബ് പ്രവർത്തനങ്ങൾ

1 ഒരു സ്ട്രിങ് ഇൻപുട്ട് ചെയ്ത് അതിലെ വലിയ അക്ഷരങ്ങൾ, ചെറിയ അക്ഷരങ്ങൾ, സംഖ്യകൾ, പ്രത്യേക കാരക്റ്ററുകൾ, വൈറ്റ് സ്പേസുകൾ എന്നിവയുടെ എണ്ണം കണ്ടുപിടിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.

2. ഒരു വാചകത്തിലെ വാക്കുകളുടെ എണ്ണം കണ്ടെത്തുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.

3. ഒരു സ്ട്രിങ് ഇൻപുട്ട് ചെയ്ത് അതിലെ എല്ലാ ചെറിയ സ്വരാക്ഷരങ്ങളും (vowels) വലിയ സ്വരാക്ഷരങ്ങളാക്കി മാറ്റുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.

4. തന്നിരിക്കുന്ന സ്ട്രിങ്ങിനെ തിരിച്ചെഴുതുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക. ഉദാഹരണത്തിന് "AND" ആണ് ഇൻപുട്ട് ചെയ്തതെങ്കിൽ ഔട്ട്പുട്ടായി "DNA" എന്ന് ലഭിക്കണം.

5. ഇൻപുട്ട് ചെയ്ത വാക്ക് ഉപയോഗിച്ച് (ഉദാഹരണത്തിന് COMPUTER) താഴെ കാണുന്നവിധം ഒരു ത്രികോണം നിർമ്മിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക.

```
C
C   O
C   O   M
C   O   M   P
C   O   M   P   U
C   O   M   P   U   T
C   O   M   P   U   T   E
C   O   M   P   U   T   E   R
```

6 തന്നിരിക്കുന്ന വാചകത്തിലെ ഓരോ വാക്കിന്റെയും ആദ്യ കാരക്റ്റർ മാത്രം പ്രദർശിപ്പിക്കുവാനുള്ള പ്രോഗ്രാം എഴുതുക. ഇവിടെ കൺസോൾ ഇൻപുട്ട്/ഔട്ട്പുട്ട് ഫങ്ഷനുകൾ മാത്രമേ ഉപയോഗിക്കാവൂ. ഉദാഹരണത്തിന് "Save Water Save Nature" എന്ന സ്ട്രിങ് ഇൻപുട്ട് ചെയ്താൽ ഔട്ട്പുട്ട് "SWSN" എന്നായിരിക്കണം.

7. ഒരു സ്ട്രിങ് പാലിൻഡ്രോം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക. (ഒരു സ്ട്രിങും അതിന്റെ തിരിച്ചെഴുതിയ രൂപവും ഒരേ പോലെയാണെങ്കിൽ ആ സ്ട്രിങ് പാലിൻഡ്രോം ആണ്. ഉദാഹരണം "MALAYALAM").

മാതൃക ചോദ്യങ്ങൾ

ഹ്രസ്വോത്തര ചോദ്യങ്ങൾ

1. putchar(97); എന്ന പ്രസ്താവനയുടെ ഔട്ട്പുട്ട് എന്തായിരിക്കും?
2. കൺസോൾ ഫങ്ഷന്റെയും സ്ട്രിങ് ഫങ്ഷന്റെയും വ്യത്യാസം എഴുതുക.
3. get() ഫങ്ഷനുപയോഗിച്ച് "computer" എന്ന സ്ട്രിങ് ഇൻപുട്ട് ചെയ്യുവാനുള്ള C++ പ്രസ്താവന എഴുതുക.
4. താഴെ കൊടുത്തിട്ടുള്ള കോഡിന്റെ ഔട്ട്പുട്ട് എഴുതുക.

```
puts("hello");
puts("friends");
```

ലഘു ഉപന്യാസം

1. താഴെ കൊടുത്തിരിക്കുന്ന C++ കോഡിന്റെ ഔട്ട്പുട്ട് എഴുതുക

```
char str[] = "Program";
for(int i=0; str[i]!='\0'; ++)
    {
    putchar(str[i]);
    putchar('_');
    }
```

2. താഴെ കൊടുത്തിരിക്കുന്ന C++ പ്രോഗ്രാമിലെ തെറ്റുകൾ കണ്ടെത്തി കാരണം വ്യക്തമാക്കുക .

```
 #include<iostream.h>
using namespace std;
int main()
     {
     char ch, str[10];
     write("Enter a character ");
     ch=getchar();
     puts("Enter a string");
     cin.getline(str);
     cout<<"The data entered are " <<ch;
     putchar(str);
     }
```

3. താഴെ കൊടുത്തിട്ടുള്ള ഫങ്ഷനുകൾ നിരീക്ഷിക്കുക. അവ സാധുവാണെങ്കിൽ, പ്രവർത്തിക്കുമ്പോൾ എന്ത് സംഭവിക്കുന്നുവെന്ന് വിവരിക്കുക. അവ അസാധുവാണെങ്കിൽ കാരണം എഴുതുക (വേരിയബിൾ സാധുവാണെന്ന് സങ്കൽപ്പിക്കുക).

```
a) getchar(ch);           b)  gets(str[5]);
c) putchar("hello");      d)  cin.getline(name,20,'.');
e) cout.write("hello world",10);
```

4. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ വായിച്ച് ചോദ്യങ്ങൾക്ക് ഉത്തരമെഴുതുക.

```
char name[20];
cin>>name;
cout<<name;
```

- ഇൻപുട്ട് സ്ട്രിങ്ങ് "Sachin Tendulkar" എന്നാണെങ്കിൽ ഔട്ട്പുട്ട് എന്തായിരിക്കും? ന്യായീകരിക്കുക.
- തന്നിരിക്കുന്ന ഇൻപുട്ട് സ്ട്രിങ് തന്നെ ഔട്ട്പുട്ടായി ലഭിക്കുന്നതിന് പ്രസ്താവനയിൽ ആവശ്യമായ മാറ്റം വരുത്തുക .

ഉപന്യാസം

1. കൺസോൾ ഇൻപുട്ട്/ഔട്ട്പുട്ട് ഫങ്ഷനുകൾ ഉദാഹരണസഹിതം വിവരിക്കുക.
2. സ്ട്രീം ഇൻപുട്ട്/ഔട്ട്പുട്ട് ഫങ്ഷനുകൾ ഉദാഹരണസഹിതം വിവരിക്കുക.

10

ഫങ്ഷനുകൾ

പ്രധാന ആശയങ്ങൾ

- മോഡുലാർ പ്രോഗ്രാമിങ്ങിന്റെ ആശയം
- C++ ലെ ഫങ്ഷനുകൾ
- മുൻനിർവചിത ഫങ്ഷനുകൾ
  - സ്ട്രിങ്ങ് ഫങ്ഷനുകൾ
  - ഗണിത ഫങ്ഷനുകൾ
  - ക്യാരക്ടർ ഫങ്ഷനുകൾ
  - I/O കൺവേർഷൻ കൈകാര്യം ഫങ്ഷനുകൾ
- ഉപയോക്തൃ നിർമിത ഫങ്ഷനുകൾ
  - ഉപയോക്തൃ നിർമിത ഫങ്ഷനുകൾ നിർമിക്കുന്നു.
  - ഫങ്ഷനുകളുടെ പ്രോട്ടോ ടൈപ്പ്
  - ഫങ്ഷനുകളുടെ ആർഗ്യുമെന്റ്
  - ഡിഫാൾട്ട് ആർഗ്യുമെന്റുകളുള്ള ഫങ്ഷനുകൾ
  - ഫങ്ഷനുകൾ വിളിക്കുന്നതിനുള്ള മാർഗ്ഗങ്ങൾ
- വേരിയബിളുകൾ, ഫങ്ഷനുകൾ എന്നിവയുടെ വ്യാപ്തിയും ജീവനവും
- സ്വയം ആവർത്തിക്കുന്ന ഫങ്ഷനുകൾ
- ഹെഡർ ഫയലുകളുടെ നിർമാണം

P5T6F3

കഴിഞ്ഞ അധ്യായങ്ങളിൽ ലളിതമായ ചില പ്രോഗ്രാമുകൾ നാം ചർച്ച ചെയ്തു. എന്നാൽ സങ്കീർണമായ പ്രശ്നങ്ങൾ പരിഹരിക്കുന്നതിന് ആയിരക്കണക്കിന് വരികളുള്ള വലിയ പ്രോഗ്രാമുകൾ ആവശ്യമുണ്ട്. അധ്യായം 4 ൽ ചർച്ച ചെയ്തതുപോലെ സങ്കീർണമായ പ്രശ്നങ്ങൾ ചെറിയ പ്രശ്നങ്ങളായി വിഭജിക്കുകയും അവ ഓരോന്നും പരിഹരിക്കുന്നതിന് വേണ്ട പ്രോഗ്രാമുകൾ എഴുതുകയും ചെയ്യുന്നു. മറ്റൊരു രീതിയിൽ പറഞ്ഞാൽ നാം വലിയ പ്രോഗ്രാമുകളെ ചെറിയ ഉപപ്രോഗ്രാമുകളായി വിഭജിക്കുന്നു. C++ ൽ ഫങ്ഷൻ എന്നത് വലിയ പ്രോഗ്രാമുകളെ ചെറിയ ഉപപ്രോഗ്രാമുകളായി വിഭജിക്കുന്നതിനുള്ള ഒരു മാർഗ്ഗമാണ്. `main()`, `sqrt()`, `gets()`, `getchar()` തുടങ്ങിയ ഫങ്ഷനുകൾ ഇതിനോടകം തന്നെ നാം പഠിച്ചിട്ടുണ്ട് ഇതിൽ `main()` ഫങ്ഷൻ ഒഴികെ ബാക്കിയെല്ലാം പ്രത്യേക ഉദ്യമത്തിനു വേണ്ടി തയാറാക്കിയതും ഉടനെ തന്നെ ഉപയോഗത്തിന് ലഭ്യമാകുന്നവയുമാണ്. അതുകൊണ്ട് ഇത്തരം ഫങ്ഷനുകൾ ബിൽട്ട്-ഇൻ-ഫങ്ഷൻ അല്ലെങ്കിൽ മുൻനിർവചിത ഫങ്ഷനുകൾ എന്ന് അറിയപ്പെടുന്നു. ഇത്തരം ഫങ്ഷനുകൾക്ക് പുറമേ പ്രത്യേക ഉദ്യമത്തിനായി ഫങ്ഷനുകൾ നമുക്ക് നിർവചിക്കാൻ കഴിയും. ഇവയെ യൂസർ ഡിഫൈൻഡ് അഥവാ ഉപയോക്തൃ നിർവചിത ഫങ്ഷൻ എന്ന് വിളിക്കുന്നു. ഈ അധ്യായത്തിൽ മുൻ നിർവചിത ഫങ്ഷനുകളെ കുറിച്ച് നാം വിപുലമായി ചർച്ച ചെയ്യുകയും നമ്മുടെ സ്വന്തം ഫങ്ഷനുകൾ നിർവ്വചിക്കുന്നത് എങ്ങനെ എന്ന് പഠിക്കുകയും ചെയ്യും. ഇതിലേക്ക് പോകുന്നതിന് മുൻപേ മോഡുലാർ പ്രോഗ്രാമിങ് എന്ന പ്രോഗ്രാമിങ് ശൈലിയെക്കുറിച്ച് നമുക്ക് സ്വയം പരിചയപ്പെടാം.

10.1 മോഡുലാർ പ്രോഗ്രാമിങ്ങിന്റെ ആശയം (Concept of Modular programming)

ഒരു സ്കൂളിന് ആവശ്യമായ സോഫ്റ്റ്‌വെയറിന്റെ കാര്യം നമുക്ക് പരിഗണിക്കാം. വ്യത്യസ്ത ഉദ്യമങ്ങൾക്ക് വേണ്ട ധാരാളം പ്രോഗ്രാമുകൾ അടങ്ങിയ വളരെ വലുതും സങ്കീർണ്ണവുമായ ഒരു സോഫ്റ്റ്‌വെയർ ആണിത്. ചിത്രം 10.1 കാണിച്ചിരിക്കുന്നത് പോലെ സങ്കീർണമായ സ്കൂൾ

ചിത്രം 10.1 മോഡ്യുലാർ പ്രോഗ്രാമിങ് ശൈലി

പ്രവർത്തനങ്ങളെ ചെറിയ ഉദ്യമങ്ങളോ മോഡ്യൂളുകളോ ആയി വിഭജിച്ച് സമാന്തരമായി നിർമ്മിച്ചതിന് ശേഷം ഒരുമിച്ച് കൂട്ടി പൂർണ്ണമായ ഒരു സോഫ്റ്റ്‌വെയർ നിർമിക്കുവാൻ കഴിയും. പ്രോഗ്രാമിങ്ങിൽ മുഴുവൻ പ്രശ്നത്തേയും ചെറിയ പ്രശ്നങ്ങളാക്കി വിഭജിച്ച് പ്രത്യേകം പ്രോഗ്രാമുകൾ എഴുതി അവ പരിഹരിക്കുകയും ചെയ്യും. ഈ തരത്തിലുള്ള സമീപനം മോഡുലാർ പ്രോഗ്രാമിങ് എന്നറിയപ്പെടുന്നു. ഓരോ ഉപഉദ്യമത്തേയും ഒരു മോഡ്യൂളായി പരിഗണിക്കുകയും ഓരോ മോഡ്യൂളുകളിലും നാം പ്രോഗ്രാം എഴുതുകയും ചെയ്യുന്നു. വലിയ പ്രോഗ്രാമുകളെ ചെറിയ ഉപപ്രോഗ്രാമാക്കി വിഭജിക്കുന്ന പ്രവർത്തനത്തെ മോഡുലറൈസേഷൻ എന്ന് വിളിക്കുന്നു.

മോഡുലറൈസേഷൻ നടപ്പിലാക്കുന്നതിന് കമ്പ്യൂട്ടർ പ്രോഗ്രാമിങ് ഭാഷകൾക്ക് വിവിധ മാർഗ്ഗങ്ങൾ ഉണ്ട്. ഉപപ്രോഗ്രാമുകളെ (sub program) സാധാരണയായി ഫങ്ഷനുകൾ എന്നാണ് വിളിക്കുന്നത്. C++ ഉം ഫങ്ഷനുകൾ കൊണ്ട് മോഡ്യുലാർ പ്രോഗ്രാമിങ് നടപ്പിലാക്കുന്നു.

മോഡ്യുലാർ പ്രോഗ്രാമിങ്ങിന്റെ മേൻമകൾ

മോഡ്യുലാർ ശൈലിയിൽ ഉള്ള പ്രോഗ്രാമിങ്ങിന് നിരവധി ഗുണങ്ങൾ ഉണ്ട്. ഇത് പ്രോഗ്രാമിന്റെ വലിപ്പവും സങ്കീർണ്ണതയും കുറച്ച് പ്രോഗ്രാം കൂടുതൽ വായനാ സുഖമുള്ളതും വീണ്ടും ഉപയോഗിക്കാൻ സാധിക്കുന്നതും, തെറ്റുകൾ കണ്ടുപിടിച്ച് അവ മാറ്റുന്ന പ്രവർത്തനം എളുപ്പത്തിലാക്കുകയും ചെയ്യുന്നു. ഈ പ്രത്യേകതകൾ വിശദമായി നമുക്ക് ചർച്ച ചെയ്യാം.

പ്രോഗ്രാമിന്റെ വലിപ്പം കുറക്കുന്നു: ചില സന്ദർഭങ്ങളിൽ ഒരു പ്രോഗ്രാമിലെ ചില നിർദേശങ്ങൾ പ്രോഗ്രാമിന്റെ വിവിധ ഭാഗങ്ങളിൽ ആവർത്തിച്ചേക്കാം. $\frac{x^5 + y^7}{\sqrt{x} + \sqrt{y}}$; എന്ന പദപ്രയോഗം പരിഗണിക്കുക. x ന്റെയും y യുടേയും വിലകൾ ഉപയോഗിച്ച് ഈ പദപ്രയോഗത്തിന്റെ വില കണ്ടുപിടിക്കുന്നതിന് താഴെകൊടുത്തിരിക്കുന്ന നിർദേശങ്ങൾ നമുക്ക് ഉപയോഗിക്കണ്ടേതുണ്ട്.

1. x ന്റെ അഞ്ചാമത്തെ വർഗ്ഗം കണ്ടുപിടിക്കുക.
2. y യുടെ ഏഴാമത്തെ വർഗ്ഗം കണ്ടുപിടിക്കുക.
3. ഘട്ടം ഒന്നിലും രണ്ടിലും ലഭിച്ച ഫലങ്ങൾ കൂട്ടുക.
4. x ന്റെ വർഗ്ഗമൂലം കണ്ടുപിടിക്കുക.
5. y യുടെ വർഗ്ഗമൂലം കണ്ടുപിടിക്കുക.
6. ഘട്ടം 4 ലും 5 ലും ലഭിച്ച ഫലങ്ങൾ കൂട്ടുക.
7. ഘട്ടം 3 ൽ ലഭിച്ച ഫലത്തെ ഘട്ടം 6 ൽ ലഭിച്ച ഫലം കൊണ്ട് ഭാഗിക്കുക.

ഘട്ടം 1 ന്റെയും ഘട്ടം 2 ന്റെയും ഉത്തരങ്ങൾ കണ്ടുപിടിക്കുന്നതിന് പ്രത്യേകം ലൂപ്പുകൾ

ആവശ്യമാണെന്ന് നമുക്ക് അറിയാം. ഒരു സംഖ്യയുടെ വർഗ്ഗമൂലം കാണുന്നതിനാവശ്യമായ യുക്തിയുടെ സങ്കീർണ്ണത നിങ്ങൾക്ക് സങ്കൽപ്പിക്കാൻ കഴിയുമോ?

വ്യത്യസ്ത ഇടങ്ങളിൽ വ്യത്യസ്ത ഡാറ്റ പ്രവർത്തിപ്പിക്കുന്നതിന് ഒരേ നിർദേശങ്ങൾ പ്രോഗ്രാമിന് ആവ്യശ്യമാണ് എന്നത് ഇതിൽ നിന്ന് വ്യക്തമാണ്. ആവർത്തിക്കുന്ന ഉദ്യമങ്ങൾ വേർതിരിക്കുന്നതിനും ഇതിന് വേണ്ട നിർദേശങ്ങൾ എഴുതാനും മോഡുലാർ സമീപനം സഹായിക്കുന്നു. ഇത്തരം നിർദേശങ്ങളുടെ കൂട്ടത്തിന് പേര് നിർദേശിക്കുവാനും ആ പേര് ഉപയോഗിച്ച് ഇവയെ പ്രവർത്തിപ്പിക്കുന്നതിനും നമുക്ക് കഴിയും അങ്ങനെ പ്രോഗ്രാമിന്റെ വലിപ്പം കുറക്കുന്നു.

തെറ്റിനുള്ള സാധ്യത: *പ്രോഗ്രാമിന്റെ വലിപ്പം കുറയുമ്പോൾ സ്വാഭാവികമായും വാക്യഘടനയിലെ തെറ്റുകളും കുറയും.* യുക്തിപരമായ തെറ്റുകൾ ഉണ്ടാവാനുള്ള സാധ്യത പരിമിതപ്പെടും. സങ്കീർണ്ണമായ പ്രശ്നങ്ങൾ പരിഹരിക്കപ്പെടുമ്പോൾ പ്രശ്നത്തിന്റെ എല്ലാവശങ്ങളും നമുക്ക് പരിഗണിക്കേണ്ടതായി വരും. അതിനാൽ പ്രശ്ന പരിഹാരത്തിന് വേണ്ട യുക്തിയും സങ്കീർണമാകും. എന്നാൽ മോഡ്യുലറൈസ് ചെയ്യപ്പെട്ട ഒരു പ്രോഗ്രാമിൽ ഒരു സമയം ഒരു മോഡ്യൂളിൽ മാത്രം നാം ശ്രദ്ധിച്ചാൽ മതിയാകും. ഔട്ട്പുട്ടിൽ എന്തെങ്കിലും തെറ്റ് കണ്ടുപിടിക്കപ്പെട്ടാൽ ബന്ധപ്പെട്ട മോഡ്യൂൾ കണ്ടെത്തി അവിടെ വച്ച് തന്നെ തെറ്റ് തിരുത്തുവാനും നമുക്ക് കഴിയും.

പ്രോഗ്രാമിങ്ങിന്റെ സങ്കീർണ്ണത കുറക്കുന്നു: മുകളിൽ കണ്ടെത്തിയ രണ്ട് ഗുണങ്ങളുടേയും മുഴുവൻ ഫലം പ്രോഗ്രാമിങ്ങിന്റെ സങ്കീർണത കുറയ്ക്കുന്നു എന്നതാകുന്നു. നാം പ്രശ്നത്തെ ചെറിയ ഭാഗങ്ങളായി കൃത്യമായി ഭാഗിക്കുകയാണെങ്കിൽ പ്രശ്ന പരിഹാരത്തിനു വേണ്ട യുക്തിവികസനം ലളിതമാകും. അങ്ങനെ മോഡുലറൈസേഷൻ ഒരു സമയത്ത് ലഘൂകരിക്കപ്പെട്ട ഒരു ഉദ്യമം നമ്മുടെ മനസ്സിലേക്ക് കൊണ്ട് വന്ന് പ്രോഗ്രാമിങ്ങിന്റെ വലിപ്പം കുറക്കുകയും അവയിലുള്ള തെറ്റുകൾ കണ്ടുപിടിച്ച് തിരുത്തുന്ന പ്രവർത്തനം എളുപ്പത്തിലാക്കി പ്രോഗ്രാമിന്റെ സങ്കീർണ്ണത കുറക്കുകയും ചെയ്യുന്നു.

പുനരുപയോഗം മെച്ചപ്പെടുത്തുന്നു: ഓരോ തവണയും പുതിയതായി ഒരു ഫങ്ഷൻ എഴുതുന്നതിന് പകരം ഒരിക്കൽ എഴുതപ്പെട്ട ഒരു ഫങ്ഷൻ മറ്റനവധി പ്രോഗ്രാമുകളിൽ പിന്നീട് ഉപയോഗിക്കച്ചേക്കാം. ഇത് പ്രോഗ്രാം വികസനത്തിന് വേണ്ട സമയം കുറക്കുന്നു.

മോഡ്യുലാർ പ്രോഗ്രാമിന്റെ ന്യൂനതകൾ: മോഡ്യുലാർ പ്രോഗ്രാമിങ്ങിന് പ്രബലമായ മേൻമകൾ ഉണ്ടെങ്കിലും പ്രശ്നത്തെ ശരിയായ രീതിയിൽ വിഭജിക്കുക എന്നത് ഒരു വെല്ലുവിളി ആണ്. ഓരോ ഉപ പ്രശ്നങ്ങളും മറ്റുള്ളവയിൽ നിന്ന് സ്വതന്ത്രമായിരിക്കണം. മോഡ്യുളുകളുടെ പ്രവർത്തന ശ്രേണി തയാറാക്കുമ്പോൾ അങ്ങേയറ്റം ശ്രദ്ധപുലർത്തണം.

10.2 C++ ലെ ഫങ്ഷനുകൾ (Functions in C++)

കോഫി ഉണ്ടാക്കുന്ന യന്ത്രത്തിന്റെ കാര്യം നമുക്ക് പരിഗണിക്കാം. ചിത്രം 10.2 നെ അടിസ്ഥാനമാക്കി അതിന്റെ പ്രവർത്തനങ്ങൾ ചർച്ച ചെയ്യാം. യന്ത്രത്തിലേക്ക് വെള്ളം, പാൽ, പഞ്ചസാര, കാപ്പി പ്പൊടി എന്നിവ കൊടുക്കുന്നു. യന്ത്രം മുൻകൂട്ടി സൂക്ഷിച്ചിട്ടുള്ള നിർദേശങ്ങൾക്ക് അനുസരിച്ച് പ്രവർത്തിപ്പിക്കുകയും ഇവ ഉപയോഗിച്ച് തയ്യാറാക്കുന്ന കോഫി ഒരു കപ്പിൽ ശേഖരിക്കുകയും ചെയ്യുന്നു. അതിനുവേണ്ട നിർദ്ദേശങ്ങൾ താഴെ കൊടുത്തിരിക്കുന്നതു പോലെ ആയിരിക്കും.

ചിത്രം 10. 2 കോഫി ഉണ്ടാക്കുന്ന യന്ത്രത്തിന്റെ പ്രവർത്തനം

1. 60 മില്ലി പാൽ, 120 മില്ലി വെള്ളം 5 ഗ്രാം കാപ്പിപ്പൊടി 20 ഗ്രാം പഞ്ചസാര എന്നിവ യന്ത്രത്തിന്റെ സംഭരണ സ്ഥലത്ത് നിന്ന് എടുക്കുക.
2. മിശ്രിതം തിളപ്പിക്കുക.
3. അവയെ നിർഗമന മാർഗ്ഗത്തിലേക്ക് കൈമാറുക.

ഈ പ്രവർത്തനങ്ങൾ തുടങ്ങുവാൻ യന്ത്രത്തിൽ ഒരു ബട്ടൺ സാധാരണയായി ഉണ്ടാകും. ഈ ബട്ടണിന് make coffee എന്ന പേര് ഉപയോഗിച്ച് നമുക്ക് നാമകരണം ചെയ്യാം. ഈ പ്രവർത്തനം പ്രതീകാത്മമായി താഴെ പറയുന്ന രീതിയിൽ രേഖപ്പെടുത്താൻ കഴിയും. കപ്പ് = കാപ്പി ഉണ്ടാക്കുക (വെള്ളം, പാൽ, പഞ്ചസാര, കാപ്പിപൊടി). ഇവയെല്ലാം പ്രോഗ്രാമിലെ ഫങ്ഷനുകളുമായി നമുക്ക് താരതമ്യം ചെയ്യാം. കാപ്പി ഉണ്ടാക്കുക എന്ന പദം ഫങ്ഷന്റെ പേര് ആയും (വെള്ളം, പാൽ, പഞ്ചസാര, കാപ്പിപൊടി) എന്നിവ ഫങ്ഷന് വേണ്ട പാരാമീറ്ററുകളായും "Coffee" തിരിച്ച് കിട്ടുന്ന ഫലവുമാണ്. ഇത് കപ്പിൽ സംഭരിക്കുന്നു. കപ്പിന് പകരം ഗ്ലാസ്സോ, ടംബ്ലറോ അല്ലെങ്കിൽ മറ്റേതെങ്കിലും പാത്രമോ നമുക്ക് ഉപയോഗിക്കാം. അതുപോലെ തന്നെ ഒരു C++ ഫങ്ഷൻ പാരാമീറ്ററുകൾ സ്വീകരിക്കുകയും അതിൽ പ്രവർത്തിച്ച് ഫലം തിരിച്ച് നൽകുകയും ചെയ്യുന്നു. ചിത്രം 10.3 ഒരു ഫങ്ഷനായി കണക്കാക്കാം. *അതിന് 3,5 2,6 എന്നീ വിലകൾ പാരാമീറ്ററുകളായി സ്വീകരിച്ച അവ തമ്മിൽ കൂട്ടുകയും* തുക C എന്ന വേരിയബിളിൽ ശേഖരിക്കുകയും ചെയ്യുന്നു. അത് താഴെ പറയുന്ന രീതിയിൽ എഴുതാം.

ചിത്രം 10.3: കൂട്ടുന്നതിന് ഉള്ള ഫങ്ഷൻ

C = Add (3, 5, 2, 6)

ഒരു പ്രോഗ്രാമിൽ പ്രശ്നപരിഹാരത്തിന്റെ ഭാഗമായി ഒരു പ്രത്യേക ഉദ്യമം നിർവഹിക്കുന്നതിന് നാമകരണം ചെയ്യപ്പെട്ട ഒരു കൂട്ടം നിർദ്ദേശങ്ങളുടെ ഘടകമാണ് ഫങ്ഷൻ എന്ന് നമുക്ക് പറയാം. എല്ലാ ഫങ്ഷനുകൾക്കും പരാമീറ്ററുകൾ ആവശ്യമാണ് എന്നതും അവയെല്ലാം ചില വില തിരിച്ചു നൽകണമെന്നതും നിർബന്ധമില്ല വിവിധ ഉദ്യമങ്ങൾക്ക് എപ്പോഴും ഉപയോഗിക്കാൻ കഴിയുന്ന ഫങ്ഷനുകളുടെ വിലപ്പെട്ട ശേഖരം C++ നൽകുന്നു. (getch(), pow() sqrt()) തുടങ്ങിയ ഫങ്ഷനുകളിൽ അവ ചെയ്യേണ്ട ഉദ്യമങ്ങൾ നേരത്തെ തന്നെ എഴുതപ്പെട്ടതും തെറ്റുകൾ തിരുത്തി കമ്പയിൽ ചെയ്ത് അവയുടെ നിർവ്വചനങ്ങൾ ഹെഡർ ഫയലുകൾ എന്ന് വിളിക്കുന്ന ഫയലുകളിൽ സൂക്ഷിച്ചിരിക്കുകയും ചെയ്യുന്നു. ഉപയോഗത്തിന് തയ്യാറായ ഇത്തരം ഉപപ്രോഗ്രാമുകളെ **മുൻനിർവ്വചിത ഫങ്ഷനുകൾ** അല്ലെങ്കിൽ **അന്തർനിർമിത ഫങ്ഷനുകൾ** (built-in functions) എന്ന് വിളിക്കുന്നു.

വലിയ പ്രോഗ്രാമുകൾ എഴുതുമ്പോൾ മോഡ്യുലറൈസേഷൻ നടത്തുന്നതിന് ഇത്തരം മുൻനിർവചിത ഫങ്ഷനുകൾ മതിയാവില്ല. ചില പ്രത്യേക ഉദ്യമങ്ങൾ നിർവഹിക്കുന്നതിന് നമ്മുടെ സ്വന്തം ഫങ്ഷനുകൾ നിർവചിക്കുന്നതിന് വേണ്ട സൗകര്യം C++ നൽകുന്നു. നിർവഹിക്കേണ്ട ഉദ്യമം, പേര്, ആവശ്യമുള്ള ഡാറ്റ എന്നിങ്ങനെ ഒരു ഫങ്ഷനുമായി ബന്ധപ്പെട്ട സകലതും ഉപയോക്താവിനാൽ തീരുമാനിക്കപ്പെടുന്നതിനാൽ അവ ഉപയോക്തൃ നിർവചിത ഫങ്ഷനുകൾ (user defined function) എന്ന് അറിയപ്പെടുന്നു. അപ്പോൾ (main ( )) ഫങ്ഷന്റെ ആവശ്യം എന്താണ്? ഉപയോക്താവ് ഉദ്യമം തീരുമാനിക്കുന്നു എന്ന അർത്ഥത്തിൽ ഇവ ഉപയോക്തൃ നിർമിത ഫങ്ഷൻ ആയി പരിഗണിക്കാവുന്നതാണ്. പ്രോഗ്രാമിന്റെ പ്രവർത്തനം main () ഫങ്ഷനിൽ നിന്ന് ആരംഭിക്കുന്നതിനാൽ ഇത് C++

ലെ ഒഴിച്ചു കൂടാൻ സാധിക്കാത്ത ഫങ്ഷനാണ്. main () ഫങ്ഷനില്ലാതെ C++ പ്രോഗ്രാം പ്രവർത്തിക്കില്ല. എല്ലാ ഫങ്ഷനുകളും ഒരു സ്റ്റേറ്റ്മെന്റിൽ നിന്ന് അവ വിളിക്കുമ്പോഴാണ് പ്രവർത്തിക്കുന്നത്.

10.3 മുൻകൂട്ടി നിർവചിച്ച ഫങ്ഷനുകൾ (Predefind Functions)

വിവിധ ഉദ്യമങ്ങൾക്ക് വേണ്ടി C++ ധാരാളം ഫങ്ഷനുകൾ ലഭ്യമാക്കുന്നു. ഏറ്റവും സാധാരണയായി ഉപയോഗിക്കുന്ന ഫങ്ഷനുകൾ മാത്രം നാം ചർച്ചചെയ്യുന്നു. ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുമ്പോൾ അവയിൽ ചിലതിന് നിശ്ചയിച്ചിട്ടുള്ള ഉദ്യമം നിർവഹിക്കുന്നതിന്, ഡാറ്റ ആവശ്യമാണ്. ഫങ്ഷന്റെ പേരിന് ശേഷം പാരന്തസീസ് എന്ന ഒരു ജോഡി ബ്രാക്കറ്റുകൾക്കുള്ളിൽ ഉപയോഗിക്കുന്ന ഈ ഡാറ്റയെ പരാമീറ്ററുകൾ അല്ലെങ്കിൽ ആർഗ്യുമെന്റുകൾ എന്ന് നാം വിളിക്കുന്നു.

ഉദ്യമങ്ങൾ നിർവഹിച്ചതിനു ശേഷം ഫലങ്ങൾ നൽകുന്ന ചില ഫങ്ഷനുകൾ ഉണ്ട്. ഈ ഫലം ഫങ്ഷൻ തിരിച്ചു നൽകുന്ന വില എന്നറിയപ്പെടുന്നു. ചില ഫങ്ഷനുകൾ ഒരു വിലയും തിരിച്ചു തരുന്നില്ല. പകരം അവയുടെ പ്രത്യേക ഉദ്യമം നിർവ്വഹിക്കുന്നു. തുടർന്നുള്ള ഭാഗങ്ങളിൽ സ്ട്രിങ്ങുകൾ കൈകാര്യം ചെയ്യുന്നതിനും ഗണിത പ്രക്രിയകൾ നടത്തുന്നതിനും ക്യാരക്ടർ ഡാറ്റയിൽ പ്രവർത്തിക്കുന്നതിനുമുള്ള ഫങ്ഷനുകൾ നാം ചർച്ചചെയ്യും. ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുമ്പോൾ അതുമായി ബന്ധപ്പെട്ട ഹെഡർ ഫയലുകൾ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ടതാണ്.

10.3.1 സ്ട്രിങ് ഫങ്ഷനുകൾ (String Functions)

സ്ട്രിങ്ങുകൾ കൈകാര്യം ചെയ്യുന്നതിന് ധാരാളം സ്ട്രിങ് ഫങ്ഷനുകൾ C++ ൽ ലഭ്യമാണ്. അധ്യായം 9 ൽ ചർച്ചചെയ്തത് പോലെ C++ൽ സ്ട്രിങ് ഡാറ്റാടൈപ്പ് ഇല്ലാത്തതിനാൽ സ്ട്രിങ് കൈകാര്യം ചെയ്യുന്നതിന് ക്യാരക്ടറുകളുടെ അറെ ആണ് ഉപയോഗിക്കുന്നത് അതുകൊണ്ട് തുടർന്നു വരുന്ന ചർച്ചകളിൽ സ്ട്രിങ് എന്ന പദം വരുമ്പോഴെല്ലാം അത് ഒരു ക്യാരക്ടർ അറെ ആണെന്ന് അനുമാനിക്കുക. സാധാരണയായി ഉപയോഗിക്കുന്ന സ്ട്രിങ് ഫങ്ഷനുകൾ താഴെ കൊടുക്കുന്നവയാണ്. ഈ ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് നമ്മുടെ C++ പ്രോഗ്രാമിൽ `cstring` (ടർബോ C++ൽ `string.h`) എന്ന ഹെഡർഫയൽ ഉൾപ്പെടുത്തേണ്ടതുണ്ട്.

a. `strlen()`

ഒരു സ്ട്രിങ്ങിന്റെ നീളം കണ്ടുപിടിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. സ്ട്രിങ്ങിന്റെ നീളം കൊണ്ട് അർത്ഥമാക്കുന്നത് സ്ട്രിങ്ങിലെ അക്ഷരങ്ങളുടെ എണ്ണമാണ് അതിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
int strlen(string);
```

ഈ ഫങ്ഷൻ ഒരു സ്ട്രിങ് ആർഗ്യുമെന്റായി സ്വീകരിക്കുകയും സ്ട്രിങ്ങിന്റെ നീളം തിരിച്ചു നൽകുകയും ചെയ്യുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഇത് വിവരിക്കുന്നു.

```
char str[] = "Welcome";
int n;
n = strlen(str);
cout << n;
```

ഇവിടെ `strlen()` എന്ന ഫങ്ഷന് ഒരു സ്ട്രിങ് വേരിയബിളിനെ ആർഗ്യുമെന്റായി സ്വീകരിക്കുകയും അതിൽ അടങ്ങിയിരിക്കുന്ന സ്ട്രിങ്ങിലെ ക്യാരക്ടറുകളുടെ എണ്ണം,

അതായത് 7 എന്ന വില n എന്ന വേരിയബിളിലേക്ക് തിരിച്ച് നൽകുന്നു. അതുകൊണ്ട് n എന്ന വേരിയബിളിന്റെ വിലയായി പ്രോഗ്രാം കോഡ് 7 പ്രദർശിപ്പിക്കുന്നു. അറെ ഡിക്ലറേഷൻ താഴെ കൊടുത്തിരിക്കുന്നത് പോലെ ആണെങ്കിലും ഔട്ട്പുട്ട് ഇത് തന്നെ ആയിരിക്കും.

```
char str [10]= "Welcome";
```

ഡിക്ലറേഷനിൽ അറെയുടെ വലിപ്പം കൊടുത്തിരിക്കുന്നത് ശ്രദ്ധിക്കുക. താഴെ കാണിച്ചിരിക്കുന്നത് പോലെ ആർഗ്യുമെന്റ് ഒരു സ്ട്രിങ്ങ് സ്ഥിര വിലയും ആയേക്കാം.

```
n= strlen ("computer");
```

മുകളിലത്തെ നിർദേശം 8 എന്ന വില തിരിച്ച് നൽകുകയും അത് n ൽ സംഭരിക്കുകയും ചെയ്യുന്നു.

b. strcpy()

ഒരു സ്ട്രിങ്ങിനെ മറ്റൊരു സ്ട്രിങ്ങിലേക്ക് പകർത്തുന്നതിന് ഈ ഫങ്ങ്ഷൻ ഉപയോഗിക്കുന്നു. ഇതിന്റെ വാക്യ ഘടന താഴെ നൽകുന്നു.

```
strcpy(string1, string2);
```

ഈ ഫങ്ഷൻ string 2 നെ string 1 ലേക്ക് പകർത്തുന്നു. ഇവിടെ സ്ട്രിങ്ങ് 1 ഉം സ്ട്രിങ്ങ് 2 ഉം ക്യാരക്ടറുകളുടെ അറെ അല്ലെങ്കിൽ സ്ട്രിങ് സറിരാങ്കങ്ങൾ ആണ്. ഫങ്ങ്ഷന്റെ പ്രവർത്തനത്തിന് ആവശ്യമായ ആർഗ്യുമെന്റുകളാണ് ഇവ. താഴെ കൊടുക്കുന്ന കോഡ് ഇവയുടെ പ്രവർത്തനം വിശദമാക്കുന്നു.

```
char s1[10]. s2[10] = "Welcome";
strcpy (s1, s2);
cout << s1;
```

സ്ട്രിങ്ങ് വേരിയബിൾ s1 ൽ അടങ്ങിയിരിക്കുന്ന "Welcome" എന്ന സ്ട്രിങ്ങ് സക്രീനിൽ പ്രദർശിപ്പിക്കപ്പെടും. രണ്ടാമത്തെ ആർഗ്യുമെന്റ് ഒരു സ്ട്രിങ് സറിരാംഗമായി താഴെകൊടുക്കുന്നു.

```
char str [10];
strcpy (str. "Welcome");
```

ഇവിടെ "Welcome" എന്ന സ്ട്രിങ് സ്ഥിരാങ്കം വേരിയബിൾ str ൽ സംഭരിക്കും. str="Welcome" എന്ന അസൈൻമെന്റ് പ്രസ്താവന തെറ്റാണ്. എന്നാൽ ഒരു ക്യാരക്ടർ അറെയിലേക്ക് പ്രഖ്യാപന സമയത്ത്. വില നമുക്ക് നേരിട്ട് നൽകാവുന്നതാണ്.

```
char str [10] = "Welcome" ;
```

c. strcat()

ഒരു സ്ട്രിങ്ങിലേക്ക് മറ്റൊരു സ്ട്രിങ് കൂട്ടിച്ചേർക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. തത്ഫലമായി ലഭിക്കുന്ന സ്ട്രിങ്ങിന്റെ നീളം രണ്ട് സ്ട്രിങ്ങിന്റെയും നീളത്തിന്റെ ആകെ തുക ആകുന്നു. ഫങ്ങ്ഷന്റെ വാക്യ ഘടന താഴെകൊടുത്തിരിക്കുന്നു.

```
strcat(string1, string2);
```

ഇവിടെ string1, string2 എന്നിവ ക്യാരക്ടറുകളുടെ അറെയോ സ്ട്രിങ്ങ് സ്ഥിരാങ്കങ്ങളോ ആയിരിക്കും. string2, string1 ലേക്ക് കൂട്ടിച്ചേർക്കുന്നു അതുകൊണ്ട് ആദ്യത്തെ ആർഗ്യുമെന്റിന്റെ വലിപ്പം രണ്ട് സ്ട്രിങ്ങുകൾ ഒരുമിച്ച് ഉൾക്കൊള്ളാൻ കഴിയുന്നതായിരിക്കണം. ഈ ഫങ്ഷന്റെ പ്രയോഗം കാണിക്കുന്ന ഒരു ഉദാഹരണം നമുക്ക് കാണാം. താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം ശ്രദ്ധിക്കുക.

```
char s1[20] = "Welcome", s2[10] = " to C++";
strcat(s1,s2);
cout << s1;
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന C++ കോഡ് പ്രവർത്തിക്കുമ്പോൾ s1 ന്റെ വിലയായ "Welcome to C++" എന്ന ഔട്ട്പുട്ട് ലഭിക്കും. s2 എന്ന string വൈറ്റ് സ്പേസോഡു കൂടിയാണ് ആരംഭിച്ചിരിക്കുന്നത് എന്നത് ശ്രദ്ധിക്കുക.

d. strcmp()

രണ്ട് സ്ട്രിങ്ങുകൾ തമ്മിൽ താരതമ്യം ചെയ്യുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഈ താരതമ്യത്തിൽ സ്ട്രിങ്ങുകളിലെ ക്യാരക്ടറുകളുടെ അക്ഷരമാലാക്രമം (ASCII വില) പരിഗണിക്കപ്പെടുന്നു. ഫങ്ഷന്റെ വാക്യ ഘടന താഴെ കൊടുത്തിരിക്കുന്നു.

```
strcmp(string1, string2)
```

മൂന്ന് വ്യത്യസ്ത സന്ദർഭങ്ങളിൽ ഫങ്ഷൻ താഴെ കൊടുത്തിരിക്കുന്ന ഏതെങ്കിലും വിലകൾ തിരിച്ച് നൽകുന്നു.

- string1, string2 എന്നിവ ഒരേ പോലെ ആണെങ്കിൽ 0 തിരിച്ചു നൽകും.
- string1 അക്ഷരമാലാക്രമത്തിൽ string2 നേക്കാൾ ചെറുതാണെങ്കിൽ ഒരു നെഗറ്റീവ് വില തിരിച്ച് നൽകും.
- string1 അക്ഷരമാലക്രമത്തിൽ string2 നേക്കാൾ വലുതാണെങ്കിൽ ഒരു പോസിറ്റീവ് വില തിരികെ നൽകും.

താഴകൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഈ ഫങ്ഷന്റെ പ്രവർത്തനം കാണിക്കുന്നു.

```
char s1[]="Deepthi", s2[]="Divya";
int n;
n = strcmp(s1,s2);
if(n==0)
    cout<<"Both the strings are same";
else if(n < 0)
    cout<<"s1 < s2";
else
    cout<<"s1 > s2";
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം പ്രവർത്തിക്കുമ്പോൾ s1 < s2 എന്ന ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കുമെന്ന് വ്യക്തമാണ്.

e. strcmpi()

വലിയ അക്ഷരങ്ങളോ ചെറിയ അക്ഷരങ്ങളോ പരിഗണിക്കാതെ രണ്ട് സ്ട്രിങ്ങുകൾ താരതമ്യം ചെയ്യുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. അതായത് ഈ ഫങ്ഷൻ അപ്പർകേയ്സ് ലോവർകേയ്സ് അക്ഷരങ്ങൾ താരതമ്യത്തിന് ഒരേപോലെ പരിഗണിക്കും. ഈ ഫങ്ഷന്റെ വാക്യഘടനയും പ്രവർത്തനരീതിയും strcmp( ) പോലെ ആണെങ്കിലും ഇത് കെയ്സ് സെൻസിറ്റീവല്ല. ഈ ഫങ്ഷനും strcmp( ) യെ പോലെ വിലകൾ തിരിച്ചു നൽകുന്നു താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം പരിഗണിക്കുക.

