OOP in C++ (အပိုင္း-၁)

Dr. ေအာင္ဝင္းထြဋ္ (BluePhoenix)

License: MIT Opensource License



စာေရးသူ၏ အမွာစာ

ကၽြန္ေတာ္႔အေနနဲ႔ C နဲ႔ C++ ကို ၁၉၉၉ ခုႏွစ္ေလာက္မွာ စတင္ေလ႔လာခဲ႔ပါတယ္။ အဲဒီႏွစ္ထဲမွာပဲ တကၠသိုလ္ကြန္ပ်ဴတာခန္းထဲက ကြန္ပ်ဴတာေတြထဲမွာ ျပန္႔ၾကဲေနတဲ႔ Turbo C version 2.0 ဖိုင္ေလးေတြကို စုစည္းၿပီး C++ နဲ႔ installer software ေရးျဖစ္ခဲ႔ပါတယ္။ အဲဒီအေတြ႔အၾကံဳကို ၂၀၀၀ ခုႏွစ္ ေအာက္တိုဘာလထုတ္ ကြန္ပ်ဴတာ ဂ်ာနယ္ မွာ Installation software construction ဆိုတဲ႔ ေခါင္းစဥ္နဲ႔ ေဆာင္းပါး တစ္ေစာင္ ေရးသားခဲ႔ပါတယ္။ အဲဒီေဆာင္းပါးဟာ ကၽြန္ေတာ္႔ ဘဝမွာ ပထမဆံုး ပံုႏွိပ္ေဖာ္ျပခံခဲ႔ရတဲ႔ နည္းပညာ ေဆာင္းပါး တစ္ေစာင္ပဲ ျဖစ္ခဲ႔ပါတယ္။ ဒီလိုနဲ႔ ကၽြန္ေတာ္႔ ငယ္ဘဝမွာ အစြဲအလမ္းအႀကီးဆံုးက C++ ဘာသာစကား ျဖစ္မွန္းမသိ ျဖစ္လာခဲ႔ပါတယ္။

မိမိဘာသာ အမ်ားဆံုး ေလ႔လာခဲ႔ရတဲ႔အတြက္ C++ လို႔ဆိုေပမယ္႔ တကယ္တမ္းက ေက်ာင္းမွာ သင္ခဲ႔ဘူးတဲ႔ C language style နဲ႔ပဲ ေရးျဖစ္တာ မ်ားပါတယ္။ OOP ဆိုတာကို အဲဒီတုန္းက ၾကားဘူးရံုပဲ ရွိပါတယ္။ ေနာက္ပိုင္း ႏွစ္ေတာ္ေတာ္ၾကာ အခ်ိန္ယူၿပီး ျဖည္းျဖည္းခ်င္း ေလ႔လာခဲ႔ပါတယ္။

လြန္ခဲ႔တဲ႔ ၃ ႏွစ္ေလာက္က စၿပီး ကၽြန္ေတာ္နဲ႔ ရင္းႏွီးတဲ႔ YCC ေက်ာင္းသားအခ်ိဳ႕ စာလာေမးတာေၾကာင္႔ သူတို႔ လက္ရွိ အသံုးျပဳေနတဲ႔ သင္ရိုး စာအုပ္ေတြကို ဖတ္ၿပီး ျပန္ရွင္းျပ ေပးခဲ႔ရပါတယ္။ အဲဒီစာအုပ္ေတြထဲက စာမ်က္ႏွာ တစ္ေထာင္ေက်ာ္ပါတဲ႔ Robert Lafore ရဲ႕ Object-Oriented Programming in C++ (Fourth Edition) စာအုပ္ေလးဟာ C++ ကို OOP သံုးၿပီး ေရးသား နည္းေတြကို အေျခခံက်က် ရွင္းျပထားတဲ႔ စာအုပ္ေကာင္းတစ္အုပ္ ျဖစ္တယ္ဆိုတာကို သတိထားမိခဲ႔ပါတယ္။ ဒါေၾကာင္႔ အပတ္စဥ္ အခ်ိန္အနည္းငယ္ယူၿပီး C++ သင္ခန္းစာေလးေတြကို အဲဒီ စာအုပ္ကေန ေကာက္ႏႈတ္ ေရးသားဖို႔ ဆံုးျဖတ္လိုက္တာ ျဖစ္ပါတယ္။ ဒီသင္ခန္းစာမ်ားဟာ C++ နဲ႔ OOP ကို စတင္ေလ႔လာေနတဲ႔ ေက်ာင္းသားမ်ားကိုသာ အဓိက ရည္ရြယ္ ေရးသားသြားမွာ ျဖစ္ပါတယ္ ခင္ဗ်ာ။

ေဒါက္တာ ေအာင္ဝင္းထြဋ္

7-5-2015

ျပင္ဦးလြင္ၿမိဳ႕

**အခန္း(၁)**

C++ Programming Basics

**C++ ကို စတင္ေလ႔လာျခင္း**

ပရိုဂရမ္းမင္း ဘာသာစကားတိုင္းမွာ အေျခခံအက်ဆံုး ပရိုဂရမ္ အေသးေလးေတြ ေရးဖို႔အတြက္ေတာင္ မရွိမျဖစ္ လိုအပ္တဲ႔ အေျခခံ စည္းမ်ဥ္းေတြ ရွိၾကပါတယ္။ ဒီအခန္းမွာေတာ႔ basic program construction, variables ေတြနဲ႔ input/output (I/O) ေတြအေၾကာင္းကို အေျခခံက်က် ရွင္းလင္း တင္ျပေပးသြားမွာပါ။ ဒါ႔အျပင္ comments ေရးသားနည္း၊ arithmetic operators မ်ားႏွင္႔ increment operator အသံုးျပဳနည္း၊ data conversion ျပဳလုပ္ျခင္းနဲ႔ library functions အသံုးျပဳနည္းမ်ားကိုပါ ထည္႔သြင္းေဆြးေႏြးေပးသြားမွာပါ။ စာဖတ္သူတို႔အေနနဲ႔ ဘာေတြ ေျပာေနတာလဲဆိုၿပီး စိတ္မညစ္သြားၾကပါနဲ႔။ ဒီအခန္းကို ဆံုးေအာင္ ႀကိဳးစားဖတ္ၾကည္႔လိုက္တာနဲ႔ ဒီအေၾကာင္းအရာေတြကို ကိုယ္တိုင္ ေရးသားအသံုးျပဳႏိုင္ေအာင္ ကၽြမ္းက်င္သြားမွာပါ။

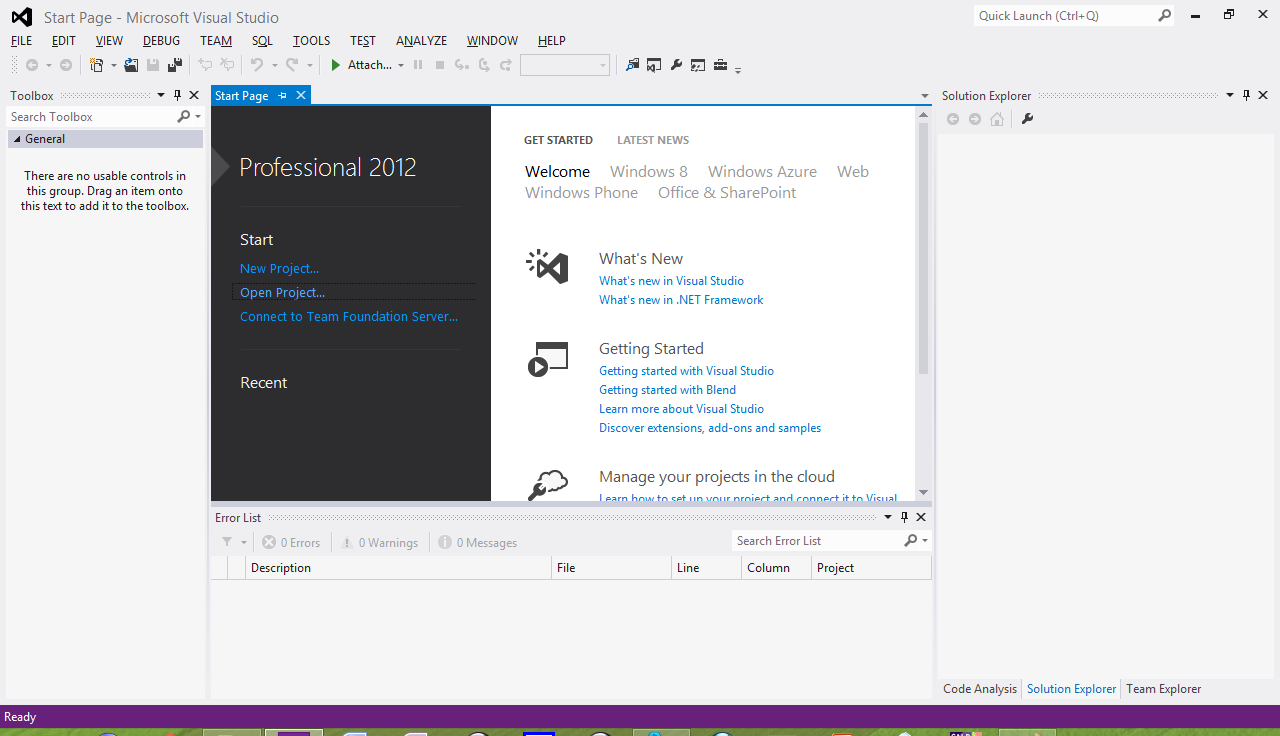
**C++ ကို ဘယ္လို compiler ေတြ အသံုးျပဳၿပီး ေရးၾကမလဲ ?**

ကၽြန္ေတာ္ စတင္ေလ႔လာကာစ အခ်ိန္မ်ားတုန္းက Turbo C++ 3.0 ကို အဓိက သံုးၿပီး DOS command ေတြနဲ႔ အလုပ္ရႈပ္ခဲ႔ၾကပါတယ္။ အခု အခ်ိန္မွာေတာ႔ ေက်ာင္းသားအမ်ားစုဟာ C-Free ဆိုတဲ႔ compiler ကို သံုးေနၾကတာ သတိထားမိခဲ႔ပါတယ္။ ဒါေပမယ္႔ ကၽြန္ေတာ္႔အေနနဲ႔ Visual Studio 2008 ဒါမွ မဟုတ္ Visual Studio 2012 ကို သံုးသင္႔တယ္လို႔ အၾကံျပဳပါရေစ။ ဒီေဆာင္းပါးမ်ားမွာေတာ႔ Visual Studio 2012 ကို အသံုးျပဳၿပီး ေရးသားတင္ျပသြားမွာ ျဖစ္ပါတယ္ ခင္ဗ်ာ။

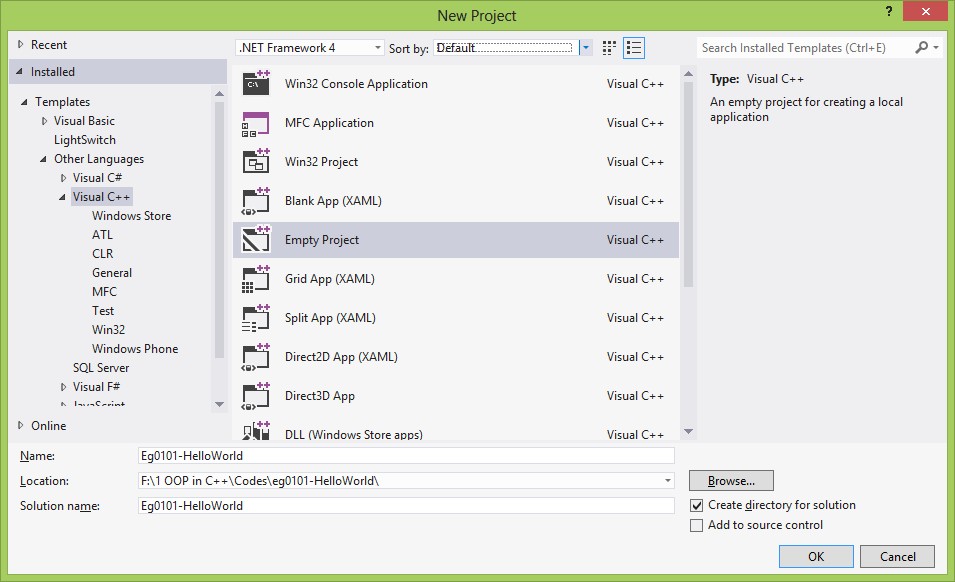
တကယ္ေတာ႔ compiler ဆိုတာက ကၽြန္ေတာ္တို႔ ေရးသားထားတဲ႔ soruce files **(.CPP)** ဖိုင္ေတြကို Windows ေပၚမွာ RUN လို႔ ရတဲ႔ executable files **(.EXE)** အျဖစ္ ေျပာင္းလဲေပးတဲ႔ ပရိုဂရမ္ေလး တစ္ခုပဲ ျဖစ္ပါတယ္။ အဲဒီ **(.EXE)** ဖိုင္ေလးေတြကို compile ေတြထဲကေန တိုက္ရိုက္ RUN လို႔ ရသလို DOS command prompt ကလည္း RUN လို႔ ရပါတယ္။

**Visual Studio 2012 ကို အသံုးျပဳ၍ C++ program တစ္ပုဒ္ ေရးသားစမ္းသပ္ျခင္း**

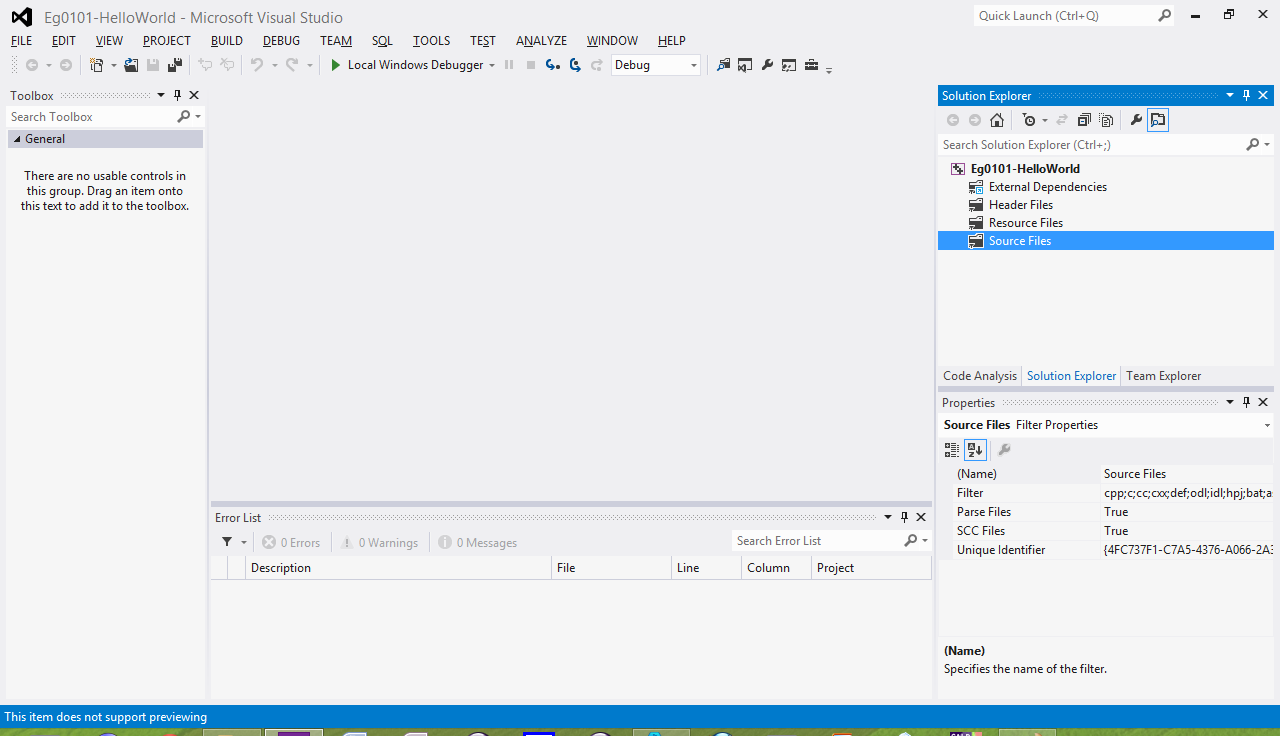
ပထမဆံုး Visual Studio 2012 ကို ဖြင္႔လိုက္ပါ။ ေအာက္ပါ အတိုင္း ျမင္ရႏိုင္ပါတယ္။ ကိုယ္တင္ထားတဲ႔ version အေပၚ မူတည္ၿပီး အတိအက်ေတာ႔ တူခ်င္မွ တူပါလိမ္႔မယ္။



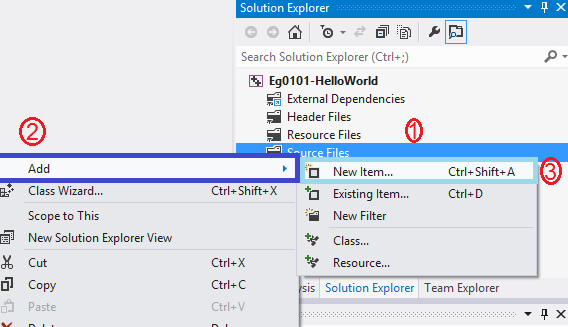
New Project... ကို ႏွိပ္လိုက္ပါ။ ေအာက္ပါအတိုင္း New Project dialog box ေပၚလာပါလိမ္႔မယ္။



