10

ഫങ്ഷനുകൾ

പ്രധാന ആശയങ്ങൾ

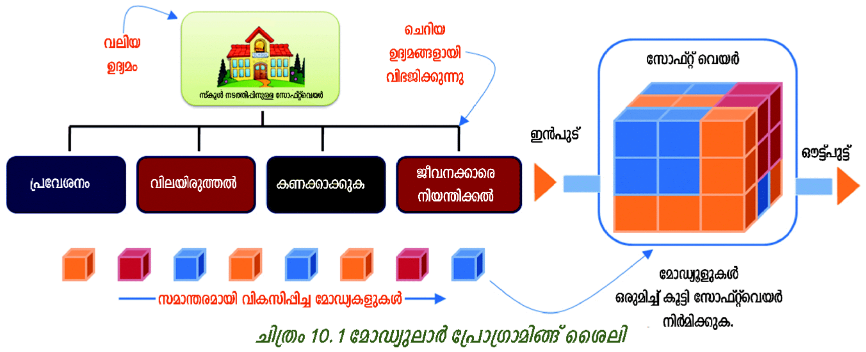
- മോഡുലാർ പ്രോഗ്രാമിങ്ങിന്റെ ആശയം
- C++ ലെ ഫങ്ഷനുകൾ
- മുൻനിർവചിത ഫങ്ഷനുകൾ
    - സ്ട്രിങ്ങ് ഫങ്ഷനുകൾ
    - ഗണിത ഫങ്ഷനുകൾ
    - ക്യാരക്ടർ ഫങ്ഷനുകൾ
    - I/O കൺവേർഷൻ കൈകാര്യം ഫങ്ഷനുകൾ
- ഉപയോക്തൃ നിർമിത ഫങ്ഷനുകൾ
    - ഉപയോക്തൃ നിർമിത ഫങ്ഷനുകൾ നിർമിക്കുന്നു.
    - ഫങ്ഷനുകളുടെ പ്രോട്ടോ ടൈപ്പ്
    - ഫങ്ഷനുകളുടെ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

P5T6F3

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

10.1 മോഡുലാർ പ്രോഗ്രാമിങ്ങിന്റെ ആശയം (Concept of Modular programming)

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



ചിത്രം 10.1 മോഡ്യുലാർ പ്രോഗ്രാമിങ്ങ് ശൈലി

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

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

മോഡ്യുലാർ പ്രോഗ്രാമിങ്ങിന്റെ മേൻമകൾ

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

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

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

ഘട്ടം 1 ന്റെയും ഘട്ടം 2 ന്റെയും ഉത്തരങ്ങൾ കണ്ടുപിടിക്കുന്നതിന് പ്രത്യേകം ലൂപ്പുകൾ

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

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

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

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

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

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

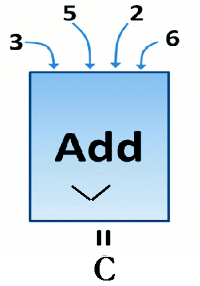
10.2 C++ ലെ ഫങ്ഷനുകൾ (Functions in C++)

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

ചിത്രം 10. 2 കോഫി ഉണ്ടാക്കുന്ന യന്ത്രത്തിന്റെ പ്രവർത്തനം.

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

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



ചിത്രം 10.3: കൂട്ടുന്നതിന് ഉള്ള ഫങ്ഷൻ

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

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

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

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

10.3 മുൻകൂട്ടി നിർവചിച്ച ഫങ്ഷനുകൾ (Predefind Functions)

വിവിധ ഉദ്യമങ്ങൾക്ക് വേണ്ടി C++ ധാരാളം ഫങ്ഷനുകൾ ലഭ്യമാക്കുന്നു. ഏറ്റവും സാധാരണയായി ഉപയോഗിക്കുന്ന ഫങ്ഷനുകൾ മാത്രം നാം ചർച്ചചെയ്യുന്നു. ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുമ്പോൾ അവയിൽ ചിലതിന് നിശ്ചയിച്ചിട്ടുള്ള ഉദ്യമം നിർവഹിക്കുന്നതിന്, ഡാറ്റ ആവശ്യമാണ്. ഫങ്ഷന്റെ പേരിന് ശേഷം പാരന്തസീസ് എന്ന ഒരു ജോഡി ബ്രാക്കറ്റുകൾക്കുള്ളിൽ ഉപയോഗിക്കുന്ന ഈ ഡാറ്റയെ പരാമീറ്ററുകൾ അല്ലെങ്കിൽ ആർഗ്യുമെന്റുകൾ എന്ന് നാം വിളിക്കുന്നു.
ഉദ്യമങ്ങൾ നിർവഹിച്ചതിനു ശേഷം ഫലങ്ങൾ നൽകുന്ന ചില ഫങ്ഷനുകൾ ഉണ്ട്. ഈ ഫലം ഫങ്ഷൻ തിരിച്ചു നൽകുന്ന വില എന്നറിയപ്പെടുന്നു. ചില ഫങ്ഷനുകൾ ഒരു വിലയും തിരിച്ചു തരുന്നില്ല. പകരം അവയുടെ പ്രത്യേക ഉദ്യമം നിർവ്വഹിക്കുന്നു. തുടർന്നുള്ള ഭാഗങ്ങളിൽ സ്ട്രിങ്ങുകൾ കൈകാര്യം ചെയ്യുന്നതിനും ഗണിത പ്രക്രിയകൾ നടത്തുന്നതിനും ക്യാരക്ടർ ഡാറ്റയിൽ പ്രവർത്തിക്കുന്നതിനുമുള്ള ഫങ്ഷനുകൾ നാം ചർച്ചചെയ്യും. ഇത്തരം ഫങ്ഷനുകൾ ഉപയോഗിക്കുമ്പോൾ അതുമായി ബന്ധപ്പെട്ട ഹെഡർ ഫയലുകൾ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ടതാണ്.

10.3.1 സ്ട്രിങ് ഫങ്ഷനുകൾ (String Functions)

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

a. `strlen()`

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

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

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

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

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

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

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

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

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

മുകളിലത്തെ നിർദേശം 8 എന്ന വില തിരിച്ച് നൽകുകയും അത് nൽ സംഭരിക്കുകയും ചെയ്യുന്നു.

b. strcpy()

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

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

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

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

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

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

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

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

c. strcat()

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

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

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

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

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

d. strcmp()

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

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

മൂന്ന് വ്യത്യസ്ത സന്ദർഭങ്ങളിൽ ഫങ്ഷൻ താഴെ കൊടുത്തിരിക്കുന്ന ഏതെങ്കിലും വിലകൾ തിരിച്ച് നൽകുന്നു.

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

താഴകൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഈ ഫങ്ഷന്റെ പ്രവർത്തനം കാണിക്കുന്നു.

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

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

e. strcmpi()

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

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

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

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

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

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

സ്ട്രിങ് കൈകാര്യം ചെയ്യുന്നതിനുള്ള ഫങ്ഷനുകൾ അടങ്ങിയ ഹെഡർ ഫയൽ

10.3.2 ഗണിത ഫങ്ഷനുകൾ (Mathametical Functions)

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

a. abs()

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

```
int abs(int)
```

ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്ന രീതി ഉദാഹരണമായി താഴെ നൽകുന്നു.

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

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

b. sqrt()

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

```
double sqrt(double)
```

താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം പരിശോധിക്കുക. ഇത് 5 എന്ന വില പ്രദർശിപ്പിക്കും.

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

c. pow()

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

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

താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം ഇതിന്റെ പ്രവർത്തനം വിവരിക്കുന്നു.
ഉദാഹരണം:

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

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

d. sin()

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

വാക്യഘടന ആണ്

```
double sin(double)
```

∠30⁰ (കോൺ 30⁰) യുടെ സൈൻ (Sin ഫങ്ഷൻ ഉപയോഗിച്ച്) വില താഴെപറയുന്ന രീതിയിൽ കണ്ടുപിടിക്കാം.

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

മുകളിൽ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കോഡ് .4999770 എന്ന് പ്രദർശിപ്പിക്കും.

e. cos ()

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

```
double cos(double)
```

∠ 30⁰ യുടെ കൊസൈൻ വില കണ്ടുപിടിക്കുന്നതിന് താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ഉപയോഗിക്കാം.

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

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

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

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

(കർണ്ണം)2 = (പാദം)2 + (ലംബം)2

പ്രോഗ്രാം 10.2 ഗണിത ഫങ്ഷനുകൾ ഉപയോഗിച്ച് ഒരു മട്ടത്രികോണത്തിന്റെ വശങ്ങളുടെ നീളം കണ്ടുപിടിക്കുന്നത്

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

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

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

ഡിഗ്രിയിലുള്ള കോൺ റേഡിയനിലേക്ക് മാറ്റുന്നു

മുകളിലുള്ള പ്രോഗ്രാമിന്റെ മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

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

10.3.3 ക്യാരക്ടർ ഫങ്ഷനുകൾ (Character functions)

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

a. isupper()

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

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

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

ഉദാഹരണം:

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

താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകൾ പരിഗണിക്കുക.

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

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

b. islower()

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

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

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

ഉദാഹരണം:

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

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

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

c. isalpha()

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

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

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

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

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

എന്നാൽ താഴെകൊടുത്തിരിക്കുന്ന പ്രസ്താവന 1 പ്രദർശിപ്പിക്കുന്നു കാരണം തന്നിരിക്കുന്ന ക്യാരക്ടർ ഒരു അക്ഷരമാണ്.

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

d. isdigit()

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

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

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

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

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

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

e. isalnum()

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

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

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

താഴെ കൊടുത്തിരിക്കുന്ന ഓരോ പ്രസ്താവനയും പ്രവർത്തിക്കുമ്പോൾ 1 തിരിച്ച് നൽകുന്നു.

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

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

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

f. toupper()

തന്നിരിക്കുന്ന ക്യാരക്ടർ വലിയ അക്ഷരത്തിലേക്ക് മാറ്റുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു ഇതിന്റെ വാക്യഘടന ആണ്

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

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

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

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

എന്നാൽ താഴെ തന്നിരിക്കുന്ന പ്രസ്താവനയുടെ ഔട്ട്പുട്ട് 'A' തന്നെ ആയിരിക്കും.

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

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

g. tolower()

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

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

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

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

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

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

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

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

പ്രോഗ്രാം 10.3 തന്നിരിക്കുന്ന സ്ട്രിങ്ങിലെ വിവിധ തരത്തിലുള്ള ക്യാരക്ടറുകൾ എണ്ണുന്നതിന്.

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

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

i യുടെ വില ക്യാരക്ടറിലേക്ക് പോയിന്റ് ചെയ്യുമ്പോൾ ലൂപ്പ് അവസാനിക്കും

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

ഒരു മാതൃകാ ഔട്ട്പുട്ട് താഴെ നൽകുന്നു.