```
char s1[]="SANIL", s2[]="sanil";
int n;
```

```
    n = strcmpi(s1,s2);
    if(n==0)
        cout<<"strings are same";
    else if(n < 0)
        cout<<"s1 < s2";
    else
        cout<<"s1 > s2";
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഒരു C++ പ്രോഗ്രാമിൽ പ്രവർത്തിക്കുമ്പോൾ ലഭിക്കുന്ന ഔട്ട്പുട്ട് `"strings are same"` എന്നായിരിക്കും, എന്തുകൊണ്ടെന്നാൽ `strcmpi()` ലോവർകേയ്സ് അക്ഷരങ്ങളേയും അപ്പർകേയ്സ് അക്ഷരങ്ങളേയും ഒരേ പോലെ കരുതുന്നു പ്രോഗ്രാം 10.1. രണ്ട് സ്ട്രിങ്ങുകൾ താരതമ്യം ചെയ്യുകയും കൂട്ടി ച്ചേർക്കുകയും ചെയ്യുന്നു. പുതിയതായി രൂപപ്പെട്ട സ്ട്രിങ്ങിന്റെ നീളവും പ്രദർശിപ്പിക്കുന്നു.

പ്രോഗ്രാം 10.1 രണ്ട് സ്ട്രിങ്ങുകൾ വ്യത്യസ്തമാണെങ്കിൽ ഇവ രണ്ടും കൂട്ടിച്ചേർക്കുന്നതിനും അതിന്റെ നീളം കണ്ടുപിടിക്കുന്നതിനും.

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char s1[20], s2[20], s3[20];
cout<<"Enter two strings: ";
cin>>s1>>s2;
int n=strcmp(s1, s2);
if (n==0)
    cout<<"\nThe input strings are same";
else
{
    cout<<"\nThe input strings are not same";
    strcpy(s3, s1);//Copies the string in s1 into s3
    strcat(s3, s2); //Appends the string in s2 to that in s3
    cout<<"\nString after concatenation is: "<<s3;
    cout<<"\nLength of the new string is: "<<strlen(s3);
}
return 0;
}
```

സ്ട്രിങ് കൈകാര്യം ചെയ്യുന്നതിനുള്ള ഫങ്ഷനുകൾ അടങ്ങിയ ഹെഡർ ഫയൽ

10.3.2 ഗണിത ഫങ്ഷനുകൾ (Mathametical Functions)

ഇനി നമുക്ക് C++ൽ ലഭ്യമായ ഗണിത ഫങ്ഷനുകളെക്കുറിച്ച് ചർച്ച ചെയ്യാം. പ്രോഗ്രാമിൽ ഈ ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് `Cmath` (ടർബോ C++ ൽ `math.h`) എന്ന ഹെഡർ ഫയൽ നാം ഉപയോഗിക്കണം.

a. abs()

ഒരു പൂർണ്ണ സംഖ്യയുടെ (integer) അഖണ്ഡവില കണ്ടുപിടിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഇത് ഒരു പൂർണ്ണ സംഖ്യ (integer) ആർഗ്യുമെന്റ് ആയി എടുത്ത് അതിന്റെ അഖണ്ഡവില തിരിച്ച് നൽകുന്നു. ഇതിന്റെ വാക്യഘടനയാണ്,

```
int abs(int)
```

ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്ന രീതി ഉദാഹരണമായി താഴെ നൽകുന്നു.

```
int n = -25;
cout << abs(n);
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കോഡ് 25 എന്ന് പ്രദർശിപ്പിക്കും. ഒരു ദശാംശ സംഖ്യയുടെ കേവലവില (absolute value) കണ്ടുപിടിക്കണമെങ്കിൽ fabs() എന്ന ഫങ്ഷൻ മുകളിൽ കൊടുത്തിരിക്കുന്നത് പോലെ നമുക്ക് ഉപയോഗിക്കാം. അത് ദശാംശ സംഖ്യ തിരികെ നൽകുന്നു.

b. sqrt()

sqrt() ഒരു സംഖ്യയുടെ വർഗ്ഗമൂലം കണ്ടുപിടിക്കുന്നതിന് ഉപയോഗിക്കുന്നു. ഈ ഫങ്ഷന്റെ ആർഗ്യുമെന്റിന്റെ ഡാറ്റ ഇനം int, float or double എന്നിവ ആകാം. പോസിറ്റീവ് ആർഗ്യുമെന്റിന്റെ വർഗ്ഗമൂലം ഈ ഫങ്ഷൻ തിരിച്ച് നൽകുന്നു. ഇതിന്റെ വാക്യഘടനയാണ്.

```
double sqrt(double)
```

താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം പരിശോധിക്കുക. ഇത് 5 എന്ന വില പ്രദർശിപ്പിക്കും.

```
int n = 25;
float b = sqrt(n);
cout << b;
```

c. pow()

ഒരു സംഖ്യയുടെ പവർ കണ്ടുപിടിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. x,y എന്നീ രണ്ട് ആർഗ്യുമെന്റ്കൾ ഇത് ഉപയോഗിക്കുന്നു x,y എന്നിവയുടെ ഡാറ്റാ ഇനം int, float അല്ലെങ്കിൽ double ആകുന്നു. ഈ ഫങ്ഷൻ x^y യുടെ ഫലം തിരിച്ചു നൽകുന്നു. ഇതിന്റെ വാക്യഘടനയാണ്.

```
double pow(double, int)
```

താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം ഇതിന്റെ പ്രവർത്തനം വിവരിക്കുന്നു.

ഉദാഹരണം:

```
int x = 5, y = 4, z;
z = pow(x, y);
cout << z;
```

pow(x,y) രണ്ട് ആർഗ്യുമെന്റുകൾ ഉപയോഗിക്കുന്നു. മുകളിൽ കൊടുത്തിരിക്കുന്ന ഉദാഹരണത്തിൽ 625 എന്ന് പ്രദർശിപ്പിക്കും.

d. sin()

ഒരു ത്രികോണമിതിയായ ഈ ഫങ്ഷൻ ഒരു കോണിന്റെ സൈൻ (Sine) വില കണ്ടുപിടിക്കുന്നു. ഈ ഫങ്ഷൻ ഡബിൾ തരത്തിലുള്ള ആർഗ്യുമെന്റ് സ്വീകരിച്ച് അതിന്റെ സൈൻ വില തിരിച്ച് നൽകും. ആർഗ്യുമെന്റ് റേഡിയൻ അളവിലാണ് നൽകേണ്ടത്. ഇതിന്റെ

വാക്യഘടന ആണ്

```
double sin(double)
```

∠30^0 (കോൺ 30^0) യുടെ സൈൻ (Sin ഫങ്ഷൻ ഉപയോഗിച്ച്) വില താഴെപറയുന്ന രീതിയിൽ കണ്ടുപിടിക്കാം.

```
float x = 30*3.14/180; //To convert angle into raidant
cout << sin(x);
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കോഡ് .4999770 എന്ന് പ്രദർശിപ്പിക്കും.

e. cos ()

ഈ ഫങ്ഷൻ ഒരു കോണിന്റെ കൊസൈൻ വില കണ്ടുപിടിക്കാൻ ഉപയോഗിക്കുന്നു. ഈ ഫങ്ഷൻ ഒരു ഡബിൾ (double) ഇനത്തിലുള്ള ആർഗ്യുമെന്റ് സ്വീകരിച്ച് കൊസൈൻ വില തിരിച്ച് നൽകുന്നു. ഇവിടെയും കോണിന്റെ അളവ് റേഡിയനിൽ നൽകണം ഇതിന്റെ വാക്യഘടന ആണ്

```
double cos(double)
```

∠ 30^0 യുടെ കൊസൈൻ വില കണ്ടുപിടിക്കുന്നതിന് താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ഉപയോഗിക്കാം.

```
double x = cos(30*3.14/180);
cout << x;
```

ഇവിടെ cos() ന് നൽകിയിരിക്കുന്ന ആർഗ്യുമെന്റ് ഡിഗ്രി അളവിലുള്ള കോണിനെ റേഡിയനിലേക്ക് മാറ്റുന്ന പദപ്രയോഗമാണ്. മുകളിലുള്ള കോഡ് വില 0.866158 എന്ന് പ്രദർശിപ്പിക്കും.

ഒരു ത്രികോണത്തിന്റെ ഒരു കോണും ഒരു വശവും തന്നിരുന്നാൽ ആ മട്ടത്രികോണത്തിന്റെ വശങ്ങളുടെ അളവ് കണ്ടുപിടിക്കുന്നതിനു വേണ്ട പ്രോഗ്രാം 10.2 ൽ കൊടുത്തിരിക്കുന്നു. വിവിധ ഗണിത ഫങ്ഷനുകൾ ഇതിൽ ഉപയോഗിച്ചിരിക്കുന്നു. അതിന് വേണ്ട സൂത്രവാക്യങ്ങൾ താഴെ കൊടുക്കുന്നു.

$$\text{Sin}\,\theta = \frac{\text{എതിർ വശം}}{\text{കർണ്ണം}} \qquad \text{Cos}\,\theta = \frac{\text{സമീപ വശം}}{\text{കർണ്ണം}} \qquad \text{Tan}\,\theta = \frac{\text{എതിർ വശം}}{\text{സമീപ വശം}}$$

(കർണ്ണം)2 = (പാദം)2 + (ലംബം)2

പ്രോഗ്രാം 10.2 ഗണിത ഫങ്ഷനുകൾ ഉപയോഗിച്ച് ഒരു മട്ടത്രികോണത്തിന്റെ വശങ്ങളുടെ നീളം കണ്ടുപിടിക്കുന്നത്

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
const float pi=22.0/7;
int angle, side;
float radians, length, opp_side, adj_side, hyp;
cout<<"Enter the angle in degree: ";
```

(#include<cmath>: ഗണിത ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് ആവശ്യമായ ഹെഡർ ഫയൽ)

```
cin>>angle;
radians=angle*pi/180;
cout <<"\n1. Opposite Side"
     <<"\n2. Adjacent Side"
     <<"\n3. Hypotenuse";
cout <<"\nInput 1, 2 or 3 to specify the side: ";
cin>>side;
cout<<"Enter the length: ";
cin>>length;
switch(side)
{
    case 1: opp_side=length;
            adj_side=opp_side / tan(radians);
            hyp= sqrt(pow(opp_side,2) + pow(adj_side,2));
            break;
    case 2: adj_side=length;
            hyp=adj_side / cos(radians);
            opp_side=sqrt(pow(hyp,2) - pow(adj_side,2));
            break;
    case 3: hyp=length;
            opp_side=hyp * sin(radians);
            adj_side=sqrt(pow(hyp,2) - pow(opp_side,2));
}
cout<<"Angle in degree = "<<angle;
cout<<"\nOpposite Side = "<<opp_side;
cout<<"\nAdjacent Side = "<<adj_side;
cout<<"\nHypotenuse    = "<<hyp;
return 0;
}
```

ഡിഗ്രിയിലുള്ള കോൺ റേഡിയനിലേക്ക് മാറ്റുന്നു

മുകളിലുള്ള പ്രോഗ്രാമിന്റെ മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
Enter the angle in degree: 30
1. Opposite Side
2. Adjacent Side
3. Hypotenuse
Input 1, 2 or 3 to specify the side: 1
Enter the length: 6
Angle in degree = 30
Opposite Side   = 6
Adjacent Side   = 10.38725
Hypotenuse      = 11.995623
```

10.3.3 ക്യാരക്ടർ ഫങ്ഷനുകൾ (Character functions)

ക്യാരക്ടറുകളിൽ വിവിധ പ്രവർത്തനങ്ങൾ നടത്തുവാൻ C++ ൽ ലഭ്യമായ വിവിധ ക്യാരക്ടർ ഫങ്ഷനുകൾ താഴെ കൊടുത്തിരിക്കുന്നു. ഈ ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് cctype (ടർബോ c++ ൽ ctype.h) എന്ന ഹെഡർ ഫയൽ പ്രോഗ്രാമിൽ കൂട്ടിച്ചേർക്കേണ്ടതുണ്ട്.

a. isupper()

തന്നിരിക്കുന്ന ക്യാരക്ടർ വലിയ അക്ഷരത്തിൽ (upper case) ഉള്ളതാണോ, അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഈ ഫങ്ഷന്റെ വാക്യഘടന ആണ്,

```
int isupper(char c)
```

തന്നിരിക്കുന്ന ക്യാരക്ടർ വലിയ അക്ഷരത്തിൽ (upper case) ആണെങ്കിൽ 1 ഉം അല്ലങ്കിൽ പൂജ്യവും ഈ ഫങ്ഷൻ തിരിച്ച് നൽകുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവന പൂജ്യം എന്ന വില n ലേക്ക് നൽകുന്നു.

ഉദാഹരണം:

```
int n = isupper('x');
```

താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ പരിഗണിക്കുക.

```
char c = 'A';
int n = isupper(c);
```

മുകളിലുള്ള പ്രസ്താവനകളുടെ പ്രവർത്തനത്തിന് ശേഷം n ന്റെ വില 1 ആയിരിക്കും എന്തുകൊണ്ടെന്നാൽ തന്നിരിക്കുന്ന ക്യാരക്ടർ വലിയ അക്ഷരത്തിൽ ഉള്ളതാണ്.

b. islower()

തന്നിരിക്കുന്ന ഒരു ക്യാരക്ടർ ചെറിയ അക്ഷരത്തിൽ (lower case) ഉള്ളതാണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഇതിന്റെ വാക്യഘടനയാണ്.

```
int islower(char c)
```

തന്നിരിക്കുന്ന ക്യാരക്ടർ ചെറിയ അക്ഷരത്തിലാണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ പൂജ്യവും ഈ ഫങ്ഷൻ തിരിച്ച് നൽകുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ പ്രവർത്തിച്ചതിന് ശേഷം വേരിയബിൾ nന്റെ വില 1 ആയിരിക്കും. എന്തുകൊണ്ടെന്നാൽ തന്നിരിക്കുന്ന ക്യാരക്ടർ ചെറിയ അക്ഷരത്തിലുള്ളതാണ്.

ഉദാഹരണം:

```
char ch = 'x';
int n = islower(ch);
```

എന്നാൽ താഴെകൊടുത്തിരിക്കുന്ന പ്രസ്താവനയിൽ n ന്റെവില പൂജ്യമായിരിക്കും. കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ അപ്പർ കേയ്സിലുള്ളതാണ്.

```
int n = islower('A');
```

c. isalpha()

തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്ഷരമാണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഇതിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
int isalpha(char c);
```

തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്ഷരമാണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ പൂജ്യവും ഈ ഫങ്ഷൻ തിരിച്ചു നൽകുന്നു.

താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവന n ലേക്ക് പൂജ്യം എന്ന വില നൽകുന്നു. കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്ഷരം അല്ല

```
int n = isalpha('3');
```

എന്നാൽ താഴെകൊടുത്തിരിക്കുന്ന പ്രസ്താവന 1 പ്രദർശിപ്പിക്കുന്നു കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്ഷരമാണ്.

```
cout << isalpha('a');
```

d. isdigit()

തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്കം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഇതിന്റെ വാക്യ ഘടന ആണ്

```
int isdigit(char c);
```

തന്നിരിക്കുന്ന ക്യാരക്ടർ അക്കമാണെങ്കിൽ ഈ ഫങ്ഷൻ 1 ഉം അല്ലെങ്കിൽ പൂജ്യവും തിരിച്ച് നൽകുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ പ്രവർത്തിക്കുമ്പോൾ n എന്ന വേരിയബിളിന്റെ വില 1 ഉം ആയിരിക്കും. കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്കം ആണ്.

```
n = isdigit('3');
```

താഴെ നൽകിയിരിക്കുന്ന പ്രസ്താവനകൾ പ്രവർത്തിക്കുമ്പോൾ n എന്ന വേരിയബിളിന്റെ വില പൂജ്യം ആയിരിക്കും കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്കം അല്ല.

```
char c = 'b';
int n = isdigit(c);
```

e. isalnum()

തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്ഷരമോ അക്കമോ ആണോയെന്ന് ഈ ഫങ്ഷൻ പരിശോധിക്കുന്നു. ഇതിന്റെ വാക്യഘടനയാണ്

```
int isalnum (char c)
```

തന്നിരിക്കുന്ന ക്യാരക്ടർ അക്ഷരമോ അക്കമോ ആണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ പൂജ്യവും തിരിച്ചു നൽകുന്നു.

താഴെ കൊടുത്തിരിക്കുന്ന ഓരോ പ്രസ്താവനയും പ്രവർത്തിക്കുമ്പോൾ 1 തിരിച്ച് നൽകുന്നു.

```
n = isalnum('3');
cout << isalnum('A');
```

എന്നാൽ താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ പ്രവർത്തിപ്പിക്കുമ്പോൾ പൂജ്യം എന്ന വില n ലേക്ക് നൽകുന്നു കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്കമോ അക്ഷരമോ അല്ല.

```
char c = '-';
int n = isalnum(c);
```

f. toupper()

തന്നിരിക്കുന്ന ക്യാരക്ടർ വലിയ അക്ഷരത്തിലേക്ക് മാറ്റുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു ഇതിന്റെ വാക്യഘടന ആണ്

```
char toupper(char c)
```

തന്നിരിക്കുന്ന ക്യാരക്ടറിന്റെ വലിയ അക്ഷരം ഈ ഫങ്ഷൻ തിരിച്ച് നൽകുന്നു. തന്നിരിക്കുന്ന ക്യാരക്ടർ വലിയ അക്ഷരത്തിൽ ഉള്ളതാണെങ്കിൽ ഔട്ട് പുട്ടും അതുതന്നെ ആയിരിക്കും.

താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവന 'A' യെ വേരിയബിൾ c യിലേക്ക് നൽകുന്നു.

```
char c = toupper('a');
```

എന്നാൽ താഴെ തന്നിരിക്കുന്ന പ്രസ്താവനയുടെ ഔട്ട്പുട്ട് 'A' തന്നെ ആയിരിക്കും.

```
cout << (char)toupper('A');
```

ഈ പ്രസ്താവനയിലെ (char) ഉപയോഗിച്ച് ഡാറ്റാ തരം മാറ്റിയിരിക്കുന്നത് ശ്രദ്ധിക്കുക. ഈ രീതി ഉപയോഗിച്ചില്ലെങ്കിൽ ഔട്ട്പുട്ട് A യുടെ ASCII വിലയായ 65 ആയിരിക്കും.

g. tolower()

തന്നിരിക്കുന്ന ക്യാരക്ടറിനെ ചെറിയ അക്ഷരത്തിലേക്ക് മാറ്റുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഇതിന്റെ വാക്യഘടനയാണ്.

```
char tolower(char c)
```

തന്നിരിക്കുന്ന ക്യാരക്ടറിന്റെ ചെറിയ അക്ഷരം ഫങ്ഷൻ തിരിച്ചു നൽകുന്നു. തന്നിരിക്കുന്ന ക്യാരക്ടർ ചെറിയ അക്ഷരത്തിലുള്ളതാണെങ്കിൽ ഔട്ട്പുട്ടും അതുതന്നെ ആയിരിക്കും. ഈ പ്രസ്താവന പരിഗണിക്കുക.

```
c = tolower('A');
```

മുകളിലത്തെ സ്റ്റേറ്റ്മെന്റ് പ്രവർത്തിച്ചതിന്ശേഷം വേരിയബിൾ c യുടെ വില 'a' ആയിരിക്കും. എന്നാൽ താഴെ കൊടുത്തിരിക്കുന്ന സ്റ്റേറ്റ്മെന്റുകൾ പ്രവർത്തിക്കുമ്പോൾ വേരിയബിൾ 'c' യുടെ വില 'a' ആയിരിക്കും.

```
char x = 'a';
char c = tolower(x) ;
```

tolower(), toupper() എന്നീ ഫങ്ഷനുകളുടെ കാര്യത്തിൽ ആർഗ്യുമെന്റ് ഒരു അക്ഷരമല്ലെങ്കിൽ തന്നിരിക്കുന്ന ക്യാരക്ടർ തന്നെ തിരിച്ചു നൽകും.

പ്രോഗ്രാം 10.3 ൽ ക്യാരക്ടർ ഫങ്ഷനുകളുടെ ഉപയോഗം വിവരിച്ചിരിക്കുന്നു. ഈ പ്രോഗ്രാം ഒരു വാചകം സ്വീകരിക്കുകയും സ്ട്രിങ്ങിലെ ചെറിയ അക്ഷരങ്ങൾ, വലിയ അക്ഷരങ്ങൾ, അക്കങ്ങൾ എന്നിവയുടെ എണ്ണം കണ്ടുപിടിക്കുകയും ചെയ്യുന്നു. ഇത് മൊത്തം സ്ട്രിങ്ങിനെ വലിയ അക്ഷരത്തിലും ചെറിയ അക്ഷരത്തിലും പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു.

പ്രോഗ്രാം 10.3 തന്നിരിക്കുന്ന സ്ട്രിങ്ങിലെ വിവിധ തരത്തിലുള്ള ക്യാരക്ടറുകൾ എണ്ണുന്നതിന്.

```
#include <iostream>
#include <cstdio>
#include <cctype>
using namespace std;
int main()
```

```
{
char text[80];
int Ucase=0, Lcase=0, Digit=0, i;
cout << "Enter a line of text: ";
gets(text);
for(i=0; text[i]!='\0'; i++)
    if (isupper(text[i])) Ucase++;
        else if (islower(text[i])) Lcase++;
            else if (isdigit(text[i])) Digit++;
cout << "\nNo. of uppercase letters = " << Ucase;
cout << "\nNo. of lowercase letters = " << Lcase;
cout << "\nNo. of digits = " << Digit;
cout << "\nThe string in uppercase form is\n";
i=0;
while (text[i]!='\0')
{
    putchar(toupper(text[i]));
    i++;
}
cout << "\nThe string in lowercase form is\n";
i=0;
do
{
    putchar(tolower(text[i]));
    i++;
} while(text[i]!='\0');
return 0;
}
```

i യുടെ വില ക്യാരക്ടറിലേക്ക് പോയിന്റ് ചെയ്യുമ്പോൾ ലൂപ്പ് അവസാനിക്കും

putchar() ന് പകരം cout<< ഉപയോഗിക്കുമ്പോൾ ക്യാരക്ടറിന്റെ ASCII വില പ്രദർശിപ്പിക്കും

ഒരു മാതൃകാ ഔട്ട്പുട്ട് താഴെ നൽകുന്നു.

```
Enter a line of text : The vehicle ID is KL01 AB101
No. of uppercase letters = 7
No. of lowercase letters = 11
No. of digits = 5
The string in uppercase form is
THE VEHICLE ID IS KL01 AB101
The string in lowercase form is
the vehicle id is kl01 ab101
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

10.3.4: പരിവർത്തന ഫങ്ഷനുകൾ (Conversion functions)

ഒരു സ്ട്രിങ്ങിനെ പൂർണ്ണസംഖ്യയായും പൂർണ്ണസംഖ്യയെ സ്ട്രിങ്ങ് ആയും പരിവർത്തനം ചെയ്യുന്നതിന് ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നു. C++ ൽ ലഭ്യമായ വിവിധ പരിവർത്തന ഫങ്ഷനുകൾ താഴെ കൊടുത്തിരിക്കുന്നു. ഒരു പ്രോഗ്രാമിൽ ഇത്തരം

ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് **cstdlib** (ടർബോ C++ ൽ **stdlib.h**) എന്ന ഹെഡർ ഫയൽ ഉൾപ്പെടുത്തേണ്ടതാണ്.

a. itoa()

ഒരു പൂർണ സംഖ്യയെ സ്ട്രിങ് ഇനത്തിലേക്ക് മാറ്റുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. ഫങ്ഷന്റെ വാക്യഘടനയാണ്,

```
itoa(int n, char c[], int len)
```

ഈ ഫങ്ഷന് മൂന്ന് ആർഗ്യുമെന്റുകൾ ആവശ്യമെന്ന് വാക്യഘടനയിൽ നിന്ന് നമുക്ക് കാണാവുന്നതാണ്. ഇതിൽ ആദ്യത്തേത് പരിവർത്തനം ചെയ്യേണ്ട സംഖ്യയാണ്. രണ്ടാമത്തേത് പരിവർത്തനം ചെയ്ത് ലഭിക്കുന്ന സ്ട്രിങ് സംഭരിക്കുന്ന ക്യാരക്ടർ അറെയും അവസാനത്തേത് ക്യാരക്ടർ അറെയുടെ വലിപ്പവും ആകുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഈ ഫങ്ഷൻ വിശദമാക്കുന്നു.

```
int n = 2345;
char c[10];
itoa(n, c, 10);
cout << c;
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കോഡ് "2345" എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കും.

b. atoi()

ഒരു സ്ട്രിങ് വിലയെ പൂർണസംഖ്യയാക്കി മാറ്റുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു.

ഫങ്ഷന്റെ വാക്യഘടനയാണ്

```
int atoi(char c[]);
```

ഈ ഫങ്ഷൻ ഒരു സ്ട്രിങ്ങിനെ ആർഗ്യുമെന്റായി സ്വീകരിച്ച് സ്ട്രിങ്ങിന്റെ പൂർണ്ണ സംഖ്യ രൂപത്തിൽ തിരിച്ചു നൽകുകയും ചെയ്യുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് "2345" എന്ന സ്ട്രിങ്ങിനെ 2345 എന്ന പൂർണ്ണ സംഖ്യയാക്കി മാറ്റുന്നു.

```
int n;
char c[10] = "2345";
n = atoi(c);
cout << n;
```

സ്ട്രിങ്ങിൽ അക്കങ്ങൾ അല്ലാത്ത ക്യാരക്ടറുകൾ ഉൾക്കൊണ്ടിട്ടുണ്ടെങ്കിൽ, ഔട്ട്പുട്ട് പൂജ്യം (0) ആയിരിക്കും. എന്നാൽ സ്ട്രിങ് അക്കങ്ങളിൽ ആരംഭിക്കുകയാണെങ്കിൽ ആ ഭാഗം മാത്രമേ പൂർണ്ണ സംഖ്യയായി മാറ്റുകയുള്ളൂ. ഈ ഫങ്ഷന്റെ ചില ഉപയോഗങ്ങളും അവയുടെ ഔട്ട്പുട്ടും താഴെ കൊടുത്തിരിക്കുന്നു.

(i) `atoi("Computer")` - പൂജ്യം (0) തിരിച്ചു നൽകും.

(ii) `atoi("12.56")`-12 തിരിച്ചു നൽകും.

(iii) `atoi("a2b")` - പൂജ്യം (0) തിരിച്ചു നൽകും.

(iv) `atoi("2ab")` - 2 തിരിച്ചു നൽകും.

(v) `atoi(".25")` - പൂജ്യം (0) തിരിച്ചു നൽകും.

(vi) `atoi("5+3")` - 5 തിരിച്ചു നൽകും.

ഈ ഫങ്ഷനുകൾ ഉപയോഗിച്ച് പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യുന്നത് പ്രോഗ്രാം 10.4 ൽ വിവരിക്കുന്നു. ഈ പ്രോഗ്രാമിൽ ജനനതീയ്യതിയുടെ മൂന്ന് ഭാഗങ്ങൾ (ദിവസം, മാസം, വർഷം) സ്വീകരിച്ച് അതിനെ തീയതി രൂപത്തിൽ പ്രദർശിപ്പിക്കുന്നു.

പ്രോഗ്രാം 10.4 ജനനതീയതി തീയതി രൂപത്തിൽ പ്രദർശിപ്പിക്കുന്നതിന്

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
int main()
{
char dd[10], mm[10], yy[10], dob[30];
int d, m, y;
cout<<"Enter day, month and year in your Date of Birth: ";
cin>>d>>m>>y;
itoa(d, dd, 10);
itoa(m, mm, 10);
itoa(y, yy, 10);
strcpy(dob, dd);
strcat(dob, "-");
strcat(dob, mm);
strcat(dob, "-");
strcat(dob, yy);
cout<<"Your Date of Birth is "<<dob;
return 0;
}
```

പ്രോഗ്രാം 10.4 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
Enter day, month and year in your Date of Birth: 26  11  1999
Your Date of Birth is 26-11-1999
```

10.3.5 ഇൻപുട്ട്/ഔട്ട്പുട്ട് കൈകാര്യം ചെയ്യുന്ന ഫങ്ഷൻ (I/O Manipulating functions)

C++ ലെ ഇൻപുട്ട്, ഔട്ട്പുട്ട് പ്രവർത്തനങ്ങൾ കൈകാര്യം ചെയ്യുവാൻ ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നു. ഒരു പ്രോഗ്രാമിൽ ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുവാൻ **ciomanip** ഹെഡർ ഫയൽ ഉൾപ്പെടുത്തേണ്ടതുണ്ട്.

setw()

ഒരു സ്ട്രിങ്ങ് പ്രദർശിപ്പിക്കുവാൻ അനുവദിക്കപ്പെട്ട സ്ഥലം ഈ ഫങ്ഷൻ വ്യക്തമാക്കുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ഇതിന്റെ അനന്തരഫലം കാണിക്കുന്നു.

```
char s[]="hello";
cout<<setw(10)<<s<<setw(10)<<"friends";
```

മുകളിലത്തെ കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
hello    friends
```

`hello, Friends` എന്നീ സ്ട്രിങ്ങുകൾ പ്രദർശിപ്പിക്കുന്നതിന് 10 ക്യാരക്ടർ സ്ഥാനങ്ങൾ അനുവദിച്ചിരിക്കുന്നു. സ്ട്രിങ്ങിന്റെ വലുപ്പം ഇതിനേക്കാൾ കുറവാണെങ്കിൽ ബാക്കി സ്ഥലം ശൂന്യസ്ഥലങ്ങളായി നിലനിൽക്കും.

നമുക്ക് ചെയ്യാം

താഴെ കൊടുത്തിരിക്കുന്ന രീതിയിൽ ഒരു ചാർട്ട് തയാറാക്കി അതിലെ നിരകളിൽ നാം ഇതുവരെ ചാർട്ട് ചെയ്ത എല്ലാ മുൻ നിർവചിത ഫങ്ഷനുകളും ഉപയോഗിക്കാം

Function	Usage	Syntax	Example	Output

സ്വയം പരിശോധിക്കാം

1. മോഡ്യുളാർ പ്രോഗ്രാമിങ് എന്നാൽ എന്ത്?
2. C++ ലെ ഒരു ഫങ്ഷൻ എന്നാലെന്ത്?
3. ക്യാരക്ടർ ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് ആവശ്യമായ ഹെഡർ ഫയലിന്റെ പേര് എഴുതുക.
4. അനുവദിച്ച സ്ഥലത്ത് ഡാറ്റ പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഫങ്ഷന്റെ പേര് എഴുതുക.
5. കൂട്ടത്തിൽ ചേരാത്തത് എടുത്തെഴുതി അതിനുള്ള കാരണം നൽകുക.

   (a) `strlen()` (b) `itoa()` (c) `strcpy()` (d) `strcat()`

10.4 ഉപയോക്തൃ നിർവചിത ഫങ്ഷനുകൾ (User defind functions)

നമ്മൾ ഇതുവരെ ചർച്ച ചെയ്ത എല്ലാ പ്രോഗ്രാമിലും `main()` എന്ന പേരിലുള്ള ഫങ്ഷൻ ഉണ്ട്. പ്രോഗ്രാമിന്റെ ആദ്യത്തെ വരി പ്രീ-പ്രോസസറിന് നിർദേശം നൽകുന്ന പ്രസ്താവനയാണ് എന്ന് നമുക്കറിയാം. യഥാർത്ഥത്തിൽ അതിന് ശേഷമുള്ളത് ഫങ്ഷന്റെ നിർവചനമാണ്. പ്രോഗ്രാമുകളിലെ `void main()`നെ ഫങ്ഷന്റെ ഫങ്ഷൻ ഹെഡർ (അല്ലെങ്കിൽ ഫങ്ഷൻ ഹെഡിങ്ങ്) എന്ന് വിളിക്കുന്നു. ഇതിനെ തുടർന്ന് { } എന്ന ആവരണങ്ങൾക്കുള്ളിൽ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകളെ ബോഡി എന്നു വിളിക്കുന്നു.

ഫങ്ഷൻ നിർവചനത്തിന്റെ വാക്യഘടന താഴെ കൊടുത്തിരിക്കുന്നതാണ്.

```
data_type function_name(argument_list)
{
    statements in the body;
}
```

ഡാറ്റ ഇനം എന്നത് C++ ലെ ഏതെങ്കിലും സാധുതയുള്ള ഡാറ്റ ഇനമാണ്. ഒരു ഉപയോക്തൃ നിർവഹിത പദം (ഐഡന്റിഫയർ) ആണ്. function_name ഐശ്ചികമായ പരാമീറ്ററുകളുടെ കൂട്ടമാണ് ആർഗ്യുമെന്റ് ലിസ്റ്റ്. ഡാറ്റ ഇനങ്ങളോട് കൂടിയ ഒരു കൂട്ടം വേരിയബിളുകളെ കോമ ഉപയോഗിച്ച് വേർതിരിക്കുന്നു. ഫങ്ഷന്റെ ചട്ടകൂട്ടിൽ ഉൾക്കൊള്ളിച്ചിരിക്കുന്ന C++ സ്റ്റേറ്റ്മെന്റുകൾ ഫങ്ഷന്റെ നിർവഹണത്തിന് ആവശ്യമാണ്. ഒരിക്കൽ ഒരു ഫങ്ഷൻ നിർമ്മിക്കാൻ നാം തീരുമാനിച്ചാൽ താഴെ കൊടുത്തിരിക്കുന്ന ചോദ്യങ്ങൾക്ക് ഉത്തരം നൽകേണ്ടതുണ്ട്.

(i) ഫങ്ഷൻ ഹെഡറിൽ ഏത് ഡാറ്റ ഇനം ഉപയോഗിക്കും?

(ii) എത്ര ആർഗ്യുമെന്റുകൾ ആവശ്യമുണ്ട്? അവ ഓരോന്നിന്റെയും ഡാറ്റ ഇനം എന്തായിരിക്കും?

getchar(), strcpy(), sqrt() എന്നീ മുൻ നിർവചിത ഫങ്ഷനുകൾ എങ്ങനെയാണ് ഉപയോഗിച്ചതെന്ന് നമുക്ക് അറിയാമല്ലോ. ഒരു C++ പ്രസ്താവനയിൽ ഈ ഫങ്ഷനുകൾ വിളിക്കുമ്പോൾ (ഉപയോഗിക്കുമ്പോൾ) അവ പ്രവർത്തിക്കുമെന്ന് നാം കണ്ടിട്ടുണ്ട്.

getchar() എന്ന ഫങ്ഷൻ ഒരു ആർഗ്യുമെന്റും ഉപയോഗിക്കുന്നില്ല. എന്നാൽ strcpy() പ്രവർത്തിക്കുന്നതിന് രണ്ട് സ്ട്രിങ് ആർഗ്യുമെന്റുകൾ വേണം ഈ ആർഗ്യുമെന്റുകൾ ഇല്ലാതെ ഈ ഫങ്ഷൻ പ്രവർത്തിക്കില്ല. അതിന് കാരണം ഇത് നിർവചിച്ചിരിക്കുന്നത്, രണ്ട് സ്ട്രിങ് (ക്യാരക്ടർ അറെ) ആർഗ്യുമെന്റ് ഉപയോഗിച്ചാണ്. എന്നാൽ sqrt() യ്ക്ക് ആർഗ്യുമെന്റായി ഒരു സംഖ്യ ആവശ്യമാണ്. അതിനോടൊപ്പം തന്നിരിക്കുന്ന ആർഗ്യുമെന്റിൽ മുൻകൂട്ടി തയ്യാറാക്കിയ പ്രവർത്തനം നടത്തിയ ശേഷം ഒരു ഫലം (ഡബിൾ ഡാറ്റാടൈപ്പ്) തിരികെ നൽകുന്നു. മുകളിൽ പരാമർശിച്ച ഫലത്തെ ഫങ്ഷന്റെ റിട്ടേൺ വാല്യു (return value) എന്ന് വിളിക്കുന്നു. ഈ വില ഫങ്ഷന്റെ റിട്ടേൺ വിലയെ ആശ്രയിച്ചിരിക്കുന്നു. മറ്റൊരു രീതിയിൽ പറഞ്ഞാൽ ഫങ്ഷന്റെ ഡാറ്റ ഇനത്തിന് അനുസരിച്ചുള്ള വില ആയിരിക്കണം ഫങ്ഷൻ തിരികെ നൽകേണ്ടത്.

അതുകൊണ്ട് ഫങ്ഷന്റെ ഡാറ്റ ഇനത്തെ ഫങ്ഷന്റെ റിട്ടേൺ ഇനം എന്നും പറയാറുണ്ട്. നാം main() ഫങ്ഷനിൽ return 0; എന്ന സ്റ്റേറ്റ്മെന്റ് ഉപയോഗിക്കുന്നത്, GCC യുടെ ആവശ്യാനുസരണം main() ന്റെ തിരികെ നൽകുന്ന വില int ഡാറ്റ ഇനം ആയി നിർവചിച്ചിരിക്കുന്നതിനാലാണ്.

ആർഗ്യുമെന്റുകളുടെ എണ്ണവും ഇനവും (data type) ഫങ്ഷന്റെ പ്രവർത്തനത്തിന് ആവശ്യമായ ഡാറ്റയെ ആശ്രയിച്ചിരിക്കുന്നു. എന്നാൽ setw(), gets() തുടങ്ങിയ ഫങ്ഷനുകൾ വിലകൾ ഒന്നും തിരിച്ച് നൽകുന്നില്ല. ഇത്തരം ഫങ്ഷനുകളുടെ ഹെഡറിൽ void എന്ന് റിട്ടേൺ ഇനമായി ഉപയോഗിക്കുന്നു. ഒരു ഫങ്ഷൻ ഒന്നുകിൽ ഒരു വില തിരിച്ചു നൽകും, അല്ലെങ്കിൽ ഒരു വിലയും തിരിച്ചു നൽകുന്നില്ല.

10.4.1. ഉപയോക്തൃ നിർമിത ഫങ്ഷനുകൾ നിർമിക്കുന്നു (Creating user defined functions)

മുകളിൽ ചർച്ചചെയ്ത വാക്യഘടനയെ അടിസ്ഥാനമാക്കി നമുക്ക് ഫങ്ഷനുകൾ നിർമിക്കാം. ഒരു സന്ദേശം പ്രദർശിപ്പിക്കാനുള്ള ഫങ്ഷൻ ആണ് താഴെ കൊടുത്തിരിക്കുന്നത്.

```
void saywelcome()
{
    cout<<"Welcome to the world of functions";
}
```

ഫങ്ഷന്റെ പേര് saywelcome() എന്നാണ്. ഇതിന്റെ ഡാറ്റ ഇനം (റിട്ടേൺ ടൈപ്പ്) വോയിഡ് (void) ആണ്. ഇതിന് ആർഗ്യുമെന്റുകൾ ഇല്ല. ഫങ്ഷൻ ചട്ടക്കൂട്ടിൽ ഒരു പ്രസ്താവന മാത്രമേ ഉള്ളൂ.

ഇപ്പോൾ നമുക്ക് രണ്ട് സംഖ്യകളുടെ തുക കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു ഫങ്ഷൻ നിർവ്വചിക്കാം. ഒരേ ഉദ്യമത്തിനായി നാല് വിവിധ തരത്തിലുള്ള ഫങ്ഷൻ നിർവചനങ്ങൾ നൽകുന്നു. എന്നാൽ അവയുടെ നിർവചന ശൈലികൾ വ്യത്യാസപ്പെട്ടിരിക്കുന്നതിനാൽ അവയിലോരോന്നിന്റെയും ഉപയോഗം മറ്റൊന്നിൽ നിന്നും വ്യത്യാസപ്പെട്ടിരിക്കുന്നു.

ഫങ്ഷൻ 1	ഫങ്ഷൻ 2

`void sum1()`
`{`
`int a, b, s;`
`cout<<"Enter 2 numbers: ";`
`cin>>a>>b;`
`s=a+b;`
`cout<<"Sum="<<s;`
`}`
	`int sum2()`
	`{`
	`int a, b, s;`
	`cout<<"Enter 2 numbers: ";`
	`cin>>a>>b;`
	`s=a+b;`
	`return s;`
	`}`

ഫങ്ഷൻ 3	ഫങ്ഷൻ 4

`void sum3(int a, int b)`
`{`
`int s;`
`s=a+b;`
`cout<<"Sum="<<s;`
`}`
	`int sum4(int a, int b)`
	`{`
	`int s;`
	`s=a+b;`
	`return s;`
	`}`

നമുക്ക് ഈ ഫങ്ഷനുകൾ വിശകലനം ചെയ്ത് അവ എങ്ങനെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു എന്ന് നോക്കാം. എല്ലാ ഫങ്ഷനുകളുടേയും ഉദ്യമം ഒന്നു തന്നെയാണ്. എന്നാൽ പരാമീറ്ററുകളുടെ എണ്ണത്തിലും റിട്ടേൺ ഇനത്തിലും അവ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു. ടേബിൾ 10.1 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ വോയിഡ് അല്ലാത്ത ഡാറ്റ ഇനം ഉപയോഗിച്ച് നിർവചിച്ചിരിക്കുന്ന ഫങ്ഷനുകൾ അതിന് ചേർന്ന തരത്തിലുള്ള വില തിരിച്ചു നൽകും. ഇതിന് വേണ്ടിയാണ് റിട്ടേൺ പ്രസ്താവന ഉപയോഗിച്ചിരിക്കുന്നത്. (ഫങ്ഷൻ 2ഉം, ഫങ്ഷൻ 4ഉം പരിശോധിക്കുക). റിട്ടേൺ (return statement) പ്രസ്താവന വിളിച്ചിരിക്കുന്ന ഫങ്ഷനിലേക്ക് ഒരു വില തിരിച്ച് നൽകുന്നതിനോടൊപ്പം പ്രോഗ്രാം നിയന്ത്രണം തിരിച്ച് കൈമാറുകയും ചെയ്യുന്നു. അതുകൊണ്ട് ഒരു റിട്ടേൺ സ്റ്റേറ്റ്മെന്റ് പ്രവർത്തിച്ചു കഴിഞ്ഞാൽ ഫങ്ഷനിലെ പിന്നീട് വരുന്ന പ്രസ്താവനകൾ പ്രവർത്തിക്കുകയില്ല എന്നത് ഓർക്കുക.

പേര്	ആർഗ്യുമെന്റുകൾ	തിരിച്ച് നൽകുന്ന വില
`sum1()`	ആർഗ്യുമെന്റുകൾ ഇല്ല	ഒരു വിലയും തിരിച്ച് നൽകുന്നില്ല
`sum2()`	ആർഗ്യുമെന്റുകൾ ഇല്ല	പൂർണ്ണ സംഖ്യ തിരിച്ച് നൽകുന്നു
`sum3()`	രണ്ട് പൂർണ്ണ സംഖ്യകൾ	ഒരു വിലയും തിരിച്ച് നൽകുന്നില്ല
`sum4()`	രണ്ട് പൂർണ്ണ സംഖ്യകൾ	പൂർണ്ണ സംഖ്യ തിരിച്ച് നൽകുന്നു

പട്ടിക 10.1 : ഫങ്ഷനുകളുടെ വിശകലനം

ഒട്ടുമിക്ക ഫങ്ഷനുകളിലും റിട്ടേൺ പ്രസ്താവന ഫങ്ഷന്റെ അവസാനമാണ് നൽകുന്നത്. `void` ഡാറ്റ ഇനം ഉപയോഗിച്ച് നിർവചിച്ച ഫങ്ഷനുകളുടെ ചട്ടക്കൂടിനുള്ളിൽ റിട്ടേൺ പ്രസ്താവന ഉണ്ടായേക്കാം. എന്നാൽ അതിന് നമുക്ക് ഒരു വിലയും നൽകാൻ കഴിയില്ല. `main()` ഫങ്ഷന്റെ റിട്ടേൺ ഇനം ഒന്നുകിൽ `void` അല്ലെങ്കിൽ `int` ആണ്.

ഇനി നമുക്ക് ഈ ഫങ്ഷനുകൾ എങ്ങനെ വിളിക്കണമെന്നും അവ എങ്ങനെ പ്രവർത്തിക്കുമെന്നും നോക്കാം. `main()` ഫങ്ഷൻ ഒഴികെ മറ്റൊരു ഫങ്ഷനും സ്വയം പ്രവർത്തിക്കുക ഇല്ല എന്ന് നമുക്ക് അറിയാം. മുൻ നിർവചിതമോ ഉപയോക്തൃ നിർവചിതമോ ആയ മറ്റ് ഉപ ഫങ്ഷനുകൾ `main()` ഫങ്ഷനിലോ മറ്റ് ഉപയോക്തൃ നിർവചിത ഫങ്ഷനിലോ വിളിക്കുമ്പോൾ മാത്രമേ പ്രവർത്തിക്കൂ. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാമിൽ ചതുരത്തിനുള്ളിൽ നൽകിയിരിക്കുന്ന കോഡ്, ഫങ്ഷൻ വിളിക്കുന്നതിനെ സൂചിപ്പിക്കുന്നു. ഇവിടെ `main()` വിളിക്കുന്നു ഫങ്ഷനും `sum1()`, `sum2()`, `sum3()`, `sum4()` എന്നിവ വിളിക്കപ്പെട്ട ഫങ്ഷനുകളുമാണ്.

