3

പ്രോഗ്രാമിങ് തത്വങ്ങളും പ്രശ്നപരിഹരണവും

പ്രധാന ആശയങ്ങൾ

- **കമ്പ്യൂട്ടറുകളുടെ സഹായത്തോടെയുള്ള പ്രശ്ന പരിഹാരം**
- **പ്രശ്ന പരിഹാരത്തിനുള്ള സമീപനങ്ങൾ**
    - o ടോപ് ഡൗൺ രൂപകല്പന
    - o ബോട്ടം അപ് രൂപകല്പന
- **പ്രോഗ്രാമിങ്ങിലെ വിവിധ ഘട്ടങ്ങൾ**
    - o പ്രശ്നം തിരിച്ചറിയൽ (Problem Identification)
    - o അൽഗോരിതങ്ങളും ഫ്ളോച്ചാർട്ടുകളും തയാറാക്കൽ (Preparing algorithms and Flowcharts)
    - o പ്രോഗ്രാം കോഡിങ് (Program Coding)
    - o പരിഭാഷ (Translation)
    - o ഡീബഗ്ഗിങ് (Debugging)
    - o പ്രവർത്തനവും പരീക്ഷണവും (Execution and Testing)
    - o വിവരണം തയാറാക്കൽ (Documentation)
- **അൽഗോരിതങ്ങളുടെ പ്രകടനം വിലയിരുത്തൽ**

Z7I2F4

ഡാറ്റാ പ്രോസസ്സിങ് എന്ന ആശയവും, അതിൽ കമ്പ്യൂട്ടറുകൾക്കുള്ള പങ്കും നാം പഠിച്ചിട്ടുണ്ട്. ഹാർഡ്‌വെയർ, സോഫ്റ്റ്‌വെയർ, ഉപയോക്താക്കൾ എന്നിവ അടങ്ങുന്ന ഒരു സംവിധാനം എന്ന നിലയിലും ഈ ഘടകങ്ങളെല്ലാം മുൻ അധ്യായത്തിൽ നാം വിശദമായി ചർച്ച ചെയ്തിട്ടുണ്ട്. സോഫ്ട്‌വെയറിന്റെ നിർവചനം നമുക്ക് ഓർത്തെടുത്ത് നോക്കാം. ലളിതമായ രീതിയിൽ പറഞ്ഞാൽ, കമ്പ്യൂട്ടർ ഉപയോഗിച്ച് പ്രശ്നങ്ങൾ പരിഹരിക്കുന്നതിനുള്ള പ്രോഗ്രാമുകളുടെ ശേഖരമാണ് സോഫ്റ്റ്‌വെയർ. കമ്പ്യൂട്ടറിന് സ്വന്തമായി ഒന്നും ചെയ്യാനുള്ള കഴിവില്ലെന്ന് നമുക്കറിയാവുന്നതാണ്. ഒരു ജോലി നിർവഹിക്കണമെങ്കിൽ വ്യക്തമായ നിർദേശങ്ങൾ കമ്പ്യൂട്ടറിന് ആവശ്യമാണ്. ആയതിനാൽ ഒരു പ്രശ്ന പരിഹാരത്തിന് കമ്പ്യൂട്ടർ നിർവഹിക്കേണ്ട നിർദേശങ്ങളുടെ ക്രമം വ്യക്തമാക്കേണ്ടത് അത്യാവശ്യമാണ്. ഇത്തരത്തിൽ കമ്പ്യൂട്ടറിന് മനസ്സിലാകുന്ന ഒരു ഭാഷയിൽ എഴുതിയിരിക്കുന്ന, ക്രമത്തിലുള്ള നിർദേശങ്ങൾ 'കമ്പ്യൂട്ടർ പ്രോഗ്രാം' എന്ന പേരിൽ അറിയപ്പെടുന്നു. കമ്പ്യൂട്ടർ പ്രോഗ്രാം എഴുതുക എന്നത് ഒരു വെല്ലുവിളിയാണ്. എന്നിരുന്നാലും പ്രോഗ്രാമിങ്ങിന്റെ വിവിധ ഘട്ടങ്ങളും പ്രശ്നപരിഹാരസാങ്കേതികത്വ ആശയങ്ങളും ആർജിച്ചുകൊണ്ട് നമുക്ക് അതിനായി ശ്രമിക്കാം.

3.1 കമ്പ്യൂട്ടറുകൾ ഉപയോഗിച്ചുള്ള പ്രശ്ന പരിഹാരം (Problem solving using computers)

കമ്പ്യൂട്ടറിലേക്ക് നമ്മൾ നിർദേശങ്ങൾ നൽകിയാൽ മാത്രമെ അതിനു പ്രശ്നങ്ങൾ പരിഹരിക്കാൻ കഴിയുകയുള്ളൂ. നിർദേശങ്ങളിൽ അടങ്ങിയിരിക്കുന്ന ക്രിയകൾ മനസ്സിലായാൽ അതിനനുസരിച്ച് കമ്പ്യൂട്ടർ പ്രവർത്തിക്കും. നിർദേശം (Instruction) ഒരു ക്രിയാധിഷ്ഠിത പ്രസ്താവനയാണ്. ഏതു പ്രവൃത്തിയാണ് നിർവഹി

ക്കേണ്ടത് എന്ന് കംപ്യൂട്ടറിനോട് അത് പറയുന്നു. കൃത്യതയോടും സൂക്ഷ്മതയോടും കൂടി ചെയ്യേണ്ട പ്രവൃത്തി വ്യക്തമാക്കിയാൽ മാത്രമേ ഒരു നിർദേശത്തെ കമ്പ്യൂട്ടറിനു നിർവഹിക്കാൻ കഴിയുകയുള്ളൂ. പ്രശ്നപരിഹാരത്തിനായി പ്രോഗ്രാമർമാർ ഒരു പ്രത്യേക ക്രമത്തിൽ നിർദേശങ്ങൾ എഴുതുന്നു എന്ന് മുൻ അധ്യായത്തിൽ നാം പഠിച്ചിട്ടുണ്ട്. പ്രോഗ്രാം വികസിപ്പിക്കുകയും കമ്പ്യൂട്ടറിൽ സ്ഥിരമായി സൂക്ഷിക്കുകയും ചെയ്തു കഴിഞ്ഞാൽ നമുക്ക് ആവശ്യാനുസരണം അവ പ്രവർത്തിപ്പിക്കാൻ കമ്പ്യൂട്ടറിനോട് ആവശ്യപ്പെടാം

കമ്പ്യൂട്ടറിനു സാമാന്യബുദ്ധിയോ അന്തർജ്ഞാനമോ ഇല്ലാത്തതിനാൽ ഒരു പ്രോഗ്രാം രൂപകല്പന ചെയ്യുമ്പോൾ പ്രശ്ന പരിഹാരത്തിന്റെ യുക്തിയും നിർദേശങ്ങളുടെ ഘടനയും വ്യക്തമായി നാം മനസ്സിലാക്കിയിരിക്കണം. മനുഷ്യരായ നാം അനുഭവങ്ങളുടെ അടിസ്ഥാനത്തിൽ വിഷയാധിഷ്ഠിതവും വൈകാരികവുമായ പരിഗണനകൾക്കനുസരിച്ച്, തീരുമാനങ്ങൾ കൈക്കൊള്ളുന്നു. അത്തരം മൂല്യാധിഷ്ഠിത വിധിന്യായങ്ങൾ പലപ്പോഴും സാമാന്യബോധത്തെ ആശ്രയിച്ചിട്ടായിരിക്കും. ഇതിനു വിരുദ്ധമായി, ഒരു കമ്പ്യൂട്ടറിന് വികാരപ്രകടനമോ സാമാന്യബുദ്ധിയോ ഇല്ല. അതുകൊണ്ടാണ് കമ്പ്യൂട്ടറിന് സ്വന്തമായ ബുദ്ധിവൈഭവം ഇല്ല എന്നു നാം പറയുന്നത്.

ഒരു വിധത്തിൽ പറഞ്ഞാൽ, കമ്പ്യൂട്ടറിനെ ഒരു 'അനുസരണയുള്ള ഭൃത്യൻ' ആയി കണക്കാക്കാം. 'സാമാന്യബോധം' ഉപയോഗിക്കാതെയുള്ള അനുസരണശീലം പലപ്പോഴും അലോസരപ്പെടുത്തുന്നതും ഫലമില്ലാത്തതുമാകുന്നു.ഉദാഹരണത്തിന്, 'തപാൽ ഓഫീസിൽ പോയി പത്ത് 5 രൂപ സ്റ്റാമ്പുകൾ വാങ്ങുക'. എന്ന് നിർദേശിച്ച് തന്റെ അനുസരണയുള്ള ഭൃത്യനെ തപാൽ ഓഫീസിലേക്ക് യജമാനൻ അയച്ചു എന്ന് കരുതുക. ഭൃത്യൻ കാശുമായി തപാൽ ഓഫീസിൽ പോയിട്ട് ദീർഘനേരമായും തിരികെ വരുന്നില്ല. ചിന്താകുലനായ യജമാനൻ ഭൃത്യനെ അന്വേഷിച്ചു തപാൽ ഓഫീസിലെത്തുമ്പോൾ കാണുന്നത് സ്റ്റാമ്പുകളുമായി നിൽക്കുന്ന ഭൃത്യനെയാണ്.കോപിഷ്ഠനായ യജമാനൻ ഭൃത്യനോട് കാരണം അന്വേഷിക്കുമ്പോൾ, തന്നോട് പത്തു 5 രൂപ സ്റ്റാമ്പുകൾ വാങ്ങുവാൻ മാത്രമേ കല്പിച്ചിട്ടുള്ളൂ എന്നും, അവയുമായി തിരികെ വരാൻ നിർദേശിച്ചിട്ടില്ല എന്നുമുള്ള മറുപടിയാണ് ഭൃത്യനിൽ നിന്ന് ലഭിക്കുന്നത്. .

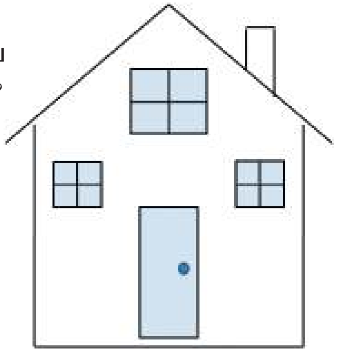
3.2 പ്രശ്നപരിഹാരത്തിലെ സമീപനങ്ങൾ (Approaches in problem solving)

ഒരു പ്രശ്നം വ്യത്യസ്ത രീതികളിലൂടെ പരിഹരിക്കാവുന്നതാണ്. സമീപനം പോലും വ്യത്യസ്തമായിരിക്കാം. നമ്മുടെ ദൈനംദിന ജീവിതത്തിൽ, അസുഖം ബാധിക്കുമ്പോൾ നാം ഒരു അലോപ്പതി, ആയുർവേദം അല്ലെങ്കിൽ ഹോമിയോപ്പതി ചികിത്സകനെ സമീപിച്ച് വൈദ്യചികിത്സ തേടുന്നു. ഒരേ രോഗമാണ് ചികിത്സിക്കുന്നതെങ്കിലും ഇവരിൽ ഓരോരുത്തരുടേയും സമീപനങ്ങൾ വ്യത്യസ്തമായിരിക്കും. അതു പോലെ പ്രശ്നപരിഹാരത്തിന് വ്യത്യസ്ത സമീപനങ്ങൾ ഉപയോഗിക്കുന്നു. പ്രശ്നപരിഹാരത്തിനുള്ള പ്രശസ്തമായ രണ്ടു രൂപകൽപ്പനാ രീതികൾ നമുക്ക് പരിചയപ്പെടാം ടോപ് ഡൗൺ (Top Down) രൂപകല്പനയും ബോട്ടം അപ് (Bottom Up) രൂപകല്പനയും

3.2.1 ടോപ് ഡൗൺ രൂപകല്പന (Top down design)

ചിത്രം 3.1 നോക്കുക. ഈ ചിത്രം വരയ്ക്കണമെന്ന് നിങ്ങളോട് ആവശ്യപ്പെടുകയാണെങ്കിൽ, എങ്ങനെയാണ് നിങ്ങൾ അത് വരയ്ക്കുക? ഒരു പക്ഷേ താഴെ പറയുന്ന പ്രകാരമായിരിക്കാം :

1. വീടിന്റെ രേഖാചിത്രം വരയ്ക്കുക.
2. ചിമ്മിനി വരയ്ക്കുക
3. വാതിൽ വരയ്ക്കുക
4. ജാലകങ്ങൾ വരയ്ക്കുക



ചിത്രം 3 .1 ഒരു വീടിന്റെ രേഖാചിത്രം

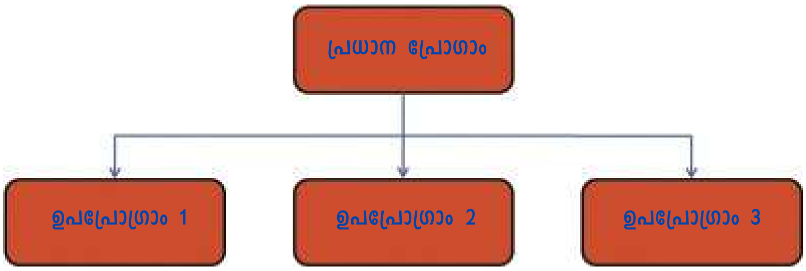
മുകളിൽ വിവരിച്ച നടപടിക്രമങ്ങളെ ഇപ്രകാരം സംഗ്രഹിക്കാം :

ഘട്ടം 3 ലെ വാതിൽ വരക്കുമ്പോഴുള്ള നടപടിക്രമം താഴെ കൊടുത്തിരിക്കുന്നു :
3.1 വാതിലിന്റെ ബാഹ്യരേഖ
3.2 ഷേഡിംഗ്
3.3 വാതിലിന്റെ പിടി

അത് പോലെ താഴെ പറയുന്ന പ്രകാരം ജാലകങ്ങൾ വരയ്ക്കാം
4.1 ജാലകത്തിന്റെ ബാഹ്യരേഖ
4.2 ഷേഡിംഗ്
4.3 തിരശ്ചീനവും ലംബവുമായ വരകൾ

തന്നിരിക്കുന്ന പ്രശ്നത്തെ (ഇവിടെ ചിത്രം വരയ്ക്കുക എന്നത്) ചെറിയ ക്രിയകളായി (Task) വിഭജിച്ചിരിക്കുന്നു. അതുപ്രകാരം ഈ പ്രശ്നം പരിഹരിക്കാൻ നാല് പ്രവർത്തനങ്ങൾ കണ്ടെത്തിയിട്ടുണ്ട്.ഇവയിൽ ചിലത് (ഇവിടെ വാതിലുകളും ജാലകങ്ങളും വരയ്ക്കുന്നത്) വീണ്ടും വിഭജിച്ചിട്ടുണ്ട്. അങ്ങനെ സങ്കീർണമായ ഒരു പ്രശ്നം,വിവിധ ക്രിയകളായി വിഭജിച്ച്, ഓരോ ക്രിയയെയും ലളിതമായ പ്രവർത്തനങ്ങളിലൂടെ പ്രാവർത്തികമാക്കാൻ കഴിയും. ഈ പ്രശ്ന പരിഹാര രീതി ടോപ് ഡൗൺ രൂപകല്പന (Top Down Design) എന്ന് അറിയപ്പെടുന്നു.

ഏറ്റവും ഫലവത്താണെന്നു തെളിയിക്കപ്പെട്ട പ്രോഗ്രാമിങ് സമീപനങ്ങളിലൊന്നാണിത്. ചിത്രം 3.2 ൽ കാണിച്ചിരിക്കുന്നതുപോലെ, ടോപ് ഡൗൺ രൂപകല്പന എന്നത് തന്നിരിക്കുന്ന നടപടി ക്രമത്തെ അല്ലെങ്കിൽ കൃത്യത്തെ ഘടകങ്ങൾ ആക്കുകയും ഏറ്റവും അടിസ്ഥാനപരമായ ക്രിയകൾ അടങ്ങിയ ഘടകം ലഭിക്കുന്നത് വരെ ഓരോ ഘടകത്തെയും വീണ്ടും വിഭജിക്കുകയും ചെയ്യുന്നു പ്രക്രിയയാണ്. ഒരു പൊതുവായ പ്രശ്നം മുകളിലെ തലം മുതൽ ആരംഭിക്കുകയും അതിലെ ഓരോ ഉപവിഭാഗത്തിനും പ്രത്യേക പരിഹാരങ്ങൾ രൂപകല്പന നടത്തുകയും ചെയ്യുന്നതിനാൽ ടോപ് ഡൗൺ വിഭജനം എന്ന പേരിലും ഇത് അറിയപ്പെടുന്നു. തന്നിരിക്കുന്ന പ്രധാന പ്രശ്നത്തിന് ഫലപ്രദമായ പരിഹാരം ലഭിക്കണമെങ്കിൽ ഓരോ ഉപപ്രശ്നവും മറ്റൊന്നിൽ നിന്ന് സ്വതന്ത്രമായിരിക്കണം. അപ്രകാരമായാൽ ഓരോ ഉപപ്രശ്നവും സ്വതന്ത്രമായി പരിഹരിക്കാനും പരിശോധിക്കാനും സാധിക്കും.



ചിത്രം 3.2: ഒരു പ്രശ്നത്തെ വിഭജിക്കുന്നു.

വിഭജനത്തിലൂടെ പ്രശ്നം പരിഹരിക്കുന്നത് കൊണ്ടുള്ള പ്രയോജനങ്ങൾ താഴെപ്പറയുന്നവയാണ്:

- പ്രശ്നത്തെ വിഭജിക്കുന്നതു കൊണ്ട് ഓരോ ഭാഗത്തും എന്ത് പ്രവൃത്തിയാണ് ചെയ്യേണ്ടതെന്നതിനെക്കുറിച്ച് ഒരു ധാരണ ലഭിക്കാൻ നമ്മെ സഹായിക്കുന്നു.

- ഓരോ ഘട്ടത്തിലും തിരിച്ചറിയുന്ന പുതിയ ഉപപ്രശ്നങ്ങളിൽ സങ്കീർണത കുറവായതിനാൽ അതിന്റെ പ്രശ്ന പരിഹാരം എളുപ്പത്തിൽ ലഭിക്കുന്നു.
- പ്രശ്ന പരിഹാരത്തിലെ ചില ഭാഗങ്ങൾ പുനരുപയോഗിക്കാൻ കഴിയുന്നതായിരിക്കാം.
- പ്രശ്നവിഭജനത്തിലൂടെ ഒന്നിലധികം ആളുകൾക്ക് ഒരേ സമയം പ്രശ്ന പരിഹാരത്തിൽ പങ്കാളികളാകാൻ സാധിക്കുന്നു.

3.2.2 ബോട്ടം അപ് രൂപകല്പന (Bottom Up Design)

ഒരു വീടിന്റെ നിർമ്മാണപ്രവർത്തനം പരിഗണിക്കുക. ടോപ് ഡൗൺ രൂപകല്പനയല്ല മറിച്ചു ബോട്ടം അപ് രൂപകല്പനയാണ് നമ്മൾ ഇവിടെ പിന്തുടരുന്നത്. അസ്ഥിവാരമിടുക എന്നത് ആദ്യത്തെ പ്രവൃത്തിയും മേൽക്കൂര പണിയുക എന്നത് അവസാനത്തെ പ്രവൃത്തിയുമാണ്. ഇവിടെയും പ്രധാന പ്രവർത്തനത്തെ ഉപപ്രവർത്തനങ്ങളായി വിഭജനം നടത്തുന്നു. ഇവയിൽ ത്തന്നെ ചില പ്രവർത്തനങ്ങൾ പൂർത്തിയായാൽ മാത്രമേ മറ്റു പ്രവർത്തനങ്ങൾ നടത്തുവാൻ കഴിയൂ. എന്നാൽ താഴെത്തട്ടിലുള്ള എല്ലാ പ്രവർത്തനങ്ങളും പൂർത്തിയായാൽ മാത്രമേ പ്രധാന പ്രവർത്തനമായ മേൽക്കൂര പണിയൽ സാധ്യമാവുകയുള്ളൂ.