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

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

10.3.4: പരിവർത്തന ഫങ്ഷനുകൾ (Conversion functions)

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

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

a. itoa()

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

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

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

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

മുകളിൽ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കോഡ് "2345" എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കും.

b. atoi()

ഒരു സ്ട്രിങ് വിലയെ പൂർണസംഖ്യയാക്കി മാറ്റുന്നതിന് ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നു.

ഫങ്ഷന്റെ വാക്യഘടനയാണ്

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

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

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

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

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

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

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

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

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

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

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

പ്രോഗ്രാം 10.4 ജനനതീയതി തീയതി രൂപത്തിൽ പ്രദർശിപ്പിക്കുന്നതിന്

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

പ്രോഗ്രാം 10.4 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

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

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

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

setw()

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

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

മുകളിലത്തെ കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
hello   friends
```

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

നമുക്ക് ചെയ്യാം

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

Function	Usage	Syntax	Example	Output

സ്വയം പരിശോധിക്കാം

1. മോഡ്യുളാർ പ്രോഗ്രാമിങ് എന്നാൽ എന്ത്?
2. C++ ലെ ഒരു ഫങ്ഷൻ എന്നാലെന്ത്?
3. ക്യാരക്ടർ ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നതിന് ആവശ്യമായ ഹെഡർ ഫയലിന്റെ പേര് എഴുതുക.
4. അനുവദിച്ച സ്ഥലത്ത് ഡാറ്റ പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഫങ്ഷന്റെ പേര് എഴുതുക.
5. കൂട്ടത്തിൽ ചേരാത്തത് എടുത്തെഴുതി അതിനുള്ള കാരണം നൽകുക.
   (a) `strlen()` (b) `itoa()` (c) `strcpy()` (d) `strcat()`

10.4 ഉപയോക്തൃ നിർവചിത ഫങ്ഷനുകൾ (User defind functions)

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

ഫങ്ഷൻ നിർവചനത്തിന്റെ വാക്യഘടന താഴെ കൊടുത്തിരിക്കുന്നതാണ്.

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

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

(i) ഫങ്ഷൻ ഹെഡറിൽ ഏത് ഡാറ്റ ഇനം ഉപയോഗിക്കും?

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

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

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

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

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

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

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

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

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

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

ഫങ്ഷൻ 1	*ഫങ്ഷൻ 2*

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

ഫങ്ഷൻ 3	*ഫങ്ഷൻ 4*

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

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

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

പട്ടിക 10.1 : ഫങ്ഷനുകളുടെ വിശകലനം

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

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

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

പ്രോഗ്രാമിന്റെ ഔട്ട്പുട്ട് ചുവടെ ചേർക്കുന്നു.

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

sum1() ഫങ്ഷനിൽ a ക്കും b യുടേയും ഇൻപുട്ട്

sum 2() ഫങ്ഷനിൽ a, b യുടേയും ഇൻപുട്ട്

main() ഫങ്ഷനിൽ x, y എന്നിവയുടേയും വേണ്ട ഇൻപുട്ട്

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

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

പ്രോഗ്രാം 10.5 പെർഫക്ട് സംഖ്യ ആണോ എന്ന് പരിശോധിക്കുന്നു. പ്രോഗ്രാം 10.6

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

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

പട്ടിക 10.2 : ഫംങ്ഷനുകളുടെ വിശകലനം.

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

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

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

10.4.2 ഫങ്ഷനുകളുടെ പ്രോട്ടോടൈപ്പ് (Prototype of functions)

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

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

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

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

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

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

10.4.3 ഫങ്ഷനുകളുടെ ആർഗ്യുമെന്റുകൾ (Arguments of functions)

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

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

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

താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ഭാഗം വിവിധ ഫങ്ഷൻ വിളികൾ വിവരിക്കുന്നു.

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

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

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

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

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

പ്രോഗ്രാം 10.7 : nCr ന്റെ വില കണ്ടു പിടിക്കുന്നതിന്

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

ഫങ്ഷൻ പ്രോട്ടോ ടൈപ്പ്

യഥാർത്ഥ ആർഗ്യു മെന്റുകൾ

സൂത്രവാക്യത്തിന് അനുസൃതമായ ഫങ്ഷൻ വിളി

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

ഫങ്ഷൻ ഹെഡർ

യഥാക്രമ ആർഗ്യുമെന്റുകൾ

ഫാക്ടോറിയൽ തിരിച്ച് നൽകുന്നു

താഴെ കൊടുത്തിരിക്കുന്നത് ഒരു മാതൃക ഔട്ട്പുട്ട് ആണ്.

```
    Enter the values of n and r : 5    2
    5C2 = 10
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ടുകൾ

10.4.4. തനതു ആർഗ്യുമെന്റുകളോട് കൂടിയ ഫങ്ഷനുകൾ (Functions with default arguments)

നമുക്ക് താഴെ പറയുന്ന ആർഗ്യുമെന്റ് പട്ടികയോട് കൂടിയ `TimeSec()` എന്ന ഒരു ഫങ്ഷൻ പരിഗണിക്കാം. ഈ ഫങ്ഷൻ സമയത്തെ പ്രതിനിധീകരിക്കുന്ന മണിക്കൂറുകൾ, മിനിട്ട്, സെക്കന്റ് എന്നിവക്കുള്ള മൂന്ന് സംഖ്യകൾ സ്വീകരിക്കുന്നു.

```
    long TimeSec(int H, int M=0, int S=0)
    {
        long sec = H * 3600 + M * 60 + S;
        return sec;
    }
    SecTime()
```

തന്നിരിക്കുന്ന സമയത്തെ ഫങ്ഷൻ സെക്കന്റുകളിലേക്ക് മാറ്റുന്നു. `M`, `S` എന്നീ ആർഗ്യുമെന്റുകൾക്ക് തനതു വിലയായി പൂജ്യം നൽകിയിരിക്കുന്നത് ശ്രദ്ധിക്കുക. അതുകൊണ്ട് ഈ ഫങ്ഷൻ താഴെ പറയുന്ന രീതികളിൽ വിളിക്കാം.

```
    long s1 = TimeSec(2,10,40);
    long s2 = TimeSec(1,30);
    long s3 = TimeSec(3);
```

ആർഗ്യൂമെന്റുകളുടെ പട്ടികയിലെ തനതുവില നൽകുന്ന എല്ലാ ആർഗ്യൂമെന്റുകളും വലത്തു നിന്ന് ഇടത്തോട്ട് നൽകണം എന്നത് പ്രധാനമായും ശ്രദ്ധിക്കേണ്ടതാണ്. ഒരു ഫങ്ഷൻ വിളിക്കുമ്പോൾ യഥാർത്ഥ ആർഗ്യൂമെന്റുകൾ (actual arguments) ഫോർമൽ ആർഗ്യുമെന്റുകളിലേക്ക് ഇടത് ഭാഗം മുതൽ നൽകുന്നു.

ആദ്യത്തെ പ്രസ്താവന പ്രവർത്തിച്ചപ്പോൾ 2,10,40 എന്നീ വിലകൾ യഥാക്രമം ഫോർമൽ പരാമീറ്ററുകളായ H, M, S എന്നിവയിലേക്ക് ഫങ്ഷൻ വിളിയോടൊപ്പം അയക്കുന്നു. M, S എന്നിവയുടെ തനതു വിലകളെ യഥാർത്ഥ ആർഗ്യൂമെന്റുകൾ തിരുത്തി എഴുതുന്നു. രണ്ടാമത്തെ പ്രസ്താവനയിലുള്ള ഫങ്ഷൻ വിളിക്കുമ്പോൾ H നും M നും യഥാർത്ഥ ആർഗ്യൂമെന്റുകളുടെ വിലകൾ കിട്ടുമ്പോൾ S അതിന്റെ തനതു വിലയായ പൂജ്യത്തിൽ (0) പ്രവർത്തിക്കുന്നു. അതുപോലെ മൂന്നാമത്തെ പ്രസ്താവന പ്രവർത്തിക്കുമ്പോൾ H ന് വിളിച്ചിരിക്കുന്ന ഫങ്ഷനിൽ നിന്നു വിലകിട്ടുന്നു, എന്നാൽ M ഉം S ഉം അവയുടെ

തനത് വില ഉപയോഗിക്കുന്നു. അതുകൊണ്ട് ഫങ്ഷൻ വിളികൾക്ക് ശേഷം s1, s2, s3 എന്നിവയുടെ വിലകൾ യഥാക്രമം 7840, 5400, 10800 എന്നിങ്ങനെ ആയിരിക്കും.

ഫങ്ഷനുകളുടെ ആർഗ്യുമെന്റുകൾക്ക് തനത് വില നൽകി നിർവ്വചിക്കാൻ സാധിക്കും എന്ന് നാം കണ്ടു കഴിഞ്ഞു. തനത് വില നൽകിയ ആർഗ്യുമെന്റുകളെ തനത് (ഡിഫാൾട്ട്) ആർഗ്യുമെന്റുകൾ എന്നു വിളിക്കുന്നു. *ഇത് ഒരു ഫങ്ഷന് വ്യത്യസ്ത എണ്ണം ആർഗ്യു മെന്റുകൾ കൊണ്ട് വിളിക്കുന്നതിന് ഒരു പ്രോഗ്രാമറെ അനുവദിക്കുന്നു.* അതായത് തനത് ആർഗ്യുമെന്റുകൾക്ക് വിലകൾ നൽകിയോ നൽകാതെയോ ഫങ്ഷനെ വിളിക്കാൻ കഴിയും.

10.4.5 ഫങ്ഷൻ വിളിക്കുന്നതിനുള്ള വിവിധ രീതികൾ (Methods of calling Functions)

നിങ്ങളുടെ ടീച്ചർ ക്ലാസിലെ എല്ലാ വിദ്യാർത്ഥികളുടേയും രക്ഷകർത്താക്കളെ നിങ്ങ ളുടെ വിദ്യാലയത്തിൽ ഒരു പരിപാടിയിലേക്ക് ക്ഷണിക്കുന്നതിനുള്ള ഒരു കത്ത് തയ്യാറാ ക്കുവാൻ നിങ്ങളോട് ആവശ്യപ്പെട്ടു എന്ന് കരുതുക. കത്തിന്റെ മാതൃകയും രക്ഷാകർത്താ ക്കളുടെ പേര് അടങ്ങുന്ന പട്ടികയും നൽകാൻ ടീച്ചർക്ക് കഴിയും. പേരുകളുടെ പട്ടിക രണ്ട് വിധത്തിൽ നൽകാവുന്നതാണ്. ഒന്നുകിൽ യഥാർത്ഥ പട്ടിക അല്ലെങ്കിൽ അതിന്റെ ഫോട്ടോകോപ്പി. ഈ രണ്ട് രീതികളിൽ പേരിന്റെ പട്ടിക വ്യത്യാസം എന്താണ്? ടീച്ചർ യഥാർത്ഥ പട്ടികയാണ് നിങ്ങൾക്ക് നൽകുന്നതെങ്കിൽ, അതിൽ മറ്റൊന്നും എഴുതാതെയും രേഖപ്പെടുത്താതെയും ശ്രദ്ധാപൂർവ്വം ഉപയോഗിക്കണം. എന്തുകൊണ്ടെന്നാൽ ടീച്ചർക്ക് അതേ പട്ടിക തന്നെ ഭാവിയിൽ ആവശ്യമായി വരാം. എന്നാൽ ഫോട്ടോകോപ്പിയാണ് നിങ്ങൾക്ക് ലഭിക്കുന്നതെങ്കിൽ അതിൽ എഴുതുവാനോ രേഖപ്പെടുത്തുവാനോ നിങ്ങൾക്ക് സാധിക്കും. എന്തുകൊണ്ടെന്നാൽ ഫോട്ടോകോപ്പിയിൽ വരുത്തുന്ന വ്യത്യാസങ്ങൾ യഥാർത്ഥ പട്ടികയെ ബാധിക്കുന്നില്ല.