```
int main()
{
    int x, y, z=5, result;
    cout << "\nCalling the first function\n";
    sum1();
    cout << "\nCalling the second function\n";
    result = sum2();
    cout << "Sum given by function 2 is " << result;
    cout << "\nEnter values for x and y : ";
    cin >> x >> y;
    cout << "\nCalling the third function\n";
    sum3(x,y);
    cout << "\nCalling the fourth function\n";
    result = sum4(z,12);
    cout << "Sum given by function 4 is " << result;
    cout << "\nEnd of main function"
}
```

പ്രോഗ്രാമിന്റെ ഔട്ട്പുട്ട് ചുവടെ ചേർക്കുന്നു.

```
Calling the first function
Enter 2 numbers: 10  25
Sum=35
Calling the second function
Enter 2 numbers: 5   7
Sum given by function 2 is 12
Enter values for x and y : 8   13
Calling the third function
Sum=21
Calling the fourth function
Sum given by function 4 is 17
End of main function
```

sum1() ഫങ്ഷനിൽ a ക്കും b യുടേയും ഇൻപുട്ട്

sum 2() ഫങ്ഷനിൽ a, b യുടേയും ഇൻപുട്ട്

main() ഫങ്ഷനിൽ x, y എന്നിവയുടേയും വേണ്ട ഇൻപുട്ട്

ഫങ്ഷൻ 4 ന് നൽകിയിരിക്കുന്ന ഉദ്യമത്തിന് രണ്ട് സംഖ്യകൾ ആവശ്യമായതിനാൽ രണ്ട് ആർഗ്യുമെന്റുകൾ നാം നൽകുന്നു. ഫങ്ഷൻ ചില കണക്കുകൂട്ടലുകൾ നിർവ്വഹിച്ച് ഒരു ഉത്തരം നൽകുന്നു. ഒരേ ഒരു ഉത്തരം മാത്രം ഉള്ളതിനാൽ അത് തിരിച്ച് നൽകാൻ കഴിയും. രണ്ട് സംഖ്യകളുടെ തുക കണ്ടുപിടിക്കുന്നതിന് ഈ ഫങ്ഷൻ താരതമ്യേന നല്ലതാണ്.

ഇനി നമുക്ക് തന്നിരിക്കുന്ന ഒരു സംഖ്യ പെർഫക്ട് ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം നമുക്ക് എഴുതാം. ആ സംഖ്യ ഒഴികെ അതിന്റെ ബാക്കി ഘടകങ്ങളുടെയെല്ലാം തുക ആ സംഖ്യ തന്നെ ആണെങ്കിൽ അതിനെ പെർഫക്ട് സംഖ്യ എന്ന് പറയുന്നു. ഉദാഹരണത്തിന് 28 ഒരു പെർഫക്ട് സംഖ്യ ആകുന്നു. എന്തുകൊണ്ടെന്നാൽ 28 അല്ലാതെയുള്ള ഘടകങ്ങൾ 1,2,4,7,14 എന്നിവ ആകുന്നു. ഈ ഘടകങ്ങളുടെ തുക 28 ആണ്. ഈ പ്രശ്നം പരിഹരിക്കുന്നതിനായി നമുക്ക് ഒരു സംഖ്യ ആർഗ്യുമെന്റ് ആയി സ്വീകരിക്കുകയും അതിന്റെ ഘടകങ്ങളുടെ തുക തിരിച്ചു നൽകുന്നതിനുള്ള ഒരു ഫങ്ഷൻ നിർവചിക്കുകയും ചെയ്യാം. ഈ ഫങ്ഷൻ ഉപയോഗിച്ച് നമുക്ക് ഒരു പ്രോഗ്രാം എഴുതാം. എന്നാൽ C++ പ്രോഗ്രാമിൽ നാം എവിടെയാണ് ഉപയോക്തൃ നിർവചിത ഫങ്ഷൻ എഴുതുന്നത്? താഴെ കൊടുത്തിരിക്കുന്ന പട്ടിക ഉപയോക്തൃ നിർവചിത ഫങ്ഷൻ എഴുതുന്നതിന് വേണ്ട രണ്ട് ശൈലികൾ കാണിക്കുന്നു.

പ്രോഗ്രാം 10.5   പെർഫക്ട് സംഖ്യ ആണോ എന്ന് പരിശോധിക്കുന്നു.   പ്രോഗ്രാം 10.6

ഫങ്ഷൻ main() ന് മുൻപ്	ഫങ്ഷൻ main() ന് ശേഷം

`#include<iostream>`
`using namespace std;`
`int sumfact(int N)`
`{ int i, s = 0;`
`  for(i=1; i<=N/2; i++)`
`     if (N%i == 0)`
	`#include<iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`  int num;`
	`  cout<<"Enter the Number: ";`
	`cin>>num;`

```
s = s + i;
  return s;
}
//Definition above main()
int main()
{
  int num;
  cout<<"Enter the Number: ";
  cin>>num;
  if (num==sumfact(num))
     cout<<"Perfect number";
  else
     cout<<"Not Perfect";
  return 0;
}
```

```
  if (num==sumfact(num))
     cout<<"Perfect number";
  else
     cout<<"Not Perfect";
  return 0;
}
//Definition below main()
int sumfact(int N)
{ int i, s = 0;
  for(i=1; i<=N/2; i++)
     if (N%i == 0)
        s = s + i;

    return s;
}
```

പട്ടിക 10.2 : ഫംങ്ഷനുകളുടെ വിശകലനം.

പ്രോഗ്രാം 10.5 കംപൈയിൽ ചെയ്യുമ്പോൾ അവിടെ ഒരു തെറ്റും ഉണ്ടാവില്ല പ്രോഗ്രാം പ്രവർത്തിക്കുമ്പോൾ നമുക്ക് താഴെ കൊടുത്തിരിക്കുന്ന ഔട്ട്പുട്ട് ലഭിക്കും.

```
Enter the Number: 28
Perfect number
```

നാം പ്രോഗ്രാം 10.6 കംപൈയിൽ ചെയ്യുമ്പോൾ അവിടെ ഒരു തെറ്റ് ഉണ്ടാകും `sumfact () was not declared in this scope' എന്താണ് ഈ പിശക് അർത്ഥമാക്കുന്നത് എന്ന് നമുക്ക് നോക്കാം.

10.4.2 ഫങ്ഷനുകളുടെ പ്രോട്ടോടൈപ്പ് (Prototype of functions)

ഒരു C++ പ്രോഗ്രാമിൽ എത്ര ഫങ്ഷനുകൾ വേണമെങ്കിലും ഉൾപ്പെടുത്താം എന്ന് നാം കണ്ടു കഴിഞ്ഞു. എന്നാൽ അതിന്റെ പ്രവർത്തനം തുടങ്ങുവാൻ ഒരു main() ഫങ്ഷൻ ഉണ്ടായിരിക്കണം. ഫങ്ഷനുകളുടെ നിർവചനങ്ങൾ നാം ആഗ്രഹിക്കുന്ന രീതിയിൽ ഏത് ക്രമത്തിലും എഴുതാം. നമുക്ക് ആദ്യം തന്നെ main() ഫങ്ഷൻ നിർവചിക്കുകയും മറ്റെല്ലാം ഫങ്ഷനുകളും അതിന് ശേഷമോ മുമ്പോ നൽകാവുന്നതാണ്. പ്രോഗ്രാം 10.5 ൽ main() ഫങ്ഷൻ മറ്റ് എല്ലാ ഉപയോക്തൃ നിർമ്മിത ഫങ്ഷനുകൾക്ക് ശേഷമാണ് നൽകിയിരിക്കുന്നത്. എന്നാൽ പ്രോഗ്രാം 10.6 ൽ main() ഫങ്ഷൻ മറ്റ് എല്ലാ ഫങ്ഷനുകൾക്കും മുമ്പാണ് നിർവചിച്ചിരിക്കുന്നത്. നാം ഈ പ്രോഗ്രാം കംപൈൽ ചെയ്യുമ്പോൾ അത് ഒരു തെറ്റ് ചൂണ്ടിക്കാണിക്കും. "sumfact was not declared in this scope" ഇത് എന്തുകൊണ്ടെന്നാൽ sumfact() എന്ന ഫങ്ഷൻ വിളിച്ചിരിക്കുന്നത് അതിന്റെ നിർവചനത്തിന് മുൻപേ ആണ്. main() ഫങ്ഷന്റെ കംപൈലേഷൻ സമയത്ത് കംപൈലർ sumfact() എന്ന ഫങ്ഷൻ വിളി (function call) എത്തുമ്പോൾ അതിന് അങ്ങനെ ഒരു ഫങ്ഷനെ കുറിച്ച് അറിവില്ല. അങ്ങനെ ഒരു ഫങ്ഷൻ ഉണ്ടോ എന്നും അതിന്റെ പ്രയോഗരീതി ശരിയാണോ അല്ലയോ എന്നും കംപൈലറിന് പരിശോധിക്കാൻ

സാധിക്കില്ല. അങ്ങനെ അത് ഫങ്ഷൻ പ്രോട്ടോടൈപ്പിന്റെ അഭാവം മൂലമുള്ള ഒരു തെറ്റ് ചൂണ്ടിക്കാണിക്കുന്നു. ഒരു ഫങ്ഷൻ പ്രോട്ടോടൈപ്പ് എന്നത് ഒരു ഫങ്ഷന്റെ പ്രഖ്യാപനം ആണെന്നും അതിലൂടെ ഫങ്ഷന്റെ പേര്, അതിന്റെ റിട്ടേൺ ഇനം, ആർഗ്യുമെന്റുകളുടെ എണ്ണവും ഇനവും അതിന്റെ പ്രാപ്യത എന്നിവ കമ്പയിലറിന് ലഭ്യമാകുന്നു. ഈ വിവരങ്ങൾ പ്രോഗ്രാമിലെ ഫങ്ഷൻ കാൾ ശരിയാണോ എന്ന് പരിശോധിക്കുന്നതിന് അത്യാവശ്യമാണ്. ഈ വിവരങ്ങൾ ഫങ്ഷൻ ഹെഡറിൽ ലഭ്യമാണ്. അതിനാൽ ഫങ്ഷൻ ഹെഡർ (ഫങ്ഷൻ പ്രോട്ടോടൈപ്പ്) ഫങ്ഷനെ വിളിയ്ക്കുന്നതിന് മുമ്പായി പ്രസ്താവനയായി എഴുതാം. ഇതിന്റെ ഘടന താഴെ കൊടുത്തിരിക്കുന്നു.

```
data_type function_name(argument_list);
```

പ്രോട്ടോടൈപ്പിൽ ആർഗ്യുമെന്റിന്റെ പേരുകൾ നൽകേണ്ടതില്ല. അതുകൊണ്ട് താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവന `main()` ഫങ്ഷനിലെ ഫങ്ഷൻ വിളിയ്ക്ക് മുൻപ് കൂട്ടിച്ചേർത്ത് പ്രോഗ്രാം 10.6 ലെ തെറ്റ് തിരുത്തണം.

```
int sumfact(int);
```

ഒരു വേരിയബിൾ പ്രഖ്യാപിക്കുന്നത് പോലെ ഫങ്ഷനും അത് പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നതിന് മുൻപേ പ്രഖ്യാപിച്ചിരിക്കണം. പ്രോഗ്രാമിൽ ഒരു ഫങ്ഷൻ അത് ഉപയോഗിക്കുന്നതിന് മുൻപേ നിർവ്വചിച്ചിട്ടുണ്ടെങ്കിൽ ഫങ്ഷൻ പ്രഖ്യാപനം പ്രത്യേകമായി നടത്തേണ്ടതില്ല. പ്രഖ്യാപന പ്രസ്താവന `main()` ഫങ്ഷന് പുറത്തും നൽകാവുന്നതാണ്. പ്രോട്ടോടൈപ്പിന്റെ സ്ഥാനം ഫങ്ഷന്റെ പ്രാപ്യതക്കനുസരിച്ച് വ്യത്യസപ്പെട്ടിരിക്കുന്നു. നമുക്ക് ഇത് ഈ അധ്യായത്തിൽ പിന്നീടുള്ള ഭാഗത്ത് ചർച്ച ചെയ്യാം. ഫങ്ഷൻ നിർവചനത്തിന്റെ സ്ഥാനം എവിടെ ആയിരുന്നാലും പ്രോഗ്രാമിന്റെ പ്രവർത്തനം `main()` ഫങ്ഷനിൽ നിന്ന് ആരംഭിക്കും.

10.4.3 ഫങ്ഷനുകളുടെ ആർഗ്യുമെന്റുകൾ (Arguments of functions)

ഫങ്ഷനിലേക്ക് ഡാറ്റാ ലഭിക്കുന്നതിനായി ആർഗ്യുമെന്റുകൾ അല്ലെങ്കിൽ പാരാമീറ്ററുകൾ ഉപയോഗിക്കാമെന്ന് നാം കണ്ടു. ഫങ്ഷൻ കാളിൽ ആർഗ്യുമെന്റുകളുടെ പ്രാധാന്യം എന്താണെന്ന് നമുക്ക് നോക്കാം. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷൻ പരിഗണിക്കുക.

```
float SimpleInterest(long P, int N, float R)
{
    float amt;
    amt = P * N * R / 100;
    return amt;
}
```

ഈ ഫങ്ഷൻ തന്നിരിക്കുന്ന മുതൽ പലിശനിരക്ക്, കാലം എന്നിവ ഉപയോഗിച്ച് സാധാരണ പലിശ കണക്കാക്കുന്നു.

താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ഭാഗം വിവിധ ഫങ്ഷൻ വിളികൾ വിവരിക്കുന്നു.

```
cout << SimpleInterest(1000,3,2);//Function call 1
int x, y; float z=3.5, a;
cin >> x >> y;
```

```
    a = SimpleInterest(x, y, z); //Function call 2
```

ആദ്യത്തെ പ്രസ്താവന പ്രവർത്തിക്കുമ്പോൾ 1000, 3, 2 എന്നീ വിലകൾ ഫങ്ഷൻ നിർവചനത്തിലെ ആർഗ്യുമെന്റ് ലിസ്റ്റിലേക്ക് അയക്കുന്നു. ആർഗ്യുമെന്റുകളായ P, N, R എന്നിവയ്ക്ക് യഥാക്രമം 1000, 3, 2 എന്നീ വിലകൾ ലഭിക്കുന്നു. അതേപോലെ അവസാനത്തെ പ്രസ്താവന പ്രവർത്തിക്കുമ്പോൾ x, y, z എന്നീ വേരിയബിളുകളുടെ വിലകൾ യഥാക്രമം ആർഗ്യുമെന്റുകളായ P, N, R എന്നിവയിലേക്ക് അയക്കുന്നു.

x, y, z എന്നീ വേരിയബിളുകളെ ആക്ചൽ ആർഗ്യുമെന്റുകൾ അല്ലെങ്കിൽ യഥാർത്ഥ പരാമീറ്ററുകൾ എന്ന് വിളിക്കുന്നു. കാരണം പ്രവർത്തനത്തിനായി ഫങ്ഷനിലേക്ക് അയക്കുന്ന യഥാർത്ഥ ഡാറ്റയാണിവ. ഫങ്ഷൻ ഹെഡറിൽ ഉപയോഗിക്കുന്ന P, N, R എന്നീ വേരിയബിളുകൾ ഫോർമൽ ആർഗ്യുമെന്റുകൾ അല്ലെങ്കിൽ ഫോർമൽ പാരാമീറ്ററുകൾ എന്ന് അറിയപ്പെടുന്നു. വിളിക്കുന്ന ഫങ്ഷനിൽ നിന്നും അയക്കുന്ന ഡാറ്റ സ്വീകരിക്കാൻ വേണ്ടിയാണ് ഈ ആർഗ്യുമെന്റുകൾ ഉപയോഗിച്ചിരിക്കുന്നത്. വിളിച്ച ഫങ്ഷനിൽ നിന്നും വിളിക്കപ്പെട്ട ഫങ്ഷനിലേക്ക് വിലകൾ അയക്കുന്നതിനുള്ള ഉപാധിയാണ് **ആർഗ്യുമെന്റുകൾ** അല്ലെങ്കിൽ **പരാമീറ്ററുകൾ.** ഫങ്ഷൻ നിർവചനത്തിൽ ആർഗ്യുമെന്റുകളായി ഉപയോഗിച്ച വേരിയബിളുകൾ ഫോർമൽ ആർഗ്യുമെന്റുകൾ എന്നറിയപ്പെടുന്നു. ഫങ്ഷൻ കോളിൽ ഉപയോഗിച്ച സ്ഥിരവിലകൾ, വേരിയബിളുകൾ അല്ലെങ്കിൽ പദപ്രയോഗങ്ങൾ എന്നിവ യഥാർത്ഥ ആർഗ്യുമെന്റുകൾ എന്ന് അറിയപ്പെടുന്നു. ഫങ്ഷൻ പ്രോട്ടോടൈപ്പിൽ വേരിയബിളുകൾ ഉപയോഗിക്കുകയാണെങ്കിൽ അവ ഡമ്മി ആർഗ്യുമെന്റുകൾ എന്ന പേരിൽ അറിയപ്പെടുന്നു.

ഇപ്പോൾ നമുക്ക് fact() എന്ന ഫങ്ഷൻ ഉപയോഗിച്ച് ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കുകയും അത് nCr ന്റെ വില കാണുന്നതിനായി ഉപയോഗിക്കുകയും ചെയ്യാം (പ്രോഗ്രാം 10.7) നമുക്ക് അറിയാവുന്നതു പോലെ N എന്ന സംഖ്യയുടെ ഫാക്ടോറിയൽ ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ ഗുണനഫലമാകുന്നു. nCr ന്റെ വില $\frac{n!}{r!(n-r)!}$ എന്ന സൂത്രവാക്യം ഉപയോഗിച്ച് കണ്ടുപിടിക്കുന്നു. ഇവിടെ n! സൂചിപ്പിക്കുന്നത് n എന്ന സംഖ്യയുടെ ഫാക്ടോറിയലാണ്.

പ്രോഗ്രാം 10.7 : nCr ന്റെ വില കണ്ടു പിടിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
int fact(int);
int main()
{
    int n,r;
    int ncr;
    cout<<"Enter the values of n and r : ";
    cin>>n>>r;
    ncr=fact(n)/(fact(r)*fact(n-r));
```

ഫങ്ഷൻ പ്രോട്ടോ ടൈപ്പ്

യഥാർത്ഥ ആർഗ്യുമെന്റുകൾ

സൂത്രവാക്യത്തിന് അനുസൃതമായ ഫങ്ഷൻ വിളി

```
    cout<<n<<"C"<<r<<" = "<<ncr;
    return 0;
}
int fact(int N)
{
    int f;
    for(f=1; N>0; N--)
        f=f*N;
    return f;
}
```

ഫങ്ഷൻ ഹെഡർ

യഥാക്രമ ആർഗ്യുമെന്റുകൾ

ഫാക്ടോറിയൽ തിരിച്ച് നൽകുന്നു

താഴെ കൊടുത്തിരിക്കുന്നത് ഒരു മാതൃക ഔട്ട്പുട്ട് ആണ്.

```
    Enter the values of n and r : 5    2
    5C2 = 10
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ടുകൾ

10.4.4. തനതു ആർഗ്യുമെന്റുകളോട് കൂടിയ ഫങ്ഷനുകൾ (Functions with default arguments)

നമുക്ക് താഴെ പറയുന്ന ആർഗ്യുമെന്റ് പട്ടികയോട് കൂടിയ `TimeSec()` എന്ന ഒരു ഫങ്ഷൻ പരിഗണിക്കാം. ഈ ഫങ്ഷൻ സമയത്തെ പ്രതിനിധീകരിക്കുന്ന മണിക്കൂറുകൾ, മിനിട്ട്, സെക്കന്റ് എന്നിവക്കുള്ള മൂന്ന് സംഖ്യകൾ സ്വീകരിക്കുന്നു.

```
    long TimeSec(int H, int M=0, int S=0)
    {
        long sec = H * 3600 + M * 60 + S;
        return sec;
    }
    SecTime()
```

തന്നിരിക്കുന്ന സമയത്തെ ഫങ്ഷൻ സെക്കന്റുകളിലേക്ക് മാറ്റുന്നു. `M`, `S` എന്നീ ആർഗ്യുമെന്റുകൾക്ക് തനതു വിലയായി പൂജ്യം നൽകിയിരിക്കുന്നത് ശ്രദ്ധിക്കുക. അതുകൊണ്ട് ഈ ഫങ്ഷൻ താഴെ പറയുന്ന രീതികളിൽ വിളിക്കാം.

```
    long s1 = TimeSec(2,10,40);
    long s2 = TimeSec(1,30);
    long s3 = TimeSec(3);
```

ആർഗ്യുമെന്റുകളുടെ പട്ടികയിലെ തനതുവില നൽകുന്ന എല്ലാ ആർഗ്യുമെന്റുകളും വലത്തു നിന്ന് ഇടത്തോട്ട് നൽകണം എന്നത് പ്രധാനമായും ശ്രദ്ധിക്കേണ്ടതാണ്. ഒരു ഫങ്ഷൻ വിളിക്കുമ്പോൾ യഥാർത്ഥ ആർഗ്യുമെന്റുകൾ (actual arguments) ഫോർമൽ ആർഗ്യുമെന്റുകളിലേക്ക് ഇടത് ഭാഗം മുതൽ നൽകുന്നു.

ആദ്യത്തെ പ്രസ്താവന പ്രവർത്തിച്ചപ്പോൾ 2,10,40 എന്നീ വിലകൾ യഥാക്രമം ഫോർമൽ പരാമീറ്ററുകളായ H, M, S എന്നിവയിലേക്ക് ഫങ്ഷൻ വിളിയോടൊപ്പം അയക്കുന്നു. M, S എന്നിവയുടെ തനതു വിലകളെ യഥാർത്ഥ ആർഗ്യുമെന്റുകൾ തിരുത്തി എഴുതുന്നു. രണ്ടാമത്തെ പ്രസ്താവനയിലുള്ള ഫങ്ഷൻ വിളിക്കുമ്പോൾ H നും M നും യഥാർത്ഥ ആർഗ്യുമെന്റുകളുടെ വിലകൾ കിട്ടുമ്പോൾ S അതിന്റെ തനതു വിലയായ പൂജ്യത്തിൽ (0) പ്രവർത്തിക്കുന്നു. അതുപോലെ മൂന്നാമത്തെ പ്രസ്താവന പ്രവർത്തിക്കുമ്പോൾ H ന് വിളിച്ചിരിക്കുന്ന ഫങ്ഷനിൽ നിന്നു വിലകിട്ടുന്നു. എന്നാൽ M ഉം S ഉം അവയുടെ

തനത് വില ഉപയോഗിക്കുന്നു. അതുകൊണ്ട് ഫങ്ഷൻ വിളികൾക്ക് ശേഷം s1, s2, s3 എന്നിവയുടെ വിലകൾ യഥാക്രമം 7840, 5400, 10800 എന്നിങ്ങനെ ആയിരിക്കും.

ഫങ്ഷനുകളുടെ ആർഗ്യുമെന്റുകൾക്ക് തനത് വില നൽകി നിർവ്വചിക്കാൻ സാധിക്കും എന്ന് നാം കണ്ടു കഴിഞ്ഞു. തനത് വില നൽകിയ ആർഗ്യുമെന്റുകളെ തനത് (ഡിഫാൾട്ട്) ആർഗ്യുമെന്റുകൾ എന്നു വിളിക്കുന്നു. *ഇത് ഒരു ഫങ്ഷന് വ്യത്യസ്ത എണ്ണം ആർഗ്യുമെന്റുകൾ കൊണ്ട് വിളിക്കുന്നതിന് ഒരു പ്രോഗ്രാമറെ അനുവദിക്കുന്നു.* അതായത് തനത് ആർഗ്യുമെന്റുകൾക്ക് വിലകൾ നൽകിയോ നൽകാതെയോ ഫങ്ഷനെ വിളിക്കാൻ കഴിയും.

10.4.5 ഫങ്ഷൻ വിളിക്കുന്നതിനുള്ള വിവിധ രീതികൾ (Methods of calling Functions)

നിങ്ങളുടെ ടീച്ചർ ക്ലാസിലെ എല്ലാ വിദ്യാർത്ഥികളുടേയും രക്ഷകർത്താക്കളെ നിങ്ങളുടെ വിദ്യാലയത്തിൽ ഒരു പരിപാടിയിലേക്ക് ക്ഷണിക്കുന്നതിനുള്ള ഒരു കത്ത് തയ്യാറാക്കുവാൻ നിങ്ങളോട് ആവശ്യപ്പെട്ടു എന്ന് കരുതുക. കത്തിന്റെ മാതൃകയും രക്ഷാകർത്താക്കളുടെ പേര് അടങ്ങുന്ന പട്ടികയും നൽകാൻ ടീച്ചർക്ക് കഴിയും. പേരുകളുടെ പട്ടിക രണ്ട് വിധത്തിൽ നൽകാവുന്നതാണ്. ഒന്നുകിൽ യഥാർത്ഥ പട്ടിക അല്ലെങ്കിൽ അതിന്റെ ഫോട്ടോകോപ്പി. ഈ രണ്ട് രീതികളിൽ പേരിന്റെ പട്ടിക വ്യത്യാസം എന്താണ്? ടീച്ചർ യഥാർത്ഥ പട്ടികയാണ് നിങ്ങൾക്ക് നൽകുന്നതെങ്കിൽ, അതിൽ മറ്റൊന്നും എഴുതാതെയും രേഖപ്പെടുത്താതെയും ശ്രദ്ധാപൂർവ്വം ഉപയോഗിക്കണം. എന്തുകൊണ്ടെന്നാൽ ടീച്ചർക്ക് അതേ പട്ടിക തന്നെ ഭാവിയിൽ ആവശ്യമായി വരാം. എന്നാൽ ഫോട്ടോകോപ്പിയാണ് നിങ്ങൾക്ക് ലഭിക്കുന്നതെങ്കിൽ അതിൽ എഴുതുവാനോ രേഖപ്പെടുത്തുവാനോ നിങ്ങൾക്ക് സാധിക്കും. എന്തുകൊണ്ടെന്നാൽ ഫോട്ടോകോപ്പിയിൽ വരുത്തുന്ന വ്യത്യാസങ്ങൾ യഥാർത്ഥ പട്ടികയെ ബാധിക്കുന്നില്ല.

ക്ഷണക്കത്ത് തയ്യാറാക്കുന്ന ജോലി ഒരു ഫങ്ഷനായി നമുക്ക് പരിഗണിക്കാം. പേര് അടങ്ങുന്ന പട്ടിക ഫങ്ഷൻ ആർഗ്യുമെന്റ് ആണ്. ആർഗ്യുമെന്റ് ഫങ്ഷനിലേക്ക് രണ്ട് രീതിയിൽ അയയ്ക്കാൻ സാധിക്കും. ആദ്യത്തേത് പേര് അടങ്ങുന്ന പട്ടികയുടെ കോപ്പി കൈമാറുക, മറ്റേത്, യഥാർത്ഥ പട്ടികതന്നെ കൈമാറുക. ക്ഷണക്കത്ത് തയ്യാറാക്കുമ്പോൾ യഥാർത്ഥ പട്ടിക തന്നെയാണ് കൈമാറുന്നതെങ്കിൽ പട്ടികയിൽ ഉണ്ടാക്കുന്ന ഏത് മാറ്റവും യഥാർത്ഥ പട്ടികയെ തന്നെ ബാധിക്കും. അതുപോലെ, C++ ലും രണ്ട് രീതിയിൽ ഫങ്ഷനിലേക്ക് ആർഗ്യുമെന്റുകൾ അയയ്ക്കാം. ആർഗ്യുമെന്റുകൾ അയയ്ക്കുന്ന രീതിയെ അവലംബമാക്കി ഫങ്ഷൻ വിളിക്കുന്ന രീതിയെ കാൾ-ബൈ-വാല്യൂ രീതിയെന്നും കാൾ ബൈ റഫറൻസ് രീതിയെന്നും തരംതിരിക്കാം. ആർഗ്യുമെന്റ് കൈമാറ്റ രീതികൾ വിശദമായി താഴെ വിവരിച്ചിരിക്കുന്നു

a. കാൾ-ബൈ-വാല്യൂ (പാസ് ബൈ വാല്യൂ) രീതി (Call by value Method)

ഈ രീതിയിൽ, ആക്ചൽ ആർഗ്യുമെന്റുകളിൽ അടങ്ങിയ വിലകൾ ഫോർമൽ ആർഗ്യുമെന്റുകളിലേക്ക് (Formal argument) അയയ്ക്കുന്നു. മറ്റൊരു വിധത്തിൽ പറഞ്ഞാൽ ആക്ചൽ ആർഗ്യുമെന്റിന്റെ ഒരു പകർപ്പ് ഫങ്ഷനിലേക്ക് അയയ്ക്കുന്നു. അതിനാൽ ഫങ്ഷനകത്ത് യഥാക്രമ ആർഗ്യുമെന്റ് പുതുക്കപ്പെട്ടാലും, വിളിക്കുന്ന ഫങ്ഷനിലെ ആക്ചൽ ആർഗ്യുമെന്റിൽ വ്യത്യാസം പ്രതിഫലിക്കുന്നില്ല. മുൻപ് ചർച്ച ചെയ്ത എല്ലാ ഫങ്ഷനുകളിലും ആർഗ്യുമെന്റുകളുടെ വിലയാണ് അയച്ചത്. താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം കാണുക:

```
void change(int n)
{
     n = n + 1;
     cout << "n = " << n << '\n';
}
int main()
     {
     int x = 20;
     change(x);
     cout << "x = " << x;
     }
```

change () ഫങ്ഷനിലെ n പരാമീറ്ററിന് 20 എന്ന വില ശേഖരിക്കാൻ അതിന്റെ സ്വന്തം മെമ്മറി സ്ഥലം ഉണ്ട്.

x ന്റെ വില change () ഫങ്ഷനിലെ n ലേക്ക് കൈമാറ്റം ചെയ്യപ്പെടുന്നു.

മുകളിലത്തെ പ്രോഗ്രാം ഭാഗത്ത് പരാമർശിച്ചിരിക്കുന്നതു പോലെ, നാം ഒരു ആർഗ്യുമെന്റ് കൈമാറ്റം ചെയ്യുമ്പോൾ വേരിയബിൾ x ന്റെ ഒരു പകർപ്പ് ഫങ്ഷനിലേക്ക് കൈമാറ്റം ചെയ്യപ്പെടുന്നു.

മറ്റൊരു വിധത്തിൽ പറഞ്ഞാൽ x എന്ന വേരിയബിളിന്റെ വിലമാത്രമേ ഫങ്ഷനിലേക്ക് അയയ്ക്കുന്നുള്ളു. അതിനാൽ ഫങ്ഷനിലെ ഫോർമൽ പരാമീറ്ററിന് 20 എന്ന വില ലഭിക്കും. നാം n ന്റെ വില കൂട്ടുകയാണെങ്കിൽ, അത് x എന്ന വേരിയബിളിന്റെ വിലയെ ബാധിക്കുന്നില്ല. ഈ കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
n = 21
x = 20
```

ഒരു ഫങ്ഷൻ, കാൾ-ബൈ-വാല്യൂ രീതിയിൽ വിളിച്ചാൽ ആർഗ്യുമെന്റുകൾക്ക് എന്ത് സംഭവിക്കുമെന്ന് പട്ടിക 10.2 ൽ കാണിക്കുന്നു.

ഫങ്ഷൻ വിളിക്ക് മുൻപ്	ഫങ്ഷൻ വിളിക്ക് ശേഷം	ഫങ്ഷന്റെ പ്രവർത്തനത്തിന് ശേഷം

main()
{
....
....
}
x: 20
	main()
	{
	....
	....
	}
	x: 20
		main()
		{
		....
		....
		}
		x: 20

change(int n)
{
....
}
n:
	change(int n)
	{
	....
	}
	n: 20
		change(int n)
		{
		....
		}
		n: 21

പട്ടിക 10.2 കാൾ ബൈ വാല്യൂ പ്രവർത്തനം.

b. കാൾ ബൈ റഫറൻസ് (പാസ്സ് ബൈ റഫറൻസ്) രീതി (Call by reference (Pass by reference) Method)

ഒരു ആർഗ്യുമെന്റ് റഫറൻസായി അയക്കുമ്പോൾ യഥാർത്ഥ ആർഗ്യുമെന്റിന്റെ റഫറൻസ് (അഡ്രസ്സ്) ഫങ്ഷനിലേക്ക് അയക്കുന്നു. ഇതിന്റെ ഫലമായി യഥാർത്ഥ ആർഗ്യുമെന്റിന് അനുവദിച്ച മെമ്മറിസ്ഥലം യഥാക്രമ ആർഗ്യുമെന്റും കൂടി പങ്കിടും. അതുകൊണ്ട് വിളിച്ച ഫങ്ഷനിലെ യഥാക്രമ ആർഗ്യുമെന്റിന് എന്തെങ്കിലും മാറ്റം സംഭവിച്ചാൽ ആ മാറ്റം ഫങ്ഷനിലെ യഥാർത്ഥ ആർഗ്യുമെന്റിലും പ്രതിഫലിപ്പിക്കും. C++ ൽ ആർഗ്യുമെന്റുകൾ റഫറൻസായി അയക്കുന്നതിന് യഥാക്രമ പരാമീറ്ററായി റഫറൻസ് വേരിയബിൾ ഉപയോഗിക്കുന്നു. ഒരു റഫറൻസ് വേരിയബിൾ മറ്റൊരു വേരിയബിളിന്റെ അപരനാമമാണ്. ഫങ്ഷൻ ഹെഡറിലെ ഡാറ്റ ഇനത്തിനും വേരിയബിളിനും ഇടയിൽ ഒരു ആമ്പർസാന്റ് ചിഹ്നം (&) ഉപയോഗിക്കുന്നു. മറ്റ് വേരിയബിളുകളെപ്പോലെ റഫറൻസ് വേരിയബിളുകൾക്ക് പ്രത്യേകമായ മെമ്മറിസ്ഥലം അനുവദിക്കുന്നില്ല. പകരം യഥാർത്ഥ ആർഗ്യുമെന്റിന് അനുവദിച്ച മെമ്മറി സ്ഥലം പങ്കിടും. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷന്റെ യഥാക്രമം പരാമീറ്ററായി റഫറൻസ് വേരിയബിൾ ഉപയോഗിക്കുന്നു. അതുകൊണ്ട് ഫങ്ഷൻ വിളിക്കായി കാൾബൈ റഫറൻസ് രീതി പ്രാവർത്തികമാക്കുന്നു.

```
void change(int & n)
{
     n = n + 1;
     cout << "n = " << n << '\n';
}
int main()
{
     int x=20;
     change(x);
     cout << "x = " << x;
}
```

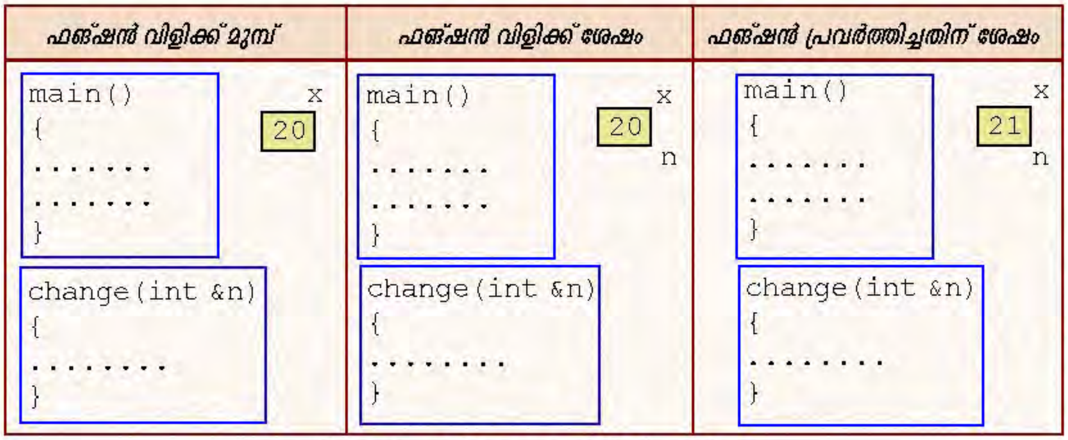
n എന്ന പരാമീറ്റർ ഒരു റഫറൻസ് വേരിയബിൾ ആകുന്നു. അതിനാൽ അതിന് പ്രത്യേകമായ മെമ്മറി അനുവദിക്കുന്നില്ല

x ന്റെ റഫറൻസ് change () ഫങ്ഷനിലെ n ലേക്ക് കൈമാറ്റം ചെയ്യപ്പെടും. തൽഫലമായി മെമ്മറി പങ്കുവയ്ക്കുന്നു.

change() ലെ ഫങ്ഷൻ ഹെഡറിന് മാത്രമേ മാറ്റം ഉള്ളു എന്നത് ശ്രദ്ധിക്കുക. പരാമീറ്റർ n ന്റെ പ്രഖ്യാപനത്തിലെ & ചിഹ്നം അർത്ഥമാക്കുന്നത് ഒരു റഫറൻസ് വേരിബിളാണ് അത് എന്നാണ്. അതുകൊണ്ട് റഫറൻസ് അയച്ചു കൊണ്ട് ഫങ്ഷൻ വിളിക്കപ്പെടും. തന്മൂലം x എന്ന നെ change() ഫങ്ഷനിലേക്ക് അയക്കുമ്പോൾ n ന് x ന്റെ അഡ്രസ് കിട്ടുന്നതിനാൽ അവ മെമ്മറിസ്ഥലം പങ്കിടും. മറ്റൊരു തരത്തിൽ പറഞ്ഞാൽ വേരിയബിളുകളായ n ഉം x ഉം ഒരേ മെമ്മറിസ്ഥലം പരാമർശിക്കുന്നു. നാം main() ഫങ്ഷനിൽ x എന്ന പേരും change() ഫങ്ഷനിൽ n എന്ന പേരും ഉപയോഗിച്ച് ഒരേ മെമ്മറിസ്ഥലം പരാമർശിക്കുന്നു. അതുകൊണ്ട് n ന്റെ വിലയിൽ വ്യത്യാസം വരുത്തുമ്പോൾ യഥാർത്ഥത്തിൽ x വിലയിലാണ് മാറ്റം ഉണ്ടാകുന്നത്. മുകളിലുള്ള പ്രോഗ്രാം പ്രവർത്തിക്കുമ്പോൾ നമുക്ക് താഴെ കൊടുത്തിരിക്കുന്ന ഔട്ട്പുട്ട് ലഭിക്കും.

```
n = 21
x = 21
```

ഫങ്ഷൻ വിളിക്ക് വേണ്ടി കാൾ-ബൈ-റഫറൻസ് ഉപയോഗിക്കുമ്പോൾ ആർഗ്യുമെന്റുകൾക്ക് ഉണ്ടാകുന്ന മാറ്റങ്ങൾ ടേബിൾ 10.3 ൽ വർണ്ണിക്കുന്നു.



പട്ടിക 10.3. കാൾ ബൈ റഫറൻസ് പ്രവർത്തനം

ഫങ്ഷൻ വിളിയുടെ ഈ രണ്ട് രീതികൾ പട്ടിക 10.4 ൽ കാണിച്ചിരിക്കുന്നതു പോലെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു.

കാൾ ബൈ വാല്യൂ രീതി	കാൾ ബൈ റഫറൻസ് രീതി
• യഥാക്രമ പരാമീറ്ററുകളായി സാധാരണ വേരിയബിളുകൾ ഉപയോഗിക്കുന്നു. • സ്ഥിരവിലകൾ, വേരിയബിളുകൾ, അല്ലെങ്കിൽ പദപ്രയോഗങ്ങൾ എന്നിവ യഥാർത്ഥ പരാമീറ്ററുകൾ ആയി ഉപയോഗിക്കാം. • യഥാക്രമ പരാമീറ്ററുകളിൽ ഉണ്ടാക്കുന്ന വ്യത്യാസങ്ങൾ യഥാർത്ഥ പരാമീറ്ററുകളിൽ പ്രതിഫലിക്കുന്നില്ല. • യഥാക്രമ ആർഗ്യുമെന്റുകൾക്ക് പ്രത്യേകം മെമ്മറി ആവശ്യമുണ്ട്.	• യഥാക്രമ പരാമീറ്ററുകളിൽ റഫറൻസ് വേരിയബിളുകൾ ഉപയോഗിക്കുന്നു. • വേരിയബിളുകൾ മാത്രമേ യഥാർത്ഥ പരാമീറ്ററുകൾ ആകൂ. • യഥാക്രമ പരാമീറ്ററിന് ഉണ്ടാക്കുന്ന വ്യത്യാസങ്ങൾ യഥാർത്ഥ പരാമീറ്ററുകളിൽ പ്രതിഫലിക്കും. • യഥാർത്ഥ ആർഗ്യുമെന്റുകളുടെ മെമ്മറി യഥാക്രമ ആർഗ്യുമെന്റുകൾ പങ്കിടുന്നു.

പട്ടിക 10.4. കാൾ ബൈ വാല്യൂ v/s കാൾ ബൈ റഫറൻസ്

കാൾ ബൈ റഫറൻസ് രീതി വിശദമാക്കുന്നതിന് അനുയോജ്യമായ ഒരു ഉദാഹരണം നമുക്ക് ചർച്ച ചെയ്യാം. ഈ പ്രോഗ്രാം `main()` ഫങ്ഷനിലെ രണ്ട് വേരിയബിളുകളുടെ വിലകൾ പരസ്പരം കൈമാറുന്നതിന് കാൾ ബൈ റഫറൻസ് രീതിയിൽ വിളിക്കാൻ കഴിയുന്ന ഒരു ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. രണ്ട് വേരിയബിളുകളുടെ വിലകൾ പരസ്പരം കൈമാറുന്ന പ്രക്രിയയെ സ്വാപ്പിങ്ങ് എന്ന് പറയുന്നു.

പ്രോഗ്രാം 10.8 രണ്ട് വേരിയബിളുകളുടെ വിലകൾ പരസ്പരം കൈമാറുന്നതിന്.

```
#include <iostream>
using namespace std;
void swap(int & x, int & y)
{
    int t = x;
    x = y;
    y = t;
```

```
}
int main()
{
    int m, n;
    m = 10;
    n = 20;
    cout<<"Before swapping m= "<< m <<" and n= "<<n;
    swap(m, n);
    cout<<"\nAfter swapping m= "<< m <<" and n= "<<n;
    return 0;
}
```

നമുക്ക് പ്രോഗ്രാം 10.8 ലെ സ്റ്റേറ്റ്മെന്റുകളിലൂടെ കടന്നു പോകാം. ആക്ചൽ ആർഗ്യുമെന്റുകളായ m ഉം n ഉം ഫങ്ഷനിലേക്ക് റഫൻസ് ആയി അയക്കുന്നു. swap() ഫങ്ഷന് ഉള്ളിൽ x ന്റെയും y യുടേയും വിലകൾ പരസ്പരം കൈമാറ്റം ചെയ്യപ്പെടുന്നു. അപ്പോൾ യഥാർത്ഥത്തിൽ m ലും n ലുമാണ് മാറ്റം നടക്കുന്നത്. അതിനാൽ മുകളിലത്തെ പ്രോഗ്രാം കോഡിന്റെ ഔട്ട്പുട്ട്,

```
    Before swapping m = 10 and n= 20
    After swapping m = 20 and n= 10
```

യഥാക്രമ ആർഗ്യുമെന്റിന് പകരം സാധാരണ വേരിയബിൾ ഉപയോഗിച്ച് മുകളിലെ പ്രോഗ്രാമിൽ മാറ്റം വരുത്തി അതിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുക. കമ്പ്യൂട്ടർ ലാബിൽ ഈ കോഡ് പ്രവർത്തിപ്പിച്ച് നിങ്ങളുടെ ഉത്തരം പരിശോധിക്കുക.

സ്വയം പരിശോധിക്കാം

1. C++ പ്രോഗ്രാമുകളിലെ ഏറ്റവും ഒഴിച്ചു കൂടാനാവാത്ത ഫങ്ഷൻ ഏതെന്ന് തിരിച്ചറിയുക.
2. ഒരു ഫങ്ഷൻ ഹെഡറിന്റെ മൂന്ന് ഘടകങ്ങൾ പട്ടികപ്പെടുത്തുക.
3. ഫങ്ഷൻ പ്രോട്ടോടൈപ്പ് എന്നാൽ എന്ത്?
4. വിളിക്കുന്ന ഫങ്ഷനിൽ നിന്നും വിളിച്ച ഫങ്ഷനിലേക്ക് ഡാറ്റ അയക്കുന്നതിന് ഏത് ഘടകമാണ് ഉപയോഗിക്കുന്നത്.
5. C++ ൽ ഉപയോഗിക്കുന്ന രണ്ട് പരാമീറ്റർ കൈമാറ്റരീതികൾ ഏതൊക്കെയാണ്?