Installed->Templates->Other Languages->Visual C++ ကို ေရြးလိုက္ပါ။ ညာဘက္က ေပၚလာတဲ႔ Visual C++ project အမ်ိဳးအစားေတြထဲက Empty Project ကို ေရြးခ်ယ္လိုက္ပါ။ Name ေနရာမွာ ကိုယ္ေရးမယ္႔ ပရိုဂရမ္ရဲ႕ နာမည္ကို ေရးပါ။ ဖိုင္ေတြ သိမ္းဖို႔ ေနရာကို Browse... လုပ္ၿပီး ရွာေပးလိုက္ပါ။ OK ကို ႏွိပ္လိုက္ရင္ ေအာက္ပါအတိုင္း ျမင္ရပါလိမ္႔မယ္။



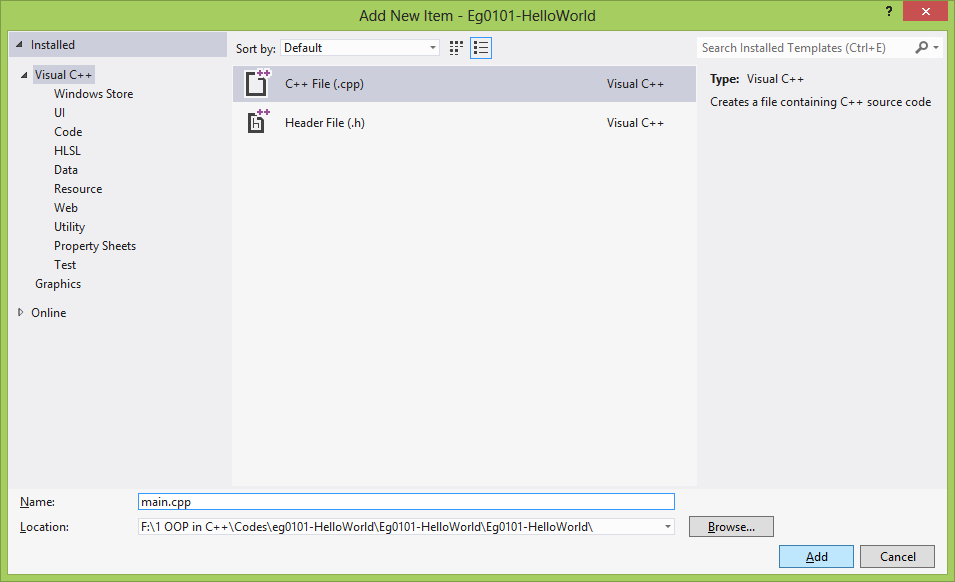
Empty Project ကို ျပဳလုပ္ထားတာ ျဖစ္လို႔ Source Files ဆိုတဲ႔ folder ထဲမွာ ဘာဖိုင္မွ မရွိေသးပါဘူး။ **.CPP** ဖိုင္တစ္ခု ထည္႔သြင္းဖို႔ ေအာက္ပါ အဆင္႔မ်ားအတိုင္း လုပ္ေဆာင္ပါ။



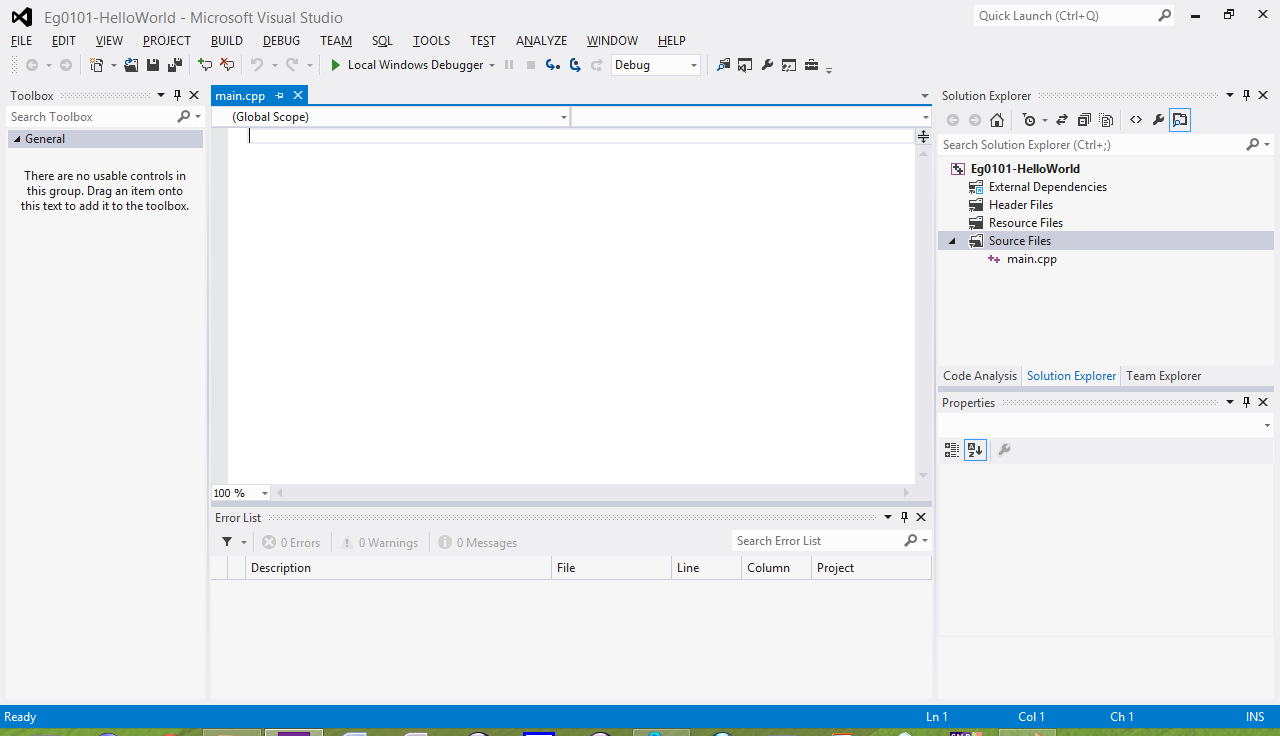
(1) Solution Explorer ထဲက Source Files ကို right click ျပဳလုပ္ပါ။

(2) Add ကို ေရြးခ်ယ္ပါ။

(3) New Item.. ကို ကလစ္ ျပဳလုပ္ပါ။



အထက္ပါ ပံုအတိုင္း Add New Item dialog box ေပၚလာရင္ C++ File(.cpp) ကို ေရြးခ်ယ္ပါ။ Name ေနရာတြင္ မိမိ ေရးသားမယ္႔ ပရိုဂရမ္ နာမည္ (ဒီဥပမာမွာေတာ႔ main.cpp) ေပးၿပီး Add ကိုႏွိပ္လိုက္ပါ။ ကိုယ္ေပးလိုက္တဲ႔ နာမည္နဲ႔ .cpp ဖိုင္ အလြတ္ တစ္ခုကို ေတြ႔ရပါလိမ္႔မယ္။



အခုဆိုရင္ coding မ်ားကို စတင္ ဝင္ေရာက္ ေရးသားႏိုင္ၿပီ ျဖစ္ပါတယ္။ ေအာက္ပါ စာသားမ်ားကို ရိုက္ထည္႔လိုက္ပါမယ္။

#include<iostream>

using namespace std;

int main()

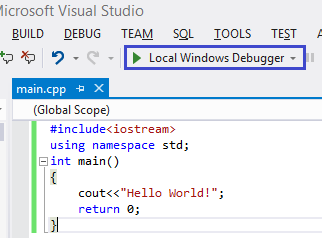
{

cout<<"Hello World!";

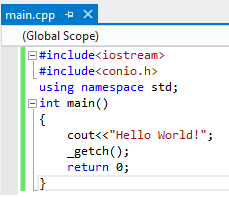
return 0;

}

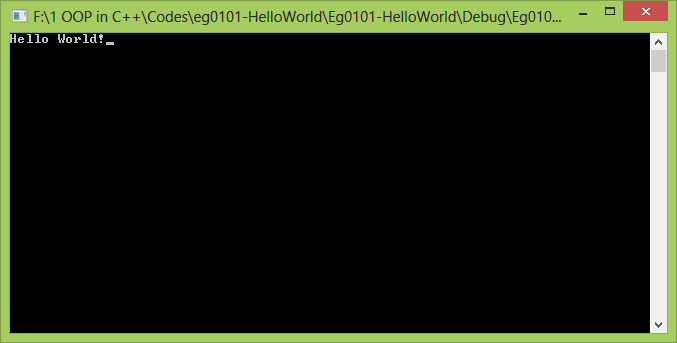
ဒီပရိုဂရမ္ေလးဟာ အရိုးရွင္းဆံုး C++ ပရိုဂရမ္ေလးပဲ ျဖစ္ပါတယ္။ Hello World! ဆိုတဲ႔ စာသားေလးကို ကြန္ပ်ဴတာ ဖန္သားျပင္ေပၚမွာ ျပသေပးမယ္႔ ပရိုဂရမ္ေလးပါ။ RUN ၾကည္႔ဖို႔အတြက္ Toolbar က Local Windows Debugger ဆိုတာေလးကို ႏွိပ္လိုက္ပါ။



Screen အမဲေလးတစ္ခု ေပၚလာၿပီး ခ်က္ခ်င္း ျပန္ေပ်ာက္သြားပါလိမ္႔မယ္။ အဲဒီ output ကို ၾကည္႔ဖို႔ Visual Studio 2012 မွာ ကုဒ္ အပိုေလးေတြ ထပ္ထည္႔ေပးရပါမယ္။ ဒါမွ Screen ေပ်ာက္မသြားဘဲ key တစ္ခ်က္ အႏွိပ္ကို ေစာင္႔ေနမွာ ျဖစ္ပါတယ္။ \_getch() ဆိုတဲ႔ function ေလးကို return 0; ရဲ႕ ေရွ႕မွာ ထပ္ထည္႔ေပးရပါမယ္။ ေနာက္ၿပီး သူ႔ရဲ႕ header file ျဖစ္တဲ႔ conio.h ကို လည္း include ျပဳလုပ္ေပးရမွာ ျဖစ္ပါတယ္။ ခုဆိုရင္ ကုဒ္က ေအာက္ပါအတိုင္း ျဖစ္သြားပါၿပီ။



အခုအခ်ိန္ ျပန္ RUN ၾကည္႔လိုက္မယ္ ဆိုရင္ေတာ႔ ေအာက္ပါအတိုင္း ေတြ႔ရမွာ ျဖစ္ပါတယ္။



ပရိုဂရမ္ကို အဆံုးသတ္ဖို႔ Key တစ္ခုခု ႏွိပ္လိုက္ရံုပဲ ျဖစ္ပါတယ္။

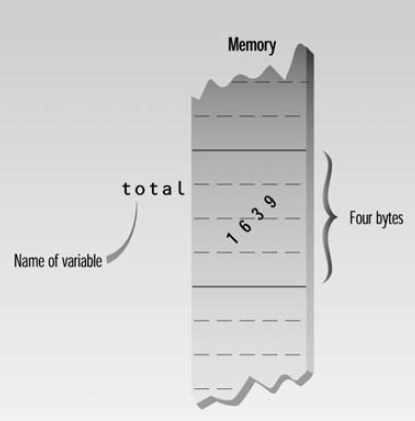
**Data types and Variables**

Variables ေတြဟာ ပရိုဂရမ္းမင္း ဘာသာတိုင္းအတြက္ အေျခခံ အက်ဆံုး အစိတ္အပိုင္းမ်ား ျဖစ္ၾကပါတယ္။ အခ်က္အလက္ေတြကို ယာယီ သိမ္းဆည္းဖို႔ Variable ေတြကို နာမည္ သတ္မွတ္ေပးထားၿပီး ဖန္တီးႏိုင္သလို တန္ဖိုးေတြလဲ သတ္မွတ္ ထည္႔သြင္းႏိုင္ပါတယ္။ Variables ေတြကို ကြန္ပ်ဴတာ မွတ္ဥာဏ္ (RAM) ရဲ႕ ေနရာအခ်ိဳ႕မွာ ဖန္တီးထားတာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ Variable တစ္ခုကို တန္ဖိုးတစ္ခု ထည္႔သြင္း လိုက္တယ္ဆိုတာဟာ တကယ္ေတာ႔ အဲဒီ Variable ကို သတ္မွတ္ေပးထားတဲ႔ RAM ထဲမွာ အဲဒီတန္ဖိုးကို သိမ္းဆည္းထားလိုက္တာပဲ ျဖစ္ပါတယ္။ အဲဒီလို သိမ္းဆည္းထားတဲ႔ ေနရာမွာ အခ်က္အလက္ရဲ႕ အမ်ိဳးအစားေပၚမူတည္ၿပီး ခြဲျခားသိမ္းဆည္းေပးဖို႔ လိုပါတယ္။ ဒါေၾကာင္႔ Variables ေတြကို data types ေတြ ခြဲျခားၿပီး ဖန္တီးေလ႔ ရွိပါတယ္။ ပရိုဂရမ္းမင္း ဘာသာစကားအမ်ားစုမွာ သံုးေလ႔ ရွိတဲ႔ data types ေတြကေတာ႔ integers၊ floating-point numbers ေတြနဲ႔ characters ေတြပဲ ျဖစ္ၾကပါတယ္။

Integer variable ေတြဟာ 130000၊ ဒါမွ မဟုတ္ -23 တို႔လို အျပည္႔ကိန္းေတြကို ေရတြက္ ထည္႔သြင္းရာမွာ အသံုးျပဳပါတယ္။ floating-point numbers ေတြနဲ႔ မတူတဲ႔ အခ်က္ကေတာ႔ integers ေတြမွာ ဒႆမေနာက္က တန္ဖိုးေတြ (အပိုင္းကိန္း တန္ဖိုးေတြ) မပါရွိတာပဲ ျဖစ္ပါတယ္။

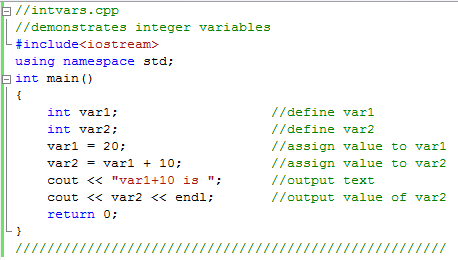
**Defining Integer Variables**

Integer variable ေတြ အမ်ိဳးအစားမ်ားစြာ ရွိပါတယ္။ ဒါေပမယ္႔ အသံုးအမ်ားဆံုးကေတာ႔ int ပဲ ျဖစ္ပါတယ္။ ေနရာယူတဲ႔ မွတ္ဥာဏ္ အရြယ္အစားကေတာ႔ system ေပၚ မူတည္ၿပီး ကြဲျပားႏိုင္ပါတယ္။ ဥပမာ- 32-bit system Windows ေတြမွာ int ဟာ 4 bytes (32 bits) ေနရာယူပါတယ္။ ဒါေၾကာင္႔ int ဟာ -2,147,483,648 ကေန 2,147,483,647 ၾကား တန္ဖိုးရွိတဲ႔ ဂဏန္းေတြကို သိမ္းဆည္းေပးႏိုင္ပါတယ္။



ဒါေပမယ္႔ ယခင္ Windows system အေဟာင္းေတြနဲ႔ MS-DOS စနစ္ေတြမွာေတာ႔ int ဟာ 2 bytes ပဲ ေနရာယူပါတယ္။ data types ေတြ ေနရာယူတဲ႔ သတ္မွတ္ခ်က္ေတြကိုေတာ႔ LIMITS ဆိုတဲ႔ header file ထဲမွာ ေရးသားထားပါတယ္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ လက္ရွိသံုးေနတဲ႔ compiler ရဲ႕ help ကေနလည္း ရွာေဖြ ဖတ္ရႈႏိုင္ပါတယ္။

ဥပမာ-အေနနဲ႔ int data type သံုးထားတဲ႔ ပရိုဂရမ္ေလးတစ္ခုကို ေရးျပပါ႔မယ္။



အထက္ပါ ပရိုဂရမ္ေလးမွာ **int var1; int var2;** ဆိုတဲ႔ ကုဒ္ႏွစ္ေၾကာင္းဟာ integer variable var1 နဲ႔ var2 တို႔ကို ေၾကျငာ သတ္မွတ္ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။ ေရွ႕က int ဆိုတဲ႔ keyword ေလးက variable ရဲ႕ type ကို သတ္မွတ္ေပးတာ ျဖစ္ပါတယ္။ အဲဒီ statement ေတြကို declaration ေတြလို႔ ေခၚဆိုၿပီး အျခား program statements ေတြလိုပဲ စာေၾကာင္း အဆံုးမွာ semicolon (**;**) နဲ႔ ပိတ္ေပးရမွာ ျဖစ္ပါတယ္။