ക്ഷണക്കത്ത് തയ്യാറാക്കുന്ന ജോലി ഒരു ഫങ്ഷനായി നമുക്ക് പരിഗണിക്കാം. പേര് അട ങ്ങുന്ന പട്ടിക ഫങ്ഷൻ ആർഗ്യൂമെന്റ് ആണ്. ആർഗ്യൂമെന്റ് ഫങ്ഷനിലേക്ക് രണ്ട് രീതി യിൽ അയയ്ക്കാൻ സാധിക്കും. ആദ്യത്തേത് പേര് അടങ്ങുന്ന പട്ടികയുടെ കോപ്പി കൈമാ റുക, മറ്റേത്, യഥാർത്ഥ പട്ടികതന്നെ കൈമാറുക. ക്ഷണക്കത്ത് തയ്യാറാക്കുമ്പോൾ യഥാർത്ഥ പട്ടിക തന്നെയാണ് കൈമാറുന്നതെങ്കിൽ പട്ടികയിൽ ഉണ്ടാക്കുന്ന ഏത് മാറ്റവും യഥാർത്ഥ പട്ടികയെ തന്നെ ബാധിക്കും. അതുപോലെ, C++ ലും രണ്ട് രീതിയിൽ ഫങ്ഷ നിലേക്ക് ആർഗ്യൂമെന്റുകൾ അയയ്ക്കാം. ആർഗ്യുമെന്റുകൾ അയയ്ക്കുന്ന രീതിയെ അവലംബമാക്കി ഫങ്ഷൻ വിളിക്കുന്ന രീതിയെ കാൾ-ബൈ-വാല്യൂ രീതിയെന്നും കാൾ ബൈ റഫറൻസ് രീതിയെന്നും തരംതിരിക്കാം. ആർഗ്യൂമെന്റ് കൈമാറ്റ രീതികൾ വിശദമായി താഴെ വിവരിച്ചിരിക്കുന്നു

a. കാൾ-ബൈ-വാല്യൂ (പാസ് ബൈ വാല്യൂ) രീതി (Call by value Method)

ഈ രീതിയിൽ, ആക്ചൽ ആർഗ്യൂമെന്റുകളിൽ അടങ്ങിയ വിലകൾ ഫോർമൽ ആർഗ്യൂ മെന്റുകളിലേക്ക് (Formal argument) അയയ്ക്കുന്നു. മറ്റൊരു വിധത്തിൽ പറഞ്ഞാൽ ആക്ചൽ ആർഗ്യൂമെന്റിന്റെ ഒരു പകർപ്പ് ഫങ്ഷനിലേക്ക് അയയ്ക്കുന്നു. അതിനാൽ ഫങ്ഷനകത്ത് യഥാക്രമ ആർഗ്യൂമെന്റ് പുതുക്കപ്പെട്ടാലും, വിളിക്കുന്ന ഫങ്ഷനിലെ ആക്ചൽ ആർഗ്യൂമെന്റിൽ വ്യത്യാസം പ്രതിഫലിക്കുന്നില്ല. മുൻപ് ചർച്ച ചെയ്ത എല്ലാ ഫങ്ഷനുകളിലും ആർഗ്യൂമെന്റുകളുടെ വിലയാണ് അയച്ചത്. താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം കാണുക:

```
void change(int n)
{
    n = n + 1;
    cout << "n = " << n << '\n';
}
int main()
    {
    int x = 20;
    change(x);
    cout << "x = " << x;
    }
```

change () ഫങ്ഷനിലെ n പരാമീറ്ററിന് 20 എന്ന വില ശേഖരിക്കാൻ അതിന്റെ സ്വന്തം മെമ്മറി സ്ഥലം ഉണ്ട്.

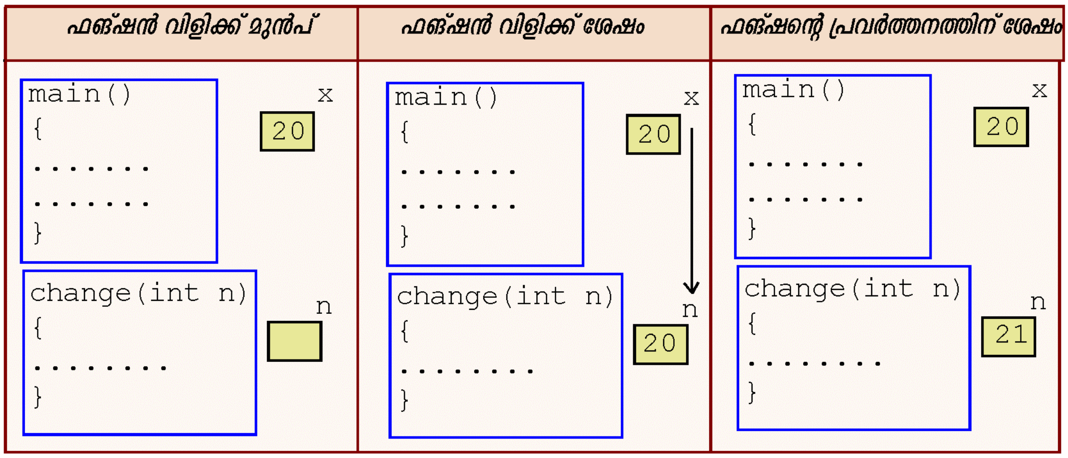
x ന്റെ വില change () ഫങ്ഷനിലെ n ലേക്ക് കൈമാറ്റം ചെയ്യപ്പെടുന്നു.

മുകളിലത്തെ പ്രോഗ്രാം ഭാഗത്ത് പരാമർശിച്ചിരിക്കുന്നതു പോലെ, നാം ഒരു ആർഗ്യുമെന്റ് കൈമാറ്റം ചെയ്യുമ്പോൾ വേരിയബിൾ x ന്റെ ഒരു പകർപ്പ് ഫങ്ഷനിലേക്ക് കൈമാറ്റം ചെയ്യപ്പെടുന്നു.

മറ്റൊരു വിധത്തിൽ പറഞ്ഞാൽ x എന്ന വേരിയബിളിന്റെ വിലമാത്രമേ ഫങ്ഷനിലേക്ക് അയയ്ക്കുന്നുള്ളൂ. അതിനാൽ ഫങ്ഷനിലെ ഫോർമൽ പരാമീറ്ററിന് 20 എന്ന വില ലഭിക്കും. നാം n ന്റെ വില കൂട്ടുകയാണെങ്കിൽ, അത് x എന്ന വേരിയബിളിന്റെ വിലയെ ബാധിക്കുന്നില്ല. ഈ കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
n = 21
x = 20
```

ഒരു ഫങ്ഷൻ, കാൾ-ബൈ-വാല്യൂ രീതിയിൽ വിളിച്ചാൽ ആർഗ്യുമെന്റുകൾക്ക് എന്ത് സംഭവിക്കുമെന്ന് പട്ടിക 10.2 ൽ കാണിക്കുന്നു.



പട്ടിക 10.2 കാൾ ബൈ വാല്യൂ പ്രവർത്തനം.

b. കാൾ ബൈ റഫറൻസ് (പാസ്സ് ബൈ റഫറൻസ്) രീതി (Call by reference (Pass by reference) Method)

ഒരു ആർഗ്യൂമെന്റ് റഫറൻസായി അയക്കുമ്പോൾ യഥാർത്ഥ ആർഗ്യൂമെന്റിന്റെ റഫറൻസ് (അഡ്രസ്സ്) ഫങ്ഷനിലേക്ക് അയക്കുന്നു. ഇതിന്റെ ഫലമായി യഥാർത്ഥ ആർഗ്യൂമെന്റിന് അനുവദിച്ച മെമ്മറിസ്ഥലം യഥാക്രമ ആർഗ്യൂമെന്റും കൂടി പങ്കിടും. അതുകൊണ്ട് വിളിച്ച ഫങ്ഷനിലെ യഥാക്രമ ആർഗ്യൂമെന്റിന് എന്തെങ്കിലും മാറ്റം സംഭവിച്ചാൽ ആ മാറ്റം ഫങ്ഷനിലെ യഥാർത്ഥ ആർഗ്യൂമെന്റിലും പ്രതിഫലിപ്പിക്കും. C++ ൽ ആർഗ്യൂമെന്റുകൾ റഫറൻസായി അയക്കുന്നതിന് യഥാക്രമ പരാമീറ്ററായി റഫറൻസ് വേരിയബിൾ ഉപയോഗിക്കുന്നു. ഒരു റഫറൻസ് വേരിയബിൾ മറ്റൊരു വേരിയബിളിന്റെ അപരനാമമാണ്. ഫങ്ഷൻ ഹെഡറിലെ ഡാറ്റ ഇനത്തിനും വേരിയബിളിനും ഇടയിൽ ഒരു ആമ്പർസാന്റ് ചിഹ്നം (&) ഉപയോഗിക്കുന്നു. മറ്റ് വേരിയബിളുകളെപ്പോലെ റഫറൻസ് വേരിയബിളുകൾക്ക് പ്രത്യേകമായ മെമ്മറിസ്ഥലം അനുവദിക്കുന്നില്ല. പകരം യഥാർത്ഥ ആർഗ്യൂമെന്റിന് അനുവദിച്ച മെമ്മറി സ്ഥലം പങ്കിടും. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷന്റെ യഥാക്രമം പരാമീറ്ററായി റഫറൻസ് വേരിയബിൾ ഉപയോഗിക്കുന്നു. അതുകൊണ്ട് ഫങ്ഷൻ വിളിക്കായി കാൾബൈ റഫറൻസ് രീതി പ്രാവർത്തികമാക്കുന്നു.