10.5 വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും (Scope and life of variables and functions)

ഒന്നിലധികം ഫങ്ഷനുകൾ അടങ്ങിയ C++ പ്രോഗ്രാം നാം ചർച്ച ചെയ്തു. മുൻ നിർവചിത ഫങ്ഷനുകൾ അതിന് അനുബന്ധമായ ഹെഡർ ഫയലുകൾ ഉൾപ്പെടുത്തിയാണ് പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നത്. ഉപയോക്തൃ നിർമ്മിത ഫങ്ഷനുകൾ main() ഫങ്ഷന് മുമ്പോ ശേഷമോ നിർവ്വചിക്കപ്പെടുന്നു. ഫങ്ഷനുകൾ നിർവ്വചിക്കുമ്പോൾ ഫങ്ഷൻ പ്രോട്ടോടൈപ്പിനുള്ള പ്രസക്തി നാം കണ്ടു കഴിഞ്ഞു. വേരിയബിളുകൾ ഫങ്ഷൻ ചട്ടക്കൂടിലും ആർഗ്യുമെന്റുകളായും നാം ഉപയോഗിച്ചു. ഇനി നമുക്ക് വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും പ്രോഗ്രാമിലുള്ള ലഭ്യതയും പ്രാപ്യതയും ചർച്ച ചെയ്യാം. ഒരു പ്രോഗ്രാമിൽ ലോക്കൽ വേരിയബിളുകളുടെ പ്രാപ്യത പ്രോഗ്രാം 10.9 വിവരിക്കുന്നു.

പ്രോഗ്രാം 10.9. വേരിയബിളുകളുടെ വ്യാപ്തിയും ജീവനവും വിവരിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int cube(int n)
{
    int cb;
    cout<< "The value of x passed to n is " << x;
    cb = n * n * n;
    return cb;
}
int main()
{
    int x, result;
    cout << "Enter a number : ";
    cin >> x;
    result = cube(x);
    cout << "Cube = " << result;
    cout << "\nCube = "<<cb;
}
```

ഇത് തെറ്റ് ആകുന്നു. എന്തുകൊണ്ടെന്നാൽ വേരിയബിൾ x, main() ഫങ്ഷനിലാണ് പ്രഖ്യാപിച്ചത്. അതു കൊണ്ട് മറ്റ് ഫങ്ഷനിൽ അത് ഉപയോഗിക്കാൻ കഴിയില്ല.

ഇത് തെറ്റ് ആകുന്നു എന്തുകൊണ്ടെന്നാൽ cb എന്ന വേരിയബിൾ cube() എന്ന ഫങ്ഷനിലാണ് പ്രഖ്യാപിച്ചത്. അതുകൊണ്ട് മറ്റ് ഫങ്ഷനിൽ അത് ഉപയോഗിക്കാൻ കഴിയില്ല.

നാം പ്രോഗ്രാം കമ്പയിൽ ചെയ്യുമ്പോൾ, കോളൗട്ടിൽ കാണിച്ചിരിക്കുന്ന കാരണങ്ങ ളാൽ രണ്ട് തെറ്റുകൾ ഉണ്ടാകും. വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും ലഭ്യത, പ്രാപ്യത എന്നീ ആശയത്തെ വ്യാപ്തി, ജീവനം, എന്നീ പദങ്ങൾ കൊണ്ട് സൂചിപ്പിക്കുന്നു. പ്രോഗ്രാമിന്റെ ഏത് ഭാഗത്താണോ ഒരു വേരിയബിൾ ഉപയോഗിക്കുന്നത് അതാണ് അതിന്റെ വ്യാപ്തി (Scope) മുകളിലത്തെ പ്രോഗ്രാമിൽ വേരിയബിൾ cb യുടെ വ്യാപ്തി ഫങ്ഷൻ cube() ൽ ആകുന്നു എന്തുകൊണ്ടെന്നാൽ അത് ആ ഫങ്ഷനിലാണ് പ്രഖ്യാപിച്ചത്. അതിനാൽ ഈ വേരിയബിൾ ആ ഫങ്ഷന് പുറത്ത് ഉപയോഗിക്കാൻ കഴിയില്ല. ഈ വ്യാപ്തി **ലോക്കൽ വ്യാപ്തി** എന്ന് അറിയപ്പെടുന്നു. ഒരു ഫങ്ഷന്റെ പ്രവർത്തനം പൂർത്തീകരിക്കുമ്പോൾ ആ ഫങ്ഷന്റെ ഉള്ളിലെ എല്ലാ വേരിയബിളുകൾക്കും അനുവദിച്ച മെമ്മറി സ്വതന്ത്രമാക്കുന്നു. മറ്റൊരു തരത്തിൽ പറയുമ്പോൾ ഒരു ഫങ്ഷന് ഉള്ളിൽ പ്രഖ്യാപിച്ച വേരിയബിളുകളുടെ സമയം ആ ഫങ്ഷനിലെ അവസാനത്തെ നിർദേശം പ്രവർത്തിക്കുന്നതോട് കൂടി അവസാനിക്കുന്നു. അതുകൊണ്ട് main() ഫങ്ഷനിൽ n എന്ന വേരിയബിൾ ഉപയോഗിക്കുമ്പോൾ അതിൽ നിന്നും വിളിക്കുന്ന ഫങ്ഷനിലെ n എന്ന ആർഗ്യുമെന്റിൽ നിന്നും വിളിക്കപ്പെട്ട ഫങ്ഷനിലെ വേരിയബിൾ nൽ നിന്നും അത് വ്യത്യസ്തമായിരിക്കും. ഒരു ഫങ്ഷന് ഉള്ളിൽ പ്രഖ്യാപിച്ച വേരിയബിളുകൾക്കും യഥാക്രമ പരാമീറ്ററുകൾക്കും ലോക്കൽ വ്യാപ്തി ഉണ്ടായിരിക്കും.

വേരിയബിളിനെ പോലെ തന്നെ ഫങ്ഷനുകൾക്കും വ്യാപ്തി ഉണ്ട്. എവിടെ ആണോ ഒരു ഫങ്ഷൻ പ്രഖ്യാപിച്ചിരിക്കുന്നത് അവിടെയാണ് ആ ഫങ്ഷൻ ഉപയോഗിക്കാൻ കഴിയുക. അതായത് ഫങ്ഷന് ലോക്കൽ വ്യാപ്തി ഉണ്ടെന്ന് പറയാം. അത് main() ഫങ്ഷന് മുമ്പേ മറ്റ് ഫങ്ഷനുകൾക്ക് പുറമെയോ പ്രഖ്യാപിച്ചാൽ ആ ഫങ്ഷന്റെ വ്യാപ്തി പ്രോഗ്രാം മുഴുവൻ ആയിരിക്കും. അതായത് പ്രോഗ്രാമിലെ ഏത് സ്ഥലത്തും ഫങ്ഷൻ ഉപയോഗിക്കാൻ കഴിയും. ഈ വ്യാപ്തി **ഗ്ലോബൽ വ്യാപ്തി** എന്ന് അറിയപ്പെടുന്നു. വേരിയബിളുകളും ഇപ്രകാരം ഗ്ലോബൽ വ്യാപ്തിയിൽ പ്രഖ്യാപിക്കാൻ കഴിയും. അങ്ങനെയുള്ള പ്രഖ്യാ

പനങ്ങൾ പ്രോഗ്രാമിലെ എല്ലാ ഫങ്ഷനുകൾക്കും പുറത്ത് ആയിരിക്കും. ഒരു പ്രോഗ്രാമിലെ വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും, ജീവനവും സംബന്ധിച്ച് കൂടുതൽ വ്യക്തത കിട്ടുവാൻ പ്രോഗ്രാം 10.10 നോക്കുക.

പ്രോഗ്രാം 10.10 വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും വിവരിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int cb;          //global variable
void test()//global function since defined above other functions
{
    int cube(int n); //It is a local function
    cb=cube(x); //Invalid call. x is local to main()
    cout<<cb;
}
int main() // beginning of main() function
{
    int x=5; //local variable
    test(); //valid call since test() is a global function
    cb=cube(x); //Invalid call. cube() is local to test()
    cout<<cb;
}
int cube(int n)//Argument n is local variable
{
    int val= n*n*n; //val is local variable
    return val;
}
```

വ്യാപ്തിയും ജീവനും	ലോക്കൽ	ഗ്ലോബൽ
വേരിയബിളുകൾ	• ഒരു ഫങ്ഷൻ അല്ലെങ്കിൽ പ്രസ്താവനകളുടെ കൂട്ടത്തിന് ഉള്ളിൽ പ്രഖ്യാപിക്കുന്നു. • ആ ഫങ്ഷൻ അല്ലെങ്കിൽ ബ്ലോക്കിൽ മാത്രമേ ലഭ്യമാകൂ. • ഫങ്ഷൻ അല്ലെങ്കിൽ ബ്ലോക്ക് പ്രവർത്തിക്കുമ്പോൾ മെമ്മറി അനുവദിക്കുകയും ഫങ്ഷന്റെയോ ബ്ലോക്കിന്റെയോ പ്രവർത്തനം പൂർത്തിയാകുമ്പോൾ അവ സ്വതന്ത്രമാക്കുകയും ചെയ്യുന്നു.	• എല്ലാ ഫങ്ഷന്റെയും പുറമെ പ്രഖ്യാപിക്കുന്നു. • പ്രോഗ്രാമിലെ എല്ലാ ഫങ്ഷനുകൾക്കും ലഭ്യമാണ്. • പ്രോഗ്രാമിന്റെ പ്രവർത്തനം തുടങ്ങുന്നതിന് തൊട്ട് മുൻപേ മെമ്മറി അനുവദിക്കുകയും പ്രോഗ്രാം അവസാനിക്കുമ്പോൾ അവ സ്വതന്ത്രമാവുകയും ചെയ്യുന്നു.
ഫങ്ഷനുകൾ	• ഒരു കൂട്ടം പ്രസ്താവനകൾക്ക് ഉള്ളിലോ ഫങ്ഷന് ഉള്ളിലോ പ്രഖ്യാപിക്കുകയും വിളിക്കുന്ന ഫങ്ഷന് ശേഷം നിർവ്വചിക്കുകയും ചെയ്യുന്നു. • ആ ഫങ്ഷൻ അല്ലെങ്കിൽ ബ്ലോക്കിന് ഉള്ളിൽ മാത്രമേ പ്രാപ്യമാകൂ.	• മറ്റ് എല്ലാ ഫങ്ഷനുകളുടേയും വെളിയിൽ പ്രഖ്യാപിക്കുകയോ നിർവ്വചിക്കുകയോ ചെയ്യുന്നു. • പ്രോഗ്രാമിലെ എല്ലാ ഫങ്ഷനുകൾക്കും പ്രാപ്യമാണ്.

ടേബിൾ 10.5 : വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും

തന്നിരിക്കുന്ന കാൾ ഔട്ടുകൾ ഫങ്ഷനുകളുടെ വ്യാപ്തിയും ജീവനവും വിശദമാക്കുന്നു. ഒരു ഫങ്ഷൻ മറ്റൊരു ഫങ്ഷന്റെ ചട്ടക്കൂടിനുള്ളിൽ പ്രഖ്യാപിക്കുകയാണെങ്കിൽ അതിനെ ലോക്കൽ ഫങ്ഷൻ എന്ന് വിളിക്കുന്നു. അത് ആ ഫങ്ഷന് ഉള്ളിൽ മാത്രമേ ഉപയോഗിക്കാൻ കഴിയൂ. ഒരു ഫങ്ഷൻ മറ്റെല്ലാ ഫങ്ഷനുകളുടേയും ചട്ടക്കൂടിന് പുറമെ പ്രഖ്യാപിച്ചാൽ അതിനെ ഗ്ലോബൽ ഫങ്ഷൻ എന്ന് വിളിക്കുന്നു. ഒരു ഗ്ലോബൽ ഫങ്ഷൻ പ്രോഗ്രാമിൽ ഉടനീളം ഉപയോഗിക്കാൻ കഴിയും. മറ്റൊരു തരത്തിൽ പറഞ്ഞാൽ ഒരു ഗ്ലോബൽ ഫങ്ഷന്റെ വ്യാപ്തി പ്രോഗ്രാം മുഴുവനാകുമ്പോൾ ലോക്കൽ ഫങ്ഷന്റെ വ്യാപ്തി അത് പ്രഖ്യാപിച്ചിരിക്കുന്ന ഫങ്ഷനിൽ മാത്രം ആയിരിക്കും. വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും ടേബിൾ 10.6 ൽ സംഗ്രഹിച്ചിരിക്കുന്നു.

10.6 സ്വയം ആവർത്തിക്കുന്ന ഫങ്ഷനുകൾ (Recursive Function)

സാധാരണയായി ഒരു ഫങ്ഷൻ മറ്റൊരു ഫങ്ഷനെയാണ് വിളിക്കുന്നത്. ഇനി നമുക്ക് അതിനെ തന്നെ വിളിക്കുന്ന ഒരു ഫങ്ഷൻ നിർവ്വചിക്കാം. ഒരു ഫങ്ഷൻ ആ ഫങ്ഷനെ തന്നെ വിളിക്കുന്ന പ്രവർത്തനത്തെ സ്വയം ആവർത്തനം എന്ന് അറിയപ്പെടുന്നു. ആ ഫങ്ഷനെ സ്വയം ആവർത്തിക്കുന്ന ഫങ്ഷൻ എന്നും പറയുന്നു. സങ്കീർണ്ണമായ ചില അൽഗോരിതങ്ങൾ സ്വയം ആവർത്തന രീതി ഉപയോഗിച്ച് വളരെ എളുപ്പത്തിൽ ലഘൂകരിക്കാം.

```
int function1()
{
     ............
     ............
     int n = function1();    //calling itself
     ............
     ............
}
```

ഒരു സ്വയം ആവർത്തി ഫങ്ഷന്റെ മാതൃക താഴെ പറയുന്നത് പോലെ ആകും.

സ്വയം ആവർത്തിത ഫങ്ഷനുകളിൽ സാധാരണയായി ചില ഉപാധിയെ അടിസ്ഥാനമാക്കിയായിരിക്കും ഫങ്ഷൻ വിളിക്കുന്നത്. താഴെ കൊടുത്തിരിക്കുന്ന സ്വയം ആവർത്തിത ഫങ്ഷനിലൂടെ ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കാം. നെഗറ്റീവ് സംഖ്യയുടെ ഫാക്ടോറിയൽ നിർവ്വചിച്ചിട്ടില്ല എന്നത് ശ്രദ്ധിക്കുക. അതുകൊണ്ട് n ന് പൂജ്യമോ പോസിറ്റീവ് സംഖ്യയോ കൊടുക്കാൻ കഴിയും.

```
int factorial(int n)
{
     if((n==1)||(n==0))
          return 1;
     else if (n>1)
          return (n * factorial(n-1));
     else
          return 0;
}
```

ഉപയോക്താവ് ഈ ഫങ്ഷൻ വിളിക്കുമ്പോൾ അത് എങ്ങനെ പ്രവർത്തിക്കുമെന്ന് നമുക്ക് ചർച്ച ചെയ്യാം.

```
f = factorial(3);
```

ഫങ്ഷൻ ആദ്യത്തെ തവണ വിളിക്കുമ്പോൾ, n വില 3 ആകുന്നു. if പ്രസ്താവനയിലെ വ്യവസ്ഥ വിലയിരുത്തുമ്പോൾ വില തെറ്റ് എന്ന് കണക്കാക്കുന്നു. അതുകൊണ്ട് else ചട്ടക്കൂട് പ്രവർത്തിക്കും. n ന്റെ വില 3 ആകുമ്പോൾ else ബ്ലോക്കിലെ നിർദ്ദേശം

`return (3 * factorial(3-1));` എന്ന് ആകും.

അത് ലഘൂകരിക്കുമ്പോൾ `return (3 * factorial(2));`........... (i), `3 * factorial(2)`, കണ്ടുപിടിക്കുന്നതിന് വേണ്ടി `factorial(2)` ന്റെ വില കണ്ടുപിടിക്കേണ്ടതാവശ്യമാണ്. `factorial()` എന്ന ഫങ്ഷൻ വീണ്ടും 2 എന്ന ആർഗ്യുമെന്റ് ഉപയോഗിച്ച് വിളിക്കുന്നു. ഫങ്ഷൻ അതേ ഫങ്ഷനുള്ളിൽ തന്നെ വിളിക്കുന്നു. n എന്ന പരാമീറ്ററിന്റെ വിലയായ 2 കൊണ്ട് `factorial()` ഫങ്ഷൻ പ്രവർത്തിക്കുമ്പോൾ if ബ്ലോക്കിലെ വ്യവസ്ഥ തെറ്റ് ആയി കണക്കാക്കുന്നു, അതുകൊണ്ട് else ബ്ലോക്ക് മാത്രമേ പ്രവർത്തിക്കൂ. else ബ്ലോക്കിലെ നിർദ്ദേശം return (2 * factorial(2-1)); എന്നാകുന്നു.

ഇത് ലഘൂകരിക്കുമ്പോൾ `return (2 * factorial(1));` .... (ii)

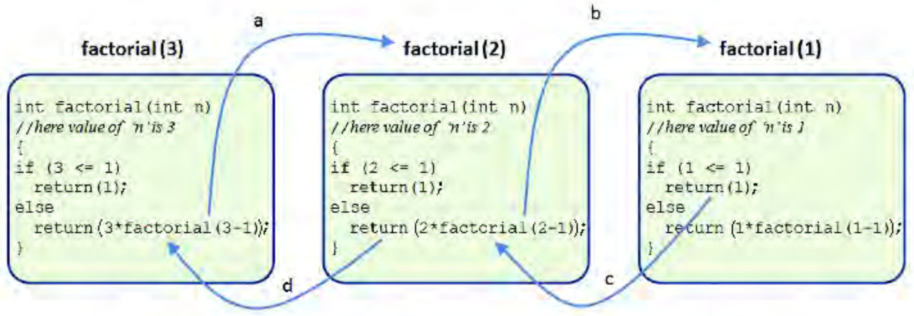
ഇതിന്റെ തിരിച്ചു നൽകുന്ന വില കണ്ടുപിടിക്കുന്നതിനായി അത് n എന്ന പരാമീറ്ററിന്റെ വിലയായി 1 നൽകിക്കൊണ്ട് വീണ്ടും `factorial()` ഫങ്ഷൻ വിളിക്കുന്നു. ഇപ്പോൾ if സ്റ്റേറ്റ്മെന്റിന്റെ വ്യവസ്ഥ ശരി (true) ആയി കണക്കാക്കുന്നതിനാൽ `factorial(1)` എന്ന ഫങ്ഷൻ കാളിന്റെ വില ആയി 1 എന്ന് തിരിച്ചു നൽകും. ഇപ്പോൾ പ്രോഗ്രാമിന് (ii) എന്ന് രേഖപ്പെടുത്തിയ നിർദ്ദേശത്തിന്റെ തിരിച്ചു നൽകുന്ന വില കണ്ടുപിടിക്കാൻ കഴിയും. നിർദ്ദേശം (ii). `return 2*1` എന്ന് ആകും. അത് `return 2` ന് തുല്യമായിരിക്കും.

അതായത് `factorial (2)` എന്ന ഫങ്ഷൻ തിരിച്ചു നൽകിയ 2 എന്ന വില (i) എന്ന് രേഖപ്പെടുത്തിയ നിർദ്ദേശത്തിലേക്ക് നൽകുന്നു. അതിനാൽ നിർദ്ദേശം (i) `return 3 x 2` എന്ന് ആകുന്നു. അതിന് സമമാണ് `return 6;`

ഇപ്പോൾ `factorial (3)` എന്ന ഫങ്ഷൻ വിളിയുടെ തിരിച്ചു നൽകുന്ന വിലയായി 6 ലഭിക്കുന്നു.

`f = factorial (3)` എന്ന നിർദ്ദേശത്തിന്റെ പ്രവർത്തനം ചിത്രം 10.4 ൽ സംഗ്രഹിക്കുന്നു.

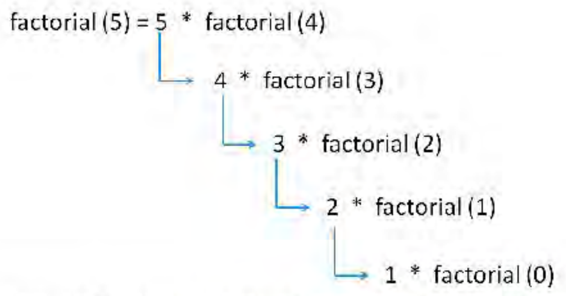
`factorial (3)` എന്ന ഫങ്ഷൻ വിളിയുടെ പ്രവർത്തനം `factorial(2)` എന്ന ഫങ്ഷന്റെ വില ലഭിക്കുന്നത് വരെ താമസിക്കുന്നു. ഫങ്ഷന്റെ പ്രവർത്തനം `factorial (1)` ന്റെ വില കിട്ടുന്നത് വരെ താമസിക്കും. ഒരിക്കൽ ഈ ഫങ്ഷൻ കാളിന് 1 എന്ന തിരിച്ച് നൽകുന്ന വിലയായി ലഭിച്ചാൽ അത് വിലയെ മുൻപ് നടന്ന ഫങ്ഷൻ വിളികളിലേക്ക് തിരികെ നൽകും. ഫങ്ഷന്റെ ഓരോ വിളിയിലും ആർഗ്യുമെന്റുകൾക്കും തിരിച്ച് നൽകുന്ന വിലകൾക്കും എന്ത് സംഭവിക്കുന്നു എന്നത് ചിത്രം 10.4 ൽ കാണിക്കുന്നു.



ചിത്രം 10. 4; ഒരു സ്വയം ആവർത്തിത ഫങ്ഷൻ വിളിയുടെ നിയന്ത്രണ ഗതി

ഒരു ന്യൂനസംഖ്യ ആർഗ്യുമെന്റ് ആയി `factorial()` എന്ന ഫങ്ഷൻ വിളിക്കുമ്പോൾ ഫങ്ഷൻ പൂജ്യം തിരികെ നൽകും എന്നത് ശ്രദ്ധിക്കുക. ഒരു നെഗറ്റീവ് സംഖ്യയുടെ ഫാക്ടോറിയൽ പൂജ്യം ആണെന്ന് ഇതിന് അർത്ഥമില്ല. ഗണിതത്തിൽ ഒരു ന്യൂനസംഖ്യയുടെ ഫാക്ടോറിയൽ നിർവചിച്ചിട്ടില്ല. ഉദാഹരണത്തിന് `factorial(-3)` എന്ന ഫങ്ഷൻ വിളി അസാധുവായ വിളിയാണെന്നത് വിളിക്കുന്ന ഫങ്ഷനിൽ രേഖപ്പെടുത്തും.

`factorial()` എന്ന ഫങ്ഷൻ 5 എന്ന വില ഉപയോഗിച്ച് വിളിക്കുമ്പോൾ ഉണ്ടാകുന്ന പ്രവർത്തനം ചിത്രം 10.5 ൽ കാണിക്കുന്നു.



ചിത്രം 10.5 : factorial (5) ന്റെ സ്വയം ആവർത്തന പ്രവർത്തനം

ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കുന്നതിനായി സ്വയം ആവർത്തിത ഫങ്ഷനല്ലാതെ താഴെ കൊടുക്കുന്ന ഫങ്ഷനും ഉപയോഗിക്കാം.

```
int factorial(int n)
{
    int f=1;
    /*  The formula n×(n-1)×(n-2)× ... ×2×1 is appliled
    instead of 1 × 2 × 3× ... ×(n-1)×n to find the factorial
    */
    for(int i=n; i>1; i--)
        f *= i;
    return f;
}
```

ഈ രണ്ട് ഫങ്ഷനുകൾ തമ്മിലുള്ള വ്യത്യാസം നമുക്ക് താരതമ്യം ചെയ്യാം.

റിക്കർഷൻ ഉപയോഗിക്കുന്ന എല്ലാ ഫങ്ഷനുകളും റിക്കർഷൻ ഉപയോഗിക്കാതെയും എഴുതാം. പിന്നെ എന്തിന് വേണ്ടിയാണ് നാം ആവർത്തനം ഉപയോഗിക്കുന്നത്? ചില പ്രോഗ്രാമർമാർക്ക് ആവർത്തനം ഉപയോഗിക്കുന്നതാണ് മറ്റേതിനേക്കാൾ വളരെ ലളിതമാണ്. എല്ലാ ഫങ്ഷനുകളിലും സ്വയം ആവർത്തന രീതി ഉപയോഗിക്കാൻ കഴിയില്ല.

ഒരു ഫങ്ഷനിൽ സ്വയം ആവർത്തനം നടത്തുവാൻ കഴിയുമോ ഇല്ലയോ എന്ന് എങ്ങനെ നമുക്ക് മനസ്സിലാക്കാം. ഒരു ഫങ്ഷനകത്ത് അതേ ഫങ്ഷന്റെ തന്നെ ഔട്ട്പുട്ടിൽ ചില പ്രവർത്തനങ്ങൾ നടത്തുവാൻ നമുക്ക് കഴിയുമെങ്കിൽ അത്തരം ഫങ്ഷനിൽ സ്വയം ആവർത്തനം ഉപയോഗിക്കാം. ഉദാഹരണത്തിന് ആദ്യത്തെ n എണ്ണൽ സംഖ്യകളുടെ തുക കാണുന്നതിന് നമുക്ക് `sum(n)` എന്ന ഫങ്ഷൻ താഴെ കൊടുത്തിരിക്കുന്നതു പോലെ എഴുതുവാൻ കഴിയും.

```
sum(n) = n + sum(n-1)
```

തന്നിരിക്കുന്ന ഒരു ദശ സംഖ്യയെ അതിന് തുല്യമായ ബൈനറി സംഖ്യയാക്കി മാറ്റുന്ന പ്രോഗ്രാം നമുക്ക് ചർച്ച ചെയ്യാം. അധ്യായം 2 ൽ പരിവർത്തന രീതി നാം ചർച്ച ചെയ്തു.

പ്രോഗ്രാം 10.11. ഒരു ഡസിമൽ നമ്പറിന് (ദശാംശ സംഖ്യയ്ക്ക്) തുല്യമായ ബൈനറി സംഖ്യ പ്രദർശിപ്പിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
void Binary(int);
int main()
{
    int decimal;
    cout<<"Enter an integer number: ";
    cin>>decimal;
    cout<<"Binary equivalent of "<<decimal<<" is ";
    Binary(decimal);
    return 0;
}

void Binary(int n)    //Definition of a recursive function
{
    if (n>1)
        Binary(n/2);
    cout<<n%2;
}
```

പ്രോഗ്രാം 10.11 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

`Enter an integer number: 19` ചെറിയ സംഖ്യ `input` നൽകി കൊണ്ട് പ്രോഗ്രാമിന്റെ പ്രവർത്തനം വിശദീകരിക്കുക.

`Binary equivalent of 19 is 10011`

10.7 ഹെഡർ ഫയലുകളുടെ നിർമ്മാണം

ഇതുവരെ നാം ചർച്ച ചെയ്ത എല്ലാ പ്രോഗ്രാമുകളുടേയും തുടക്കത്തിൽ `#include <iostream>` എന്നത് എല്ലായ്പ്പോഴും ഉപയോഗിക്കുന്നു എന്ന് നമുക്കറിയാം. നാം ഈ പ്രസ്താവന ഉപയോഗിക്കുന്നത് എന്തിന് വേണ്ടിയാണ്?

യഥാർത്ഥത്തിൽ C++ പ്രോഗ്രാമുകളിൽ നാം ഉപയോഗിക്കുന്ന ധാരാളം വേരിയബിളുക ളുടേയും ഒബ്ജക്ടുകളുടേയും പ്രഖ്യാപനങ്ങളും നിർവചനങ്ങളും iostream എന്ന ഹെഡർ ഫയലിൽ അടങ്ങിയിരിക്കുന്നു. പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്ന cin, cout എന്നീ ഒബ്ജക്ടുകൾ ഈ ഹെഡൽ ഫയലിലാണ് പ്രഖ്യാപിച്ചിരിക്കുന്നത്. അതിനാൽ ഒരു പ്രോഗ്രാമിൽ ഈ ഹെഡർ ഫയൽ ഉൾപ്പെടുത്തുമ്പോൾ ഒബ്ജക്ടുകളുടെയും ഫങ്ഷനുകളുടെയും നിർവ്വചനങ്ങളും പ്രഖ്യാപനങ്ങളും കമ്പയിലറിന് കമ്പയിലേഷൻ സമയത്ത് ലഭ്യമാകും. *ഈ ഫങ്ഷനുകളുടേയും ഒബ്ജക്ടുകളുടേയും പ്രവർത്തിക്കാവുന്ന കോഡ് പ്രോഗ്രാമുമായി ബന്ധിപ്പിക്കുകയും അവ എപ്പോൾ എവിടെ വച്ച് വിളിക്കുമ്പോളും അവ പ്രവർത്തിക്കും.* നമ്മുടെ സ്വന്തം വേരിയബിളുകളും ഫങ്ഷനുകളും ഉൾക്കൊള്ളുന്ന ഹെഡൽ ഫയലുകൾ ഇതുപോലെ നമുക്ക് നിർമ്മിക്കാൻ കഴിയും. ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കാണുന്നതിനുള്ള ഒരു ഫങ്ഷൻ എഴുതി, ആ ഫങ്ഷൻ പല പ്രോഗ്രാമിലും ഉപയോഗിക്കണം എന്ന് കരുതുക. എല്ലാ പ്രോഗ്രാമുകളിലും ഫാക്ടോറിയൽ ഫങ്ഷൻ നിർവചിക്കുന്നതിന് പകരം ആ ഫങ്ഷനെ ഒരു ഹെഡർ ഫയലിൽ സ്ഥാപിക്കുകയും ഈ ഹെഡർ ഫയൽ എല്ലാ പ്രോഗ്രാമിലും ഉൾപ്പെടുത്തുകയും ചെയ്യാം.

നമുക്ക് ഒരു ഹെഡർ ഫയൽ എങ്ങനെ നിർമ്മിക്കാൻ കഴിയും എന്നത് താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം കാണിക്കുന്നു. ഏതെങ്കിലും IDE എഡിറ്ററിൽ താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കൊടുക്കുക.

```
int factorial(int n)
{
    int f=1;
    for (int i=1; i<=n; i++)
        f *= i;
    return f;
}
```

ഈ ഫയൽ **factorial.h** എന്ന പേരിൽ സേവ് ചെയ്യുകയും അതിനുശേഷം താഴെ കൊടുത്തിരിക്കുന്നത് പോലെ ഒരു C++ പ്രോഗ്രാം നിർമിക്കുകയും ചെയ്യുക.

```
#include <iostream>
#include "factorial.h"//includes user-defined headerfile
using namespace std;
int main()
```

```
{
    int n;
    cout<<"Enter a number : ";
    cin >> n;
    cout<<"Factorial : " << factorial(n);
}
```

നമുക്ക് പ്രോഗ്രാം വിജയകരമായി കമ്പയിൽ ചെയ്യുവാനും പ്രവർത്തിക്കുവാനും കഴിയും.

`#include "factorial.h"` എന്ന നിർദ്ദേശം ആൻഗുലാർ ബ്രാക്കറ്റുകൾക്ക് (<,>) പകരം ഡബിൾ ക്വോട്ട്സ് ആണ് ഉപയോഗിച്ചത് എന്നത് ശ്രദ്ധിക്കുക. ഇത് എന്തുകൊണ്ടെന്നാൽ ഒരു ഫയൽ ഉൾപ്പെടുത്തുന്നതിനായി നാം ആൻകുലാർ ബ്രാക്കറ്റുകൾ (< >) ഉപയോഗിക്കുമ്പോൾ കമ്പയിൽ അതിനെ `include` ഡയറക്ടറിയിൽ പരതും. എന്നാൽ നാം ഡബിൾ ക്വോട്സ് ഉയോഗിക്കുമ്പോൾ അതിനെ ഇപ്പോൾ പ്രവർത്തിക്കുന്ന ഡയറക്ടറിയിൽ മാത്രമേ പരതുകയുള്ളൂ. സാധാരണയായി `factorial.h` എന്ന ഫയൽ സേവ് ചെയ്യുന്നത് C++ പ്രോഗ്രാം സേവ് ചെയ്യപ്പെട്ട അതേ ഡയറക്ടറിയിലായിരിക്കും. അതുകൊണ്ട് ഫയൽ ഉൾപ്പെടുത്താൻ നാം ഉദ്ധരണി ഉപയോഗിക്കണം. ഏതെങ്കിലും ഒരു C++ പ്രോഗ്രാമിന് ഫാക്ടോറിയൽ ഫങ്ഷൻ ഉപയോഗിക്കേണ്ട ആവശ്യം ഉണ്ടെങ്കിൽ `#include" factorial.h"` എന്ന സ്റ്റേറ്റ്മെന്റ് പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ട ആവശ്യം മാത്രമേ ഉള്ളൂ. ഇതേ രീതിയിൽ ഒരു ഹെഡർ ഫയലിൽ എത്ര ഫങ്ഷനുകൾ വേണമെങ്കിലും ഉൾപ്പെടുത്തുകയും ഏത് പ്രോഗ്രാമിലും ഈ ഫങ്ഷനുകൾ ഉപയോഗിക്കുകയും ചെയ്യാം.

സ്വയം പരിശോധിക്കാം

1. ഒരു ഫങ്ഷന്റെ പ്രോട്ടോടൈപ്പ് പ്രീ. പ്രോസസർ ഡിറക്ടീവിന് ശേഷം നൽകുമ്പോൾ അവയുടെ വ്യാപ്തി .... ആയിരിക്കും.
2. സ്വയം ആവർത്തനം എന്നാൽ എന്ത്?
3. C++ ലെ മുൻ നിർവ്വചിത ഫങ്ഷനുകളുടെ വ്യാപ്തി എന്ത് ആകുന്നു.
4. ഒരു ഫങ്ഷന്റെ ആർഗ്യുമെന്റുകൾക്ക് .... വ്യാപ്തി ഉണ്ട്.

നമുക്ക് സംഗ്രഹിക്കാം

പ്രോഗ്രാമിങ്ങ് എളുപ്പത്തിൽ ആക്കുന്ന ഒരു സമീപനമാണ് മോഡുലാർ പ്രോഗ്രാമിങ്. C++ ഫങ്ഷനിലൂടെ മോഡുലറൈസേഷന് സൗകര്യം ഒരുക്കുന്നു. ഒരു പ്രത്യേക ഉദ്യമം നടത്തുന്നതിന് പ്രോഗ്രാമിൽ ഉൾക്കൊള്ളിച്ചിരിക്കുന്ന പേരോടു കൂടിയ ഒരു ഘടകമാണ് ഫങ്ഷൻ. C++ൽ മുൻ നിർവചിത ഉപയോക്തൃ നിർവചിത എന്നീ രണ്ട് തരം ഫങ്ഷനുകൾ ഉണ്ട്. മുൻ നിർവചിത ഫങ്ഷനുകൾ ഉപയോഗിക്കണമെങ്കിൽ അതുമായി ബന്ധപ്പെട്ട ഹെഡൽ ഫയൽ നാം പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തണം. വിളിക്കുന്ന ഫങ്ഷന് ശേഷമാണ് നിർവ്വചിച്ചിരിക്കുന്നതെങ്കിൽ അത്തരം ഫങ്ഷൻ പ്രഖ്യാപിക്കേണ്ടത് ആവശ്യമാണ്. ഫങ്ഷൻ വിളിക്കുമ്പോൾ വിളിക്കുന്ന ഫങ്ഷനിൽ നിന്നും വിളിച്ച ഫങ്ഷനിലേക്ക് ഡാറ്റ ആർഗ്യുമെന്റിലൂടെ അയച്ചേക്കാം. ആർഗ്യൂമെന്റുകളെ യഥാക്രമ (ഫോർമൽ) പരാമീറ്റർ, ആച്വൽ (യഥാർത്ഥ) പരാമീറ്റർ എന്നിങ്ങനെ രണ്ടായി തരംതിരിക്കാം. ഫങ്ഷനിലേക്ക് പരാമീറ്റർ അയയ്ക്കുന്നതിന് കാൾ-ബൈ-വാല്യൂ രീതിയോ അല്ലെങ്കിൽ കാൾ ബൈ റഫറൻസ് രീതിയോ ഉപയോഗിക്കാം. ഒരു പ്രോഗ്രാമിലെ വേരിയബിളുകൾക്കും ഫങ്ഷനുകൾക്കും അവ പ്രഖ്യാപിച്ചിരിക്കുന്ന സ്ഥലത്തിനനുസരിച്ച് വ്യാപ്തിയും ജീവനവും ഉണ്ട്. ഒരു ഫങ്ഷൻ, മറ്റൊരു ഫങ്ഷനെ വിളിക്കാമെന്നതു പോലെ, ഒരു ഫങ്ഷൻ അതിനെ തന്നെ വിളിക്കുന്ന പ്രവർത്തനമായ സ്വയം ആവർത്തനവും C++ അനുവദിക്കുന്നു. ഉപയോക്തൃ നിർവചിത ഫങ്ഷനുകൾ ശേഖരിക്കാൻ പുതിയ ഹെഡർഫയലുകളും നമുക്ക് നിർമിക്കാൻ കഴിയും. അതിലൂടെ ഈ ഫങ്ഷനുകൾ മറ്റ് പ്രോഗ്രാമുകളിലും ഉപയോഗിക്കാം.

പഠന നേട്ടങ്ങൾ

ഈ അധ്യായം പൂർത്തിയാക്കിയതിന് ശേഷം പഠിതാവിന് പ്രാപ്തമാകുന്നത്

- മോഡുലാർ പ്രോഗ്രാമിങ്ങ് ശൈലിയും അവയുടെ മേന്മകളും തിരിച്ചറിയുന്നു.
- പ്രശ്നപരിഹാരത്തിനായി മുൻ നിർവചിത ഫങ്ങ്ഷനുകൾ ഉപയോഗിക്കുന്നു.
- പ്രശ്നപരിഹാരത്തിൽ ഏർപ്പെട്ടിരിക്കുന്ന പ്രത്യേക ഉദ്യമങ്ങൾ പ്രവർത്തിക്കുന്നതിനായി ഉപഫങ്ഷനുകൾ നിർവ്വചിക്കുന്നു.
- ഉപയോക്താവ് നിർവ്വചിച്ച ഉപഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നു.
- സ്വയം ആവർത്തന ഫങ്ഷനുകൾ നിർവ്വചിക്കുകയും പ്രശ്ന പരിഹാരത്തിന് അവ ഉപയോഗിക്കുകയും ചെയ്യുന്നു.

ലാബ് പ്രവർത്തനങ്ങൾ

1. ഒരു സംഖ്യ സ്വീകരിച്ച് അത് അഭിഭാജ്യം ആണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ 0 ഉം തിരിച്ച് നൽകുന്നതിനുള്ള ഒരു ഫങ്ഷൻ നിർവ്വചിക്കുക. ഈ ഫങ്ഷൻ ഉപയോഗിച്ച് 100 നും 200 നും ഇടക്കുള്ള എല്ലാ അഭാജ്യ സംഖ്യകളും പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.
2. ഒരു ഫങ്ഷൻ ഉപയോഗിച്ച് തന്നിരിക്കുന്ന മൂന്ന് അല്ലെങ്കിൽ രണ്ട് സംഖ്യകളിൽ ഏറ്റവും ചെറിയ സംഖ്യ കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക (തനത് ആർഗ്യുമെന്റുകളുടെ ആശയം ഉപയോഗിക്കുക).
3. ഒരു ഉപയോക്തൃ നിർവചിത ഫങ്ഷന്റെ സഹായത്തോടെ ഒരു സംഖ്യയിലെ അക്കങ്ങളുടെ തുക കണ്ടുപിടിക്കുക. (അതായത് സംഖ്യ 3245 ആണെങ്കിൽ, ഉത്തരം 3+2+4+5 = 14 ആയിരിക്കണം).
4. ഒരു ഫങ്ഷൻ ഉപയോഗിച്ച് തന്നിരിക്കുന്ന രണ്ട് സംഖ്യകളുടെ LCM കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.
5. ഒരു ഫങ്ഷൻ ഉപയോഗിച്ച് തന്നിരിക്കുന്ന പരിധിയിൽപ്പെട്ട എല്ലാ പാലിൻഡ്രോം നമ്പറുകളും പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. ഫങ്ഷൻ സംഖ്യ സ്വീകരിക്കുകയും പാലിൻഡ്രോം ആണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ പൂജ്യവും തിരിച്ചു നൽകണം.

മാതൃകാ ചോദ്യങ്ങൾ

ലഘുചോദ്യോത്തരങ്ങൾ

1. പ്രോഗ്രാമിങ്ങിൽ ടോപ്-ഡൗൺ, ബോട്ടം-അപ്പ് രൂപകൽപ്പനകൾ എങ്ങനെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു?
2. C++ ലെ ഒരു ഫങ്ഷൻ എന്താണ്?
3. ഒരു ഫങ്ഷന് അതിനെ തന്നെ വിളിക്കാനുള്ള കഴിവിനെ .... എന്ന് പറയുന്നു.
4. C++ പ്രോഗ്രാമുകളിൽ ഹെഡർ ഫയലുകളുടെ കർത്തവ്യം എഴുതുക.
5. ഫങ്ഷൻ നിർവചനത്തിനായി വോയിഡ് (void) ഡാറ്റ തരം ഉപയോഗിക്കുന്നത് എപ്പോഴാണ്?

ലഘു ഉപന്യാസതരം :

1. ആക്ചൽ (യഥാർത്ഥ) ഫോർമൽ (യഥാക്രമം) എന്നീ ആർഗ്യുമെന്റുകൾ തമ്മിലുള്ള വ്യത്യാസങ്ങൾ കണ്ടെത്തുക.
2. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷനുകൾക്ക് വേണ്ട ഫങ്ഷൻ പ്രോട്ടോടൈപ്പുകൾ നിർമിക്കുക.

   a) Total () - രണ്ട് ഡബിൾ ആർഗ്യുമെന്റുകൾ സ്വീകരിച്ച് ഒരു ഡബിൾ ഡാറ്റതരം തിരിച്ച് നൽകുന്നു

b) Math () - ഒരു വിലയും സ്വീകരിക്കുകയോ തിരിച്ചു നൽകുകയോ ചെയ്യുന്നില്ല.

3. റിട്ടേൺ പ്രസ്താവന, exit() ഫങ്ഷൻ എന്നിവ വേർതിരിച്ച് എഴുതുക.
4. ലോക്കൽ, ഗ്ലോബൽ എന്നീ വേരിയബിളുകളുടെ വ്യാപ്തി ഉദാഹരണ സഹിതം ചർച്ച ചെയ്യുക.
5. ഫങ്ഷനെ വിളിക്കുന്നതിന് ഉപയോഗിക്കുന്ന കാൾ-ബൈ-വാല്യൂ രീതിയും കാൾ-ബൈ-റഫറൻസ് രീതിയും തമ്മിലുള്ള വ്യത്യാസങ്ങൾ കണ്ടെത്തുക.
6. C++ ൽ എല്ലാം ആർഗ്യുമെന്റുകളും ഉപയോഗിക്കാതെ ഫങ്ഷൻ വിളിക്കുവാൻ കഴിയും. എങ്ങനെ?
7. സ്വയം ആവർത്തനത്തിൽപ്പെട്ടിരിക്കുന്ന പ്രവർത്തനം എഴുതുക.

ദീർഘ ചോദ്യോത്തരം

1. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷൻ നോക്കുക

```
int sum(int a,int b=o,int(=o)
{relarn (a+b+c)}
```

a പരാമീറ്റർ ലിസ്റ്റിനെ സംബന്ധിച്ച് ഫങ്ഷന്റെ പ്രത്യേകതകൾ എന്താണ്?

b താഴെകൊടുത്തിരിക്കുന്ന ഫങ്ഷനുകളുടെ ഔട്ട്പുട്ട് എഴുതി അതിന്റെ പ്രവർത്തനം വിശദമാക്കുക ഫങ്ഷൻ കാൾ തെറ്റാണെങ്കിൽ അതിന്റെ കാരണം എഴുതുക.

```
(i) cont<<sum (1,2,3):   (ii)cont <<sum (5,2);
(iii) cout<<sun();       (1v) cout <<sum(o);
```

2 `int fun (in, inl);` എന്നത് ഒരു ഫങ്ഷന്റെ പ്രോട്ടോടൈപ്പ് ആണ്. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷൻ വിളികൾ അസാധുവാണ് ഓരോന്നിന്റെയും കാരണം എഴുതുക.