ကၽြန္ေတာ္တို႔ အေနနဲ႔ variables ေတြကို အသံုးမျပဳခင္မွာ ႀကိဳတင္ ေၾကျငာေပးထားရမွာ ျဖစ္ပါတယ္။ ဆန္သြားဝယ္မယ္႔သူဟာ ဆန္အိတ္ကို ႀကိဳယူသြားရသလိုပါပဲ။ ဒါေၾကာင္႔ မ်ားေသာအားျဖင္႔ ပရိုဂရမ္ရဲ႕ အစပိုင္းမွာ variable ေတြကို ေၾကျငာထားေလ႔ ရွိပါတယ္။ ဒါေပမယ္႔ တကယ္က ပရိုဂရမ္ရဲ႕ မည္သည္႔ ေနရာမွာမဆို variable ေတြကို ေၾကျငာေပး ႏိုင္ပါတယ္။

**Declarations and Definitions**

Declaration နဲ႔ Definition ၾကားက ျခားနားခ်က္ကို ရွင္းျပခ်င္ပါတယ္။ Declare လုပ္တာဟာ variable ကို ဘယ္လိုေခၚမယ္၊ ဘာ data type ျဖစ္တယ္ဆိုတာကို program အတြင္းမွာ မိတ္ဆက္ ေၾကျငာေပးတာ ျဖစ္ပါတယ္။ Definition ဆိုတာကေတာ႔ memory ထဲမွာ အဲဒီ variable ကို ေနရာယူလိုက္တာပဲ ျဖစ္ပါတယ္။ အထက္က ပရိုဂရမ္ထဲမွာ ေရးထားတဲ႔ int var1; int var2; ဆိုတဲ႔ ကုဒ္ႏွစ္ေၾကာင္းဟာ declarations ေတြ ျဖစ္သလို definition ေတြလည္း ျဖစ္ပါတယ္။ မ်ားေသာအားျဖင္႔ အခ်ိဳ႕ေနရာေတြမွာ အဲဒီ အခ်က္ႏွစ္ခုက တူေလ႔ ရွိပါတယ္။ ဒါေပမယ္႔ ေနာက္ပိုင္းမွာ definitions မဟုတ္တဲ႔ declarations ေတြ အေၾကာင္းကို အေသးစိတ္ ရွင္းျပသြားမွာ ျဖစ္ပါတယ္။

**Variable Names**

Variables ေတြကို နာမည္ေပးတဲ႔ ေနရာမွာ လိုက္နာရတဲ႔ စည္းကမ္း အခ်ိဳ႕ ရွိပါတယ္။

* နာမည္ကို စာလံုး အႀကီး ဒါမွမဟုတ္ အေသးနဲ႔ ေရးႏိုင္ပါတယ္။
* 1 ကေန 9 အထိ ဂဏန္းေတြ ကိုလဲ ထည္႔သြင္းႏိုင္ေပမယ္႔ အစ စာလံုးကိုေတာ႔ အကၡရာနဲ႔ပဲ စရမွာ ျဖစ္ပါတယ္။
* နာမည္ ၾကားမွာ space ျခားလို႔ မရေပမယ္႔ under-score (\_) ကို ေတာ႔ အသံုးျပဳလို႔ ရပါတယ္။
* နာမည္ကို စိတ္ႀကိဳက္ စာလံုးေရ အနည္းအမ်ား ေပးႏိုင္ပါတယ္။
* စာလံုး အႀကီးနဲ႔ အေသးကို compiler က ခြဲျခား သတ္မွတ္ထားတာ ျဖစ္လို႔ var ဟာ VAR နဲ႔ မတူႏိုင္တာကို သတိျပဳရမွာ ျဖစ္ပါတယ္။

ေနာက္ ျခြင္းခ်က္ တစ္ခုက C++ မွာ အသံုးျပဳဖို႔ သတ္မွတ္ထားတဲ႔ keywords ေတြကိုေတာ႔ နာမည္ေပးတဲ႔ အခ်ိန္မွာ ခ်န္လွပ္ထားခဲ႔ရမွာ ျဖစ္ပါတယ္။ (ဥပမာ- **if, else, while, do, case, new, class, int, float, double, char,...**)။

C++ ပရိုဂရမ္မာ အမ်ားစုဟာ variable ေတြကို နာမည္ေပးဖို႔ စာလံုးအေသးေတြပဲ သံုးေလ႔ ရွိပါတယ္။ အခ်ိဳ႕ကေတာ႔ စာလံုးအႀကီးနဲ႔ အေသးကို ေရာေရးေလ႔ ရွိပါတယ္။ (ဥပမာ - IntVar, dataCount)။ const ေတြကို ေရးသားတဲ႔ ေနရာမွာေတာ႔ စာလံုးအႀကီးေတြပဲ သံုးေလ႔ ရွိၾကပါတယ္။ ဘယ္လိုပံုစံမ်ိဳးပဲေရးေရး ပရိုဂရမ္တစ္ပုဒ္မွာေတာ႔ သတ္သတ္မွတ္မွတ္ စတိုင္ တစ္မ်ိဳးနဲ႔ပဲ ေရးသားသင္႔ပါတယ္။ C++ မွာ class ေတြနဲ႔ function ေတြကို နာမည္ေပးတဲ႔ ေနရာမွာလဲ အထက္ပါ ဥပေဒသမ်ားကိုပဲ အသံုးျပဳပါတယ္။

Variable ေတြကို နာမည္ေပးတဲ႔ ေနရာမွာ အဲဒီ variable ရဲ႕ ရည္ရြယ္ခ်က္ကို လူတိုင္း နားလည္လြယ္မယ္႔ နာမည္မ်ိဳးကို ေပးဖို႔ လိုပါတယ္။ ဥပမာ ေျပာရရင္ bT ဒါမွမဟုတ္ t ဆိုတာထက္စာရင္ boilerTemperature လို႔ နာမည္ေပးတာမ်ိဳးက ပိုမိုရွင္းလင္းလြယ္ကူေစပါတယ္။

**Assignment statements**

အထက္ပါ ပရိုဂရမ္ထဲက **var1 = 20; var2 = var1 + 10;** ဆိုတဲ႔ ကုဒ္ ႏွစ္ေၾကာင္းကို ေလ႔လာၾကည္႔ၾကရေအာင္။ ညီမွ်ျခင္း သေကၤတ (**=**) ဟာ တကယ္ေတာ႔ သူ႔ရဲ႕ ညာဘက္က တန္ဖိုးေတြကို ဘယ္ဘက္က variable ေတြထဲကို assign ျပဳလုပ္ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။ ပထမ ကုဒ္စာေၾကာင္းမွာပါတဲ႔ var1 ဟာ ယခင္က တန္ဖိုး မရွိေသးေပမယ္႔ ဒီကုဒ္ေၾကာင္႔ တန္ဖိုး 20 ျဖစ္လာပါတယ္။

**Integer Constants**

အဲဒီ ပထမ စာေၾကာင္းမွာ ပါတဲ႔ 20 သည္ integer constant ပဲ ျဖစ္ပါတယ္။ constants ေတြဟာ ပရိုဂရမ္ တစ္ပုဒ္မွာ တန္ဖိုး မေျပာင္းလဲဘဲ တည္ရွိေနၾကပါတယ္။ integer constant ေတြကေတာ႔ ဒႆမ တန္ဖိုး မပါရွိဘဲ integer ရဲ႕ သတ္မွတ္ တန္ဖိုးေတြ အတြင္းမွာ ရွိတဲ႔ အျပည္႔ကိန္း တန္ဖိုးေတြပဲ ျဖစ္ပါတယ္။ ကုဒ္ ဒုတိယ စာေၾကာင္းမွာပါတဲ႔ (+) သေကၤတကေတာ႔ var1 ကို 10 ေပါင္းထည္႔ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။ အဲဒီ 10 ကလဲ ေနာက္ထပ္ integer constant တစ္ခုပဲ ျဖစ္ပါတယ္။ အဲဒီလို ေပါင္းလို႔ ရလာတဲ႔ တန္ဖိုးကို (=) operator က var2 ထဲကို assign ျပဳလုပ္ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။

**Output Variations**

နမူနာ ပရိုဂရမ္ထဲက cout << "var1+10 is "; ဆိုတဲ႔ စာေၾကာင္းက ကၽြန္ေတာ္တို႔ ယခင္ သင္ခန္းစာေတြမွာ ေလ႔လာခဲ႔သလိုမ်ိဳး string constant တစ္ခု (var1+10 is ) ကို display မွာ ျပသေပးမွာ ျဖစ္ပါတယ္။ ေနာက္ စာသား တစ္ေၾကာင္း ျဖစ္တဲ႔ cout << var2 << endl; ကေတာ႔ variable var2 ထဲမွာ သိမ္းဆည္းထားတဲ႔ တန္ဖိုးကို ျပသေပးမွာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ ဒီပရိုဂရမ္ကို run ၾကည္႔လိုက္မယ္ဆိုရင္ var1+10 is 30 ဆိုတာကို ျမင္ရမွာ ျဖစ္ပါတယ္။ မွတ္သားရမွာ တစ္ခုက cout နဲ႔ << operator ေတြဟာ integer နဲ႔ string ေတြကို ခြဲျခား ကိုင္တြယ္ႏိုင္စြမ္း ရွိတယ္ ဆိုတာပါပဲ။ ဒါေၾကာင္႔ C language မွာ printf() အသံုးျပဳသလို (display လုပ္ရမယ္႔ variable သာမကပဲ) data type ေတြ ခြဲျခားဖို႔ %d %c %s ေတြ ထည္႔ေပးစရာမလိုဘဲ အဆင္ေျပေျပ လုပ္ေဆာင္ႏိုင္စြမ္း ရွိပါတယ္။

ကၽြန္ေတာ္တို႔ အေနဲ႔ cout ႏွစ္ေၾကာင္းက ထြက္လာတဲ႔ အေျဖက တစ္ေၾကာင္းတည္း ျဖစ္ေနတာကို သတိထားမိမွာပါ။ အလိုအေလ်ာက္ တစ္ေၾကာင္း ဆင္းသြားတာမ်ိဳး မရွိပါဘူး။ လိုအပ္ရင္ ကၽြန္ေတာ္တို႔ ကိုယ္တိုင္ ထည္႔သြင္းေပးဖို႔ လိုအပ္မွာ ျဖစ္ပါတယ္။ ေနာက္ပိုင္းမွာ '\n' လိုမ်ိဳး escape sequence ေတြနဲ႔ manipulator ေတြ အသံုးျပဳပံုကို အေသးစိတ္ ရွင္းလင္းသြားမွာ ျဖစ္ပါတယ္။

**The endl Manupulator**

cout ေနာက္ဆံုး စာေၾကာင္းထဲက ေနာက္ဆံုး စကားလံုးဟာ endl ျဖစ္ပါတယ္။ တကယ္ေတာ႔ တစ္ေၾကာင္း ဆင္းခ်င္တဲ႔ ေနရာမွာသံုးရတဲ႔ manipulator တစ္မ်ိဳးပဲ ျဖစ္ပါတယ္။ '\n' နဲ႔ သေဘာသဘာဝခ်င္း တူညီေပမယ္႔ အသံုးျပဳရတာ အဓိပၸါယ္ ပိုမိုရွင္းလင္းေစပါတယ္။ manipulator ေတြကေတာ႔ output stream ကို ျပဳျပင္ပံုေဖာ္ေပးႏိုင္တဲ႔ ညႊန္ၾကားခ်က္ေတြပဲ ျဖစ္ပါတယ္။ endl အေနနဲ႔ '\n' နဲ႔ မတူတဲ႔ အခ်က္ကေတာ႔ output buffer ကို flush ျပဳလုပ္ေပးႏိုင္တာပဲ ျဖစ္ပါတယ္။

**Other Integer Types**

အထက္မွာ တင္ျပခဲ႔သလိုပဲ int အျပင္ အျခား integer data types ေတြ ရွိေနပါေသးတယ္။ အသံုးအမ်ားဆံုးကေတာ႔ long နဲ႔ short ပဲ ျဖစ္ပါတယ္။ (တကယ္ေတာ႔ char ဆိုတာလဲ integer data type လို႔ ေျပာလို႔ ရႏိုင္ပါတယ္။ ေနာက္ပိုင္းမွာ အလ်င္းသင္႔သလို ရွင္းျပေပးသြားပါ႔မယ္။) ကၽြန္ေတာ္တို႔ အထက္မွာ တင္ျပခဲ႔သလိုမ်ိဳး int ဟာ system ေပၚမူတည္ၿပီး အရြယ္အစား ကြဲျပားေပမယ္႔ long နဲ႔ short တို႔ကေတာ႔ system ေပၚ မူတည္မေနဘဲ အရြယ္အစား သတ္မွတ္ခ်က္ ပံုေသ ျဖစ္ပါတယ္။

long အေနနဲ႔ 4 bytes အျမဲေနရာယူေလ႔ရွိၿပီး 32-bit Windows system မွာ ရွိတဲ႔ int နဲ႔ အရြယ္အစားခ်င္း တူပါတယ္။ ဒါေၾကာင္႔ long ဟာ -2,147,483,648 ကေန 2,147,483,647 ၾကား တန္ဖိုးရွိတဲ႔ ဂဏန္းေတြကို သိမ္းဆည္းေပးႏိုင္ပါတယ္။ long ကို ေနာက္တစ္နည္း long int လို႔လဲ ေရးႏိုင္ပါတယ္။ အဓိပၸါယ္ကေတာ႔ အတူတူပဲ ျဖစ္ပါတယ္။ 32-bit system မွာ long ကို သံုးရတာဟာ int နဲ႔ တူညီေနလို႔ သိပ္မထူးျခားေပမယ္႔ 16-bit system ေတြမွာေတာ႔ 4-bytes ေနရာယူတဲ႔ integer type တစ္ခုကို ဖန္တီးဖို႔ အဆင္ေျပေစပါတယ္။ short ကေတာ႔ 2-bytes ေနရာယူၿပီး 16-bit system က int နဲ႔ အရြယ္အစား တူညီပါတယ္။ short ဟာ -32,768 ကေန 32,767 ၾကားမွာရွိတဲ႔ ဂဏန္းေတြကို သိမ္းဆည္းႏိုင္ပါတယ္။ short ကိုေတာ႔ ေခတ္သစ္ Windows system ေတြမွာ သာမန္အားျဖင္႔ အသံုးျပဳေလ႔ မရွိဘဲ memory ကို ေခၽြတာဖို႔ သိပ္လိုမွပဲ သံုးၾကပါေတာ႔တယ္။ int ဟာ short ထက္ ႏွစ္ဆ အရြယ္အစား ပိုႀကီးၿပီး အခ်က္အလက္ေတြကိုလဲ ပိုမိုျမန္ဆန္စြာ ေရးဖတ္ႏိုင္လို႔ အသံုးမ်ားပါတယ္။ ကၽြန္ေတာ္တို႔ အေနနဲ႔ long data type အမ်ိဳးအစား constant တစ္ခုကို ဖန္တီးခ်င္တယ္ ဆိုရင္ ဂဏန္းတန္ဖိုးရဲ႕ ေနာက္မွာ L ကို ကပ္ထည္႔ေပးလိုက္ရမွာ ျဖစ္ပါတယ္။ ဥပမာ -

**longvar = 7678L; //assigns long constant 7678 to longvar**

အသံုးနည္းတဲ႔ integer data types ေတြ ရွိပါေသးတယ္။ compiler ေတြေပၚမူတည္ၿပီး ကြဲျပားႏိုင္ပါတယ္။ အဲဒီ integer ေတြက သူတို႔ ေနရာယူတဲ႔ bits အတိအက်ကို သတ္မွတ္ထားပါတယ္။ (1-byte = 8-bits)။ အဲဒီ type ေတြရဲ႕ အမည္ေရွ႕မွာ underscores ႏွစ္ခု ကို ေရွ႕က ခံထားပါတယ္။ ဥပမာ- **\_\_int8, \_\_int16, \_\_int32, \_\_int64** တို႔ပဲ ျဖစ္ပါတယ္။ \_\_int8 က 32-bit system က char နဲ႔ တူညီၿပီး၊ \_\_int16 က short နဲ႔ \_\_int32 က int နဲ႔ long တို႔နဲ႔ တူညီပါတယ္။ \_\_int64 ကေတာ႔ အႀကီးဆံုး integer value ကို ဂဏန္း ၁၉ လံုးအထိ သိမ္းဆည္းႏိုင္ပါတယ္။ ဒီ type ေတြကို သံုးရာမွာ အရြယ္အစား ပံုေသျဖစ္တဲ႔ အားသာခ်က္ ရွိေပမယ္႔ လက္ေတြ႔မွာေတာ႔ အသံုးနည္းတာကို ေတြ႔ရပါတယ္။

**Character Variables**

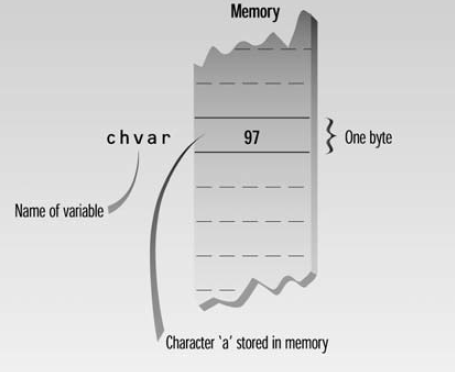
char type ဟာ -128 ကေန 127 ၾကားရွိတဲ႔ integers ေတြကို သိမ္းဆည္းေပးႏိုင္ပါတယ္။ char variable ဟာ memory မွာ 1 byte (eight bits) ပဲ ေနရာယူပါတယ္။ တခါတေလမွာ character variable ေတြကို သူလက္ခံႏိုင္တဲ႔ တန္ဖိုးေတြၾကားမွာ ေဘာင္ဝင္တဲ႔ ဂဏန္းေတြကို သိမ္းဖို႔လဲ သံုးႏိုင္ပါတယ္။ ဒါေပမယ္႔ မ်ားေသာအားျဖင္႔ char variable ေတြကို ASCII characters ေတြ သိမ္းဆည္းဖို႔ပဲ အသံုးခ်ေလ႔ ရွိပါတယ္။