```
void change(int & n)
{
    n = n + 1;
    cout << "n = " << n << '\n';
}
int main()
{
    int x=20;
    change(x);
    cout << "x = " << x;
}
```

n എന്ന പരാമീറ്റർ ഒരു റഫറൻസ് വേരിയബിൾ ആകുന്നു. അതിനാൽ അതിന് പ്രത്യേകമായ മെമ്മറി അനുവദിക്കുന്നില്ല.

x ന്റെ റഫറൻസ് change () ഫങ്ഷനിലെ n ലേക്ക് കൈമാറ്റം ചെയ്യപ്പെടും. തൽഫലമായി മെമ്മറി പങ്കുവയ്ക്കുന്നു.

change() ലെ ഫങ്ഷൻ ഹെഡറിന് മാത്രമേ മാറ്റം ഉള്ളു എന്നത് ശ്രദ്ധിക്കുക. പരാമീറ്റർ n ന്റെ പ്രഖ്യാപനത്തിലെ & ചിഹ്നം അർത്ഥമാക്കുന്നത് ഒരു റഫറൻസ് വേരിബിളാണ് അത് എന്നാണ്. അതുകൊണ്ട് റഫറൻസ് അയച്ചു കൊണ്ട് ഫങ്ഷൻ വിളിക്കപ്പെടും. തന്മൂലം x എന്ന നെ change() ഫങ്ഷനിലേക്ക് അയക്കുമ്പോൾ n ന് x ന്റെ അഡ്രസ് കിട്ടുന്നതിനാൽ അവ മെമ്മറിസ്ഥലം പങ്കിടും. മറ്റൊരു തരത്തിൽ പറഞ്ഞാൽ വേരിയബിളുകളായ n ഉം x ഉം ഒരേ മെമ്മറിസ്ഥലം പരാമർശിക്കുന്നു. നാം main() ഫങ്ഷനിൽ x എന്ന പേരും change() ഫങ്ഷനിൽ n എന്ന പേരും ഉപയോഗിച്ച് ഒരേ മെമ്മറിസ്ഥലം പരാമർശിക്കുന്നു. അതുകൊണ്ട് n ന്റെ വിലയിൽ വ്യത്യാസം വരുത്തുമ്പോൾ യഥാർത്ഥത്തിൽ x വിലയിലാണ് മാറ്റം ഉണ്ടാകുന്നത്. മുകളിലുള്ള പ്രോഗ്രാം പ്രവർത്തിക്കുമ്പോൾ നമുക്ക് താഴെ കൊടുത്തിരിക്കുന്ന ഔട്ട്പുട്ട് ലഭിക്കും.

```
n = 21
x = 21
```

ഫങ്ഷൻ വിളിക്ക് വേണ്ടി കാൾ-ബൈ-റഫറൻസ് ഉപയോഗിക്കുമ്പോൾ ആർഗ്യൂമെന്റുകൾക്ക് ഉണ്ടാകുന്ന മാറ്റങ്ങൾ ടേബിൾ 10.3 ൽ വർണ്ണിക്കുന്നു.

ഫങ്ഷൻ വിളിക്ക് മുമ്പ്	ഫങ്ഷൻ വിളിക്ക് ശേഷം	ഫങ്ഷൻ പ്രവർത്തിച്ചതിന് ശേഷം

main()
{
....
....
}
x: 20
change(int &n)
{
....
}
	main()
	{
	....
	....
	}
	x: 20 n
	change(int &n)
	{
	....
	}
		main()
		{
		....
		....
		}
		x: 21 n
		change(int &n)
		{
		....
		}

പട്ടിക 10.3. കാൾ ബൈ റഫറൻസ് പ്രവർത്തനം

ഫങ്ഷൻ വിളിയുടെ ഈ രണ്ട് രീതികൾ പട്ടിക 10.4 ൽ കാണിച്ചിരിക്കുന്നതു പോലെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു.

കാൾ ബൈ വാല്യൂ രീതി	കാൾ ബൈ റഫറൻസ് രീതി
• യഥാക്രമ പരാമീറ്ററുകളായി സാധാരണ വേരിയബിളുകൾ ഉപയോഗിക്കുന്നു. • സ്ഥിരവിലകൾ, വേരിയബിളുകൾ, അല്ലെങ്കിൽ പദപ്രയോഗങ്ങൾ എന്നിവ യഥാർത്ഥ പരാമീറ്ററുകൾ ആയി ഉപയോഗിക്കാം. • യഥാക്രമ പരാമീറ്ററുകളിൽ ഉണ്ടാക്കുന്ന വ്യത്യാസങ്ങൾ യഥാർത്ഥ പരാമീറ്ററുകളിൽ പ്രതിഫലിക്കുന്നില്ല. • യഥാക്രമ ആർഗ്യൂമെന്റുകൾക്ക് പ്രത്യേകം മെമ്മറി ആവശ്യമുണ്ട്.	• യഥാക്രമ പരാമീറ്ററുകളിൽ റഫറൻസ് വേരിയബിളുകൾ ഉപയോഗിക്കുന്നു. • വേരിയബിളുകൾ മാത്രമേ യഥാർത്ഥ പരാമീറ്ററുകൾ ആകൂ. • യഥാക്രമ പരാമീറ്ററിന് ഉണ്ടാക്കുന്ന വ്യത്യാസങ്ങൾ യഥാർത്ഥ പരാമീറ്ററുകളിൽ പ്രതിഫലിക്കും. • യഥാർത്ഥ ആർഗ്യൂമെന്റുകളുടെ മെമ്മറി യഥാക്രമ ആർഗ്യൂമെന്റുകൾ പങ്കിടുന്നു.

പട്ടിക 10.4. കാൾ ബൈ വാല്യൂ v/s കാൾ ബൈ റഫറൻസ്

കാൾ ബൈ റഫറൻസ് രീതി വിശദമാക്കുന്നതിന് അനുയോജ്യമായ ഒരു ഉദാഹരണം നമുക്ക് ചർച്ച ചെയ്യാം. ഈ പ്രോഗ്രാം main() ഫങ്ഷനിലെ രണ്ട് വേരിയബിളുകളുടെ വിലകൾ പരസ്പരം കൈമാറുന്നതിന് കാൾ ബൈ റഫറൻസ് രീതിയിൽ വിളിക്കാൻ കഴിയുന്ന ഒരു ഫങ്ഷൻ ഉപയോഗിക്കുന്നു. രണ്ട് വേരിയബിളുകളുടെ വിലകൾ പരസ്പരം കൈമാറുന്ന പ്രക്രിയയെ സ്വാപ്പിങ്ങ് എന്ന് പറയുന്നു.

പ്രോഗ്രാം 10.8 രണ്ട് വേരിയബിളുകളുടെ വിലകൾ പരസ്പരം കൈമാറുന്നതിന്.

```
#include <iostream>
using namespace std;
void swap(int & x, int & y)
{
    int t = x;
    x = y;
    y = t;
```

```
}
int main()
{
    int m, n;
    m = 10;
    n = 20;
    cout<<"Before swapping m= "<< m <<" and n= "<<n;
    swap(m, n);
    cout<<"\nAfter swapping m= "<< m <<" and n= "<<n;
    return 0;
}
```

നമുക്ക് പ്രോഗ്രാം 10.8 ലെ സ്റ്റേറ്റ്മെന്റുകളിലൂടെ കടന്നു പോകാം. ആക്ചൽ ആർഗ്യുമെന്റുകളായ m ഉം n ഉം ഫങ്ഷനിലേക്ക് റഫൻസ് ആയി അയക്കുന്നു. swap() ഫങ്ഷന് ഉള്ളിൽ x ന്റെയും y യുടേയും വിലകൾ പരസ്പരം കൈമാറ്റം ചെയ്യപ്പെടുന്നു. അപ്പോൾ യഥാർത്ഥത്തിൽ m ലും n ലുമാണ് മാറ്റം നടക്കുന്നത്. അതിനാൽ മുകളിലത്തെ പ്രോഗ്രാം കോഡിന്റെ ഔട്ട്പുട്ട്,

```
Before swapping m = 10 and n= 20
After swapping m = 20 and n= 10
```

യഥാക്രമ ആർഗ്യുമെന്റിന് പകരം സാധാരണ വേരിയബിൾ ഉപയോഗിച്ച് മുകളിലെ പ്രോഗ്രാമിൽ മാറ്റം വരുത്തി അതിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുക. കമ്പ്യൂട്ടർ ലാബിൽ ഈ കോഡ് പ്രവർത്തിപ്പിച്ച് നിങ്ങളുടെ ഉത്തരം പരിശോധിക്കുക.

സ്വയം പരിശോധിക്കാം

1. C++ പ്രോഗ്രാമുകളിലെ ഏറ്റവും ഒഴിച്ചു കൂടാനാവാത്ത ഫങ്ഷൻ ഏതെന്ന് തിരിച്ചറിയുക.
2. ഒരു ഫങ്ഷൻ ഹെഡറിന്റെ മൂന്ന് ഘടകങ്ങൾ പട്ടികപ്പെടുത്തുക.
3. ഫങ്ഷൻ പ്രോട്ടോടൈപ്പ് എന്നാൽ എന്ത്?
4. വിളിക്കുന്ന ഫങ്ഷനിൽ നിന്നും വിളിച്ച ഫങ്ഷനിലേക്ക് ഡാറ്റാ അയക്കുന്നതിന് ഏത് ഘടകമാണ് ഉപയോഗിക്കുന്നത്.
5. C++ ൽ ഉപയോഗിക്കുന്ന രണ്ട് പരാമീറ്റർ കൈമാറ്റരീതികൾ ഏതൊക്കെയാണ്?

10.5 വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും (Scope and life of variables and functions)

ഒന്നിലധികം ഫങ്ഷനുകൾ അടങ്ങിയ C++ പ്രോഗ്രാം നാം ചർച്ച ചെയ്തു. മുൻ നിർവചിത ഫങ്ഷനുകൾ അതിന് അനുബന്ധമായ ഹെഡർ ഫയലുകൾ ഉൾപ്പെടുത്തിയാണ് പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്നത്. ഉപയോക്തൃ നിർമ്മിത ഫങ്ഷനുകൾ main() ഫങ്ഷന് മുമ്പോ ശേഷമോ നിർവ്വചിക്കപ്പെടുന്നു. ഫങ്ഷനുകൾ നിർവ്വചിക്കുമ്പോൾ ഫങ്ഷൻ പ്രോട്ടോടൈപ്പിനുള്ള പ്രസക്തി നാം കണ്ടു കഴിഞ്ഞു. വേരിയബിളുകൾ ഫങ്ഷൻ ചട്ടക്കൂടിലും ആർഗ്യുമെന്റുകളായും നാം ഉപയോഗിച്ചു. ഇനി നമുക്ക് വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും പ്രോഗ്രാമിലുള്ള ലഭ്യതയും പ്രാപ്യതയും ചർച്ച ചെയ്യാം. ഒരു പ്രോഗ്രാമിൽ ലോക്കൽ വേരിയബിളുകളുടെ പ്രാപ്യത പ്രോഗ്രാം 10.9 വിവരിക്കുന്നു.

പ്രോഗ്രാം 10.9. വേരിയബിളുകളുടെ വ്യാപ്തിയും ജീവനവും വിവരിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int cube(int n)
{
    int cb;
    cout<< "The value of x passed to n is " << x;
    cb = n * n * n;
    return cb;
}
int main()
{
    int x, result;
    cout << "Enter a number : ";
    cin >> x;
    result = cube(x);
    cout << "Cube = " << result;
    cout << "\nCube = "<<cb;
}
```

ഇത് തെറ്റ് ആകുന്നു. എന്തുകൊണ്ടെന്നാൽ വേരിയബിൾ x, main() ഫങ്ഷനിലാണ് പ്രഖ്യാപിച്ചത്. അതു കൊണ്ട് മറ്റ് ഫങ്ഷനിൽ അത് ഉപയോഗിക്കാൻ കഴിയില്ല.