```
a)fun(2,4);         b) cout<<fun();  c) val=fun(2.5,3.3);
d)cin>>fun(a,b);  e) 2=fun(3);
```

11

കമ്പ്യൂട്ടർ ശൃംഖലകൾ

പ്രധാന ആശയങ്ങൾ

കമ്പ്യൂട്ടർ ശൃംഖലകൾ (Computer Networks)
- ശൃംഖലയുടെ ആവശ്യകത
- ചില പ്രധാന പദങ്ങൾ

ഡാറ്റ വിനിമയ സംവിധാനം (Data Communication System)

വിനിമയ മാധ്യമം (Communication Medium)
- ഗൈഡഡ് മീഡിയം (Guided Medium)
- അൺ ഗൈഡഡ് മീഡിയം (Unguided Medium)
- റേഡിയോ തരംഗങ്ങൾ ഉപയോഗിച്ചുള്ള വയർ ലെസ്സ് വിനിമയ സാങ്കേതികവിദ്യകൾ

ഡാറ്റ വിനിമയ ഉപകരണങ്ങൾ (Data Communication Devices)
- എൻ.ഐ.സി. (NIC), ഹബ് (HUB), സ്വിച്ച് (SWITCH), റിപ്പീറ്റർ (Repeater), ബ്രിഡ്ജ് (Birdge), റൂട്ടർ (Router), ഗേറ്റ്‌വേ (Gagteway)

ഡാറ്റ ടെർമിനൽ ഉപകരണങ്ങൾ (Data Terminal Equipments)
- മോഡം (Modem), മൾട്ടിപ്ലെക്സർ (Multiplexer) /ഡിമൾട്ടിപ്ലെക്സർ (Demultiplexer)
- ശൃംഖല ക്രമീകരണ രീതികൾ (Network Topologies)
- ബസ് (Bus), സ്റ്റാർ (Star), റിങ് (Ring), മെഷ് (Mesh)

വിവിധ തരം ശൃംഖലകൾ
- പാൻ (PAN), ലാൻ (LAN), മാൻ (MAN), വാൻ (WAN)

ശൃംഖലയുടെ യുക്ത്യാധിഷ്ഠിത തരംതിരിവുകൾ/വിഭജനം
- പീർ- ടു- പീർ (Peer - to - peer)
- ക്ലയന്റ് സർവർ (Client - Server)

ശൃംഖലാ പെരുമാറ്റ ചട്ടങ്ങൾ/നിയമങ്ങൾ (Network Protocol)
- TCP/IP (HTTP, FTP, DNS)

ഉപയോക്താവിനെയും കമ്പ്യൂട്ടറുകളെയും ശൃംഖലയിൽ തിരിച്ചറിയൽ
- MAC വിലാസം (MAC Address)
- ഐപി വിലാസം (IP Address)
- യൂണിഫോം റിസോഴ്സ് ലൊക്കേറ്റർ (Uniform Resoure Locator)

P7W8T9

പത്താം ക്ലാസ് പരീക്ഷയുടെ ഫലം അറിയുവാനോ പതിനൊന്നാം ക്ലാസ്സിൽ പ്രവേശനം കിട്ടിയോ എന്ന് പരിശോധിക്കുന്നതിനോ നിങ്ങൾ ഇന്റർനെറ്റ് ഉപയോഗിച്ചിട്ടുണ്ടോ? പണം പിൻവലിക്കുന്നതിനായി നിങ്ങൾ എ ടി എം സന്ദർശിച്ചിട്ടുണ്ടോ? കമ്പ്യൂട്ടറിൽ നിന്ന് പാട്ടുകൾ, ചിത്രങ്ങൾ, സിനിമാശകലങ്ങൾ എന്നിവ സെൽ ഫോണിലേക്ക് മാറ്റുവാനോ, ഇന്റർനെറ്റ് ഉപയോഗിച്ച് ട്രെയിൻ ടിക്കറ്റ് ബുക്ക് ചെയ്യുവാനോ നിങ്ങൾ ശ്രമിച്ചിട്ടുണ്ടോ? ഈ ചോദ്യങ്ങളിൽ ഏതെങ്കിലും ഒന്നിന് നിങ്ങളുടെ ഉത്തരം 'അതെ' എന്നാണെങ്കിൽ, നിങ്ങൾ കമ്പ്യൂട്ടർ ശൃംഖലയുടെ സേവനം ഉപയോഗപ്പെടുത്തിയിട്ടുണ്ട് എന്ന് അനുമാനിക്കാം. കമ്പ്യൂട്ടർ ശൃംഖലയുടെ പ്രവർത്തനങ്ങളെക്കുറിച്ചും അവയുടെ ഗുണങ്ങളെക്കുറിച്ചുമാണ് ഈ അദ്ധ്യായത്തിൽ പഠിക്കുന്നത്. ഇതോടൊപ്പം ഈ മേഖലയിൽ ഉപയോഗിക്കുന്ന വിവിധ ഉപകരണങ്ങളെക്കുറിച്ചും മാധ്യമങ്ങളെക്കുറിച്ചും നമുക്ക് ചർച്ച ചെയ്യാം. കൂടാതെ വിവിധതരം കമ്പ്യൂട്ടർ ശൃംഖലകളെക്കുറിച്ചും ശൃംഖലകളിലൂടെ വിനിമയം നടത്തുവാനാവശ്യമായ നിയമങ്ങളെക്കുറിച്ചും ചർച്ച ചെയ്യാം.

11.1 കമ്പ്യൂട്ടർ ശൃംഖല (Computer network)

ഒരു വിനിമയ ഇലക്ട്രോണിക് മാധ്യമത്തിലൂടെ പരസ്പരം ബന്ധിപ്പിച്ചിട്ടുള്ള കമ്പ്യൂട്ടറുകളുടെയും മറ്റു കമ്പ്യൂട്ടിങ് ഹാർഡ്‌വയർ ഉപകരണങ്ങ

ളുടെയും (പ്രിന്ററുകൾ, സ്കാനറുകൾ, മോഡം, CD ഡ്രൈവുകൾ തുടങ്ങിയവ) ഒരു കൂട്ടമാണ് കമ്പ്യൂട്ടർ ശൃംഖല. ഈ ഉപകരണങ്ങൾക്ക് പരസ്പരം വിവരങ്ങൾ വിനിമയം നടത്തുവാനും, നിർദ്ദേശങ്ങൾ കൈമാറുവാനും, ഡാറ്റയും ഉപകരണങ്ങളും പരസ്പരം പങ്കിടുവാനും സാധിക്കുന്നു. ഒരു ശൃംഖലയിൽ ഉള്ള കമ്പ്യൂട്ടറുകളെ കേബിളുകൾ, ടെലിഫോൺ ലൈനുകൾ, റേഡിയോ തരംഗങ്ങൾ, ഇൻഫ്രാറെഡ് തരംഗങ്ങൾ, ഉപഗ്രഹങ്ങൾ ഇവയിലേതെങ്കിലും ഉപയോഗിച്ച് പരസ്പരം ബന്ധിപ്പിക്കാം.

11.1.1 ശൃംഖലയുടെ ആവശ്യകത (Need for network)

ഒരു കമ്പ്യൂട്ടർ ശൃംഖലയ്ക്ക് ഉത്തമ ഉദാഹരണമാണ് ഇന്റർനെറ്റ്. ഇമെയിൽ, ഓൺ ലൈൻ പത്രങ്ങൾ, ബ്ലോഗുകൾ, ചാറ്റിങ് ഇന്റർനെറ്റ് അധിഷ്ഠിത സേവനങ്ങൾ തുടങ്ങിയവ ഇല്ലാത്ത ഒരു ലോകത്തെ കുറിച്ച് നമുക്ക് ചിന്തിക്കുവാൻ കഴിയില്ല. പരസ്പരം ബന്ധിപ്പിച്ചിട്ടില്ലാത്ത കമ്പ്യൂട്ടറുകൾ ഉപയോഗിക്കുന്നതിനേക്കാൾ പലമേന്മകളും പരസ്പരം ബന്ധിപ്പിച്ച കമ്പ്യൂട്ടറുകൾക്ക് ഉണ്ട്. അവയിൽ ചിലത് ചുവടെ ചേർക്കുന്നു.

- വിഭവം പങ്കുവെയ്ക്കൽ (Resource sharing)
- വില പ്രകടന അനുപാതം (Price preformance ratio)
- വിവര വിനിമയം (Communication)
- വിശ്വാസ്യത (Reliability)
- വിപുലീകരിക്കുവാനുള്ള സാധ്യത (Scalability)

വിഭവം പങ്കുവെയ്ക്കൽ: കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ലഭ്യമായ ഹാർഡ്‌വെയറും സോഫ്റ്റ്‌വെയറും പങ്കിടുന്നതിനെയാണ് വിഭവങ്ങളുടെ പങ്കുവെയ്ക്കൽ എന്നതുകൊണ്ട് ഉദ്ദേശിക്കുന്നത്. ഉദാഹരണത്തിന് ഒരു കമ്പ്യൂട്ടറിന്റെ ഡിവിഡി ഡ്രൈവിൽ നിന്നാണ് ഒരു ഡിവിഡി യുടെ ഉള്ളടക്കം മറ്റൊരു കമ്പ്യൂട്ടറിൽ ഉപയോഗിക്കുന്നത്. അതുപോലെ, മറ്റ് ഹാർഡ്‌വെയർ ഉപകരണങ്ങളായ ഹാർഡ് ഡിസ്ക്, പ്രിന്റർ, സ്കാനർ, തുടങ്ങിയവയും സോഫ്റ്റ്‌വെയറുകളായ അപ്ലിക്കേഷൻ സോഫ്റ്റ്‌വെയർ, ആന്റി വൈറസുകൾ തുടങ്ങിയവയും കമ്പ്യൂട്ടർ ശൃംഖല വഴി പരസ്പരം പങ്കിടാം.

വില പ്രകടന അനുപാതം: ഒരു കമ്പ്യൂട്ടറിൽ ലഭ്യമായ വിഭവങ്ങൾ ശൃംഖലയിലുള്ള മറ്റ് കമ്പ്യൂട്ടറുകളുമായി എളുപ്പത്തിൽ പങ്കിടുവാൻ കഴിയുന്നു. ലൈസൻസുള്ള സോഫ്റ്റ്‌വെയർ ഓരോ കമ്പ്യൂട്ടറിനും വാങ്ങുന്നതിനുള്ള ചെലവ് അത്തരം സോഫ്റ്റ്‌വെയറിന്റെ ശൃംഖല പതിപ്പുകൾ വാങ്ങിക്കൊണ്ടു കുറയ്ക്കുവാൻ കഴിയും. വിഭവങ്ങളുടെ ഇത്തരത്തിലുള്ള ഉപയോഗം കമ്പ്യൂട്ടറിന്റെ പ്രകടനത്തെ ബാധിക്കാത്ത വിധത്തിലും, കൂടാതെ കുറഞ്ഞ ചിലവിൽ, ഗണ്യമായ ലാഭത്തിലേക്കു നയിക്കുന്ന തരത്തിലും ആയിരിക്കും.

വിവര വിനിമയം: ഇമെയിൽ, ചാറ്റിങ്, വീഡിയോ കോൺഫെറൻസിങ് തുടങ്ങിയ സേവനങ്ങളിലൂടെ ശൃംഖലയിലുള്ള മറ്റേതെങ്കിലും ഉപഭോക്താവുമായി വിവര വിനിമയം നടത്തുവാൻ കമ്പ്യൂട്ടർ ശൃംഖല സഹായിക്കുന്നു. ഉദാഹരണമായി ലക്ഷ്യസ്ഥാന

ത്തിലേക്കുള്ള ദൂരം കണക്കിലെടുക്കാതെ വളരെ വേഗത്തിൽ സന്ദേശങ്ങൾ അയക്കുവാനും സ്വീകരിക്കുവാനും കഴിയുന്നു.

വിശ്വാസ്യത : കമ്പ്യൂട്ടർ ശൃംഖല ഉപയോഗിച്ച് ഒന്നിലധികം കമ്പ്യൂട്ടറുകളിൽ ആവശ്യമായ വിവരങ്ങളുടെ നിരവധി പകർപ്പുകൾ സൂക്ഷിക്കുവാൻ കഴിയുന്നു. ഉദാഹരണത്തിന്, ഒരു കമ്പ്യൂട്ടറിൽ സംരക്ഷിച്ചിട്ടുള്ള C++ ഫയലുകൾ, ചിത്രങ്ങൾ അല്ലെങ്കിൽ പാട്ടുകൾ എന്നിവ ഇതേ ശൃംഖലയിലെ മറ്റു കമ്പ്യൂട്ടറുകളിൽ സൂക്ഷിക്കാവുന്നതാണ്. ഇങ്ങനെ സൂക്ഷിക്കുന്നത് കൊണ്ട്, ഏതെങ്കിലും കമ്പ്യൂട്ടറിന് തകരാറുണ്ടായാൽ (ശരിയായി പ്രവർത്തിക്കാതിരിക്കുക, യാദൃശ്ചികമായി ഫയലുകൾ നഷ്ടപ്പെട്ട് പോകുക) ഈ ഫയലുകളെ കമ്പ്യൂട്ടർ ശൃംഖലയിൽ നിന്നും വീണ്ടെടുക്കുവാൻ സാധിക്കുന്നു.

വിപുലീകരിക്കുവാനുള്ള സാധ്യത: കമ്പ്യൂട്ടർ ശൃംഖലയിലേയ്ക്ക് കമ്പ്യൂട്ടറുകളുടെ എണ്ണം കൂട്ടിയും കുറച്ചും ശൃംഖലയുടെ പ്രവർത്തന ക്ഷമത ഉയർത്തുകയും താഴ്ത്തുകയും ചെയ്യാം. ഇതിനുപുറമെ ശൃംഖലയിലേയ്ക്ക് കൂടുതൽ സംഭരണ ഉപകരണങ്ങൾ ഉൾപ്പെടുത്തി ശൃംഖലയുടെ സംഭരണ ശേഷി വർദ്ധിപ്പിക്കാം

11.1.2 ചില പ്രധാന പദങ്ങൾ (Some key terms)

കമ്പ്യൂട്ടർ ശൃംഖലയുമായി ബന്ധപ്പെട്ട ചില പ്രധാന പദങ്ങൾ ചുവടെ വിശദമാക്കുന്നു.

ബാൻഡ്‌വിഡ്ത്ത് (Bandwidth) : ബാൻഡ് വിഡ്ത് എന്നാൽ നിശ്ചിത സമയത്ത് നിശ്ചിത മാധ്യമത്തിലൂടെ അയയ്ക്കാവുന്ന ഡാറ്റയുടെ അളവാണ്. നിങ്ങൾ ഒരു ഹൈവേയിലൂടെയോ അല്ലെങ്കിൽ ഒരു പൊതുറോഡിലൂടെയോ സഞ്ചരിക്കുകയാണ് എന്ന് വിചാരിക്കുക. റോഡിന്റെ വീതി കൂടുന്തോറും അതിലൂടെ കടന്നു പോകുവാൻ കഴിയുന്ന വാഹനങ്ങളുടെ എണ്ണം കൂടുന്നതായി കാണാം. മാത്രമല്ല ഇവിടെ ഇടുങ്ങിയ റോഡിനേക്കാൾ വേഗത്തിൽ വാഹനങ്ങൾക്ക് സഞ്ചരിക്കാം. അതുകൊണ്ടു ഒരു വീതിയുള്ള റോഡിന്, ഇടുങ്ങിയ റോഡിനേക്കാൾ ബാൻഡ് വിഡ്ത് കൂടുതലാണ് എന്ന് നമുക്ക് മനസിലാക്കാം.

ഒരുശൃംഖലയിൽ കമ്പ്യൂട്ടറുകൾക്കിടയിൽ പരമാവധി കൈമാറ്റം ചെയ്യുവാൻ കഴിയുന്ന ഡാറ്റയുടെ അളവിനെ ബാൻഡ്‌വിഡ്ത്ത് എന്ന് പറയാം. ബിറ്റ്സ് പെർ സെക്കന്റ് (പ്രതി നിമിഷമാത്രകൾ) (ബിപിഎസ്) എന്ന രീതിയിൽ ഡിജിറ്റൽ സമ്പ്രദായത്തിൽ ഇതിനെ അളക്കുന്നു. ബാൻഡ്‌വിഡ്ത് കൂടുതലാവുമ്പോൾ ഡാറ്റയ്ക്കു വേഗത്തിൽ സഞ്ചരിക്കുവാൻ കഴിയുന്നു, ആയതിനാൽ ഒരു പ്രത്യേക സമയപരിധിക്കുള്ളിൽ ശൃംഖലയിലൂടെ വലിയ അളവിൽ ഡാറ്റ കൈമാറ്റം ചെയ്യാവുന്നതാണ്. ഉദാഹരണത്തിന് കേബിൾ മോഡം വഴിയുള്ള ഇന്റർനെറ്റ് കണക്ഷൻ 25 Mbps ബാൻഡ് വിഡ്ത് നൽകുന്നു.

നോയ്സ് (Noise): ഡാറ്റ സിഗ്നലിന്റെ ഗുണനിലവാരം കുറയ്ക്കുന്നതോ, സിഗ്നലുകളുടെയോ ഡാറ്റയുടെ നീക്കത്തെ തടസപ്പെടുത്തുന്നതോ ആയ മറ്റൊരു അനഭിമതമായ തരംഗമാണ് 'നോയ്സ്' (Noise). സമീപത്തുള്ള സംപ്രേക്ഷണ ഉപകരണങ്ങളിൽ

നിന്നും, മറ്റു യന്ത്രങ്ങളിൽ നിന്നും കേബിളുകളിൽ നിന്നും, പുറത്തു വരുന്ന സിഗ്നലുകളാണ് ഇതിനു കാരണം. ഒരു ശൃംഖലയിൽ (Network) കൈമാറ്റം ചെയ്യപ്പെടുന്ന ടെക്സ്റ്റുകൾ, പ്രോഗ്രാമുകൾ, ചിത്രങ്ങൾ, ഓഡിയോ തുടങ്ങിയ എല്ലാ ഡാറ്റയേയും നോയ്സ് പ്രതികൂലമായി ബാധിക്കുന്നു.

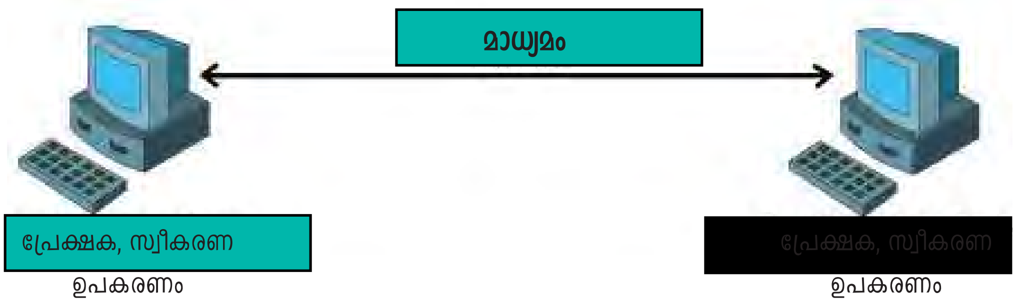
നോഡ് (Node): കമ്പ്യൂട്ടർ ശൃംഖലയിലേക്കു നേരിട്ട് ബന്ധിപ്പിച്ചിട്ടുള്ള ഏത് ഉപകരണത്തെയും (കമ്പ്യൂട്ടർ, സ്കാനർ, പ്രിൻറർ മുതലായവ) നോഡ് എന്ന് പറയുന്നു. ഉദാഹരണമായി, സ്കൂളിൽ കമ്പ്യൂട്ടർ ശൃംഖലയിലേക്ക് ബന്ധിപ്പിച്ചിരിക്കുന്ന കമ്പ്യൂട്ടറുകളെ നോഡ് എന്നാണ് അറിയപ്പെടുന്നത്. നമ്മുടെ കമ്പ്യൂട്ടറിനെ ഇന്റർനെറ്റുമായി ബന്ധിപ്പിക്കുമ്പോൾ, ആ കമ്പ്യൂട്ടർ ഇന്റർനെറ്റിലെ ഒരു നോഡ് ആയി മാറുന്നു.

നമുക്ക് ചെയ്യാം

നിങ്ങളുടെ സ്കൂളിലെ കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ഉപയോഗിച്ചിട്ടുള്ള ഹാർഡ്‌വെയറിന്റെയും സോഫ്റ്റ്‌വെയറിന്റെയും പട്ടിക തയ്യാറാക്കുക.

11.2 ഡാറ്റയുടെ വിനിമയ സമ്പ്രദായം (Data communication system)

വിവരവിനിമയത്തിനും പങ്കുവെയ്ക്കലിനും വേണ്ടി ഒരു കമ്പ്യൂട്ടർ ശൃംഖലയിലെ ഉപകരണങ്ങളെ വിവിധ രീതിയിൽ ബന്ധിപ്പിച്ചിരിക്കുന്നു. ഒരു സംപ്രേക്ഷണ മാധ്യമത്തിലൂടെ രണ്ടു ഉപകരണങ്ങൾ തമ്മിൽ നടത്തുന്ന ഡിജിറ്റൽ വിവരങ്ങളുടെ കൈമാറ്റത്തെ ഡാറ്റയുടെ വിനിമയം അഥവാ ഡാറ്റ കമ്മ്യൂണിക്കേഷൻ (Data Communication) എന്ന് പറയുന്നു. ചിത്രം 8.1 ൽ ഡാറ്റയുടെ വിനിമയ സംവിധാനത്തിന്റെ പൊതു പ്രാതിനിധ്യം കാണിക്കുന്നു.



ചിത്രം 8.1: ഡാറ്റ വിനിമയ സംവിധാനം

ഒരു ഡാറ്റയുടെ വിനിമയ സംവിധാനം നിർമ്മിക്കുന്നതിന് താഴെപ്പറയുന്ന അഞ്ച് അടിസ്ഥാന ഘടകങ്ങൾ ആവശ്യമാണ്.

സന്ദേശം (Message) : വിനിമയം ചെയ്യേണ്ട പ്രധാന വിവരങ്ങൾ ആണ് ഇത്. ഇതിൽ ടെക്സ്റ്റുകൾ, ചിത്രങ്ങൾ, ഓഡിയോ, വീഡിയോ തുടങ്ങിയവ ഉൾപ്പെടുന്നു.

പ്രേക്ഷകൻ (Sender): സന്ദേശം അയയ്ക്കുവാൻ ഉപയോഗിക്കുന്ന കമ്പ്യൂട്ടറിനെയും, ഉപകരണങ്ങളെയും, പ്രേക്ഷകനെന്നോ, ഉറവിടം എന്നോ, സംപ്രേക്ഷണ സാമഗ്രി എന്നോ വിളിക്കാം.

സ്വീകർത്താവ് (Receiver): സ്വീകർത്താവ് എന്നത് സന്ദേശങ്ങൾ സ്വീകരിക്കുന്ന കമ്പ്യൂട്ടറോ അനുബന്ധ ഉപകരണങ്ങളോ ആകാം.

മാധ്യമം (Medium):- പ്രേക്ഷകനിൽ നിന്ന് സ്വീകർത്താവിലേയ്ക്ക് സന്ദേശം സഞ്ചരിക്കുന്ന ഭൗതിക പാതയാണ് ഇത്. നോഡുകൾ തമ്മിൽ പരസ്പരം ബന്ധിപ്പിച്ചിരിക്കുന്ന രീതിയെ ഇത് സൂചിപ്പിക്കുന്നു.

പ്രൊട്ടോക്കോൾ (Protocol):- പ്രേക്ഷകനും സ്വീകർത്താവും സന്ദേശങ്ങൾ കൈമാറ്റം ചെയ്യുമ്പോൾ പാലിക്കേണ്ട നിയമങ്ങളെ പ്രോട്ടോകോൾ (protocol) എന്ന് വിളിക്കാം.

11.3 വിവര വിനിമയ മാധ്യമം (Communication medium)

ഒരു ഉപകരണത്തിൽ നിന്ന് മറ്റൊന്നിലേക്കു സന്ദേശം വഹിക്കുവാൻ കഴിയുന്ന ഒരു മാധ്യമം ഉണ്ടെങ്കിൽ മാത്രമേ ഡാറ്റയുടെ വിനിമയ പ്രക്രിയ പൂർണ്ണമാകുകയുള്ളൂ. ഒരു കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ഡാറ്റ കൈമാറ്റം ചെയ്യുവാൻ ഉപയോഗിക്കുന്ന മാധ്യമത്തെ വിവരവിനിമയ പാത അല്ലെങ്കിൽ വിനിമയ മാധ്യമം എന്ന് വിളിക്കാം. ഒരു കമ്പ്യൂട്ടർ ശൃംഖലയിൽ വിവരവിനിമയത്തിനായി രണ്ടു തരത്തിലുള്ള മാധ്യമങ്ങളെ ഉപയോഗിക്കാം. ഗൈഡഡ് മാധ്യമവും അൺഗൈഡഡ് മാധ്യമവും ഗൈഡഡ് മാധ്യമത്തിൽ കേബിളുകൾ ഉപയോഗിക്കുന്നു. അതെ സമയം അൺഗൈഡഡ് മാധ്യമത്തിൽ റേഡിയോ തരംഗങ്ങൾ, മൈക്രോവേവ് തരംഗങ്ങൾ അല്ലങ്കിൽ ഇൻഫ്രാറെഡ് തരംഗങ്ങൾ എന്നിവയാണ് ഡാറ്റ അയയ്ക്കുവാനായി ഉപയോഗിക്കുന്നത്.

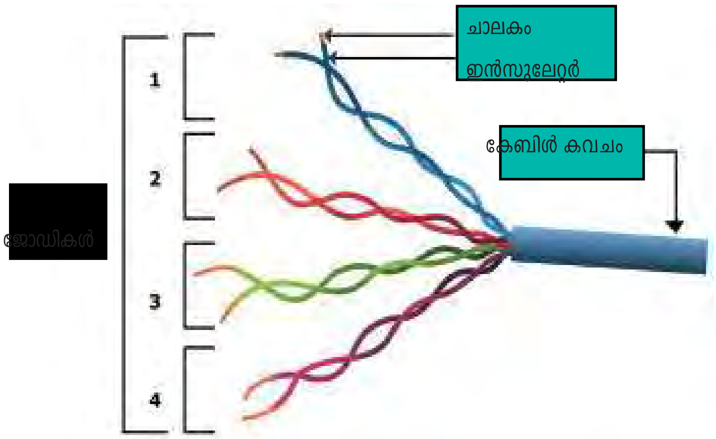
11.3.1 ഗൈഡഡ് മാധ്യമം (Guided Medium (wired))

കോയാക്സിൽ കേബിൾ (Coaxial cable), ട്വിസ്റ്റഡ് പെയർ കേബിൾ (Twisted pair cable), ഒപ്റ്റിക്കൽ ഫൈബർ കേബിൾ (Optical fibre cable) എന്നിവ കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ഡാറ്റ കൈമാറുവാനായി ഉപയോഗിക്കുന്ന ഗൈഡഡ് മാധ്യമങ്ങളാണ്.

a. ട്വിസ്റ്റഡ് പെയർ കേബിൾ (ഈതർനെറ്റ് കേബിൾ) (Twisted pair cable (Ethernet calbe)

ചെറിയ കമ്പ്യൂട്ടർ ശൃംഖലയ്ക്ക് അനുയോജ്യവും, ഏറ്റവും വ്യാപകമായി ഉപയോഗിയ്ക്കുന്നതുമാണ് ഈ മാധ്യമം. വ്യത്യസ്ത നിറങ്ങൾ കൊണ്ട് തിരിച്ചറിയുവാൻ കഴിയുന്ന നാല് ജോഡി വയറുകളെ ഒരു കവചം കൊണ്ട് സംരക്ഷിച്ചു കൊണ്ടുള്ള രൂപകല്പനയാണിത്. ട്വിസ്റ്റഡ്പെയർ രണ്ടു തരത്തിലാണ് ഉള്ളത് 1) അൺഷീൽഡ്ഡ് ട്വിസ്റ്റഡ് പെയർ ((Unshielded Twisted Pair (UTP)), 2) ഷീൽഡ്ഡ് ട്വിസ്റ്റഡ് പെയർ (Shielded Twisted Pair) (STP) എന്നും

അൺഷീൽഡഡ് ട്വിസ്റ്റഡ് പെയർ (Unshielded Twisted Pair (UTP)): പേരു പോലെ തന്നെ കവചം ഇല്ലാത്ത തരം കേബിൾ ആണിത്.



ചിത്രം 8.2: UTP കേബിളിന്റെ പ്രധാന ഭാഗങ്ങൾ കാണിച്ചിരിക്കുന്നു.

ഇതിന്റെ പ്രധാന സവിശേഷതകൾ

- വളരെ കുറഞ്ഞ ചെലവിൽ ചെറിയ ശൃംഖലകൾ നിർമ്മിക്കാം.
- കനം കുറഞ്ഞതും വഴക്കമുള്ളതും ആയ കേബിളാണ്.
- വളരെ എളുപ്പത്തിൽ ശൃംഖലാ ഉപകരണങ്ങളെ ബന്ധിപ്പിക്കാം.
- 100 m ദൂരത്തിൽ വരെ ഡാറ്റയെ വഹിച്ചു കൊണ്ട് പോകുവാനുള്ള കഴിവ് ഉണ്ട്

ഷീൽഡഡ് ട്വിസ്റ്റഡ് പെയർ (Shielded Twisted Pair (STP)): UTP കേബിളിനെപ്പോലെ തന്നെയാണ് എങ്കിലും STP യിൽ ജോഡികളായ വയറുകളെ പൊതിഞ്ഞു സൂക്ഷിക്കുന്നു. UTP കേബിളിനെ പോലെ പിന്നീട് എല്ലാറ്റിനെയും പൊതിഞ്ഞു കൊണ്ട് ഒരു കവചവും ഉണ്ടാകും.

ഇതിന്റെ പ്രധാന സവിശേഷതകൾ

- നോയിസ് (Noise) ന് എതിരെ ശക്തമായ പ്രതിരോധ സംവിധാനമാണ് ഈ കേബിളിന് ഉള്ളത്.
- ഇതിന് UTP കേബിളിനേക്കാൾ വില കൂടുതൽ ആണ്.
- UTP കേബിളുമായി താരതമ്യം ചെയ്യുമ്പോൾ STP കേബിൾ സ്ഥാപിക്കുവാൻ പ്രയാസമാണ്.

ചിത്രം 8.3: എസ്ടിപി കേബിളും RJ-45 കണക്ടറും

RJ45 എന്ന കണക്ടർ ഉപയോഗിച്ചാണ് UTP/STP കേബിളുകൾ കമ്പ്യൂട്ടറുമായി ബന്ധിപ്പിച്ചിരിക്കുന്നത്.

b. കൊയാക്സിയൽ കേബിൾ (Coaxial cable)

ഒരു കൊയാക്സിയൽ കേബിളിന്റെ ഉൾഭാഗത്ത് ഒരു ചാലകത്തെ പൊതിഞ്ഞു കൊണ്ട് ഒരു ഇൻസുലേറ്റർ ട്യൂബും വീണ്ടും അതിനെ പൊതിഞ്ഞു കൊണ്ട് ഒരു ചാലകവും (ഷീൽഡ്) ഉണ്ടായിരിക്കും. ഇതിനു പുറമെ ഒരു പ്രതിരോധ കവചവും കൂടി കാണും. ചിത്രം 8.4 കൊയാക്സിയൽ കേബിളിന്റെ ഘടന ചിത്രീകരിച്ചിരിക്കുന്നു.

കൊയാക്സിയൽ കേബിളിന്റെ സവിശേഷതകൾ.

- ദീർഘ ദൂരത്തേക്ക് (ഏകദേശം 185 m മുതൽ 500 m വരെ) ഒറ്റയടിക്ക് ഡാറ്റയെ വഹിച്ചു കൊണ്ട് പോകുവാൻ കഴിയും.
- വളരെ ഉയർന്ന ബാൻഡ്‌വിഡ്ത് ആണ് ഉള്ളത്.
- പുറംചട്ട (കവചം) ഉള്ളതുകൊണ്ട് വളരെ കുറഞ്ഞ തോതിലുള്ള വൈദ്യുതകാന്തിക തരംഗങ്ങളുടെ തടസ്സപ്പെടുത്തൽ മാത്രമേ ഉണ്ടാകുന്നുള്ളൂ.

ചിത്രം 8.4: കൊയാക്സിയൽ കേബിൾ

- ട്വിസ്റ്റഡ് പെയർ കേബിളിനേക്കാൾ കനം കൂടിയ രൂപകല്പനയാണ്.
- ട്വിസ്റ്റഡ് പെയർ കേബിളിനേക്കാൾ വഴക്കം വളരെ കുറവാണ്.
- ട്വിസ്റ്റഡ്പെയറുമായി താരതമ്യം ചെയ്യുമ്പോൾ സ്ഥാപിക്കുവാൻ പ്രയാസമാണ്.

c. ഒപ്റ്റിക്കൽ ഫൈബർ കേബിൾ (Optical fibre cable)

ഡാറ്റയെ പ്രകാശ കണികാ രൂപത്തിൽ ഒരു നീളം കൂടിയ കനം കുറഞ്ഞ ഗ്ലാസ് ട്യൂബിലൂടെ കടത്തിവിടുന്ന രൂപകല്പനയാണ് ഒപ്റ്റിക്കൽ ഫൈബറുകൾക്കുള്ളത്. പ്രകാശത്തിന്റെ വേഗതയിൽ ഡാറ്റയെ വളരെ ദൂരത്തേക്ക് സംപ്രേക്ഷണം ചെയ്യുവാൻ കഴിയുന്നു. ചിത്രം 8.5 ഒപ്റ്റിക്കൽ ഫൈബറിന്റെ പ്രധാന ഭാഗങ്ങൾ കാണിച്ചിരിക്കുന്നു.

ചിത്രം 8.5: ഒപ്റ്റിക്കൽ ഫൈബർ

ഒപ്റ്റിക്കൽ ഫൈബറിന് താഴെപ്പറയുന്ന ഭാഗങ്ങൾ ഉണ്ട്.

- കോർ: മധ്യഭാഗത്തു കൂടി പ്രകാശം കടന്നു പോകുന്ന കനം കുറഞ്ഞ ഗ്ലാസിന്റെ കുഴലാണ് ഇത്.
- ക്ലാഡിങ് : കോർ ഭാഗത്തെ പൊതിഞ്ഞു കൊണ്ട് പ്രകാശത്തെ കോറിനുള്ളിലേക്കു തന്നെ പ്രതിഫലിപ്പിക്കുന്ന പുറം ഭാഗമാണ് ഇത്.
- കോട്ടിങ് : ഈർപ്പത്തിൽ നിന്നും, തകരാറിൽ നിന്നും സംരക്ഷിക്കുന്നതിനായിട്ടുള്ള കേബിളിന്റെ പ്ലാസ്റ്റിക് കവചമാണ് ഇത്.

നൂറുകണക്കിനോ ആയിരക്കണക്കിനോ ആയ ഒപ്റ്റിക്കൽ ഫൈബർ കേബിളുകളെ പൊതിഞ്ഞിരിക്കുന്ന കവചത്തെ ജാക്കറ്റ് എന്ന് വിളിക്കുന്നു.

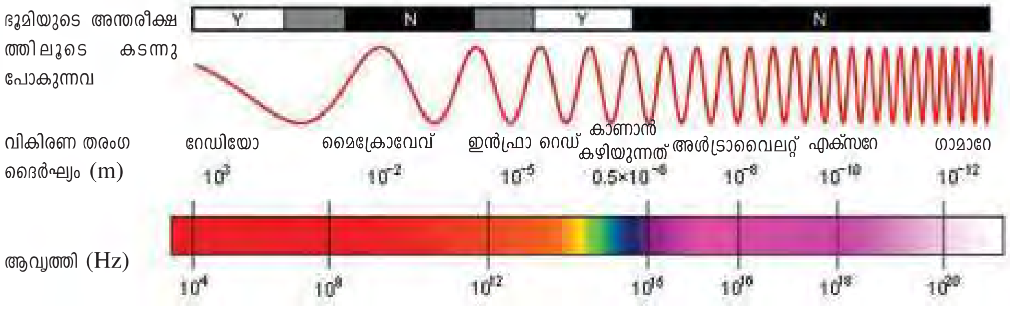
അർദ്ധചാലക ഉപകരണങ്ങളായ ലൈറ്റ് എമിറ്റിങ് ഡയോഡുകൾ (LED) ലേസർ ഡയോഡുകൾ എന്നിവ ഉപയോഗിച്ച് ഉത്ഭവ സ്ഥാനത്തുവെച്ചു ഒപ്റ്റിക്കൽ ട്രാൻസ്മിറ്റർ, വൈദ്യുത തരംഗങ്ങളെ പ്രകാശ തരംഗങ്ങൾ ആക്കി മാറ്റുന്നു (മോഡുലേഷൻ). മറുഭാഗത്ത്, ഫോട്ടോ ഡിറ്റക്ടർ അടങ്ങിയ ഒപ്റ്റിക്കൽ റിസീവർ, പ്രകാശ തരംഗങ്ങളെ ഫോട്ടോ ഇലക്ട്രിക്ക് പ്രഭാവം ഉപയോഗിച്ച് തിരികെ വൈദ്യുത തരംഗങ്ങൾ ആക്കി മാറ്റുന്നു (ഡീമോഡുലേഷൻ). ലേസർ ഡയോഡുകൾക്കു ദൂരപരിധിയും, കൈമാറ്റ വേഗതയും LED ഡയോഡുകളേക്കാൾ കൂടുതൽ ആണ് .

ഓപ്റ്റിക്കൽ ഫൈബർ കേബിളിന്റെ സവിശേഷതകൾ

- ഉയർന്ന ബാൻഡ് വിഡ്ത്തിൽ ശബ്ദവും, വീഡിയോയും ഡാറ്റയും കൈമാറുന്നു
- ഒറ്റയടിക്ക് ഡാറ്റയെ ദീർഘ ദൂരത്തേയ്ക്ക് എത്തിക്കുന്നു.
- ഡാറ്റ കൈമാറ്റം ചെയ്യുവാൻ പ്രകാശ കണികകൾ ഉപയോഗിക്കുന്നതിനാൽ വൈദ്യുത കാന്തിക തരംഗങ്ങളുമായി ഒരു കൂടിച്ചേരലും നടക്കുന്നില്ല.
- കമ്പ്യൂട്ടർ ശൃംഖലയ്ക്കു ലഭ്യമായതിൽ വെച്ച് ഏറ്റവും ചെലവേറിയതും കാര്യക്ഷമത കൂടിയതുമായ മാദ്ധ്യമമാണിത്.
- പരിപാലനവും സ്ഥാപിക്കലും (Maintenance and installation) പ്രയാസകരവും സങ്കീർണവുമാണ്.

11.3.2 അൺ ഗൈഡഡ് മീഡിയം (വയർരഹിതം) (Unguided medium (Wireless))

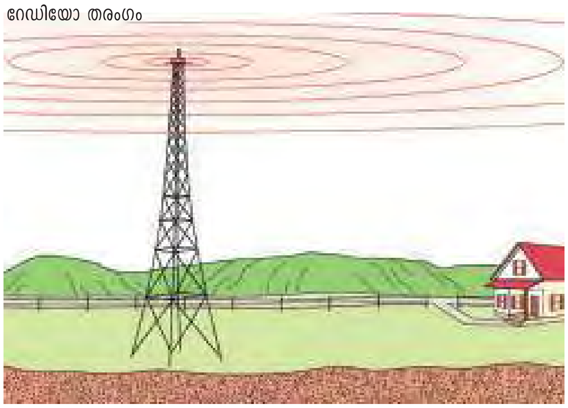
വൈദ്യുതകാന്തിക തരംഗങ്ങൾ ആണ് വയർലെസ്സ് വിവരവിനിമയത്തിനായി കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ഉപയോഗിക്കുന്നത്. തരംഗദൈർഘ്യത്തെ ഹേർട്സ് (Hetrz (Hz))ൽ ആണ് കണക്കാക്കുന്നത്. ചിത്രം 8.6 ൽ ആവൃത്തിയെ അടിസ്ഥാനമാക്കി വിവിധ തരം വൈദ്യുതകാന്തികതരംഗങ്ങൾ കാണിച്ചിരിക്കുന്നു. ഈ വിഭാഗത്തിൽ വയർലസ് വിവര വിനിമയത്തിനായി, റേഡിയോ തരംഗങ്ങളും മൈക്രോ തരംഗങ്ങളും ഇൻഫ്രാറെഡ് തരംഗങ്ങളുമാണ് ഉപയോഗിക്കുന്നത് എന്ന് നമുക്ക് മനസ്സിലാക്കാം.



ചിത്രം 8.6 : വൈദ്യുത കാന്തിക വർണ്ണരാജി (സ്പെക്ട്രം)

a. റേഡിയോ തരംഗങ്ങൾ (Radio waves)

റേഡിയോ തരംഗങ്ങളുടെ ആവൃത്തി 3 KHz മുതൽ 3 GHz വരെയാണ്. റേഡിയോ തരംഗങ്ങൾ ഹ്രസ്വ / ദീർഘ ദൂര സംപ്രേക്ഷണത്തിനു ഉപയോഗിക്കുന്നു. ഇത്തരം തരംഗങ്ങളെ വളരെ എളുപ്പത്തിൽ ഉൽപ്പാദിപ്പിക്കാം എന്നതിന് പുറമെ അവയ്ക്കു തടസ്സങ്ങൾ മറികടക്കുവാനുള്ള കഴിവും ഉണ്ട്. ഇക്കാരണത്താൽ വിവരവിനിമയ ത്തിനായി എല്ലാ മേഖലയിലും (കെട്ടിടങ്ങൾക്ക് ഉള്ളിലും പുറത്തും) റേഡിയോ തരം ഗങ്ങൾ ഉപയോഗിക്കുന്നു. കോഡ്ലസ് ഫോൺ, AM, FM റേഡിയോ സംപ്രേക്ഷണം, മൊബൈൽ ഫോൺ തുടങ്ങിയവയിൽ റേഡിയോ തരംഗങ്ങൾ ആണ് ഉപയോഗി ക്കുന്നത്.

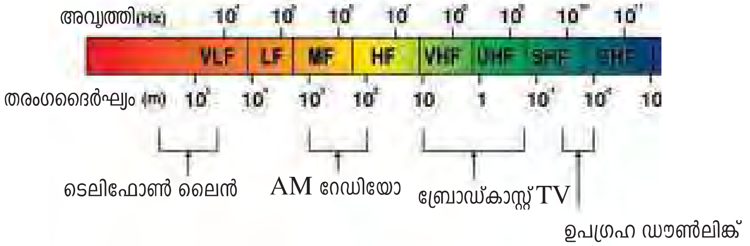


ചിത്രം 8.7 : റേഡിയോ തരംഗ പ്രസരണം

റേഡിയോ തരംഗങ്ങളുടെ വിവരവിനിമയ സവിശേഷതകൾ

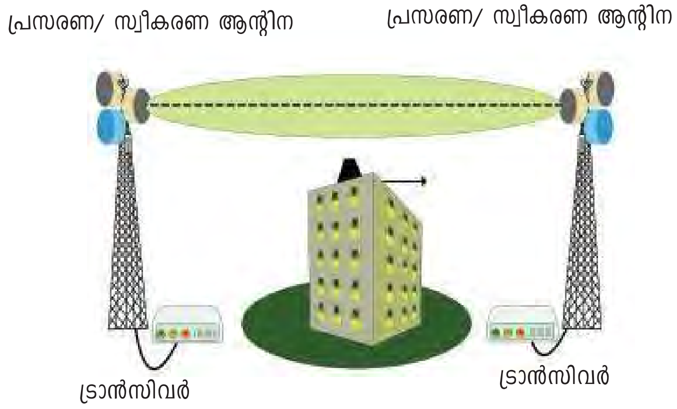
- എല്ലാ ദിശയിലേക്കും റേഡിയോ തരംഗങ്ങൾക്ക് സഞ്ചരിക്കാൻ കഴിവുള്ളതിനാൽ, സ്വീകരിക്കുവാനും പ്രസാരണം ചെയ്യുവാനും ഉപയോഗിക്കുന്ന ഉപകരണങ്ങൾ നേർക്കുനേർ വരണമെന്നില്ല.
- വയർ അധിഷ്ഠിത മാധ്യമവുമായി താരതമ്യം ചെയ്യുമ്പോൾ ഇതിന് ചെലവ് കുറവാണ്.

- മിക്ക വസ്തുക്കൾക്കുള്ളിലൂടെയും കടന്നു പോകുവാനുള്ള കഴിവുണ്ട്.
- പ്രസാരണത്തെ മോട്ടോറുകളും ഇലക്ട്രിക്ക് ഉപകരണങ്ങളും സ്വാധീനിക്കാൻ സാധ്യതയുണ്ട്.
- സുരക്ഷിതത്വം കുറഞ്ഞ വിനിമയ രീതിയാണ്.
- റേഡിയോ തരംഗങ്ങളുടെ ഉപയോഗത്തിന് ബന്ധപ്പെട്ട അധികാരികളുടെ അനുവാദം ആവശ്യമാണ്.



ചിത്രം 8.8: റേഡിയോ വിവരവിനിമയ സ്പെക്ട്രം

b. മൈക്രോ തരംഗങ്ങൾ (സൂക്ഷ്മതരംഗം) (Micro waves)



ചിത്രം 8.9: മൈക്രോവേവ് പ്രസരണം