ഇതുപോലെ പ്രോഗ്രാമിങ്ങിലും ആകെയുള്ള നടപടിക്രമങ്ങളെ വിവിധ ഘടകങ്ങളായി വിഭജിക്കുകയും, ഏറ്റവും താഴ്ന്ന തലത്തിലുള്ള ഖണ്ഡം ലഭിക്കുന്നത് വരെ ഈ ഘടകങ്ങൾ പുനർവിഭജിക്കുകയും ചെയ്യുന്നു. ഏറ്റവും താഴ്ന്ന ഘടകം മുതൽ പ്രശ്ന പരിഹാരം ആരംഭിക്കുന്നു. പ്രധാന പ്രശ്നത്തിനുള്ള പരിഹാരം, ഉപവിഭാഗങ്ങളുടെ പരിഹാരത്തിനു ശേഷം മാത്രമേ സാധ്യമാകൂ. ഈ രീതിയിലുള്ള സമീപനത്തെ പ്രശ്ന പരിഹാരത്തിനുള്ള ബോട്ടം അപ്പ് രൂപകല്പന എന്ന് പറയുന്നു. മുൻപ് പറഞ്ഞത് പോലെ ഇവിടെയും ഒരു ഉപപ്രശ്നം മറ്റൊരു ഉപപ്രശ്നത്തിൽ നിന്ന് സ്വതന്ത്രമായിരിക്കുന്നതാണ് അഭിലഷണീയം.

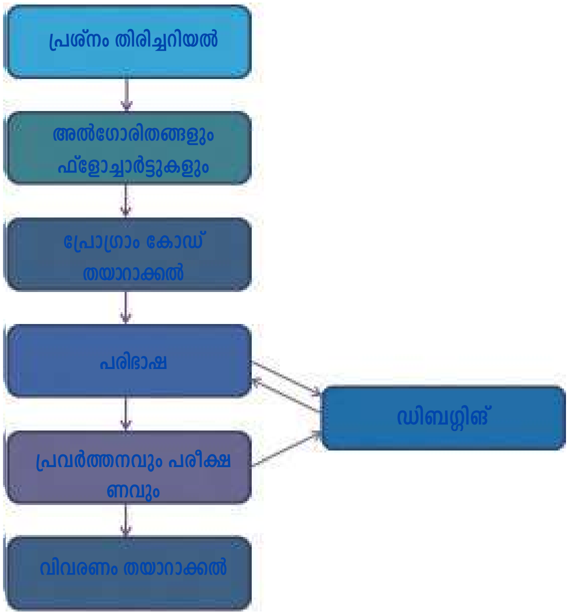
നമുക്കു ചെയ്യാം

എല്ലാ വർഷവും നമ്മുടെ സ്കൂളുകളിൽ യുവജനോത്സവം നടത്താറുണ്ട്. ഈ അവസരത്തിൽ, സാധാരണഗതിയിൽ ചുമതലകളും ഉത്തരവാദിത്വങ്ങളും വിഭജിച്ച് നൽകുന്നു. യുവജനോത്സവത്തിന്റെ വിജയകരമായ നടത്തിപ്പിന് എങ്ങനെയാണ് ഓരോ പ്രവർത്തനവും വിഭജിക്കുകയും പ്രാവർത്തികമാക്കുകയും ചെയ്യുന്നതെന്ന് ചർച്ച ചെയ്യുക.

3.3 പ്രോഗ്രാമിങ്ങിലെ വിവിധ ഘട്ടങ്ങൾ (Phases in programming)

കമ്പ്യൂട്ടർ ഉപയോഗിച്ച് പ്രശ്നം പരിഹരിക്കുക എന്നത് ഒരു വലിയ വെല്ലുവിളിയാണ് എന്ന് നാം കണ്ടുകഴിഞ്ഞു. ഇതിനായി ചിട്ടയായി ഒരു സമീപനം അത്യന്താപേക്ഷിതമാണ്. ആവശ്യമായ പ്രോഗ്രാമുകൾ നിർമ്മിക്കുന്നതിന് വിവിധ ഘട്ടങ്ങളിലൂടെ കടന്നുപോകേണ്ടതുണ്ട്. പ്രശ്ന പരിഹാരത്തിനുള്ള ജൻമസിദ്ധമായ കഴിവ് നമുക്കുണ്ടെങ്കിലും അത് ഫലപ്രദമായി ഉപയോഗപ്പെടുത്തണമെങ്കിൽ പ്രശ്നം പരിഹരിക്കുന്നതിന് ഉതകുന്ന രീതിയിലുള്ള ചിന്തയും, ആസൂത്രണവും യുക്തിസഹമായ ന്യായവാദവും വളർത്തിയെടുക്കേണ്ടതുണ്ട്. താഴെപ്പറയുന്ന ഘട്ടങ്ങൾ പിന്തുടർന്ന് നമുക്കിത് നേടിയെടുക്കാവുന്നതാണ്.

1. പ്രശ്നം തിരിച്ചറിയൽ (Problem Identification)
2. അൽഗോരിതങ്ങളും ഫ്ളോച്ചാർട്ടുകളും തയാറാക്കൽ (Preparing algorithms and Flowcharts)
3. പ്രോഗ്രാമിങ് ഭാഷ ഉപയോഗിച്ച് പ്രോഗ്രാം കോഡ് ചെയ്യൽ (Coding the Program using Programming Language)
4. പരിഭാഷ (Translation)
5. ഡീബഗ്ഗിങ് (Debugging)
6. പ്രവർത്തനവും പരീക്ഷണവും (Execution and Testing)
7. വിവരണം തയ്യാറാക്കുക (Documentation)



ചിത്രം 3.3 പ്രോഗ്രാമിങ്ങിലെ വിവിധ ഘട്ടങ്ങൾ

പ്രോഗ്രാമിങിന്റെ വിവിധ ഘട്ടങ്ങളിൽ ചെയ്യുന്ന പ്രവർത്തനങ്ങളുടെ ക്രമം ചിത്രം 3.3 ൽ കാണിച്ചിരിക്കുന്നു. ഡീബഗ്ഗിങ് ഘട്ടം പരിഭാഷയുമായും നിർവഹണവുമായും ബന്ധപ്പെട്ടിരിക്കുന്നു എന്നത് ശ്രദ്ധിക്കുക. മേൽപ്പറഞ്ഞ ഏഴ് ഘട്ടങ്ങളിൽ ഉൾപ്പെട്ടിരിക്കുന്ന പ്രവർത്തനങ്ങൾ താഴെ വിശദീകരിക്കുന്നു.

3.3.1 പ്രശ്നം തിരിച്ചറിയൽ (Problem Identification)

നിങ്ങൾക്കു വയറ് വേദന അനുഭവപ്പെടുന്നതായി കരുതുക. ഈ പ്രശ്നം ഒരു ഡോക്ടർക്കു പരിഹരിക്കാൻ കഴിയുമെന്ന് നിങ്ങൾക്കറിയാം. വേദന തുടങ്ങിയ സമയം,മുൻപ് ഇത് പോലെ വേദന അനുഭവപ്പെട്ട സാഹചര്യം, ഭക്ഷണരീതി എന്നിവയെക്കുറിച്ചുള്ള ചില ചോദ്യങ്ങൾ ഡോക്ടർ ചോദിക്കുകയും, സ്റ്റെതസ്കോപ്പ്, എക്സ്റേ അല്ലെങ്കിൽ സ്കാൻ എന്നിവ ഉപയോഗിച്ച് നിങ്ങളുടെ ശരീരത്തിന്റെ ചില ഭാഗങ്ങൾ പരിശോധിക്കുകയും ചെയ്യുന്നു.ഇവയെല്ലാം രോഗനിർണയത്തിന്റെ ഭാഗമാണ്. ഈ നടപടിക്രമങ്ങൾക്ക് ശേഷം, ഡോക്ടർ പ്രശ്നം തിരിച്ചറിഞ്ഞ് ചില വൈദ്യശാസ്ത്ര പദങ്ങൾ ഉപയോഗിച്ച് നിങ്ങളെ ബോധ്യപ്പെടുത്തുന്നു.അടുത്ത ഘട്ടം ഈ പ്രശ്നത്തിനുള്ള പരിഹാര നടപടികൾ തയാറാക്കലാണ്. അതിനെ മരുന്ന് കുറിപ്പ് എന്ന് പറയുന്നു.

ഇതിൽ നിന്നും പ്രശ്ന പരിഹാരത്തിനുള്ള നടപടികൾ സ്വീകരിക്കുന്നതിന് മുമ്പ് പ്രശ്നം വിശകലനം ചെയ്യേണ്ടത് അത്യാവശ്യമാണ് എന്ന് വ്യക്തമാകുന്നു. ഈ ഘട്ടത്തിൽ നിങ്ങൾക്ക് നടപ്പിലാക്കേണ്ട പ്രവർത്തനത്തിന് ആവശ്യമായ ഡാറ്റ, അതിന്റെ ഇനം, അളവ്, ഉപയോഗിക്കേണ്ട സൂത്രവാക്യം, ഉൾപ്പെട്ടിരിക്കുന്ന പ്രവർത്തനങ്ങൾ ലഭിക്കേണ്ട ഔട്ട്പുട്ട് എന്നിവ തിരിച്ചറിയാൻ സാധിക്കുന്നു. പ്രശ്നം വ്യക്തമായി പഠിക്കുകയും, പ്രശ്നപരിഹാരത്തിനാവശ്യമായ കൃത്യങ്ങളുടെ ക്രമം സംബന്ധിച്ച് നമുക്ക് ബോധ്യമാകുകയും ചെയ്താൽ, അടുത്ത ഘട്ടത്തിലേക്ക് പ്രവേശിക്കാം. പ്രോഗ്രാമറുടെ (പ്രശ്ന പരിഹാരകൻ) കാര്യക്ഷമത പരമാവധി ഉപയോഗപ്പെടുത്തേണ്ടതിനാൽ തീർച്ചയായും ഇത് ഒരു വെല്ലുവിളി നിറഞ്ഞ ഘട്ടമാണ്.

3.3.2 അൽഗോരിതങ്ങളും ഫ്ളോച്ചാർട്ടുകളും (Algorithms and Flowcharts)

പ്രശ്നം തിരിച്ചറിഞ്ഞു കഴിഞ്ഞാൽ അത് പരിഹരിക്കാൻ പടിപടിയായുള്ള നടപടിക്രമങ്ങൾ കൃത്യമായി വികസിപ്പിക്കേണ്ടത് അത്യാവശ്യമാണ്. ഈ നടപടിക്രമം പുതിയതോ കമ്പ്യൂട്ടർ മേഖലയ്ക്കുമാത്രം പരിമിതമായതോ അല്ല. കാലാകാലമായി ജീവിതത്തിന്റെ എല്ലാ മേഖലകളിലും തുടർന്ന് കൊണ്ടിരിക്കുന്ന ഒന്നാണിത്. നിത്യ ജീവിതത്തിൽ നിന്നെടുത്ത അത്തരത്തിലുള്ള ഒരു നടപടിക്രമം താഴെ വിവരിച്ചിരിക്കുന്നു. ഒരു മാസികയിൽ നിന്നും എടുത്തിട്ടുള്ള ഓംലെറ്റ് തയാറാക്കാനുള്ള പാചകക്കുറിപ്പാണിത്

ചേരുവകൾ

മുട്ട 2 എണ്ണം, ഉള്ളി 1 എണ്ണം (ചെറുതായി അരിഞ്ഞത്); പച്ച മുളക് 2 (ചെറുതായി അരിഞ്ഞത്); എണ്ണ 2 ടീ സ്പൂൺ, ഉപ്പ് ഒരു നുള്ള്.

രീതി

ഘട്ടം 1 : മുട്ടകൾ പൊട്ടിച്ച് ഒരു പാത്രത്തിലിട്ട് നന്നായി ഇളക്കുക.

ഘട്ടം 2 : അരിഞ്ഞു വച്ച ഉള്ളി, പച്ചമുളക്, ഉപ്പ് എന്നിവ മുട്ടയിൽ ചേർത്തിളക്കുക.

ഘട്ടം 3 : അടുപ്പിൽ ഒരു പാൻ വച്ച് അടുപ്പു കത്തിക്കുക.

ഘട്ടം 4 : പാനിൽ എണ്ണ ഒഴിച്ച്, ചൂടാകുന്നതുവരെ കാത്തിരിക്കുക.

ഘട്ടം 5 : ഘട്ടം 2 ൽ തയാറാക്കിയ മിശ്രിതം പാനിൽ ഒഴിച്ച് ഒരു ഭാഗം പൊരിയുന്നത് വരെ കാത്തിരിക്കുക.

ഘട്ടം 6 : മറിച്ചിട്ട് മറുവശം നന്നായി പൊരിക്കുക.

ഘട്ടം 7 : കുറച്ച് സെക്കൻഡുകൾക്ക് ശേഷം ഇത് എടുക്കുക.

ഫലം

കുരുമുളക് പൊടി ചേർത്ത് കഴിക്കാൻ പാകത്തിനുള്ള ഒരു ഓംലെറ്റ് തയാർ.

മുകളിൽ കൊടുത്ത പാചകകുറിപ്പിന് താഴെപ്പറയുന്ന സവിശേഷതകൾ ഉണ്ട്:

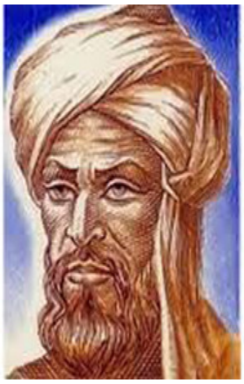
1. ഓംലെറ്റ് പാചകം ചെയ്യാൻ ആവശ്യമായ ചേരുവകളുടെ ഒരു പട്ടിക നിർമിക്കുന്നതിലൂടെയാണ് പ്രവർത്തനം ആരംഭിക്കുന്നത്. ഈ ചേരുവകളെ ഇൻപുട്ടുകൾ എന്ന് വിളിക്കാം.
2. ഇൻപുട്ടുകൾ ഉപയോഗിച്ചുള്ള പ്രവർത്തനത്തിന് ആവശ്യമായ, ക്രമത്തിലുള്ള നിർദേശങ്ങൾ നൽകുന്നു.
3. നിർദേശങ്ങൾ നടപ്പാക്കുന്നതിന്റെ ഫലമായി ചില ഔട്ട്പുട്ടുകൾ (ഇവിടെ, ഓംലെറ്റ്) ലഭിക്കുന്നു.

എന്നാൽ ഇൻപുട്ടുകൾ ഉപയോഗിച്ച് പ്രവർത്തനങ്ങൾക്കുള്ള നിർദേശങ്ങൾ കൃത്യമല്ല. അവ അവ്യക്തമാണ്. ഉദാഹരണത്തിന്, അഞ്ചാം ഘട്ടത്തിൽ "ഒരു ഭാഗം പൊരിക്കുന്നത്", ആറാം ഘട്ടത്തിൽ "നന്നായി പൊരിക്കുക" തുടങ്ങിയ നിർദേശങ്ങളുടെ വ്യാഖ്യാനം വ്യക്തികൾക്കനുസരിച്ചു വ്യത്യാസപ്പെട്ടിരിക്കും. തന്മൂലം കൃത്യമായ നിർദേശങ്ങളും, സമാന ഇൻപുട്ടുകളുമായി ഒരേ പാചകക്കുറിപ്പ് പിന്തുടരുന്ന വ്യത്യസ്ത വ്യക്തികൾക്ക്, വലുപ്പം, ആകൃതി, രുചി എന്നിവയ്ക്കനുസരിച്ച് വ്യത്യസ്ത ഓംലെറ്റ്കൾ ലഭിക്കുന്നു.

കമ്പ്യൂട്ടർ ഉപയോഗിച്ചുള്ള പ്രശ്നങ്ങൾ പരിഹരിക്കുന്നതിനുള്ള നടപടികൾ എഴുതുമ്പോൾ മുകളിൽ പറഞ്ഞ അവ്യക്തതകൾ ഒഴിവാക്കേണ്ടതാണ്.

a. അൽഗോരിതം (Algorithm)

അബു ജഅഫർ മുഹമ്മദ് ഇബ്നു മുസാ അൽ ഖവാറീസ്മി എന്ന അറബി ഗണിതജ്ഞനാണ് അൽഗോരിതം എന്ന വാക്കിന്റെ ഉപജ്ഞാതാവായി അറിയപ്പെടുന്നത്. അദ്ദേഹത്തിന്റെ പേരിന്റെ 'അൽ ഖവാരിസിമി' എന്ന പദത്തിൽ നിന്നാണ് അൽഗോരിതം എന്ന പേര് ലഭിച്ചത്. കമ്പ്യൂട്ടർ പദാവലിയിൽ ഒരു പ്രശ്നം പരിഹരിക്കുന്നതിനുള്ള ക്രമത്തിലുള്ള നിശ്ചിത നിർദേശങ്ങളെ അൽഗോരിതം എന്നു നിർവചിക്കാവുന്നതാണ്. പ്രശ്നം പരിഹരിക്കാനുള്ള ഘട്ടംഘട്ടമായ നടപടിക്രമാണ് അൽഗോരിതം. ഇതിലെ ഓരോ ഘട്ടവും ചെയ്യപ്പെടേണ്ട നിശ്ചിതമായ കൃത്യത്തെ പ്രതിനിധീകരിക്കുന്നു. എന്നിരുന്നാലും, ഒരു അൽഗോരിതം ആകണമെങ്കിൽ, ക്രമത്തിലുള്ള നിർദേശങ്ങൾക്ക് താഴെപ്പറയുന്ന സവിശേഷതകൾ ഉണ്ടായിരിക്കേണ്ടതാണ് :



ചിത്രം 3.4 അബു ജഅഫർ മുഹമ്മദ് ഇബ്നു മുസാ അൽ ഖവാറീസ്മി (780 - 850)

(i) ഇൻപുട്ടുകൾ സ്വീകരിക്കുന്നതിനുള്ള നിർദേശം (നിർദേശങ്ങൾ) കൊണ്ടായിരിക്കണം അതിന്റെ തുടക്കം. തുടർന്നുള്ള നിർദേശങ്ങൾ വഴി ഈ ഇൻപുട്ടുകൾക്ക് മേൽ പ്രവർത്തനങ്ങൾ നടപ്പിലാക്കുന്നു. ചില സാഹചര്യങ്ങളിൽ, ഉപയോഗിക്കേണ്ട ഡാറ്റ പ്രശ്നത്തിനോടൊപ്പം തന്നെ നൽകിയിരിക്കും. അത്തരം സാഹചര്യങ്ങളിൽ, അൽഗോരിതത്തിന്റെ തുടക്കത്തിൽ ഇൻപുട്ട് സ്വീകരിക്കാനുള്ള നിർദേശങ്ങൾ ഉണ്ടായിരിക്കുകയില്ല .

(ii) ഡാറ്റയെ സൂചിപ്പിക്കാൻ വേരിയബിളുകൾ ഉപയോഗിക്കുക. ഇവിടെ വേരിയബിളുകൾ എന്നതു ഗണിതശാസ്ത്രത്തിലേതു പോലെ അക്ഷരങ്ങളും അക്കങ്ങളും അടങ്ങിയ ഉപയോക്തൃ നിർവചിത വാക്കുകളാണ്. ഡാറ്റ ഇൻപുട്ട് ചെയ്യുന്നതിനും വിലകൾ/ഫലങ്ങൾ സംഭരിക്കുന്നതിനും വേരിയബിളുകൾ തീർച്ചയായും ഉപയോഗിക്കേണ്ടതാണ്.

(iii) ഓരോ നിർദേശവും കൃത്യവും സ്പഷ്ടവും ആയിരിക്കണം. മറ്റൊരു വിധത്തിൽ പറഞ്ഞാൽ, നിർദേശങ്ങൾ അവ്യക്തമായിരിക്കരുത്. മാത്രമല്ല അവ നടപ്പിലാക്കാൻ സാധ്യമായതുമാകണം.