ഇത് തെറ്റ് ആകുന്നു എന്തുകൊണ്ടെന്നാൽ cb എന്ന വേരിയബിൾ cube() എന്ന ഫങ്ഷനിലാണ് പ്രഖ്യാപിച്ചത്. അതുകൊണ്ട് മറ്റ് ഫങ്ഷനിൽ അത് ഉപയോഗിക്കാൻ കഴിയില്ല.

നാം പ്രോഗ്രാം കമ്പയിൽ ചെയ്യുമ്പോൾ, കോളൗട്ടിൽ കാണിച്ചിരിക്കുന്ന കാരണങ്ങളാൽ രണ്ട് തെറ്റുകൾ ഉണ്ടാകും. വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും ലഭ്യത, പ്രാപ്യത എന്നീ ആശയത്തെ വ്യാപ്തി, ജീവനം, എന്നീ പദങ്ങൾ കൊണ്ട് സൂചിപ്പിക്കുന്നു. പ്രോഗ്രാമിന്റെ ഏത് ഭാഗത്താണോ ഒരു വേരിയബിൾ ഉപയോഗിക്കുന്നത് അതാണ് അതിന്റെ വ്യാപ്തി (Scope) മുകളിലത്തെ പ്രോഗ്രാമിൽ വേരിയബിൾ `cb` യുടെ വ്യാപ്തി ഫങ്ഷൻ `cube()` ൽ ആകുന്നു എന്തുകൊണ്ടെന്നാൽ അത് ആ ഫങ്ഷനിലാണ് പ്രഖ്യാപിച്ചത്. അതിനാൽ ഈ വേരിയബിൾ ആ ഫങ്ഷന് പുറത്ത് ഉപയോഗിക്കാൻ കഴിയില്ല. ഈ വ്യാപ്തി **ലോക്കൽ വ്യാപ്തി** എന്ന് അറിയപ്പെടുന്നു. ഒരു ഫങ്ഷന്റെ പ്രവർത്തനം പൂർത്തീകരിക്കുമ്പോൾ ആ ഫങ്ഷന്റെ ഉള്ളിലെ എല്ലാ വേരിയബിളുകൾക്കും അനുവദിച്ച മെമ്മറി സ്വതന്ത്രമാക്കുന്നു. മറ്റൊരു തരത്തിൽ പറയുമ്പോൾ ഒരു ഫങ്ഷന് ഉള്ളിൽ പ്രഖ്യാപിച്ച വേരിയബിളുകളുടെ സമയം ആ ഫങ്ഷനിലെ അവസാനത്തെ നിർദേശം പ്രവർത്തിക്കുന്നതോട് കൂടി അവസാനിക്കുന്നു. അതുകൊണ്ട് `main()` ഫങ്ഷനിൽ `n` എന്ന വേരിയബിൾ ഉപയോഗിക്കുമ്പോൾ അതിൽ നിന്നും വിളിക്കുന്ന ഫങ്ഷനിലെ `n` എന്ന ആർഗ്യുമെന്റിൽ നിന്നും വിളിക്കപ്പെട്ട ഫങ്ഷനിലെ വേരിയബിൾ `n`ൽ നിന്നും അത് വ്യത്യസ്തമായിരിക്കും. ഒരു ഫങ്ഷന് ഉള്ളിൽ പ്രഖ്യാപിച്ച വേരിയബിളുകൾക്കും യഥാക്രമ പരാമീറ്ററുകൾക്കും ലോക്കൽ വ്യാപ്തി ഉണ്ടായിരിക്കും.

വേരിയബിളിനെ പോലെ തന്നെ ഫങ്ഷനുകൾക്കും വ്യാപ്തി ഉണ്ട്. എവിടെ ആണോ ഒരു ഫങ്ഷൻ പ്രഖ്യാപിച്ചിരിക്കുന്നത് അവിടെയാണ് ആ ഫങ്ഷൻ ഉപയോഗിക്കാൻ കഴിയുക. അതായത് ഫങ്ഷന് ലോക്കൽ വ്യാപ്തി ഉണ്ടെന്ന് പറയാം. അത് `main()` ഫങ്ഷന് മുമ്പേ മറ്റ് ഫങ്ഷനുകൾക്ക് പുറമെയോ പ്രഖ്യാപിച്ചാൽ ആ ഫങ്ഷന്റെ വ്യാപ്തി പ്രോഗ്രാം മുഴുവൻ ആയിരിക്കും. അതായത് പ്രോഗ്രാമിലെ ഏത് സ്ഥലത്തും ഫങ്ഷൻ ഉപയോഗിക്കാൻ കഴിയും. ഈ വ്യാപ്തി **ഗ്ലോബൽ വ്യാപ്തി** എന്ന് അറിയപ്പെടുന്നു. വേരിയബിളുകളും ഇപ്രകാരം ഗ്ലോബൽ വ്യാപ്തിയിൽ പ്രഖ്യാപിക്കാൻ കഴിയും. അങ്ങനെയുള്ള പ്രഖ്യാ

പനങ്ങൾ പ്രോഗ്രാമിലെ എല്ലാ ഫങ്ഷനുകൾക്കും പുറത്ത് ആയിരിക്കും. ഒരു പ്രോഗ്രാമിലെ വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും, ജീവനവും സംബന്ധിച്ച് കൂടുതൽ വ്യക്തത കിട്ടുവാൻ പ്രോഗ്രാം 10.10 നോക്കുക.

പ്രോഗ്രാം 10.10 വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും വിവരിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int cb;          //global variable
void test()//global function since defined above other functions
{
    int cube(int n); //It is a local function
    cb=cube(x); //Invalid call. x is local to main()
    cout<<cb;
}
int main() // beginning of main() function
{
    int x=5; //local variable
    test();  //valid call since test() is a global function
    cb=cube(x); //Invalid call. cube() is local to test()
    cout<<cb;
}
int cube(int n)//Argument n is local variable
{
    int val= n*n*n; //val is local variable
    return val;
}
```

വ്യാപ്തിയും ജീവനും	ലോക്കൽ	ഗ്ലോബൽ
വേരിയബിളുകൾ	• ഒരു ഫങ്ഷൻ അല്ലെങ്കിൽ പ്രസ്താവനകളുടെ കൂട്ടത്തിന് ഉള്ളിൽ പ്രഖ്യാപിക്കുന്നു. • ആ ഫങ്ഷൻ അല്ലെങ്കിൽ ബ്ലോക്കിൽ മാത്രമേ ലഭ്യമാകൂ. • ഫങ്ഷൻ അല്ലെങ്കിൽ ബ്ലോക്ക് പ്രവർത്തിക്കുമ്പോൾ മെമ്മറി അനുവദിക്കുകയും ഫങ്ഷന്റെയോ ബ്ലോക്കിന്റെയോ പ്രവർത്തനം പൂർത്തിയാകുമ്പോൾ അവ സ്വതന്ത്രമാക്കുകയും ചെയ്യുന്നു.	• എല്ലാ ഫങ്ഷന്റെയും പുറമെ പ്രഖ്യാപിക്കുന്നു. • പ്രോഗ്രാമിലെ എല്ലാ ഫങ്ഷനുകൾക്കും ലഭ്യമാണ്. • പ്രോഗ്രാമിന്റെ പ്രവർത്തനം തുടങ്ങുന്നതിന് തൊട്ട് മുൻപേ മെമ്മറി അനുവദിക്കുകയും പ്രോഗ്രാം അവസാനിക്കുമ്പോൾ അവ സ്വതന്ത്രമാവുകയും ചെയ്യുന്നു.
ഫങ്ഷനുകൾ	• ഒരു കൂട്ടം പ്രസ്താവനകൾക്ക് ഉള്ളിലോ ഫങ്ഷന് ഉള്ളിലോ പ്രഖ്യാപിക്കുകയും വിളിക്കുന്ന ഫങ്ഷന് ശേഷം നിർവ്വചിക്കുകയും ചെയ്യുന്നു. • ആ ഫങ്ഷൻ അല്ലെങ്കിൽ ബ്ലോക്കിന് ഉള്ളിൽ മാത്രമേ പ്രാപ്യമാകൂ.	• മറ്റ് എല്ലാ ഫങ്ഷനുകളുടേയും വെളിയിൽ പ്രഖ്യാപിക്കുകയോ നിർവ്വചിക്കുകയോ ചെയ്യുന്നു. • പ്രോഗ്രാമിലെ എല്ലാ ഫങ്ഷനുകൾക്കും പ്രാപ്യമാണ്.

ടേബിൾ 10.5 : വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും

തന്നിരിക്കുന്ന കാൾ ഔട്ടുകൾ ഫങ്ഷനുകളുടെ വ്യാപ്തിയും ജീവനവും വിശദമാക്കുന്നു. ഒരു ഫങ്ഷൻ മറ്റൊരു ഫങ്ഷന്റെ ചട്ടക്കൂടിനുള്ളിൽ പ്രഖ്യാപിക്കുകയാണെങ്കിൽ അതിനെ ലോക്കൽ ഫങ്ഷൻ എന്ന് വിളിക്കുന്നു. അത് ആ ഫങ്ഷന് ഉള്ളിൽ മാത്രമേ ഉപയോഗിക്കാൻ കഴിയൂ. ഒരു ഫങ്ഷൻ മറ്റെല്ലാ ഫങ്ഷനുകളുടേയും ചട്ടക്കൂടിന് പുറമെ പ്രഖ്യാപിച്ചാൽ അതിനെ ഗ്ലോബൽ ഫങ്ഷൻ എന്ന് വിളിക്കുന്നു. ഒരു ഗ്ലോബൽ ഫങ്ഷൻ പ്രോഗ്രാമിൽ ഉടനീളം ഉപയോഗിക്കാൻ കഴിയും. മറ്റൊരു തരത്തിൽ പറഞ്ഞാൽ ഒരു ഗ്ലോബൽ ഫങ്ഷന്റെ വ്യാപ്തി പ്രോഗ്രാം മുഴുവനാകുമ്പോൾ ലോക്കൽ ഫങ്ഷന്റെ വ്യാപ്തി അത് പ്രഖ്യാപിച്ചിരിക്കുന്ന ഫങ്ഷനിൽ മാത്രം ആയിരിക്കും. വേരിയബിളുകളുടേയും ഫങ്ഷനുകളുടേയും വ്യാപ്തിയും ജീവനവും ടേബിൾ 10.6 ൽ സംഗ്രഹിച്ചിരിക്കുന്നു.

10.6 സ്വയം ആവർത്തിക്കുന്ന ഫങ്ഷനുകൾ (Recursive Function)

സാധാരണയായി ഒരു ഫങ്ഷൻ മറ്റൊരു ഫങ്ഷനെയാണ് വിളിക്കുന്നത്. ഇനി നമുക്ക് അതിനെ തന്നെ വിളിക്കുന്ന ഒരു ഫങ്ഷൻ നിർവ്വചിക്കാം. ഒരു ഫങ്ഷൻ ആ ഫങ്ഷനെ തന്നെ വിളിക്കുന്ന പ്രവർത്തനത്തെ സ്വയം ആവർത്തനം എന്ന് അറിയപ്പെടുന്നു. ആ ഫങ്ഷനെ സ്വയം ആവർത്തിക്കുന്ന ഫങ്ഷൻ എന്നും പറയുന്നു. സങ്കീർണ്ണമായ ചില അൽഗോരിതങ്ങൾ സ്വയം ആവർത്തന രീതി ഉപയോഗിച്ച് വളരെ എളുപ്പത്തിൽ ലഘൂകരിക്കാം.

