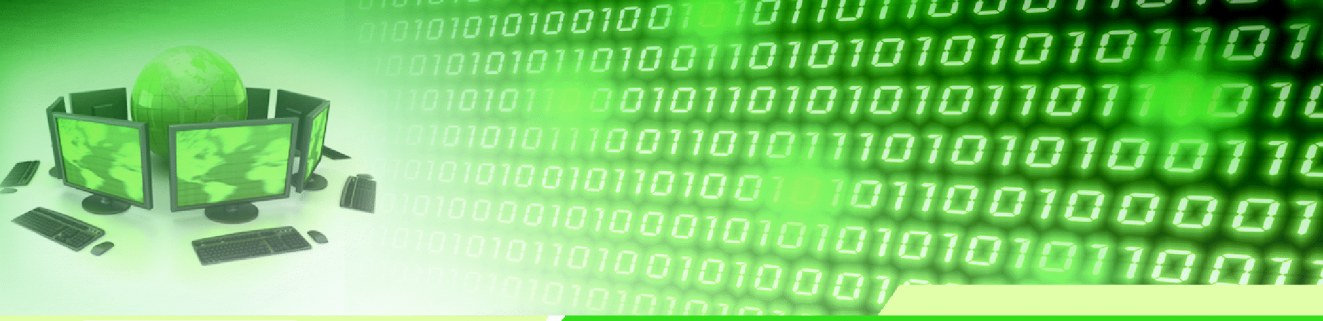


B1X4C8

1 C++ പ്രോഗ്രാമിങ്ങിനെക്കുറിച്ച് ഒരു പുനരവലോകനം

പ്രധാന പഠനനേട്ടങ്ങൾ

ഈ അധ്യായത്തിന്റെ പഠനം പൂർത്തിയാക്കുന്നതോടെ പഠിതാവ് ആർജിക്കേണ്ട പഠനനേട്ടങ്ങൾ:

- കമ്പ്യൂട്ടറിലേക്ക് ഡാറ്റ കൈമാറുന്നതിനുള്ള ഇൻപുട്ട് പ്രസ്താവനകൾ പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നു.
- വിവിധ തരത്തിലുള്ള ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഔട്ട്പുട്ട് പ്രസ്താവനകൾ പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നു.
- പ്രശ്ന പരിഹാര സമയത്ത് തീരുമാനങ്ങൾ എടുക്കുന്നതിനുള്ള വിവിധ രീതിയിലുള്ള if പ്രസ്താവനകൾ പ്രയോഗിക്കുന്നു.
- elseif ലാഡും switch പ്രസ്താവനയും താരതമ്യം ചെയ്യുന്നു.
- C++-ലെ വിവിധ ലൂപ്പ് പ്രസ്താവനകളുടെ വ്യത്യാസം തിരിച്ചറിയുന്നു.
- പ്രോഗ്രാമുകളിൽ പ്രശ്ന പരിഹാരത്തിനനുയോജ്യമായ ലൂപ്പുകളെ തെരഞ്ഞെടുക്കുന്നു.
- പ്രശ്ന പരിഹാരത്തിൽ നെസ്റ്റഡ് ലൂപ്പ് എന്ന ആശയം ഉപയോഗിക്കുകയും ഔട്ട്പുട്ട് പ്രവചിക്കുകയും ചെയ്യുന്നു.
- ലൂപ്പുകളിൽ break, continue എന്നീ പ്രസ്താവനകളുടെ സ്വാധീനം തിരിച്ചറിയുകയും ഇവ പ്രോഗ്രാമിന്റെ ഗതിനിയന്ത്രണത്തിൽ എങ്ങനെ സ്വാധീനിക്കുന്നു എന്നു വിശദീകരിക്കുകയും ചെയ്യുന്നു.

C++ ഭാഷയുടെ അടിസ്ഥാന ആശയങ്ങളെക്കുറിച്ച് പതിനൊന്നാം ക്ലാസിൽ ചർച്ച ചെയ്തു കഴിഞ്ഞു. ഇനിയുള്ള പാഠഭാഗങ്ങളിലെ പ്രധാന പഠന നേട്ടങ്ങൾ കൈവരിക്കുന്നതിന് ഇത്തരം ആശയങ്ങളെക്കുറിച്ചുള്ള നല്ല അറിവ് അത്യാവശ്യമാണ്. പതിനൊന്നാം ക്ലാസിൽ നിങ്ങൾ ആർജിച്ച C++ ഭാഷയുടെ ശേഷികളേയും നൈപുണികളേയും കുറിച്ചുള്ള ഓർമപുതുക്കലാണ് ഈ അധ്യായം. ഓരോ ആശയവും അതിനാവശ്യമുള്ള വിശദീകരണത്തോടു കൂടി അവതരിപ്പിച്ചിരിക്കുന്നു. വളരെ പ്രധാനപ്പെട്ട സെലക്ഷൻ പ്രസ്താവനകൾ, ലൂപ്പ് പ്രസ്താവനകൾ എന്നിവ പ്രോഗ്രാമുകളുടെ സഹായത്തോടെയാണ് വിശദീകരിക്കപ്പെട്ടിട്ടുള്ളത്. ചില നൂതന സവിശേഷതകളായ നെസ്റ്റഡ് ലൂപ്പുകൾ, break, continue എന്നീ പ്രസ്താവനകളുടെ ലൂപ്പുകളിലെ പ്രഭാവവും ഈ അധ്യായത്തിൽ പരിചയപ്പെടുത്തുന്നുണ്ട്. C++ പ്രോഗ്രാം നിർമിക്കുന്നതിന് IDE ഉള്ള ഗ്നു കമ്പയിലർ കളക്ഷൻ (GNU Compiler Collection -GCC) ഉപയോഗിക്കുന്നതിനാൽ, പ്രോഗ്രാമിന്റെ ഘടന, ഹെഡർ ഫയലുകൾ ഉൾപ്പെടുത്തുന്ന രീതി, ഡാറ്റാ ഇനങ്ങളുടെ വലിപ്പം തുടങ്ങിയവയെക്കുറിച്ച് കൃത്യമായ ധാരണ നമുക്കുണ്ടാവണം.

1.1. C++ ന്റെ അടിസ്ഥാന ഘടകങ്ങൾ Basics of C++

ഒരു പ്രോഗ്രാമിങ്ങ് ഭാഷ എന്ന രീതിയിൽ C++ -ലെ കാരക്റ്റർ സെറ്റിൽ തുടങ്ങി ടോക്കണുകൾ, പ്രയോഗങ്ങൾ, പ്രസ്താവന എന്നിങ്ങനെയാണ് നാം പഠിച്ചുവന്നത്. വിവിധ ഡാറ്റാ ഇനങ്ങളും അവയുടെ മോഡിഫയലുകളും നാം ചർച്ച ചെയ്തു. പ്രയോഗങ്ങളുടെ (expressions) നിർമ്മാണ സമയത്ത് ഡാറ്റാ ഇനങ്ങളുടെ മാറ്റത്തിന്റെ അനിവാര്യതയും നാം തിരിച്ചറിഞ്ഞു. പട്ടിക 1.1-ൽ ഇത്തരം ഘടകങ്ങളെ കുറിച്ച് ചുരുക്കത്തിൽ പ്രതിപാദിച്ചിരിക്കുന്നു.

C++ ഒരു

കാരക്ടർ സെറ്റ് (Character set)	C++ ഭാഷയുടെ അടിസ്ഥാന ഘടകം. അക്ഷരങ്ങൾ (a-z, A-Z) അക്കങ്ങൾ (0-9), പ്രത്യേക കാരക്ടറുകൾ (#, ; : >}+ തുടങ്ങിയ), വൈറ്റ് സ്പേസ് (സ്പേസ് ബാർ, ടാബ്, ന്യൂലൈൻ). കൂടാതെ പൂജ്യത്തിനും ഇരുന്നൂറ്റി അമ്പത്തഞ്ചിനും (0-255) ഇടയ്ക്ക് ASCII കോഡുള്ള (മൂല്യമുള്ള) മറ്റു പ്രത്യേക കാരക്ടറുകളുകൾ എന്നിങ്ങനെ ഇവയെ തരംതിരിച്ചിരിക്കുന്നു.
ടോക്കണുകൾ (tokens) -	C++ പ്രോഗ്രാമുകളുടെ അടിസ്ഥാന നിർമ്മാണ ഘടകമാണ്. ഒന്നോ അതിലധികമോ കാരക്ടറുകളാൽ നിർമ്മിതം. കീവേർഡ്, ഐഡന്റിഫയർ, ലിറ്ററൽ, പംക്ച്വുവേറ്റർ, ഓപ്പറേറ്റർ എന്നിവയായി തരംതിരിച്ചിരിക്കുന്നു.
കീവേർഡ് (Keywords)-	ഭാഷാ കമ്പയിലറുകൾക്ക് പ്രത്യേക രീതിയിലുള്ള അർഥം കൈമാറുന്ന കരുതിവയ്ക്കപ്പെട്ട വാക്കുകൾ.
ഐഡന്റിഫയറുകൾ (Identifiers)-	മെമ്മറി സ്ഥാനങ്ങൾ, പ്രസ്താവനകൾ, ഫംങ്ഷനുകൾ, ഡാറ്റ ഇനങ്ങൾ തുടങ്ങിയവയെ തിരിച്ചറിയുന്നതിനുള്ള ഉപയോക്തൃ നിർവ്വചിത വാക്കുകൾ. ഐഡന്റിഫയറുകളുടെ സാധുത ഉറപ്പുവരുത്തുന്നതിന് ചില പ്രത്യേക നിയമാവലികൾ അനുവർത്തിക്കേണ്ടതാണ്. വേരിയബിൾ, ലേബൽ, ഫങ്ഷൻ നാമം എന്നിവ ഇതിൽ ഉൾപ്പെടുന്നു.
ലിറ്ററൽ (Literals) -	പ്രോഗ്രാമിന്റെ പ്രവർത്തനസമയത്ത് വിലയ്ക്ക് യാതൊരു മാറ്റവും സംഭവിക്കാത്ത ടോക്കണുകൾ ആണ് ഇവ. സ്ഥിരാംഗങ്ങൾ എന്നും വിളിക്കപ്പെടുന്നു. പൂർണസംഖ്യാ സ്ഥിരാംഗങ്ങൾ, ദശാംശസംഖ്യാ സ്ഥിരാംഗങ്ങൾ, കാരക്ടർ സ്ഥിരാംഗങ്ങൾ, സ്ട്രിങ് സ്ഥിരാംഗങ്ങൾ എന്നിവയായി തരംതിരിച്ചിരിക്കുന്നു. ഡിജിറ്റുകളാണ് പൂർണസംഖ്യാ സ്ഥിരാംഗങ്ങൾ. വേണമെങ്കിൽ + (പ്ലസ്) - (മൈനസ്) എന്നീ ചിഹ്നങ്ങൾ ആദ്യകാരക്ടറുകളായി ചേർക്കാവുന്നതാണ്. ഫ്ളോട്ടിങ് പോയിന്റ് സ്ഥിരാംഗങ്ങളെ ദശാംശ രൂപത്തിലും എക്സ്പൊണന്റ് രൂപത്തിലും സൂചിപ്പിക്കാം. ഓരോ ജോഡി ഒറ്റ ഉദ്ധരണികൾക്കിടയിലുള്ള ഒരു കാരക്ടറിനെ കാരക്ടർ സ്ഥിരാംഗം എന്നു വിളിക്കുന്നു. എസ്കേപ് സീക്വൻസുകൾ എന്ന് വിളിക്കുന്ന ചില പ്രത്യേക കാരക്ടർ സ്ഥിരാംഗങ്ങൾ ഉണ്ട്. പ്രിന്റ് ചെയ്യാൻ കഴിയാത്ത ചില കാരക്ടറുകളായ ന്യൂലൈൻ ('\n') ടാബ് സ്പേസ് ('\t'). പംക്ച്വുവേഷൻ അടയാളങ്ങളായ ഏക ഉദ്ധരണി (' \'), ഇരട്ട ഉദ്ധരണി ('\") ചോദ്യചിഹ്നം ('\? ')തുടങ്ങിയവ ഈ ഗണത്തിൽപെടുന്നു. ഒരു കൂട്ടം കാരക്ടറുകൾ ഇരട്ട ഉദ്ധരണികളുടെ ജോഡിക്കിടയിൽ ഉൾപ്പെടുത്തിയാൽ അവയെ സ്ട്രിങ് സ്ഥിരാഗം എന്നു പറയുന്നു.
ഓപ്പറേറ്ററുകൾ (Operators)-	കമ്പയിലറിനോട് ചില പ്രവർത്തനങ്ങളെക്കുറിച്ച് സൂചിപ്പിക്കുന്ന ചിഹ്നങ്ങളാണിവ. ഇവ ഓരോന്നും ചില പ്രത്യേക പ്രവർത്തനങ്ങൾക്ക് പ്രേരിപ്പിക്കുന്നു. ഓപ്പറേറ്ററുകളുടെ എണ്ണത്തിനനുസരിച്ച് (ഓപ്പറേഷൻ നടത്തപ്പെടുന്ന ഡാറ്റ) ഓപ്പറേറ്ററുകളെ യൂനറി, ബൈനറി, ടെർനറി എന്നിങ്ങനെ തരംതിരിച്ചിരിക്കുന്നു. ഏതുതരം പ്രവർത്തനം നടത്തുന്നു എന്നതനുസരിച്ച് ഇതിനെ മറ്റൊരു രീതിയിൽ തരം തിരിക്കാം. അരിതമെറ്റിക് (+,-,*,/,%), റിലേഷണൽ (<,>,<=, >=, ==, !=), ലോജിക്കൽ (&&,.... ,!) എന്നീ ഓപ്പറേറ്ററുകളാണ് അവ. പ്രവർത്തനത്തിന്റെ ഫലമായി ഈ ഓപ്പ