တကယ္ေတာ႔ ASCII character set ဆိုတာဟာ 'a', 'B', '$', '3' အစရွိတဲ႔ characters ေတြကို ဂဏန္းေတြနဲ႔ ေဖာ္ျပတဲ႔ ပံုစံပဲ ျဖစ္ပါတယ္။ အဲဒီ ဂဏန္းေတြဟာ 0 ကေန 127 အထိ ရွိပါတယ္။ Windows system က အဲဒီ range ကို 255 လံုးအထိ တိုးခ်ဲ႕လိုက္ၿပီး ႏိုင္ငံျခား ဘာသာစကားအခ်ိဳ႕နဲ႕ သေကၤတ အခ်ိဳ႕ကို ထည္႔သြင္းလိုက္ပါတယ္။

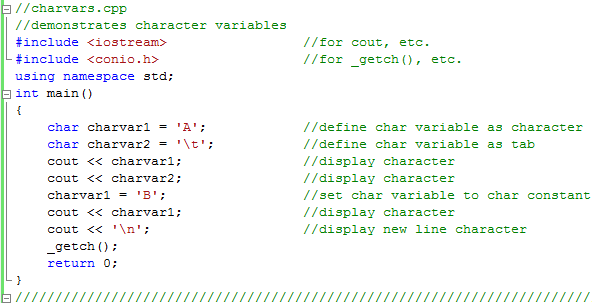
ႏိုင္ငံျခား ဘာသာစကားေတြကို ကိုင္တြယ္ေျဖရွင္းရတဲ႔ အခါမွာ 128 နဲ႔ 255 ၾကား characters ေတြဟာ စံသတ္မွတ္ထားတာ မရွိပါဘူး။ အရြယ္အစားကလဲ 1-byte ပဲရွိလို႔ သိပ္ ေသးငယ္လြန္းတာေၾကာင္႔ ဂ်ပန္ဘာသာလိုမ်ိဳး စကားလံုးအမ်ားႀကီးပါတဲ႔ ဘာသာစကားေတြကို ကိုင္တြယ္လို႔ အဆင္မေျပပါဘူး။ Standard C++ မွာေတာ႔ ႏိုင္ငံျခား ဘာသာစကားေတြကို ကိုင္တြယ္ဖို႔ wchar\_t ဆိုတဲ႔ character type အႀကီးစား တစ္ခု ပါဝင္ပါတယ္။ ဒါေပမယ္႔ ဒီသင္ခန္းစာေတြမွာေတာ႔ ASCII character set ေတြကိုပဲ အသံုးျပဳသြားမွာ ျဖစ္ပါတယ္။

**Character Constants**

Character constants မ်ားသည္ character တစ္လံုးကို single quotation marks (') ႏွစ္ခုၾကားမွာ ထည္႔သြင္းထားတာပဲ ျဖစ္ပါတယ္။ ဥပမာ - 'a', 'b'။ (string constant ေတြကိုေတာ႔ double quotation mark ႏွစ္ခုၾကားမွာ ထည္႔သြင္းေပးရပါတယ္)။ C++ compiler က အဲဒီ character constant ေတြကို သက္ဆိုင္ရာ ASCII code ေတြ အျဖစ္ ေျပာင္းလဲေပးပါတယ္။ ဥပမာ - 'a' ကို 97 အျဖစ္ ဘာသာျပန္ယူမွာ ျဖစ္ပါတယ္။



Character variables ေတြ အေနနဲ႔ character constants ေတြကို value အေနနဲ႔ ထည္႔သြင္း သိမ္းဆည္း ထားႏိုင္ပါတယ္။ နမူနာ ပရိုဂရမ္ေလးကို ေအာက္မွာ ေရးျပထားပါတယ္။



**Initialization**

Variables ေတြကို စတင္ဖန္တီးတဲ႔ အခ်ိန္မွာပဲ တန္ဖိုးေတြ တပါတည္း ထည္႔သြင္း ေပးလိုက္ႏိုင္ပါတယ္။ အဲဒီလို ထည္႔သြင္း ေပးတာကို initialization လုပ္တယ္လို႔ ေခၚပါတယ္။ အထက္က နမူနာ ပရိုဂရမ္ေလးမွာ charvar1 နဲ႔ charvar2 ဆိုတဲ႔ char variable ႏွစ္ခုကို 'A' နဲ႔ '\t' တို႔ အသီးသီး ထည္႔သြင္းၿပီး initialized ျပဳလုပ္ေပးထားတာပါ။

**Escape Sequences**

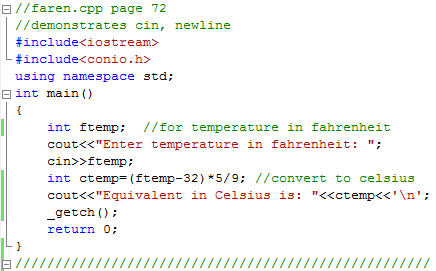
ပရိုဂရမ္ေလးထဲမွာ ပါတဲ႔ '\t' ဆိုတဲ႔ character constant ေလးဟာ အျမင္ဆန္းေနႏိုင္ပါတယ္။ အဲဒီလို character constant ေတြဟာ escape sequence ေတြပဲ ျဖစ္ပါတယ္။ backslash (\) ေၾကာင္႔ ပံုမွန္ characters ေတြ ဘာသာျပန္တဲ႔ စနစ္မ်ိဳးကေန escape ျဖစ္ၿပီး လြတ္ေျမာက္ ေနတာေၾကာင္႔ အဲဒီလို ေခၚဆိုၾကတာပါ။ ဒီ ဥပမာမွာေတာ႔ t ကို character 't' လို႔ ဘာသာျပန္လို႔ မရႏိုင္ဘဲ tab character အျဖစ္ ျပန္ဆိုရပါမယ္။ console-mode program ေတြမွာ tab တစ္ခုဟာ spaces ရွစ္ေနရာ ယူမွာ ျဖစ္ပါတယ္။ ေနာက္ထပ္ character constant တစ္ခုျဖစ္တဲ႔ '\n' ကေတာ႔ ေနာက္တစ္ေၾကာင္းဆင္းဖို႔ ညႊန္ၾကားခ်က္ ေပးတာပါ။ Escape sequences ေတြကို သီးျခားျဖစ္ေစ၊ string constant (စာေၾကာင္း) ေတြထဲမွာ ေရာေႏွာထည္႔သြင္းလို႔ ျဖစ္ေစ အသံုးျပဳႏိုင္ပါတယ္။ ေအာက္ပါ ဇယားေလးကေတာ႔ အသံုးမ်ားတဲ႔ escape sequences ေတြကို ေဖာ္ျပေပးထားပါတယ္။

|  |  |  |
| --- | --- | --- |
| No. | Escape sequences | Character |
| 1. | \a | Bell(beep) |
| 2. | \b | Backspace |
| 3. | \f | Formfeed |
| 4. | \n | Newline |
| 5. | \r | Return |
| 6. | \t | Tab |
| 7. | \\ | Backslash (\) |
| 8. | \' | Single quotation mark (') |
| 9. | \" | Double quotation mark(") |
| 10. | \xdd | Hexadecimal notation |

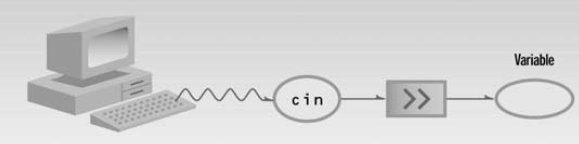
C++ မွာ backslash (\)၊ single quotation marks နဲ႔ double quotation marks ေတြကို သီးျခား ရည္ရြယ္ခ်က္ေတြနဲ႔ အသံုးျပဳထားတာပါ။ ဒါေၾကာင္႔ အဲဒီ character ေတြကို constants အေနနဲ႔ ေဖာ္ျပခ်င္တဲ႔ အခါ အေရွ႕က backslash(\) တစ္ခု ထပ္ထည္႔ၿပီး escape sequence အေနနဲ႔ အသံုးျပဳရတာ ျဖစ္ပါတယ္။ ဥပမာ - cout << "\"Run, Spot, run, \" she said."; ဆိုတဲ႔ ကုဒ္ကို ဘာသာျပန္လိ္ုက္မယ္ဆိုရင္ "Run, Spot, run," she said. ဆိုတဲ႔ စာသားကို ရရွိမွာ ျဖစ္ပါတယ္။ တခါတေလ ASCII code 127 ထက္ ပိုျမင္႔တဲ႔ keyboard ေပၚမွာ မပါဝင္တဲ႔ graphics characters ေတြကို ေဖာ္ျပခ်င္တဲ႔ အခါ '\xdd' ဆိုတဲ႔ ပံုစံမ်ိဳးနဲ႔ ေရးသားေပးရပါတယ္။ အဲဒီပံုစံမွာပါတဲ႔ dd ေနရာမွာ ကိုယ္ေရးခ်င္တဲ႔ character ရဲ႕ သက္ဆိုင္ရာ hexadecimal တန္ဖိုးကို ေရးသားေပးရမွာ ျဖစ္ပါတယ္။ ဥပမာ ကိုယ္က ေလးေထာင္႔တံုးပံုေလးကို ေဖာ္ျပခ်င္တယ္ဆိုရင္ 178 နဲ႔ ညီမွ်တဲ႔ hexadecimal တန္ဖိုး B2 ကို သံုးၿပီး '\xB2' လို႔ ေရးသားရမွာ ျဖစ္ပါတယ္။ charvars.cpp ပရိုဂရမ္ေလးမွာ ပထမဆံုး charvar1 ရဲ႕ တန္ဖိုး ('A') ကို ေရးသားမွာ ျဖစ္ၿပီး charvar2 ရဲ႕ တန္ဖိုး အေနနဲ႔ tab တစ္ခုစာ ေနရာ၈ေနရာ space ျခားၿပီးမွ ေနာက္ထပ္ သတ္မွတ္ေပးလိုက္တဲ႔ charvar1 တန္ဖိုး ('B') ကို ထုတ္ေပးမွာျဖစ္ပါတယ္။ ေနာက္ဆံုးအေနနဲ႔ newline ကို ဆင္းၿပီး ပရိုဂရမ္ ၿပီးဆံုးသြားမွာ ျဖစ္ပါတယ္။

**Input with cin**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ variable ေတြ အသံုးျပဳပံုကို ေလ႔လာခဲ႔ၾကပါတယ္။ အခုအခ်ိန္မွာ ပရိုဂရမ္ ထဲမွာ input ဘယ္လို ေတာင္းယူသလဲ ဆိုတာကို နမူနာ ပရိုဂရမ္ေလး တစ္ပုဒ္ ေရးၾကည္႔ၾကရေအာင္။ ဒီပရိုဂရမ္ေလးမွာ အသံုးျပဳသူကို ဖာရင္ဟိုက္ အပူခ်ိန္တန္ဖိုး ထည္႔သြင္းခိုင္းၿပီး စင္တီဂရိတ္ကို ေျပာင္းေပးမွာ ျဖစ္ပါတယ္။ ေလာေလာဆယ္ integer variables ေတြကို အသံုးျပဳ ေရးသားထားပါတယ္။



ဒီဥပမာေလးမွာ အသံုးျပဳမယ္႔သူကို ဖာရင္ဟိုက္တန္ဖိုး ထည္႔ေပးဖို႔ cout ကို အသံုးျပဳၿပီး အသိေပးပါမယ္။ ေနာက္တစ္ေၾကာင္ cin>>ftemp; ကေတာ႔ ကီးဘုတ္ကေန ထည္႔ေပးမယ္႔ တန္ဖိုးကို ေစာင္႔ၿပီး ထည္႔ေပးလိုက္တာနဲ႔ ftemp ထဲကို ထည္႔သြင္း သိမ္းဆည္းေပးမွာ ျဖစ္ပါတယ္။



cin ဆိုတဲ႔ keyword ကေတာ႔ standard input stream ကို ကိုင္တြယ္မဲ႔ object တစ္ခုပဲ ျဖစ္ပါတယ္။ အဲဒီ stream ဟာ redirected မလုပ္ထားဘူးဆိုရင္ keyboard ကရိုက္ထည္႔လိုက္မယ္႔ အခ်က္အလက္ေတြကို ကိုယ္စားျပဳမွာပါ။ >> operator ကေတာ႔ extraction ဒါမွမဟုတ္ get from operator လို႔ ေခၚပါတယ္။ ၄င္း operator ဟာ ဘယ္ဘက္မွာရွိတဲ႔ stream object (ဒီေနရာမွာ cin) ကို ညာဘက္မွာရွိတဲ႔ variable ထဲကို ထည္႔ေပးမွာ ျဖစ္ပါတယ္။ ပရိုဂရမ္ရဲ႕ result ကို ေအာက္မွာ ေလ႔လာၾကည္႔ႏိုင္ပါတယ္ ခင္ဗ်ာ။



**Cascading <<**

cout မွာ << operator ကို ေအာက္ပါအတိုင္း အႀကိမ္ႀကိမ္ ထပ္ၿပီး သံုးျပထားပါတယ္။

**cout<<"Equivalent in Celsius is: "<<ctemp<<'\n';**

ပထမဆံုး "Equivalent in Celsius is: " ဆိုတဲ႔ စာေၾကာင္းကို cout ကို ေပးပို႔ပါတယ္။ ေနာက္ထပ္ၿပီး ctemp ရဲ႕ တန္ဖိုးကို ေပးပို႔သလို newline character '\n' ကိုလည္း ထည္႔ေပးတာ ျဖစ္ပါတယ္။ အဲဒီ output သံုးခုကို တစ္ေၾကာင္းခ်င္းစီခြဲၿပီး cout သံုးေၾကာင္းနဲ႔ ေရးရင္လည္း အတူတူပါပဲ။ အဲဒီလို နည္းလမ္းမ်ိဳးကို cin နဲ႔ extraction operator >> မွာလဲ သံုးလို႔ ရပါတယ္။ တကယ္လို႔ ထည္႔သြင္းေပးရမယ္႔ တန္ဖိုးက သံုးခု၊ ေလးခု ျဖစ္ေနရင္ တစ္ဆက္တည္း ထည္႔ေပးလို႔ ရႏိုင္ပါတယ္။ ဒါေပမယ္႔ အသံုးျပဳမယ္႔သူကို အမွားနည္းေစဖို႔အတြက္ cout နဲ႔ တန္ဖိုး တစ္ခုခ်င္း ေတာင္းယူၿပီး cin နဲ႔ တစ္ေၾကာင္းခ်င္း ဖတ္ယူတာမ်ိဳးက လက္ေတြ႔မွာ အသံုးမ်ားပါတယ္။

**Expressions**

အတြက္အခ်က္ ျပဳလုပ္ဖို႔အတြက္ variables ေတြ၊ constants ေတြနဲ႔ operators ေတြ စီစဥ္ေရးသားထားတာကို expression လို႔ ေခၚပါတယ္။ a+12 နဲ႔ (a-37)\*b/2 စတာေတြဟာ expression ေတြပါပဲ။ အဲဒီ expression ေတြကို run လိုက္တဲ႔ အခါတိုင္းမွာ တန္ဖိုးတစ္ခု ထြက္လာမွာပါ။

Expression တစ္ခုရဲ႕ အစိတ္အပိုင္းေတြဟာလည္း expression ေတြ ျဖစ္ႏိုင္ပါတယ္။ ဒုတိယ နမူနာ စာေၾကာင္းမွာ a-37 နဲ႔ b/2 ႏွစ္ခု စလံုးဟာ expression ေတြပဲ ျဖစ္ပါတယ္။ a နဲ႔ 37 တို႔လို တစ္လံုးတည္း ျဖစ္ေနတဲ႔ variable နဲ႔ constant ေတြကိုလည္း expression ေတြအျဖစ္ မွတ္ယူႏိုင္ပါတယ္။

Expressions ေတြဟာ statements ေတြနဲ႔ မတူပါဘူး။ statements ေတြက compiler ကို တစ္ခုခုလုပ္ဖို႔ ညႊန္ၾကားၿပီး semicolon နဲ႔ အဆံုးသတ္ေလ႔ ရွိပါတယ္။ expression ေတြကေတာ႔ တြက္ခ်က္မႈတစ္ခုကို ေဖာ္ျပပါတယ္။ statement တစ္ခုမွာ expression ေတြ အမ်ားႀကီး ပါႏိုင္ပါတယ္။