(iv) ഒരു വ്യക്തിക്ക് പേപ്പറും പെൻസിലും ഉപയോഗിച്ച് നിശ്ചിത സമയം കൊണ്ട് ചെയ്തു തീർക്കാവുന്ന തരത്തിൽ അടിസ്ഥാനപരമായതായിരിക്കണം ഓരോ നിർദേശവും.

(v) അൽഗോരിതത്തിൽ പറഞ്ഞിരിക്കുന്ന എല്ലാ നടപടികളും ചെയ്യുവാനുള്ള സമയം നിശ്ചിതമായിരിക്കണം. എന്തെന്നാൽ അൽഗോരിതത്തിലെ ചില നിർദേശങ്ങൾ ആവർത്തിച്ചു നിർവഹിക്കേണ്ടവയായിരിക്കും. അത്തരത്തിലുള്ള ആവർത്തനങ്ങളുടെ എണ്ണം നിശ്ചിതമായിരിക്കണം എന്നാണ് ഇത് സൂചിപ്പിക്കുന്നത്.

(vi) അൽഗോരിത്തിൽ നൽകിയിരിക്കുന്ന നിർദേശങ്ങൾ നിർവഹിച്ചാൽ പ്രതീക്ഷിച്ച ഫലം (ഔട്ട്പുട്ട്) ലഭിച്ചിരിക്കേണ്ടതാണ്.

അൽഗോരിതം സംബന്ധിച്ച ഉൾക്കാഴ്ച നേടാൻ നമുക്ക് ഒരു ലളിതമായ ഉദാഹരണം പരിഗണിക്കാം. തന്നിരിക്കുന്ന മൂന്ന് സംഖ്യകളുടെ ആകത്തുകയും (sum) ശരാശരിയും (average) നമുക്ക് കണ്ടുപിടിക്കണം.ഈ പ്രശ്നം പരിഹരിക്കാനുള്ള നടപടിക്രമം നമുക്ക് താഴെ പറയും പ്രകാരം എഴുതാം:

ഘട്ടം 1: മൂന്ന് സംഖ്യകൾ ഇൻപുട്ട് ചെയ്യുക.
ഘട്ടം 2: ആകത്തുക ലഭിക്കുന്നതിന് ഈ സംഖ്യകൾ കൂട്ടുക.
ഘട്ടം 3: ശരാശരി ലഭിക്കുന്നതിന് ആകത്തുകയെ 3 കൊണ്ട് ഹരിക്കുക.
ഘട്ടം 4: ആകത്തുകയും ശരാശരിയും പ്രിൻറ് ചെയ്യുക.

ഇവിടെ നടപടിക്രമം ശരിയാണെങ്കിലും, ഒരു അൽഗോരിതം തയാറാക്കുമ്പോൾ, ഒരു അംഗീകൃത ഘടന നമ്മൾ പിന്തുടരേണ്ടതുണ്ട്. മുകളിൽ പറഞ്ഞ പ്രക്രിയ ഒരു അൽഗോരിതം രൂപത്തിൽ എങ്ങനെ എഴുതാം എന്ന് നോക്കാം.

ഉദാഹരണം 3.1: മൂന്ന് സംഖ്യകളുടെ ആകത്തുക,ശരാശരി എന്നിവ കാണാനുള്ള അൽഗോരിതം.

ഇൻപുട്ട് ചെയ്യുന്ന സംഖ്യകൾ സ്വീകരിക്കാൻ A, B, C എന്ന വേരിയബിളുകൾ ഉപയോഗിക്കുക. അതുപോലെ S ആകത്തുകയ്ക്കു വേണ്ടിയും, Avg ശരാശരിക്കു വേണ്ടിയുമുള്ള വേരിയബിളുകൾ ആക്കുക.

```
ഘട്ടം 1:   ആരംഭിക്കുക.
ഘട്ടം 2:   A, B, C ഇൻപുട്ട് ചെയ്യുക.
ഘട്ടം 3:   $S = A+ B + C$.
ഘട്ടം 4:   $Avg = S / 3$.
ഘട്ടം 5:   S, Avg പ്രിൻറ് ചെയ്യുക.
ഘട്ടം 6:   അവസാനിപ്പിക്കുക
```

താഴെ പറയുന്ന കാരണങ്ങളാൽ മുകളിൽ പറഞ്ഞിരിക്കുന്ന നിർദേശങ്ങളുടെ കൂട്ടത്തെ അൽഗോരിതം ആയി കണക്കാക്കുന്നു:

- ഇതിന് ഇൻപുട്ട് ഉണ്ട് (ഇൻപുട്ട് ഡാറ്റ സംഭരിക്കാൻ വേരിയബിളുകൾ A, B, C ഉപയോഗിക്കുന്നു).
- ഒരു വ്യക്തിക്ക് പേപ്പറും പെൻസിലും ഉപയോഗിച്ച് കൃത്യമായി നിർവഹിക്കാവുന്ന രൂപത്തിൽ നടപടികൾ കൃത്യമായി പ്രസ്താവിച്ചിരിക്കുന്നു. (ഘട്ടം 3 ലും ഘട്ടം 4 ലും ഉചിതമായ ഓപ്പറേറ്ററുകൾ ഉപയോഗിച്ചുകൊണ്ട്)

- ഓരോ നിർദേശവും അടിസ്ഥാനപരവും അർഥവത്തുമാണ് (ഇൻപുട്ട്, പ്രിന്റ്, കൂട്ടുക, ഹരിക്കുക).
- ഇത് ആകെത്തുക (S), ശരാശരി (Avg) എന്നിങ്ങനെ രണ്ടു ഔട്ട്പുട്ടുകൾ സൃഷ്ടിക്കുന്നു.
- ആരംഭവും അവസാനവും സൂചിപ്പിക്കാനായി തുടങ്ങുക, നിർത്തുക എന്നീ നിർദേശങ്ങൾ ഉപയോഗിച്ചിരിക്കുന്നു.

നിർദേശങ്ങളുടെ തരങ്ങൾ

നമുക്കറിയാം ഒരു കമ്പ്യൂട്ടറിന് നിർവഹിക്കാൻ കഴിയുന്ന ക്രിയകളുടെ എണ്ണം പരിമിതമാണ്. അതിനാൽ പ്രശ്നപരിഹാരത്തിന് അത്രയും നിർദേശങ്ങൾ മാത്രമേ നമുക്ക് ഉപയോഗിക്കാൻ സാധിക്കുകയുള്ളൂ. കൂടുതൽ അൽഗോരിതങ്ങൾ നിർമിക്കുന്നതിന് മുമ്പ് അൽഗോരിതം നിർമിക്കാൻ ഉപയോഗിക്കുന്ന നിർദേശങ്ങളുടെ തരം ഏതൊക്കെയാണെന്ന് നമുക്ക് നോക്കാം.

- നമ്മൾ നൽകുന്ന ഡാറ്റ കമ്പ്യൂട്ടറിന് സ്വീകരിക്കാൻ സാധിക്കുന്നു. ആയതിനാൽ ഇൻപുട്ടിനുള്ള നിർദേശങ്ങൾ നമുക്ക് ഉപയോഗിക്കാവുന്നതാണ്. ഇൻപുട്ട്, സ്വീകരിക്കുക, വായിക്കുക മുതലായ പദങ്ങൾ നമുക്ക് ഇതിനായി ഉപയോഗിക്കാം.

- കമ്പ്യൂട്ടർ ഫലങ്ങൾ ഔട്ട്പുട്ടായി നൽകുന്നു.ആയതിനാൽ നമുക്ക് ഔട്ട്പുട്ടുമായി ബന്ധപ്പെട്ട നിർദേശങ്ങൾ ഉപയോഗിക്കാം. പ്രിന്റ്, പ്രദർശിപ്പിക്കുക, എഴുതുക മുതലായ പദങ്ങൾ നമുക്ക് ഇതിന് ഉപയോഗിക്കാം.

- ഒരു മെമ്മറി സ്ഥാനത്ത് ഡാറ്റ നേരിട്ട് ശേഖരിക്കാം അല്ലെങ്കിൽ ഡാറ്റ ഒരു സ്ഥാനത്ത് നിന്ന് മറ്റൊന്നിലേക്ക് പകർത്തുകയും ചെയ്യാം. അതുപോലെ, ഡാറ്റയുടെ മുകളിലുള്ള ഗണിത പ്രവർത്തനങ്ങളുടെ ഫലങ്ങൾ മെമ്മറി സ്ഥാനങ്ങളിൽ സംഭരിക്കാം. ഇതിനായി ഗണിത ശാസ്ത്രത്തിലേതിനു സമാനമായി വിലനൽകൽ (assignment) (അല്ലെങ്കിൽ സംഭരണം) നിർദേശം നമുക്ക് ഉപയോഗിക്കാം. മൂല്യങ്ങൾ സംഭരിക്കുന്നതിന് വേരിയബിളുകൾക്കു ശേഷം സമം ചിഹ്നം (=) ഉപയോഗിക്കുന്നു. ഇവിടെ വേരിയബിളുകൾ എന്നത് മെമ്മറി സ്ഥാനങ്ങളെ സൂചിപ്പിക്കുന്നു.

- കമ്പ്യൂട്ടറിന് ഡാറ്റ മൂല്യങ്ങൾ താരതമ്യം ചെയ്ത് (ലോജിക്കൽ ഓപ്പറേഷൻ എന്ന് വിളിക്കുന്നു), അതിന്റെ അടിസ്ഥാനത്തിലുള്ള തീരുമാനങ്ങൾ എടുക്കാൻ സാധിക്കും. ഇത്തരം തീരുമാനങ്ങൾ ഒന്നോ അതിലധികമോ പ്രസ്താവനകളുടെ തിരഞ്ഞെടുക്കൽ / ഒഴിവാക്കൽ അല്ലെങ്കിൽ ഒരുകൂട്ടം നിർദേശങ്ങളുടെ ആവർത്തിച്ചുള്ള പ്രവർത്തനം എന്ന രൂപത്തിലായിരിക്കും.

b. ഫ്ളോച്ചാർട്ടുകൾ (Flowchart)

ഒരു ചിത്രം അല്ലെങ്കിൽ രേഖാചിത്രത്തിന്റെ രൂപത്തിൽ ആവിഷ്കരിക്കപ്പെട്ടിരിക്കുന്ന ഒരു സങ്കല്പം ആണ് എഴുത്ത് രൂപത്തെക്കാൾ ആളുകൾക്ക് സ്വീകാര്യമാകുക. ചില സാഹചര്യങ്ങളിൽ അൽഗോരിതം മനസ്സിലാക്കുക എന്നത് ബുദ്ധിമുട്ടേറിയതായിരിക്കും. എന്തെന്നാൽ ചില സങ്കീർണമായ പ്രവർത്തനങ്ങളും ആവർത്തിച്ചുവരുന്ന ഘട്ടങ്ങളും അതിൽ ഉൾപ്പെട്ടേക്കാം. അതിനാൽ അൽഗോരിതം ചിത്ര രൂപത്തിൽ ആവിഷ്കരിക്കുക എന്നതായിരിക്കും മെച്ചപ്പെട്ട രീതി. നിർദേശങ്ങൾ സൂചിപ്പിക്കുന്നതിനുള്ള പ്രത്യേക ചിഹ്നങ്ങളും പ്രവർത്തനങ്ങളുടെ ക്രമം സൂചിപ്പിക്കുന്നതിന് ആരോകളും ഉപയോഗിച്ച് നിർമിക്കുന്ന അൽഗോരിതത്തിന്റെ ചിത്ര ആവിഷ്കരണമാണ് ഫ്ളോച്ചാർട്ട്. ഒരു അൽഗോരിതം ചിട്ടപ്പെടുത്തുന്നതിനും മനസ്സിലാക്കുന്നതിനുമുള്ള ഒരു സഹാ

യിയായിട്ടാണ് പ്രധാനമായിട്ടും ഇത് ഉപയോഗിക്കുന്നത്. വിവിധതരത്തിലുള്ള നിർദേശങ്ങൾ സൂചിപ്പിക്കുന്നതിനായി അടിസ്ഥാനപരമായ ജ്യാമിതീയ രൂപങ്ങൾ ഫ്ളോച്ചാർട്ടുകളിൽ സാധാരണമായി ഉപയോഗിക്കുന്നു. യഥാർഥ നിർദേശങ്ങൾ ഇത്തരം രൂപങ്ങൾക്കുള്ളിൽ കൃത്യവും വ്യക്തവുമായ പ്രസ്താവനകൾ ഉപയോഗിച്ച് എഴുതുന്നു. പ്രവർത്തനങ്ങളുടെ ക്രമത്തെ അതായത് നിർദേശങ്ങൾ പ്രാവർത്തികമാക്കേണ്ട ക്രമത്തെ സൂചിപ്പിക്കുന്ന തരത്തിൽ ഈ രൂപങ്ങൾ ആരോകളോട് കൂടിയ നേർരേഖകൾ വഴി പരസ്പരം ബന്ധപ്പെടുത്തിയിരിക്കുന്നു.

സാധാരണഗതിയിൽ ഒരു അൽഗോരിതത്തെ ഫ്ളോച്ചാർട്ടിലേക്ക് രൂപഭേദം വരുത്തിയ ശേഷമാണ് നിർദേശങ്ങൾ പ്രോഗ്രാമിങ് ഭാഷയിൽ ആവിഷ്കരിക്കുന്നത്. പ്രോഗ്രാം എഴുതുന്നതിൽ ഈ ദ്വിതല (Two step approach) സമീപനത്തിന്റെ മുഖ്യ ഗുണം എന്തെന്നാൽ ഫ്ളോച്ചാർട്ട് വരയ്ക്കുന്ന സമയത്ത് പ്രോഗ്രാം ഭാഷയുടെ വിവിധ ഘടകങ്ങളുടെ വിശദാംശങ്ങളെ പറ്റി വരയ്ക്കുന്നയാൾ ചിന്തിക്കേണ്ടതില്ല. അതിനാൽ അവന്/അവൾക്ക് നടപടിക്രമത്തിന്റെ യുക്തിയിൽ (ഘട്ടംഘട്ടമായുള്ള രീതി) കൂടുതൽ ശ്രദ്ധ പതിപ്പിക്കാൻ സാധിക്കുന്നു. മാത്രമല്ല, ഫ്ളോച്ചാർട്ടിൽ, പ്രവർത്തനങ്ങളുടെ ക്രമം ചിത്രരൂപത്തിൽ സൂചിപ്പിക്കുന്നതിനാൽ നടപടിക്രമത്തിന്റെ യുക്തിയിൽ വരുന്ന തെറ്റുകൾ, പ്രോഗ്രാമിൽ തെറ്റ് കണ്ടെത്തുന്നതിനെക്കാൾ എളുപ്പത്തിൽ കണ്ടെത്താൻ കഴിയുന്നു. അൽഗോരിതവും ഫ്ളോച്ചാർട്ടും എപ്പോഴും പ്രോഗ്രാമറിനുള്ള പ്രമാണം ആകുന്നു. ഒരിക്കൽ ഇവ തയാറാകുകയും പ്രശ്ന പരിഹാരത്തിന്റെ യുക്തി ശരിയാണെന്ന് ബോധ്യമാവുകയും ചെയ്തു കഴിഞ്ഞാൽ പ്രോഗ്രാമർക്ക് പ്രോഗ്രാമിങ് ഭാഷയിലുള്ള വിവിധ നിർമ്മിതികളുടെ സഹായത്തോടെ ചെയ്യേണ്ട പ്രവർത്തനങ്ങളെ കോഡ് ചെയ്യുന്നതിൽ പ്രോഗ്രാമർക്ക് ശ്രദ്ധ ചെലുത്താൻ സാധിക്കുന്നു. പ്രോഗ്രാം തെറ്റുകൾ ഇല്ലാത്തതാണെന്നു ഉറപ്പു വരുത്താൻ സാധാരണ ഗതിയിൽ ഇതിനു കഴിയുന്നു.

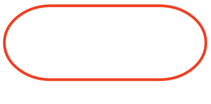
ഫ്ളോച്ചാർട്ടിലെ ചിഹ്നങ്ങൾ

അംഗീകരിക്കപ്പെട്ട അർഥവത്തായ ചിഹ്നങ്ങൾ ഉപയോഗിക്കുന്നതിലൂടെ ഫ്ളോചാർട്ടുകൾ വഴിയുള്ള പ്രോഗ്രാം യുക്തിയുടെ ആശയ വിനിമയം എളുപ്പമായി തീരുന്നു. അവശ്യമായ ചില പ്രവർത്തനങ്ങൾ സൂചിപ്പിക്കുന്നതിന് ആവശ്യമായ ചില ചിഹ്നങ്ങൾ മാത്രമാണ് നമ്മൾ ഇവിടെ പരിചയപ്പെടുന്നത്. ഈ ചിഹ്നങ്ങൾ അമേരിക്കൻ നാഷണൽ സ്റ്റാൻഡേർഡ്സ് ഇൻസ്റ്റിറ്റ്യൂട്ട് (ANSI) അംഗീകരിച്ചതാണ്.

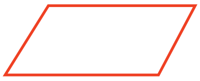
1. ടെർമിനൽ (Terminal)

പേര് സൂചിപ്പിക്കുന്നതുപോലെ, പ്രോഗ്രാം യുക്തിയുടെ ആരംഭത്തെയും അവസാനത്തേയും ഈ ചിഹ്നം സൂചിപ്പിക്കുന്നു. ഒരു ഫ്ളോച്ചാർട്ടിന്റെ ആദ്യത്തെയും അവസാനത്തെയും ചിഹ്നമാണിത്.

ഒരു അണ്ഡവൃത്തത്തിന്റെ (ellipse) ആകൃതിയാണ് ഇതിന്. തുടക്കത്തിൽ ഉപയോഗിക്കുമ്പോൾ നിർഗമനത്തിന്റെ ആരോ ഇതിൽ ഘടിപ്പിച്ചിരിക്കും. പക്ഷേ അവസാനത്തിൽ ഉപയോഗിക്കുമ്പോൾ ആഗമനത്തിന്റെ ആരോ ഇതിൽ ഘടിപ്പിച്ചിരിക്കും.

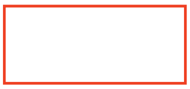


2. ഇൻപുട്ട്/ഔട്ട്പുട്ട് (input/output)



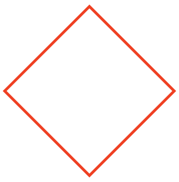
ഇൻപുട്ട്/ഔട്ട്പുട്ട് ചിഹ്നമായി ഉപയോഗിക്കുന്നത് ഒരു സമാന്തരഭുജ (Parallelogram) മാണ്. പ്രോഗ്രാമിൽ ഒരു ഇൻപുട്ട് /ഔട്ട്പുട്ട് ഉപകരണത്തിന്റെ ധർമ്മത്തെ ഇത് സൂചിപ്പിക്കുന്നു. എല്ലാ ഇൻപുട്ട്/ഔട്ട്പുട്ട് നിർദേശങ്ങളും ഈ ചിഹ്നം ഉപയോഗിച്ചാണ് ആവിഷ്കരിച്ചിരിക്കുന്നത്. ഇതിലേക്ക് ഒരു ആഗമന ആരോയും ഒരു നിർഗമന ആരോയും ഘടിപ്പിച്ചിരിക്കും.