മൈക്രോ തരംഗങ്ങളുടെ ആവൃത്തി 300 MHz (0.3GHz) മുതൽ 300 GHz വരെയാണ്. മൈക്രോ തരംഗങ്ങൾ നേർരേഖയിൽ സഞ്ചരിക്കുന്നതും ഖരപദാർഥങ്ങൾക്കുള്ളിലൂടെ കടന്നു പോകാത്തതും ആണ്. ആയതിനാൽ വളരെ ഉയരം കൂടിയ ടവറുകൾ ഉണ്ടാക്കി അതിനു മുകളിൽ മൈക്രോവേവ് ആന്റിനകൾ ഉറപ്പിച്ചാണു ദീർഘ ദൂര പ്രസരണം സാധ്യമാക്കുന്നത്. തരംഗങ്ങൾ നേർരേഖയിൽ സഞ്ചരിക്കുന്നതിനാൽ പ്രസരണം ചെയ്യുന്നതിനും സ്വീകരിക്കുന്നതിനും ഉള്ള ആന്റിനകൾ പരസ്പരം അഭിമുഖമായാണ് സ്ഥാപിച്ചിരിക്കുന്നത്. രണ്ടു മൈക്രോവേവ് ടവറുകൾ തമ്മിലുള്ള അകലം നിശ്ചയിക്കുന്നത് തരംഗങ്ങളുടെ ആവൃത്തിയും ടവറുകളുടെ ഉയരവും അനുസരിച്ച് ആണ്. ചിത്രം 8.9 യിൽ ഒരു മൈക്രോവേവ് പ്രസരണ സംവിധാനത്തിന്റെ ഭാഗങ്ങൾ ചിത്രീകരിച്ചിരിക്കുന്നു.

മൈക്രോവേവ് സംപ്രേക്ഷണത്തിന്റെ സവിശേഷതകൾ

- വയേർഡ് മാധ്യവുമായി താരതമ്യം ചെയ്യുമ്പോൾ ഇതിന് ചെലവ് കുറവാണ്
- ദുഷ്കരമായ ഭൂപ്രദേശങ്ങളിൽ സുഗമമായ വിവര വിനിമയം സാധ്യമാകുന്നു
- പ്രസാരണം നേർരേഖയിൽ ആയതിനാൽ പ്രസരണ ഉപകരണവും സ്വീകരണ ഉപകരണവും അഭിമുഖമായിത്തന്നെ സ്ഥാപിക്കണം.

c ഇൻഫ്രാറെഡ് തരംഗങ്ങൾ (Infrared waves)

ഇൻഫ്രാറെഡ് തരംഗങ്ങൾ 300 GHz മുതൽ 400 THz വരെ ആവൃത്തിയുള്ളവയാണ് ഹ്രസ്വ ദൂര സംപ്രേക്ഷണത്തിനാണ് ഇത് ഉപയോഗിക്കുന്നത് (ഏകദേശം 5m). ആപ്ലിക്കേഷനുകളെ നിയന്ത്രിക്കുവാനും വിലയിരുത്തുവാനും കൂടാതെ വിവിധ തരത്തിലുള്ള വയർലെസ്സ് വിവരവിനിമയത്തിനും ഇത് ഉപയോഗിക്കുന്നു.

ചിത്രം 8.10 : ഇൻഫ്രാറെഡ് പ്രസരണം

വിവിധ ഗാർഹിക വിനോദ ഉപകരണങ്ങളിലെ റിമോട്ടുകൾ, കോർഡ്ലെസ് മൗസ്, അനധികൃതമായി കടന്നുകയറുന്നത് ശ്രദ്ധയിൽപ്പെടുത്തുന്ന ഉപകരണങ്ങൾ തുടങ്ങിയവയിൽ ഇൻഫ്രാറെഡ് തരംഗങ്ങൾ ഉപയോഗിക്കുന്നു. (ചിത്രം 8.10 പരിശോധിക്കുക)

ഇൻഫ്രാറെഡ് തരംഗങ്ങളുടെ സവിശേഷതകൾ

- നേർരേഖയിലുള്ള വിവര വിനിമയം നടക്കുന്നതിനാൽ, വിവരങ്ങൾ ചോർത്തപ്പെടുന്നില്ല.
- രണ്ടു ഉപകരണങ്ങൾക്ക് മാത്രമേ ഒരു സമയത്തു വിവര വിനിമയം നടത്തുവാൻ സാധിക്കൂ.
- ഖര പദാർത്ഥങ്ങളെ മറികടക്കുവാനുള്ള കഴിവില്ല (റിമോട്ട് കൺട്രോളിനും ടീവിയ്ക്കും ഇടയിൽനിന്നു കൊണ്ട് റിമോട്ട് കൺട്രോൾ പ്രവർത്തിക്കുന്നുണ്ടോ എന്ന് പരിശോധിക്കാവുന്നതാണ്).
- എത്തിപ്പെടാവുന്ന ദൂരം കൂടുന്തോറും തരംഗശേഷി കുറയുന്നു.

11.3.3 റേഡിയോ തരംഗങ്ങൾ ഉപയോഗിച്ചുള്ള വയർരഹിത വിനിമയ സംവിധാനം (Wireless communication technologies using radio waves)

a. ബ്ലൂടൂത്ത് (Bluetooth)

റേഡിയോ തരംഗങ്ങൾ ആണ് ബ്ലൂ ടൂത്ത് സംവിധാനത്തിൽ ഉപയോഗിക്കുന്നത്. ഇതിന്റെ ആവൃത്തി 2.402 GHz മുതൽ 2.480 GHz വരെയാണ്. ഹ്രസ്വ ദൂര വിവര വിനിമയത്തിന് ഉപയോഗിക്കുന്ന വയർലസ് ഉപകരണങ്ങളിൽ (ഏകദേശം 10m) ഇത് ഉപയോഗിക്കുന്നു. സെൽഫോൺ, ലാപ്ടോപ്, മൗസ്, കീബോർഡ്, ടാബ്ലെറ്റുകൾ, ഹെഡ്സെറ്റ്, ക്യാമറ, എന്നിവ ബ്ലൂടൂത്ത് ഉപയോഗിക്കുന്ന ചില ഉപകരണങ്ങൾ ആണ്. (ചിത്രം 8.11 പരിശോധിക്കുക.)

ചിത്രം 8.11 : ബ്ലൂ - ടൂത്ത് പ്രസരണം

ബ്ലൂടൂത്തിന്റെ വിനിമയ സവിശേഷതകൾ

- വിവരവിനിമയം നടത്തുവാൻ നേർരേഖയിൽ പ്രസരണ ഉപകരണങ്ങൾ സ്ഥാപിക്കേണ്ട ആവശ്യമില്ല.
- ബ്ലൂടൂത്ത് ഉപയോഗിച്ച് ഒരേ സമയം എട്ടു ഉപകരണങ്ങളേവരെ ബന്ധിപ്പിക്കാം.
- വേഗതകുറഞ്ഞ വിനിമയ മാർഗമാണ് ഇത് (1 Mbps വരെ).

b. വൈ-ഫൈ (Wi-Fi)

റേഡിയോ തരംഗങ്ങൾ ഉപയോഗിച്ച് ആണ് വൈ-ഫൈ ശൃംഖല പ്രവർത്തിക്കുന്നത്. സെൽഫോൺ, ടെലിവിഷൻ, റേഡിയോ തുടങ്ങിയ ഉപകരണങ്ങളിൽ വിവരങ്ങൾ കൈമാറ്റം ചെയ്യുവാൻ വൈ-ഫൈ ഉപയോഗിക്കുന്നു. വൈ-ഫൈ ശൃംഖലയിൽ ഉപയോഗിക്കുന്ന റേഡിയോ തരംഗങ്ങളുടെ ആവൃത്തി 2.4GHz മുതൽ 5 GHz വരെയാണ്. വയർലെസ്സ് ശൃംഖലയിൽ ഇരുദിശകളിലേക്കും ഉള്ള വിവരവിനിമയമാണ് നടക്കുന്നത്.

കമ്പ്യൂട്ടറിൽ ഉള്ള വയർലെസ്സ് അഡാപ്റ്റർ ഡാറ്റയെ റേഡിയോ തരംഗങ്ങൾ ആക്കി മാറ്റുകയും അവയെ ഒരു ആന്റിന ഉപയോഗിച്ച് സംപ്രേക്ഷണം ചെയ്യുകയും ചെയ്യുന്നു. വയർലെസ് റൂട്ടർ ഇവയെ സ്വീകരിച്ച് പരിവർത്തനം ചെയ്യുന്നു. പരിവർത്തനം ചെയ്യപ്പെട്ട ഡാറ്റയെ ഇന്റർനെറ്റിലേക്കോ, ശൃംഖലയിലേക്കോ ഒരു വയേർഡ് ഈതർനെറ്റ് (ethernet) /വയർലെസ്സ് കണക്ഷൻ വഴി അയയ്ക്കപ്പെടുന്നു. ഇതുപോലെ ഇന്റർനെറ്റ് വഴി ലഭിക്കുന്ന ഡാറ്റ റൂട്ടർ വഴികടന്നു പോകുകയും, അവയെ റേഡിയോ തരംഗങ്ങൾ ആക്കി ഒരു കമ്പൂട്ടറിൽ ഉള്ള വയർലസ്സ് ആഡാപ്റ്റർ സ്വീകരിക്കുന്നത് ചിത്രം 11.12 ൽ സൂചിപ്പിച്ചിരിക്കുന്നു. ഇപ്പോൾ ഈ സാങ്കേതികവിദ്യ ലാപ്ടോപ്പിലും ഡെസ്ക്ടോപ്പിലും ഇന്റർനെറ്റ് കണക്ഷൻ പങ്കിടുവാൻ വ്യാപകമായി ഉപയോഗിക്കുന്നു.

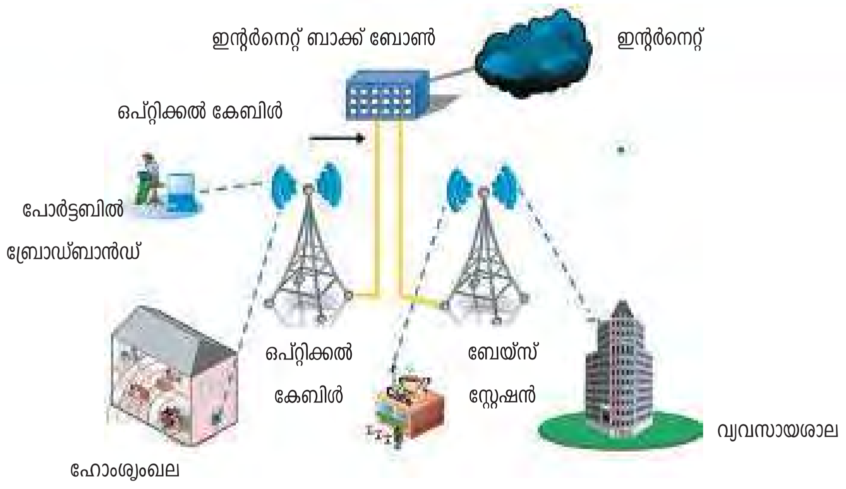
ചിത്രം 8.12: വൈ - ഫൈ പ്രസരണം

വൈ-ഫൈ പ്രസരണത്തിന്റെ സവിശേഷതകൾ

- ഉപകരണങ്ങൾ തമ്മിലുള്ള നേർകാഴ്ച ഇവിടെ ആവശ്യമില്ല.
- സംപ്രേക്ഷണത്തിന്റെ വേഗത 54Mbps വരെയാണ്.
- ഒരേ സമയം കൂടുതൽ ഉപകരണങ്ങളെ വൈ ഫൈ ഉപയോഗിച്ച് ബന്ധിപ്പിക്കാം.
- 114m (375 അടി) വരെയുള്ള വിനിമയത്തിന് ഉപയോഗിക്കുന്നു.

c. വൈ-മാക്സ് (Wi-MAX)

വേൾഡ് വൈഡ് ഇന്ററോപ്പറബിളിറ്റി ഫോർ മൈക്രോവേവ് അക്സസ് (വൈ-മാക്സ്)ന്റെ അടിസ്ഥാനം 802.16e ആണ്. ബ്രോഡ്ബാൻഡിന്റെയും വയർലെസ്സിന്റെയും സവി ശേഷതകൾ സംയോജിപ്പിച്ചാണ് വൈ-മാക്സിനു രൂപം കൊടുത്തിരിക്കുന്നത്. വൈ-മാക്സിന്റെ ആവൃത്തി 2GHz മുതൽ 11 GHz വരെയാണ്. വൈ-മാക്സ് അതിവേഗ ത്തിലും ദീർഘ ദൂരത്തിലും ഇന്റർനെറ്റ് ഉപയോഗം സാധ്യമാക്കുന്നു (നഗരത്തിലുട നീളം). അടിസ്ഥാനതലത്തിൽ വൈ മാക്സിനു രണ്ട് തരത്തിലുള്ള സജ്ജീകരണങ്ങൾ ആണ് ഉള്ളത്. സേവനദാതാവ് സാങ്കേതികവിദ്യ വിന്യസിക്കുവാൻ ആയി ഒരു പ്രത്യേക മേഖലയിൽ ഉപയോഗിച്ചിരിക്കുന്ന ഉപകരണങ്ങളും, ഉപഭോക്താവ് സ്ഥാപിച്ചിരിക്കുന്ന സ്വീകരണ ഉപകരണങ്ങളും ചേർന്നതാണ് ബെയ്സ് സ്റ്റേഷൻ. വൈ-മാക്സ് പ്രസര ണത്തിന് ഉപയോഗിക്കുന്ന അടിസ്ഥാന ഉപകരണങ്ങൾ ചുവടെ ചിത്രം 8.13 ചിത്രീകരി ച്ചിരിക്കുന്നു.



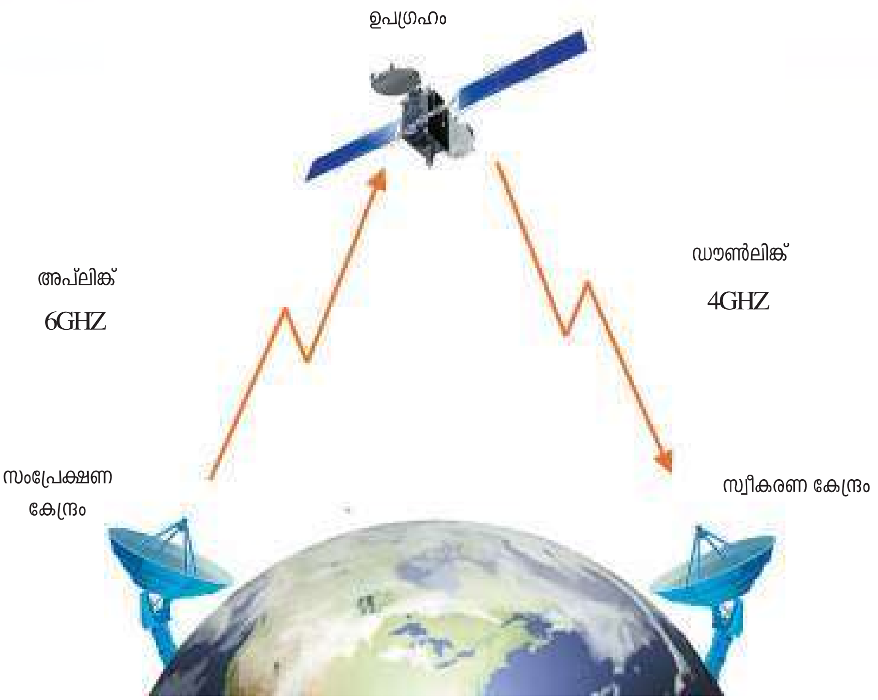
ചിത്രം 8.13 : വൈ - മാക്സ് സംപ്രേഷണം

വൈ-മാക്സ് സംപ്രേക്ഷണത്തിന്റെ സവിശേഷതകൾ

- 100 കണക്കിന് ഉപഭോക്താക്കൾക്ക് ഒരു സംപ്രേക്ഷണ നിലയവുമായി ബന്ധപ്പെടുവാൻ കഴിയുന്നു.
- 45 KM പരിധിയിൽ 70 Mbps വരെ വേഗത്തിൽ വിവരവിനിമയം നടക്കുന്നു.
- ഉപകരണങ്ങൾ തമ്മിൽ നേർരേഖയിൽ ഉള്ള വിനിമയം ഇവിടെ ആവശ്യമില്ല.
- സംപ്രേക്ഷണത്തെ മഴ, കാറ്റ് തുടങ്ങിയ പ്രതികൂല കാലാവസ്ഥ തടസ്സപ്പെടുത്തുന്നു.
- അമിതമായി ഊർജം ഉപയോഗിക്കുന്നു.
- സ്ഥാപിക്കുവാനും പ്രവർത്തിപ്പിക്കുവാനും ഉള്ള ഉയർന്ന ചെലവ്.

d. ഉപഗ്രഹ സംപ്രേക്ഷണം (Satellite link)

ദീർഘദൂര വിനിമയത്തിന് ഉപഗ്രഹശൃംഖല ഉപയോഗിച്ച് ഡാറ്റാ കൈമാറ്റം ചെയ്യപ്പെടുന്നു. സാധാരണയായി ഡാറ്റ നേർരേഖയിൽ ആണ് സഞ്ചരിക്കുന്നത്, ആയതിനാൽ ഭൂമിയെ വലം വെച്ച് വിദൂരതയിൽ ഉള്ള ഉദ്ദേശ്യ ലക്ഷ്യത്തിൽ എത്തുവാനുള്ള കഴിവ് ഡാറ്റയ്ക്ക് ഉണ്ടാവില്ല. ഇങ്ങനെയുള്ള സന്ദർഭങ്ങളിൽ ഡാറ്റയെ ഭൂസ്ഥിര ഉപഗ്രഹങ്ങളിലേക്ക് അയയ്ക്കുകയും, ഉപഗ്രഹം അടുത്ത ഉപഗ്രഹങ്ങളിലേക്കോ, വിദൂരതയിലുള്ള ലക്ഷ്യത്തിലേക്കോ എത്തിക്കുകയും ചെയ്യുന്നു. ഭൂമിയുടെ ഭ്രമണപഥത്തിൽ അതേ ദിശയിലും ഭ്രമണ വേഗതയിലും സഞ്ചരിക്കുന്ന ഉപഗ്രഹങ്ങളെ ഭൂസ്ഥിര ഉപഗ്രഹങ്ങൾ എന്ന് പറയുന്നു. ഇത്തരത്തിലുള്ള ഉപഗ്രഹങ്ങൾ ഭൂമിക്ക് മുകളിൽ നിശ്ചിത സ്ഥാനത്തുതന്നെ സ്ഥിരമായി കാണപ്പെടുന്നു. ഈ ഉപഗ്രഹങ്ങളിലെ ഇലക്ട്രോണിക് ഉപകരണങ്ങൾ ആയ ട്രാൻസ്പോണ്ടറുകൾ ഉപയോഗിച്ച് ഡാറ്റ സ്വീകരിക്കുകയും, തരംഗങ്ങളുടെ ശക്തി വർദ്ധിപ്പിച്ച് (ആംപ്ലിഫൈയിങ്), ഭൂമിയിലേക്ക് പുനഃ സംപ്രേക്ഷണം നടത്തുകയും ചെയ്യുന്നു.



ചിത്രം 8.14 : ഉപഗ്രഹ സംപ്രേഷണം

ഭൂമിയിൽ നിന്നും ഉപഗ്രഹത്തിലേക്കു തരംഗങ്ങളെ അയയ്ക്കുന്നതിനെ അപ് ലിങ്ക് എന്നും. ഉപഗ്രഹത്തിൽ നിന്ന് ഭൂമിയിലേക്ക് സംപ്രേക്ഷണം ചെയ്യുന്നതിനെ ഡൗൺ ലിങ്ക് എന്നും പറയുന്നു. ഒന്നിൽ കൂടുതൽ മൈക്രോ വേവ് ആവർത്തി തരംഗങ്ങൾ ഉപഗ്രഹസംപ്രേക്ഷണത്തിനായി ഉപയോഗിക്കുന്നു. അപ് ലിങ്കിനു വേണ്ടി ഉപയോഗിക്കുന്ന ആവൃത്തി 1.6GHz മുതൽ 30.0 GHz വരെയും ഡൗൺ ലിങ്കിനു വേണ്ടിയുള്ളത് 1.5 GHz മുതൽ 20 GHz വരെയുമാണ്. ഡൗൺ ലിങ്കിന്റെ ആവൃത്തി അപ്‌ലിങ്കിനേക്കാൾ കുറവായിരിക്കും.

ഉപഗ്രഹ സംവിധാനം ചെലവേറിയതാണ്, പക്ഷെ വളരെ കൂടിയ വ്യാപ്തിയിൽ സേവനം ലഭ്യമാക്കുവാൻ കഴിയും. പല രാജ്യങ്ങളിലും സാധാരണ, സർക്കാരുകളുടെയോ, സർക്കാർ അംഗീകരിച്ചസ്ഥാപനങ്ങളുടെയോ നിയന്ത്രണത്തിലായിരിക്കും വാർത്താ വിനിമയ ഉപഗ്രഹങ്ങൾ.

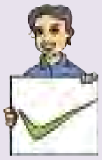
ഉപഗ്രഹ സംപ്രേക്ഷണത്തിന്റെ സവിശേഷതകൾ

- വളരെ വലിയ വ്യാപ്തിയിൽ ഉപഗ്രഹങ്ങൾ ഉപയോഗിച്ച് വിവര വിനിമയം നടത്തുവാൻ സാധിക്കുന്നു.
- ഈ സംവിധാനം ചെലവേറിയതാണ്.
- നിയമപരമായ അംഗീകാരവും അനുമതിയും ആവശ്യമാണ്.

നമുക്ക് ചെയ്യാം

ഇൻസ്റ്റിറ്റ്യൂട്ട് ഓഫ് ഇലക്ട്രിക്കൽ ആൻഡ് ഇലക്ട്രോണിക്സ് എഞ്ചിനീയേഴ്സ് എന്ന സംഘടന നിർവചിച്ച വയർലെസ് ബ്രോഡ്ബാൻഡ് സാങ്കേതികതയുടെ അടിസ്ഥാന നിർവചനമാണ് IEEE 802.16e എന്നത്. വയർലെസ് മെട്രോപൊളിറ്റൻ ഏരിയ ശൃംഖലയുടെ അടിസ്ഥാന നിർവചനം നൽകുവാനാണ് 1999 ൽ ഈ സംഘടന രൂപീകൃതമായത്.

സ്വയം പരിശോധിക്കാം



1. ഡാറ്റയുടെ വിനിമയ വ്യവസ്ഥയ്ക്ക് ആവശ്യമായ ഘടകങ്ങൾ ഏവ?
2. വിഭവങ്ങളുടെ പങ്കിടൽ (resource sharing) നിർവചിക്കുക.
3. കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ഉപയോഗിക്കുന്ന രണ്ടു വ്യത്യസ്ത വിനിമയ മാധ്യമങ്ങൾ ഏതൊക്കെ?
4. UTP/STP കേബിളിനെ കമ്പ്യൂട്ടറുമായി ബന്ധിപ്പിക്കുവാൻ ഉപയോഗിക്കുന്ന കണക്ടർ ഏത്?
5. വളരെ ദൂരത്തേക്ക് പ്രകാശ തരംഗങ്ങൾ ഉപയോഗിച്ച് ഡാറ്റ തരംഗങ്ങൾ അയക്കുവാനുള്ള ഗൈഡ് മാധ്യമമാണ് ________.
6. AM/FM റേഡിയോ സംപ്രേക്ഷണത്തിനും മൊബൈലിലും വിനിമയത്തിനായി ഉപയോഗിക്കുന്ന മാധ്യമമാണ് ________.
7. ടീവിയിലെ റിമോട്ട് കൺട്രോൾ, മൗസ് തുടങ്ങിയവയിൽ ഉപയോഗിക്കുന്ന മാധ്യമമാണ് ________.
8. സംപ്രേക്ഷണ ഉപകരണങ്ങൾ തമ്മിൽ നേർരേഖ കാഴ്ച ആവശ്യമില്ലാത്ത ഹ്രസ്വദൂര വിനിമയ സാങ്കേതികവിദ്യയാണ് ________.
9. ചെലവേറിയതും എന്നാൽ മറ്റു വയർലെസ്സ് സാങ്കേതികവിദ്യയേക്കാൾ കൂടുതൽ വ്യാപ്തിയിൽ സേവനം നടത്തുവാൻ കഴിയുന്നതുമായ വിവരവിനിമയ സാങ്കേതികവിദ്യയാണ് ________.

11.4 ഡാറ്റ വിനിമയ ഉപകരണങ്ങൾ (Data communication devices)

കമ്പ്യൂട്ടറും വിനിമയ മാധ്യമവും തമ്മിലുള്ള സമ്പർക്കമുഖ (interface) മായി ഒരു ഡാറ്റ വിനിമയ ഉപകരണം പ്രവർത്തിക്കുന്നു. ഈ ഉപകരണങ്ങൾ ഉപയോഗിച്ച് ഡാറ്റ തരംഗങ്ങളെ സംപ്രേക്ഷണം ചെയ്യുവാനും, സ്വീകരിക്കുവാനും, ശക്തി കൂട്ടുവാനും വിവിധ വിനിമയ മാധ്യമ ശൃംഖലകൾ ഉപയോഗിച്ച് വഴിതിരിച്ചു വിടുവാനും കഴിയുന്നു

11.4.1 നെറ്റ്‌വർക്ക് ഇന്റർഫേസ് കാർഡ് (Network Interface Card (NIC))

കമ്പ്യൂട്ടർ ശൃംഖലയിലേക്ക് ഒരു കമ്പ്യൂട്ടറിനെ ബന്ധിപ്പിക്കുവാനും വിവര വിനിമയം നടത്തുവാനും പ്രാപ്തമാക്കുന്ന ഉപകരണമാണ് NIC. കമ്പ്യൂട്ടറിനും ശൃംഖലയ്ക്കും

ഇടയിലുള്ള ഹാർഡ്‌വെയർ ഇന്റർഫേസ് ഉപകരണമായി ഇത് പ്രവർത്തിക്കുന്നു. ഇത് കമ്പ്യൂട്ടറിലെ പ്രത്യേക ഭാഗമായോ മദർബോർഡിന്റെ ഭാഗമായോ സ്ഥാപിച്ചിരിക്കുന്നു. കമ്പ്യൂട്ടർ ശൃംഖലയിലേക്കു ഡാറ്റയെ സജ്ജമാക്കുവാനും അയയ്ക്കുവാനും, സ്വീകരിക്കുവാനും നിയന്ത്രിക്കുവാനും NIC യ്ക്കു കഴിയും. ഡാറ്റയെ നിയന്ത്രിത രൂപത്തിലുള്ള ഘടകങ്ങളാക്കി മാറ്റുകയും, പ്രൊട്ടോക്കോളിനു വിധേയമായി പരിവർത്തനപ്പെടുത്തി, അയയ്ക്കേണ്ട മാധ്യമത്തിലേക്ക്, മേൽവിലാസം തിരിച്ചറിയുവാനുള്ള കഴിവുണ്ടാക്കി നൽകുകയും ചെയ്യുന്നു.

ചിത്രം. 8.15 (a) : NIC കാർഡ്

ചിത്രം. 8.15 (b) : വയർലസ്സ് NIC കാർഡ്

ചിത്രം 8.15(a), 8.5(b) എന്നിവയിൽ യഥാക്രമം ഒരു NIC കാർഡിന്റെയും ഒരു വയർലെസ്സ് NIC കാർഡിന്റെയും ചിത്രങ്ങൾ കൊടുത്തിരിക്കുന്നു. ചില NIC കാർഡുകൾ കേബിൾ ഉപയോഗിച്ചും (Ethernet), ചിലതു കേബിൾ ഇല്ലാതെയും (Wi-Fi) പ്രവർത്തിക്കുന്നു. കേബിൾ ശൃംഖലയിലേക്കുള്ള ജാക്കുകൾ ആണ് ഈതർനെറ്റ് NIC യിൽ ഉള്ളത്. എന്നാൽ വയർരഹിതമായ വിനിമയത്തിനുള്ള ബിൽറ്റ്-ഇൻ-ട്രാൻസ്മിറ്ററുകളും റീസിവറുകളും ആന്റിനയുമാണ് വൈ-ഫൈ NIC യിൽ ഉള്ളത്. NIC യ്ക്ക് 1Gbps വേഗതയിൽ ഡാറ്റ കൈമാറ്റം ചെയ്യുവാൻ കഴിയുന്നു.

11.4.2 ഹബ് (Hub)

ഒരു വയേർഡ് ശൃഖലയിൽ ഉൾപ്പെട്ടിരിക്കുന്ന കമ്പ്യൂട്ടറുകളെയും ഉപകരണങ്ങളെയും പരസ്പരം ബന്ധിപ്പിക്കുവാൻ ഉപയോഗിക്കുന്ന ഉപകരണമാണ് ഹബ്. ചെറുതും ലളിതവും നിഷ്ക്രിയവും വിലകുറഞ്ഞതുമായ ഉപകരണമാണ് ഇത്. ചിത്രം 8.16 പരിശോധിക്കുക. കമ്പ്യൂട്ടറുകളെ ഹബിലെ പോർട്ട് വഴി ഈതർനെറ്റ് കേബിൾ ഉപയോഗിച്ച് ബന്ധിപ്പിക്കുന്നു. ഹബിലേക്കു വരുന്ന വിവരങ്ങളുടെ പകർപ്പുകൾ പ്രസ്തുത ശൃംഖലയിൽ ഉൾപ്പെട്ടിരിക്കുന്ന എല്ലാ കമ്പ്യൂട്ടറുകളിലേക്കും കൈമാറുകയാണ് ഹബ് ചെയ്യുന്നത്. ഓരോ കമ്പ്യൂട്ടറിനും അവരവരുടെ ഡാറ്റ പാക്കറ്റുകൾ തിരിച്ചറിയുവാനുള്ള ബാധ്യതയുണ്ട്. ഒരു കമ്പ്യൂട്ടറിനെ ഉദ്ദേശിച്ച് അയച്ച പാക്കറ്റുകൾ അവ തന്നെ സ്വീകരിക്കേണ്ടതും മറ്റു കമ്പ്യൂട്ടറുകൾ അത് തിരസ്കരിക്കേണ്ടതും ആണ്. കമ്പ്യൂട്ടർ ശൃംഖലയിലെ എല്ലാ ഉപകരണങ്ങളിലേക്കും എല്ലാ ഡാറ്റയും അയയ്ക്കുന്നതിനാൽ ശൃംഖല തിരക്കേറിയതായിത്തീരുകയും ഡാറ്റ കൈമാ

ചിത്രം 8.16 : ഹബ്

റുവാനുള്ള ബാൻഡ്‌വിഡ്ത് കുറയുകയും ചെയ്യുന്നു എന്നതാണ് ഹബിന്റെ പ്രധാന പോരായ്മ.

11.4.3 സ്വിച്ച് (Switch)

നിരവധി കമ്പ്യൂട്ടറുകളെ പരസ്പരം ബന്ധിപ്പിച്ചു ഒരു ശൃംഖല രൂപീകരിക്കുവാൻ ശേഷിയുള്ള നിർമ്മിത ബുദ്ധിയോടുകൂടിയ ഉപകരണമാണ് സ്വിച്ച്. ഹബിനെക്കാൾ ഉയർന്ന പ്രവർത്തശേഷിയുള്ള ഉപകരണമാണ് സ്വിച്ച്. കാഴ്ചയിൽ ഹബിനോട് അടുത്ത സാമ്യമുണ്ട്. എന്നാൽ സ്വിച്ച് ഡാറ്റയ്ക്ക് എത്തിച്ചേരേണ്ട ലക്ഷ്യ സ്ഥാനം കൃത്യമായി ഉറപ്പു വരുത്തുകയും, ഡാറ്റ പാക്കറ്റുകൾ ഉദ്ദേശിച്ച സ്ഥാനത്തേയ്ക്ക് മാത്രം അയയ്ക്കുകയും ചെയ്യുന്നു. ശൃംഖലയിൽ ബന്ധിപ്പിച്ചിട്ടുള്ള എല്ലാ ഉപകരണങ്ങളുടെയും വിലാസം പട്ടികയായി സംഭരിച്ചു വെയ്ക്കുന്നതിനാലാണ് സ്വിച്ചിനു ഇങ്ങനെ പ്രവർത്തിക്കുവാൻ കഴിയുന്നത്. ശൃംഖലയിലെ ഒരു ഉപകരണത്തിലേക്കു ഡാറ്റ അയയ്ക്കുവാനായി, സ്വിച്ച് ഈ പാക്കറ്റിലെ വിലാസം മുൻകൂട്ടി ശേഖരിച്ച വിലാസങ്ങളുമായി താരതമ്യം ചെയ്യുന്നു, വിലാസം കണ്ടെത്തിയാൽ ലക്ഷ്യ സ്ഥാനത്തുള്ള ഉപകരണത്തിലേക്കു മാത്രം ഡാറ്റ അയയ്ക്കുന്നു. വളരെ തിരക്ക് കൂടിയ കമ്പ്യൂട്ടർ ശൃംഖലയിൽ ഹബ്ബിനെക്കാൾ നന്നായി സ്വിച്ച് പ്രവർത്തിക്കുന്നു. കാരണം വളരെ കുറഞ്ഞ അളവിൽ സന്ദേശങ്ങൾ അയയ്ക്കുന്നതിനാൽ ശൃംഖലയിൽ തിരക്ക് ഉണ്ടാകുന്നില്ല.

11.4.4 റിപ്പീറ്റർ (Repeater)

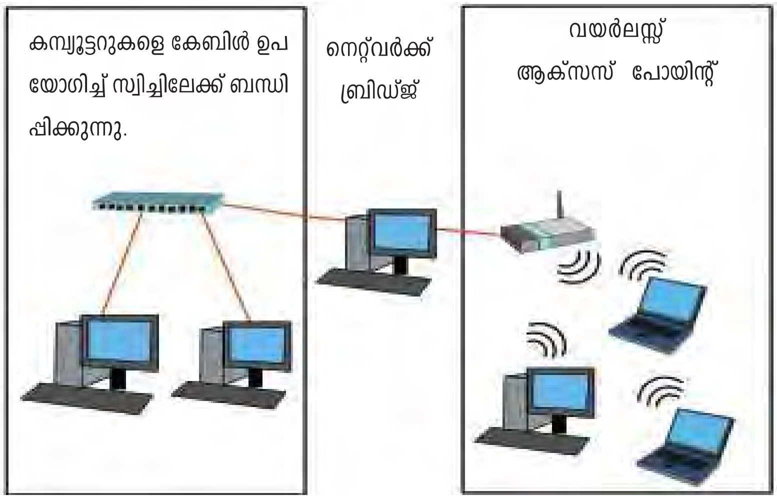
വിനിമയമാധ്യമത്തിലൂടെ വരുന്ന വൈദ്യുത കാന്തിക പ്രകാശ തരംഗങ്ങളെ ശക്തിപ്പെടുത്തുന്ന ഉപകരണമാണ് റിപ്പീറ്റർ. (ചിത്രം 8.17) വയേർഡ് മാധ്യമത്തിലൂടെയോ വയർലെസ്സിലൂടെയോ ഡാറ്റയ്ക്കു പരിമിതമായ ദൂരത്തേക്ക് മാത്രമേ ശക്തി ക്ഷയിക്കാതെ സഞ്ചരിക്കുവാൻ കഴിയൂ. ഇതിനു കാരണം നോയ്സ് ആണ്. റിപീറ്റർ ഇങ്ങനെ വരുന്ന തരംഗങ്ങളെ സ്വീകരിച്ചു ശക്തി കൂട്ടി ലക്ഷ്യ സ്ഥാനത്തേയ്ക്ക് പുനഃസംപ്രേക്ഷണം നടത്തുന്നു.

ചിത്രം 8.17: വയർലസ്സ് റിപ്പീറ്റർ

11.4.5 ബ്രിഡ്ജ് (Bridge)

ഒരു കമ്പ്യൂട്ടർ ശൃംഖലയെ പല വിഭാഗങ്ങളാക്കി വേർതിരിക്കുവാൻ ഉപയോഗിക്കുന്ന ഉപകരണമാണ് ബ്രിഡ്ജ്. നിലവിലുള്ള ശൃംഖലയെ പല വിഭാഗങ്ങളായി തരംതിരിക്കുകയും ഇവയെ ബ്രിഡ്ജ് ഉപയോഗിച്ച് ബന്ധിപ്പിക്കുകയും ചെയ്യുന്നു. ശൃംഖലയിലുള്ള ട്രാഫിക് കുറയ്ക്കുവാൻ ഇത് സഹായിക്കുന്നു. ഒരു ബ്രിഡ്ജിൽ ഡാറ്റ പാക്കറ്റുകൾ എത്തുമ്പോൾ, അതിലെ മേൽവിലാസം പരിശോധിച്ചു ബ്രിഡ്ജിന്റെ ഏതു ഭാഗത്തെ ഇവ പ്രതിനിധാനം ചെയ്യുന്നു എന്ന് കണ്ടുപിടിക്കുന്നു (ഇതേ ഭാഗത്തേക്കുള്ള നോഡുകളിലേക്കോ അതോ മറുഭാഗത്തേയ്ക്കോ എന്ന്). ഒരു മേഖലയെ പ്രതിനിധാനം

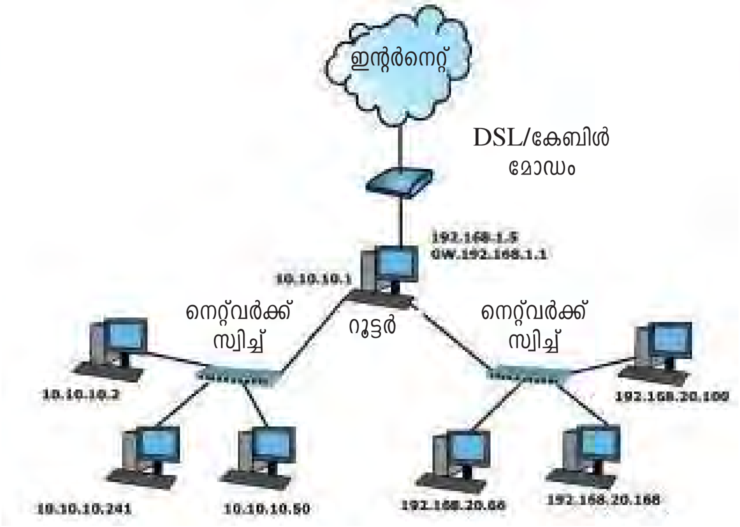
ചെയ്യുന്ന ഡാറ്റ പാക്കറ്റുകളെ മാത്രം ആ ഭാഗത്തേയ്ക്ക് ബ്രിഡ്ജ് കടത്തി വിടുന്നു. ബാക്കി ഉള്ളവ ഒഴിവാക്കുന്നു. ബ്രിഡ്ജ് വഴി കടന്നു പോകുന്ന പാക്കറ്റുകൾ മറു ഭാഗത്തുള്ള എല്ലാ നോഡുകളിലേക്കും പ്രക്ഷേപണം ചെയ്യുകയും, ലക്ഷ്യത്തിലുള്ള നോഡുകൾ മാത്രം അവ സ്വീകരിക്കുകയും ചെയ്യുന്നു. ചിത്രം 8.18 ബ്രിഡ്ജിന്റെ ധർമ്മങ്ങൾ വിശദമാക്കുന്നു.



ചിത്രം 8.18: ബ്രിഡ്ജ്

11.4.6 റൂട്ടർ (Router)

ഒരേ വിഭാഗത്തിൽപ്പെട്ടതും ഒരേ പോലുള്ള പെരുമാറ്റ ചട്ടങ്ങൾ ഉള്ളതുമായ രണ്ടു ശൃംഖലകളെ ബന്ധിപ്പിക്കുന്ന ഉപകരണമാണ് റൂട്ടർ. ഡാറ്റയ്ക്ക് സഞ്ചരിക്കുവാനാവശ്യമായ ഉചിതമായ പാത കണ്ടെത്തുന്നതിനും അങ്ങനെ ശൃംഖലയിലെ ട്രാഫിക്കിന്റെ അളവ് കുറയ്ക്കുന്നതിനും ഇവയ്ക്കു കഴിയുന്നു. ബ്രിഡ്ജിന്റെ പ്രവർത്തന രീതികളോട് ഇവയ്ക്കു സാമ്യം ഉണ്ടെങ്കിലും അവയേക്കാൾ കഴിവ് ഇതിനുണ്ട്. റൂട്ടറിനു ഉപകരണത്തിന്റെ വിലാസവും, ശൃംഖലയുടെ വിലാസവും പരിശോധിക്കുവാനുള്ള കഴിവുള്ളതോടൊപ്പം അൽഗോരിതം ഉപയോഗിച്ച് ഏറ്റവും ഉചിതമായ പാതയിലൂടെ പാക്കറ്റുകളെ ലക്ഷ്യ സ്ഥാനത്ത് എത്തിക്കുവാനും സാധിക്കുന്നു. ചിത്രം 8.19 റൂട്ടറിന്റെ ധർമം വിശദമാക്കുന്നു.



ചിത്രം 8.19: റൂട്ടർ

11.4.7 ഗേറ്റ്‌വേ (Gateway)

വിവിധ തരത്തിലും പ്രൊട്ടോക്കോളിലും പ്രവർത്തിക്കുന്ന ശൃംഖലകളെ ബന്ധിപ്പിക്കുവാൻ ഗേറ്റ്‌വേ ഉപയോഗിക്കുന്നു. ചിത്രം 8.20 പരിശോധിക്കുക. ഒരു തരത്തിലുള്ള പ്രൊട്ടോക്കോളിനെ മറ്റൊരു തരത്തിലേക്ക് വിവർത്തനം ചെയ്യുവാനും ഇവയ്ക്കു കഴിയുന്നു. ഒരു ശൃംഖലയിൽ നിന്ന് മറ്റൊരു ശൃംഖലയിലേക്കുള്ള പ്രവേശന കവാടമായി ഇത് പ്രവർത്തിക്കുന്നു. റൂട്ടറിനു സമാനമായ പ്രവർത്തനരീതിയാണ് ഇവയ്ക്കും ഉള്ളത്. ഉപകരണത്തിന്റെയും ശൃംഖലയുടെയും വിലാസം പരിശോധിക്കുകയും അൽഗോരിതത്തിന്റെ സഹായത്താൽ ഉചിതമായ പാത സ്വീകരിച്ചു പാക്കറ്റുകളെ ലക്ഷ്യസ്ഥാനത്തു എത്തിക്കുകയും ചെയ്യുന്നു. വ്യത്യസ്തമായ പ്രൊട്ടോക്കോളുള്ള ശൃംഖലകൾ തമ്മിൽ ഒരു പരസ്പരധാരണ ഉണ്ടായിരിക്കണം. ഒരു ഗേറ്റ്‌വേയ്ക്ക് ശൃംഖലയുടെ വിലാസഘടനയെ കുറിച്ച് ശരിയായ ധാരണ ഉള്ളതിനാൽ തടസ്സം ഇല്ലാതെ തുടർച്ചയായി പാക്കറ്റുകളെ ശൃംഖലയിലെ നോഡുകൾക്കിടയിൽ കൈമാറ്റം ചെയ്യുവാനുള്ള കഴിവുണ്ട്.

ചിത്രം 8.20 : ഗേറ്റ് വേ

11.5 ഡാറ്റ ടെർമിനൽ ഉപകരണങ്ങൾ (Data Terminal Equipments (DTE))

കമ്പ്യൂട്ടറിലേക്കും പുറത്തേയ്ക്കും ഉള്ള ഡാറ്റയുടെ ഒഴുക്കിനെ നിയന്ത്രിക്കുന്ന ഉപകരണമാണ് ഡാറ്റ ടെർമിനൽ ഉപകരണങ്ങൾ (Data Terminal Equipments (DTE)). ഈ ഉപകരണങ്ങൾ ടെലികമ്മ്യൂണിക്കേഷൻ ലിങ്കുമായി സംപ്രേക്ഷണ മാധ്യമത്തിന്റെ അഗ്രഭാഗത്തു ബന്ധിപ്പിച്ചിരിക്കുന്നു. പൊതുവായി ഉപയോഗിക്കുന്ന DTE ഉപകരണങ്ങളായ മോഡം, മൾട്ടിപ്ലെക്സർ എന്നിവയെ കുറിച്ച് ഇവിടെ ചർച്ച ചെയ്യുന്നു.

11.51. മോഡം (Modem)

ടെലിഫോൺ ലൈൻ ഉപയോഗിച്ച് കമ്പ്യൂട്ടറുകൾ തമ്മിൽ വിനിമയം നടത്തുവാൻ സഹായിക്കുന്ന ഇലക്ട്രോണിക് ഉപകരണമാണ് മോഡം. (ചിത്രം 8.21). മോഡുലേറ്റർ (Modulator)ഡി മോഡുലേറ്റർ (Demodulator) എന്നതിന്റെ ചുരുക്കമാണ് മോഡം (Modem). കമ്പ്യൂട്ടറിൽ നിന്ന് സ്വീകരിക്കുന്ന ഡിജിറ്റൽ സിഗ്നലിനെ ടെലിഫോൺ ലൈനിലൂടെ കടന്നുപോകുവാനായി അനലോഗ് സിഗ്നലാക്കി മാറ്റുന്നു (Modulation). അതോടൊപ്പം ടെലിഫോൺ ലൈൻ വഴിവരുന്ന അനലോഗ് സിഗ്നലിനെ ഡിജിറ്റിലായി പരിവർത്തനം ചെയ്തത് കംപ്യൂട്ടറിലേക്കു നൽകുന്നു (Demodulation). ടെലിഫോൺ ലൈൻ വഴി വിവരങ്ങൾ അയയ്ക്കുകയും സ്വീകരിക്കുകയും ചെയ്യുന്നതിന്റെ വേഗതയെ അടിസ്ഥാനമാക്കിയാണ് മോഡത്തിന്റെ വേഗത നിർണ്ണയിക്കുന്നത്. മോഡത്തിന്റെ വേഗത അളക്കുന്നത് ബിറ്റ്സ്/സെക്കന്റ് (bits / second) ആണ്.

ചിത്രം 8.21 : മോഡം ഉപയോഗിച്ചുള്ള ആശയവിനിമയം

11.5.2 മൾട്ടിപ്ലെക്സർ/ഡി മൾട്ടിപ്ലെക്സർ (Multiplexer/Demultiplexer)

ഒറ്റ കേബിൾ ഉപയോഗിച്ച് 200 ഓ അതിലധികമോ ചാനലുകളെ കൈകാര്യം ചെയ്യുന്നത് നിങ്ങളെ എപ്പോഴെങ്കിലും അതിശയിപ്പിച്ചിട്ടുണ്ടോ? ഇതിനെയാണ് മൾട്ടിപ്ലെക്സിങ് എന്ന് പറയുന്നത്. ഇതേ രീതിയിലാണ് ശൃംഖലയിലുള്ള ഡാറ്റ കൈമാറ്റവും. ഭൗതിക മാധ്യമത്തിലൂടെ ഒന്നിലേറെ തരംഗങ്ങളെ സംയോജിപ്പിച്ച് സങ്കീർണതയേറിയ ഒറ്റ തരംഗമാക്കി മാറ്റി ഒരേ സമയം വിടുന്നതിനെ മൾട്ടിപ്ളെക്സിങ് എന്നും, മറുഭാഗത്ത് ഈ തരംഗത്തെ വിഘടിപ്പിച്ചു പ്രത്യേക തരംഗങ്ങളാക്കി മാറ്റുന്നതിനെ ഡി-മൾട്ടിപ്ലക്സിങ് എന്നും പറയുന്നു. ഭൗതിക മാധ്യമത്തെ മൾട്ടിപ്ളെക്സിങ് വിവിധ ഭാഗങ്ങളാക്കി മാറ്റുന്നു. ഇതിനെ ഫ്രീക്വൻസി ചാനൽ എന്ന് പറയുന്നു.