C++ ഒരു	
	റേറ്ററുകൾ ചില വിലകൾ നൽകുന്നു. ഇൻപുട്ട് നൽകുന്നതിനായി ഗെറ്റ് ഫ്രം (>>), ഔട്ട്പുട്ട് ലഭിക്കുന്നതിനായി പുട്ട്ടു (<<), ഒരു വേരിയബിളിന് വില നൽകുന്നതിനായി (=) എന്നീ പ്രത്യേക ഓപ്പറേറ്ററുകളുമുണ്ട്. മറ്റൊരുതരം ഓപ്പറേറ്ററുകൾ ഒരു അരിത്മറ്റിക് പ്രവർത്തനത്തിനുശേഷം വില നൽകൽ പ്രവർത്തനം നടത്തുന്നു. ഇവയിൽ നൽകൽ ഓപ്പറേറ്ററുകളായ (+=), (=), (*=), (/=), (%=) എന്നിവ അടങ്ങിയിരിക്കുന്നു.
പംക്ച്വുവേറ്ററുകൾ (Punchuators):	പ്രോഗ്രാമിന് ഉപയോഗിക്കുന്ന ഭാഷയിലെ പദഘടനാ രീതിയുടെ പൂർത്തീകരണത്തിന് പ്രത്യേക കാരക്ടറുകളായ കോമ (,), അർദ്ധവിരാമം (;) ഹാഷ് (#), ബ്രാക്കറ്റുകൾ ({ }) തുടങ്ങിയവ ഉപയോഗിക്കുന്നു. ഇവ അർത്ഥപരവും ഘടനാപരമായും കമ്പയിലറിന് പ്രത്യേക സൂചനകൾ നൽകുന്നതിന് ഉപയോഗിക്കുന്നു.
ഡാറ്റ ഇനങ്ങൾ (Data types)-	ഡാറ്റ ഇനവും ഈ ഡാറ്റ ഉപയോഗിക്കുന്ന ബന്ധപ്പെട്ട ഓപ്പറേഷനുകളും തിരിച്ചറിയുന്നതിനുള്ള ഒരുപാധിയാണിവ. ഡാറ്റ ഇനങ്ങളെ അടിസ്ഥാന ഡാറ്റാ ഇനങ്ങൾ, ഉപയോക്തൃ നിർവചിത ഡാറ്റ ഇനങ്ങൾ എന്നിങ്ങനെ തരംതിരിച്ചിരിക്കുന്നു. അടിസ്ഥാന ഡാറ്റ ഇനങ്ങൾ അഭാജ്യ വിലകളെ പ്രതിനിധാനം ചെയ്യുന്നു. int, char, floar, double, void എന്നിവ ഇതിൽ ഉൾപ്പെടുത്തുന്നു. ഇവയിൽ void ഡാറ്റ ഇനമൊഴികെ ബാക്കിയുള്ളവയ്ക്കെല്ലാം അതിന്റേതായ വലിപ്പവും പരിധിയും ഉണ്ട്. ശൂന്യഗണ സൂചകമായാണ് void ഡാറ്റ ഇനം ഉപയോഗിക്കുന്നത് എന്നതിനാൽ അതിന്റെ വലിപ്പം പൂജ്യമാണ്.
ടൈപ്പ് മോഡിഫയറുകൾ (Type modifiers) -	കീവേർഡുകളായ signed, unsigned, short, long എന്നിവയാണ് ടൈപ്പ് മോഡിഫയറുകൾ. മെമ്മറി വലിപ്പവും, അടിസ്ഥാന ഡാറ്റ ഇനം പിന്തുണയ്ക്കുന്ന ഡാറ്റയുടെ പരിധിയും പരിഷ്ക്കരിക്കുന്നതിനായി ഡാറ്റ ഇനത്തോടൊപ്പം ഇവ ഉപയോഗിക്കുന്നു.
പദപ്രയോഗങ്ങൾ (Expressions) -	ഓപ്പറേഷനുകൾ നടത്തുന്നതിനാവശ്യമായ ഓപ്പറേറ്ററുകളും ഓപ്പറന്റുകളും ഉപയോഗിച്ചാണ് പദപ്രയോഗങ്ങൾ നിർമ്മിക്കുന്നത്. ഓപ്പറേഷനുകളെ അടിസ്ഥാനമാക്കി ഇവയെ അരിത്മെറ്റിക് പദപ്രയോഗം, റിലേഷണൽ പദപ്രയോഗം, ലോജിക്കൽ പദപ്രയോഗം എന്നിങ്ങനെ തരംതിരിച്ചിരിക്കുന്നു. അരിത്മെറ്റിക് എക്സ്പ്രഷനെ പൂർണസംഖ്യപദപ്രയോഗം, ദശാംശസംഖ്യപദപ്രയോഗം എന്നിങ്ങനെ രണ്ടായി തരംതിരിച്ചിരിക്കുന്നു. ഒരു പൂർണസംഖ്യ പ്രയോഗത്തിന്റെ ഓപ്പറന്റുകളായി പൂർണസംഖ്യ ഡാറ്റ മാത്രമേ ഉണ്ടാവൂ. കൂടാതെ ഇവയുടെ ഫലം എപ്പോഴും ഒരു പൂർണ സംഖ്യ വിലയുമായിരിക്കും ദശാംശസംഖ്യപദപ്രയോഗങ്ങളിൽ ഓപ്പറന്റുകളും തിരിച്ച് നൽകുന്ന വിലയും ദശാംശസംഖ്യ ആയിരിക്കും. റിലേഷണൽ പദപ്രയോഗങ്ങളിൽ സംഖ്യകളോ അക്ഷരങ്ങളോ ആയ ഡാറ്റ ഓപ്പറന്റുകളും ശരി അല്ലെങ്കിൽ തെറ്റ് തിരികെ ലഭിക്കുന്ന ഫലവുമായിരിക്കും. ലോജിക്കൽ പ്രയോഗങ്ങളിൽ ഓപ്പറന്റുകളായി റിലേഷണൽ പദപ്രയോഗങ്ങളാണ് ഉപയോഗിക്കാറുള്ളത്. ഇത്തരം പദപ്രയോഗങ്ങൾ ശരി അല്ലെങ്കിൽ തെറ്റ് എന്നീ വിലകളാണ് ഉത്തരങ്ങളായി തിരിച്ചുനൽകാറുള്ളത്.

C++ ഒരു	
ഇനം മാറ്റൽ (Tyep conversion) -	ഒരു അരിത്മെറ്റിക് പദപ്രയോഗത്തിൽ വിവിധ തരം ഓപ്പറന്റുകൾ ഉൾപ്പെട്ടിട്ടുണ്ടെങ്കിൽ ഇനം മാറ്റൽ പ്രക്രിയ നടത്തപ്പെടുന്നു. വിലയുടെ നിലവിലുള്ള ഡാറ്റ ഇനത്തെ മറ്റൊന്നിലേക്ക് മാറ്റുന്ന പ്രക്രിയാണ് ഇനം മാറ്റൽ. ആന്തരികമായും, ബാഹ്യമായും ഇനം മാറ്റൽ നടത്താവുന്നതാണ്. ആന്തരിക തരംഗമാറ്റത്തിൽ കമ്പയിലറിനാണ് ഇനം മാറലിന്റെ പൂർണ ഉത്തരവാദിത്തം. ഇത്തരം ഇനം മാറ്റത്തിൽ എല്ലായ്പ്പോഴും മെമ്മറി വലുപ്പം കുറഞ്ഞ ഇനത്തെ ഉയർന്ന ഇനത്തിലേക്കാണ് മാറ്റപ്പെടുന്നത്. അതുകൊണ്ട്തന്നെ ഇത് ഡാറ്റ ഇനം ഉയർത്തൽ എന്നും അറിയപ്പെടുന്നു. ബാഹ്യമായ ഇനം മാറ്റത്തിന്റെ പൂർണ്ണ ഉത്തരവാദിത്തം ഉപയോക്താവിൽ അർപ്പിതമാണ്. ഏത് ഡാറ്റ ഇനത്തിലേക്കാണോ മാറ്റേണ്ടതെന്ന് തീരുമാനിക്കുന്നത് ഉപയോക്താവാണ്. അതിനാൽ ഇത് ടൈപ്പ് കാസ്റ്റിങ്ങ് എന്നും അറിയപ്പെടുന്നു.