3. പ്രവർത്തനം (process)



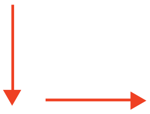
പ്രവർത്തന ഘട്ടത്തെ പ്രതിനിധീകരിക്കാൻ ദീർഘചതുരം ഉപയോഗിക്കുന്നു. സങ്കലനം, വ്യവകലനം ഗുണിനം ഹരണം തുടങ്ങിയ ഗണിത ക്രിയകൾ ചെയ്യാനും അതുപോലെ വേരിയബിളിയിലേക്ക് വില നൽകുവാനും ഈ ചിഹ്നം ഉപയോഗിക്കുന്നു. വേരിയബിളുകൾക്ക് വില നൽകുക (assignment) എന്നത് കൊണ്ടുദ്ദേശിക്കുന്നത്- ഒരു മെമ്മറി സ്ഥാനത്തു നിന്ന് മറ്റൊന്നിലേക്ക് ഡാറ്റ പകർത്തുന്നതോ (ഉദാ:a=b) അല്ലെങ്കിൽ എ.ൽ.യു. വിൽ (ALU) നിന്നും ഔട്ട്പുട്ടിനെ മെമ്മറി സ്ഥാനത്തേക്ക് പകർത്തുന്നതോ (ഉദാ:a=b+5) അല്ലെങ്കിൽ ഒരുപക്ഷേ മെമ്മറി സ്ഥാനത്തേക്ക് വില നേരിട്ട് സംഭരിക്കുന്നതോ (ഉദാ:a=2) ആകാം. പ്രോസസ്സ് ചിഹ്നത്തിനു ഒരു ആഗമന ആരോയും ഒരു നിർഗമന ആരോയും ഉണ്ടായിരിക്കും.

4. തീരുമാനം (Decision)



തീരുമാനങ്ങൾ സൂചിപ്പിക്കുന്നതിനായുള്ള ചിഹ്നമായി ചതുർഭുജം ഉപയോഗിക്കുന്നു. ഇത് തീരുമാനങ്ങൾ കൈക്കൊള്ളേണ്ട ഒരു ഘട്ടത്തെയാണ് സൂചിപ്പിക്കുന്നത്. ഈ ഘട്ടത്തിൽ നിന്ന് ഒന്നോ അതിലധികമോ ഇതര ഘട്ടങ്ങളിലേക്ക് വിഭജനം സാധ്യമാണ്. എല്ലാ നിർഗമന പാതകളും ഇവിടെ പരാമർശിച്ചിട്ടുണ്ടായിരിക്കും. എന്നിരുന്നാലും ഒരു വ്യവസ്ഥ (condition) പരിശോധിച്ച് അതിന്റെ ഫലത്തിന്റെ അടിസ്ഥാനത്തിൽ അടുത്ത ഒരു പാത മാത്രമേ തിരഞ്ഞെടുക്കപ്പെടുകയുള്ളൂ. സാധാരണഗതിയിൽ ഈ ചിഹ്നത്തിന് ഒരു ആഗമന മാർഗവും 2 നിർഗമന മാർഗങ്ങളും ഉണ്ടായിരിക്കും. ഒന്ന് വ്യവസ്ഥയുടെ ഫലം ശരിയാണെങ്കിൽ ചെയ്യേണ്ടുന്ന പ്രവൃത്തിയുടെ നേർക്കും, മറ്റേത് ഇതര മാർഗത്തിലേക്കും.

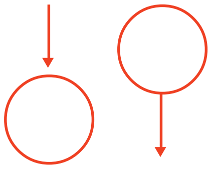
5. ഫ്ളോ ലൈനുകൾ (flow lines)



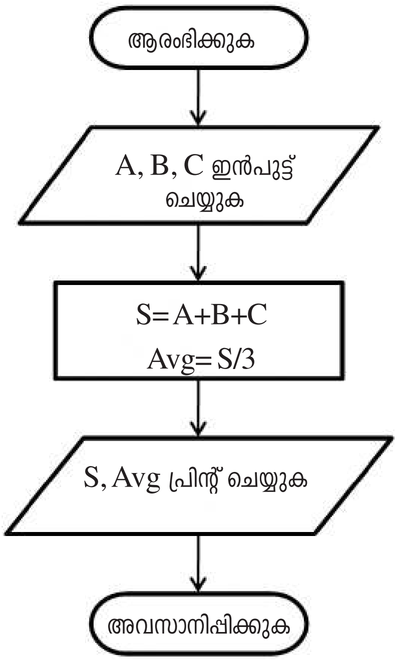
പ്രവർത്തന ക്രിയകളുടെ ഒഴുക്കിനെ സൂചിപ്പിക്കാൻ ആരോകളോട് കൂടിയ ഫ്ളോ ലൈനുകൾ ഉപയോഗിക്കുന്നു. അതായത് നിർദേശങ്ങൾ പ്രാവർത്തികമാക്കേണ്ട കൃത്യമായ ക്രമത്തെ ഇവ സൂചിപ്പിക്കുന്നു. സാധാരണഗതിയിൽ ഫ്ളോ ലൈനുകളുടെ ദിശ മുകളിൽ നിന്നും താഴേക്കും ഇടത്തുനിന്നും വലത്തേക്കും ആയിരിക്കും. എന്നാൽ ചില സാഹചര്യങ്ങളിൽ ഇത് വലതു നിന്ന് ഇടത്തേക്കും താഴെനിന്നും മുകളിലേക്കും ആവാം. ഫ്ളോ ലൈനുകൾ

പരസ്പരം ഛേദിക്കുന്നത് നല്ല പ്രവണതയല്ല. അത്തരം ഛേദനങ്ങൾ പരമാവധി ഒഴിവാക്കേണ്ടതാണ്.

6. കണക്ടർ (connector)



ഫ്ളോച്ചാർട്ടുകളുടെ വലിപ്പം കൂടുമ്പോൾ ഫ്ളോ ലൈനുകൾ പല ഭാഗങ്ങളിലും പരസ്പരം ഛേദിക്കുകയും പല സ്ഥലങ്ങളിലും ആശയക്കുഴപ്പം വരുത്തുകയും തൻമൂലം ഫ്ളോച്ചാർട്ടിന്റെ ഗ്രഹണം ബുദ്ധിമുട്ടാക്കുകയും ചെയ്യുന്നു. അതുപോലെ ഒരു പേജിൽ ഒതുങ്ങുന്നതിനെക്കാൾ വലിയ ഫ്ളോച്ചാർട്ട് ആകുമ്പോൾ ഫ്ളോ ലൈനുകളുടെ ഉപയോഗം അസാധ്യമായിത്തീരുന്നു. ഒരു ഫ്ളോച്ചാർട്ട് സങ്കീർണമാകുകയും ഫ്ളോ ലൈനുകളുടെ എണ്ണം, ദിശ എന്നിവയെക്കുറിച്ച് ആശയക്കുഴപ്പം സംഭവിക്കുകയോ അല്ലെങ്കിൽ ഫ്ളോച്ചാർട്ട് ഒന്നിൽ അധികം പേജുകളിലായി പടരുകയോ ചെയ്താൽ മുറിഞ്ഞുപോയ ഫ്ളോ ലൈനുകളെ തമ്മിൽ കൂട്ടിയോജിപ്പിക്കാൻ ഒരു ജോടി കണക്റ്റർ ചിഹ്നങ്ങൾ ഉപയോഗിക്കാം. ഫ്ളോച്ചാർട്ടിന്റെ ഒരു ഭാഗത്ത് നിന്നുമുള്ള ആഗമനത്തെ അല്ലെങ്കിൽ അതിന്റെ മറ്റൊരു ഭാഗത്തേക്കുള്ള നിർഗമനത്തെ ഈ ചിഹ്നം സൂചിപ്പിക്കുന്നു. ഒരു അക്കം അല്ലെങ്കിൽ അക്ഷരത്തോട് കൂടിയ ഒരു വൃത്തം ഉപയോഗിച്ചാണ് കണക്റ്റർ ചിഹ്നം സൂചിപ്പിക്കുന്നത്. ഒരേപോലെ അടയാളപ്പെടുത്തിയ ഒരു ജോഡി കണക്റ്റർ ചിഹ്നങ്ങൾ അൽഗോരിതത്തിന്റെ തുടർച്ചയെ സൂചിപ്പിക്കുന്നു. ഒരു വലിയ ഫ്ളോ ലൈനിന് പകരം ഒരേ ചിഹ്നങ്ങൾ / അക്കങ്ങൾ ഉള്ള രണ്ടു കണക്റ്ററുകൾ ഉപയോഗിക്കാം. അതായത് ഒരേപോലെ അടയാളപ്പെടുത്തിയ ഒരു ജോടി കണക്ടറുകളിൽ ഒന്ന് ഫ്ളോച്ചാർട്ടിന്റെ മറ്റൊരു ഭാഗത്തേക്കുള്ള നിർഗമനത്തെയും, രണ്ടാമത്തേത് ഫ്ളോച്ചാർട്ടിന്റെ മറ്റൊരു ഭാഗത്തേക്കുള്ള ആഗമനത്തെയും സൂചിപ്പിക്കുന്നു.



ചിത്രം 4.5 ആകെതുകയും ശരാശരിയും കാണാനുള്ള ഫ്ളോച്ചാർട്ട്

ഉദാഹരണം 4.1 ൽ വിശദീകരിച്ച പ്രശ്നത്തിന്റെ ഫ്ളോച്ചാർട്ട് ചിത്രം 4.5 ൽ കാണിച്ചിരിക്കുന്നു.

അൽഗോരിതത്തിലെ ഓരോ ഘട്ടത്തിലെയും നിർദേശങ്ങൾ സൂചിപ്പിക്കാൻ ഉചിതമായ ചിഹ്നങ്ങൾ ഉപയോഗിച്ചിരിക്കുന്നു.മാത്രമല്ല ഓരോ ചിഹ്നത്തിലും അതിനനുസൃതമായ നിർദേശങ്ങൾ രേഖപ്പെടുത്തുന്നു. പ്രവർത്തനങ്ങളുടെ ക്രമം ഫ്ളോ ലൈനുകൾ ഉപയോഗിച്ച് കൃത്യമായി അടയാളപ്പെടുത്തുകയും ചെയ്യുന്നു.

ഫ്ളോച്ചാർട്ടിന്റെ ഗുണങ്ങൾ

പ്രോഗ്രാം ആസൂത്രണത്തിൽ പലരീതികളിലും ഫ്ളോച്ചാർട്ടുകൾ ഗുണപ്രദങ്ങളാണ്.

- **മികച്ച ആശയവിനിമയം :** ഫ്ളോച്ചാർട്ട് ഒരു പ്രോഗ്രാമിന്റെ ചിത്രരൂപത്തിലുള്ള സൂചകം, ആയതിനാൽ ഒരു പ്രോഗ്രാമർക്ക് മറ്റൊരു പ്രോഗ്രാമറിന് പ്രോഗ്രാമിന്റെ യുക്തി ഫ്ളോച്ചാർട്ട് ഉപയോഗിച്ച് വിശദീകരിച്ചു കൊടുക്കുന്നത് പ്രോഗ്രാം വിശദീരിക്കുന്നതിനേക്കാൾ എളുപ്പമാണ്.
- **ഫലപ്രദമായ വിശകലനം :** പ്രോഗ്രാമിലെ വിവിധ ഘട്ടങ്ങൾ കൃത്യമായി ഫ്ളോച്ചാർട്ടിൽ പ്രതിപാദിച്ചിരിക്കുന്നതിനാൽ, പ്രോഗ്രാമിനെ ഫലപ്രദമായി വിശകലനം ചെയ്യാൻ ഫ്ളോച്ചാർട്ട് സഹായിക്കുന്നു.
- **ഫലപ്രദമായ സമന്വയം :** പ്രോഗ്രാമിനെ വിവിധ ഘടകങ്ങളായി തിരിക്കുകയും അവ ഓരോന്നിന്റെയും പരിഹാരം ഫ്ളോച്ചാർട്ടുകളായി പ്രത്യേകം പ്രത്യേകമായി തയാറാക്കുകയും ചെയ്താൽ, അവയെ എല്ലാം കൂട്ടി യോജിപ്പിച്ചു മൊത്തത്തിലുള്ള സിസ്റ്റത്തിന്റെ രൂപരേഖ നമുക്ക് തയാറാക്കാവുന്നതാണ്.
- **ഫലപ്രദമായ കോഡിങ് :** ഫ്ളോച്ചാർട്ട് തയ്യാറാക്കി കഴിഞ്ഞാൽ പ്രോഗ്രാമർക്കു അനുബന്ധ പ്രോഗ്രാം തയ്യാറാക്കാൻ എളുപ്പമാണ്, എന്തെന്നാൽ ഫ്ളോച്ചാർട്ട് പ്രോഗ്രാമിന്റെ ഒരു രേഖാചിത്രമായി പ്രവർത്തിക്കുന്നു. പ്രോഗ്രാമിന്റെ തുടക്കം മുതലുള്ള എല്ലാ ഘട്ടങ്ങളിലൂടെയും കടന്നു പോയി, ഒന്ന് പോലും വിട്ടുപോകാതെ അവസാനം വരെയും എത്തിച്ചേരുവാനുള്ള ഒരു സഹായിയായി ഇത് വർത്തിക്കുന്നു.

ഫ്ളോച്ചാർട്ടിന്റെ പരിമിതികൾ

ഫ്ളോച്ചാർട്ടുകൾക്ക് ഇത്തരത്തിലുള്ള ഗുണങ്ങൾ എടുത്തു പറയാമെങ്കിലും ,ചില പരിമിതികളും അവയ്ക്കുണ്ട്.

- ഉചിതമായ ചിഹ്നങ്ങളും സ്പേസും നൽകിയുള്ള ഫ്ളോച്ചാർട്ട് നിർമാണം സമയം ചെലവഴിച്ചു ചെയ്യേണ്ടതും കഠിനാധ്വാനം ആവശ്യമായതുമാണ്, പ്രത്യേകിച്ചും സങ്കീർണ്ണമായ അൽഗോരിതങ്ങൾ ആണെങ്കിൽ.
- അൽഗോരിതത്തിന്റെ യുക്തിയിലുള്ള വളരെച്ചെറിയ മാറ്റത്തിനുപോലും പുതിയ ഫ്ളോച്ചാർട്ട് ആവശ്യമായി വരുന്നു.
- ഫ്ളോച്ചാർട്ടിൽ ഉൾപ്പെടുത്തേണ്ട വിശദശാംശങ്ങളെ പറ്റി വിശദീകരിക്കുന്ന ഒരു തരത്തിലുള്ള മാനദണ്ഡങ്ങളും നിലവിലില്ല.

വിവിധ പ്രശ്നങ്ങൾ പരിഹരിക്കുന്നതിനുള്ള അൽഗോരിതങ്ങളും ഫ്ളോച്ചാർട്ടുകളും നമുക്ക് വികസിപ്പിക്കാം.

ഉദാഹരണം 3.2: ഒരു ചതുരത്തിന്റെ വിസ്തീർണവും ചുറ്റളവും കണ്ടെത്തുക

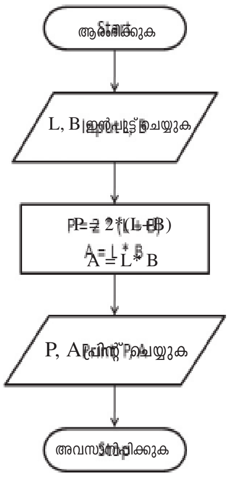
ചതുരത്തിന്റെ നീളവും വീതിയും ലഭ്യമായാൽ ഈ പ്രശ്നം നമുക്കു പരിഹരിക്കാം. താഴെ പറയുന്ന സമവാക്യം ഇതിന് വേണ്ടി ഉപയോഗിക്കാവുന്നതാണ്.

ചുറ്റളവ് = 2 x (നീളം + വീതി), വിസ്തീർണം = (നീളം x വീതി).

L, B എന്നീ വേരിയബിളുകൾ നീളം വീതി എന്നി സൂചിപ്പിക്കാനും P, A എന്നീ വേരിയബിളുകൾ വിസ്തീർണം, ചുറ്റളവ് എന്നിവ സൂചിപ്പിക്കാനും ഉപയോഗിക്കുന്നു എന്നിരിക്കട്ടെ

```
ഘട്ടം 1 :   തുടങ്ങുക
ഘട്ടം 2 :   L , B ഇൻപുട്ടായി സ്വീകരിക്കുക
```

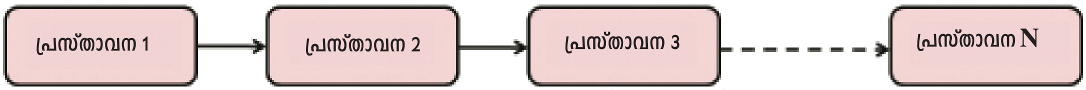
```
ഘട്ടം 3 :   $P = 2 * (L + B)$
ഘട്ടം 4 :   $A = L * B$
ഘട്ടം 5 :   P ,A  പ്രിൻറ് ചെയ്യുക.
ഘട്ടം 6.    അവസാനിപ്പിക്കുക.
```



ചിത്രം 3.6 വിസ്തീർണം ചുറ്റളവ് കാണാനുള്ള ഫ്ളോച്ചാർട്ട്

ചിത്രം 3 .6 ൽ ഇതിനുള്ള ഫ്ളോച്ചാർട്ട് നൽകിയിരിക്കുന്നു.

ഉദാഹരണം 4.1 ലും 4.2 ലും വികസിപ്പിച്ചെടുത്തിരിക്കുന്ന അൽഗോരിതങ്ങൾക്ക് ഓരോന്നിലും ആറു വീതം നിർദേശങ്ങളാണുള്ളത്. ചിത്രം 3.7 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ ഈ രണ്ടു സന്ദർഭങ്ങളിലും നിർദേശങ്ങൾ ഓരോന്നും അനുക്രമമായ രീതിയിലാണ് പ്രവർത്തിക്കുക. നിർദേശങ്ങളുടെ പ്രവർത്തനക്രമത്തെ നിയന്ത്രണഗതി (Flow of Cotnrol) എന്ന് പറയുന്നു. അപ്രകാരം മുകളിൽപ്പറഞ്ഞ രണ്ട് അൽഗോരിതങ്ങളും അനുക്രമമായ നിയന്ത്രണ ഗതിയാണ് പിന്തുടരുന്നതെന്നു നമുക്ക് പറയാവുന്നതാണ്.



ചിത്രം 3.7 അനുക്രമമായ നിയന്ത്രണ ഗതി

നമുക്കു ചെയ്യാം

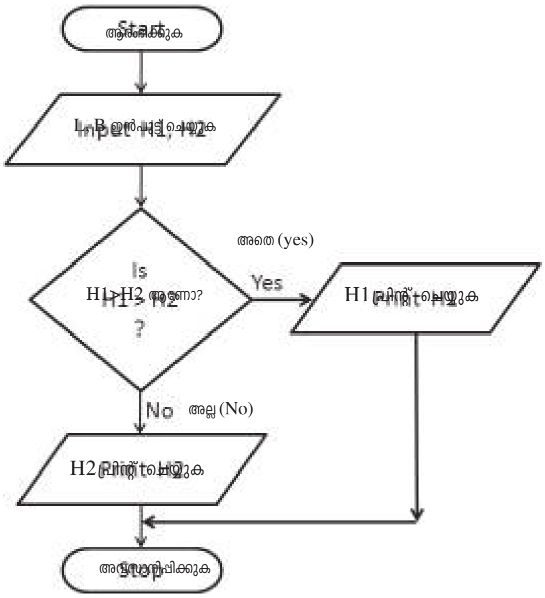
സെക്കന്റുകളായി സമയം ഇൻപുട്ട് ആയി സ്വീകരിച്ച് Hr:min:Sec രൂപത്തിൽ ലഭിക്കുവാനുള്ള അൽഗോരിതവും ഫ്ളോച്ചാർട്ടും വികസിപ്പിക്കുക. (ഉദാഹരണത്തിന് 3700 എന്ന് ഇൻപുട്ട് നൽകിയാൽ 1 Hr : 1 Min : 40 sec എന്ന ഔട്ട്പുട്ട് ലഭിക്കണം)

ഉദാഹരണം 3.3 രണ്ടു വിദ്യാർഥികളിൽ ഉയരം കൂടിയ ആളുടെ ഉയരം കണ്ടെത്തുക

ഇവിടെ,രണ്ടു വിദ്യാർത്ഥികളുടെ ഉയരത്തെ പ്രതിനിധീകരിക്കുന്ന രണ്ടു സംഖ്യകൾ ഇൻപുട്ട് ആയി സ്വീകരിക്കേണ്ടതാണ്. അവയിലെ വലിയ സംഖ്യയാണ് ഉത്തരമായിട്ടു പരിഗണിക്കുക. ഇതിനായി ഈ സംഖ്യകളെ താരതമ്യം ചെയ്യേണ്ടതുണ്ടെന്നു നമുക്കറിയാവുന്നതാണ്. അൽഗോരിതം താഴെ നൽകിയിരിക്കുന്നു.