മൾട്ടിപ്ലെക്സർ വിവിധ ഉറവിടത്തിൽ നിന്നുള്ള തരംഗങ്ങളെ സംയോജിപ്പിച്ച്, മാധ്യമത്തിന്റെ വിവിധ ചാനലുകൾ വഴി അയയ്ക്കുന്നു. സംയോജിപ്പിച്ച തരംഗങ്ങൾ മാധ്യമത്തിലൂടെ ഒരേ സമയത്തു സഞ്ചരിക്കുന്നു. ലക്ഷ്യ സ്ഥാനത്തു ഇവയെ വിഭജിച്ച് വെവ്വേറെ തരംഗങ്ങളാക്കി, ഓരോ തരംഗത്തിനും എത്തേണ്ട സ്ഥലത്തേയ്ക്ക് അയയ്ക്കുന്നു. ചിത്രം 8.22 മൾട്ടിപ്ലെക്സറിന്റെയും ഡി-മൾട്ടിപ്ലെക്സറിന്റെയും പ്രവർത്തനം വിവരിക്കുന്നു.

ചിത്രം 8.22 : മൾട്ടിപ്ലക്സർ/ഡി- മൾട്ടിപ്ലക്സർ

നമുക്ക് ചെയ്യാം

പത്തു നോഡുകൾ ഉള്ള ഒരു ചെറിയ കമ്പ്യൂട്ടർ ശൃംഖല നിർമിക്കുവാൻ ആവശ്യമായ ഉപകരണങ്ങളുടെയും മാധ്യമങ്ങളുടെയും പട്ടിക തയ്യാറാക്കുക .

സ്വയം പരിശോധിക്കാം

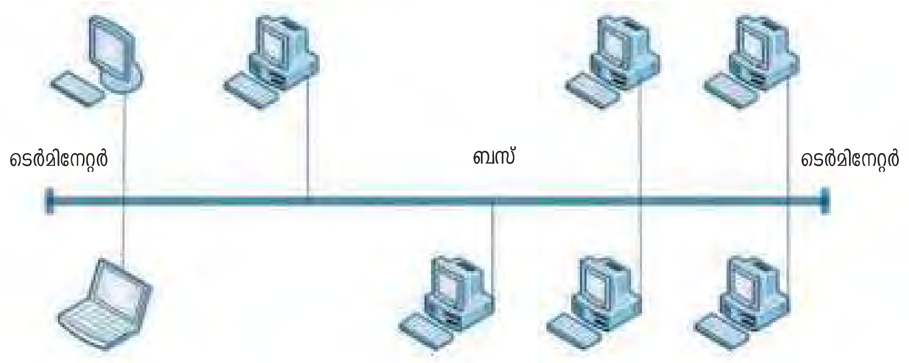
1. ഹബും സ്വിച്ചും തമ്മിൽ താരതമ്യം ചെയ്യുക.
2. റിപീറ്ററിന്റെ ആവശ്യകത എന്ത്?
3. ഒരേപോലുള്ള രണ്ടു ശൃംഖലകളെ തമ്മിൽ ബന്ധിപ്പിക്കുന്ന ഉപകരണമാണ് ________.
4. റൂട്ടറും ബ്രിഡ്ജും തമ്മിലുള്ള വ്യത്യാസം എന്താണ്?
5. വ്യത്യസ്ത പ്രൊട്ടോക്കോൾ ഉള്ള രണ്ടു വ്യത്യസ്ത ശൃംഖലകളെ പരസ്പരം ബന്ധിപ്പിക്കുന്ന ഉപകരണമാണ് ________.
6. ടെലിഫോൺ ലൈനിലൂടെ രണ്ടു കമ്പ്യൂട്ടറുകൾ തമ്മിൽ വിവരവിനിമയം നടത്തുവാൻ ഉപയോഗിക്കുന്ന ഇലക്ട്രോണിക് ഉപകരണമാണ് ________.

11.6 നെറ്റ്‌വർക്ക് ടോപ്പോളജികൾ (Network topologies)

പത്തു കമ്പ്യൂട്ടറുകൾ അടങ്ങിയ ഒരു ശൃംഖല രൂപകൽപ്പന ചെയ്യണമെന്ന് കരുതുക. ഏതൊക്കെ വിധത്തിൽ നമുക്ക് ഇവയെ പരസ്പരം ബന്ധിപ്പിക്കാം? ലഭ്യമായ മാധ്യമങ്ങളും ചില നിബന്ധനകളും വഴി നമുക്ക് ഇവയെ പല വിധത്തിൽ ബന്ധിപ്പിക്കാം ഭൗതികമായി കമ്പ്യൂട്ടറുകളെ പരസ്പരം ബന്ധിപ്പിച്ചു ശൃംഖല രൂപ കൽപ്പന ചെയ്യുന്ന രീതിയെ ടോപ്പോളജി എന്ന് പറയുന്നു. ബസ്, റിങ്, സ്റ്റാർ, മെഷ് എന്നിവയാണ് പ്രധാന ടോപ്പോളജികൾ.

11.6.1 ബസ് ടോപ്പോളജി (Bus topology)

ബസ് ടോപ്പോളജിയിൽ (ചിത്രം 11.23) പ്രധാന കേബിൾ ആയ ബസിലേയ്ക്ക് നോഡുകളെ ബന്ധിപ്പിച്ചിരിക്കുന്നു. ഒരു നോഡിനു ഡാറ്റ അയയ്ക്കണമെങ്കിൽ, അത് ബസിലേയ്ക്ക് അയയ്ക്കുന്നു. ബസിന്റെ എല്ലാ ഭാഗത്തും ഈ ഡാറ്റ എത്തിച്ചേരുന്നു. എല്ലാ നോഡുകളും ബസിൽ വരുന്ന ഡാറ്റയെ പരിശോധിക്കുന്നു. ഏതു നോഡിലേക്കാണോ ഡാറ്റ അയച്ചിരിക്കുന്നത് അത് ഡാറ്റയെ സ്വീകരിക്കുന്നു. ബസിന്റെ അഗ്രഭാഗങ്ങളിൽ ഒരു ചെറിയ ഉപകരണമായ ടെർമിനേറ്റർ ഘടിപ്പിച്ചിരിക്കുന്നു. തരംഗങ്ങൾ ബസിന്റെ അഗ്രഭാഗത്തു എത്തിയാൽ അവയെ ടെർമിനേറ്റർ ആഗിരണം ചെയ്തു നീക്കം ചെയ്യുന്നു. ഈ അവസരത്തിൽ ബസ് അടുത്ത തരംഗങ്ങളെ വഹിക്കുവാൻ പൂർണ സജ്ജമായിത്തീരുന്നു. കേബിളിലേക്കുള്ള തരംഗങ്ങളുടെ പ്രതിഫലനം ഒഴിവാക്കുവാനും, തരംഗങ്ങൾ തമ്മിൽ കൂടിച്ചേരുന്ന സാഹചര്യം ഒഴിവാക്കുവാനും ഇതിനാൽ സാധിക്കുന്നു. ഒരു നോഡിൽ നിന്ന് മറ്റെല്ലാ നോഡുകളിലേക്കും തരംഗങ്ങളെ അയയ്ക്കുന്നതിനെ ബ്രോഡ്കാസ്റ്റിംഗ് എന്ന് പറയുന്നു.

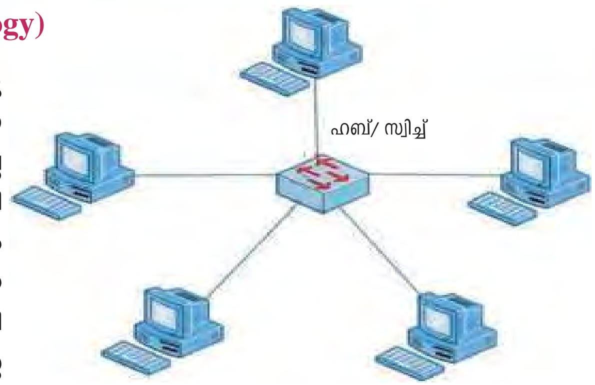


ചിത്രം 8.23: ബസ് ടോപ്പോളജി

ബസ് ടോപ്പോളജിയുടെ സവിശേഷതകൾ

- അനായാസമായി സ്ഥാപിക്കാം.
- ഇവ നിർമ്മിക്കുവാൻ വളരെ കുറച്ച് കേബിളുകൾ ഉപയോഗിക്കുന്നതിനാൽ ചെലവ് കുറവാണ്.
- ഒരു നോഡിന്റെ തകരാറ് ശൃംഖലയെ ബാധിക്കുന്നില്ല.
- ബസിന്റെയോ ടെർമിനേറ്ററിന്റെയോ തകരാറ് ശൃംഖലയെ മൊത്തമായി ബാധിക്കുന്നു.
- തകരാറ് കണ്ടെത്തുക എന്നത് ശ്രമകരമാണ്.
- ഒരു നോഡിനു മാത്രമേ ഒരു സമയത്തു ഡാറ്റ അയയ്ക്കുവാൻ കഴിയൂ.

11.6.2 സ്റ്റാർ ടോപ്പോളജി (Star topology)



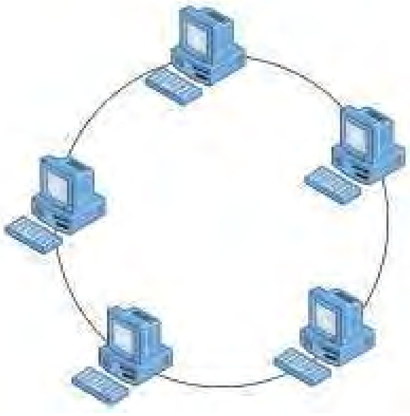
ചിത്രം 8.24 : സ്റ്റാർ ടോപ്പോളജി

ചിത്രം 8.24 ചിത്രീകരിച്ചിരിക്കുന്നതു പോലെ സ്റ്റാർ ടോപ്പോളജിയിൽ ഓരോ നോഡും ഹബ്ബിലേക്കോ അല്ലെങ്കിൽ സ്വിച്ചിലേക്കോ നേരിട്ട് ബന്ധിപ്പിച്ചിരിക്കുന്നു. ഇതിൽ ഏതെങ്കിലും ഒരു നോഡിനു ഡാറ്റ അയയ്ക്കണമെങ്കിൽ അത് സ്വിച്ചിലേക്കോ ഹബിലേക്കോ അയയ്ക്കുന്നു. ഹബിന്റെ കാര്യത്തിൽ ഈ തരംഗങ്ങളെ എല്ലാ നോഡുകളിലേക്കും സംപ്രേക്ഷണം ചെയ്യുകയും, ഉദ്ദേശിച്ച നോഡുകൾ മാത്രം അവയെ സ്വീകരിക്കുകയും ചെയ്യുന്നു. സ്വിച്ചിന്റെ കാര്യത്തിലാണെങ്കിൽ ഈ തരംഗങ്ങളെ ഉദ്ദേശിച്ച നോഡിലേക്ക് മാത്രം അയയ്ക്കുന്നു.

സ്റ്റാർ ടോപ്പോളജിയുടെ സവിശേഷതകൾ

- ബസ് ടോപ്പോളജിയെ അപേക്ഷിച്ചു പ്രായോഗിക ക്ഷമത കൂടുതലാണ്.
- അനായാസമായി സ്ഥാപിക്കാം.
- തകരാറ് കണ്ടെത്തുക എളുപ്പമാണ്.
- കേന്ദ്രസ്ഥാനത്തുള്ള ഹബ്/സ്വിച്ച് ന്റെയും ബന്ധിപ്പിക്കുവാനുള്ള കഴിവ് അനുസരിച്ചു ശൃംഖലയിൽ നോഡുകളെ കൂട്ടിച്ചേർത്തു ശൃംഖല വിപുലീകരിക്കാം.
- ഹബ്ബിനോ/സ്വിച്ചിനോ തകരാറുണ്ടായാൽ ശൃംഖലയെ മൊത്തത്തിൽ ബാധിക്കുന്നു
- ബസ് ടോപ്പോളജിയെ അപേക്ഷിച്ച് ശൃംഖല നിർമ്മിക്കുവാൻ കൂടുതൽ കേബിൾ ആവശ്യമാണ്.

11.6.3 റിങ് ടോപ്പോളജി (Ring topology)



ചിത്രം 8.25 : റിങ് ടോപ്പോളജി

റിങ് ടോപ്പോളജിയിൽ നോഡുകളെ കേബിൾ ഉപയോഗിച്ച് വൃത്താകൃതിയിൽ ബന്ധിപ്പിച്ചിരിക്കുന്നു. തുടക്കമോ അവസാനമോ ഇല്ലാത്ത ഒരു വൃത്താകൃതിയാണ് റിങ് ടോപ്പോളജിക്കുള്ളത് (ചിത്രം 8.25). ടെർമിനേറ്ററിന്റെ ആവശ്യം റിങ് ടോപ്പോളജിക്ക് ഇല്ല. ഒരു ദിശയിലേക്കു മാത്രമാണ് ഡാറ്റ സഞ്ചരിക്കുന്നത്. ഒരു നോഡിൽ നിന്ന് മറ്റൊരു നോഡിൽ എത്തുന്ന തരംഗങ്ങളെ പുനരുജ്ജീവിപ്പിച്ച് അടുത്തതിലേക്ക് അയയ്ക്കുന്നു. ഉദ്ദേശിച്ച നോഡിൽ എത്തുന്നതുവരെ ഈ

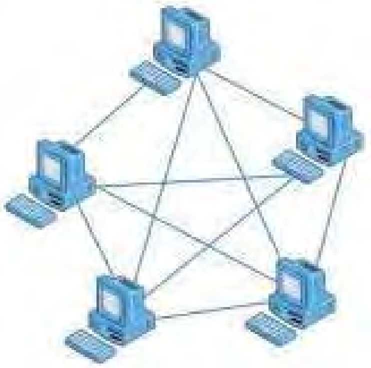
പ്രക്രിയ തുടരുന്നു. എല്ലാ നോഡുകളിലൂടെയും സഞ്ചരിക്കുന്ന തരംഗങ്ങൾ അവസാനം സംപ്രേക്ഷണം ചെയ്ത നോഡിൽ തിരിച്ചെത്തുകയും, അവിടെ നിന്നു ഇവയെ നീക്കം ചെയ്യുകയും ചെയ്യുന്നു.

റിങ് ടോപ്പോളജിയുടെ സവിശേഷതകൾ

- ഓരോ നോഡും തരംഗങ്ങളുടെ ശക്തി വർധിപ്പിക്കുന്നതിനാൽ, തരംഗങ്ങളുടെ ശക്തി വർധിപ്പിക്കേണ്ടി വരുന്നില്ല.
- വളരെ കുറച്ച് മാത്രം കേബിൾ ഉപയോഗിക്കുന്നതിനാൽ ചെലവ് കുറവാണ്.
- ഒരു നോഡ് തകരാറിലായാൽ അത് ശൃംഖലയെ മുഴുവനായി ബാധിക്കുന്നു.
- ശൃംഖലയിലേക്ക് പുതിയ നോഡുകളെ കൂട്ടിച്ചേർക്കുക പ്രയാസകരമാണ്

11.6.4 മെഷ് ടോപ്പോളജി (Mesh topology)

മെഷ് ടോപ്പോളജിയിൽ എല്ലാ നോഡുകളെയും പരസ്പരം ബന്ധിപ്പിച്ചിരിക്കുന്നു. ചിത്രം 8.26 കാണിച്ചിരിക്കുന്നത് പോലെ രണ്ടു നോഡുകൾക്കിടയിൽ ഒന്നിലേറെ പാതകൾ ഉണ്ടായിരിക്കും. ഒരു പാതയിൽ തടസ്സമുണ്ടായാലും മറ്റൊരു പാതയിലൂടെ ഡാറ്റ ലക്ഷ്യസ്ഥാനത്തു എത്തിച്ചേരുന്നു.



ചിത്രം 8.26 : മെഷ് ടോപ്പോളജി

മെഷ് ടോപ്പോളജിയുടെ സവിശേഷതകൾ

- രണ്ടു നോഡുകൾക്കിടയിൽ ഉള്ള പാത തകരാറായാലും ശൃംഖലയ്ക്കു തകരാറ് ഉണ്ടാകുന്നില്ല.
- കൂടുതൽ കേബിൾ വേണ്ടതിനാൽ ചെലവ് കൂടുതലാണ്.
- വളരെ സങ്കീർണവും കൈകാര്യം ചെയ്യുവാൻ പ്രയാസവുമാണ്.

നമുക്ക് ചെയ്യാം

നിങ്ങളുടെ സ്കൂൾ ലാബിലെ ശൃംഖലയുടെ ക്രമീകരണരീതി എന്താണ് എന്ന് മനസിലാക്കുക.

11.7 വിവിധതരം ശൃംഖലകൾ (Type of networks)

ഒരു കമ്പ്യൂട്ടർ ശൃംഖല ഭൗമ വിസ്തൃതിയിൽ പല അളവുകളിലായി വ്യാപിച്ചു കിടക്കുന്നു. ഇത് വേണമെങ്കിൽ ഒരു മേശയുടെ മുകളിലോ ഒരു റൂമിലോ ഒരു കെട്ടിടത്തിലോ ഒരു നഗരത്തിലോ, ഒരു രാജ്യത്തിനുള്ളിലോ ഭൂഖണ്ഡങ്ങളിലോ ലോകം മുഴുവനുമോ വ്യാപിച്ചു കിടക്കാം. കമ്പ്യൂട്ടർ ശൃംഖലയെ അവയുടെ വ്യാപനത്തെ അടിസ്ഥാനമാക്കി ചുവടെ ചേർത്ത രീതിയിൽ വേർതിരിക്കാം.

- PAN – പേർസണൽ ഏരിയ നെറ്റ്‌വർക്ക്
- LAN – ലോക്കൽ ഏരിയ നെറ്റ്‌വർക്ക്
- MAN – മെട്രോപൊളിറ്റൻ ഏരിയ നെറ്റ്‌വർക്ക്
- WAN – വൈഡ് ഏരിയ നെറ്റ്‌വർക്ക്

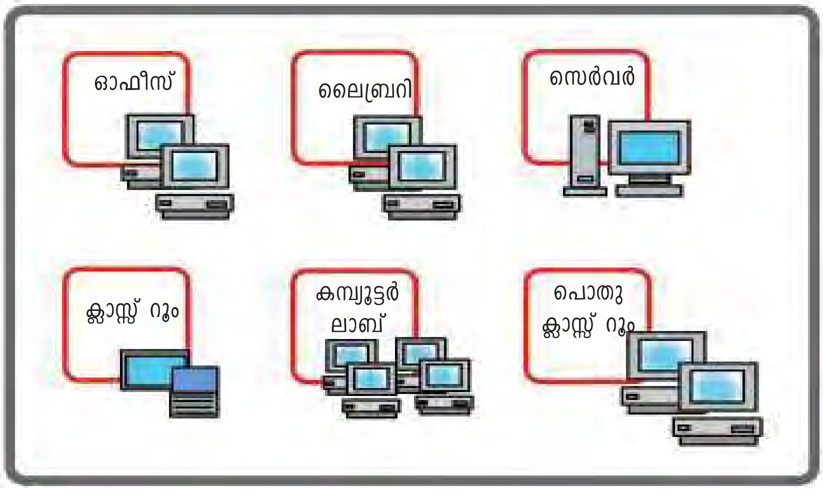
11.7.1 പേർസണൽ ഏരിയ നെറ്റ്‌വർക്ക് (Personal Area Network)

ചിത്രം 8.27: പാൻ

ഒരു വ്യക്തിയുടെ പരിധിയിലുള്ള വിനിമയ ഉപകരണങ്ങളുടെ (കമ്പ്യൂട്ടർ, മൊബൈൽ, ടാബ്‌ലെറ്റ്, പ്രിൻറർ എന്നിവ) ശൃംഖലയാണ് PAN. ഏതാനും മീറ്റർ വൃത്ത പരിധിക്കുള്ളിൽ ഇവ വ്യാപിച്ചു കിടക്കുന്നു. ചിത്രം 8.27 ഒരു പാട്ട് ഒരു മൊബൈലിൽ നിന്ന് മറ്റൊന്നിലേക്കോ, ഒരു കമ്പ്യൂട്ടറിൽ നിന്ന് MP3 പ്ലെയറിലേക്കോ അയയ്ക്കുവാൻ നാം PAN ശൃംഖല ഉണ്ടാക്കാറുണ്ട്. PAN ശൃംഖല ഉണ്ടാക്കുവാൻ ഗൈഡഡ് മാധ്യമവും (USB), അൺ ഗൈഡഡ് മാധ്യമവും (ബ്ലൂടൂത്ത്, ഇൻഫ്രാറെഡ്) ഉപയോഗിക്കാം.

11.7.2 ലോക്കൽ ഏരിയ നെറ്റ്‌വർക്ക് (Local Area Network)

ഒരു LAN ശൃംഖലയിലെ വിവര വിനിമയത്തിനും കമ്പ്യൂട്ടിങ്ങിനുമുള്ള ഉപകരണങ്ങൾ ഒരു മുറിയ്ക്കുള്ളിലോ, ഒരു കെട്ടിടത്തിനുള്ളിലോ ഒരു സ്ഥാപന പരിധിയ്ക്ക് ഉള്ളിലോ ആയിരിക്കും പരസ്പരം ബന്ധിപ്പിച്ചിരിക്കുന്നത്. ഏതാനും മീറ്ററോ ഏതാനും കിലോ മീറ്ററോ വൃത്ത പരിധിയ്ക്കുള്ളിൽ ആയിരിക്കും ഇവയുടെ പ്രവർത്തനം. സാധാരണയായി ഓഫീസിലും സ്കൂളിലും റൂമിലും ഒരു LAN ശൃംഖലമാത്രമാണ് ഉണ്ടാകാറു



ചിത്രം 8.28 : ലോക്കൽ ഏരിയ നെറ്റ്‌വർക്ക്

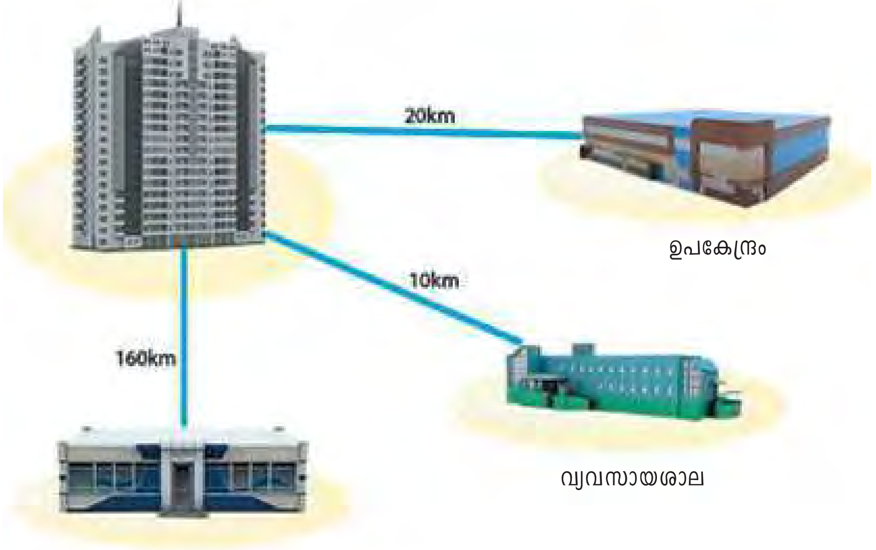
ള്ളത്, എന്നാൽ ഒരു കെട്ടിടത്തിൽ തന്നെ ഒന്നിൽ കൂടുതൽ LAN ചിലപ്പോൾ ഉണ്ടായെന്നു വരാം. (ചില സ്കൂളുകളിൽ ഓരോ ലാബിലും ഓരോ LAN ശൃംഖല ഉള്ളതു പോലെ). ചിലപ്പോൾ LAN അടുത്തടുത്ത കെട്ടിടത്തിലേക്കും വ്യാപിച്ചിരിക്കും

LAN ശൃംഖലയുടെ നിയന്ത്രണവും പരിപാലനവും, ഒരു വ്യക്തിയുടെയോ, ഒരു സ്ഥാപനത്തിന്റെയോ ഉടമസ്ഥതയിലായിരിക്കും.

ഗൈഡഡ് മാധ്യമം (വയേർഡ് മീഡിയ) (UTP കേബിളുകൾ കോയാക്സിൽ കേബിളുകൾ തുടങ്ങിയവ) ഉപയോഗിച്ചും വയർലെസ്സ് മാധ്യമം (ഇൻഫ്രാറെഡ്, റേഡിയോ തരംഗങ്ങൾ തുടങ്ങിയവ) ഉപയോഗിച്ചും ലാൻ സ്ഥാപിക്കാവുന്നതാണ്. അൺഗൈഡഡ് മാധ്യമം (Unguided Media) ഉപയോഗിച്ചാണ് LAN സ്ഥാപിക്കുന്നതെങ്കിൽ അതിനെ വയർലെസ്സ് LAN (WLAN (Wireless LAN)) എന്ന് വിളിക്കാം.

11.7.3 മെട്രോപൊളിറ്റൻ ഏരിയ നെറ്റ്വർക്ക് (Metropolitan Area Network (MAN))

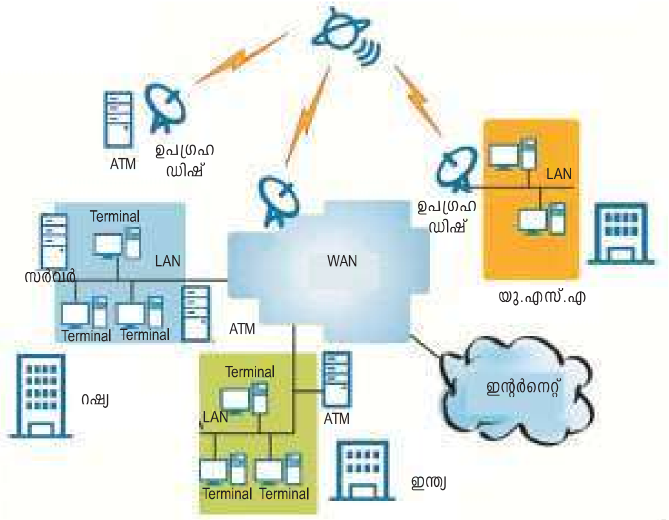
MAN ശൃംഖലയുടെ കമ്പ്യൂട്ടിങ്ങും പ്രവർത്തനവും വിനിമയ ഉപകരണങ്ങളുടെ വ്യാപനവും ഒരു നഗര പരിധിക്കുള്ളിൽ നിൽക്കുന്നു. ഇതിന്റെ വൃത്ത പരിധി നൂറു കിലോമീറ്റർ വരെ വ്യാപിച്ചു കിടക്കും. ലാൻ (LAN) ശൃംഖലകളെയും, സ്വകാര്യ കമ്പ്യൂട്ടറുകളെയും പരസ്പരം ബന്ധിപ്പിച്ചാണ് MAN സ്ഥാപിക്കുന്നത്. എല്ലാവിധ മാധ്യമങ്ങളും (ഗൈഡഡും അൺ-ഗൈഡഡും) ഇതിനായി ഉപയോഗിക്കുന്നു. MAN ന്റെ ഉടമസ്ഥതയും നിയന്ത്രണവും ഗവൺമെന്റിനോ, ഒരു വലിയ സ്ഥാപനത്തിനോ ആയിരിക്കും (ചിത്രം 8.29)



ചിത്രം 8.29 : മെട്രോപൊളിറ്റൻ ഏരിയ നെറ്റ് വർക്ക്

11.7.4 വൈഡ് ഏരിയ നെറ്റ്‌വർക്ക് (Wide Area Network (WAN))

പല നഗരങ്ങളിലും രാജ്യങ്ങളിലും ഭൂഖണ്ഡങ്ങളിലുമായി വ്യാപിച്ചു കിടക്കുന്ന വിവര വിനിമയ കമ്പ്യൂട്ടിങ് ഉപകരണങ്ങൾ WAN ശൃംഖലയിൽ ഉൾപ്പെടുന്നു. നൂറു കിലോ മീറ്റർ വൃത്തപരിധിയ്ക്കും അപ്പുറത്തേയ്ക്ക് ഇവയുടെ പ്രവർത്തനം വ്യാപിച്ചിരിക്കുന്നു. സ്വകാര്യ കമ്പ്യൂട്ടറുകൾ, LAN, MAN കൂടാതെ മറ്റു WANകളും ഇതിൽ അംഗങ്ങൾ ആയിരിക്കും. എല്ലാ തരത്തിലും ഉള്ള വിനിമയ മാധ്യമങ്ങൾ (ഗൈഡഡും അൺ ഗൈഡഡും) ഇവിടെ ഉപയോഗിക്കുന്നു ചിത്രം 8.30. WANന് ഉത്തമ ഉദാഹരണമാണ് ഇന്റർനെറ്റ്. ലോകത്തിലെ ഏറ്റവും വലിയ WAN ആയിട്ടാണ് ഇന്റർനെറ്റിനെ കണക്കാക്കുന്നത്. രാജ്യത്തിനുള്ളിലും, വിവിധ ഭൂഖണ്ഡങ്ങളിലുമായി വ്യാപിച്ചു കിടക്കുന്ന ATM ശൃംഖല, ബാങ്ക് ശൃംഖല, ഗവൺമെന്റിന്റെയും, അന്താരാഷ്ട്ര സ്ഥാപനങ്ങളുടെയും ശൃംഖലകൾ എന്നിവ WANനു ഉദാഹരണങ്ങളാണ്.



ചിത്രം 11.30: വൈഡ് ഏരിയ നെറ്റ്‌വർക്ക്

അളവുകോൽ	PAN	LAN	MAN	WAN
വ്യാപ്തി	ചെറിയ വിസ്തീർണത്തിൽ (10m വൃത്ത പരിധി)	ഏതാനും മീറ്റർ മുതൽ കിലോമീറ്റർ വരെ (10 km വൃത്ത പരിധി)	നഗര പരിധിയിൽ (100 km വൃത്ത പരിധി)	രാജ്യങ്ങളിലും ഭൂഖണ്ഡങ്ങളിലും ലോകമാകമാനവും
വിനിമയ വേഗത	അതിവേഗം	അതിവേഗം	സാമാന്യ വേഗത	വേഗത കുറവ്
സ്ഥാപിക്കുവാനുള്ള ചിലവ്	തീരെ കുറവ്	ചിലവ് കുറവ്	സാമാന്യം ചിലവ്	ചിലവേറിയ

പട്ടിക 8.1 PAN, LAN, MAN, WAN സവിശേഷതകളുടെ സംഗ്രഹം

11.8 ശൃംഖലയുടെ യുക്ത്യാധിഷ്ഠിത തരംതിരിവ് (Logical classification of networks)

ശൃംഖലയിലെ കമ്പ്യൂട്ടറുകളുടെ ചുമതലകളെ അടിസ്ഥാനമാക്കി രണ്ടായി തരംതിരിക്കാം. പീർ ടു പീർ (Peer - to - peer), ക്ലയന്റ് സെർവർ (Client - Server).

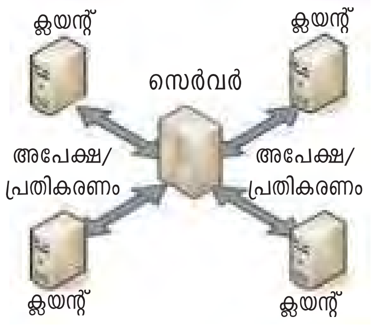
11.8.1 പീർ ടു പീർ (Peer to peer)

പിർ ടു പിർ ശൃംഖലയിൽ ഒരു കമ്പ്യൂട്ടറിനും ശൃംഖലയുടെ മുഴുവൻ ചുമതല ഉണ്ടായിരിക്കില്ല. ഇവിടെ വിവരങ്ങൾ കൈമാറുന്നതിനും ഉപകരണങ്ങൾ പങ്കിടുന്നതിനും കമ്പ്യൂട്ടറുകളെ തമ്മിൽ പരസ്പരം ബന്ധിപ്പിക്കുകയാണ് ചെയ്യുന്നത്. എല്ലാ കമ്പ്യൂട്ടറുകൾക്കും തുല്യ പരിഗണനയാണ് ഉള്ളത്. ഏതു കമ്പ്യൂട്ടറിനും ഏതു സമയത്തും ക്ലയന്റ് ആയിട്ടും സെർവർ ആയിട്ടും പ്രവർത്തിക്കാം.

ചെറിയ കമ്പ്യൂട്ടർ ശൃംഖലകൾ ആവശ്യമുള്ളതും, എന്നാൽ പൂർണ്ണ ചുമതല ഉള്ള സെർവറുകളുടെ ആവശ്യമില്ലാത്തതുമായ സ്ഥലങ്ങളിൽ (വീടുകൾ, ചെറിയ വ്യാപാര സ്ഥാപനങ്ങൾ) ഇവ അനുയോജ്യമാണ്.

11.8.2 ക്ലയന്റ് സെർവർ (Client-Server)

ഭൂരിഭാഗം ശൃംഖലകളും ക്ലയന്റ്-സെർവർ രീതിയിൽ അധിഷ്ഠിതമാണ്. ഒരു ഭക്ഷണശാലയിൽ ചെന്ന്, ആഹാര സാധനങ്ങളുടെ പട്ടിക നോക്കി, അതിൽ നിന്ന് ഇഷ്ടമുള്ളത് കടയിലെ ജോലിക്കാരനോട് (സെർവർ) ആവശ്യപ്പെടുന്നതിന് തുല്യമാണ് ഇവയുടെ പ്രവർത്തനം. ഭക്ഷണശാലയിൽ അത് ലഭ്യമാണെങ്കിൽ ആവശ്യക്കാരന് (ക്ലൈന്റ്) അത് വിതരണം ചെയ്യുകയും, ലഭ്യമല്ലെങ്കിൽ ആവശ്യം നിരാകരിക്കപ്പെടുകയും ചെയ്യുന്നു.



ചിത്രം. 8.31 : ക്ലയന്റ് - സർവർ

ക്ലയന്റ്-സെർവറിന്റെ ഘടനയിൽ ശൃംഖലയിലെ ശക്തി കൂടിയ കമ്പ്യൂട്ടർ (സെർവർ), ശക്തി കുറഞ്ഞ കമ്പ്യൂട്ടറിനു (ക്ലയന്റ്) സേവനങ്ങൾ ലഭ്യമാക്കുന്നു. ക്ലയന്റിന്റെ അഭ്യർത്ഥന അനുസരിച്ചു ഒരു സെർവർ നിർദിഷ്ട സേവനങ്ങൾ (Response) ലഭ്യമാക്കുന്നു. ഈ സേവനങ്ങളിൽ ഡാറ്റയുടെയും സോഫ്റ്റ്‌വെയറിന്റെയും ഹാർഡ്‌വെയറിന്റെയും പങ്കിടൽ ഉൾപ്പെടുന്നു. ചിത്രം 8.31 ക്ലയന്റ് സെർവറിന്റെ ഘടന ചിത്രീകരിച്ചിരിക്കുന്നു.

ക്ലയന്റ് സെർവറിന്റെ ഘടന കേന്ദ്രീകൃത സോഫ്റ്റ്‌വെയർ മാനേജ്മെന്റിന് ഉദാഹരണമാണ്. സെർവറിൽ സോഫ്റ്റ്‌വയർ ലോഡ് ചെയ്യുമ്പോൾ അവ ക്ലയന്റുകൾക്കിടയിൽ പങ്കുവെയ്ക്കപ്പെടുകയും, സെർവർ സോഫ്റ്റ്‌വയറിൽ ഉണ്ടാകുന്ന ഏതു മാറ്റവും ക്ലയന്റിൽ പ്രതിഫലിക്കുകയും ചെയ്യുന്നു. ഓരോ കമ്പ്യൂട്ടറിലും പുതിയ ഫയലും അതിന്റെ പരിവർത്തന ഫയലും ഇടുവാനുള്ള അധിക ഊർജവും സമയവും ഇതിനാൽ ലാഭിക്കാം.

സെർവറുകളുടെ തരംതിരിക്കൽ

a) **ഫയൽ സെർവർ:** ഒന്നിലധികം ഉപഭോക്താക്കളുടെ ഫയലുകൾ സൂക്ഷിക്കാനും കൈകാര്യം ചെയ്യുവാനും ഉള്ള കമ്പ്യൂട്ടർ ആണിത്.

b) **വെബ് സെർവർ :** വെബ് പേജിനുള്ള അഭ്യർഥന കൈകാര്യം ചെയ്യുന്ന കമ്പ്യൂട്ടറാണിത്.

c) **പ്രിന്റ് സെർവർ :** ക്ലയന്റുകളിൽ നിന്നും പ്രിന്ററുകളിലേക്കുള്ള പ്രിന്റിങ് ജോലികളെ മുൻഗണനയ്ക്ക് അനുസരിച്ചു പൂർത്തീകരിക്കുന്ന കമ്പ്യൂട്ടർ ആണിത്.

d) **ഡാറ്റാബേസ് സെർവർ:** പൊതുവായി സൂക്ഷിച്ചിരിക്കുന്ന ഡാറ്റായെ കാണാനും മാറ്റങ്ങൾ വരുത്താനും നീക്കം ചെയ്യുവാനും അംഗീകൃത ഉപഭോക്താവിനെ (ക്ലയന്റ്) സഹായിക്കുന്ന കമ്പ്യൂട്ടർ ആണിത്.

സ്വയം പരിശോധിക്കാം

1. ബസ് ടോപ്പോളജിയിൽ ബസിന്റെ അഗ്രഭാഗത്തു എത്തുന്ന തരംഗങ്ങളെ ________ ആഗിരണം ചെയ്യുകയും ബസിൽ നിന്ന് നീക്കം ചെയ്യുകയും ചെയ്യുന്നു.
2. ________ ടോപ്പോളജിയിൽ ഓരോ നോഡും ഹബ് / സ്വിച്ച് ലേക്ക് നേരിട്ട് ബന്ധിപ്പിച്ചിരിക്കുന്നു
3. ________ടോപ്പോളജിയിൽ ഓരോ നോഡും മറ്റു നോഡുകളുമായി നേരിട്ടു ബന്ധിപ്പിച്ചിരിക്കുന്നു.
4. താഴെപ്പറയുന്ന വിവിധ ശൃംഖലകളെ തരം തിരിക്കുക.

   ATM ന്റെ ശൃംഖല, കേബിൾ ടെലിവിഷൻ ശൃംഖല, ഒരു സ്കൂളിനുള്ളിലെ ശൃംഖല,

   ബ്ലൂടൂത്ത് ഉപയോഗിച്ചുള്ള വീടിനുള്ളിലെ ശൃംഖല, ടെലിഫോൺ ശൃംഖല, റെയിൽവേ ശൃംഖല

5 എന്താണ് PAN?

6 എന്താണ് പീർ ടു പീർ ശൃംഖല ?

11.9 ശൃംഖലയിൽ കമ്പ്യൂട്ടറുകളുടെ തിരിച്ചറിയൽ (Identification of computers over a network)

അമേരിക്കയിൽ ഉള്ള ഒരു കൂട്ടുകാരന് ഇന്ത്യയിൽ ഉള്ള നിങ്ങൾ ഒരു കത്ത് എഴുതുന്നു എന്ന് സങ്കൽപ്പിക്കുക. നിങ്ങൾ ഒരു കത്ത് എഴുതി, കവറിൽ ഇട്ടു, കവറിനു പുറത്ത് കൂട്ടുകാരന്റെ മേൽവിലാസവും എഴുതി, പുറകിൽ നിങ്ങളുടെയും മേൽവിലാസവും എഴുതുന്നു. ഈ കത്ത് ഇന്ത്യയിലെ പോസ്റ്റോഫീസിൽ ഇടുമ്പോൾ അതിനു മുകളിൽ ഇന്ത്യൻ തപാൽ വകുപ്പിന്റെ സീലും തീയതിയും അതിൽ പതിപ്പിക്കുന്നു. വിവിധ മാർഗങ്ങളിലൂടെ സഞ്ചരിച്ച കത്ത് അമേരിക്കയിലെ തപാൽ വകുപ്പിൽ എത്തുന്നു. അവിടെ വെച്ച് അമേരിക്കൻ തപാൽ വകുപ്പിന്റെ സീലും തീയതിയും പതിക്കുന്നു. അവസാനം പോസ്റ്റുമാൻ കത്ത് മേൽ വിലാസക്കാരന് കൈമാറുന്നു. കമ്പ്യൂട്ടർ ശൃംഖലയിലും ഡാറ്റയെ പാക്കറ്റുകളാക്കി ഇതേ രീതിയിൽ ആണ് കൈമാറ്റം ചെയ്യുന്നത്.

ഒരു ശൃംഖല സജ്ജീകരിച്ചു കഴിഞ്ഞാൽ, നോഡുകൾ തമ്മിൽ പരസ്പരം വിവര വിനിമയം നടത്താം. ശരിയായ വിവരവിനിമയത്തിന് നോഡുകളെ അന്യോന്യം

തിരിച്ചറിയേണ്ടത് ആവശ്യമാണ്. X എന്ന നോഡ് Y എന്ന നോഡിലേക്കു വിവരങ്ങൾ കൈമാറണമെങ്കിൽ, X ഉം Y ഉം ശൃംഖലയിൽ അന്യോന്യം തനതായി തിരിച്ചറിയത്തക്ക ആയിരിക്കണം. ഇത് എങ്ങനെ സാധിക്കുന്നു എന്ന് പരിശോധിക്കാം.

11.9.1 മീഡിയ അക്സസ് കൺട്രോൾ വിലാസം (Media Access Control (MAC) address)

ഓരോ NIC (Network Interface Card) യിലും അത് നിർമിച്ച കമ്പനിക്കാർ നൽകുന്ന വ്യത്യസ്തവും സ്ഥിരമായതും ആഗോളപരമായി അംഗീകരിച്ചിട്ടുള്ളതുമായ (പന്ത്രണ്ടക്ക ഹെക്സാ ഡെസിമൽ നമ്പറുകൾ) മേൽവിലാസമാണ് MAC അഡ്രസ്. ഒരു NIC ഉള്ള മെഷീനെ അതിന്റെ MAC വിലാസം ഉപയോഗിച്ച് തിരിച്ചറിയുന്നു. NIC യിലെ MAC വിലാസം സ്ഥിരമായിരിക്കും.

MAC വിലാസം എന്നത് 12 അക്ക ഹെക്സ ഡെസിമൽ അല്ലെങ്കിൽ 48 ബിറ്റ് ബൈനറിയാണ്. താഴെ കാണിച്ചിരിക്കുന്ന ഏതെങ്കിലും ഒരു രീതിയിൽ ആണ് MAC വിലാസം എഴുതാറുള്ളത്

MM:MM:MM:SS:SS:SS അല്ലെങ്കിൽ MM-MM-MM-SS-SS-SS

MAC വിലാസത്തിന്റെ ആദ്യഭാഗം (MM:MM:MM) അത് നിർമിച്ച കമ്പനിയുടെ തിരിച്ചറിയൽ അക്കവും രണ്ടാമത്തെ പകുതി (SS:SS:SS) NIC യ്ക്ക് ആയി കമ്പനി നൽകിയിരിക്കുന്ന ക്രമ നമ്പറുമാണ്. MAC വിലാസത്തിനു ഉദാഹരണമാണ്. താഴെ കൊടുത്തിരിക്കുന്നത്.

00:AO:C9 : 14:C8:35

ചിത്രം. 8.32 : MAC Id

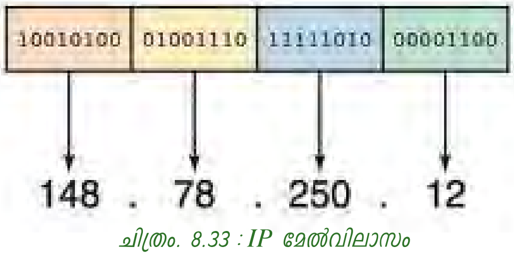
ആദ്യ പകുതി 00:AO:C9 എന്നത് ഇത് നിർമിച്ചത് ഇന്റൽ കോർപറേഷൻ ആണ് എന്ന് സൂചിപ്പിക്കുന്നു. അവസാന മൂന്നക്ക നമ്പർ ഇന്റൽ കോർപറേഷൻ NIC യ്ക്ക് നൽകിയ ക്രമനമ്പറാണ്.

11.9.2 ഇന്റർനെറ്റ് പ്രോട്ടോകോൾ (Internet Protocol (IP))

ശൃംഖലയിലെ ഓരോ നോഡിനും നൽകിയിട്ടുള്ള 4 ഭാഗങ്ങൾ ഉള്ള തനതായ നമ്പറാണ് IP മേൽവിലാസം അഥവാ IP അഡ്രസ്. ശൃംഖല മേധാവി (നെറ്റ്‌വർക്ക് അഡ്മിനിസ്ട്രേറ്റർ) അല്ലെങ്കിൽ ഇന്റർനെറ്റ് സർവീസ് പ്രൊവൈഡർ ആണ് ഓരോ നോഡിനുമുള്ള IP അഡ്രസ് രേഖപ്പെടുത്തുന്നത്. 4 ഭാഗങ്ങളാണ് ഇതിനുള്ളത്. ഓരോ ഭാഗത്തെയും ഡോട്ട് ഉപയോഗിച്ച് വേർതിരിക്കും. ഓരോ ഭാഗത്തും 0 മുതൽ 255 വരെ ഉള്ള ഒരു നമ്പറാണ് ഉണ്ടാകുക. ഒരു IP അഡ്രസ് 4 ബൈറ്റ് (32 ബിറ്റുകൾ) നമ്പർ ഉപയോഗിച്ചാണ് തയാറാക്കുന്നത്.

ഓർത്തിരിക്കുവാൻ എളുപ്പത്തിനായി IP അഡ്രസിനെ ഡെസിമൽ രൂപത്തിൽ ഡോട്ട് ഉപയോഗിച്ച് വേർതിരിച്ച നമ്പറായി, രൂപകൽപ്പന ചെയ്തിരിക്കുന്ന (ചിത്രം 8.32) ൽ നൽകിയിരിക്കുന്നു.

ഒരു ശൃംഖലയിൽ ഒരു ഉപകരണത്തിന്റെ IP മേൽവിലാസം, അതിനെ തിരിച്ചറിയുവാനായി ഉപയോഗിക്കുന്നു. ഉപകരണത്തിന്റെ IP മേൽവിലാസം ഉപയോഗിച്ച് IP പ്രോട്ടോക്കോൾ പാക്കറ്റുകളെ വഴിതിരിച്ചു വിടുന്നു.



ചിത്രം. 8.33 : IP മേൽവിലാസം

IP മേൽവിലാസത്തിനു രണ്ടു പതിപ്പുകൾ ആണ് ഉള്ളത്. പതിപ്പ് 4 (version 4) IPv4 പതിപ്പ് 6 (Version 6) IPv6. IPv4 പ്രകാരം 32 ബിറ്റ് വലുപ്പമുള്ള മേൽവിലാസം ആണ് കംപ്യൂട്ടറിനു നൽകുന്നത്, IPv6 പ്രകാരം 128 ബിറ്റ് വലുപ്പമുള്ള മേൽവിലാസം ആണ് കംപ്യൂട്ടറിനു നൽകുന്നത്. IPv4 ഉപയോഗിച്ച് 2^{32}(ഏകദേശം 4 ലക്ഷം കോടി) വ്യത്യസ്ത ഉപകരണങ്ങളെ പ്രതിനിധീകരിക്കുവാൻ കഴിയും.