പട്ടിക 1.1: Basic elements of C++ language

1.1.1 C++ പ്രോഗ്രാമിലെ വിവിധതരത്തിലുള്ള പ്രസ്താവനകൾ (Various statements in a C++ Program)

സാധാരണയായി പ്രീ പ്രോസസ്സർ നിർദ്ദേശങ്ങളോടു കൂടിയാണ് ഒരു C++ പ്രോഗ്രാം ആരംഭിക്കുന്നത്. ഒരു പ്രോഗ്രാമിൽ ഉപയോഗിച്ചിരിക്കുന്ന മുൻനിർവചിത ഐഡന്റി ഫയറുകളും ഫങ്ഷനുകളേയും കുറിച്ച് ഉള്ള വിവരങ്ങൾ അടങ്ങിയ ഹെഡർ ഫയലുകളെ # include എന്നു തുടങ്ങുന്ന പ്രീ പ്രോസസ്സർ നിർദ്ദേശം ഉപയോഗിച്ച് ബന്ധിപ്പിക്കുന്നു. using namespace പ്രസ്താവന, പ്രീ പ്രോസസ്സർ നിർദ്ദേശത്തിനുശേഷം ഉപയോഗിക്കുന്നു. സാധാരണയായി std എന്ന മുൻനിർവചിത namespace ഉപയോഗിച്ച് cin, cout തുടങ്ങിയ ഐഡന്റിഫയറുകളുടെ ഉപയോഗപരിധി നിർവചിക്കുന്നു. അതിനുശേഷം main() ഫങ്ഷൻ ആരംഭിക്കുന്നു. ഒരു C++ ന് അത്യന്താപേക്ഷിതമായ ഫങ്ഷനാണ് ഇത്. പ്രോഗ്രാമിന്റെ പ്രവർത്തനം ആരംഭിക്കുന്നതും അവസാനിക്കുന്നതും ഇതിലാണ്. പ്രഖ്യാപന പ്രസ്താവനകളും, കൃത്യം പരിഹരിക്കുവാൻ ആവശ്യമായ ഒരു കൂട്ടം പ്രവർത്തന പ്രസ്താവനകളും ഇതിൽ അടങ്ങിയിരിക്കുന്നു. ഈ പ്രസ്താവനകളെ കുറിച്ച് നമുക്ക് കൂടുതൽ മനസിലാക്കാം.

പ്രഖ്യാപന പ്രസ്താവനകൾ (Declaration statement)

വേരിയബിൾ എന്നത് മെമ്മറിയിലെ ഒരിടത്തിനെ സൂചിപ്പിക്കുവാനും, പ്രോഗ്രാമുകൾക്ക് ഡാറ്റയെ പ്രതിനിധികരിക്കുവാനും ഉള്ളതാണ്. ഒരു പ്രോഗ്രാമിൽ അവ ഉപയോഗിക്കുന്നതിന് മുമ്പ് തന്നെ നിർവചിക്കുകയും അവയുടെ ഡാറ്റ ഇനം പ്രഖ്യാപിക്കേണ്ടതുമാണ്. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ വേരിയബിൾ പ്രഖ്യാപനങ്ങൾക്ക് ഉദാഹരണങ്ങളാണ്.

```
int n, sum;
float rad, area;
signed int a,b,c;
```

വേരിയബിൾ പ്രഖ്യാപിക്കുന്നതിനോടൊപ്പം തന്നെ അവയ്ക്കുള്ള വില നൽകുന്നതാണ് ചുവടെ കാണിച്ചിരിക്കുന്നത്.

```
int n=10;
```

ഇങ്ങനെയുള്ള പ്രസ്താവനകളെ വേരിയബിളുകൾക്ക് പ്രാരംഭവില നൽകൽ പ്രസ്താവന എന്ന് അറിയപ്പെടുന്നു. വേരിയബിളിൽ നൽകിയിരിക്കുന്ന വില പ്രോഗ്രാമിൽ മറ്റ് വില കൊണ്ട് മാറ്റം

വരാം. എന്നാൽ ചുവടെ ചേർത്തിരിക്കുന്ന പ്രസ്താവന വേരിയബിൾ നൽകിയിരിക്കുന്ന വില മാറ്റം വരുത്തുവാൻ അനുവദിക്കുന്നില്ല.

```
const int n=10;
```

ഇവിടെ പ്രാരംഭവില നൽകൽ ആരംഭിച്ചിരിക്കുന്നത് const ആക്സസ് മോഡിഫയർ കൊണ്ടാണ്, ഈ കീവേർഡ് വേരിയബിളിലേ വില മാറ്റം വരുത്തുന്നത് നിയന്ത്രിക്കുന്നു.

ഇൻപുട്ട് പ്രസ്താവന (Input statement)

എക്സ്ട്രാക്ഷൻ ഓപ്പറേറ്റർ, ഗെറ്റ് ഫ്രം എന്നീ പേരുകളിലറിയപ്പെടുന്ന >> എന്ന ഓപ്പറേറ്റർ C++ ലഭ്യമാക്കിയിരിക്കുന്നു. ഇത് ഒരു ബൈനറി ഓപ്പറേറ്റർ ആയതിനാൽ ഇതിന് രണ്ട് ഓപെറന്റുകളുടെ ആവശ്യകത ഉണ്ട്. ആദ്യത്തെ ഓപെറന്റ് ആയ മുൻ നിർവചിത ഐഡന്റിഫയർ cin കീബോർഡിനെ ഇൻപുട്ട് ഒബ്ജക്റ്റ് ആയി തിരിച്ചറിയുന്നു. രണ്ടാമത്തേ ഓപെറന്റ് നിർബന്ധമായും ഒരു വേരിയബിൾ ആയിരിക്കണം. നമുക്ക് ഒന്നിൽ കൂടുതൽ ഇൻപുട്ട് സ്വീകരിക്കുന്നതിനായി അതേ പ്രസ്താവനയിൽ ഒന്നില് കൂടുതൽ നവേരിയബിൾ നമുക്ക് ഉപയോഗിക്കാം. ഒരു സാധുവായ ഉദാഹരണം ഒരു ചുവടെ നൽകിയിരിക്കുന്നു.

```
cin>>rad;
cin>>a>>b>>c;
```

ഔട്ട്പുട്ട് പ്രസ്താവന (Output statement)

ഔട്ട്പുട്ട് പ്രവർത്തനം നടത്തുന്നതിനായി ഇൻസേർഷൻ ഓപ്പറേറ്റർ അല്ലെങ്കിൽ പുട്ട് ടു ഓപ്പറേറ്റർ എന്ന് അറിയപ്പെടുന്ന ഓപ്പറേറ്ററുകൾ C++ ലഭ്യമാക്കിയിരിക്കുന്നു. ഇത് ഒരു ബൈനറി ഓപ്പറേറ്റർ ആണ്. ഇതിൽ ആദ്യത്തേ ഓപെറന്റ് ആയ മുൻനിർവചിത ഐഡന്റിഫയർ cout മോണിറ്ററിനെ ഔട്ട്പുട്ട് ഒബ്ജക്റ്റ് ആയി തിരിച്ചറിയുന്നു. രണ്ടാമത്തെ ഓപെറന്റ് ഒരു സ്ഥിരാംഗമോ, ഒരു വേരിയബിളോ അല്ലെങ്കിൽ ഒരു പദപ്രയോഗമോ ആകാം. ഔട്ട്പുട്ട് പ്രസ്താവനകൾക്ക് ചില ഉദാഹരണങ്ങൾ ചുവടെ നൽകിയിരിക്കുന്നു.

```
cout<< "hello";
cout<< area;
cout<< 25;
cout<< a+b+c;
cout<< "Sum of " << n << "numbers = " << sum;
```

വില നൽകൽ പ്രസ്താവന (Assignment statement)

മെമ്മറിയുടെ ഒരിടത്തിൽ (വേരിയബിൾ) നിർദ്ദിഷ്ട ഡാറ്റ സൂക്ഷിക്കുവാൻ വില നൽകൽ പ്രസ്താവന ഉപയോഗിക്കുന്നു. '=' ഉൾപ്പെട്ടിട്ടുള്ള പ്രസ്താവനകളെ വില നൽകൽ പ്രസ്താവന എന്ന് അറിയപ്പെടുന്നു. ഇത് ഒരു ബൈനറി ഓപ്പറേറ്റർ ആണ്. = ന്റെ ഇടത്വശത്തെ ഓപെറന്റ് ഒരു വേരിയബിൾ ആയിരിക്കണം. = ന് ശേഷമുള്ള ഓപ്റെന്റ് തരത്തിലുള്ള ഒരു സ്ഥിരാംഗമോ, ഒരു വേരിയബിളോ, ഒരു അക്കങ്ങളുടെ പദപ്രയോഗമോ ആകാം. സാധുവായ വില നൽകൽ പ്രസ്താവനകളുടെ ഉദാഹരണം ചുവടെ നൽകിയിരിക്കുന്നു.

```
n = 253;
area = 3.14*rad*rad;
a = b = c;
```

പ്രത്യേക വില നൽകൽ ഓപ്പറേറ്ററുകളെ അരിത്തമെന്റിക് വില നൽകൽ ഓപ്പറേറ്റർ എന്ന് അറിയപ്പെടുന്നു. +=, -=, *=, /- ഉം % = ആണിവ. ഇവയെല്ലാം ബൈനറി ഓപ്പറേറ്റേഴ്സ് ആണ്. ഇതിന്റെ വലതുഭാഗത്ത് ഉള്ള ഓപ്റെന്റ് ഒരു വേരിയബിൾ ആയിരിക്കണം. ഇവയുടെ പ്രവർത്തനം ചുവടെ വിശദീകരിച്ചിരിക്കുന്നു.

```
n+=2;          // n=n+2;  എന്നതിനു തുല്യം
a*=b;          // a=a*b;  എന്നതിനു തുല്യം
sum-=n%10;     // sum=sum-n%10;  എന്നതിനു തുല്യം
```