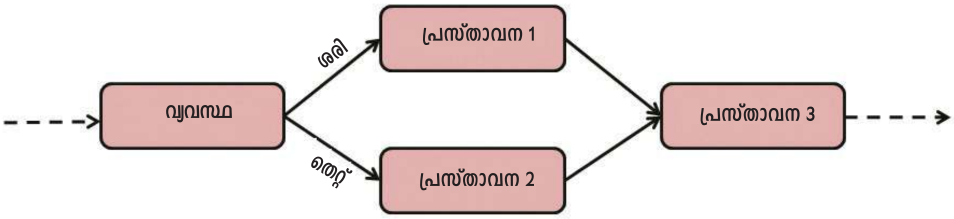
```
ഘട്ടം 1 :   തുടങ്ങുക
ഘട്ടം 2 :   H1 ,H2  ഇൻപുട്ട് ആയി സ്വീകരിക്കുക
ഘട്ടം 3 :   അഥവാ H1 >H2 ആണെങ്കിൽ
ഘട്ടം 4 :   H1 പ്രിന്റ് ചെയ്യുക
ഘട്ടം 5 :   അല്ലെങ്കിൽ
ഘട്ടം 6 :   H2 പ്രിന്റ് ചെയ്യുക
ഘട്ടം 7 :   പരിശോധന അവസാനിക്കുന്നു
ഘട്ടം 8  :  അവസാനിപ്പിക്കുക
```

ഈ അൽഗോരിതത്തിന്റെ ഫ്ളോച്ചാർട്ട് ചിത്രം 3.8 ൽ നൽകിയിരിക്കുന്നു. തീരുമാനങ്ങൾ കൈകൊള്ളാനുള്ള സങ്കേതം ഉപയോഗപെടുത്തുന്ന ഒരു അൽഗോരിതമാണിത്. ഘട്ടം 3 ൽ ഒരു


Start
ആരംഭിക്കുക
L,B ഇൻപുട്ട് ചെയ്യുക
Input H1, H2
Is H1 > H2 ?
H1>H2 ആണോ?
അതെ (yes)
Yes
H1 പ്രിന്റ് ചെയ്യുക
Print H1
No
അല്ല (No)
H2 പ്രിന്റ് ചെയ്യുക
Print H2
Stop
അവസാനിപ്പിക്കുക

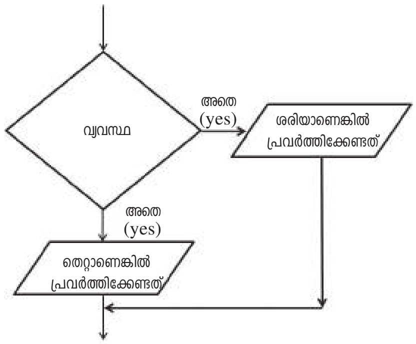
ചിത്രം 3.8 ഉയർന്ന വില കാണാനുള്ള ഫ്ളോച്ചാർട്ട്

നിബന്ധന പരിശോധിക്കുന്നു. H1 ,H2 എന്നിവയുടെ വിലകളുടെ അടിസ്ഥാനത്തിൽ അതിന്റെ ഉത്തരം തീർച്ചയായും ശരി അല്ലെങ്കിൽ തെറ്റ് എന്നായിരിക്കും. തീരുമാനം കൈക്കൊള്ളുന്നത് ഈ നിബന്ധനയുടെ ഉത്തരത്തിനെ അടിസ്ഥാനമായിട്ടായിരിക്കും. ഉത്തരം ശരി എന്നാണെങ്കിൽ ഘട്ടം 4 പ്രവർത്തിക്കും അല്ലെങ്കിൽ ഘട്ടം 6 ആണ് പ്രവർത്തിക്കുക. ഇവിടെ ഒരു നിബന്ധനയുടെ അടിസ്ഥാനത്തിൽ രണ്ടു പ്രസ്താവനകളിൽ ഏതെങ്കിലും ഒന്നാണ് (ഘട്ടം 4 അല്ലെങ്കിൽ ഘട്ടം 6) പ്രവർത്തനത്തിനായി തിരഞ്ഞെടുക്കപ്പെടുക. ഘട്ടം 3 ൽ പ്രോഗ്രാം രണ്ടു ശാഖകളായി പിരിയുന്നു. അതായത് പ്രശ്നം പരിഹരിക്കുന്നതിന് വേണ്ടി ഈ അൽഗോരിതം ഒരു തിരഞ്ഞെടുക്കൽ ഘടന ഉപയോഗപ്പെടുത്തുന്നു. ചിത്രം 3.9 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ നിബന്ധനയുടെ ഉത്തരത്തിനനുസരിച്ചു പ്രവർത്തനത്തിന്റെ ഗതി രണ്ടിൽ ഏതെങ്കിലും ഒരു പ്രസ്താവനയിലേക്ക് വിഘടിച്ചു പോകുന്നു.


വ്യവസ്ഥ
ശരി
തെറ്റ്
പ്രസ്താവന 1
പ്രസ്താവന 2
പ്രസ്താവന 3

ചിത്രം 3.9 തിരഞ്ഞെടുക്കൽ ഘടന

തിരഞ്ഞെടുക്കൽ നിർമ്മിതിയുടെ പ്രവർത്തനം ചിത്രം 3.10 ൽ കൊടുത്തിരിക്കുന്നു. നിയന്ത്രണ ഗതി നിബന്ധനയിലേക്കു വരികയും നിബന്ധന ശരി അല്ലെങ്കിൽ തെറ്റ് എന്ന് വിലയിരുത്തപ്പെടുകയും ചെയ്യുന്നു. നിബന്ധനയുടെ ഫലം ശരി എന്നാണെങ്കിൽ, ശരിയായാൽ പ്രവർത്തിക്കേണ്ട നിർദേശങ്ങളുടെ കൂട്ടം നടപ്പിലാക്കുകയും, തെറ്റായാൽ പ്രവർത്തിക്കേണ്ട നിർദേശങ്ങളുടെ കൂട്ടത്തെ ഒഴിവാക്കുകയും ചെയ്യുന്നു. നിബന്ധനയുടെ ഫലം തെറ്റ് ആണെങ്കിൽ തെറ്റായാൽ പ്രവർത്തിക്കേണ്ട നിർദേശങ്ങളുടെ കൂട്ടം നടപ്പിലാക്കുകയും ശരിയായാൽ പ്രവർത്തിക്കേണ്ട നിർദേശങ്ങളുടെ കൂട്ടാത്തെ ഒഴിവാക്കുകയും ചെയ്യുന്നു. ഇനി നമുക്ക് മറ്റൊരു പ്രശ്നം പരിഹരിക്കാം.


വ്യവസ്ഥ
അതെ (yes)
ശരിയാണെങ്കിൽ പ്രവർത്തിക്കേണ്ടത്
അതെ (yes)
തെറ്റാണെങ്കിൽ പ്രവർത്തിക്കേണ്ടത്

ചിത്രം 3.10 തിരഞ്ഞെടുക്കലിന്റെ ഫ്ളോച്ചാർട്ട്

ഉദാഹരണം 3.4 : 3 യൂണിറ്റ് ടെസ്റ്റുകളിൽ ലഭിച്ച സ്കോറുകൾ ഇൻപുട്ട് ചെയ്ത ഏറ്റവും ഉയർന്ന സ്കോർ കണ്ടെത്തുക

ഇവിടെ സ്കോറുകൾ സൂചിപ്പിക്കുന്നതിനു വേണ്ടി മൂന്നു സംഖ്യകൾ നൽകുകയും അവയിൽ ഏറ്റവും വലിയ സംഖ്യ കണ്ടുപിടിക്കുകയും ചെയ്യുന്നു. ഇതിന്റെ അൽഗോരിതം താഴെ കൊടുത്തിരിക്കുന്നു. ചിത്രം 3.11 ൽ ഫ്ളോച്ചാർട്ട് പ്രദർശിപ്പിച്ചിരിക്കുന്നു.

ഘട്ടം 1 : ആരംഭിക്കുക

ഘട്ടം 2 : M 1, M 2 , M 3 എന്നീ സംഖ്യകൾ ഇൻപുട്ട്

ആരംഭിക്കുക

ചെയ്യുക .

ഘട്ടം 3 : അഥവാ M 1 > M 2 ഉം M 1 > M 3 ആണെങ്കിൽ

ഘട്ടം 4 : M 1 പ്രിന്റ് ചെയ്യുക

ഘട്ടം 5 : അല്ലെങ്കിൽ M 2 >M 3 ആണെങ്കിൽ

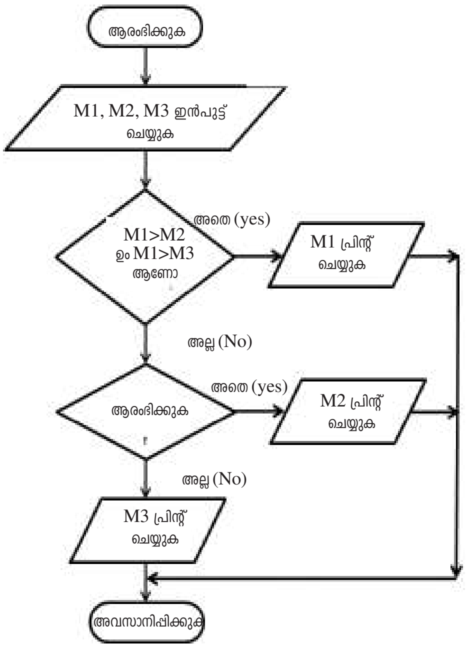
ഘട്ടം 6 : M 2 പ്രിന്റ് ചെയ്യുക

ഘട്ടം 7 : അല്ലെങ്കിൽ

ഘട്ടം 8 : M 3 പ്രിൻറ് ചെയ്യുക.

ഘട്ടം 9 : വ്യവസ്ഥാ പരിശോധനയുടെ അവസാനം

ഘട്ടം 10 : അവസാനിപ്പിക്കുക



ചിത്രം 3.11 മൂന്ന് സംഖ്യകളിൽ ഏറ്റവും വലിയ സംഖ്യ കാണാനുള്ള ഫ്ളോച്ചാർട്ട്

വ്യത്യസ്തമായ നിബന്ധനകളുടെ അടിസ്ഥാനത്തിൽ അൽഗോരിതത്തിൽ ഒന്നിലധികം തിരഞ്ഞെടുക്കൽ നിർമിതകൾ ഉപയോഗിക്കുന്നു. ഇവിടെ വ്യത്യസ്തമായ മൂന്നു പ്രവൃത്തികൾ നൽകിയിരിക്കുന്നു. എന്നാൽ അവയിൽ ഒന്ന് മാത്രമേ പ്രവർത്തികമാകുകയുള്ളൂ. ശ്രദ്ധിക്കേണ്ട മറ്റൊരു വസ്തുത, ഇവിടെ ആദ്യത്തെ നിബന്ധനയിൽ രണ്ടു താരതമ്യങ്ങൾ അടങ്ങിയിരിക്കുന്നു എന്നതാണ്. ഇത്തരം നിബന്ധനകൾ സംയുക്ത നിബന്ധനകൾ (Compound conditions) എന്ന് പറയുന്നു.

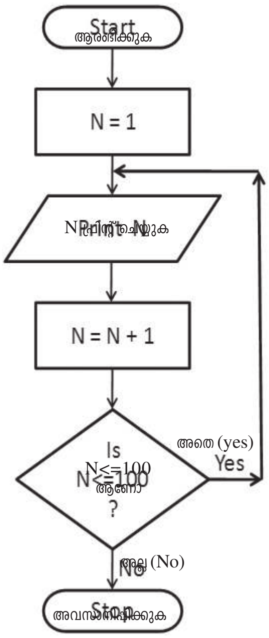
നമുക്കു ചെയ്യാം

1. *തന്നിരിക്കുന്ന സംഖ്യ ഒറ്റയാണോ ഇരട്ടയാണോ എന്ന് പരിശോധിക്കാനുള്ള അൽഗോരിതം തയാറാക്കുക. ഫ്ളോച്ചാർട്ട് വരയ്ക്കുക.*
2. *ദിവസത്തെ സൂചിപ്പിക്കുന്ന സംഖ്യ ഇൻപുട്ട് ആയി നൽകിയാൽ ദിവസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കാനുള്ള അൽഗോരിതവും ഫ്ളോച്ചാർട്ടും തയാറാക്കുക. (ഉദാഹരണത്തിന് 1 ഇൻപുട്ട് നൽകിയാൽ ഔട്ട്പുട്ട് Sunday എന്നായിരിക്കണം. അഥവാ 2 ആണ് ഇൻപുട്ടെങ്കിൽ ഔട്ട്പുട്ട് Monday എന്നായിരിക്കണം. 1 മുതൽ 7 വരെയുള്ള സംഖ്യ അല്ല ഇൻപുട്ടെങ്കിൽ 'INVALID DATA ' എന്നായിരിക്കേണ്ടതാണ്.*
3. *പത്താം തരത്തിലെ മൂല്യനിർണയ വ്യവസ്ഥയുടെ അടിസ്ഥാനത്തിൽ ഒരു സ്കോർ (പരമാവധി 100) സ്വീകരിച്ചു ഗ്രേഡ് കാണാനുള്ള അൽഗോരിതം തയാറാക്കുക.*

ഒരു പ്രവൃത്തി തന്നെ ആവർത്തിച്ചു നിർവഹിക്കേണ്ട ഒരു സാഹചര്യം പരിഗണിക്കുക. ഉദാഹരണത്തിന് ആദ്യത്തെ 100 എണ്ണൽ സംഖ്യകൾ പ്രിൻറ് ചെയ്യണമെന്നിരിക്കട്ടെ എങ്ങനെയാണ് നമുക്കത് ചെയ്യാൻ സാധിക്കുക? നമുക്കറിയാം ആദ്യത്തെ സംഖ്യ 1 ആണ് .അത് പ്രിന്റ് ചെയ്യേണ്ടതാണ്.ആദ്യത്തെ സംഖ്യയോട് 1 കൂട്ടിയാൽ അടുത്ത സംഖ്യ ലഭിക്കുന്നു. അതും പ്രിന്റ് ചെയ്യണം. ഇതിൽ നിന്ന് ഒരു കാര്യം വ്യക്തമാണ്, സംഖ്യ പ്രിൻറ് ചെയ്യുക സംഖ്യയോടു 1 കൂട്ടുക എന്നീ പ്രവൃത്തികൾ ആവർത്തിച്ച് ചെയ്യേണ്ടവയാണ്. അവസാനത്തെ സംഖ്യ പ്രിൻറ് ചെയ്തു കഴിഞ്ഞാൽ പ്രവർത്തനം അവസാനിപ്പിക്കേണ്ടതാണ് .ഇതിനു വേണ്ടിയുള്ള ഒരു അൽഗോരിതം നമുക്ക് തയാറാക്കാം.

ഉദാഹരണം 3.5: 1 മുതൽ 100 വരെയുള്ള സംഖ്യകൾ പ്രിന്റ് ചെയ്യാൻ

```
ഘട്ടം 1 :   ആരംഭിക്കുക
ഘട്ടം 2 :   N =1
ഘട്ടം 3 :   N പ്രിന്റ് ചെയ്യുക
ഘട്ടം 4 :   N =N + 1
ഘട്ടം 5 :   അഥവാ N <=100 ആണെങ്കിൽ
            ഘട്ടം 3 ലേക്ക് പോകുക
ഘട്ടം 6 :   അവസാനിപ്പിക്കുക
```



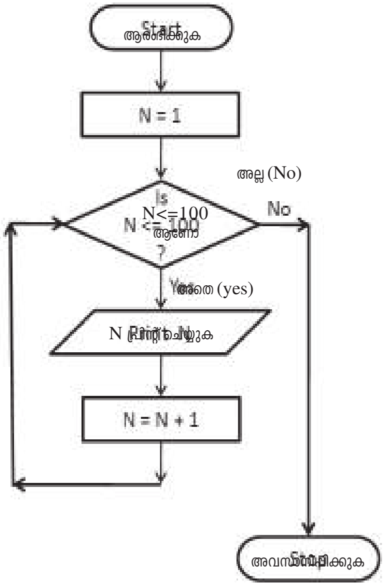
ചിത്രം 3.12 : 1 മുതൽ 100 വരെയുള്ള സംഖ്യകൾ പ്രിന്റ് ചെയ്യാനുള്ള ഫ്ളോച്ചാർട്ട്

ഉദാഹരണം 3.5ൽ കൊടുത്തിരിക്കുന്ന അൽഗോരിതത്തിൽ ഘട്ടം 5 ൽ ഒരു വ്യവസ്ഥ പരിശോധിക്കുന്നു. അഥവാ വ്യവസ്ഥ ശരിയാണെങ്കിൽ നിയന്ത്രണ ഗതി ഘട്ടം 3 ലേക്ക് തിരിച്ചു പോകുന്നു. അതു കാരണം വ്യവസ്ഥ ശരിയായിരിക്കുന്നതു വരെ ഘട്ടം 3, ഘട്ടം 4, ഘട്ടം 5 എന്നിവ ആവർത്തിച്ചു പ്രവർത്തിച്ചു കൊണ്ടിരിക്കും. ഇവിടെ ഒരു ലൂപ്പ് രൂപം കൊണ്ടതായി നമുക്ക് പറയാം. ഘട്ടം 3,4,5 എന്നിവ ചേർന്നതാണ് ആ ലൂപ്പ്. വ്യവസ്ഥ തെറ്റാകുമ്പോൾ മാത്രമെ നിയന്ത്രണം ലൂപ്പിനു പുറത്തേക്കു വരുകയുള്ളൂ. ഈ അൽഗോരിതത്തിന്റെ ഫ്ളോച്ചാർട്ട് ചിത്രം 3.12 ൽ കാണിച്ചിരിക്കുന്നു.

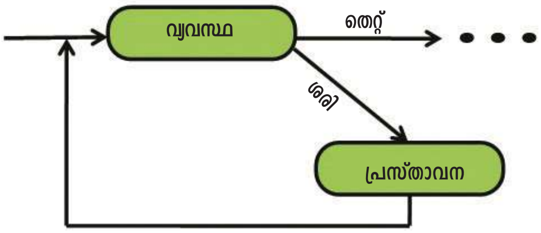
മുകളിൽപ്പറഞ്ഞ അൽഗോരിതത്തെ താഴെ പറയും പ്രകാരം ലഘൂകരിക്കാം

```
ഘട്ടം 1 :   ആരംഭിക്കുക
ഘട്ടം 2 :   N = 1
ഘട്ടം 3 :   N <100 ആയിരിക്കുന്നത് വരെ ഘട്ടം 4 ഉം 5 ഉം ആവർത്തിക്കുക
ഘട്ടം 4 :   N പ്രിന്റ് ചെയ്യുക
ഘട്ടം 5 :   N = N + 1
ഘട്ടം 6 :   അവസാനിപ്പിക്കുക
```

ഘട്ടം 3 ൽ 'ആവർത്തിക്കുക', 'ആയിരിക്കുന്നത് വരെ' മുതലായ വാക്കുകൾ ലൂപ്പ് നിർമ്മിക്കാൻ ഉപയോഗിക്കുന്നു. ആവർത്തിച്ചു പ്രവർത്തിക്കേണ്ട പ്രസ്താവനകൾ 'ആവർത്തിക്കുക' എന്ന വാക്കിന്റെ കൂടെ പ്രസ്താവിക്കുകയും പരിശോധിക്കേണ്ട വ്യവസ്ഥ 'ആയിരിക്കുന്നത് വരെ' എന്ന വാക്കിന്റെ കൂടെയും നൽകുന്നു. അൽഗോരിതം വ്യത്യസ്തമായിരിക്കുന്നത് പോലെ ചിത്രം 3.13 ൽ കാണിച്ചിരിക്കുന്ന ഫ്ളോച്ചാർട്ടും അല്പം വ്യത്യസ്തമായിരിക്കും. ചിത്രം 3.14 ൽ ലൂപ്പിന്റെ പ്രവർത്തന ശൈലി കാണിച്ചിരിക്കുന്നു.