**Precedence**

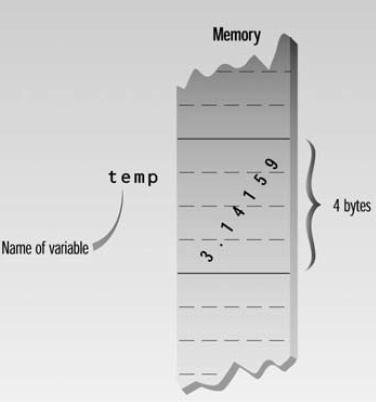
နမူနာ ပရိုဂရမ္ေလးထဲက (ftemp-32)\*5/9 ဆိုတဲ႔ ကုဒ္ေလးကို ေလ႔လာၾကရေအာင္။ အဲဒီမွာပါတဲ႔ လက္သဲကြင္း () ေလးသာ မပါခဲ႔ဘူးဆိုရင္ 32 ကို 5 နဲ႔ အရင္ေျမွာက္မွာ ျဖစ္ပါတယ္။ ဘာလို႔လဲ ဆိုေတာ႔ \* က - ထက္ပိုၿပီး ဦးစားေပးအဆင္႔ျမင္႔လို႔ ျဖစ္ပါတယ္။ လက္သဲကြင္းပါလာတဲ႔ အခါမွာေတာ႔ - ကို အရင္ဆံုး လုပ္ေဆာင္ၿပီး ေနာက္မွ \* ကို ေဆာင္ရြက္ပါတယ္။ တကယ္ေတာ႔ လက္သဲကြင္းထဲက operation ေတြကို အရင္ဆံုး လုပ္ေဆာင္မွာ ျဖစ္ပါတယ္။ **\*** နဲ႔ **/** ကေတာ႔ ဦးစားေပးအဆင္႔ တူညီပါတယ္။ ဒါဆို ဘယ္ဘက္မွာရွိတဲ႔ operator ကို အရင္ လုပ္ေဆာင္မွာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ **\*** ၿပီးမွ **/** လုပ္မွာ ျဖစ္ပါတယ္။ ဒီဦးစားေပး အဆင္႔အတန္း သတ္မွတ္ပံုဟာ သခၤ်ာ ဘာသာရပ္နဲ႔ အျခား ပရိုဂရမ္းမင္းေတြမွာလည္း အတူတူပဲ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ C++ မွာ ဦးစားေပး အဆင္႔ သတ္မွတ္ခ်က္ဟာ အေရးႀကီးတဲ႔ အေၾကာင္းအရာ တစ္ခု ျဖစ္ၿပီး operators ေတြ အေၾကာင္း ရွင္းျပတဲ႔ အခန္းေရာက္ရင္ အက်ယ္ ေဆြးေႏြးေပးသြားမွာပါ။

ယခင္ သင္ခန္းစာမ်ားမွာ int နဲ႔ char ဆိုတဲ႔ type ႏွစ္မ်ိဳးအေၾကာင္းကို ရွင္းျပခဲ႔ပါတယ္။ တကယ္ေတာ႔ အဲဒီ type ႏွစ္ခု စလံုးက ဒႆမ မပါတဲ႔ integer ဂဏန္းေတြနဲ႔ ကိုယ္စားျပဳ ေဖာ္ျပခဲ႔ၾကတာပါ။ ယခု ဆက္လက္ၿပီး floating-point variable မ်ားအေၾကာင္းကို ဆက္လက္ ေဆြးေႏြးသြားမွာ ျဖစ္ပါတယ္။

**Floating Point Types**

Floating-point variable ေတြဟာ ဒႆမ ပါတဲ႔ ဂဏန္းေတြကို ကိုယ္စားျပဳ သိမ္းဆည္းထားႏိုင္ပါတယ္။ ဥပမာ - 3.1415927, 0.0000625, 10.2 စတာေတြပဲ ျဖစ္ပါတယ္။ ၄င္းတို႔ အားလံုးမွာ decimal point (.) ရဲ႕ ဘယ္ဘက္က integer part ေတြ ျဖစ္ၾကၿပီး၊ ညာဘက္ကေတာ႔ fractional part ေတြပဲ ျဖစ္ပါတယ္။ floating-point variable ေတြဟာ သခၤ်ာ ပညာရွင္ေတြအေနနဲ႔ အကြာအေဝး၊ ဧရိယာ၊ အပူခ်ိန္ စတာေတြကို တိုင္းတာဖို႔ သံုးတဲ႔ real numbers ေတြကို ကိုယ္စားျပဳပါတယ္။ အဲဒီ တိုင္းတာမႈေတြမွာ မ်ားေသာအားျဖင္႔ အပိုင္းကိန္း အစိတ္အပိုင္းေတြ ပါေနေလ႔ ရွိပါတယ္။ C++ မွာေတာ႔ float, double နဲ႔ long double ဆိုၿပီး floating-point variable သံုးမ်ိဳး ရွိပါတယ္။

**Type float**



ပံု-၅.၁ ကြန္ပ်ဴတာ မွတ္ဥာဏ္၌ float type variable တစ္ခု ေနရာယူျခင္း

float ကေတာ႔ အဲဒီ floating-point variable သံုးမ်ိဳးထဲမွာ အငယ္ဆံုး ျဖစ္ၿပီး 3.4×10-38 ကေန 3.4×1038 ၾကားထဲမွာ ဂဏန္း ခုႏွစ္လံုး precision ရွိတဲ႔ အခ်က္အလက္ေတြကို သိမ္းဆည္းႏိုင္ပါတယ္။ ေအာက္ပါ ဥပမာေလးကေတာ႔ floating-point number ေတြကိုသံုးၿပီး စက္ဝိုင္း တစ္ခုရဲ႕ ဧရိယာကို ရွာတဲ႔ ပရိုဂရမ္ေလးပါ။

// circarea.cpp

// demonstrates floating point variables

#include <iostream> //for cout, etc.

#include <conio.h> //for \_getch()

using namespace std;

int main()

{

float rad; //variable of type float

const float PI = 3.14159F; //type const float

cout << "Enter radius of circle: "; //prompt

cin >> rad; //get radius

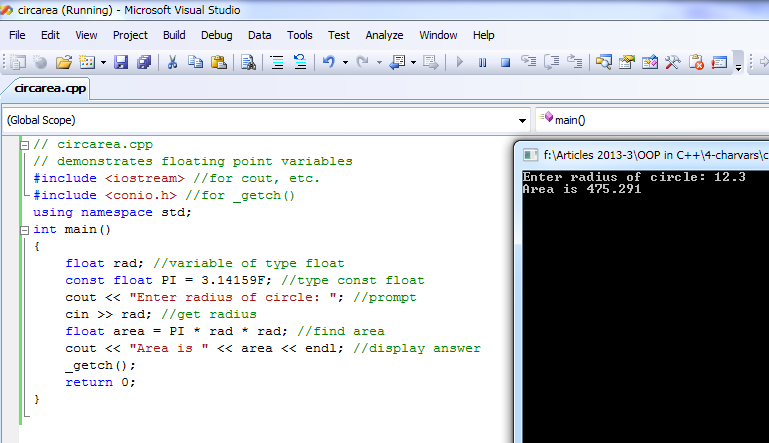
float area = PI \* rad \* rad; //find area

cout << "Area is " << area << endl; //display answer

\_getch();

return 0;

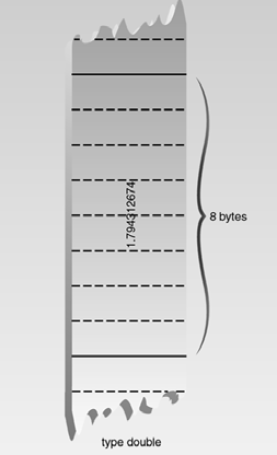
}



ပံု-၅.၂ circarea.cpp ႏွင္႔ testing

**Type double and long double**

ပိုၿပီး အရြယ္အစား ႀကီးမားတဲ႔ floating point types ေတြျဖစ္တဲ႔ double နဲ႔ long double ေတြဟာ ကြန္ပ်ဴတာ မွတ္ဥာဏ္မွာ ေနရာပိုယူၿပီး တန္ဖိုးေတြ ပိုထည္႔သြင္းႏိုင္သလို ပိုမိုတိက်လာတာက လြဲလို႔ float နဲ႔ အတူတူပါပဲ။ double type အေနနဲ႔ သိမ္းဆည္းဖို႔ 8 bytes လိုအပ္ၿပီး 1.7×10-308 ကေန 1.7×10308 ၾကား ကိန္းဂဏန္းေတြကို သိမ္းဆည္းေပးႏိုင္ၿပီး ဒႆမ ၁၅ ေနရာအထိ တိက်ပါတယ္။ long double type ကေတာ႔ ကြန္ပိုင္းလာ အေပၚ မူတည္ေနေပမယ္႔ မ်ားေသာအားျဖင္႔ double နဲ႔ တူေလ႔ ရွိပါတယ္။



ပံု-၅.၃ ကြန္ပ်ဴတာ မွတ္ဥာဏ္၌ double type variable တစ္ခု ေနရာယူျခင္း

**Floating-Point Constants**

circarea.cpp ပရိုဂရမ္ေလးမွာ အသံုးျပဳခဲ႔တဲ႔ 3.14159F ဆိုတဲ႔ ဂဏန္းေလးဟာ floating-point constant ေလးပဲ ျဖစ္ပါတယ္။ decimal point ကို ျမင္လိုက္တာနဲ႔ integer type မဟုတ္မွန္း သိႏိုင္သလို ေနာက္ဆံုးက F ကေတာ႔ float type ျဖစ္ေၾကာင္း သတ္မွတ္ေပးလိုက္တာပါ။ double အတြက္ကေတာ႔ ေနာက္ဆံုးမွာ ဘာ suffix letter မွ ထည္႔ေပးလိုက္စရာ မလိုဘဲ သာမာန္ ဒႆမကိန္း ပံုစံ ေရးေပးဖို႔ပဲ ျဖစ္ပါတယ္။ double က default type ျဖစ္ေနလို႔ပါပဲ။ long double အတြက္ကေတာ႔ L ကို ထည္႔ေပးဖို႔ လိုမွာ ျဖစ္ပါတယ္။

ကၽြန္ေတာ္တို႔ အေနနဲ႔ floating-point constants ေတြကို exponential notation ကိုသံုးၿပီး ေရးသားႏိုင္ပါတယ္။ ဒီနည္းနဲ႔ သုညေတြ အမ်ားႀကီး သံုးစရာ မလိုေတာ႔ပါဘူး။ ဥပမာ 1,000,000,000 ကို 1.0E9 လို႔ ေဖာ္ျပႏိုင္ပါတယ္။ 1234.56 ကိုလည္း 1.23456E3 လို႔ ေရးႏိုင္ပါတယ္။ ဒီေတာ႔ 109 ကို E9 နဲ႔ 103 ကို E3 လို႔ အလြယ္တကူ ေရးႏိုင္ပါတယ္။ E ေနာက္မွာ ေရးတဲ႔ ဂဏန္းကို ထပ္ညႊန္း exponent လို႔ ေခၚၾကပါတယ္။ exponent ကို ၾကည္႔ၿပီး ဒႆမေနာက္ ဘယ္ေနရာ ေရႊ႕ေပးရမယ္ဆိုတာကို သိႏိုင္ပါတယ္။ exponent ေတြက positive နဲ႔ negative တန္ဖိုး ႏွစ္ခုစလံုး ျဖစ္ႏိုင္ပါတယ္။ 6.35239E-5 ဆိုတာဟာ 0.0000635239 ကို ေဖာ္ျပတာပါ။

**The const Qualifier**

circarea.cpp ပရိုဂရမ္ထဲမွာ float အျပင္ const ဆိုတဲ႔ စကားလံုး ပါလာပါတယ္။

**const float PI = 3014159F; //type const float**

Constant အတြက္သံုးတဲ႔ const ဆိုတဲ႔ ေသာ႔ခ်က္ စကားလံုးကို variable ေတြေၾကျငာတဲ႔ data type ေတြရဲ႕ ေရွ႕မွာ ေရးရပါတယ္။ အဲဒီ variable ရဲ႕ တန္ဖိုးဟာ ပရိုဂရမ္ တစ္ခုလံုးမွာ ေျပာင္းလဲလို႔ မရေတာ႔ပါဘူး။ တကယ္လို႔ အဲဒီ တန္ဖိုးကို ပရိုဂရမ္ထဲမွာ ေျပာင္းလဲဖို႔ ႀကိဳးစားခဲ႔မယ္ ဆိုရင္ error message ရရွိမွာ ျဖစ္ပါတယ္။

တကယ္ေတာ႔ const qualifier ေလးဟာ ကၽြန္ေတာ္တို႔ ပံုေသ သတ္မွတ္ထားခ်င္တဲ႔ တန္ဖိုးတစ္ခုကို မေတာ္တဆ ေျပာင္းလဲမိျခင္းမွ ကာကြယ္ဖို႔ သံုးရတာ ျဖစ္ပါတယ္။ ကုဒ္ ကို ဖတ္မိတဲ႔ လူတိုင္းကိုလဲ ဒီတန္ဖိုးဟာ ေျပာင္းလဲလို႔ မရဘူးဆိုတာ သတိေပးထားတာပဲ ျဖစ္ပါတယ္။

**The #define Directive**

C++ မွာ သိပ္မသံုးၾကေတာ႔တဲ႔ ပံုစံ တစ္ခု ျဖစ္ပါတယ္။ constants ေတြကို #define ဆိုတဲ႔ preprocessor directive နဲ႔ သတ္မွတ္ေပးႏိုင္ပါတယ္။ ဒီနည္းနဲ႔ identifier ေတြနဲ႔ ညီမွ် text phrase ေတြကို ဖန္တီးေပးမွာပါ။ ဥပမာ - #define PI 3.14159 ဆိုတဲ႔ စာေၾကာင္းကို ပရိုဂရမ္ရဲ႕ အစမွာ ေရးသားျခင္းျဖင္႔ PI ဆိုတဲ႔ စာသားကို 3.14159 ျဖင္႔ အစားထိုး ေပးမွာ ျဖစ္ပါတယ္။ ဒီလို ေရးသားနည္းကို C language မွာ အေတာ္ေလး ေခတ္စားခဲ႔ဘူးပါတယ္။ ဒါေပမယ္႔ ဘယ္ data type ဆိုတာကို မသတ္မွတ္ႏိုင္တဲ႔ အားနည္းခ်က္ေၾကာင္႔ အမွားအယြင္းေတြ ျဖစ္လာႏိုင္ပါတယ္။ ဒါေၾကာင္႔ C++ နဲ႔ C# မွာ const ကို သံုးၿပီး ေရးသားၾကတာပါ။ ဒါေပမယ္႔ ပရိုဂရမ္ အေဟာင္းေတြကို ဖတ္ၾကည္႔မယ္ ဆိုရင္ေတာ႔ #define ေတြ အမ်ားႀကီး ေတြ႔ရႏိုင္ပါတယ္။

**Type bool**

int data type မွာ ျဖစ္ႏိုင္ေျခရွိတဲ႔ တန္ဖိုးေတြ ဘီလီယံနဲ႔ ခ်ီၿပီး ရွိႏိုင္ပါတယ္။ char မွာေတာ႔ ၂၅၆ လံုး ျဖစ္ႏိုင္ပါတယ္။ bool ကေတာ႔ true နဲ႔ false ဆိုၿပီး ႏွစ္မ်ိဳးပဲ ျဖစ္ႏိုင္ပါတယ္။ သီအိုရီအရ bool ဟာ one bit ပဲ ေနရာယူပါတယ္။ ဒါေပမယ္႔ တကယ္႔လက္ေတြ႔မွာ bit အေနနဲ႔ သိမ္းခဲ႔ရင္ byte ကေန ျပန္ၿပီး ခြဲထုတ္ယူရတာ ျဖစ္လို႔ အခ်ိန္ပိုၾကာတာေၾကာင္႔ ကြန္ပိုင္လာက ကိုင္တြယ္ရယူဖို႔ ျမန္ဆန္တဲ႔ byte အေနနဲ႔ ပဲ သိမ္းဆည္းတာ ျဖစ္ပါတယ္။ ႏိႈင္းယွဥ္ စစ္ေဆးခ်က္ေတြရဲ႕ ရလာဒ္ေတြကို သိမ္းဆည္းဖို႔ bool type ကို အသံုးမ်ားပါတယ္။ ဥပမာ ကိုကို ဟာ ေမာင္ေမာင္ထက္ အသက္ႀကီးလား ဆိုတဲ႔ ႏိႈ္င္းယွဥ္ခ်က္ မွာ မွန္ခဲ႔ရင္ bool value - true ကို ရရွိမွာ ျဖစ္ၿပီး မွားခဲ႔ပါက false ကို ရရွိမွာ ျဖစ္ပါတယ္။ bool ဆိုတဲ႔ နာမည္ကို true-or-false တန္ဖိုးေတြ နဲ႔ logical operators ေတြကို သံုးၿပီး စဥ္းစားေတြးေခၚနည္းကို တီထြင္ခဲ႔တဲ႔ ၁၉ ရာစု အဂၤလိပ္ သခၤ်ာပညာရွင္ George Boole အား အစြဲျပဳၿပီး ဂုဏ္ျပဳေခၚဆိုၾကတာပါ။