ഓപ്പറേറ്ററുകളായ ++ ഉം -- ഉം C++ ലെ പ്രത്യേക തരം ഓപ്പറേറ്ററുകൾ ആണ്. ഒരു തരത്തിൽ വില നൽകൽ പ്രസ്താവനകൾക്ക് സമാനമാണ്. ഇവ യൂനറി ഓപ്പറേറ്ററുകളാണ്. അവയുടെ ഓപെറന്റ് വേരിയബിൾ ആയിരിക്കണം. ചുവടെ നൽകിയിരിക്കുന്ന പ്രസ്താവനകൾ ഇവയുമായി ബന്ധപ്പെട്ട പ്രവർത്തനങ്ങൾ വിശദമാക്കിയിരിക്കുന്നു.

```
n++;           // n=n+1;  എന്നതിനു തുല്യം
a--;           // a=a-1;  എന്നതിനു തുല്യം
```

ഈ ഓപെറേറ്ററുകൾക്ക് രണ്ട് തരം പതിപ്പുകൾ ആണ് പോസ്റ്റ്ഫിക്സ് രൂപമെന്നും, പ്രിഫിക്സ് രൂപമെന്നും a++; ഉം a--; ഇവ യഥാക്രമം ഇൻക്രിമെന്റിന്റെയും ഡിക്രിമെന്റിന്റെയും പോസ്റ്റ് ഫിക്സ് രൂപം ++a; ഉം --a ഉം പ്രിഫിക്സ് രൂപവും ഏതു രൂപത്തിലായാലും ++ ഓപ്പറേറ്റർ ഓപെറന്റ് വേരിയബിളിന്റെ കൂടെ ഒന്ന് കൂട്ടുകയും അതിന്റെ ഫലം അവിടെത്തന്നെ സൂക്ഷിക്കുകയും ചെയ്യുന്നു.

ഈ രണ്ട് പതിപ്പുകളും വില നൽകൽ പ്രസ്താവനകളുടെ കൂടെയോ ഔട്ട്പുട്ട് പ്രസ്താവനകളുടെ കൂടെ ഉപയോഗിക്കുമ്പോൾ വ്യത്യസ്ത ഫലമാകും കിട്ടുക. 'a' എന്നത് 5 എന്ന വില സൂക്ഷിച്ചിട്ടുള്ള ഒരു ഇന്റിജർ വേരിയബിളാണ് എന്നും, b എന്നത് മറ്റൊരു വേരിയബിൾ ആണ് എന്നും സങ്കൽപ്പിക്കുക. b = a++ എന്ന പ്രസ്താവനയുടെ പ്രവർത്തനത്തിനു ശേഷം b യിലെ വില 5 ഉം a യിലെ വില 6 ഉം ആകുന്നു. അതായത്, b=a++; എന്നത് b=a; a=a+1 ഉം എന്ന തുടർച്ചയായ പ്രസ്താവനകൾക്ക് സമാനമാണ്. ആയതിനാൽ ഇത്തരം വർദ്ധന നൽകൽ രീതിയെ ഉപയോഗിക്കുക. വ്യത്യാസം വരുത്തുക രീതി എന്നറിയപ്പെടുന്നു.

എന്നാൽ b=++a; എന്ന പ്രസ്താവന a=a+1; b=a; എന്ന തുടർച്ചയായ പ്രസ്താവനകൾക്ക് സമാനമാണ്. ആയതിൽ a യുടെയും b യുടെയും വില 6 ആയി മാറുന്നു. ഈ രീതിയെ വ്യത്യാസം വരുത്തുക ഉപയോഗിക്കുക എന്നറിയപ്പെടുന്നു.

അതുപോലെ cout<<1--; എന്ന പ്രസ്താവന 5 എന്ന് പ്രദർശിപ്പിക്കുകയും, എന്നാൽ a യുടെ വില 4 ആയി മാറുകയും ചെയ്യുന്നു. ഇതിന് തുല്യമായ പ്രസ്താവന എന്നത് cout<<a; a=a-1 എന്നാണ്. എന്നാൽ cout<<--a; എന്നതിന് സമാനമായ പ്രസ്താവനകൾ a=a-1; cout<<a എന്നാണ്.

ഇൻപുട്ട്, ഔട്ട്പുട്ട്, വില നൽകൽ ഓപറേറ്ററുകൾ (>>,<< ഉം =) തുടങ്ങിയവ ഇത്തരം പ്രസ്താവനകളിൽ ഒന്നിലധികം തവണ വരുന്നതായി കാണാം. ഇതിനെ കാസ്കാഡിംഗ് (cascading) എന്ന് അറിയപ്പെടുന്നു. ഓരോ വിഭാഗത്തിലെയും കാസ്കാഡിംഗ് ഇൻപുട്ട്, ഔട്ട്പുട്ട് വില നൽകൽ ഓപ്പറേറ്റർ തുടങ്ങിയവയുടെ ഉദാഹരണം ചുവടെ നൽകിയിരിക്കുന്നു.

```
cin >> a >> b >> c;
cout << "Sum of " << n << "numbers = " << sum;
a = b = c;
```

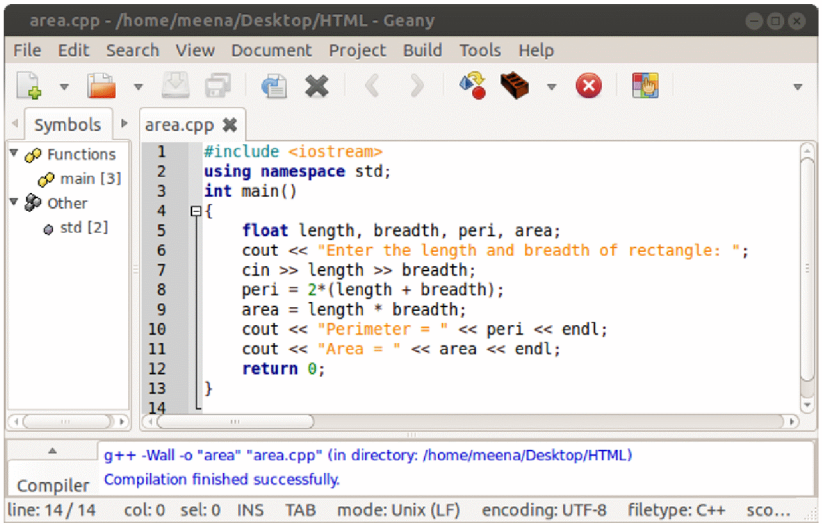
1.1.2 ഒരു C++ പ്രോഗ്രാമിലെ പ്രോഗ്രാമിന്റെ ഘടന (Structure of a C++ program)

ഒരു C++ പ്രോഗ്രാമിന്റെ അടിസ്ഥാന ഘടന പ്രോഗ്രാം 1.1 കാണിച്ചിരിക്കുന്നു. ഒരു ചതുരത്തിന്റെ നീളവും വീതിയും സ്വീകരിച്ച് അതിന്റെ വിസ്തീർണ്ണവും ചുറ്റളവും കണ്ടെത്തുന്നു.

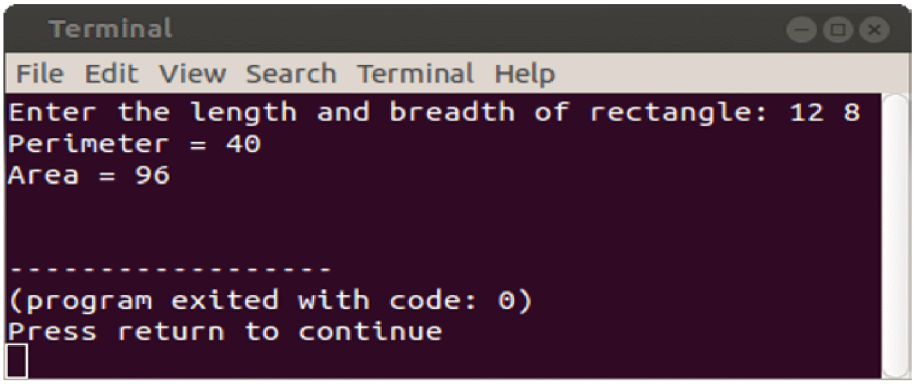
പ്രോഗ്രാം:- 1.1 ഒരു ചതുരത്തിന്റെ വിസ്തീർണവും ചുറ്റളവും കണ്ടുപിടിക്കാൻ

```
#include <iostream>
using namespace std;
int main()
{
    float length, breadth, peri, area;
    cout << "Enter the length and breadth of rectangle: ";
    cin >> length >> breadth;
    peri = 2*(length + breadth);
    area = length * breadth;
    cout << "Perimeter = " << peri << endl;
    cout << "Area = " << area << endl;
    return 0;
}
```

cin; ഉം cout ഉം എന്ന ഐഡന്റിഫയറുകൾ ഉപയോഗിക്കുന്നതിനാൽ പ്രോഗ്രാം 1.1 iostream എന്ന ഹെഡർ ഫയൽ ഉപയോഗിക്കുന്നു. cin ഉം cout ഉം പ്രത്യേകമായി ഉപയോഗിക്കുന്നതിന് രണ്ടാമത്തേ വരി അത്യന്താപേക്ഷിതമാണ്. using namespace std എന്ന പ്രസ്താവന cin ഉം cout ഉം main പ്രോഗ്രാമിൽ ലഭ്യമാക്കുന്നു. GCC യിൽ main ( ) എന്ന ഫങ്ഷൻ നാമത്തിന് മുന്നിലായി int എന്ന ഡാറ്റ തരം ഉപയോഗിക്കുന്നു. float ഡാറ്റ തരം ഉപയോഗിച്ച് വേരിയബിളുകൾ പ്രഖ്യാപിച്ചിരിക്കുന്നു. ഇൻപുട്ട്, ഔട്ട്പുട്ട് ഓപ്പറേറ്ററുകളുടെ കാസ്കാഡിംഗ് ഉപയോഗിച്ചിരിക്കുന്നു. വില നൽകൽ പ്രസ്താവനയിൽ സൂത്രവാക്യങ്ങൾ ഉപയോഗിച്ച് പ്രശ്നം നിർദ്ദാരണം ചെയ്തിരിക്കുന്നു. ഓരോ ഫലത്തിനു ശേഷം പുതിയ വരിയിൽ ഫലം ദൃശ്യമാകുന്നതിനായി '/n' ന് പകരം endl ഉപയോഗിച്ചിരിക്കുന്നു. ചിത്രം 1.1 (a) പ്രോഗ്രാം 1.1ന്റെ Geany IDE യിൽ ഉള്ള സ്ക്രീനിന്റെ ചിത്രവും ചിത്രം 1.1 (b) പ്രോഗ്രാമിന്റെ പ്രവർത്തനഫലം കാണിച്ചിരിക്കുന്നു.