ശൃംഖലയിലേക്കു ബന്ധിപ്പിക്കേണ്ട ഉപകരണങ്ങളുടെ (മൊബൈൽ ഫോൺ, വീട്ടുപകരണങ്ങൾ, വ്യക്തിഗത വിനിമയോപാധികൾ) എണ്ണം നാൾക്കുനാൾ അതിവേഗം വർധിച്ചു വരുന്നതിനാൽ IPv4 വിഭാഗത്തിലുള്ള വിലാസങ്ങൾ ഉപയോഗിച്ച് തീരുന്നു. ഈ പ്രതിസന്ധി മറികടക്കുന്നതിനായാണ് IPv6 വികസിപ്പിച്ച് എടുത്തത്. അത് ഇപ്പോൾ ഉപയോഗിച്ച് തുടങ്ങിയിരിക്കുന്നു.

IPv6 ഉപയോഗിച്ച് 2^{128}(ഏകദേശം 4 ലക്ഷം കോടി × 4 ലക്ഷം കോടി 4 ലക്ഷം കോടി ×4 ലക്ഷം കോടി) വിവിധതരം ഉപകരണങ്ങളെ പ്രതിനിധീകരിക്കാം.

നിങ്ങളുടെ സ്കൂളിലെ കമ്പ്യൂട്ടർ ശൃംഖലയിലെ ഓരോ ഉപകരണങ്ങളുടെയും MAC ID യും IP അഡ്രസ്സും കണ്ടുപിടിച്ചു ഒരു പട്ടിക താഴെ കാണിച്ചിരിക്കുന്നത് പോലെ തയാറാക്കുക (IPCONFIG/ALL എന്ന നിർദേശം, കമാൻഡ് പ്രോംറ്റിൽ ഉപയോഗിക്കുക)

ക്രമ നം.	കമ്പ്യൂട്ടറിന്റെ പേര്	IP	MAC
1.			
2.			
3.			

11.10 ശൃംഖലകളിലെ പ്രോട്ടോക്കോളുകൾ (Network Protocols)

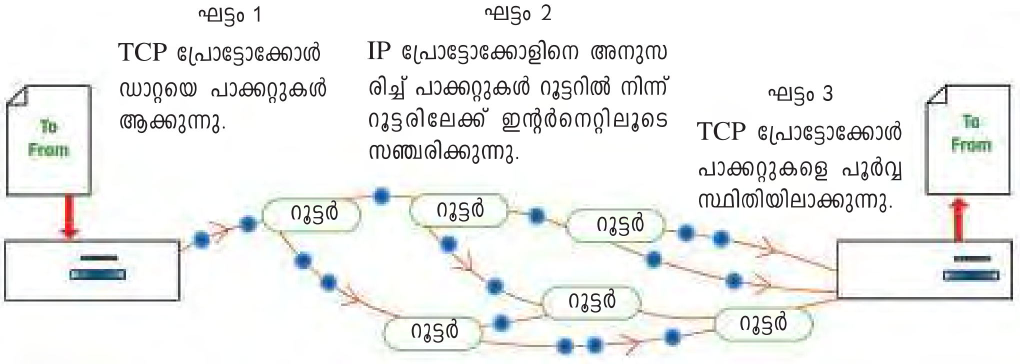
ശൃംഖലയിലെ ഉപകരണങ്ങൾ തമ്മിൽ വിവരങ്ങൾ പരസ്പരം കൈമാറുമ്പോൾ സ്വീകരിക്കേണ്ട പ്രത്യേക നിയമങ്ങളാണ് പ്രോട്ടോക്കോളുകൾ. ഡാറ്റ ഫോർമാറ്റിങ്, ഡാറ്റ കംപ്രസ്സിങ്, പിശകുകളുടെ പരിശോധന, തിരിച്ചറിയൽ, പരസ്പരം ബന്ധിപ്പിക്കൽ, ഡാറ്റാ ലക്ഷ്യസ്ഥാനത്തു എത്തിച്ചേർന്നു എന്ന് ഉറപ്പുവരുത്തൽ എന്നിവയ്ക്കായി ഓരോ പ്രോട്ടോക്കോളിനും അതിന്റേതായ നിയമങ്ങളുണ്ട്.

പ്രത്യേക ഉദ്ദേശ്യങ്ങൾക്കു വേണ്ടിയും, സാഹചര്യങ്ങൾക്കു വേണ്ടിയും നിരവധി കമ്പ്യൂട്ടർ ശൃംഖല പ്രോട്ടോക്കോളുകൾ നിർമ്മിച്ചിട്ടുണ്ട്. TCP/IP, SPx/IPx തുടങ്ങിയവയാണ് പൊതുവായി ഉപയോഗിക്കുന്ന ചില പ്രോട്ടോക്കോളുകൾ (Protocols).

TCP/IP

ഇന്റർനെറ്റിലും സാധാരണ ശൃംഖലകളിലും പരസ്പരം ബന്ധിപ്പിച്ചിട്ടുള്ള ഉപകരണങ്ങളിൽ ഉപയോഗിക്കുന്ന നിയമങ്ങളാണ് TCP/IP (ട്രാൻസ്മിഷൻ കൺട്രോൾ പ്രോട്ടോകോൾ/ ഇന്റർനെറ്റ് പ്രോട്ടോക്കോൾ) (TCP/IP Transmission control protocol/Internet protocol) എന്നത്. ഇന്റർനെറ്റിൽ ഇലക്ട്രോണിക് ഉപകരണങ്ങൾ (കമ്പ്യൂട്ടർ പോലുള്ള) എങ്ങനെ ബന്ധിപ്പിക്കണമെന്നും അവ തമ്മിൽ എങ്ങനെ വിവര വിനിമയം നടത്തണമെന്നും TCP/IP നിർവചിച്ചിരിക്കുന്നു

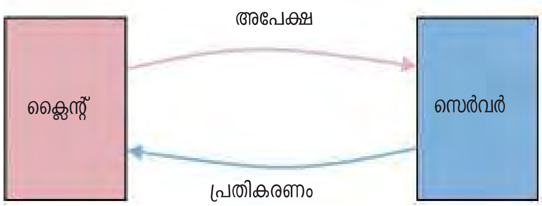
ഒരു കമ്പ്യൂട്ടറിൽ നിന്ന് മറ്റൊന്നിലേക്ക് ഡാറ്റ അയയ്ക്കുമ്പോൾ, TCP/IP ആദ്യം അവയെ വിഭജിച്ചു ചെറിയ പാക്കറ്റുകൾ ആക്കുകയും പിന്നീട് അയയ്ക്കുകയും ചെയ്യുന്നു. സ്വീകരിക്കേണ്ട കമ്പ്യൂട്ടറിൽ ഈ പാക്കറ്റുകൾ കിട്ടിക്കഴിഞ്ഞാൽ, ഈ പാക്കറ്റുകളിൽ തെറ്റുകളോ കേടുപാടുകളോ ഉണ്ടോ എന്ന് പരിശോധിക്കുന്നു. തകരാറുകൾ കണ്ടെത്തിയാൽ, ഈ പാക്കറ്റുകൾ വീണ്ടും അയയ്ക്കുന്നതിനുള്ള നിർദേശം TCP സമർപ്പിക്കുന്നു. തകരാറൊന്നും ഇല്ലെങ്കിൽ പാക്കറ്റുകളെ TCP യിൽ നിർദ്ദേശിച്ചിട്ടുള്ള നിയമങ്ങൾക്കനുസരിച്ചു സംയോജിപ്പിച്ച് യഥാർഥ സന്ദേശം ആക്കി മാറ്റുകയും ചെയ്യുന്നു. TCP/IP നിയമങ്ങളിൽ നടക്കുന്ന വിവിധ പ്രവർത്തനങ്ങൾ ചിത്രം 8.32 കൊടുത്തിരിക്കുന്നു. ഈ പാക്കറ്റുകൾ ലക്ഷ്യ സ്ഥാനത്തു എത്തിച്ചേരുന്നു എന്ന് ഉറപ്പാക്കുന്നത് ഇന്റർനെറ്റ് പ്രോട്ടോക്കോൾ ആണ്. ഒരേ സന്ദേശത്തിന്റെ വിവിധ പാക്കറ്റുകൾ പല പാതകളിലൂടെയാണ് സഞ്ചരിക്കുന്നതെങ്കിലും അവ ഒരേ ലക്ഷ്യസ്ഥാനത്തു എത്തിച്ചേരുകയും അവയെ അവിടെ വെച്ച് സംയോജിപ്പിക്കുകയും ചെയ്യുന്നു. HTTP, FTP, DNS തുടങ്ങിയ പ്രോട്ടോക്കോളുകളും TCP/IP പ്രോട്ടോക്കോളിനുണ്ട്.



ചിത്രം 8.34 : TCP/IP *യുടെ പ്രവർത്തനം*

a. HTTP

ഹൈപ്പർ ടെക്സ്റ്റ് ട്രാൻസ്ഫർ പ്രോട്ടോകോൾ (Hyper Text Transfer Protocol) എന്നാണ് HTTP യുടെ പൂർണ രൂപം. ക്ലയന്റിൽ നിന്നുള്ള അഭ്യർഥന കൈമാറ്റം ചെയ്യുവാനും, സെർവറിൽ നിന്ന് പ്രതികരണങ്ങൾ സ്വീകരിക്കുവാനുമുള്ള അംഗീകൃത പെരുമാറ്റ ചട്ടങ്ങളാണിത്. ക്ലയന്റിൽ നിന്ന് ബ്രൗസർ വഴി അപേക്ഷ സ്വീകരിക്കുന്ന സർവർ, HTTPവഴി സേവനം നൽകുകയും ചെയ്യുന്നു. ഇത്തരം അഭ്യർഥനയുടെയും പ്രതികരണത്തിന്റെയും ജോഡികൾ HTTP സെഷൻ എന്ന് അറിയപ്പെടുന്നു. (ചിത്രം 8.35)



*ചിത്രം . 8.35 : **HTTP** സെഷൻ*

ക്ലയന്റിൽ നിന്നുള്ള നിർദ്ദേശത്തെ തുടർന്ന് സെർവർ പ്രതികരിക്കുന്നത് രണ്ടു രീതിയിലാണ്. സെർവറിൽ മുൻകൂട്ടി സൂക്ഷിച്ചിട്ടുള്ള ഫയൽ അയച്ചു കൊടുത്തോ (Static രീതി) സെർവറിൽ സൂക്ഷിച്ചിട്ടുള്ള പ്രോഗ്രാം കോഡിന്റെ പ്രവർത്തന ഫലമായിട്ടുള്ള ഫയൽ അയച്ചു കൊടുത്തോ (Dynamic രീതി) ആകാം അത്.

HTTP യുടെ രണ്ടു പ്രധാന സവിശേഷതകൾ

- HTTP യിൽ വിവര വിനിമയ മാധ്യമത്തിന്റെ സ്വാധീനമില്ല.
- HTTP അസ്ഥിരമാണ് (അഭ്യർഥനയുടെയും പ്രതികരണത്തിന്റെയും സമയത്തു മാത്രം) ക്ലയന്റ് സർവർ ബന്ധം പരസ്പരം നിലനിർത്തുകയും അതിനുശേഷം ബന്ധം നിശേഷം വിച്ഛേദിക്കുകയും ചെയ്യുന്നു.

b. FTP

എഫ് ടി പി യുടെ പൂർണരൂപം ഫയൽ ട്രാൻസ്ഫർ പ്രോട്ടോകോൾ (File Transfer Protocol) എന്നാണ്. ഡാറ്റയും പ്രോഗ്രാം ഫയലുകളും ശൃംഖല വഴി പരസ്പരം കൈമാറ്റം ചെയ്യുവാൻ ഉപയോഗിക്കുന്ന അടിസ്ഥാന പ്രോട്ടോകോൾ ആണിത്. ഇന്റർനെറ്റിലൂടെ ലളിതമായ രീതിയിൽ കമ്പ്യൂട്ടറുകൾ തമ്മിൽ ഫയലുകൾ കൈമാറാനുള്ള മാർഗമാണ് ഇത്. TCP യും IP യും ഉപയോഗിച്ച് അയയ്ക്കുകയും സ്വീകരിക്കുകയും ചെയ്യുന്നു.

സെർവറിലെ സുരക്ഷാ മാർഗങ്ങൾ ആയ യൂസർ നാമവും പാസ്സ് വേർഡും ഉപയോഗിച്ച് ഫയലുകൾ സുരക്ഷിതമായി കൈമാറ്റം ചെയ്യുന്നത് ക്ലയന്റ് സെർവർ ഘടനയായ FTP ഉപയോഗിച്ചാണ്. FTP ക്ലയന്റ് പ്രോഗ്രാമുകളായ FileZ-illa, CUTEFTP എന്നിവ ഉപയോഗിച്ച് ഫയലുകൾ വളരെ എളുപ്പത്തിൽ അയയ്ക്കുവാനും സ്വീകരിക്കുവാനും കഴിയുന്നു.

c. DNS

ഡൊമൈൻ നെയിം സിസ്റ്റം (Domain Name System) എന്നാണ് DNS ന്റെ പൂർണ രൂപം. വെബ് ബ്രൗസറിന്റെ അഡ്രെസ്സ് ബാറിൽ നമ്മൾ ടൈപ്പ് ചെയ്യുന്ന വെബ് മേൽവിലാസത്തിന്റെ (ഡൊമൈൻ നാമം) IP മേൽവിലാസം DNS നമുക്ക് നൽകുന്നു.

(മൊബൈൽ ഫോണിൽ ഒരു പേര് തിരഞ്ഞെടുക്കുമ്പോൾ അതിൽ ഫോൺ നമ്പർ ഉള്ളത് പോലെ)

DNS നു അതിന്റേതായ ശൃംഖലകൾ ഉണ്ട്. ഇന്റർനെറ്റിൽ ഉള്ള എല്ലാ വെബ്സൈറ്റുകളുടെയും IP മേൽവിലാസങ്ങളും ഡൊമൈയിൻ നാമങ്ങളും ഒരു ഡാറ്റാബേസിൽ ശേഖരിച്ചിട്ടുണ്ട്. ഇന്റർനെറ്റിലെ ഓരോ നോഡിന്റെയും IP മേൽവിലാസം സ്ഥിരമാണ് എന്നതാണ് DNS ന്റെ അടിസ്ഥാനം. ഒരു DNS നു ഒരു ഡൊമൈൻ നാമത്തിനെ വിവർത്തനം ചെയ്തു IP മേൽവിലാസമാക്കുവാൻ കഴിഞ്ഞില്ലെങ്കിൽ അത് അടുത്ത DNS നോടും, അതിനും കഴിഞ്ഞില്ലെങ്കിൽ അതിനടുത്തതിനോടും വിവരവിനിമയം നടത്തും. ഈ പ്രക്രിയ ശരിയായ IP മേൽവിലാസം കിട്ടുന്നത് വരെ തുടരുന്നു.

നമുക്ക് ചെയ്യാം

TCP/IP, HTTP, FTP, DNS എന്നിവയല്ലാതെ ഏതെങ്കിലും അഞ്ചു പ്രോട്ടോക്കോളുകളെക്കുറിച്ച് കുറിപ്പ് തയാറാക്കുക.

11.11 യൂണിഫോം റിസോഴ്സ് ലൊക്കേറ്റർ (Uniform Resource Location (URL))

യൂണിഫോം റിസോഴ്സ് ലൊക്കേറ്റർ എന്നതാണ് URL- ന്റെ പൂർണ രൂപം. URL എന്നത് ക്രമീകരിച്ച വാക്കുകൾ ഉപയോഗിച്ച് വെബ് ബ്രൗസറുകൾ, ഇമെയിൽ പ്രോഗ്രാമുകൾ, മറ്റു സോഫ്റ്റ്വെയറുകൾ തുടങ്ങിയവയെ ഇന്റർനെറ്റിൽ തിരിച്ചറിയുവാൻ സഹായിക്കുന്ന ഒന്നാണ്. ഇന്റെർനെറ്റിലുള്ള എല്ലാ വിഭവങ്ങൾക്കും (resources) തനതായ URL ഉണ്ടായിരിക്കും. ഫയലുകൾ, അതുൾപ്പെടുന്ന വെബ്പേജുകൾ മറ്റു ഡോക്യൂമെന്റുകൾ, ഗ്രാഫിക്സ്, പ്രോഗ്രാമുകൾ തുടങ്ങിയവയാണ് ശൃംഖല വിഭവങ്ങൾ (Network resources). ഒരു URL-ൽ അക്ഷരങ്ങൾ, അക്കങ്ങൾ, ചിഹ്നങ്ങളും ഉണ്ട്.

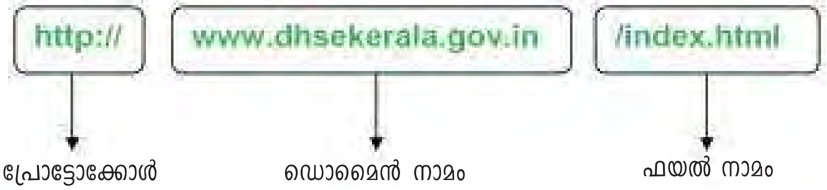
ഒരു URL മേൽവിലാസത്തെ മൂന്നായി തരം തിരിച്ചിരിക്കുന്നു

a) നെറ്റ്വർക്ക് പ്രോട്ടോക്കോൾ

b) ഡൊമൈൻ നാമം (ഹോസ്റ്റിന്റെ പേര് അല്ലങ്കിൽ വിലാസം)

c) ഫയൽ നാമം

ഉദാഹരണത്തിന് http://www.dhsekerala.gov.in/index.html എന്ന URL ന് മൂന്നു ഭാഗങ്ങൾ ഉണ്ട്. ചിത്രം 8.36 ഈ URL ന്റെ വിവിധ ഘടകങ്ങൾ കാണിച്ചിരിക്കുന്നു.



ചിത്രം 8.36 : URL ന്റെ ഘടകങ്ങൾ

മൂന്നു വിഭാഗങ്ങളുടെയും വിശദ വിവരങ്ങൾ ചുവടെ ചേർത്തിരിക്കുന്നു.

a) പ്രോട്ടോക്കോൾ (Protocol)

ഡൊമൈനിൽ നിന്ന് വിവരങ്ങൾ ഏതു പ്രോട്ടോകോൾ ഉപയോഗിച്ചാണ് സ്വീകരിക്കേണ്ടത് എന്നത് ബ്രൗസറിനെ അറിയിക്കുന്നു.

b) ഡൊമൈൻ നാമം (Domain name)

ഡൊമൈൻ നാമം എന്നത് ഡൊമൈൻ നെയിം സിസ്റ്റം വഴി സെർവറിനു നൽകിയ പേരാണ്. ഒരു URL ലെ ഡൊമൈൻ നാമം ഒരു വെബ് സെർവറിനെ കണ്ടെത്തുവാൻ സഹായിക്കുന്നു. വളരെ എളുപ്പത്തിൽ ഓർമ്മ നിൽക്കുന്ന വിധത്തിൽ ഇന്റർനെറ്റ് ഉപഭോക്താക്കൾക്ക് ഹ്രസ്വനാമത്തിൽ കിട്ടുന്നു. കമ്പ്യൂട്ടർ ഉപയോഗിച്ച് ഇന്റർനെറ്റിലൂടെ വിവരവിനിമയം നടത്താൻ, IP അഡ്രസ്സ് ഉപയോഗിക്കാവുന്നതാണ്. എന്നാൽ എല്ലാ കമ്പ്യൂട്ടറുകളുടെയും IP അഡ്രസ് ഓർത്തിരിക്കുക എന്നത് പ്രായോഗികമല്ല. അതു കൊണ്ട് വെബ് സർവറിനു പേര് നൽകുകയും, ഈ പേരിനു തുല്യമായ IP വിലാസങ്ങളുടെ ഒരു പട്ടിക ഉണ്ടാക്കി സൂക്ഷിക്കുക എന്ന സമ്പ്രദായം കൊണ്ട് വന്നു. ഇതിനെയാണ് ഡൊമൈൻ നാമം എന്ന് പറയുന്നത്. ഉദാഹരണം dhsekerala.gov.in, keralaresults.nic.in ,google.com, gmail.com.

ഒരു ഡൊമൈൻ നാമത്തിനു സാധാരണ ഒന്നിൽ കൂടുതൽ ഭാഗങ്ങൾ ഉണ്ട്. ടോപ് ലെവൽ ഡൊമൈൻ അഥവാ പ്രാഥമിക ഡൊമൈൻ, ഉപ ഡൊമൈൻ എന്നിവ.

മുകളിൽ സൂചിപ്പിച്ച ഉദാഹരണത്തിൽ in എന്നത് പ്രാഥമിക ഡൊമൈനും, gov എന്നത് in ന്റെ ഉപ ഡൊമൈനും, dhsekerala എന്നത് gov യുടെ ഉപ ഡൊമൈനുമാണ്.

വളരെ കുറച്ചു പ്രാഥമിക ഡൊമൈനുകൾ ആണ് ഉള്ളത്. അവയെ രണ്ടു വിഭാഗങ്ങൾ ആയി തരം തിരിച്ചിരിക്കുന്നു. പൊതുവായ ഡൊമൈൻ നാമങ്ങൾ (Generic domain names) എന്നും രാജ്യത്തിന്റെ പ്രത്യേക ഡൊമൈൻ നാമങ്ങൾ (Country specific domain names) എന്നും. പൊതുവായ/രാജ്യ ഡൊമൈൻ നാമങ്ങളുടെ ഉദാഹരണങ്ങൾ പട്ടിക 8.2 ൽ കൊടുത്തിരിക്കുന്നു.

Generic Domain Names	
·com	Commercial business
·edu	Educational institutions
·gov	Government agencies
·mil	Military
·net	Network organizations
·org	Organizations (nonprofit)

Country Specific Domain Names	
·in	India
·au	Australia
·ca	Canada
·ch	China
.jp	Japan
.us	United States of America

പട്ടിക 8.2 : പൊതുവായതും രാജ്യത്തിന്റെ പ്രത്യേക ഡൊമൈൻ നാമങ്ങളും

c.ഫയൽ നാമം (File Name)

ഏതു ഫയൽ ആണോ തുറക്കേണ്ടത് അതിനെ സൂചിപ്പിക്കുന്നതാണ് ഈ ഭാഗം. ചിത്രം 8.35ലെ ഉദാഹരണത്തിൽ കൊടുത്തിരിക്കുന്ന ഡൊമൈൻ നാമം നൽകുമ്പോൾ വെബ് സർവർ index.html എന്ന ഫയലാണ് അയച്ചു തരിക.

നമുക്ക് ചെയ്യാം

പൊതുവായ ഡൊമൈനും രാജ്യത്തിന്റെ ഡൊമൈൻ നാമവും ഉൾകൊള്ളുന്ന URL ന്റെ സാധുവായ ഉദാഹരണ പട്ടിക തയ്യാറാക്കുക. തുറന്ന് വന്ന ഫയലിന്റെ പേര് എന്താണ് എന്ന് ശ്രദ്ധിക്കുക. (തുറന്നതിനുശേഷം അഡ്രസ് ബാറിൽ കാണുന്ന പേര് ആകും ഫയലിന്റെ പേര്)

നമുക്ക് സംഗ്രഹിക്കാം

ഈ നൂറ്റാണ്ടിന്റെ അവശ്യഘടകമായ കമ്പ്യൂട്ടർ ശൃംഖലയെ കുറിച്ച് നമ്മൾ ഈ അധ്യായത്തിൽ പഠിച്ചു. ശൃംഖലയുടെ പ്രാധാന്യത്തെക്കുറിച്ചും അവ നൽകുന്ന നേട്ടങ്ങളെ കുറിച്ചും ചർച്ച ചെയ്തു. വിവിധ വിവര വിനിമയ ഭൗതിക മാധ്യമങ്ങളുടെ നിർമ്മിതിയെക്കുറിച്ചും അവയുടെ നേട്ടങ്ങളും കോട്ടങ്ങളും അവയുടെ പ്രവർത്തനങ്ങളെക്കുറിച്ചും നാം ചർച്ച ചെയ്തു. ശൃംഖല രൂപകൽപന ചെയ്യുമ്പോൾ, ഉപയോഗിക്കുന്ന വിവിധതരം ഉപകരണങ്ങളെക്കുറിച്ചും മനസ്സിലാക്കി.

വിവിധതരം ശൃംഖലയെക്കുറിച്ച് ചർച്ച ചെയ്യുന്നതിന് മുൻപ്, ടോപ്പോളജി എന്ന പദത്തിലൂടെ വിവിധ വിധത്തിലുള്ള ശൃംഖലയുടെ ക്രമീകരണളെക്കുറിച്ചു പഠിച്ചു. TCP/IP പോലുള്ള ശൃംഖല പ്രോട്ടോക്കോൾ ഉപയോഗിച്ച് വിവരങ്ങൾ കൈമാറ്റം ചെയ്യുന്നത് എങ്ങനെ എന്ന് ചർച്ച ചെയ്തു.ഒരു ശൃംഖലയിലെ നോഡിനെ കണ്ടെത്തുന്നത് എങ്ങനെ എന്ന് പഠിച്ചു. URL നെ കുറിച്ചുള്ള ചർച്ചയോടു കൂടി ഈ പാഠഭാഗം ഉപസംഹരിച്ചു.

പഠനനേട്ടങ്ങൾ

ഈ അധ്യായം പൂർത്തീകരണത്തോടെ പഠിതാവിന്

- വിവരവിനിമയ മാധ്യമത്തെ തിരഞ്ഞെടുക്കുവാനും മനസ്സിലാക്കുവാനും കഴിയുന്നു.
- വ്യത്യസ്ത ശൃംഖലകളെ താരതമ്യം ചെയ്യുന്നു.
- [redacted] യുക്താധിഷ്ഠിത തരംതരിവുകൾ തിരിച്ചറിയുന്നു.
- ശൃംഖലയിലൂടെ ഡാറ്റ അയയ്ക്കുന്നത് മനസ്സിലാക്കുന്നു.
- ലളിതമായ ഒരു ശൃംഖല നിർമിക്കുന്നു.
- ശൃംഖലയിലെ ഒരു നോഡ് തിരിച്ചറിയുന്നു.
- ഒരു URL ന്റെ വിവിധ ഭാഗങ്ങൾ തിരിച്ചറിയുന്നു.

മാതൃകാ ചോദ്യങ്ങൾ

ഹ്രസ്വോത്തര ചോദ്യങ്ങൾ

1. പ്രകാശ തരംഗങ്ങളുടെ രൂപത്തിൽ വിവരങ്ങൾ വഹിച്ചു കൊണ്ട് പോകുന്ന സംപ്രേക്ഷണ മാധ്യമമാണ് ________________.

   a) കൊയാക്സിയൽ കേബിൾ b) ട്വിസ്റ്റഡ് പെയർ
   c) വൈ-ഫൈ d) ഒപ്റ്റിക്കൽ ഫൈബർ

2. വ്യത്യസ്ത പ്രോട്ടോക്കോളുള്ള വ്യത്യസ്ത ശൃംഖലകളെ പരസ്പരം ബന്ധിപ്പിക്കുന്ന ഉപകരണമാണ് ____________.

   a) റൂട്ടർ b) ബ്രിഡ്ജ് c) സ്വിച്ച് d) ഗേറ്റ്‌വേ

3. __________ ക്രമീകരണത്തിൽ ഒരു കമ്പ്യൂട്ടറിന്റെ തകരാർ മൊത്തം ശൃംഖലയുടെ പ്രവർത്തനത്തെയും ബാധിക്കുന്നു

   a) ബസ് b) റിങ് c) സ്റ്റാർ d) ഇവയൊന്നും ഇല്ല

4. വിവിധ ഉപകരണങ്ങളിൽ നിന്നുള്ള തരംഗങ്ങളെ ഒരൊറ്റ വിനിമയ മാധ്യമത്തിലൂടെ ഒരേ സമയത്തു കടത്തിവിടുവാൻ __________ ഉപകരണം ഉപയോഗിക്കുന്നു.

   a) മോഡം b) സ്വിച്ച് c) റൂട്ടർ d) മൾട്ടിപ്ളെക്സർ

5. സാറ്റലൈറ്റ് ലിങ്കുകൾ പൊതുവെ ഉപയോഗിക്കുന്നത്

   a) PANS b) LANS c) MANS d) ഇവയിലെല്ലാം

ലഘു ഉപന്യാസ ചോദ്യങ്ങൾ

1. ബാൻഡ് വിഡ്ത് നിർവ്വചിക്കുക.
2. ശൃംഖലകളുമായി ബന്ധപ്പെട്ട രണ്ടു ഉപകരണങ്ങൾ ആണ് സ്വിച്ചും ഹബ്ബും. ഇവയെ വേർതിരിക്കുക.
3. IP അഡ്രസ് എന്നാൽ എന്താണ്? ഒരു ഉദാഹരണം എഴുതുക.
4. എന്താണ് TCP/IP? ഇതിന്റെ പ്രാധാന്യം എന്ത്?
5. കമ്പ്യൂട്ടർ ശൃംഖലയെ നിർവ്വചിക്കുക.
6. എന്താണ് ബ്ലൂടൂത്ത്?
7. എന്താണ് മോഡം?
8. റൂട്ടറും ഗേറ്റ്‌വേയും തമ്മിലുള്ള വ്യത്യാസം എന്താണ്?
9. കമ്പ്യൂട്ടർ ശൃംഖല നിർമ്മിക്കുന്നതിന്റെ ആവശ്യകത വിശദീകരിക്കുക?
10. കമ്പ്യൂട്ടർ ശൃംഖലയുടെ ഉപയോഗങ്ങൾ എന്തൊക്കെയാണ്?
11. മൈക്രോവേവ് സംപ്രേക്ഷണത്തിന്റെ പോരായ്മകൾ എന്തൊക്കെയാണ്? എങ്ങനെ അതിനെ മറികടക്കാം?

12. വൈ-ഫൈ യുടെ സവിശേഷതകൾ എന്തൊക്കെയാണ്?

13. ഒരു അന്തർദേശീയ സ്കൂൾ 45 m ചുറ്റളവിൽ സ്ഥാപിച്ചിരിക്കുന്ന കമ്പ്യൂട്ടറുകളെ തമ്മിൽ ബന്ധിപ്പിക്കുവാൻ ആലോചിക്കുന്നു. ഇതിന് ഉതകുന്ന സാമ്പത്തിക ലാഭമുള്ളതും അതി വേഗതയുള്ളതും ആയ മാധ്യമം തെരഞ്ഞെടുക്കുക

14. എന്താണ് NIC? ശൃംഖലയിൽ അവയുടെ പ്രാധാന്യം എന്താണ്?

15. ഒരു സ്ഥാപനത്തിലെ കമ്പ്യൂട്ടർ ശൃംഖലയുടെ മേലധികാരിയാണ് നിങ്ങൾ എന്ന് സങ്കൽപ്പിക്കുക. ശൃംഖലയിലെ 10 Mbps ന്റെ Switch മാറ്റി 10 Mbps ന്റെ hub വെയ്ക്കുവാൻ നിങ്ങളോടു മേലധികാരി നിർദ്ദേശിക്കുന്നു? ഇതിനോട് നിങ്ങൾ യോജിക്കുന്നുണ്ടോ? നിങ്ങളുടെ അഭിപ്രായം സാധൂകരിക്കുക?

16. നിങ്ങളുടെ ബയോഡാറ്റ 10KM അകലെയുള്ള കൂട്ടുകാരന്റെ കമ്പ്യൂട്ടറിലേക്കു ടെലിഫോൺ ശൃംഖല വഴി കൈമാറ്റം ചെയ്യണമെങ്കിൽ

    എ) രണ്ട് ഭാഗത്തും ഉണ്ടായിരിക്കേണ്ട ഉപകരണത്തിന്റെ പേര് എഴുതുക?

    ബി) രണ്ടു കമ്പ്യൂട്ടറുകൾ തമ്മിൽ ബന്ധം സ്ഥാപിച്ചു കഴിഞ്ഞാൽ, ഈ ഉപകരണത്തിലൂടെ ഫയലുകൾ അയയ്ക്കുകയും സ്വീകരിക്കുകയും ചെയ്യുന്നത് എങ്ങനെയാണ്?

17. ഒരു കമ്പ്യൂട്ടർ ശൃംഖലയിൽ റിപ്പീറ്റർ ഉപയോഗിക്കേണ്ടി വരുന്നത് എപ്പോൾ?

18. ഇൻഫ്രാറെഡും, ബ്ലൂടൂത്ത് സംപ്രേക്ഷണവും തമ്മിൽ താരതമ്യം ചെയ്യുക?

19. ടെലിഫോൺ ശൃംഖലയുമായി കമ്പ്യൂട്ടറുകളെ ബന്ധിപ്പിക്കുവാൻ ഉപയോഗിക്കുന്ന ഉപകരണമേത്? ഇതിന്റെ പ്രവർത്തനം വിശദീകരിക്കുക?

20. LANടോപ്പോളജി വിശദീകരിക്കുക?

21. TCP/IP പ്രോട്ടോക്കോൾ ചുരുക്കി എഴുതുക?

22. എന്താണ് MAC അഡ്രസ്? MAC അഡ്രസും IP അഡ്രസും തമ്മിലുള്ള വ്യത്യാസം എന്താണ്?

ഉപന്യാസ ചോദ്യങ്ങൾ

1. കമ്പ്യൂട്ടർ ശൃംഖലകളെ അവയുടെ വലുപ്പമനുസരിച് എങ്ങനെ തരം തിരിച്ചിരിക്കുന്നു?
2. വ്യത്യസ്ത LAN ടോപ്പോളജികളെ താരതമ്യം ചെയ്യുക?
3. വിവിധ തരത്തിലുള്ള ഗൈഡഡ് വിനിമയ ചാനലുകളെ കുറിച്ച് വിശദീകരിക്കുക?
4. വ്യത്യസ്ത അൺ ഗൈഡഡ് മാധ്യമങ്ങൾ തമ്മിൽ താരതമ്യം ചെയ്യുക?
5. പ്രോട്ടോക്കോൾ എന്ന പദം നിർവ്വചിക്കുക? ഏതെങ്കിലും രണ്ടു വിനിമയ പ്രോട്ടോക്കോളുകൾ ചുരുക്കി വിശദീകരിക്കുക?
6. ശൃംഖലയിൽ ഉപയോഗിക്കുന്ന വിവിധ തരത്തിലുള്ള വിവര വിനിമയ ഉപകരണങ്ങളെ കുറിച്ച് ചുരുക്കി വിശദീകരിക്കുക?

7. താഴെ പറയുന്ന സന്ദർഭങ്ങളിൽ ഏതു തരത്തിലുള്ള വിനിമയ മാധ്യമമാണ് അനുയോജ്യമാകുക?

a. LAN സ്ഥാപിക്കുക.

b. ലാപ്ടോപ്പിൽ നിന്നും മൊബൈയിലേക്കു ഡാറ്റ കൈമാറുക.

c. ഒരു മൊബൈൽ ഫോണിൽ നിന്ന് മറ്റൊരു മൊബൈൽ ഫോണിലേക്കു ഡാറ്റ കൈമാറുക.

d. ഒന്നിൽ കൂടുതൽ ഉപകരണങ്ങൾ നിയന്ത്രിക്കുന്ന ഒരു റിമോട്ട് കൺട്രോൾ ഉണ്ടാക്കുക.

f. രണ്ടു രാജ്യത്തുള്ള രണ്ടു സ്ഥാപനങ്ങൾ തമ്മിലുള്ള അതിവേഗ വിവരവിനിമയം.

g. കുന്നിൻപ്രദേശത്തുള്ള (മലയോര മേഖലകളിൽ) വിവരവിനിമയം.

h. നഗരത്തിനുള്ളിലോ നഗര പരിധിക്കുള്ളിലോ കേബിൾ ഉപയോഗിച്ചുള്ള ചിലവേറിയ വിവരവിനിമയം.

പദാവലി

അംഗീകൃത തത്വങ്ങൾ	:	postulates
അനന്യത നിയമം	:	identtiy law
അഷ്ടസംഖ്യ	:	octal number
അസംബ്ലി ഭാഷ	:	assembly language
അസ്ഥിര ദശാംശ സംഖ്യ ലിറ്ററൽ	:	floating point numeric literal/floating point literal
അസ്ഥിര പ്രാഥമിക മെമ്മറി	:	volatile primary memory
അറെകളുടെ പ്രഖ്യാപനം	:	array declaration
അന്തർനിർമിത ഫങ്ഷനുകൾ	:	built-in functions
ആവർത്തന പ്രസ്താവനകൾ	:	iteration statements
ആധാരം	:	base
ആസ്കി	:	ASCII
ഇടം നൽകൽ	:	allocation
ഇസ്കി	:	ISCII
ഉപയോക്തൃ നിർവചിക്കുന്ന	:	user-defined
ഉപയോക്തൃ നിർവചിത ഫങ്ഷനുകൾ	:	user defined function
ഏക ഉദ്ധരണി (ഏക സൂചകം)	:	single quote
കടന്നുപോകൽ	:	traversal
കമ്പ്യൂട്ടർ ശൃംഖലകൾ	:	computer networks
ക്രമപ്പെടുത്തൽ	:	sort
ക്രമ നിയമം	:	commutative Law
ഗണിതം	:	arithmetic
ചിഹ്നവും മൂല്യവും	:	sign and magnitude
തലക്കെട്ട്	:	header
തനത്	:	default
തിരയൽ	:	searching

തീരുമാനമെടുക്കൽ പ്രസ്താവനകൾ	: decision making statement
ദശസംഖ്യാ സമ്പ്രദായം	: decimal number system
ദ്വയസംഖ്യ	: binary number
ദ്വൈത സിദ്ധാന്തം	: principle of dualtiy
ദ്വിതീയ സംഭരണം	: secondary storage
നൽകിയ ഇടം തിരികെ എടുക്കൽ	: de-allocation
നിർദ്ദേശം വ്യാഖ്യാനിക്കുക.	: command interpretation
നിയന്ത്രണ പ്രസ്താവന	: control statement
നീക്കിവെയ്പ്	: allocation
ഡാറ്റ	: data
ഡാറ്റ ഇനം	: data type
പദപ്രയോഗം	: expression
പരിശോധനാ പ്രയോഗം	: test expression
പരിവർത്തന ഫങ്ഷനുകൾ	: correction function
പരിഷ്കരിക്കൽ പ്രസ്താവന	: updation statement
പരിവർത്തനം	: conversion
പൂരകം	: complement
പൂരക നിയമം	: complementary law
പൂർണസംഖ്യ	: integer
പ്രഖ്യാപനം	: declaration
പ്രാരംഭവില നൽകൽ	: initialisation
പ്രയോഗം	: expression
പ്രസ്താവന	: statement
പ്രതിനിധാനം	: representation
പ്രാഥമിക സംഭരണം	: primary storage
പ്രൊസസ്സ് കൈകാര്യം ചെയ്യുക	: process management
പ്രസ്താവന	: statement

ഫങ്ഷൻ നാമം	:	function name
ഫയൽ നാമം	:	file name
ഫയൽ കൈകാര്യം ചെയ്യുക	:	file management
ഫ്ളോട്ടിങ് പോയിന്റ് നമ്പർ	:	floating point number
ബഹുമുഖ അറെ	:	multi dimensional array
ഭാഷ പ്രൊസസ്സർ	:	language processor
ഭൗതിക ഘടകങ്ങൾ	:	physcial component
ബീജഗണിതം	:	algebra
മൂലസംഖ്യ	:	radix
മെമ്മറി കൈകാര്യം ചെയ്യുക	:	memory management
മെമ്മറി സ്ഥാനം	:	memory location
മെമ്മറി നീക്കിവെയ്ക്കൽ	:	memory allocation
യുക്തി വിചിന്തനം (ലോജിക്കൽ റീസണിംഗ്)	:	logical reasoning
യുക്തിപരമായ നിഷേധം	:	logical negation
യന്ത്ര ഭാഷ	:	machine language
രേഖീയ തിരയൽ	:	linear search
വർഗപൂരക നിയമം	:	involution law
വർഗസമ നിയമം	:	idempotent law
വിതരണ നിയമം	:	distributive law
വേരിയബിൾ	:	variable
ശ്രേണി	:	sequence
സാമൂഹിക മാധ്യമങ്ങൾ	:	social media
സംയോജന നിയമം	:	associative law
സിദ്ധാന്തം	:	theorem
സ്ഥാന വില	:	weight
സ്ഥാനീയ സംഖ്യ	:	positional Number
സ്വയം ആവർത്തിക്കുന്ന ഫങ്ഷനുകൾ	:	recursive function

സ്വാംശീകരണ നിയമം	:	absorption Law
സ്വതന്ത്ര ഓപ്പൺ സോഴ്സ്	:	free and open source
സുസ്ഥിര ദ്വിതീയ മെമ്മറി	:	non-volatile secondary memory
1 ന്റെ പൂരകം	:	1’s Complement
2 ന്റെ പൂരകം	:	2’s Complement