**The setw Manipulator**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ endl အေၾကာင္းကို ေရးသားစဥ္က အခ်က္အလက္ေတြကို screen မွာ ျပသတဲ႔အခါ insertion operator (<<) နဲ႔ တြဲသံုးတဲ႔ manipulator ျဖစ္ေၾကာင္း ရွင္းျပခဲ႔ဘူးပါတယ္။ အခု သင္ခန္းစာမွာေတာ႔ output ရဲ႕ field width ကို ေျပာင္းလဲ ေပးႏိုင္တဲ႔ setw manipulator အေၾကာင္းပဲ ျဖစ္ပါတယ္။

ကၽြန္ေတာ္တို႔ cout ကို သံုးၿပီး တန္ဖိုး တစ္ခုကို screen မွာ ျပသမယ္ဆိုရင္ အဲဒီတန္ဖိုးဟာ တိက်တဲ႔ width ရွိတဲ႔ မျမင္ရတဲ႔ ေလးေထာင္႔ ေနရာတစ္ခုကို ယူထားတယ္လို႔ စိတ္ကူး ၾကည္႔ႏိုင္ပါတယ္။ default field ကေတာ႔ အဲဒီ တန္ဖိုး ဆန္႔သေလာက္ ေနရာယူထားတာပါ။ ဥပမာ- integer 567 ဟာ အကၡရာ သံုးလံုးစာ ေနရာယူမွာ ျဖစ္ၿပီး "pajamas" ဆိုတဲ႔ string ကေတာ႔ ခုႏွစ္ေနရာ ယူမွာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ အခ်ိဳ႕ ေနရာေတြမွာေတာ႔ ကၽြန္ေတာ္တို႔ စိတ္ႀကိဳက္ ျပင္ဆင္ သတ္မွတ္ေပးဖို႔ လိုလာပါတယ္။ နမူနာ အေနနဲ႔ width1.cpp ပရိုဂရမ္ေလးကို ေလ႔လာၾကည္႔ရေအာင္။ ဒီပရိုဂရမ္ေလးမွာ ၿမိဳ႕သံုးခုရဲ႕ အမည္ကို ေကာ္လံ တစ္ခုမွာ ျပသထားၿပီး အျခားေကာ္လံမွာေတာ႔ သက္ဆိုင္ရာ လူဦးေရမ်ားကို ေဖာ္ျပေပးမွာ ျဖစ္ပါတယ္။

// width1.cpp

// demonstrates need for setw manipulator

#include <iostream>

#include <conio.h>

using namespace std;

int main()

{

long pop1=2425785, pop2=47, pop3=9761;

cout << "LOCATION " << "POP." << endl

<< "Portcity " << pop1 << endl

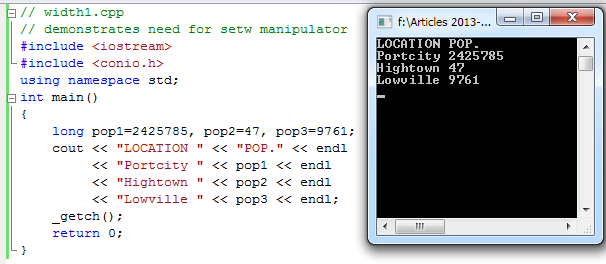
<< "Hightown " << pop2 << endl

<< "Lowville " << pop3 << endl;

\_getch();

return 0;

}



ပံု-၅.၄ width1.cpp ႏွင္႔ testing

ကၽြန္ေတာ္တို႔ သတိထားမိတဲ႔အတိုင္း output မွာပါတဲ႔ လူဦးေရ အေရအတြက္ ေတြဟာ ႏိႈင္းယွဥ္ရ ခက္ခဲေနပါတယ္။ တကယ္လို႔ ညာဘက္ကို ညွိထားမယ္ ဆိုရင္ ဖတ္ရ ပိုမို လြယ္ကူလာမွာ ျဖစ္ပါတယ္။ ၿမိဳ႕နာမည္ေတြ ေဖာ္ျပတဲ႔အခါမွာလည္း ေနာက္က ဂဏန္းေတြၾကားမွာ ေနရာလြတ္ (spaces) ေတြ ျခားေပးႏိုင္မယ္ဆိုရင္ ပိုၿပီး အဆင္ေျပလာမွာပါ။ အထက္က width1.cpp ကိုပဲ အနည္းငယ္ ျပင္ဆင္ၿပီး with2.cpp နာမည္နဲ႔ setw manipulator အသံုးျပဳကာ ျပန္ေရးပါမယ္။

// width2.cpp

// demonstrates setw manipulator

#include <iostream>

#include <iomanip> // for setw

#include <conio.h>

using namespace std;

int main()

{

long pop1=2425785, pop2=47, pop3=9761;

cout << setw(8) << "LOCATION" << setw(12)

<< "POPULATION" << endl

<< setw(8) << "Portcity" << setw(12) << pop1 << endl

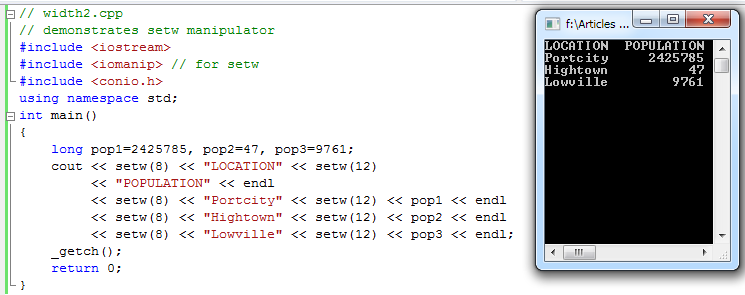
<< setw(8) << "Hightown" << setw(12) << pop2 << endl

<< setw(8) << "Lowville" << setw(12) << pop3 << endl;

\_getch();

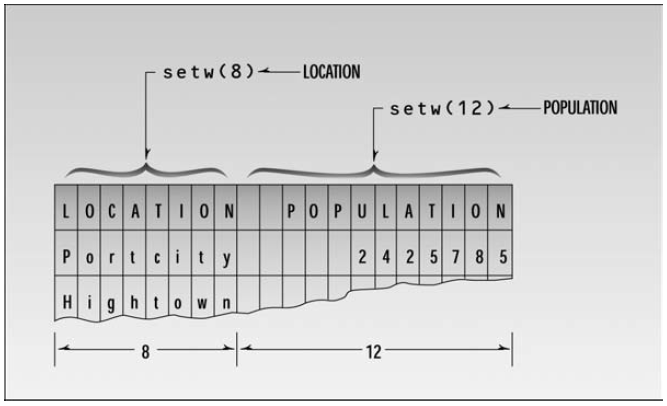
return 0;

}



ပံု-၅.၅ width2.cpp ႏွင္႔ testing

setw manipulator က သူ႔ေနာက္က ဂဏန္းေတြနဲ႔ စာသားေတြကို n character အက်ယ္ရွိတဲ႔ field အတြင္းမွာ ေဖာ္ျပေပးပါတယ္။ အဲဒီေနရာမွာ n ဆိုတာက setw(n) functin ကြင္းထဲက argument (n) ပဲ ျဖစ္ပါတယ္။ အဲဒီ တန္ဖိုးက field ထဲမွာ right-justified ျဖစ္ေနမွာပါ။ အထက္က ပံု-၅.၅ မွာ output မွာ ျမင္ရမယ္႔ ပံုကို ျပထားပါတယ္။ ထူးျခားခ်က္အေနနဲ႔ လူဦးေရေတြကို ေဖာ္ျပတဲ႔ ေနရာမွာ 2-byte ေနရာယူၿပီး အမ်ားဆံုး တန္ဖိုး 32767 ပဲ သိမ္းလို႔ရတဲ႔ အတြက္ overflow ျဖစ္ႏိုင္ေျခရွိတဲ႔ integer type ေတြကို မသံုးဘဲ long ကို သံုးထားတာပါ။



ပံု-၅.၆ width2.cpp ႏွင္႔ testing

**Cascading the Insertion Operator**

width1.cpp နဲ႔ width2.cpp မွာ ပါရွိတဲ႔ cout statement ထဲမွာ cout ကို တစ္ခါပဲ သံုးထားၿပီး စာေၾကာင္းေရ အေတာ္မ်ားမ်ား ေရးထားတာကို သတိထားမိမွာပါ။ ကြန္ပိုင္လာ က whitespace ေတြကို ထည္႔မစဥ္းစားတာနဲ႔ insertion operator (<<) ေတြကို cascade လုပ္ၿပီး ထပ္ခါ ထပ္ခါ ေရးလို႔ရတဲ႔ အခ်က္ေတြကို အသံုးခ်ထားတာ ျဖစ္တယ္။ cout ေလးေၾကာင္းေရးၿပီး ရလာမယ္႔ ရလာဒ္နဲ႔ အတူတူပဲ ျဖစ္ပါတယ္။

**Multiple Deifinitions**

Variables ေတြျဖစ္တဲ႔ pop1, pop2 နဲ႔ pop3 တို႔ကို ဖန္တီးစဥ္ တန္ဖိုးေတြပါ တစ္ပါတည္း သတ္မွတ္ေပးထားတာကို ေတြ႔ရပါမယ္။ ဒီနည္းနဲ႔ data type တူညီတဲ႔ variables ေတြကို ေကာ္မာ ခံၿပီး ဖန္တီးေပးျခင္း၊ တန္ဖိုး သတ္မွတ္ေပးျခင္းျဖင္႔ ကုဒ္မ်ားကို ေလ်ာ႔ခ်ႏိုင္ပါတယ္။

**The IOMANIP Header File**

endl, setw() စတဲ႔ manipulators ေတြကို အသံုးျပဳဖို႔ဆိုရင္ ကၽြန္ေတာ္တို႔ iostream header file ထဲမွာ မပါတဲ႔ အတြက္ iomanip header file ကို ထပ္မံ ေၾကျငာေပးရမွာ ျဖစ္ပါတယ္။ width2.cpp မွာ #include <iomanip> // for setw ဆိုၿပီး ေရးသားသြားတာပါ။

**Data types Conversion**

ယခင္ သင္ခန္းစာမ်ားမွာ data types မ်ား အသံုးျပဳပံုကို အေသးစိတ္ ရွင္းျပခဲ႔ပါတယ္။ ယခု သင္ခန္းစာမွာေတာ႔ အဲဒီ data types မ်ားကို တစ္ခုကေန အျခားတစ္ခုသို႔ ေျပာင္းလဲနည္းမ်ား အေၾကာင္း ေဆြးေႏြး တင္ျပသြားမွာ ျဖစ္ပါတယ္။

C language မွာလိုပဲ C++ မွာ မတူညီတဲ႔ data type ေတြ တြဲဘက္ အသံုးျပဳခြင္႔ ေပးထားပါတယ္။ ဥပမာအေနနဲ႔ ေအာက္ပါ mixed.cpp ပရိုဂရမ္ေလးကို ေလ႔လာၾကည္႔ရအာင္။

// mixed.cpp

// shows mixed expressions

#include <iostream>

#include <conio.h>

using namespace std;

int main()

{

int count = 7;

float avgWeight = 155.5F;

double totalWeight = count \* avgWeight;

cout << "totalWeight = " << totalWeight << endl;

\_getch();

return 0;

}

ဒီ ဥပမာမွာ double data type (totalWeight) ရရွိဖို႔အတြက္ int data type (count) ကို float data type (avgWeight) နဲ႔ ေျမွာက္ထားတာ ေတြ႕ရမွာ ျဖစ္ပါတယ္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ RUN ၾကည္႔တဲ႔ အခါမွာ Error မျပဘဲ အလုပ္လုပ္တာကို ေတြ႕ရပါတယ္။ compiler အေနနဲ႔ အဲဒီလို မတူညီတဲ႔ data types ေတြ အခ်င္းခ်င္း ေပါင္း၊ ႏုတ္၊ ေျမွာက္၊ စား အစရွိတဲ႔ arithmetic operation ျပဳလုပ္တာေတြကို လက္ခံပါတယ္။

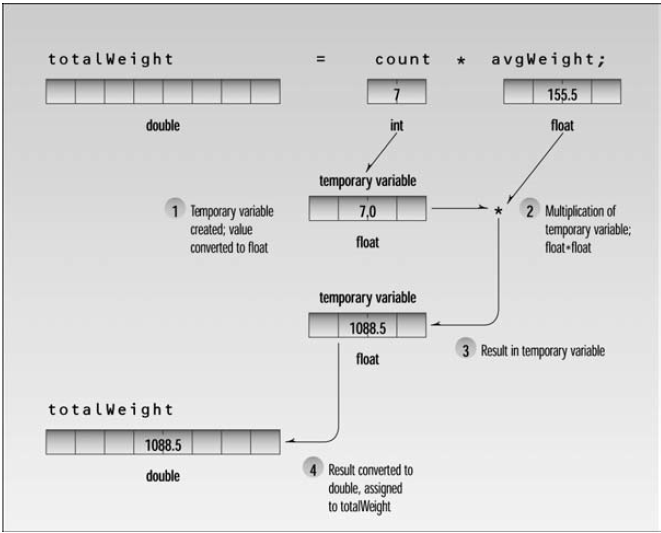
အခ်ိဳ႕ ပရိုဂရမ္းမင္း ဘာသာစကားေတြမွာ ေရးသားရတာ ဒီေလာက္ မလြတ္လပ္ပါဘူး။ အခုလို မတူညီတဲ႔ data type ေတြ ေရာေရးတဲ႔အခါမ်ိဳးမွာ error ျပမွာ ျဖစ္ပါတယ္။ အဲဒီ ဘာသာစကားေတြရဲ႕ compiler ေတြက ကၽြန္ေတာ္တို႔အေနနဲ႔ မွားယြင္းေရးသားမိတယ္ လို႔ ယူဆၿပီး အဲဒီအမွားကေန ကယ္တင္ဖို႔ ႀကိဳးစားၾကတာပဲ ျဖစ္ပါတယ္။ C နဲ႔ C++ မွာေတာ႔ ကၽြန္ေတာ္တုိ႔ အဲဒီလို ေရာေႏွာေရးသားမႈကို ရည္ရြယ္ခ်က္ တစ္ခုေၾကာင္႔လို႔ ယူဆနားလည္ေပးၿပီး ကူညီ လုပ္ေဆာင္ေပးမွာ ျဖစ္ပါတယ္။ အဲဒီ အခ်က္ဟာ C နဲ႔ C++ ေခတ္စားတဲ႔ အေၾကာင္းတစ္ခုလဲပဲ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ပိုၿပီးလြတ္လပ္တဲ႔ ဘာသာစကား ေရးသားနည္း တစ္ခုဟာ ပိုၿပီး အမွား လုပ္မိေစတတ္ ပါတယ္။

**Automatic Conversion**

compiler အေနနဲ႔ အဲဒီလို ေရာေႏွာ ေရးသားထားတဲ႔ expressions ေတြနဲ႔ ရင္ဆိုင္ရတဲ႔ အခါမွာ အလုပ္လုပ္သြားပံုကို ေလ႔လာၾကည္႔ရေအာင္။ ေအာက္မွာ ေဖာ္ျပထားတဲ႔ Table အရ data types ေတြကို higher နဲ႔ lower ဆိုၿပီး အၾကမ္းဖ်င္း ခြဲျခားထားႏိုင္ပါတယ္။

|  |  |
| --- | --- |
| **Data Type** | **Order** |
| long double | Highest |
| double |  |
| float |  |
| long |  |
| int |  |
| short |  |
| char | Lowest |

အေပါင္းနဲ႔ အေျမွာက္ တို႔လို arithmetic operators ေတြဟာ တူညီတဲ႔ data type ရွိတဲ႔ operands ေတြကိုပဲ လုပ္ေဆာင္ေပးခ်င္ပါတယ္။ ဒါေၾကာင္႔ ေရာေႏွာေနတဲ႔ data types ေတြနဲ႔ ရင္ဆိုင္ရတဲ႔ အခါမွာ data type အနိမ္႔ကို အျမင္႔သို႔ အလိုအေလ်ာက္ ေျပာင္းလဲေပးမွာ ျဖစ္ပါတယ္။ mixed.cpp မွာ int (count) float ကိုေျပာင္းလဲေပးၿပီး float (avgWeight) နဲ႔ မေျမွာက္ခင္ tempoerary variavle တစ္ခုမွာ သိမ္းဆည္းထားေပးပါတယ္။ float ႏွစ္ခု ေျမွာက္လို႔ ရလာတဲ႔ ရလာဒ္ကို double ေျပာင္းၿပီး totalWeight ထဲကို ထည္႔ေပးပါတယ္။ အဲဒီ ျဖစ္စဥ္ကို ေအာက္မွာ ပံုနဲ႔ ျပထားပါတယ္။