ചിത്രം 1.1(a) : ജീനിIDE യിൽ പ്രോഗ്രാം 1.1



ചിത്രം 1.1(b) : ജീനിIDE യുടെ ടെർമിനൽ വിൻഡോയിൽ പ്രോഗ്രാം 1.1 ന്റെ ഔട്ട്പുട്ട്

ഒരു C++ പ്രോഗ്രാം എഴുതുമ്പോൾ, നമ്മൾ 'using namespace std' എന്ന പ്രസ്താവന ഉപയോഗിക്കുന്നു. എന്തുകൊണ്ട്?

ഒരു പ്രോഗ്രാമിൽ ഒരേ വ്യാപ്തിയിൽ ഒരേ പേരിലുള്ള ഒന്നിലധികം ഐഡന്റിഫയറുകൾ (വേരിയബിളുകൾ അല്ലെങ്കിൽ ഫങ്ഷനുകൾ) ഉണ്ടായിരിക്കാൻ പാടില്ല. നമ്മുടെ വീട്ടിൽ രണ്ടോ അതിലധികമോ ആളുകൾക്ക് (അല്ലെങ്കിൽ ജീവജാലങ്ങൾക്ക്) ഏരേ പേരുണ്ടാവില്ല. അങ്ങനെയുണ്ടെങ്കിൽ തീർച്ചയായും വീട്ടിനുള്ളിൽ അവരെ പേരു കൊണ്ട് തിരിച്ചറിയുക എന്നത് വിഷമകരമാകും. അതുകൊണ്ട് നമ്മുടെ വീട്ടിന്റെ പരിധിയിൽ ഒര പേരും അനന്യമായിരിക്കണം. എന്നാൽ നമ്മുടെ അയൽപക്കത്തെ വീട്ടിൽ സമാനമായ പേരുള്ള ഒരാൾ അല്ലെങ്കിൽ ജീവജാലം ഉണ്ടായിരിക്കാം. അതാത് പരിധിക്കുള്ളിൽ വ്യക്തികളെ പേരു കൊണ്ട് തിരിച്ചറിയുന്നതിന് ഇത് യാതൊരു ആശയകുഴപ്പവുമുണ്ടാക്കില്ല. പക്ഷെ പുറമെ നിന്നൊരു വ്യക്തിയ്ക്ക് പേരു മാത്രം ഉപയോഗിച്ച് കൊണ്ട് ഇവരെ തിരിച്ചറിയാൻ കഴിയില്ല. അതിനാൽ വീട്ടുപേരും കൂടി പരാമർശിക്കേണ്ടതുണ്ട്.

നെയിം സ്പേസ് എന്ന ആശയം വീട്ടുപേരിനു സമാനമാണ്. ഒരു പ്രത്യേക നെയിം സ്പേസുമായി വ്യത്യസ്ത ഐഡന്റിഫയറുകൾ ബന്ധപ്പെട്ടിരിക്കുന്നു. ഓരോ ഇനവും വ്യത്യസ്തമായിരിക്കുന്ന ഒരു ഗണത്തിന്റെ പേരാണിത്. വേരിയബിളുകൾക്കും ഫങ്ഷനുകൾക്കുമായി പ്രത്യേകം നെയിം സ്പേസുകൾ സൃഷ്ടിക്കുന്നതിന് ഉപയോക്താവിനു അനുവാദമുണ്ട്. ഒരു നെയിം സ്പേസിനു പേരു കൊടുക്കാൻ നമുക്ക് ഒരു ഐഡന്റിഫയർ ഉപയോഗിക്കാം. പ്രോഗ്രാമിങ്ങിൽ ഉപയോഗിക്കുന്ന ഘടകങ്ങളെ ഏത് നെയിം സ്പേസിൽ തിരയണമെന്ന് `using` എന്ന കീവേർഡ് സാങ്കേതികമായി കംപൈലറിനോട് പറയുന്നു. C++ ൽ `stadard` എന്നതിന്റെ ചുരുക്കെഴുത്താണ് `std.cin, cout` തുടങ്ങിയ മറ്റ് പല ഒബ്ജക്ടുകളും നിർവചിച്ചിട്ടുള്ള ഒരു നെയിം സ്പേസ് ആണിത്. അതിനാൽ ഒരു പ്രോഗ്രാമിൽ ഇവ ഉപയോഗിക്കണമെങ്കിൽ `std::cin, std::cout` എന്ന മാതൃക നാം പിന്തുടരേണ്ടതാണ്. `using namespace std` എന്ന പ്രസ്താവന പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നതിലൂടെ ഇത്തരത്തിലുള്ള വിശദമായ പരാമർശങ്ങൾ ഒഴിവാക്കാവുന്നതാണ്. അത്തരമൊരു സാഹചര്യത്തിൽ കംപൈലർ `cin, cout, endl` മുതലായവയ്ക്കായി ഈ നെയിം സ്പേസിൽ തിരയുന്നു. `cin, cout, endl` അല്ലെങ്കിൽ അതുപോലെയുള്ളവ എപ്പോഴൊക്കെ ഒരു C++ പ്രോഗ്രാമിൽ കമ്പ്യൂട്ടർ കാണുന്നുവോ, അവയെ `std::cin, std::cout, std::endl` എന്നിങ്ങനെ വ്യാഖ്യാനിക്കുന്നു.

`using namespace std` എന്ന പ്രസ്താവന യഥാർഥത്തിൽ ഒരു ഫങ്ഷനെ കൂട്ടിച്ചേർക്കുന്നില്ല, `cin, cout, endl` തുടങ്ങി സമാനമായവയേ കൂട്ടിച്ചേർക്കുന്നത് `include<iostream>` എന്ന പ്രസ്താവനയാണ്.

നിങ്ങളുടെ പുരോഗതി അറിയുക

1. C++ പ്രോഗ്രാമിൽ ടോക്കൺ എന്നാലെന്ത്?
2. താഴെ നൽകിയിരിക്കുന്ന ടോക്കണുകൾ ശ്രദ്ധിക്കുക. ഓരോന്നും ഏത് വിഭാഗത്തിൽ ഉൾപ്പെടുന്നു എന്ന് കണ്ടെത്തുക.

   i. number ii. 23.98 iii. "\0" iv. cin v. ++
   vi. void vii. '\\' viii. ; ix. = x. a
3. C++ പ്രോഗ്രാമിൽ # include പ്രസ്താവനയുടെ പങ്കെന്ത്?
4. cin >> 25; എന്ന പ്രസ്താവനയിലെ തെറ്റ് കണ്ടെത്തുക.
5. C++ ന്റെ ടൈപ്പ് മോഡിഫയറുകൾ പട്ടികപ്പെടുത്തുക?

1.2 നിയന്ത്രണ പ്രസ്താവനകൾ (Control statements)

പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യാൻ ഒരു പ്രോഗ്രാമിന്റെ തനത് പ്രവർത്തനക്രമത്തെ ആയി മാറ്റം വരുത്തേണ്ടതായി വരും. ഇത് ചിലപ്പോൾ തെരെഞ്ഞെടുക്കൽ, ഒഴിവാക്കൽ, ഒന്നോ അധിലധികമോ പ്രസ്താവനകളുടെ ആവർത്തിച്ചുള്ള നിർദ്ധാരണം എന്നിവയാകാം. ഇത്തരം തീരുമാനങ്ങൾ കൈക്കൊള്ളുന്നത് ചില നിബന്ധനകളെ അടിസ്ഥാനമാക്കിയാണ്. C++ ഈ ആവശ്യം നിറവേറ്റുന്നത് നിയന്ത്രണ പ്രസ്താവനകളുടെ സഹായത്തോടെയാണ്. നിയന്ത്രണ പ്രസ്താവനകളെ രണ്ടായി തരം തിരിക്കാം. (1) തീരുമാനമെടുക്കൽ/തെരെഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ (2) ആവർത്തന പ്രസ്താവനകൾ എങ്ങനെ ഇത്തരം പ്രസ്താവനകൾ പ്രശ്നങ്ങൾ പരിഹരിക്കുവാൻ സഹായകമാകുന്നു എന്ന് നമുക്ക് കാണാം.

1.2.1 തീരുമാനങ്ങൾ എടുക്കുന്നതിനുള്ള പ്രസ്താവനകൾ (Selection statements)

നിബന്ധനകളെ ആധാരമാക്കി ഒരു കൃത്യത്തെ തിരെഞ്ഞെടുക്കുന്നതിനായി C++ രണ്ടു തരം പ്രസ്താവനകൾ ആണ അപയോഗിക്കുന്നത്. If, Switch എന്നിവയാണ് ഇവ. If പ്രസ്താവനയ്ക്ക് രണ്ടു തരം പതിപ്പുകൾ ആണ് ഉള്ളത്: ലളിതമായ if, if-else ഉം else. if .... ചുവടെ ചേർത്തിരിക്കുന്ന പ്രോഗ്രാം ഈ പ്രസ്താവനകളുടെ പ്രവർത്തന രീതികൾ വിവരിക്കുന്നു.