```
int function1()
{
    ............
    ............
    int n = function1();    //calling itself
    ............
    ............
}
```

ഒരു സ്വയം ആവർത്തി ഫങ്ഷന്റെ മാതൃക താഴെ പറയുന്നത് പോലെ ആകും.

സ്വയം ആവർത്തിത ഫങ്ഷനുകളിൽ സാധാരണയായി ചില ഉപാധിയെ അടിസ്ഥാനമാക്കിയായിരിക്കും ഫങ്ഷൻ വിളിക്കുന്നത്. താഴെ കൊടുത്തിരിക്കുന്ന സ്വയം ആവർത്തിത ഫങ്ഷനിലൂടെ ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കാം. നെഗറ്റീവ് സംഖ്യയുടെ ഫാക്ടോറിയൽ നിർവ്വചിച്ചിട്ടില്ല എന്നത് ശ്രദ്ധിക്കുക. അതുകൊണ്ട് n ന് പൂജ്യമോ പോസിറ്റീവ് സംഖ്യയോ കൊടുക്കാൻ കഴിയും.

```
int factorial(int n)
{
    if((n==1)||(n==0))
        return 1;
    else if (n>1)
        return (n * factorial(n-1));
    else
        return 0;
}
```

ഉപയോക്താവ് ഈ ഫങ്ഷൻ വിളിക്കുമ്പോൾ അത് എങ്ങനെ പ്രവർത്തിക്കുമെന്ന് നമുക്ക് ചർച്ച ചെയ്യാം.

```
f = factorial(3);
```

ഫങ്ഷൻ ആദ്യത്തെ തവണ വിളിക്കുമ്പോൾ, `n` വില 3 ആകുന്നു. `if` പ്രസ്താവനയിലെ വ്യവസ്ഥ വിലയിരുത്തുമ്പോൾ വില തെറ്റ് എന്ന് കണക്കാക്കുന്നു. അതുകൊണ്ട് `else` ചട്ടക്കൂട് പ്രവർത്തിക്കും. `n` ന്റെ വില 3 ആകുമ്പോൾ `else` ബ്ലോക്കിലെ നിർദ്ദേശം

`return (3 * factorial(3-1));` എന്ന് ആകും.

അത് ലഘൂകരിക്കുമ്പോൾ `return (3 * factorial(2));`.......... (i), `3 * factorial(2)`, കണ്ടുപിടിക്കുന്നതിന് വേണ്ടി `factorial(2)` ന്റെ വില കണ്ടുപിടിക്കേണ്ടതാവശ്യമാണ്. `factorial()` എന്ന ഫങ്ഷൻ വീണ്ടും 2 എന്ന ആർഗ്യൂമെന്റ് ഉപയോഗിച്ച് വിളിക്കുന്നു. ഫങ്ഷൻ അതേ ഫങ്ഷനുള്ളിൽ തന്നെ വിളിക്കുന്നു. `n` എന്ന പരാമീറ്ററിന്റെ വിലയായ 2 കൊണ്ട് `factorial()` ഫങ്ഷൻ പ്രവർത്തിക്കുമ്പോൾ `if` ബ്ലോക്കിലെ വ്യവസ്ഥ തെറ്റ് ആയി കണക്കാക്കുന്നു. അതുകൊണ്ട് `else` ബ്ലോക്ക് മാത്രമേ പ്രവർത്തിക്കൂ. `else` ബ്ലോക്കിലെ നിർദ്ദേശം return (2 * factorial(2-1)); എന്നാകുന്നു.

ഇത് ലഘൂകരിക്കുമ്പോൾ `return (2 * factorial(1));` .... (ii)

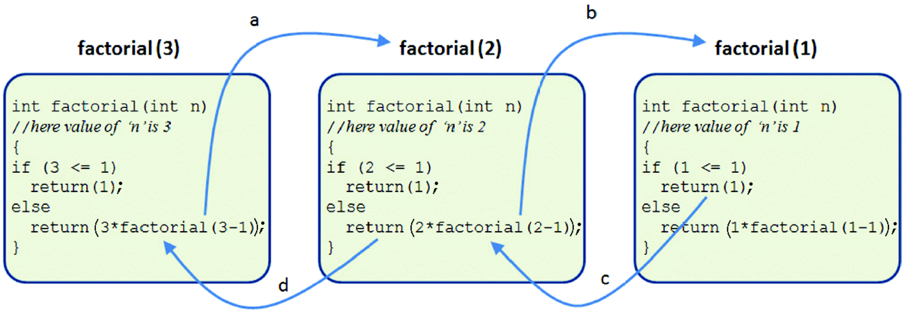
ഇതിന്റെ തിരിച്ചു നൽകുന്ന വില കണ്ടുപിടിക്കുന്നതിനായി അത് `n` എന്ന പരാമീറ്ററിന്റെ വിലയായി 1 നൽകിക്കൊണ്ട് വീണ്ടും `factorial()` ഫങ്ഷൻ വിളിക്കുന്നു. ഇപ്പോൾ `if` സ്റ്റേറ്റ്മെന്റിന്റെ വ്യവസ്ഥ ശരി `(true)` ആയി കണക്കാക്കുന്നതിനാൽ `factorial(1)` എന്ന ഫങ്ഷൻ കാളിന്റെ വില ആയി 1 എന്ന് തിരിച്ചു നൽകും. ഇപ്പോൾ പ്രോഗ്രാമിന് `(ii)` എന്ന് രേഖപ്പെടുത്തിയ നിർദ്ദേശത്തിന്റെ തിരിച്ചു നൽകുന്ന വില കണ്ടുപിടിക്കാൻ കഴിയും. നിർദ്ദേശം `(ii).return 2*1` എന്ന് ആകും. അത് `return 2` ന് തുല്യമായിരിക്കും.

അതായത് `factorial (2)` എന്ന ഫങ്ഷൻ തിരിച്ചു നൽകിയ 2 എന്ന വില `(i)` എന്ന് രേഖപ്പെടുത്തിയ നിർദ്ദേശത്തിലേക്ക് നൽകുന്നു. അതിനാൽ നിർദ്ദേശം `(i) return 3 x 2` എന്ന് ആകുന്നു. അതിന് സമമാണ് `return 6;`

ഇപ്പോൾ `factorial (3)` എന്ന ഫങ്ഷൻ വിളിയുടെ തിരിച്ചു നൽകുന്ന വിലയായി 6 ലഭിക്കുന്നു.

`f = factorial (3)` എന്ന നിർദ്ദേശത്തിന്റെ പ്രവർത്തനം ചിത്രം 10.4 ൽ സംഗ്രഹിക്കുന്നു.

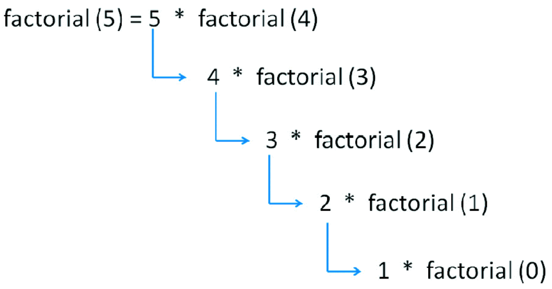
`factorial (3)` എന്ന ഫങ്ഷൻ വിളിയുടെ പ്രവർത്തനം `factorial(2)` എന്ന ഫങ്ഷന്റെ വില ലഭിക്കുന്നത് വരെ താമസിക്കുന്നു. ഫങ്ഷന്റെ പ്രവർത്തനം `factorial (1)` ന്റെ വില കിട്ടുന്നത് വരെ താമസിക്കും. ഒരിക്കൽ ഈ ഫങ്ഷൻ കാളിന് 1 എന്ന തിരിച്ച് നൽകുന്ന വിലയായി ലഭിച്ചാൽ അത് വിലയെ മുൻപ് നടന്ന ഫങ്ഷൻ വിളികളിലേക്ക് തിരികെ നൽകും. ഫങ്ഷന്റെ ഓരോ വിളിയിലും ആർഗ്യൂമെന്റുകൾക്കും തിരിച്ച് നൽകുന്ന വിലകൾക്കും എന്ത് സംഭവിക്കുന്നു എന്നത് ചിത്രം 10.4 ൽ കാണിക്കുന്നു.



ചിത്രം 10. 4; ഒരു സ്വയം ആവർത്തിത ഫങ്ഷൻ വിളിയുടെ നിയന്ത്രണ ഗതി

ഒരു ന്യൂനസംഖ്യ ആർഗ്യുമെന്റ് ആയി `factorial()` എന്ന ഫങ്ഷൻ വിളിക്കുമ്പോൾ ഫങ്ഷൻ പൂജ്യം തിരികെ നൽകും എന്നത് ശ്രദ്ധിക്കുക. ഒരു നെഗറ്റീവ് സംഖ്യയുടെ ഫാക്ടോറിയൽ പൂജ്യം ആണെന്ന് ഇതിന് അർത്ഥമില്ല. ഗണിതത്തിൽ ഒരു ന്യൂനസംഖ്യ യുടെ ഫാക്ടോറിയൽ നിർവചിച്ചിട്ടില്ല. ഉദാഹരണത്തിന് `factorial(-3)` എന്ന ഫങ്ഷൻ വിളി അസാധുവായ വിളിയാണെന്നത് വിളിക്കുന്ന ഫങ്ഷനിൽ രേഖപ്പെടുത്തും.

`factorial()` എന്ന ഫങ്ഷൻ 5 എന്ന വില ഉപയോഗിച്ച് വിളിക്കുമ്പോൾ ഉണ്ടാകുന്ന പ്രവർത്തനം ചിത്രം 10.5 ൽ കാണിക്കുന്നു.



ചിത്രം 10.5 : factorial (5) ന്റെ സ്വയം ആവർത്തന പ്രവർത്തനം

ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കുന്നതിനായി സ്വയം ആവർത്തിത ഫങ്ഷ നല്ലാതെ താഴെ കൊടുക്കുന്ന ഫങ്ഷനും ഉപയോഗിക്കാം.