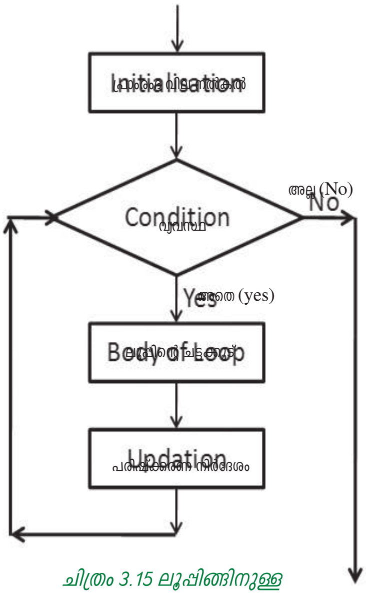

Start
ആരംഭിക്കുക
N = 1
Is N <= 100 ?
N<=100 ആണോ
അല്ല (No)
No
Yes
അതെ (yes)
Print N
N പ്രിന്റ് ചെയ്യുക
N = N + 1
Stop
അവസാനിപ്പിക്കുക

ചിത്രം 3.13: 1 മുതൽ 100 വരെയുള്ള സംഖ്യകൾ പ്രിന്റ് ചെയ്യാനുള്ള ഫ്ളോച്ചാർട്ട്


വ്യവസ്ഥ
തെറ്റ്
ശരി
പ്രസ്താവന

ചിത്രം 3 .14: ലൂപ്പിംഗ് നിർമിതി

ഒരു ലൂപ്പിനു നാല് ഘടകങ്ങൾ ഉണ്ട്. സ്വാഭാവികമായും അവയിൽ ഒന്ന് വ്യവസ്ഥ (Condition) തന്നെ. വ്യവസ്ഥ നൽകുന്നതിനു വേണ്ടി ഒരു വേരിയബിളെങ്കിലും ഉപയോഗിക്കണം എന്നു നമുക്കറിയാം. ഇതിനെ ലൂപ്പ് നിയന്ത്രണ വേരിയബിൾ എന്ന് വിളിക്കാം. വ്യവസ്ഥ പരിശോധിക്കുന്നതിന് മുമ്പ് ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന് ഒരു വില ലഭ്യമാക്കേണ്ടതാണ്. ഇൻപുട്ട് അല്ലെങ്കിൽ വില നൽകൽ (Assignment) വഴി ഇത് സാധ്യമാകുന്നതാണ്. അത്തരത്തിലുള്ള നിർദേശങ്ങളെ ലൂപ്പിന്റെ പ്രാരംഭ വില നൽകൽ നിർദേശങ്ങൾ (Initialization Instructions) എന്ന് പറയുന്നു. പരിഷ്കരണ നിർദേശം (Update Instruction) എന്ന മൂന്നാമത്തെ ഘടകമാണ് ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വില മാറ്റുന്നത്. ഇത് വളരെ അത്യാവശ്യമാണ്. എന്തെന്നാൽ പരിഷ്കരണ നിർദേശം ഇല്ലെങ്കിൽ ലൂപ്പിന്റെ പ്രവർത്തനം ഒരിക്കലും അവസാനിക്കില്ല. ആവർത്തിച്ചു പ്രവർത്തിക്കേണ്ട ഒരു കൂട്ടം നിർദേശങ്ങളാണ് ലൂപ്പിന്റെ ചട്ടക്കൂട് (Body of the Loop) എന്ന നാലാമത്തെ ഘടകം. ചിത്രം 3.15 ൽ കാണിച്ചിരിക്കുന്ന ഫ്ളോച്ചാർട്ട് ലൂപ്പിങ് ഘടന വ്യക്തമാക്കുന്നു.


Initialisation
പ്രാരംഭ വില നൽകൽ
Condition
വ്യവസ്ഥ
അല്ല (No)
No
Yes
അതെ (yes)
Body of Loop
ലൂപ്പിന്റെ ചട്ടക്കൂട്
Updation
പരിഷ്കരണ നിർദേശം

ചിത്രം 3.15 ലൂപ്പിങ്ങിനുള്ള ഫ്ളോച്ചാർട്ട്

പ്രരാരംഭ വില നൽകൽ നിർദേശമാണ് ആദ്യം പ്രവർത്തിക്കുക. ശേഷം വ്യവസ്ഥ പരിശോധിക്കുന്നു. വ്യവസ്ഥ ശരിയാണെങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂടും തുടർന്ന് പരിഷ്കരണ നിർദേശവും പ്രവർത്തിക്കുന്നു. പരിഷ്കരണ നിർദേശം പ്രവർത്തിച്ചു കഴിഞ്ഞാൽ വീണ്ടും വ്യവസ്ഥ പരിശോധിക്കുന്നു. വ്യവസ്ഥ തെറ്റാകുന്നത് വരെ ഈ പ്രക്രിയ തുടരുന്നു. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രാവർത്തികമാകുന്നതിനു മുമ്പ് വ്യവസ്ഥ പരിശോധിക്കുന്ന ലൂപ്പിനെ ആഗമന നിയന്ത്രിത ലൂപ്പ് (entry cotnrolled Loop) എന്ന് വിളിക്കുന്നു. മറ്റൊരു രീതിയിലുള്ള ലൂപ്പിംഗ് നിർമ്മിതിയും നിലവിലുണ്ട്. അതിൽ ലൂപ്പിന്റെ ചട്ടക്കൂടും പരിഷ്കരണ നിർദേശവും പ്രവർത്തിച്ചു കഴിഞ്ഞ ശേഷം മാത്രമേ വ്യവസ്ഥ പരിശോധിക്കുകയുള്ളു. ഇത്തരത്തിലുള്ള ലൂപ്പിനെ നിർഗമന നിയന്ത്രിത ലൂപ്പ് (Exit Cotnrolled Loop) എന്ന് വിളിക്കുന്നു.

ഉദാഹരണം 3.6: ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ തുക കണ്ടെത്തുക

ഇവിടെ N ന്റെ വില നമ്മൾ ഇൻപുട്ട് ആയി നൽകുന്നു. 1 മുതൽ N വരെയുള്ള സംഖ്യകളുടെ തുക കണ്ടുപിടിക്കേണ്ടതാണ്. തുക സംഭരിക്കുന്നതിനായി 'S' എന്ന വേരിയബിളാണെന്നിരിക്കട്ടെ, ചിത്രം 3.16 ൽ ഈ അൽഗോരിതത്തിൻറെ ഫ്ളോച്ചാർട്ട് കാണിച്ചിരിക്കുന്നു.

ഘട്ടം 1 : തുടങ്ങുക

ഘട്ടം 2 : N ലേക്ക് വില ഇൻപുട്ട് ആയി സ്വീകരിക്കുക

ഘട്ടം 3 : A=1 ,S=0

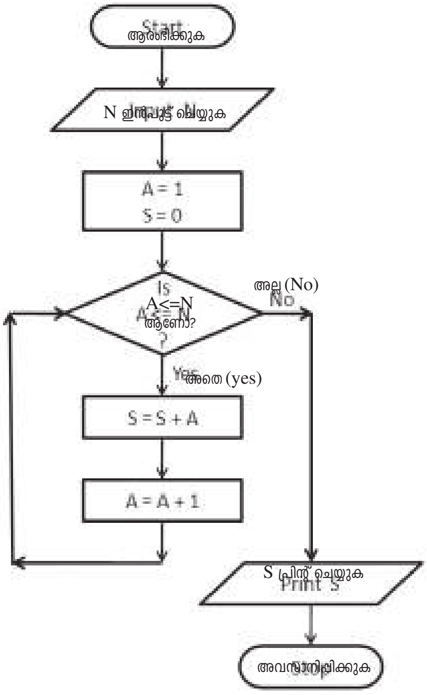
ഘട്ടം 4 : (A<=N) ആയിരിക്കുന്നത് വരെ ഘട്ടം 5 ഉം 6 ഉം ആവർത്തിക്കുക

ഘട്ടം 5 : S =S +A

ഘട്ടം 6 : A =A +1

ഘട്ടം 7 : S പ്രിന്റ് ചെയ്യുക.

ഘട്ടം 8 : അവസാനിപ്പിക്കുക



ചിത്രം 3.16 ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ ആകെത്തുക കാണാനുള്ള ഫ്ളോച്ചാർട്ട്

ഈ അൽഗോരിതം ഒരു ആഗമന നിയന്ത്രിത ലൂപ്പ് ഉപയോഗിക്കുന്നു. അടുത്ത ഉദാഹരണത്തിൽ പ്രശ്ന പരിഹാരത്തിനായി ഒരു നിർഗമന നിയന്ത്രിത ലൂപ്പ് ഉപയോഗിക്കുന്ന അൽഗോരിതം കാണാം.

ഉദാഹരണം 3.7 : തന്നിരിക്കുന്ന സംഖ്യയുടെ ആദ്യത്തെ 10 ഗുണിതങ്ങൾ കണ്ടെത്തുക

ഘട്ടം 1 : തുടങ്ങുക

ഘട്ടം 2 : N ഇൻപുട്ട് ആയി സ്വീകരിക്കുക

ഘട്ടം 3 : A=1

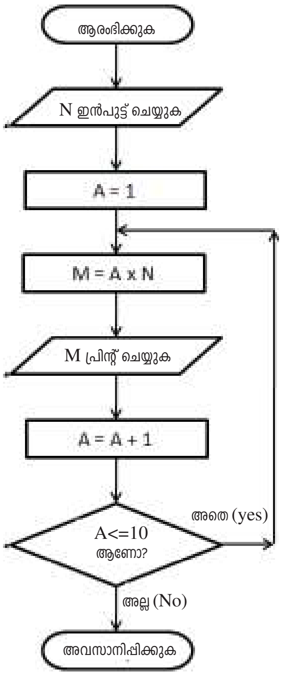
ഘട്ടം 4 : M =A x N

ഘട്ടം 5 : M പ്രിന്റ് ചെയ്യുക

ഘട്ടം 6 : A=A+1

ഘട്ടം 7 : (A<=10) ആയിരിക്കുന്നത് വരെ ഘട്ടം 4 മുതൽ 5 വരെ ആവർത്തിക്കുക

ഘട്ടം 8 : അവസാനിപ്പിക്കുക



ചിത്രം 3.17 തന്നിരിക്കുന്ന സംഖ്യയുടെ ആദ്യത്തെ 10 ഗുണിതങ്ങൾ കണ്ടുപിടിക്കാനുള്ള ഫ്ളോച്ചാർട്ട്

ഈ അൽഗോരിതത്തിന് അനുസൃതമായ ഫ്ളോച്ചാർട്ട് ചിത്രം 3.17 ൽ നൽകിയിരിക്കുന്നു. ഇവിടെ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിച്ച ശേഷം മാത്രം പരിശോധിക്കപ്പെടുന്ന ഒരു വ്യവസ്ഥ അടങ്ങിയ ഒരു ലൂപ്പ് ഉപയോഗിച്ചിരിക്കുന്നു. ആഗമന നിയന്ത്രിത ലൂപ്പും നിർഗമന നിയന്ത്രിത ലൂപ്പും തമ്മിലുള്ള താരതമ്യം പട്ടിക 3.1 ൽ കാണിച്ചിരിക്കുന്നു.

ആഗമന നിയന്ത്രിത ലൂപ്	നിർഗമന നിയന്ത്രിത ലൂപ്
• ലൂപ് ചട്ടക്കൂട് പ്രാവർത്തികമാക്കുന്നതിനു മുമ്പ് വ്യവസ്ഥ പരിശോധിക്കുന്നു • ലൂപ് ചട്ടക്കൂട് ഒരിക്കൽ പോലും പ്രവർത്തിച്ചില്ല എന്ന് വരാം • ലൂപ് ചട്ടക്കൂട് പ്രാവർത്തികമാക്കുന്നതിൽ നിന്നും ഒഴിവാക്കണമെങ്കിൽ ഇത് ഉപയോഗിക്കുന്നു.	• ലൂപ് ചട്ടക്കൂട് പ്രാവർത്തികമാക്കിയതിന് ശേഷം വ്യവസ്ഥ പരിശോധിക്കുന്നു • ലൂപ് ചട്ടക്കൂട് കുറഞ്ഞത് ഒരു തവണ എങ്കിലും നിർബന്ധമായും പ്രവർത്തിച്ചിരിക്കും • ചട്ടക്കൂടിന്റെ സാധാരണ നിലയിലുള്ള പ്രവർത്തനം ഉറപ്പുവരുത്തണമെങ്കിൽ ഉപയോഗിക്കുന്നു

പട്ടിക 3.1: ലൂപുകളുടെ താരതമ്യം

പഠന പ്രവർത്തനങ്ങളുടെ ഭാഗമായി നൽകിയിരിക്കുന്ന പ്രശ്നപരിഹാരത്തിനുള്ള അൽഗോരിതങ്ങളും ഫ്ളോച്ചാർട്ടും ഉപയോഗിച്ച് നമുക്ക് പരിശീലിക്കാം

നമുക്കു ചെയ്യാം

താഴെ പറയുന്ന പ്രശ്നങ്ങൾക്കു വേണ്ടി അൽഗോരിതവും ഫ്ളോച്ചാർട്ടും തയാറാക്കുക

1. 100 ൽ താഴെയുള്ള എല്ലാ ഇരട്ട സംഖ്യകളും അവരോഹണ ക്രമത്തിൽ പ്രിന്റ് ചെയ്യുക
2. 100 നും 200 നും ഇടയിലുള്ള ഒറ്റ സംഖ്യകളുടെ തുക കണ്ടുപിടിക്കുക
3. തന്നിരിക്കുന്ന സംഖ്യയുടെ ഗുണനപ്പട്ടിക പ്രിൻറ് ചെയ്യുക
4. ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ (FACTORIAL) കണ്ടുപിടിക്കുക
5. ഒരു സംഖ്യ ഇൻപുട്ട് ചെയ്ത് അവിഭാജ്യ സംഖ്യയാണോ എന്ന് പരിശോധിക്കുക

3.3.3 പ്രോഗ്രാം കോഡ് തയാറാക്കൽ (Coding the Program)

അൽഗോരിതവും ഫ്ളോച്ചാർട്ടും രൂപകൽപന ചെയ്തു കഴിഞ്ഞാൽ പ്രോഗ്രാമിങ്ങിന്റെ അടുത്ത പടി നിർദേശങ്ങൾ സൂക്ഷ്മവും സംക്ഷിപ്തവുമായ പ്രതീകങ്ങൾ ഉപയോഗിച്ച് ആവിഷ്കരിക്കുക എന്നതാണ്. അതായത് നിർദേശങ്ങൾ പ്രോഗ്രാമിങ് ഭാഷയിൽ ആവിഷ്കരിക്കുന്നു. പ്രശ്ന പരിഹാരത്തിന് വേണ്ടി ഇത്തരം പ്രോഗ്രാം നിർദേശങ്ങൾ എഴുതുന്നതിനെയാണ് കോഡിങ് എന്ന് പറയുന്നത്. കമ്പ്യൂട്ടറിൽ കോഡ് എഴുതുന്നതിനായി ടെക്സ്റ്റ് എഡിറ്റർ പ്രോഗ്രാമുകൾ ലഭ്യമാണ്.

ആശയവിനിമയത്തിനുള്ള ഒരു സംവിധാനമാണ് ഭാഷ. ഇംഗ്ലീഷ് മലയാളം മുതലായ സ്വാഭാവിക ഭാഷകൾ ഉപയോഗിച്ച് നാം നമ്മുടെ ആശയങ്ങളും വികാരങ്ങളും പരസ്പരം പങ്കുവയ്ക്കുന്നു .

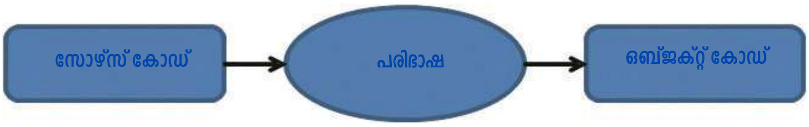
അത് പോലെ ഉപയോക്താവിനും കമ്പ്യൂട്ടറിനും ഇടയിൽ ആശയവിനിമയം നടത്താൻ പ്രോഗ്രാമിങ് ഭാഷ ഉപയോഗിക്കുന്നു. കമ്പ്യൂട്ടർ പ്രോഗ്രാം എഴുതുന്ന വ്യക്തിക്ക് കംപ്യൂട്ടറിനു മനസ്സിലാകുന്ന ഭാഷയും പരിചിതമായിരിക്കണം. മനുഷ്യർക്ക് മനസ്സിലാക്കാനും ഉപയോഗിക്കാനും ബുദ്ധിമുട്ടുള്ള ദ്വയാങ്ക ഭാഷ (BInary Language) മാത്രമേ കമ്പ്യൂട്ടറിന് അറിയുകയുള്ളൂ എന്നത് നമ്മൾ നേരത്തെ കണ്ടതാണ്. അതിനാൽ അധ്യായം മൂന്നിൽ പഠിച്ചതു പോലെ ഇംഗ്ലീഷ് ഭാഷയ്ക്ക് സമാനമായതും മനുഷ്യർക്ക് സഹൃദയമായതുമായ ഉയർന്നതല ഭാഷ (High Level Language) (HLL) നമുക്ക് ഉപയോഗിക്കാവുന്നതാണ്. ഉയർന്നതല ഭാഷയിൽ എഴുതിയിരിക്കുന്ന പ്രോഗ്രാമുകളെ യന്ത്ര ഭാഷയിലേക്കു വിവർത്തനം അല്ലെങ്കിൽ മൊഴിമാറ്റം ചെയ്യാൻ ലാംഗ്വേജ് പ്രോസസ്സർ ഉപയോഗിക്കുന്നു. ഉയർന്നതല ഭാഷയിൽ എഴുതിയിരിക്കുന്ന പ്രോഗ്രാം, സോഴ്സ് കോഡ് (Source Code) എന്നറിയപ്പെടുന്നു.

ഒരു പ്രോഗ്രാമർ ആകണമെങ്കിൽ പ്രോഗ്രാമിലെ നിർദേശങ്ങൾ ആവിഷ്ക്കരിക്കുന്നതിനു വേണ്ടിയുള്ള ബേസിക് (BASIC), കോബോൾ (COBOL), പാസ്കൽ (PASCAL), C++ തുടങ്ങിയ ഉയർന്നതല ഭാഷകളിൽ ഏതിലെങ്കിലും ഒന്നിൽ നൈപുണ്യം കൈവരിക്കേണ്ടതാണ്. പ്രോഗ്രാമുകൾ തയാറാക്കുന്നതിന് വേണ്ടി ഓരോ പ്രോഗ്രാമിങ് ഭാഷയും അതിന്റേതായ അക്ഷരമാലകളും (Charecter Set), ശബ്ദകോശങ്ങളും (Voccabulary), വ്യാകരണങ്ങളും (Grammar) (നമുക്കതിനെ വാക്യഘടന (Syntax) എന്ന് വിളിക്കാം) ഉണ്ടായിരിക്കും.

ഈ ഭാഷ ഉപയോഗിച്ച് പ്രോഗ്രാം തയ്യാറാക്കി കഴിഞ്ഞാൽ അത് ഒരു ഫയലിൽ (സോഴ്സ് ഫയൽ) സൂക്ഷിക്കുകയും പ്രോഗ്രാമിങ്ങിന്റെ അടുത്ത ഘട്ടത്തിലേക്ക് കടക്കുകയും ചെയ്യുന്നു.