အဲဒီလို data types ေတြ အလိုအေလ်ာက္ ေျပာင္းလဲေပးသြားတာဟာ မျမင္သာပါဘူး။ ကၽြန္ေတာ္တို႔ အေနနဲ႔ ျဖစ္ခ်င္တာေတြကို C++ compiler က အလိုအေလ်ာက္ လုပ္ေပးသြားတာ ျဖစ္တဲ႔အတြက္ အမ်ားႀကီး စဥ္းစားေနစရာလဲ မလိုအပ္ပါဘူး။ ဒါေပမယ္႔ တစ္ခုတစ္ရံမွာ အဲဒီလို ေျပာင္းလဲတာအခ်ိဳ႕ကို compiler အေနနဲ႔ အဆင္ေျပေျပ မျပဳလုပ္ေပးႏိုင္ပါဘူး။ ေနာက္တစ္ခုက ကၽြန္ေတာ္တို႔အေနနဲ႔ objects ေတြ ျပဳလုပ္ျခင္းျဖင္႔ ကိုယ္ပိုင္ data types ေတြဖန္တီးတဲ႔ အခ်ိန္မွာ မတူညီတဲ႔ object data type ေတြကို ေရာေႏွာ အသံုးျပဳဖို႔အတြက္ compiler က ျပဳလုပ္မေပးႏိုင္ပါဘူး။ အဲဒီအခါမွာေတာ႔ ကိုယ္ပိုင္ conversion routine ေတြကို ေရးသား အသံုးျပဳဖို႔ လိုလာမွာ ျဖစ္ပါတယ္။ compiler ဟာ bulit-in dtat types ေတြကိုပဲ အလိုအေလ်ာက္ ေျပာင္းလဲ ေပးႏိုင္စြမ္း ရွိပါတယ္။

**Casts**

တစ္ခါ တစ္ရံမွာ compiler အေနနဲ႔ အလိုအေလ်ာက္ type conversion ျပဳလုပ္မေပးႏိုင္တဲ႔ အေျခအေနမ်ိဳးမွာ data type တစ္ခုကို ကၽြန္ေတာ္တို႔ သတ္မွတ္ေပးထားတဲ႔ အျခား data type တစ္ခုသို႔ ေျပာင္းလဲေပးျခင္းအား type casts သို႔မဟုတ္ Casts လုပ္တယ္လို႔ ေခၚဆိုႏိုင္ပါတယ္။

C++ မွာ static casts, dynamic casts, reinterpret casts နဲ႔ const casts ဆိုၿပီး casts အမ်ိဳးအစား အမ်ားအျပားရွိပါတယ္။ ဒီသင္ခန္းစာမွာေတာ႔ static casts အေၾကာင္းကိုပဲ ေလ႔လာသြားမွာ ျဖစ္ၿပီး သီးျခား အေျခအေနမ်ားမွာ အသံုးျပဳရတဲ႔ က်န္ရွိေနတဲ႔ casts မ်ားကိုေတာ႔ သက္ဆိုင္ရာ သင္ခန္းစာမ်ားမွာ အလ်င္းသင္႔သလို ထည္႔သြင္း ရွင္းျပေပးသြားမွာ ျဖစ္ပါတယ္။

ေအာက္ပါ နမူနာသည္ int data type အား char data type အျဖစ္ ေျပာင္းလဲေပးေသာ ကုဒ္ ျဖစ္သည္။

**aCharVar = static\_cast<char>(anIntVar);**

ကၽြန္ေတာ္ တို႔ type ေျပာင္းလဲေပးခ်င္တဲ႔ variable (anIntVar) ကို လက္သဲကြင္း () ထဲ ထည္႔သြင္းထားရမွာ ျဖစ္ၿပီး ေျပာင္းလဲမည္႔ data type (char) ကိုေတာ႔ angle brackets <> ထဲမွာ ထည္႔ေပးရမွာ ျဖစ္ပါတယ္။ ဒီနည္းနဲ႔ aCharVar ထဲကို တန္ဖိုး မထည္႔သြင္းခင္မွာ char data type ကို ေျပာင္းလဲ ေပးႏိုင္ပါတယ္။

Standard C++ မတိုင္မီ C ေခတ္တုန္းက ကၽြန္ေတာ္တို႔အေနနဲ႔ အျခား နည္းလမ္းေဟာင္းမ်ားကို cast လုပ္ဖို႔ သံုးခဲ႔ၾကဘူးပါတယ္။ **aCharVar = (char) anIntVar; ဒါမွ မဟုတ္ aCharVar = char(anIntVar);** ဆိုၿပီး ေရးခဲ႔ၾကပါတယ္။ ဒါေပမယ္႔ အားနည္းခ်က္တစ္ခုက ကုဒ္ေတြအမ်ားႀကီးထဲမွာ casts လုပ္ထားတာကို ရွာရခက္ေလာက္ေအာင္ ေပ်ာက္ေနတတ္ပါတယ္။ Find နဲ႔ ရွာဖို႔လည္း မလြယ္ပါဘူး။ အခု static\_cast ကေတာ႔ အဲဒီ ျပႆနာကို ေျဖရွင္းေပးႏိုင္ပါတယ္။ ဒါေၾကာင္႔ ယခင္ နည္းလမ္းေဟာင္းေတြကို သံုးႏိုင္ေသးေပမယ္႔ static\_cast နည္းလမ္းကိုပဲ အသံုးျပဳၾကဖို႔ တိုက္တြန္းလိုပါတယ္။

**Operators**

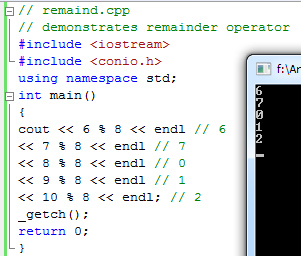
ယခင္ သင္ခန္းစာမ်ားမွာ data types conversion ျပဳလုပ္ပံုမ်ားကို ရွင္းျပခဲ႔ပါတယ္။ ယခု သင္ခန္းစာမွာေတာ႔ Operators မ်ားအေၾကာင္း ေဆြးေႏြး တင္ျပသြားမွာပါ။

**Arithmetic Operators**

C++ မွာ ပံုမွန္ arithmetic operator ၄ မ်ိဳးျဖစ္တဲ႔ **+. - .\*. /** တို႔ကို အသံုးျပဳၿပီး ေပါင္းျခင္း၊ ႏုတ္ျခင္း၊ ေျမွာက္ျခင္း၊ စားျခင္းမ်ားကို ျပဳလုပ္ေပးပါတယ္။ အဲဒီ arithmetic operators ေတြဟာ အျခား ပရိုဂရမ္းမင္း ဘာသာစကား ေတြမွာလိုပဲ algebra သခၤ်ာမွာ သံုးတဲ႔ပံုစံမ်ိဳးကို data types အမ်ိဳးမ်ိဳးမွာ အသံုးခ်ႏိုင္သလို အျခား arithmetic operators မ်ားလည္း ရွိပါေသးတယ္။

**The Remainder Operator**

Remainder Operator (%) ဟာ integer အမ်ိဳးအစား variable (char, short, int, long) ေတြအတြက္ပဲ အသံုးခ်လို႔ ရပါတယ္။ တခါတေလမွာ modulus operator လို႔လည္း ေခၚတတ္ၾကပါတယ္။ ၄င္းဟာ ဂဏန္းတစ္ခုကို အျခား ဂဏန္းတစ္ခုနဲ႔ စားလို႔ ရလာတဲ႔ အၾကြင္း တန္ဖိုးကို ရွာေပးပါတယ္။ ေအာက္မွာ ေဖာ္ျပထားတဲ႔ remaind.cpp ကို ေလ႔လာၾကည္႔ရေအာင္။



အထက္ပါ ပရိုဂရမ္ေလးမွာ ၆ ကေန ၁၀ အထိ ဂဏန္းေတြကို remainder operator ကို သံုးၿပီး ၈ နဲ႔စားလို႔ရလာတဲ႔ အၾကြင္းေတြကို ရွာထားပါတယ္။ အေျဖေတြကေတာ႔ ၆၊ ၇၊ ၀၊ ၁ နဲ႔ ၂ တို႔ ျဖစ္ပါတယ္။ Remainder operator ကို အေျခအေန အမ်ိဳးမ်ိဳးမွာ သံုးႏိုင္ပါတယ္။ ေနာက္ပိုင္း အလ်င္းသင္႔သလို ဥပမာမ်ားနဲ႔ ရွင္းလင္းသြားပါဦးမယ္။ ramaind.cpp မွာပါတဲ႔ **cout<<6%8** ဆိုတဲ႔ ကုဒ္မွာ remainder operator ကို ပထမဦးဆံုး အလုပ္ စလုပ္ပါတယ္။ ဘာျဖစ္လို႔လဲ ဆိုေတာ႔ **%** operator က **<<** operator ထက္ precedence ပိုျမင္႔လို႔ပဲ ျဖစ္ပါတယ္။ တကယ္လို႔ ပိုေသခ်ာ ေစခ်င္တယ္ ဆိုရင္ေတာ႔ လက္သဲကြင္း အဖြင္႔အပိတ္ တစ္စံုၾကားမွာ **(6%8)** ဆိုၿပီး ေရးေပးရမွာ ျဖစ္ပါတယ္။

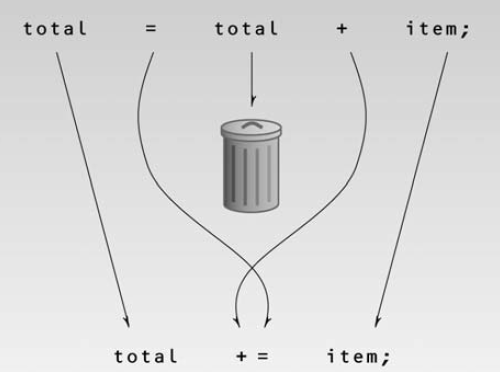
**Arithmetic Assignment Operator**

C++ ကို ေလ႔လာတဲ႔ အခါမွာ ကုဒ္ေတြကို အတိုခ်ံဳ႕ေရးႏိုင္တဲ႔ နည္းလမ္းအမ်ားအျပား ထည္႔သြင္းထားတာကို ေတြ႔ရွိရပါတယ္။ အဲဒီထဲက တစ္ခုကေတာ႔ arithmetic assignment operator ပဲ ျဖစ္ပါတယ္။ ၄င္းဟာ C++ ကို တမူထူးျခားတဲ႔ အသြင္ ျဖစ္ေစပါတယ္။

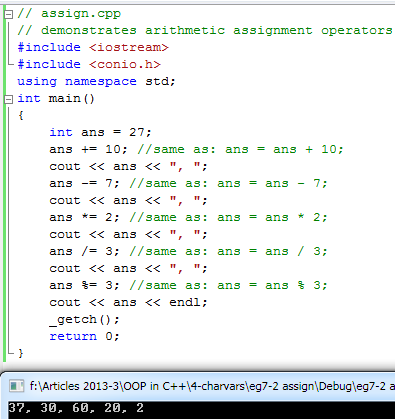
ပံုမွန္အားျဖင္႔ ပရိုဂရမ္းမင္း ဘာသာစကားေတြမွာ total တန္ဖိုးထဲကို item တန္ဖိုးတစ္ခု ထပ္ေပါင္းထည္႔ဖို႔ အတြက္ total = total + item; လို႔ ေရးေလ႔ ရွိၾကပါတယ္။ ဒါေပမယ္႔ အဲဒီ ကုဒ္ထဲမွာ total ကို ႏွစ္ခါ ထပ္ေရးရတဲ႔ အတြက္ ကုဒ္ကို တိုေအာင္ ေရးခ်င္သူမ်ား ဘဝင္မက် ျဖစ္ရပါတယ္။ ဒါေၾကာင္႔ C++ က ပိုမိုက်စ္လစ္တဲ႔ ေရးနည္းကို ပံ႔ပိုးေပးထားပါတယ္။ Arithmetic operator ကို assignment operator နဲ႔ ေပါင္းစပ္ၿပီး ႏွစ္ခါထပ္ေနတဲ႔ operand ကို ျဖဳတ္ထုတ္လိုက္ပါတယ္။ ဒီနည္းနဲ႔ အထက္ပါ ကုဒ္ကို ယခုလို တိုတိုရွင္းရွင္း နဲ႔ ျပင္ေရးလို႔ ရပါတယ္ -

**Total += item;**

ေအာက္မွာျပထားတဲ႔ ပံုေလးကေတာ႔ အဲဒီ ကုဒ္ ႏွစ္မ်ိဳး ထပ္တူညီပံုကို ေဖာ္ျပထားတာပဲ ျဖစ္ပါတယ္။



အထက္ပါ ကုဒ္မွာ arithmetic operator + နဲ႔ assignment operator = ကို ေပါင္းထားတဲ႔ += ကို သံုးျပသြားသလို **-=**, **\*=**, **/=** နဲ႔ **%=** စတဲ႔ operator မ်ားကိုလဲ အသံုးျပဳႏိုင္ပါတယ္။ ေအာက္ပါ assign.cpp ပရိုဂရမ္ေလးမွာ နမူနာ သံုးျပထားပါတယ္။



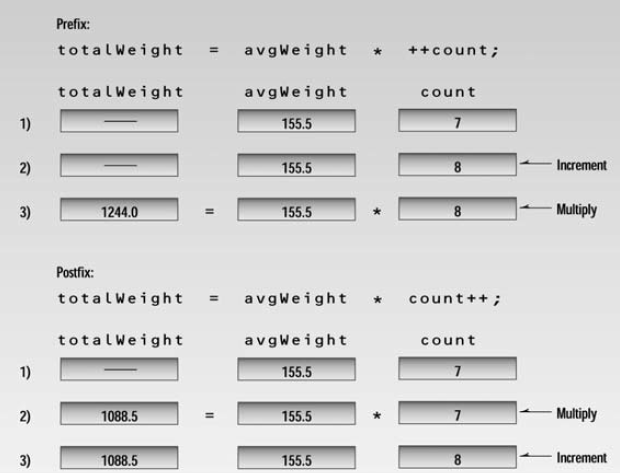
ကၽြန္ေတာ္တို႔အေနနဲ႔ ကုဒ္ေတြ ေရးသားတဲ႔ ေနရာမွာ arithmetic assignment operator ေတြကို မျဖစ္မေန အသံုးျပဳဖို႔ေတာ႔ မလိုပါဘူး။ ဒါေပမယ္႔ အဲဒီ operator ေတြဟာ C++ ရဲ႕ common feature ေတြ ျဖစ္တဲ႔အျပင္ ကုဒ္ေတြကို တိုရင္း လိုရွင္းျဖစ္ေအာင္ ကူညီေပးတဲ႔အတြက္ အလ်င္းသင္႔သလို အသံုးျပဳေစလိုပါတယ္။ ကၽြန္ေတာ္႔ သင္ခန္းစာေတြမွာလည္း ၄င္းတို႔ကို မၾကာခဏ ထည္႔သြင္း အသံုးျပဳသြားမွာ ျဖစ္ပါတယ္။

**Increment Operators**

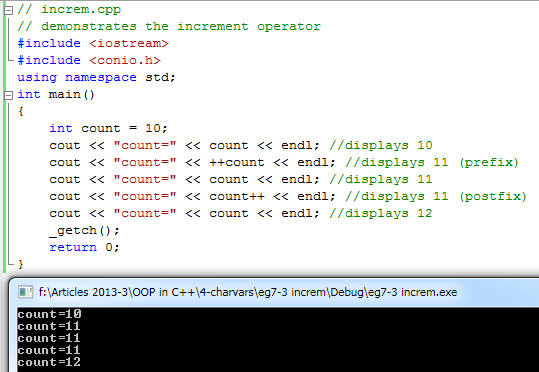
ကၽြန္ေတာ္တို႔ အေနနဲ႔ arithmetic assignment operator ကို မသိခင္တုန္းက (count ဆိုတဲ႔) variable ထဲကို တန္ဖိုး တစ္ခု တိုးခ်င္ရင္ count = count + 1; ဆိုၿပီး ေရးခဲ႔ၾကပါတယ္။ arithmetic assignment operator ကို သံုးတတ္လာေတာ႔ count += 1; ဆိုၿပီး အတိုခ်ံဳ႕ ေရးႏိုင္လာပါတယ္။ ဒါေပမယ္႔ C++ မွာ ဒီထက္ပိုၿပီး တိုေအာင္ ခ်ံဳ႕ေရးလို႔ရတဲ႔ increment operator (++) ဆိုတာ ရွိပါတယ္။ ဒါေၾကာင္႔ အထက္ပါ ကုဒ္ေတြကို ++cont; ဆိုၿပီး အတိုဆံုးေရးလို႔ ရလာပါတယ္။ ရလာဒ္ကေတာ႔ တန္ဖိုးကို ၁ တိုးေပးမွာပဲ ျဖစ္ပါတယ္။

**Prefix and Postfix**

Increment operator ကို အသံုးျပဳတဲ႔ ေနရာမွာ variable ရဲ႕ ေရွ႕ကေန သံုးတဲ႔ prefix ပံုစံနဲ႔ ေနာက္ကေန တြဲဘက္သံုးတဲ႔ postfix ပံုစံဆိုၿပီး ႏွစ္မ်ိဳး သံုးလို႔ ရပါတယ္။ အဲဒီ ႏွစ္ခုရဲ႕ ျခားနားခ်က္ကို ေလ႔လာၾကည္႔ၾကရေအာင္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ ကုဒ္ေတြထဲမွာ အျခား operation ေတြနဲ႔ ေရာေထြးၿပီး variable တစ္ခုကို increment လုပ္ရတဲ႔ အေျခအေနမ်ိဳးကို မၾကာခဏ ၾကံဳရတတ္ပါတယ္။ ဥပမာ- totalWeight = avgWeight \* ++count; အဲဒီလို အေျခအေနမ်ိဳးမွာ ျပႆနာက count ကို increment အရင္လုပ္မလား၊ အရင္ ေျမွာက္မလား ဆိုတာပါပဲ။ ဒီကုဒ္အရေတာ႔ prefix notation ကို သံုးထားတဲ႔ အတြက္ increment ကို အရင္လုပ္ၿပီးမွာ avgWeight နဲ႔ ေျမွာက္မွာ ျဖစ္ပါတယ္။ တကယ္လို႔ postfix notation ကိုသံုးၿပီး totalWeight = avgWeight \* count++; လို႔ ေရးသားခဲ႔မယ္ဆိုရင္ေတာ႔ avgWeight နဲ႔ အရင္ေျမွာက္ၿပီးမွ increment လုပ္မွာ ျဖစ္ပါတယ္။ ဒီျဖစ္စဥ္ကို ေအာက္က ပံုမွာ အေသးစိတ္ ရွင္းျပထားပါတယ္။