പ്രോഗ്രാം:- 1.2 തന്നിരിക്കുന്ന മൂന്നു സ്കോറിൽ നിന്ന് ഏറ്റവും ഉയർന്ന CE സ്കോർ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    short int ce1, ce2, ce3, final_ce;
    cout<<"Enter three CE scores: ";
    cin>>ce1>>ce2>>ce3;
    if (ce1>ce2)
      final_ce=ce1;//നിബന്ധന ശരി എങ്കിൽ പ്രവർത്തിക്കുന്നു.
    else
      final_ce=ce2;//നിബന്ധന തെറ്റ് എങ്കിൽ പ്രവർത്തിക്കുന്നു.
```

```
    if (ce3>final_ce) final_ce=c3;//No else block for this if
    cout<<"Final CE Score is "<<final_ce;
    return 0;
}
```

പ്രോഗ്രാം 1.2 Short int ഡാറ്റ ഇനമാണ് വേരിയബിൾ പ്രഖ്യാപിക്കുവാനായി ഉപയോഗിച്ചിരിക്കുന്നത്. Gcc-യിൽ int ഡാറ്റ ഇനം 4 ബൈറ്റ്സ് മെമ്മറി ഉപയോഗിക്കുമ്പോൾ short 2 ബൈറ്റ്സ് മാത്രമേ ഉപയോഗിക്കുന്നുള്ളൂ. പ്രോഗ്രാം if-else പ്രസ്താവനയും ലളിതമായ if പ്രസ്താവനയും ഉപയോഗിക്കുന്നു. പ്രാരംഭത്തിൽ ce1>ce2 എന്ന നിബന്ധന പ്രസ്താവന പ്രവർത്തിക്കുന്നു. ഈ പ്രസ്താവന ശരിയെങ്കിൽ final-ce യിലേക്ക് ce1 ന്റെ വില നൽകുന്നു, അത് അല്ലെങ്കിൽ ce2 ന്റെ വിലയാകും final-ce യിലെക്ക് നൽകുക. ഇതിനുശേഷം ce3 യുടെ വില final-ce യിലുള്ള വിലയുമായി താരതമ്യം ചെയ്യുന്നു. ce3 യിലെ സ്കോർ വലുതാണെങ്കിൽ, final-ce യിൽ ce3 യുടെ വില ശേഖരിക്കുകയും, അല്ലെങ്കിൽ ഒരു മാറ്റവും final-ce യിൽ ഉണ്ടാകുന്നതുമില്ല.

else if ലാഡർ എന്നത് ബഹുമുഖ ശാഖകൾ ഉള്ള പ്രസ്താവനയാണ്. പ്രോഗ്രാം 1.3 else-if ലാഡറിന്റെ പ്രവർത്തനം വിവരിക്കുന്നു. ഒരു ക്യാരക്റ്റർ സ്വീകരിക്കുകയും അത് അക്ഷരമാണോ, അക്കമാണോ, അതല്ല മറ്റ് ക്യാരക്റ്റർ ആണോ എന്ന് ഔട്ട്പുട്ട് നൽകുന്നു.

പ്രോഗ്രാം 1.3 ക്യാരക്റ്റർ ചെറിയ അക്ഷരമാണോ, വലിയ അക്ഷരമാണോ, അക്ഷരമാണോ, മറ്റു ക്യാരക്റ്റർ ആണോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout<<"Enter a character: ";
    cin>>ch;
    if (ch>='A' && ch<='Z')
        cout<<"Uppercase letter";
            else if (ch>='a' && ch<='z')
                cout<<"Lowercase letter";
                    else if (ch>='0' && ch<='9')
                        cout<<"Digit";
                            else
                                cout<<"Other character";
    return 0;
}
```

പ്രോഗ്രാം 1.3 വ്യത്യസ്ത നാല് വിലകളിൽ നിന്ന് ഒന്നിനെ തിരഞ്ഞെടുക്കുവാൻ else if ലാഡറോ else if സ്റ്റെയർ കെയ്സോ ഉപയോഗിക്കുന്നു.

ചില അവസരങ്ങളിൽ ഇന്റിജർ തുല്യത നിബന്ധനകൾ ഉപയോഗിച്ചാണ് തീരുമാനം എടുത്തിരിക്കുന്നത്.

else if ലാഡറിന് പകരം switch പ്രസ്താവന ഉപയോഗിക്കാവുന്നതാണ്. ക്യാരക്റ്റർ ഡാറ്റ ഇനവും ഒരു ഇൻറ്റിജർ ആയതിനാൽ, തുല്യതാ പരിശോധനയ്ക്ക് അവ ഉപയോഗിക്കാവുന്നതാണ്. പ്രോഗ്രാം 1.4 ഈ ആശയം വിശദീകരിക്കുന്നു. ഈ പ്രോഗ്രാം a, b, c d എന്നീ നാല് അക്ഷരങ്ങളിൽ നിന്ന് ഒന്ന് സ്വീകരിക്കുന്നു. 'a' എന്ന അക്ഷരമാണ് എങ്കിൽ 'Abacus' എന്ന് വാക്ക് പ്രദർശിപ്പിക്കുന്നു. അതുപോലെ 'Binary' എന്ന് 'b' യ്ക്കും 'computer' എന്ന് 'c' യ്ക്കും 'debugging' എന്ന് 'd' യ്ക്കും പ്രദർശിപ്പിക്കുന്നു. തന്നിരിക്കുന്ന പ്രശ്നത്തിന് default പ്രസ്താവനയുടെ ആവശ്യമില്ല. തന്നിരിക്കുന്ന നാല് ക്യാരക്റ്റർ അല്ലാതെ മറ്റൊന്നു ഇൻപുട്ട് നൽകിയാൽ പ്രോഗ്രാം, ഈ അവസ്ഥയിൽ പ്രതികരിക്കുകയില്ല. ആയതിനാൽ കൂടുതൽ ഉപയോക്ത സൗഹൃദമാക്കുവാനായി default case കൂടി പ്രോഗ്രാമിൽ ഉപയോഗിച്ചിരിക്കുന്നു.

പ്രോഗ്രാം 1.4 തന്നിരിക്കുന്ന ക്യാരക്റ്ററിനു പകരം ഒരു വാക്ക് പ്രദർശിപ്പിക്കുന്നു

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout<<"Enter a, b, c or d: ";
    cin>>ch;
    switch(ch)
    {
      case 'a': cout<<"Abacus";
                break;
      case 'b': cout<<"Binary";
                break;
      case 'c': cout<<"Computer";
                break;
      case 'd': cout<<"Debugging";
                break;
      default : cout<<"Invalid input!!";
    }
    return 0;
}
```

പ്രോഗ്രാം 1.4 ബഹുമുഖ ശാഖ എന്ന ആശയം ഉപയോഗിച്ചിരിക്കുന്നു. വ്യത്യസ്ത അവസ്ഥയിൽ നിന്ന് ഒരെണ്ണം മാത്രം പ്രവർത്തിക്കാനും തിരഞ്ഞെടുക്കൽ എന്നത് switch ലെ പ്രയോഗം നൽകുന്ന വിലയും, ഏതെങ്കിലും case പ്രസ്താവനകളിൽ കൊടുത്തിരിക്കുന്ന സ്ഥിരാംഗത്തിന് തുല്യമാണോ എന്നതിനെ അടിസ്ഥാനമാക്കിയാണ്. ഒരു സ്ഥിരാഗവുമാകും തുല്യത കണ്ടെത്തിയില്ലെങ്കിൽ default case ആയിരിക്കും പ്രവർത്തിക്കുക.

നമുക്കു ചെയ്യാം.

പ്രോഗ്രാം 1.4 ലെ switch പ്രസ്താവനയേ മാറ്റി else if ലാഡർ ഉപയോഗിക്കുക. പ്രോഗ്രാം 1.4 ൽ നിന്ന് break പ്രസ്താവന ഒഴിവാക്കിയാൽ എന്താകും ഔട്ട്പുട്ട്?

പ്രോഗ്രാം 1.3 ൽ നിന്ന് else-if ഗ്രാഡറിന് പകരം switch ഉപയോഗിക്കുവാൻ കഴിയില്ല. എന്തുകൊണ്ട്?

കണ്ടീഷണൽ ഓപ്പറേറ്റർ (conditional operator (?))

ഇത് C++ ലെ ഒരു ടേർണറി ഓപ്പറേറ്റർ ആണ് ഇത് ഉപയോഗിക്കുന്നതിന് മൂന്ന് ഓപ്പറന്റുകൾ ആവശ്യമാണ്. പ്രോഗ്രാം 1.2 ൽ ഉപയോഗിച്ചിരിക്കുന്ന if-else പ്രസ്താവനയേ ഒഴിവാക്കി ചുവടെ കൊടുത്തിരിക്കുന്നത് പോലെ എഴുതാം.

final-ce = (ce1>ce2)? ce1:ce2;

മൂന്നു സ്കോറിൽ നിന്ന് വലിയ സ്കോർ കണ്ടെത്തുന്നതിനായി കണ്ടിഷണൽ ഓപ്പറേറ്ററിന്റെ നെസ്റ്റിംഗ് ഉപയോഗിക്കാവുന്നതാണ്.

final - ce = (ce1>ce2) ? (ce1>ce3) ? (cel:ce3) : [(ce2>ce3)?ce2:ce3);

ആവർത്തന പ്രസ്താവനകൾ (Looping statements)

C++ ൽ മൂന്ന് തരം ആവർത്തന പ്രസ്താവനകൾ ഉണ്ട് while, for ഉം do-while. ഒരു ആവർത്തന പ്രസ്താവനയക്ക് 4 ഘടകങ്ങൾ ആണ് ഉള്ളത്. പ്രാരംഭ വില നൽകൽ (init) പരിശോധന പ്രയോഗം (Test expression), പരിഷ്ക്കരിക്കൽ പ്രസ്താവന (Updation statement), ലൂപ്പിന്റെ ചട്ടക്കൂട് (Body of loop). ആവർത്തിക്കപ്പെടേണ്ട പ്രസ്താവനകൾ ഉപയോഗിച്ച് ലൂപ്പിൻറെ ചട്ടക്കൂട് രൂപപ്പെടുത്തുന്നു. പരിശോധന പ്രയോഗത്തിന്റെ വില ശരിയായി തുടരുന്നത് വരെ ലൂപ്പ് പ്രവർത്തിക്കുന്നു. പരിശോധ പ്രയോഗത്തിൽ ഉപയോഗിച്ചിരിക്കുന്ന വേരിയബിളിനെ ലൂപ്പ് നിയന്ത്രണ വേരിയബിൾ (Loop contraol variable) എന്ന് അറിയപ്പെടുകയും ഇതിന്റെ പ്രാരംഭവില പ്രാരംഭ വില നൽകൽ പ്രസ്താവന വഴി ലഭ്യമാകുകയും ചെയ്യുന്നു. പരിഷ്ക്കരിക്കൽ പ്രസ്താവന ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വിലയിൽ മാറ്റം വരുത്തുന്നു ഈ പ്രസ്താവന അടുത്ത ആവർത്തനത്തിന് മുന്നേ പ്രവർത്തിക്കുന്നു.

ലൂപ്പിംഗ് പ്രസ്താവന അഥവാ ആവർത്തന പ്രസ്താവനകളെ രണ്ട് രീതിയിൽ തരം തിരിക്കാം ആഗമന നിയന്ത്രമ ലൂപ്പ് എന്നും ബഹിർഗമന നിയന്ത്രണ ലൂപ്പ് എന്നും. ആഗമന നിയന്ത്രണ ലൂപ്പിൽ പരിശോധന പ്രയോഗം ലൂപ്പിന്റെ പ്രർത്തനത്തിന് മുൻപ് തന്നേ പരിശോധിക്കപ്പെടുന്നു. ലൂപ്പിന് ഉള്ളിലേക്ക് കടക്കണമെങ്കിൽ പരിശോധ പ്രയോഗത്തിന്റെ ഫലം ശരിയാകണം. while ഉം for ഉം ആഗമന നിയന്ത്രണ ലൂപ്പിന് ഉദാഹരണങ്ങൾ ആണ്. എന്നാൽ ബഹിർഗമന നിയന്ത്രണ ലൂപ്പിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ച ശേഷമാണ് നിബന്ധകൾ പരിശോധിക്കുന്നത്. ആയതിനാൽ ഒരു തവണയെങ്കിലും സഹിർഗമന നിയന്ത്രണ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ചിരിക്കും. do-while ലൂപ്പ് ഈ വിഭാഗത്തിൽപ്പെടുന്നതാണ്.

ലൂപ്പിലെ ഘടകങ്ങളായ പ്രാരംഭവില നൽകൽ, പരിശോധ പ്രയോഗം, പരിഷ്കരിക്കൽ പ്രസ്താവന എന്നിവ ഒരുമിച്ചാമ്മ for പ്രസ്താവനയിൽ നൽകിയിരിക്കുന്നത്. എന്നാൽ while ഉം do-whild ലും പ്രാരംഭവില നൽകൽ പ്രയോഗം ലൂപ്പ് ന് മുമ്പും പരിഷ്കരിക്കൽ പ്രസ്താവന ലൂപ്പിന്റെ ചട്ടക്കൂടിനുള്ളിലും നൽകിയിരിക്കുന്നു. പരിശോധ പ്രയോഗം while എന്ന വാക്കിന്റെ കൂടെ തന്നെ നൽകിയിരിക്കുന്നു. ഇവയുടെ പ്രവർത്തനം മനസ്സിലാക്കുവാനായി ചില പ്രോഗ്രാമുകൾ നോക്കാം.

പ്രോഗ്രാം 1.5 ഒരു നമ്പറിലെ അക്കങ്ങളുടെ തുക കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int num, sum=0, dig;
    cout<<"Enter a number: ";
    cin>>num;
    while (num>0)
    {
        dig=num%10;
        sum=sum+dig;
        num=num/10;
    }
    cout<<"Sum of the digits of the input number = "<<sum;
    return 0;
}
```