3.3.4 പരിഭാഷ (Translation)

സോഴ്സ് കോഡ് വികസിപ്പിക്കുന്നതിനായി ഭാഷ തിരഞ്ഞെടുക്കുമ്പോൾ ഉപയോഗിക്കുന്ന ഡാറ്റയുടെ അളവ്, പ്രവർത്തനത്തിന്റെ സങ്കീർണ്ണത, ഫയലുകളുടെ ഉപയോഗം മുതലായ ചില മാനദണ്ഡങ്ങൾ പരിഗണിക്കേണ്ടതുണ്ട്. പ്രോഗ്രാമിങ് ഭാഷ തിരഞ്ഞെടുക്കുകയും സോഴ്സ് കോഡ് തയ്യാറാക്കുകയും ചെയ്തു കഴിഞ്ഞാൽ അതിനെ ലാംഗ്വേജ് പ്രോസെസ്സറിൻറെ സഹായത്തോടെ പരിഭാഷ ചെയ്യേണ്ടതാണ്. ഉയർന്നതല ഭാഷയിൽ എഴുതപ്പെട്ടിരിക്കുന്ന പ്രോഗ്രാമിനെ അതിനു തുല്യമായ യന്ത്രഭാഷാ പതിപ്പിലേക്കു രൂപഭേദം വരുത്തുന്ന പ്രക്രിയയാണ് പരിഭാഷ (Translation). കംപൈലർ അല്ലെങ്കിൽ ഇൻറർപ്രെറ്റർ ഇതിനായി ഉപയോഗിക്കുന്നു. പ്രോഗ്രാമിലുള്ള സിന്റാക്സ് എററുകൾ (Syntax Error) ഈ ഘട്ടത്തിൽ ദൃശ്യമാകുന്നു. സോഴ്സ് കോഡ് അടങ്ങിയ ഫയൽ തുറന്നു ഈ തെറ്റുകൾ തിരുത്തേണ്ടതാണ്. അതിനു ശേഷം സോഴ്സ് കോഡ് വീണ്ടും കമ്പൈൽ ചെയ്യാനായി (പരിഭാഷ) നൽകുന്നു. "No Errors or Warnings" അല്ലെങ്കിൽ "Sucessfull Compilation" എന്ന സന്ദേശം ലഭിക്കുന്നത് വരെ ഈ പ്രക്രിയ തുടർന്ന് കൊണ്ടിരിക്കും. പൂർണമായും യന്ത്രഭാഷയിലെ നിർദേശങ്ങൾ അടങ്ങിയ പ്രോഗ്രാം നമുക്കിപ്പോൾ ലഭ്യമാകുന്നു.സോഴ്സ് കോഡിൻറെ ഈ പതിപ്പ് ഒബ്ജക്റ്റ് കോഡ് (Object Code) എന്ന് അറിയപ്പെടുന്നു. കംപൈലർ തന്നെ ഈ ഒബ്ജക്റ്റ് കോഡിനെ മറ്റൊരു ഒരു ഫയലിൽ സൂക്ഷിക്കുകയും ചെയ്യുന്നു



ചിത്രം 3.18 പരിഭാഷാ പ്രവർത്തനം

ഇങ്ങനെ ഒബ്ജക്റ്റ് കോഡ് ലഭിച്ചു കഴിഞ്ഞാൽ പ്രോഗ്രാം ഉപയോഗിക്കുന്നിടത്തോളം കാലം ഈ കോഡ് ആ കമ്പ്യൂട്ടറിൽ തന്നെ ഉണ്ടായിരിക്കേണ്ടതാണ്.

3.3.5 ഡീബഗ്ഗിങ് (Debugging)

പ്രോഗ്രാമിങ്ങിലുള്ള തെറ്റുകൾ കണ്ടെത്തുകയും അവ തിരുത്തുകയും ചെയ്യുന്ന ഘട്ടമാണ് ഡീബഗ്ഗിങ്.മനുഷ്യർ കമ്പ്യൂട്ടറുകളെ പ്രോഗ്രാം ചെയ്യുന്നിടത്തോളം കാലം പ്രോഗ്രാമുകളിൽ തെറ്റുകൾ സംഭവിക്കാം. പ്രോഗ്രാമിങ്ങിലുള്ള തെറ്റുകളെ 'ബഗ്ഗ്' എന്ന് പറയുന്നു.തെറ്റുകൾ കണ്ടുപിടിക്കുകയും തിരുത്തുകയും ചെയ്യുന്ന പ്രക്രിയയെ 'ഡീബഗ്ഗിങ്' എന്ന് വിളിക്കുന്നു. പൊതുവെ പ്രോഗ്രാമിങ്ങിൽ രണ്ടു തരത്തിലുള്ള തെറ്റുകളാണ് വന്നുകൂടുക സിന്റാക്സ് എററുകളും ലോജിക്കൽ തെറ്റുകളും. പ്രോഗ്രാമിങ് ഭാഷയുടെ നിയമങ്ങൾ അല്ലെങ്കിൽ വാക്യഘടന പാലിക്കാത്തതു കൊണ്ട് സംഭവിക്കുന്ന തെറ്റുകളെയാണ് സിന്റാക്സ് എറർ എന്ന് വിളിക്കുന്നത്. തെറ്റായ ചിഹ്നങ്ങൾ ഉപയോഗിക്കുക, തെറ്റായ വാക്യപ്രയോഗം, നിർവചിക്കപ്പെടാത്ത പദങ്ങളുടെ ഉപയോഗം, നിയമവിരുദ്ധമായ വാക്യരചന അല്ലെങ്കിൽ പദങ്ങളുടെ ഉപയോഗം മുതലായ കാരണങ്ങൾ

കൊണ്ടാണ് സാധാരണ അത്തരം തെറ്റുകൾ സംഭവിക്കുക. പരിഭാഷക്കു വേണ്ടി പ്രോഗ്രാം നൽകി കഴിഞ്ഞാൽ തന്നെ ലാംഗ്വേജ് പ്രൊസസ്സറുകൾ സിൻറ്റാക്സ് തെറ്റുകൾ കണ്ടെത്തുന്നു. തെറ്റുകൾ സംഭവിച്ചിരിക്കുന്ന പ്രസ്താവനകളുടെ ലൈൻ നമ്പറുകൾ അതിനോടൊപ്പം സംഭവിച്ചിരിക്കുന്ന തെറ്റിനെ കുറിച്ചുള്ള സൂചനകളും അവ നൽകുന്നു. എറർ മെസ്സേജുകളായി പ്രദർശിപ്പിക്കുന്നു. ഇൻറ്റർപ്രെറ്ററുകളുടെ കാര്യത്തിൽ, പ്രവർത്തന ഘട്ടത്തിലാണ് സിൻറ്റാക്സ് തെറ്റുകൾ കണ്ടുപിടിച്ച് പ്രദർശിപ്പിക്കുന്നത്. ഡീബഗ്ഗിങ് പ്രക്രിയക്കായി വേണ്ടി വരുന്ന സമയവും പരിശ്രമവുമാണ് ഒരു പ്രോഗ്രാമിങ് ഭാഷ ഉപയോഗിക്കുന്നതിൽ പ്രോഗ്രാമർക്കുള്ള പ്രാപ്തി എത്രത്തോളമുണ്ട് എന്ന് തീരുമാനിക്കുന്നത്. എല്ലാ സിൻറ്റാക്സ് തെറ്റുകളും തിരുത്തിക്കഴിഞ്ഞാൽ മാത്രമേ ഒബ്ജക്റ്റ് പ്രോഗ്രാം നിർമ്മിക്കപ്പെടുകയുള്ളു.

പ്രോഗ്രാമിങ് യുക്തിയുടെ ആസൂത്രണത്തിലുള്ള അപാകതകൾ കാരണമാണ് ലോജിക്കൽ എറർ എന്ന് പേരുള്ള രണ്ടാമത്തെ തരം തെറ്റുകൾ സംഭവിക്കുന്നത്.സിൻറ്റാക്സ് എററുകൾ ഇല്ലെങ്കിൽ ലാംഗ്വേജ് പ്രൊസസ്സറുകൾ വിജയകരമായി സോഴ്സ് കോഡിനെ പരിഭാഷപ്പെടുത്തുന്നു. പ്രോഗ്രാമിന്റെ പ്രവർത്തന ഘട്ടത്തിൽ കമ്പ്യൂട്ടർ, പ്രോഗ്രാം നിർദേശങ്ങൾ പിന്തുടരുകയും അവക്ക് അനുസൃതമായ ഔട്ട്പുട്ട് നൽകുകയും ചെയ്യുന്നു. പക്ഷെ ഈ ഔട്ട്പുട്ട് ശരിയായിരിക്കണമെന്നില്ല. ഇതിനെയാണ് ലോജിക്കൽ എറർ എന്ന് പറയുന്നത്. ലോജിക്കൽ എറർ സംഭവിക്കുമ്പോൾ പ്രോഗ്രാം തെറ്റായ ഔട്ട്പുട്ട് ആണ് തരുന്നത് എന്ന് മാത്രമേ നമുക്ക് ഗ്രഹിക്കാൻ സാധിക്കുകയുള്ളു.എന്താണ് തെറ്റ് എന്ന് കമ്പ്യൂട്ടർ നമ്മോടു പറയുന്നില്ല. പ്രോഗ്രാമർ അല്ലെങ്കിൽ ഉപയോക്താവാണ് അത് കണ്ടെത്തേണ്ടത്. ലോജിക്കൽ തെറ്റുകൾ ഉണ്ടോ ഇല്ലയോ എന്നറിയുന്നതിനായി പ്രോഗ്രാം പരീക്ഷിക്കപ്പെടേണ്ടതാണ്. അതിനായി പ്രോഗ്രാമിങിന്റെ അടുത്ത ഘട്ടത്തിലേക്ക് നമുക്ക് കടക്കാം.

3.3.6 പ്രവർത്തനവും പരീക്ഷണവും (Execution and Testing)

മുകളിൽ പറഞ്ഞത് പോലെ ലോജിക്കൽ തെറ്റുകൾ കൂടി തിരുത്തിയാൽ മാത്രമേ ഒരു പ്രോഗ്രാം തെറ്റുകളിൽ നിന്നും മുക്തമാണ് എന്ന് നമുക്ക് പറയാൻ കഴിയൂ. ആയതിനാൽ കംപൈൽ ചെയ്യപ്പെട്ട പ്രോഗ്രാമിന്റെ പതിപ്പ് പരീക്ഷണത്തിനായി പ്രവർത്തിപ്പിക്കേണ്ടതാണ്. ശരിയായ ഫലങ്ങൾ ലഭ്യമാകുന്നുണ്ടോ എന്ന് പരിശോധിക്കുകയാണ് പരീക്ഷണത്തിന്റെ ഉദ്ദേശ്യം. 'അറിയാവുന്ന ഫലങ്ങൾ' ലഭിക്കുന്നതിന് വേണ്ടിയുള്ള പരീക്ഷണ ഡാറ്റ നൽകി പ്രോഗ്രാം പ്രാവർത്തികമാക്കുക എന്ന പ്രക്രിയയാണ് പരീക്ഷണ ഘട്ടത്തിൽ ഉൾപ്പെട്ടിരിക്കുന്നത്. അതായത് പ്രോഗ്രാമിൽ ഉൾപ്പെട്ടിരിക്കുന്ന ക്രിയകൾ മനുഷ്യൻ തന്നെ ചെയ്യുകയും, ലഭ്യമാകുന്ന ഔട്ട്പുട്ട് കമ്പ്യൂട്ടർ മുഖേന ചെയ്യുമ്പോൾ ലഭിക്കുന്ന ഔപുട്ടുമായി താരതമ്യം ചെയ്യുകയും വേണം. പ്രോഗ്രാം യുക്തിയുടെ കൃത്യത ഈ പരീക്ഷണ ഘട്ടം കൊണ്ട് നിർണയിക്കാവുന്നതാണ്. പരീക്ഷണത്തിനായുള്ള ഡാറ്റ തിരഞ്ഞെടുക്കുമ്പോൾ പ്രോഗ്രാം യുക്തിയുടെ എല്ലാ വശങ്ങളും പരീക്ഷിക്കപ്പെടുന്നുണ്ട് എന്ന് ഉറപ്പാക്കേണ്ടതാണ്. അതിനാൽ തന്നെ ഉചിതമായ ഡാറ്റ തിരഞ്ഞെടുക്കുക എന്നത് പ്രോഗ്രാം പരീക്ഷണത്തിൽ വളരെ പ്രാധാന്യമുള്ളതാണ്.

തെറ്റായ യുക്തി മൂലം ലഭിക്കുന്ന തെറ്റായ ഔട്ട്പുട്ടുകളെ കുറിച്ചാണ് നാം ഇതുവരെ ചർച്ച ചെയ്തു കൊണ്ടിരുന്നത്. എന്നാൽ പ്രോഗ്രാം പ്രവർത്തനത്തെ തടസ്സപ്പെടുത്താവുന്ന മറ്റൊരു തരം തെറ്റ് സംഭവിക്കാൻ സാധ്യതയുണ്ട്. ഒരു ക്രിയയിൽ അനുചിതമായ ഡാറ്റ വരുന്നത് മൂലം സംഭവിക്കാവുന്ന ഒന്നാണത്.ഉദാഹരണത്തിന്, A =B / C എന്ന നിർദേശം പരിഗണിക്കുക. Cയുടെ വില പൂജ്യം ആയാൽ ഈ പ്രസ്താവന പ്രോഗ്രാമിന്റെ പ്രവർത്തനത്തെ തടസ്സപ്പെടുത്തുന്നു (പൂജ്യം കൊണ്ടുള്ള ഹരണം മൂലം). ഇത്തരം സാഹചര്യങ്ങളിൽ പ്രോഗ്രാമിംഗ് ഭാഷയിലുള്ള തെറ്റുകൾ കൈകാര്യം ചെയ്യുന്ന ഫങ്ക്ഷനുകൾ (Error handling function) എറർ മെസ്സേജുകൾ പ്രദർശിപ്പിക്കുന്നു.ഇത്തരം തെറ്റുകളെ റൺ ടൈം എറർ എന്ന് വിളിക്കുന്നു.ഡാറ്റ പ്രോസസ്സ് ചെയ്യപ്പെടുന്നതിനു മുമ്പ് ഡാറ്റയുടെ സാധുത പരിശോധിക്കാനുള്ള അനുബന്ധ നിർദേശങ്ങൾ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തുക വഴി ഇത്തരം തെറ്റുകൾ തിരുത്താവുന്നതാണ്.

3.3.7 വിവരണം തയ്യാറാക്കൽ (Documentation)

ഉചിതമായ രീതിയിയിൽ വിവരണം തയ്യാറാക്കാത്ത ഒരു കംപ്യൂട്ടർവൽകൃത സംവിധാനം ഒരിക്കലും പൂർണമാണെന്നു പറയാൻ നമുക്ക് കഴിയില്ല. വാസ്തവത്തിൽ, പ്രശ്നപഠന ഘട്ടം മുതൽ അത് പ്രയോഗത്തിൽ വരുത്തുന്നത് വരെ തുടർച്ചയായി നടന്നുകൊണ്ടിരിക്കുന്ന ഒരു പ്രക്രിയയാണ് വിവരണം തയ്യാരാക്കൽ. ഇതിന്റെ ഭാഗമായി സോഴ്സ് കോഡിൽ നമുക്ക് കമെന്റുകൾ ഉൾപ്പെടുത്താവുന്നതാണ്. ഇത് ആന്തരിക വിവരണം (Internal Documentation) എന്നറിയപ്പെടുന്നു. ഡീബഗ്ഗിങ് ഘട്ടത്തിലും പിൽക്കാലത്ത് പ്രോഗ്രാമിൽ വരുന്ന മാറ്റങ്ങൾ ഉൾപ്പെടുത്താനും ഇത് സഹായിക്കുന്നു. പ്രോഗ്രാം നിർമാണ സമയത്ത് ഉപയോഗിച്ച യുക്തി പിന്നീട് നമ്മുടെ തന്നെ പ്രോഗ്രാമിലൂടെ കടന്നു പോകുമ്പോൾ നമുക്ക് ഓർമയുണ്ടായിരിക്കണമെന്നില്ല. മാത്രമല്ല ചില സാഹചര്യങ്ങളിൽ ഒരു വ്യക്തി എഴുതിയ പ്രോഗ്രാം മറ്റൊരു വ്യക്തിക്ക് ഭാവിയിൽ മാറ്റേണ്ടതായി വരാം. ഒരു പ്രോഗ്രാമിൽ കൃത്യമായി വിവരണം തയ്യാറാക്കിയാൽ നമ്മൾ ഉപയോഗിച്ച യുക്തി മനസ്സിലാക്കാനും പ്രോഗ്രാമിൽ ഒരു പ്രസ്താവന എന്ത് കൊണ്ടാണ് ഉപയോഗിച്ചിരിക്കുന്നത് എന്ന് മനസ്സിലാക്കാനും സാധിക്കുന്നു. എന്നിരുന്നാലും പ്രോഗ്രാം പരിഭാഷക്കായി നൽകുമ്പോൾ ലാംഗേജ് പ്രൊസസ്സറുകൾ പ്രോഗ്രാമിന്റെ വിവരണ ഭാഗങ്ങൾ പരിഭാഷക്കായി പരിഗണിക്കുകയില്ല.

പ്രോഗ്രാമിന്റെ ഭാഗമായ കമെന്റുകൾ വിവരണം തയ്യാറാക്കുന്ന ഘട്ടത്തിന്റെ ഒരു ഭാഗം മാത്രമാണ്. സിസ്റ്റം മാനുവൽ, ഉപയോക്തൃ മാനുവൽ എന്നിവ തയ്യാറാക്കുക എന്നത് വിവരണം തയ്യാറാക്കുന്നതിന്റെ മറ്റൊരു പ്രക്രിയയാണ്. കമ്പ്യൂട്ടർ വ്യവസ്ഥയുടെ പ്രവർത്തനം, അവയുടെ ആവശ്യകത, പ്രോഗ്രാമുകൾ ഇൻസ്റ്റാൾ ചെയ്യുക, അവ ഉപയോഗിക്കുന്ന രീതികൾ എന്നിവ ഉൾപ്പെടുന്ന ഹാർഡ് കോപ്പികളാണിവ. വിവിധ ആവശ്യങ്ങൾക്ക് വേണ്ടിയുള്ള സോഫ്റ്റ്വെയറുകൾ തയ്യാറാക്കുമ്പോൾ ഇത്തരം മാനുവലുകൾ നിർബന്ധമാണ്. ഇത്തരം വിവരണം തയ്യാറാക്കുന്നതിനെയാണ് ബാഹ്യമായ വിവരണം (External Documentation) എന്ന് പറയുന്നത്.

ഇപ്പോൾ നാം ഒരു പ്രശ്നത്തെ വിശകലനം ചെയ്യുകയും പരിഹാരത്തിനുള്ള യുക്തി കണ്ടെത്തുകയും, ഫ്ളോ ചാർട്ട് രൂപത്തിൽ പ്രതിപാദിക്കുകയും, പ്രോഗ്രാമിങ് ഭാഷയിൽ കോഡ് തയ്യാറാക്കുകയും, സിന്റാക്സിലെ പിശകുകൾ നീക്കം ചെയ്ത ശേഷം പരിഭാഷപ്പെടുത്തുകയും ചെയ്തു. കൂടാതെ ലോജിക്കൽ തെറ്റുകളും റൺ ടൈം തെറ്റുകളും നീക്കം ചെയ്ത് ശേഷം ഔട്ട്പുട്ടിന്റെ കൃത്യത പരിശോധിക്കുകയും അവസാനമായി പ്രോഗ്രാമിന്റെ വിവരണം തയ്യാറാക്കുകയും ചെയ്തു.

സ്വയം വിലയിരുത്താം

1. അൽഗോരിതം എന്നാൽ എന്താണ് ?
2. അൽഗോരിതത്തിന്റെ ചിത്ര ആവിഷ്കരണമാണ് __________.
3. ഏതു ഫ്ളോച്ചാർട്ട് ചിഹ്നമാണ് എപ്പോഴും ജോഡികളായി ഉപയോഗിക്കുന്നത്?
4. ഏതു ഫ്ളോച്ചാർട്ട് ചിഹ്നത്തിനാണ് ഒരു ആഗമന മാർഗവും രണ്ടോ അതിലധികമോ നിർഗമന മാർഗങ്ങളും ഉള്ളത് ?
5. HLL ൽ എഴുതിയിരിക്കുന്ന പ്രോഗ്രാം _________ എന്ന് അറിയപ്പെടുന്നു.
6. ഡീബഗ്ഗിങ് എന്നാലെന്ത് ?
7. ഒബ്ജക്റ്റ് കോഡ് എന്നാലെന്ത് ?