```
int factorial(int n)
{
    int f=1;
    /*  The formula n×(n-1)×(n-2)× ... ×2×1 is applied
    instead of 1 × 2 × 3× ... ×(n-1)×n to find the factorial
    */
    for(int i=n; i>1; i--)
        f *= i;
    return f;
}
```

ഈ രണ്ട് ഫങ്ഷനുകൾ തമ്മിലുള്ള വ്യത്യാസം നമുക്ക് താരതമ്യം ചെയ്യാം.

റിക്കർഷൻ ഉപയോഗിക്കുന്ന എല്ലാ ഫങ്ഷനുകളും റിക്കർഷൻ ഉപയോഗിക്കാതെയും എഴുതാം. പിന്നെ എന്തിന് വേണ്ടിയാണ് നാം ആവർത്തനം ഉപയോഗിക്കുന്നത്? ചില പ്രോഗ്രാമർമാർക്ക് ആവർത്തനം ഉപയോഗിക്കുന്നതാണ് മറ്റെതിനേക്കാൾ വളരെ ലളിതമാണ്. എല്ലാ ഫങ്ഷനുകളിലും സ്വയം ആവർത്തന രീതി ഉപയോഗിക്കാൻ കഴിയില്ല.

ഒരു ഫങ്ഷനിൽ സ്വയം ആവർത്തനം നടത്തുവാൻ കഴിയുമോ ഇല്ലയോ എന്ന് എങ്ങനെ നമുക്ക് മനസ്സിലാക്കാം. ഒരു ഫങ്ഷനകത്ത് അതേ ഫങ്ഷന്റെ തന്നെ ഔട്ട്പുട്ടിൽ ചില പ്രവർത്തനങ്ങൾ നടത്തുവാൻ നമുക്ക് കഴിയുമെങ്കിൽ അത്തരം ഫങ്ഷനിൽ സ്വയം ആവർത്തനം ഉപയോഗിക്കാം. ഉദാഹരണത്തിന് ആദ്യത്തെ n എണ്ണൽ സംഖ്യകളുടെ തുക കാണുന്നതിന് നമുക്ക് sum(n) എന്ന ഫങ്ഷൻ താഴെ കൊടുത്തിരിക്കുന്നതു പോലെ എഴുതുവാൻ കഴിയും.

```
sum(n) = n + sum(n-1)
```

തന്നിരിക്കുന്ന ഒരു ദശ സംഖ്യയെ അതിന് തുല്യമായ ബൈനറി സംഖ്യയാക്കി മാറ്റുന്ന പ്രോഗ്രാം നമുക്ക് ചർച്ച ചെയ്യാം. അധ്യായം 2 ൽ പരിവർത്തന രീതി നാം ചർച്ച ചെയ്തു.

പ്രോഗ്രാം 10.11. ഒരു ഡസിമൽ നമ്പറിന് (ദശാംശ സംഖ്യയ്ക്ക്) തുല്യമായ ബൈനറി സംഖ്യ പ്രദർശിപ്പിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
void Binary(int);
int main()
{
    int decimal;
    cout<<"Enter an integer number: ";
    cin>>decimal;
    cout<<"Binary equivalent of "<<decimal<<" is ";
    Binary(decimal);
    return 0;
}

void Binary(int n)    //Definition of a recursive function
{
    if (n>1)
        Binary(n/2);
    cout<<n%2;
}
```

പ്രോഗ്രാം 10.11 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

`Enter an integer number: 19` ചെറിയ സംഖ്യ input നൽകി കൊണ്ട് പ്രോഗ്രാമിന്റെ പ്രവർത്തനം വിശദീകരിക്കുക.

`Binary equivalent of 19 is 10011`

10.7 ഹെഡർ ഫയലുകളുടെ നിർമ്മാണം

ഇതുവരെ നാം ചർച്ച ചെയ്ത എല്ലാ പ്രോഗ്രാമുകളുടേയും തുടക്കത്തിൽ #include <iostream> എന്നത് എല്ലായ്പ്പോഴും ഉപയോഗിക്കുന്നു എന്ന് നമുക്കറിയാം. നാം ഈ പ്രസ്താവന ഉപയോഗിക്കുന്നത് എന്തിന് വേണ്ടിയാണ്?

യഥാർത്ഥത്തിൽ C++ പ്രോഗ്രാമുകളിൽ നാം ഉപയോഗിക്കുന്ന ധാരാളം വേരിയബിളുകളുടേയും ഒബ്ജക്ടുകളുടേയും പ്രഖ്യാപനങ്ങളും നിർവചനങ്ങളും iostream എന്ന ഹെഡർ ഫയലിൽ അടങ്ങിയിരിക്കുന്നു. പ്രോഗ്രാമിൽ ഉപയോഗിക്കുന്ന cin, cout എന്നീ ഒബ്ജക്ടുകൾ ഈ ഹെഡൽ ഫയലിലാണ് പ്രഖ്യാപിച്ചിരിക്കുന്നത്. അതിനാൽ ഒരു പ്രോഗ്രാമിൽ ഈ ഹെഡർ ഫയൽ ഉൾപ്പെടുത്തുമ്പോൾ ഒബ്ജക്ടുകളുടെയും ഫങ്ഷനുകളുടെയും നിർവ്വചനങ്ങളും പ്രഖ്യാപനങ്ങളും കമ്പയിലറിന് കമ്പയിലേഷൻ സമയത്ത് ലഭ്യമാകും. *ഈ ഫങ്ഷനുകളുടേയും ഒബ്ജക്ടുകളുടേയും പ്രവർത്തിക്കാവുന്ന കോഡ് പ്രോഗ്രാമുമായി ബന്ധിപ്പിക്കുകയും അവ എപ്പോൾ എവിടെ വച്ച് വിളിക്കുമ്പോളും അവ പ്രവർത്തിക്കും.* നമ്മുടെ സ്വന്തം വേരിയബിളുകളും ഫങ്ഷനുകളും ഉൾക്കൊള്ളുന്ന ഹെഡൽ ഫയലുകൾ ഇതുപോലെ നമുക്ക് നിർമ്മിക്കാൻ കഴിയും. ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കാണുന്നതിനുള്ള ഒരു ഫങ്ഷൻ എഴുതി, ആ ഫങ്ഷൻ പല പ്രോഗ്രാമിലും ഉപയോഗിക്കണം എന്ന് കരുതുക. എല്ലാ പ്രോഗ്രാമുകളിലും ഫാക്ടോറിയൽ ഫങ്ഷൻ നിർവചിക്കുന്നതിന് പകരം ആ ഫങ്ഷനെ ഒരു ഹെഡർ ഫയലിൽ സ്ഥാപിക്കുകയും ഈ ഹെഡർ ഫയൽ എല്ലാ പ്രോഗ്രാമിലും ഉൾപ്പെടുത്തുകയും ചെയ്യാം.

നമുക്ക് ഒരു ഹെഡർ ഫയൽ എങ്ങനെ നിർമ്മിക്കാൻ കഴിയും എന്നത് താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം കാണിക്കുന്നു. ഏതെങ്കിലും IDE എഡിറ്ററിൽ താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം കൊടുക്കുക.

```
int factorial(int n)
{
    int f=1;
    for (int i=1; i<=n; i++)
        f *= i;
    return f;
}
```

ഈ ഫയൽ **factorial.h** എന്ന പേരിൽ സേവ് ചെയ്യുകയും അതിനുശേഷം താഴെ കൊടുത്തിരിക്കുന്നത് പോലെ ഒരു C++ പ്രോഗ്രാം നിർമിക്കുകയും ചെയ്യുക.

```
#include <iostream>
#include "factorial.h"//includes user-defined headerfile
using namespace std;
int main()
```

```
{
    int n;
    cout<<"Enter a number : ";
    cin >> n;
    cout<<"Factorial : " << factorial(n);
}
```

നമുക്ക് പ്രോഗ്രാം വിജയകരമായി കമ്പയിൽ ചെയ്യുവാനും പ്രവർത്തിക്കുവാനും കഴിയും.

`#include "factorial.h"` എന്ന നിർദ്ദേശം ആൻഗുലാർ ബ്രാക്കറ്റുകൾക്ക് (<,>) പകരം ഡബിൾ ക്വോട്ട്സ് ആണ് ഉപയോഗിച്ചത് എന്നത് ശ്രദ്ധിക്കുക. ഇത് എന്തുകൊണ്ടെന്നാൽ ഒരു ഫയൽ ഉൾപ്പെടുത്തുന്നതിനായി നാം ആൻകുലാർ ബ്രാക്കറ്റുകൾ (< >) ഉപയോഗിക്കുമ്പോൾ കമ്പയിൽ അതിനെ `include` ഡയറക്ടറിയിൽ പരതും. എന്നാൽ നാം ഡബിൾ ക്വോട്സ് ഉയോഗിക്കുമ്പോൾ അതിനെ ഇപ്പോൾ പ്രവർത്തിക്കുന്ന ഡയറക്ടറിയിൽ മാത്രമേ പരതുകയുള്ളൂ. സാധാരണയായി `factorial.h` എന്ന ഫയൽ സേവ് ചെയ്യുന്നത് C++ പ്രോഗ്രാം സേവ് ചെയ്യപ്പെട്ട അതേ ഡയറക്ടറിയിലായിരിക്കും. അതുകൊണ്ട് ഫയൽ ഉൾപ്പെടുത്താൻ നാം ഉദ്ധരണി ഉപയോഗിക്കണം. ഏതെങ്കിലും ഒരു C++ പ്രോഗ്രാമിന് ഫാക്ടോറിയൽ ഫങ്ഷൻ ഉപയോഗിക്കേണ്ട ആവശ്യം ഉണ്ടെങ്കിൽ `#include" factorial.h"` എന്ന സ്റ്റേറ്റ്മെന്റ് പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ട ആവശ്യം മാത്രമേ ഉള്ളൂ. ഇതേ രീതിയിൽ ഒരു ഹെഡർ ഫയലിൽ എത്ര ഫങ്ഷനുകൾ വേണമെങ്കിലും ഉൾപ്പെടുത്തുകയും ഏത് പ്രോഗ്രാമിലും ഈ ഫങ്ഷനുകൾ ഉപയോഗിക്കുകയും ചെയ്യാം.

സ്വയം പരിശോധിക്കാം

1. ഒരു ഫങ്ഷന്റെ പ്രോട്ടോടൈപ്പ് പ്രീ. പ്രോസസർ ഡിറക്ടീവിന് ശേഷം നൽകുമ്പോൾ അവയുടെ വ്യാപ്തി .... ആയിരിക്കും.
2. സ്വയം ആവർത്തനം എന്നാൽ എന്ത്?
3. C++ ലെ മുൻ നിർവ്വചിത ഫങ്ഷനുകളുടെ വ്യാപ്തി എന്ത് ആകുന്നു.
4. ഒരു ഫങ്ഷന്റെ ആർഗ്യുമെന്റുകൾക്ക് .... വ്യാപ്തി ഉണ്ട്.