പ്രോഗ്രാം 1.5 ൽ num എന്ന ലൂപ്പ് നിയന്ത്രണ വേരിയബിൾ ആയി പരിശോധന പ്രയോഗത്തിൽ ഉപയോഗിച്ചിരിക്കുന്നു. ഇൻപുട്ട് പ്രസ്താവന വഴി ഈ വേരിയബിളിന്റെ പ്രാരംഭ വില നൽകൽ നടത്തിയിരിക്കുന്നു. ലൂപ്പ് ചട്ടക്കൂടിൽ ആണ് ഇതിന്റെ പരിഷ്കരിക്കൽ പ്രസ്താവന നൽകിയിരിക്കുന്നത്. ഇൻപുട്ട് പ്രസ്താവന വഴി ലഭിക്കുന്ന അക്കം പോസിറ്റിവ് നമ്പർ അല്ലെങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയില്ല.

for പ്രസ്താവന ഉപയോഗിച്ച് നിർമിച്ച ലൂപ്പ് ഉള്ള മറ്റൊരു പ്രോഗ്രാം നമുക്ക് കാണാം.

പ്രോഗ്രാം 1.6 ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ തുക കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int n, sum=0;
    cout<<"Enter the limit: ";
    cin>>n;
    for(int i=1; i<=n; i++)
        sum=sum+i;
    cout<<"Sum of the first "<<n<<" natural numbers = "<<sum;
    return 0;
}
```

പ്രോഗ്രാം 1.6 ൽ i എന്നത് ഒരു ലൂപ്പ് നിയന്ത്രണ വേരിയബിൾ ആയും ഇതിനെ ഒരു വില നൽകൽ പ്രസ്താവന വഴി പ്രാരംഭ വില നൽകൽ നടത്തിയിരിക്കുന്നു. ഇതിന്റെ പരിഷ്കരിക്കൽ വില കൂട്ടൽ പ്രസ്താവന വഴി ആണ് ചെയ്തിരിക്കുന്നത്.

ഈ പ്രോഗ്രാമിന് ലൂപ്പ് ആധാരമാക്കിയുള്ള എണ്ണൽ ആണ് നടക്കുന്നത്. for പ്രസ്താവന ഇത് പോലുള്ള അവസരങ്ങളിൽ for പ്രസ്താവനകൾ ആണ് ഏറ്റവും അനുയോജ്യം. നേരത്തെ സൂചിപ്പിച്ചത് പോലെ do while ലൂപ്പ് ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവിശ്യം എങ്കിലും പ്രവർത്തിപ്പിച്ചിരിക്കും. ഇതിന്റെ ഉപയോഗം താഴത്തേ പ്രോഗ്രാമിൽ വിശദീകരിച്ചിരിക്കുന്നു.

പ്രോഗ്രാം 1.7 ഒരു കൂട്ടം വിദ്യാർഥികളുടെ ഉയര അളവിന്റെ ശരാശരി കണക്കാക്കുന്നു.

```
#include <iostream>
using namespace std;
int main()
{
    float hgt, sum=0, avg_hgt;
    short n=0;
    char ch;
    do
    {
        cout<<"Enter the height: ";
        cin>>hgt;
        n++;
        sum=sum+hgt;
        cout<<"Any more student (Y/N)? ";
        cin>>ch;
    }while (ch=='Y' || ch=='y');
    avg_hgt=sum/n;
    cout<<"Average Height = "<<avg_hgt;
    return 0;
}
```

പ്രോഗ്രാം 1.7 ന്റെ ലൂപ്പ്ചട്ടക്കൂട് ഉപയോക്താവ് 'y' അല്ലെങ്കിൽ 'y' എന്ന് നൽകുന്നിടത്തോളം അവർത്തിച്ച് പ്രവർത്തിച്ച് കൊണ്ടിരിക്കും.

നമുക്കു ചെയ്യാം.

പ്രോഗ്രാം 1.5 ന്റെ ലൂപ്പ് പ്രസ്താവന for ഉം do-while ഉം ഉപയോഗിച്ച് മാറ്റിയെടുക്കുക.

പ്രോഗ്രാം 1.6 ന്റെ ലൂപ്പ് പ്രസ്താവന whild ഉം do-while ഉം ഉപയോഗിച്ച് മാറ്റിയെടുക്കുക.

പ്രോഗ്രാം 1.7 ന്റെ ലൂപ്പ് പ്രസ്താവന for ഉം while ഉം ഉപയോഗിച്ച് മാറ്റിയെഴുതുക.

നിങ്ങളുടെ പുരോഗതി അറിയുക

1. C++ ലെ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ ഏതെല്ലാം?
2. ലൂപ്പ് പ്രസ്താവനകളുടെ നാല് ഘടകങ്ങൾ ഏന്തൊക്കെയാണ്?
3. ആഗമന നിയന്ത്രണ ലൂപ്പ് പ്രസ്താവനയ്ക്ക് ഉദാഹരണം എഴുതുക.
4. കണ്ടിഷനൽ ഓപ്പറേറ്റർ (?) ക്ക് തുല്യമായ നിയന്ത്രണ കൈമാറ്റ പ്രസ്താവന ഏതാണ്.
5. എല്ലാ തരത്തിലുള്ള switch പ്രസ്താവനകളും if ന്റെ വിവിധ രൂപത്തിലുള്ള പ്രസതാവനകൾ ഉപയോഗിച്ച് മാറ്റിയെഴുതാം. ശരിയോ തെറ്റോ എന്ന് പ്രസ്താപിക്കുക.

1.3 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 പൂർത്തിയാക്കിയതിനുശേഷം മിനുട്ട് സൂചി അടുത്ത സ്ഥാനത്തേക്ക് മാറുന്നു. മിനുട്ട് സൂചിയുടെ ഓരോ സ്ഥാനത്തിനും അനുസൃതമായി സെക്കന്റ് സൂചി ഭ്രമണം 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.

ചിത്രം 1.2 ൽ നൽകിയിരിക്കുന്നതു പോലെ മിനിറ്റു സൂചി മാറ്റില്ലാതെ തുടർന്നു കൊണ്ട് സെക്കന്റ് സൂചിയുടെ മൂല്യം 0 ൽ നിന്ന് 59 ലേക്ക് മാറുന്നു. സെക്കന്റ് സൂചിയുടെ മൂല്യം 59 ൽ എത്തിയാൽ പിന്നെ മാറ്റം മിനിറ്റു സൂചിയിലായിരിക്കും. മിനിറ്റു സൂചിയിലെ മാറ്റത്തിനു ശേഷം സെക്കന്റ് സൂചിയുടെ മൂല്യം പൂജ്യത്തിലേക്കു വീണ്ടുമെത്തുന്നു.

ചിത്രം. 1.2: ഡിജിറ്റൽ വാച്ചിലൂടെ നെസ്റ്റിങ് ലൂപ്പ് എന്ന ആശയം

C++ ലെ എല്ലാ ലൂപ്പുകളും നെസ്റ്റിങ് അനുവദിക്കുന്നു. നെസ്റ്റഡ് ലൂപ്പിന്റെ പ്രവർത്തനം മനസ്സിലാക്കുന്നതിനുള്ള ഒരു ഉദാഹരണം ചുവടെ ചേർത്തിരിക്കുന്നു.

```
for( i=1; i<=2; ++i)          Outer loop
{
      for(j=1; j<=3; ++j)     Inner loop
      {
```

```
            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 ആന്തരികലൂപ്പ് 3 തവണ പ്രവർത്തിക്കുന്നു. ഓരോ തവണയും `j<=3` എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും ശരിയായതിനാൽ ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു.

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

The first 1 is of `i` and the second 1 is of `j`

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.

Iterations	Outer loop (i)	Inner loop (j)	Output
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

ഠമയഹല 1.2: ഋഃലരൗശേീി ീള മ ിലലേറ ഹീീു

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 7.8: ത്രികോണാകൃതിയിൽ നക്ഷത്രചിഹ്നം പ്രദർശിപ്പിക്കുന്നതിന്.

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

നമുക്കു ചെയ്യാം

`cout<<i;` എന്ന പ്രസ്താവനയ്ക്ക് പകരം `cout<<'*';` എന്ന പ്രസ്താവന ആണെങ്കിൽ പ്രോഗ്രാം 1.8 ന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും?

അതുപോലെ, `cout<<j;` എന്ന പ്രസ്താവന ആണെങ്കിൽ എന്തായിരിക്കും ഔട്ട്പുട്ട്.