3.4 അൽഗോരിതങ്ങളുടെ പ്രകടനം വിലയിരുത്തൽ (Performance evaluation of algorithms)

വിവിധ പ്രശ്നങ്ങൾ പരിഹരിക്കുന്നതിനായി നമ്മൾ അൽഗോരിതങ്ങൾ തയാറാക്കിയിട്ടുണ്ട്. ചിലപ്പോൾ ചില പ്രശ്നങ്ങൾ പരിഹരിക്കുന്നതിന് വ്യത്യസ്തമായ യുക്തി പ്രയോഗിക്കാമായിരുന്നു എന്ന് നമുക്ക് തോന്നാം. ഒരേ പ്രശ്നം തന്നെ വ്യത്യസ്തമായ ഒരു കൂട്ടം നിർദേശങ്ങൾ ഉപയോഗിച്ച് പരിഹരിക്കാവുന്നതാണ്. എന്നിരുന്നാലും വളരെ കുറഞ്ഞ കമ്പ്യൂട്ടർ വിഭവങ്ങൾ ഉപയോഗപ്പെടുത്തി, കുറഞ്ഞ സമയം കൊണ്ട് വളരെ കൃത്യമായ ഫലം നൽകുന്ന പ്രോഗ്രാം തയ്യാറാക്കുന്നയാളാണ് ഒരു സമർത്ഥനായ പ്രോഗ്രാമർ എന്ന് പറയുന്നത്. ഒരു അൽഗോരിതത്തിന്റെ പ്രകടനം അളക്കുന്നത് ടൈം കോംപ്ലക്സിറ്റി (Time complexity), സ്പേസ് കോംപ്ലക്സിറ്റി (Space complexity) എന്നിവയുടെ അടിസ്ഥാനത്തിലാണ്. ഏറ്റവും കുറവ് മെമ്മറി ഉപയോഗപ്പെടുത്തി ഏറ്റവും വേഗത്തിൽ പ്രവർത്തിക്കുന്ന അൽഗോരിതത്തെയാണ് പ്രശ്ന പരിഹാരത്തിനുള്ള ഏറ്റവും നല്ല അൽഗോരിതം ആയി കണക്കാക്കുന്നത്.

അൽഗോരിതം 1		അൽഗോരിതം 2	
ഘട്ടം 1 :	തുടങ്ങുക	ഘട്ടം 1 :	തുടങ്ങുക
ഘട്ടം 2 :	A ,B ,C ഇൻപുട്ട് ചെയ്യുക	ഘട്ടം 2 :	A ,B ,C ഇൻപുട്ട് ചെയ്യുക
ഘട്ടം 3 :	S =A +B +C	ഘട്ടം 3 :	S=A+B+C
ഘട്ടം 4 :	AVG =S / 3	ഘട്ടം 4 :	AVG =(A +B +C) / 3
ഘട്ടം 5 :	S, AVG പ്രിന്റ് ചെയ്യുക	ഘട്ടം 5 :	S, AVG പ്രിന്റ് ചെയ്യുക
ഘട്ടം 6 :	അവസാനിപ്പിക്കുക	ഘട്ടം 6 :	അവസാനിപ്പിക്കുക

പട്ടിക 3 .2 മൂന്നു സംഖ്യകളുടെ തുകയും ശരാശരിയും കാണാനുള്ള രണ്ടു അൽഗോരിതങ്ങൾ

മൂന്നു സംഖ്യകളുടെ തുകയും ശരാശരിയും കാണാനുള്ള രണ്ടു അൽഗോരിതങ്ങൾ പട്ടിക 3.2ൽ കൊടുത്തിരിക്കുന്നു. അവയെ നമുക്ക് താരതമ്യം ചെയ്യാം. രണ്ടു അൽഗോരിതങ്ങളും ഘട്ടം 4 ൽ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു. അൽഗോരിതം2 ൽ ഒരേ ഡാറ്റയിൽ രണ്ടു തവണ സങ്കലനം ചെയ്യുന്നു (ഘട്ടം 3, ഘട്ടം 4).സ്വാഭാവികമായിട്ടും ഈ അൽഗോരിതം അൽഗോരിതം1 നെക്കാൾ കൂടുതൽ സമയം പ്രവർത്തനത്തിനായി ഉപയോഗിക്കുന്നു. അതിനാൽ കോഡിങ്ങിനു നല്ലതു അൽഗോരിതം 1 ആകുന്നു.

ഒരു പ്രസ്താവന തിരഞ്ഞെടുക്കുന്നതിന് വേണ്ടി താരതമ്യ ക്രിയകൾ ഉൾപ്പെടുന്ന മറ്റൊരു ഉദാഹരണം നമുക്കു പരിശോധിക്കാം. മൂന്നു സംഖ്യകളിൽ വലുത് ഏതാണെന്നു കണ്ടെത്താനുള്ള

അൽഗോരിതം ഉദാഹരണം 3.4 ൽ നമ്മൾ ചർച്ച ചെയ്യുകയുണ്ടായി. അതേ പ്രശ്നം പരിഹരിക്കുന്നതിനുള്ള രണ്ടു അൽഗോരിതങ്ങൾ പട്ടിക 4.3 ൽ നൽകിയിരിക്കുന്നു.

ഉദാഹരണം 3.3 ൽ മൂന്ന് താരതമ്യ ക്രിയകളും ഒരു ലോജിക്കൽ ക്രിയയും ഉപയോഗിച്ചിരിക്കുന്നു. ഏറ്റവും വലിയ വില M3 ൽ (മൂന്നാമത്തെ വേരിയബിൾ) വരുമ്പോളാണ് ഈ ക്രിയകൾ എല്ലാം പ്രവർത്തിക്കുക. പ്രവർത്തനത്തിന്റെ വേഗത അളക്കുന്നതിനായി താരതമ്യ ക്രിയക്ക് 1 സെക്കന്റ് സമയമെടുക്കുന്നു എന്ന് നമുക്ക് അനുമാനിക്കാം. അങ്ങനെയാണെങ്കിൽ ഏറ്റവും വേഗം കൂടിയ ഫലം 3 സെക്കന്റ് വേഗം കുറവുള്ള ഫലം 4 സെക്കന്റ് വേണ്ടി വരുന്നതായി കാണാം. ശരാശരി വേഗം 3.5 സെക്കന്റ് ആയിരിക്കും.

അൽഗോരിതം 1	അൽഗോരിതം 2
ഘട്ടം 1 : തുടങ്ങുക	ഘട്ടം 1 : തുടങ്ങുക
ഘട്ടം 2 : M 1 ,M 2 ,M 3 ഇൻപുട്ട് ആയി സ്വീകരിക്കുക	ഘട്ടം 2 : M 1, M 2, M 3 ഇൻപുട്ട് ആയി സ്വീകരിക്കുക
ഘട്ടം 3 : അഥവാ M 1 >M 2 ഉം M 1 > M 3 ഉം ആണെങ്കിൽ	ഘട്ടം 3 : അഥവാ M 1 >M 2 ആണെങ്കിൽ
ഘട്ടം 4 : M 1 പ്രിന്റ് ചെയ്യുക	ഘട്ടം 4 : Big = M 1
ഘട്ടം 5 : അഥവാ M 2 >M 1 ഉം M 2 > M 3 ഉം ആണെങ്കിൽ	ഘട്ടം 5 : അല്ലെങ്കിൽ
ഘട്ടം 6 : M 2 പ്രിന്റ് ചെയ്യുക	ഘട്ടം 6 : Big = M 2
ഘട്ടം 7 : അഥവാ M 3 >M 1 ഉം M 3 > M 2 ഉം ആണെങ്കിൽ	ഘട്ടം 7 : അഥവാ M 3 >Big ആണെങ്കിൽ Big = M 3
ഘട്ടം 8 : M3 പ്രിന്റ് ചെയ്യുക	ഘട്ടം 8 : Big പ്രിന്റ് ചെയ്യുക
ഘട്ടം 9 : അവസാനിപ്പിക്കുക	ഘട്ടം 9 : അവസാനിപ്പിക്കുക

പട്ടിക 3.3 മൂന്നു സംഖ്യകളിൽ ഏറ്റവും വലുത് കണ്ടെത്താനുള്ള അൽഗോരിതങ്ങൾ

ഇനി നമുക്ക് പട്ടിക 3.3 ലുള്ള അൽഗോരിതം1 വിശകലനം ചെയ്യാം. അതിൽ മൂന്ന് താരതമ്യ ക്രിയകൾ അടങ്ങിയ മൂന്ന് If (അഥവാ) പ്രസ്താവനകൾ അടങ്ങിയിരിക്കുന്നു. മുകളിൽ പ്രസ്താവിച്ചിട്ടുള്ള അനുമാനം പിന്തുടരുകയാണെങ്കിൽ വേരിയബിലിന്റെ വില ഏതാണെങ്കിലും നമുക്ക് ഫലം 9 സെക്കന്റിൽ ലഭിക്കുന്നതാണ്. അപ്പോൾ ശരാശരി വേഗം എന്ന് പറയുന്നത്തു 9 സെക്കന്റ് ആണ്. പക്ഷെ പട്ടിക 3.3 ൽ പറയുന്ന അൽഗോരിതം2ൽ രണ്ടു If (അഥവാ) പ്രസ്താവനകൾ ഉപയോഗിച്ചിരിക്കുന്നു. അൽഗോരിതം2 ൽ വേരിയബിളുകളിൽ ഏതു വില വന്നാലും താരതമ്യം ചെയ്യാൻ വേണ്ട സമയം 2 സെക്കന്റ് ആയിരിക്കും. അപ്പോൾ ശരാശരി വേഗം എന്ന് പറയുന്നത് 2 സെക്കന്റ് ആയിരിക്കും.എന്നു പറഞ്ഞാൽ അൽഗോരിതം 2 മറ്റു രണ്ടു അൽഗോരിതങ്ങളെക്കാൾ മെച്ചപ്പെട്ടതാണ്.

ലൂപ്പുകൾ അടങ്ങിയ മറ്റൊരു ഉദാഹരണം കൂടി നമുക്ക് നോക്കാം. 100 നും 200 നും ഇടയിലുള്ള എല്ലാ ഒറ്റ സംഖ്യകളുടെ തുകയും എല്ലാ ഇരട്ട സംഖ്യകളുടെ തുകയും കണ്ടുപിടിക്കാനുള്ള രണ്ടു അൽഗോരിതങ്ങൾ പട്ടിക 3.4 ൽ കൊടുത്തിരിക്കുന്നു.

അൽഗോരിതം 1

```
ഘട്ടം 1 :  തുടങ്ങുക
ഘട്ടം 2 :  N =100, ES =0
ഘട്ടം 3 :  (N <=200) ആയിരിക്കുന്നതു വരെ ഘട്ടം 4 മുതൽ 6 വരെ ആവർത്തിക്കുക
ഘട്ടം 4 :  അഥവാ N/2 ന്റെ ശിഷ്ടം = 0 ആണെങ്കിൽ
ഘട്ടം 5 :          ES =ES + N
ഘട്ടം 6 :  N =N +1
ഘട്ടം 7 :  ES പ്രിന്റ് ചെയ്യുക
ഘട്ടം 8 :  N =100, OS =0
ഘട്ടം 9 :  (N <=200 ) ആയിരിക്കുന്നതു വരെ ഘട്ടം 10 മുതൽ 12 വരെ ആവർത്തിക്കുക
ഘട്ടം 10 : അഥവാ N/2 ന്റെ ശിഷ്ടം = 1 ആണെങ്കിൽ
ഘട്ടം 11 :         OS =OS + N
ഘട്ടം 12 : N =N +1
ഘട്ടം 13 : OS പ്രിന്റ് ചെയ്യുക
ഘട്ടം 14 : അവസാനിപ്പിക്കുക
```

അൽഗോരിതം 2

```
ഘട്ടം 1 :  തുടങ്ങുക
ഘട്ടം 2 :  N =100 ,ES =0,OS =0
ഘട്ടം 3 :  (N <=200) ആയിരിക്കുന്നതു വരെ ഘട്ടം 4 മുതൽ 8 വരെ ആവർത്തിക്കുക
ഘട്ടം 4 :  അഥവാ N/2 ന്റെ ശിഷ്ടം = 0 ആണെങ്കിൽ
ഘട്ടം 5 :          ES =ES + N
ഘട്ടം 6 :  അല്ലെങ്കിൽ
ഘട്ടം 7 :          OS =OS + N
ഘട്ടം 8 :  N =N +1
ഘട്ടം 9 :  ES പ്രിന്റ് ചെയ്യുക
ഘട്ടം 10 : OS പ്രിന്റ് ചെയ്യുക
ഘട്ടം 11 : അവസാനിപ്പിക്കുക
```

പട്ടിക 3.4 ഒറ്റ സംഖ്യകളുടെയും ഇരട്ട സംഖ്യകളുടെയും തുക കാണാനുള്ള അൽഗോരിതങ്ങൾ

അൽഗോരിതം1 രണ്ടു ലൂപ്പുകളും അൽഗോരിതം2 ഒരു ലൂപ്പും ഉപയോഗിക്കുന്നു. സ്വാഭാവികമായിട്ടും അൽഗോരിതം2 നെ അപേക്ഷിച്ച് അൽഗോരിതം1 ന് പ്രാരംഭ വിലനൽകാനും, പരിശോധനയ്ക്ക് വേണ്ടിയും ലൂപ്പ് വേരിയബിൾ പുതുക്കുന്നതിന് വേണ്ടിയും മറ്റും ഇരട്ടി സമയം ആവശ്യമായി വരുന്നു. അൽഗോരിതം2 മെച്ചപ്പെട്ടതും കാര്യക്ഷമവുമാണ് എന്ന് പട്ടികയിൽ നിന്ന് തന്നെ വ്യക്തമാണ്. അതിനാൽ തന്നെ പ്രശ്ന പരിഹാരത്തിനുള്ള യുക്തി വികസിപ്പിക്കുന്നതിനു മുമ്പ് വ്യത്യസ്തവും വിഭിന്നവുമായി ചിന്തിക്കേണ്ടതു അനിവാര്യമാണ്.

നമുക്ക് സംഗ്രഹിക്കാം

ഒരു കമ്പ്യൂട്ടർ ഭാഷയിൽ ക്രമത്തിൽ എഴുതിയിരിക്കുന്ന നിർദേശങ്ങളാണ് പ്രോഗ്രാം. പ്രോഗ്രാമിങ് പ്രക്രിയ ചില ഘട്ടങ്ങളിലൂടെ കടന്നു പോകുന്നു. അൽഗോരിതവും ഫ്ളോച്ചാർട്ടും തയ്യാറാക്കുന്നത് പ്രോഗ്രാമിങ്ങിന്റെ യുക്തി വികസിപ്പിക്കാൻ സഹായിക്കുന്നു. HLL ൽ തയ്യാറാക്കിയിരിക്കുന്ന പ്രോഗ്രാമിനെ സോഴ്സ് കോഡ് എന്ന് വിളിക്കുന്നു. അതിനെ യന്ത്ര ഭാഷയിലേക്കു പരിഭാഷപ്പെടുത്തേണ്ടതാണ്. അതിന്റെ ഫലമായി ലഭിക്കുന്ന കോഡ്, ഒബ്ജക്റ്റ് കോഡ് എന്ന് അറിയപ്പെടുന്നു. ഡീബഗ്ഗിങ് എന്ന് വിളിക്കുന്ന പ്രക്രിയയിലൂടെ പ്രോഗ്രാമിൽ സംഭവിച്ചിരിക്കുന്ന തെറ്റുകളെ നീക്കം ചെയ്യുന്നു. പരിഭാഷപ്പെടുത്തിയ പതിപ്പ് കമ്പ്യൂട്ടർ പ്രവർത്തിപ്പിക്കുന്നു. ഉചിതമായ വിവരണം തയ്യാറാക്കുന്നതിന് പിൽക്കാലത്തു പ്രോഗ്രാമിൽ മാറ്റം വരുത്തുന്നതിന് സഹായകമാകുന്നു. പ്രശ്ന പരിഹാരത്തിന് വ്യത്യസ്തങ്ങളായ യുക്തികൾ പ്രയോഗിക്കാമെങ്കിലും പ്രോഗ്രാമിന്റെ പ്രകടനം അളക്കുന്നത് ടൈം കോംപ്ലക്സിറ്റിയുടെയും സ്പേസ് കോംപ്ലക്സിറ്റിയുടെയും അടിസ്ഥാനത്തിലാണ്.

പഠന നേട്ടങ്ങൾ

ഈ അദ്ധ്യായം പൂർത്തിയാകുന്നതോടെ പഠിതാവിന്

- പ്രശ്ന പരിഹാരത്തിന്റെ വിവിധ വശങ്ങൾ വിശദീകരിക്കാൻ സാധിക്കുന്നു.
- പ്രശ്നങ്ങൾ പരിഹരിക്കാനുള്ള അൽഗോരിതങ്ങൾ വികസിപ്പിക്കാൻ സാധിക്കുന്നു.
- അൽഗോരിതത്തിൽ കൃത്യത ഉറപ്പു വരുത്താൻ ഫ്ളോച്ചാർട്ടുകൾ വരയ്ക്കാൻ സാധിക്കുന്നു.
- പ്രശ്നം പരിഹരിക്കാനുള്ള ഏറ്റവും നല്ല അൽഗോരിതം തിരഞ്ഞെടുക്കാൻ സാധിക്കുന്നു.

മാതൃക ചോദ്യങ്ങൾ

ഒറ്റ വാക്യത്തിൽ ഉത്തരം എഴുതുക

1. അൽഗോരിതം എന്നാലെന്ത് ?
2. പ്രശ്ന പരിഹാരത്തിൽ കമ്പ്യൂട്ടറിന്റെ പങ്ക് എന്താണ് ?
3. ഫ്ളോച്ചാർട്ടിൽ കണക്റ്ററിന്റെ ഉപയോഗമെന്ത് ?
4. പ്രോഗ്രാമിൽ ലോജിക്കൽ തെറ്റുകൾ എന്നാലെന്ത് ?

ലഘു വിവരണാത്മകം

1. കമ്പ്യൂട്ടർ പ്രോഗ്രാം എന്നാലെന്ത് ? പ്രോഗ്രാമുകൾ തയ്യാറാക്കുന്നതിന് അൽഗോരിതങ്ങൾ എങ്ങനെ സഹായിക്കുന്നു ?
2. 3 സംഖ്യകളുടെ തുകയും ശരാശരിയും കണ്ടുപിടിക്കാനുള്ള അൽഗോരിതം എഴുതുക
3. ആദ്യത്തെ 100 എണ്ണൽ സംഖ്യകൾ പ്രദർശിപ്പിക്കാനുള്ള ഫ്ളോച്ചാർട്ട് വരയ്ക്കുക
4. ഫ്ളോച്ചാർട്ടുകളുടെ പരിമിതികൾ എന്തെല്ലാം?
5. ഡീബഗ്ഗിങ് എന്നാലെന്ത്?
6. ഒരു പ്രോഗ്രാമിൽ വിവരണം തയ്യാറാക്കുന്നതിന്റെ ആവശ്യകത എന്ത്?

വിവരണാത്മകം

1. അൽഗോരിതത്തിന്റെ സവിശേഷതകൾ എന്തെല്ലാം?
2. ഫ്ളോച്ചാർട്ട് ഉപയോഗിക്കുന്നത് കൊണ്ടുള്ള ഗുണങ്ങൾ എന്തെല്ലാം?
3. പ്രോഗ്രാമിങിന്റെ വിവിധ ഘട്ടങ്ങളെ പറ്റി ചുരുക്കത്തിൽ വിവരിക്കുക