നമുക്ക് സംഗ്രഹിക്കാം

പ്രോഗ്രാമിങ്ങ് എളുപ്പത്തിൽ ആക്കുന്ന ഒരു സമീപനമാണ് മോഡുലാർ പ്രോഗ്രാമിങ്. C++ ഫങ്ഷനിലൂടെ മോഡുലറൈസേഷന് സൗകര്യം ഒരുക്കുന്നു. ഒരു പ്രത്യേക ഉദ്യമം നടത്തുന്നതിന് പ്രോഗ്രാമിൽ ഉൾക്കൊള്ളിച്ചിരിക്കുന്ന പേരോടു കൂടിയ ഒരു ഘടകമാണ് ഫങ്ഷൻ. C++ൽ മുൻ നിർവചിത ഉപയോക്തൃ നിർവചിത എന്നീ രണ്ട് തരം ഫങ്ഷനുകൾ ഉണ്ട്. മുൻ നിർവചിത ഫങ്ഷനുകൾ ഉപയോഗിക്കണമെങ്കിൽ അതുമായി ബന്ധപ്പെട്ട ഹെഡർ ഫയൽ നാം പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തണം. വിളിക്കുന്ന ഫങ്ഷന് ശേഷമാണ് നിർവ്വചിച്ചിരിക്കുന്നതെങ്കിൽ അത്തരം ഫങ്ഷൻ പ്രഖ്യാപിക്കേണ്ടത് ആവശ്യമാണ്. ഫങ്ഷൻ വിളിക്കുമ്പോൾ വിളിക്കുന്ന ഫങ്ഷനിൽ നിന്നും വിളിച്ച ഫങ്ഷനിലേക്ക് ഡാറ്റ ആർഗ്യുമെന്റിലൂടെ അയച്ചേക്കാം. ആർഗ്യൂമെന്റുകളെ യഥാക്രമ (ഫോർമൽ) പരാമീറ്റർ, ആച്ചൽ (യഥാർത്ഥ) പരാമീറ്റർ എന്നിങ്ങനെ രണ്ടായി തരംതിരിക്കാം. ഫങ്ഷനിലേക്ക് പരാമീറ്റർ അയയ്ക്കുന്നതിന് കാൾ-ബൈ-വാല്യൂ രീതിയോ അല്ലെങ്കിൽ കാൾ ബൈ റഫറൻസ് രീതിയോ ഉപയോഗിക്കാം. ഒരു പ്രോഗ്രാമിലെ വേരിയബിളുകൾക്കും ഫങ്ഷനുകൾക്കും അവ പ്രഖ്യാപിച്ചിരിക്കുന്ന സ്ഥലത്തിനനുസരിച്ച് വ്യാപ്തിയും ജീവനവും ഉണ്ട്. ഒരു ഫങ്ഷൻ, മറ്റൊരു ഫങ്ഷനെ വിളിക്കാമെന്നതു പോലെ, ഒരു ഫങ്ഷൻ അതിനെ തന്നെ വിളിക്കുന്ന പ്രവർത്തനമായ സ്വയം ആവർത്തനവും C++ അനുവദിക്കുന്നു. ഉപയോക്തൃ നിർവചിത ഫങ്ഷനുകൾ ശേഖരിക്കാൻ പുതിയ ഹെഡർഫയലുകളും നമുക്ക് നിർമിക്കാൻ കഴിയും. അതിലൂടെ ഈ ഫങ്ഷനുകൾ മറ്റ് പ്രോഗ്രാമുകളിലും ഉപയോഗിക്കാം.

പഠന നേട്ടങ്ങൾ

ഈ അധ്യായം പൂർത്തിയാക്കിയതിന് ശേഷം പഠിതാവിന് പ്രാപ്തമാകുന്നത്

- മോഡുലാർ പ്രോഗ്രാമിങ്ങ് ശൈലിയും അവയുടെ മേന്മകളും തിരിച്ചറിയുന്നു.
- പ്രശ്നപരിഹാരത്തിനായി മുൻ നിർവചിത ഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നു.
- പ്രശ്നപരിഹാരത്തിൽ ഏർപ്പെട്ടിരിക്കുന്ന പ്രത്യേക ഉദ്യമങ്ങൾ പ്രവർത്തിക്കുന്നതിനായി ഉപഫങ്ഷനുകൾ നിർവ്വചിക്കുന്നു.
- ഉപയോക്താവ് നിർവ്വചിച്ച ഉപഫങ്ഷനുകൾ ഉപയോഗിക്കുന്നു.
- സ്വയം ആവർത്തന ഫങ്ഷനുകൾ നിർവ്വചിക്കുകയും പ്രശ്ന പരിഹാരത്തിന് അവ ഉപയോഗിക്കുകയും ചെയ്യുന്നു.

ലാബ് പ്രവർത്തനങ്ങൾ

1. ഒരു സംഖ്യ സ്വീകരിച്ച് അത് അഭിഭാജ്യം ആണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ 0 ഉം തിരിച്ച് നൽകുന്നതിനുള്ള ഒരു ഫങ്ഷൻ നിർവ്വചിക്കുക. ഈ ഫങ്ഷൻ ഉപയോഗിച്ച് 100 നും 200 നും ഇടക്കുള്ള എല്ലാ അഭാജ്യ സംഖ്യകളും പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.
2. ഒരു ഫങ്ഷൻ ഉപയോഗിച്ച് തന്നിരിക്കുന്ന മൂന്ന് അല്ലെങ്കിൽ രണ്ട് സംഖ്യകളിൽ ഏറ്റവും ചെറിയ സംഖ്യ കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക (തനത് ആർഗ്യൂമെന്റുകളുടെ ആശയം ഉപയോഗിക്കുക).
3. ഒരു ഉപയോക്തൃ നിർവചിത ഫങ്ഷന്റെ സഹായത്തോടെ ഒരു സംഖ്യയിലെ അക്കങ്ങളുടെ തുക കണ്ടുപിടിക്കുക. (അതായത് സംഖ്യ 3245 ആണെങ്കിൽ, ഉത്തരം 3+2+4+5 = 14 ആയിരിക്കണം).
4. ഒരു ഫങ്ഷൻ ഉപയോഗിച്ച് തന്നിരിക്കുന്ന രണ്ട് സംഖ്യകളുടെ LCM കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.
5. ഒരു ഫങ്ഷൻ ഉപയോഗിച്ച് തന്നിരിക്കുന്ന പരിധിയിൽപ്പെട്ട എല്ലാ പാലിൻഡ്രോം നമ്പറുകളും പ്രദർശിപ്പിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. ഫങ്ഷൻ സംഖ്യ സ്വീകരിക്കുകയും പാലിൻഡ്രോം ആണെങ്കിൽ 1 ഉം അല്ലെങ്കിൽ പൂജ്യവും തിരിച്ചു നൽകണം.

മാതൃകാ ചോദ്യങ്ങൾ

ലഘുചോദ്യോത്തരങ്ങൾ

1. പ്രോഗ്രാമിങ്ങിൽ ടോപ്-ഡൗൺ, ബോട്ടം-അപ്പ് രൂപകൽപ്പനകൾ എങ്ങനെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു?
2. C++ ലെ ഒരു ഫങ്ഷൻ എന്താണ്?
3. ഒരു ഫങ്ഷന് അതിനെ തന്നെ വിളിക്കാനുള്ള കഴിവിനെ .... എന്ന് പറയുന്നു.
4. C++ പ്രോഗ്രാമുകളിൽ ഹെഡർ ഫയലുകളുടെ കർത്തവ്യം എഴുതുക.
5. ഫങ്ഷൻ നിർവചനത്തിനായി വോയിഡ് (void) ഡാറ്റ തരം ഉപയോഗിക്കുന്നത് എപ്പോഴാണ്?

ലഘു ഉപന്യാസതരം :

1. ആക്ചൽ (യഥാർത്ഥ) ഫോർമൽ (യഥാക്രമം) എന്നീ ആർഗ്യൂമെന്റുകൾ തമ്മിലുള്ള വ്യത്യാസങ്ങൾ കണ്ടെത്തുക.
2. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷനുകൾക്ക് വേണ്ട ഫങ്ഷൻ പ്രോട്ടോടൈപ്പുകൾ നിർമിക്കുക.
   a) Total () - രണ്ട് ഡബിൾ ആർഗ്യൂമെന്റുകൾ സ്വീകരിച്ച് ഒരു ഡബിൾ ഡാറ്റതരം തിരിച്ച് നൽകുന്നു

b) Math () - ഒരു വിലയും സ്വീകരിക്കുകയോ തിരിച്ചു നൽകുകയോ ചെയ്യുന്നില്ല.

3. റിട്ടേൺ പ്രസ്താവന, exit() ഫങ്ഷൻ എന്നിവ വേർതിരിച്ച് എഴുതുക.
4. ലോക്കൽ, ഗ്ലോബൽ എന്നീ വേരിയബിളുകളുടെ വ്യാപ്തി ഉദാഹരണ സഹിതം ചർച്ച ചെയ്യുക.
5. ഫങ്ഷനെ വിളിക്കുന്നതിന് ഉപയോഗിക്കുന്ന കാൾ-ബൈ-വാല്ല്യൂ രീതിയും കാൾ-ബൈ-റഫറൻസ് രീതിയും തമ്മിലുള്ള വ്യത്യാസങ്ങൾ കണ്ടെത്തുക.
6. C++ ൽ എല്ലാം ആർഗ്യുമെന്റുകളും ഉപയോഗിക്കാതെ ഫങ്ഷൻ വിളിക്കുവാൻ കഴിയും. എങ്ങനെ?
7. സ്വയം ആവർത്തനത്തിൽപ്പെട്ടിരിക്കുന്ന പ്രവർത്തനം എഴുതുക.

ദീർഘ ചോദ്യോത്തരം

1. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷൻ നോക്കുക

```
int sum(int a,int b=o,int(=o)
{relarn (a+b+c)}
```

a പരാമീറ്റർ ലിസ്റ്റിനെ സംബന്ധിച്ച് ഫങ്ഷന്റെ പ്രത്യേകതകൾ എന്താണ്?

b താഴെകൊടുത്തിരിക്കുന്ന ഫങ്ഷനുകളുടെ ഔട്ട്പുട്ട് എഴുതി അതിന്റെ പ്രവർത്തനം വിശദമാക്കുക ഫങ്ഷൻ കാൾ തെറ്റാണെങ്കിൽ അതിന്റെ കാരണം എഴുതുക.

```
(i) cont<<sum (1,2,3):   (ii)cont <<sum (5,2);
(iii) cout<<sun();        (1v) cout <<sum(o);
```

2 `int fun (in, inl);` എന്നത് ഒരു ഫങ്ഷന്റെ പ്രോട്ടോടൈപ്പ് ആണ്. താഴെ കൊടുത്തിരിക്കുന്ന ഫങ്ഷൻ വിളികൾ അസാധുവാണ് ഓരോന്നിന്റെയും കാരണം എഴുതുക.

```
a)fun(2,4);        b) cout<<fun();   c) val=fun(2.5,3.3);
d)cin>>fun(a,b);   e) 2=fun(3);
```