1.4 ജംമ്പ് പ്രസ്താവനകൾ (Jump Statements)

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

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

1.4.1 goto പ്രസ്താവന (goto statement)

goto പ്രസ്താവന ഉപയോഗിച്ച് പ്രോഗ്രാം നിയന്ത്രണത്തെ ഫംങ്ഷനിലെ ഏതു സ്ഥല ത്തേക്കും മാറ്റാൻ സാധിക്കും. ഒരു goto പ്രസ്താവനയുടെ ലക്ഷ്യസ്ഥാനം ലേബൽ (ഒരു ഐഡന്റിഫയർ) ഉപയോഗിച്ച് അടയാളപ്പെടുത്തുന്നു. goto പ്രസ്താവനയുടെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

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

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:
    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 പ്രസ്താ വനക്കു ശേഷം എത്തുന്നു. സ്ട്രക്ചേർഡ് പ്രോഗ്രാമിങ് goto ന്റെ ഉപയോഗം പ്രോത്സാഹി പ്പിക്കുന്നില്ല

1.4.2 ബ്രേക്ക് പ്രസ്താവന (break statement)

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 കഴി ഞ്ഞു. 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.

നെസ്റ്റഡ് ലൂപ്പ് സാധാരണയായി i=1, i=2, i=3 എന്നീ വിലകൾക്ക് അനുസരിച്ച് പ്രവർത്തിക്കുന്നു. iയുടെ 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.

1.43 കൺഡിന്യു പ്രസ്താവന (continue statement)

continue പ്രസ്താവന മറ്റൊരു ജംമ്പ് 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.

നമുക്കു ചെയ്യാം.

പട്ടിക 7.3 break, continue എന്നീ പ്രസ്താവനകൾ തമ്മിലുള്ള താരതമ്യം കാണിക്കുന്നു. പട്ടിക പൂർത്തിയാക്കുക.

break പ്രസ്താവന	continue പ്രസ്താവന
•	• ലൂപ്പിന്റെ കൂടെ മാത്രം ഉപയോഗിക്കുന്നു.
• ബ്ലോക്കിലെ അവശേഷിക്കുന്ന പ്രസ്താവനകൾ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം സ്വിച്ചിനോ ലൂപ്പിനോ പുറത്തേക്കു കൊണ്ടു വരുന്നു.	• ....
•	• പരിശോധന പ്രസ്താവനയുടെ വില തെറ്റാകുമ്പോൾ മാത്രം പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന് പുറത്തു കൊണ്ടു പോകുന്നു.

പട്ടിക 1.3: break, continue എന്നീ പ്രസ്താവനകൾ തമ്മിലുള്ള താരതമ്യം

നെസ്റ്റഡ് ലൂപ്പു ആവശ്യമുള്ള ഒരു പ്രോഗ്രാം നമുക്ക് എഴുതാം.

പ്രോഗ്രം 1.9: തന്നിരിക്കുന്ന സംഖ്യ അഭാജ്യ സംഖ്യയാണോ അല്ലെയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include<iostream>
using namespace std;
int main()
{   short int n, i, flag;
    cout<<"Prime numbers below 100 are...\n";
    for(n=2; n<=100; n++)  //Outer loop
    {   flag=1;
        for(i=2; i<=n/2; i++)     //Inner loop
            if(n%i==0)
            {   flag=0;
                break;//Takes the control outside the iiner loop
            }
        if(flag==1) cout<<n<<'\t';
    }
    return 0;
}
```

പ്രോഗ്രാം 1.8 ൽ, പുറത്തെ ലൂപ്പിലൂടെ n എന്നാ വേരിയബിൾ 2 മുതൽ 100 വരെയുള്ള വിലകൾ പ്രോഗ്രാമിന് നൽകുന്നു. ഈ ഓരോ വിലകളും അഭാജ്യമാണോ എന്ന് അകത്തെ ലൂപ്പ് ഉപയോഗിച്ച് പരിശോധിക്കുന്നു. 2 മുതൽ n/2 വരെയുള്ള ഏതെങ്കിലും ഒരു വില n ന്റെ ഘടകമായി വന്നാൽ, flag എന്നാ വേരിയബിളിന്റെ വില 1-ൽ നിന്ന് 0-മായി മാറ്റിക്കൊണ്ട് അകത്തെ ലൂപ്പ് പ്രവർത്തനം അവസാനിപ്പിക്കുന്നു. അകത്തെ ലൂപ്പിന്റെ പ്രവർത്തനത്തിന് ശേഷവും flag ന്റെ വില 1 ആയി നിലനിൽക്കുന്ന n നെ അഭാജ്യ സംഖ്യയായി പ്രദർശിപ്പിക്കുന്നു.

നമുക്ക് സംഗ്രഹിക്കാം

C++ ഭാഷയുടെ അടിസ്ഥാന ആശയങ്ങളെക്കുറിച്ചുള്ള അറിവുകൾ നമ്മൾ ഓർമ്മ പുതിക്കിയല്ലോ. ക്യാരക്ടർ സെറ്റ്, ടാറ്റ തരങ്ങൾ, തരങ്ങളുടെ പരീക്ഷണങ്ങൾ, പദപ്രയോഗങ്ങൾ, തരങ്ങളുടെ പരിവർത്തനം തുടങ്ങിയവയുടെ ആശയങ്ങൾ ക്യാപ്സ്യൂൾ രൂപത്തിൽ ഇവിടെ അവതരിപ്പിച്ചു. ഉദാഹരണസഹിതം വിവിധ തരത്തിലുള്ള C++ പ്രസ്താവനകളും അനുസ്മരിച്ചു. വിവിധ തരത്തിലുള്ള നിയന്ത്രണ കൈമാറ്റ് പ്രസ്താവനകൾ പ്രോഗ്രാമുകളുടെ സഹായത്താൽ ചുരുക്ക രൂപത്തിൽ വിശദീകരിക്കുകയും, നെസ്റ്റേഡ് ലൂപ്പുകളും, രണ്ടു ജമ്പ് പ്രസ്താവനകൾ ആയ ബ്രേക്ക്, കണ്ടിന്യൂ തുടങ്ങിയവ പുതിയ ആശയങ്ങളായും അവതരിപ്പിച്ചു. ഈ പുസ്തകത്തിലെ അധ്യായം രണ്ടു, മൂന്നു എന്നിവയിലെ ആശയങ്ങൾ മനസിലാക്കുവാൻ ഇവയെക്കുറിച്ചുള്ള വ്യക്തമായ ധാരണ ആവശ്യമാണ്.

നമുക്ക് പരിശീലിക്കാം

1. 100 നും 200 നും ഇടയ്ക്കുള്ള പാലിൻഡ്രോം അക്കങ്ങൾ (മുന്നോട്ടും പിന്നോട്ടും ഒരു വായിക്കാൻ കഴിയുന്നവ) പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.
2. 1000 താഴെയുള്ള ആംസ്ട്രോങ് അക്കങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക
3. 1000 താഴെയുള്ള പെർഫക്റ്റ് അക്കങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.
4. ഒരു അക്കത്തിന്റെ ഗുണനപട്ടിക പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.
5. ഉപഭോക്താക്കളുടെ വൈദ്യുതി ബിൽ പ്രദർശിപ്പിക്കുന്നതിനുള്ള C++ പ്രോഗ്രാം എഴുതുക.

   300 യൂണിറ്റ് വരെ : Rs. 5.00/- 1 യൂണിറ്റിന്
   350 യൂണിറ്റ് വരെ : Rs. 5.70/- 1 യൂണിറ്റിന്
   400 യൂണിറ്റ് വരെ : Rs. 6.10/- 1 യൂണിറ്റിന്
   500 യൂണിറ്റ് വരെ : Rs. 6.70/- 1 യൂണിറ്റിന്
   500 യൂണിറ്റ് മുകളിൽ : Rs. 7.50/- 1 യൂണിറ്റിന്

   എത്ര ഉപഭോക്താക്കളുടെ വിവരങ്ങളും ഉൾപ്പെടുത്താനുള്ള സൗകര്യം പ്രോഗ്രാമിൽ ലഭ്യമായിരിക്കണം.

നമുക്ക് വിലയിരുത്താം

1. താഴെ നൽകിയിരിക്കുന്ന കോഡ് ശകലത്തിലെ തെറ്റ് കണ്ടെത്തുക.

```
for (short i=1; i<5; ++i)
    for (i=5; i>0; --i)
        cout<<i<<"\t";
```

2. break, continue പ്രസ്താവനകൾ താരതമ്യം ചെയ്യുക

3. താഴെ നൽകിയിരിക്കുന്ന പ്രോഗ്രാം ശകലത്തിന്റെ ഔട്ട്പുട്ട് എഴുതുക.

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

4. ചുവടെ ചേർത്തിരിക്കുന്ന ഔട്ട്പുട്ട് ലഭിക്കുന്നതായി നെസ്റ്റഡ് ലൂപ്പ് ഉപയോഗിച്ച് പ്രോഗ്രാം എഴുതുക.

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

5. ചുവടെ ചേർത്തിരിക്കുന്ന C++ കോഡ് ശ്രദ്ധിക്കുക

```
for (n=1; n<5; ++n)
{
    cout<<i;
    if (i==2) continue;
    if (i%3==0) break;
    cout<<"Hello";
}
```

ഈ കോഡിന്റെ ഔട്ട്പുട്ട് തിരഞ്ഞെടുക്കുക

a. 1Hello2Hello3Hello4Hello
b. 1Hello2Hello3
c. 1Hello23
d. 1Hello23Hello

6. ചുവടെ ചേർത്തിരിക്കുന്ന C++ കോഡ് ശകലം ശ്രദ്ധിക്കുക. ലൂപ്പിങ് പ്രസ്താവന ഉപയോഗിച്ച് മാറ്റിയെഴുതുക.

```
        cin>>n;
loop:   r=n%10;
        s=s*10+r;
        n=n/10;
        if (n!=0) goto loop;
        cout<<s;
```

7. C++ ലെ കാരക്ടർ കോൺസ്റ്റന്റ് അല്ലാത്തത് തിരഞ്ഞെടുക്കുക?

   a. `'\t'` b. `'a'` c. `'9'` d. `'9a'`

8. ചുവടെ ചേർത്തിരിക്കുന്നതിൽ അസാധുവായ ഐഡന്റിഫയർ തിരഞ്ഞെടുത്ത് കാരണം എഴുതുക.

   a. `unsigned` b. `cpp` c. `2num` d. `cout`

9. `switch` പ്രസ്താവനകളിൽ `break` ഉപയോഗിച്ചില്ലെങ്കിൽ എന്ത് സംഭവിക്കും?