ေအာက္မွာ ေဖာ္ျပထားတဲ႔ increm.cpp ပရိုဂရမ္ေလးမွာေတာ႔ prefix နဲ႔ postfix ႏွစ္ခုလံုးကို သံုးျပထားတာ ေတြ႔ရပါလိမ္႔မယ္။



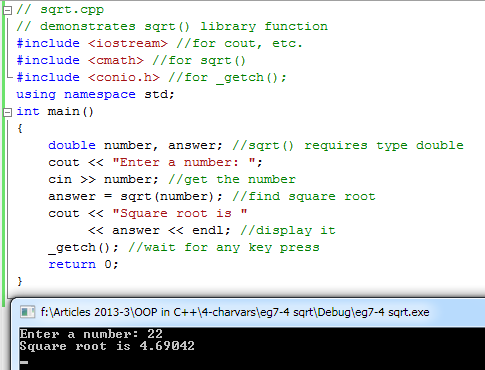
ဒီပရိုဂရမ္ေလးမွာ ပထမဆံုး count ရဲ႕ တန္ဖိုး 10 ကို cout သံုးၿပီး ထုတ္ျပပါတယ္။ ဒုတိယ စာေၾကာင္းမွာ prefix ကိုသံုးၿပီး count ကို increment လုပ္ထားတဲ႔ အတြက္ cout မထုတ္ခင္မွာ count တန္ဖိုး 10 ကေန 11 ျဖစ္သြားၿပီး အေျဖက 11 ျဖစ္လာပါတယ္။ ေနာက္တစ္ေၾကာင္းမွာ postfix ကို သံုးၿပီး increment လုပ္တာ ျဖစ္တဲ႔အတြက္ cout ထုတ္တဲ႔ အခ်ိန္မွာ increment မလုပ္ရေသးပါဘူး။ ဒါေၾကာင္႔ 11 ကိုပဲ ထုတ္ေပးတာပါ။ ဒါေပမယ္႔ cout ထုတ္ၿပီးၿပီးခ်င္း increment လုပ္ေပးမွာ ျဖစ္တဲ႔အတြက္ ေနာက္တစ္ေၾကာင္းနဲ႔ cout အထုတ္မွာ တန္ဖိုးက 12 ျဖစ္ေနတာကို ေတြ႔ရမွာ ျဖစ္ပါတယ္။

**The Decrement (--) Operator**

Decrement operator (--) လုပ္ေဆာင္ပံုေတြက increment operator လိုပဲ ျဖစ္ပါတယ္။ တစ္ခုကြဲျပားတာက decrement operator အေနနဲ႔ variable ရဲ႕ တန္ဖိုးထဲကေန 1 ႏုုတ္ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။ ၄င္းကိုလဲ prefix နဲ႔ postfix ပံုစံ ႏွစ္မ်ိဳးနဲ႔ အသံုးျပဳႏိုင္ပါတယ္။

**Library Functions**

C++ ရဲ႕ လုပ္ေဆာင္မႈ အေတာ္မ်ားမ်ားကို library functions ေတြ ေခၚယူ အသံုးျပဳျခင္းအားျဖင္႔ ေရးသားေလ႔ ရွိပါတယ္။ ၄င္း functions မ်ားကို အသံုးျပဳၿပီး ဖိုင္မ်ားကို ရယူသံုးစြဲျခင္း၊ သခၤ်ာဆိုင္ရာ တြက္ခ်က္မႈမ်ား ျပဳလုပ္ျခင္း၊ data မ်ား conversion ျပဳလုပ္ျခင္းႏွင္႔ အျခားေျမာက္မ်ားလွစြာေသာ လုပ္ေဆာင္မႈမ်ားကို ေဆာင္ရြက္ေပးႏိုင္မွာ ျဖစ္ပါတယ္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ ဒီအခန္းမွာေတာ႔ library function အခ်ိဳ႕ အသံုးျပဳပံုကို sqrt.cpp ဆိုတဲ႔ နမူနာ ပရိုဂရမ္ေလးသံုးၿပီး မိတ္ဆက္ေပးခ်င္ပါတယ္။ ဒီပရိုဂရမ္ေလးမွာ sqrt() ဆိုတဲ႔ library function ေလးကို သံုးၿပီး user ထည္႔သြင္းေပးလိုက္တဲ႔ ဂဏန္းေတြကို square root ရွာေပးမွာ ျဖစ္ပါတယ္။



ပထမဆံုး user ဆီက ေတာင္းလို႔ ရလာတဲ႔ တန္ဖိုးကို sqrt() ဆိုတဲ႔ library function ရဲ႕ ေနာက္က လက္သဲကြင္း အဖြင္႔အပိတ္ထဲကို argument အေနနဲ႔ ထည္႔သြင္းေပးလိုက္တဲ႔အခါမွာ library function က အဲဒီ argument ရဲ႕ square root တန္ဖိုးကို တြက္ထုတ္ၿပီး ျပန္ေပးပါတယ္။ အဲဒီ တန္ဖိုးကို assignment operator (=) သံုးၿပီး answer ဆိုတဲ႔ variable ထဲကို ထည္႔သြင္းေပးတာ ျဖစ္ပါတယ္။ Library function တစ္ခုကို အသံုးျပဳတဲ႔ ေနရာမွာ ထည္႔သြင္းေပးရမယ္႔ argument နဲ႔ ျပန္ရလာမယ္႔ return value ေတြဟာ မွန္ကန္တဲ႔ data type ေတြ ျဖစ္ေနဖို႔ လိုပါတယ္။ ဒီဥပမာမွာ ပါတဲ႔ sqrt() function မွာေတာ႔ argument ေရာ return value ပါ double ျဖစ္တဲ႔အတြက္ variables ႏွစ္ခုစလံုးကို double နဲ႔ အသံုးျပဳခဲ႔တာ ျဖစ္ပါတယ္။

**Header Files**

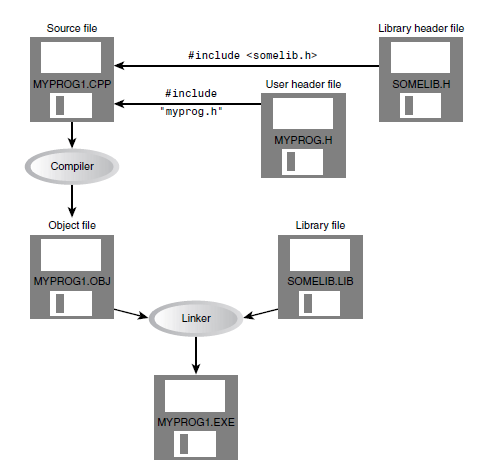
ကၽြန္ေတာ္တို႔ အေနနဲ႔ cout, sqrt() စတဲ႔ objects ေတြ၊ library functions ေတြကို ယူသံုးတဲ႔ အခါတိုင္းမွာ သက္ဆိုင္ရာ header file ေတြကို #include သံုးၿပီး ေၾကျငာေပးဖို႔ လိုမွာ ျဖစ္ပါတယ္။ sqrt() ရဲ႕ header file ဟာ cmath ျဖစ္တာေၾကာင္႔ #include<cmath> ဆိုၿပီး ေရးသားေပးရပါမယ္။ အဲဒီလို ေရးသားျခင္းအားျဖင္႔ preprocessor က cmath ဆိုတဲ႔ header file ကို ကၽြန္ေတာ္တို႔ရဲ႕ source file ျဖစ္တဲ႔ sqrt.cpp ထဲကို ေပါင္းစပ္ ထည္႔သြင္းေပးမွာ ျဖစ္ပါတယ္။ တကယ္လို႔ header file ကို ေၾကျငာေပးဖို႔ ေမ႔ေလ်ာ႔ခဲ႔တယ္ဆိုရင္ေတာ႔ compiler ကေန ‘sqrt’ unidentified identifier ဆိုတဲ႔ error message ေပးမွာ ျဖစ္ပါတယ္။

**Library Files**

Compiler တစ္ခုဟာ executable ဖိုင္ (.exe) တစ္ခု တည္ေဆာက္ဖို႔အတြက္ library functions ေတြနဲ႔ objects ေတြ ပါဝင္တဲ႔ ဖိုင္ေတြကို ခ်ိတ္ဆက္ၿပီး ဖန္တီး ယူရတာ ျဖစ္ပါတယ္။ အဲဒီဖိုင္ေတြမွာ အဲဒီ function ေတြနဲ႔ objects ေတြရဲ႕ machine-executable code ေတြ ပါဝင္ပါတယ္။ Library ဖိုင္ေတြဟာ မ်ားေသာအားျဖင္႔ .lib extension နဲ႔ အဆံုးသတ္ေလ႔ ရွိပါတယ္။ sqrt() function ကို အဲဒီလို ဖိုင္ေတြထဲမွာ ထည္႔သြင္းထားတာ ျဖစ္ပါတယ္။ အဲဒီ function ေတြကို linker က library ဖိုင္ကေနၿပီး ခြဲထုတ္ကာ ပရိုဂရမ္က ေခၚယူ အသံုးျပဳႏိုင္ေအာင္ ခ်ိတ္ဆက္ေပးမွာ ျဖစ္ပါတယ္။ အဲဒီ အေသးစိတ္ အခ်က္အလက္ေတြကို ကၽြန္ေတာ္တို႔ ဘာမွလုပ္ေပးဖို႔ မလိုအပ္ပါဘူး။ Compiler က လိုအပ္သလို ေဆာင္ရြက္ေပးသြားမွာပါ။ ဒါေပမယ္႔ ပရိုဂရမ္မာ တစ္ေယာက္ အေနနဲ႔ကေတာ႔ အဲဒီဖိုင္ေတြအေၾကာင္းကို သိထားဖို႔ လိုအပ္မွာ ျဖစ္ပါတယ္။

**Header Files and Library Files**

Library ဖိုင္နဲ႔ header ဖိုင္ေတြရဲ႕ ဆက္စပ္မႈကို နားလည္ရ ခက္ေနတတ္ပါတယ္။ ဒါေၾကာင္႔ အနည္းငယ္ ထပ္မံ ရွင္းျပလိုပါတယ္။ ကၽြန္ေတာ္တို႔ ေရးသားတဲ႔ ပရိုဂရမ္ထဲမွာ sqrt() ဆိုတဲ႔ library function တစ္ခုကို ထည္႔သံုးခ်င္တယ္ ဆိုပါစို႔။ ဒါဆို အဲဒီ function ပါတဲ႔ library ဖိုင္ကို ကၽြန္ေတာ္တို႔ ပရိုဂရမ္နဲ႔ ခ်ိတ္ဆက္ေပးဖို႔ လိုလာပါတယ္။ အဲဒီလို ျပဳလုပ္ဖို႔ linker က တာဝန္ယူ ေဆာင္ရြက္ေပးပါတယ္။ ဒါေပမယ္႔ ဒီမွာ မၿပီးေသးပါဘူး။ ကၽြန္ေတာ္တို႔ ေရးထားတဲ႔ source ဖိုင္ ထဲမွာ အဲဒီ library ဖိုင္ထဲမွာ ပါတဲ႔ functions ေတြရဲ႕ နာမည္ေတြ၊ အမ်ိဳးအစားေတြနဲ႔ အျခား အခ်က္အလက္ေတြကို သိဖို႔ လိုအပ္ေနပါေသးတယ္။ အဲဒီအခ်က္အလက္ေတြဟာ တကယ္ေတာ႔ header file ထဲမွာ စုစည္းထားတာ ျဖစ္ၿပီး header file တစ္ခုစီမွာ သက္ဆိုင္ရာ functions အခ်ိဳ႕ရဲ႕ အခ်က္အလက္မ်ားကို သိမ္းဆည္းထားပါတယ္။ functions ေတြကိုေတာ႔ စုစည္းၿပီး library file အေနနဲ႔ စုစည္းထားေလ႔ ရွိတယ္။ အဲဒီ library ဖိုင္ တစ္ခုအတြက္ header file အမ်ားအျပား ခြဲျခား ထားတတ္ပါတယ္။ ဥပမာ iostream header file ထဲမွာ cout အစရွိတဲ႔ I/O functions အမ်ားအျပားနဲ႔ objects ေတြရဲ႕ အခ်က္အလက္ေတြကို စုစည္းထားသလို cmath header file ထဲမွာေတာ႔ sqrt() အစရွိတဲ႔ သခၤ်ာ functions ေတြရဲ႕ အခ်က္အလက္ေတြ ပါဝင္ပါတယ္။ တကယ္လို႔ ကၽြန္ေတာ္တို႔အေနနဲ႔ string ႏွစ္ခုကို ႏိႈင္းယွဥ္တဲ႔ strcpy() ဆိုတဲ႔ function ကို အသံုးျပဳခ်င္တယ္ ဆိုရင္ေတာ႔ string.h ကို ေၾကျငာေပးဖို႔ လိုမွာ ျဖစ္ပါတယ္။ C++ မွာ header file ေတြ အျမဲတမ္း အသံုးျပဳေလ႔ ရွိၿပီး ကၽြန္ေတာ္တို႔ အေနနဲ႔ library function တစ္ခု၊ ဒါမွမဟုတ္ ႀကိဳတင္သတ္မွတ္ ေရးသားထားတဲ႔ object ေတြ operator ေတြကို သံုးတဲ႔ အခ်ိန္တိုင္းမွာ သက္ဆိုင္ရာ header file ေတြကို ေၾကျငာေပးရပါမယ္။ ေအာက္က ပံုေလးမွာ header file နဲ႔ library ဖိုင္ေတြ အသံုးျပဳပံုကို ရွင္းျပထားပါတယ္။



**Two Ways to Use #include**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ #include ကို နည္းလမ္း ႏွစ္မ်ိဳးနဲ႔ အသံုးျပဳႏိုင္ပါတယ္။ ပထမ တနည္းက angle brackets <> ၾကားမွာ filename ေတြကို ေရးတဲ႔ နည္းျဖစ္ပါတယ္။ အထက္က နမူနာ ပရိုဂရမ္ေတြထဲကလို #include<iostream> နဲ႔ #include<cmath> တို႔ကို ေရးခဲ႔သလိုမ်ိုးပါ။ ဒီလို ေရးလိုက္တာနဲ႔ compiler က အဲဒီဖိုင္ေတြကို INCLUDE ဆိုတဲ႔ directory ထဲမွာ သြားရွာမွာ ျဖစ္ပါတယ္။ အဲဒီ directory ထဲမွာ compiler ကို ထုတ္လုပ္သူက ထည္႔ေပးလိုက္တဲ႔ header files ေတြ အကုန္လံုးကို စုစည္းထားတာ ျဖစ္ပါတယ္။ သာမန္အားျဖင္႔ ဒီနည္းကိုပဲ သံုးၾကရပါတယ္။ ဒါေပမယ္႔ တကယ္လို႔ ကၽြန္ေတာ္တို႔ရဲ႕ ကိုယ္ပိုင္ header file ေတြ ေရးသား အသံုးျပဳလာၾကမယ္ ဆိုရင္ေတာ႔ #include”myheader.h” ဆိုတဲ႔ ပံုစံမ်ိဳးနဲ႔ quotation marks “” ၾကားမွာ ေရးသားေပးရမွာ ျဖစ္ပါတယ္။ အဲဒီလို ေရးသားျခင္း အားျဖင္႔ compiler က အဲဒီဖိုင္ကို INCLUDE directory မွာ ရွာမယ္႔အစား current directory ထဲမွာ သြားရွာပါလိမ္႔မယ္။ တကယ္ေတာ႔ current directory ဆိုတာ ကၽြန္ေတာ္တို႔ ေရးသားေနတဲ႔ ပရိုဂရမ္ရဲ႕ source ဖိုင္ ရွိတဲ႔ ေနရာျဖစ္ပါတယ္။ အဲဒီေတာ႔ ကၽြန္ေတာ္တို႔ ေရးသားထားတဲ႔ header file နဲ႔ source ဖိုင္ေတြကို ေနရာတစ္ခုထဲမွာ စုစည္းထားဖို႔လိုပါတယ္။ Multifile Programs မ်ားအေၾကာင္း ေရးသားတဲ႔ အခ်ိန္က်မွ အခုထက္ပိုၿပီး အေသးစိတ္ ေဆြးေႏြးေပးပါဦးမယ္။

**Summary (Chapter 1)**

ဒီ Chapter မွာ C++ ရဲ႕ အေျခခံအက်ဆံုး အစိတ္အပိုင္းက function ေတြ ျဖစ္တယ္ဆိုတာကို ေလ႔လာခဲ႔ၿပီးပါၿပီ။ ပရိုဂရမ္ တစ္ပုဒ္မွာ ပထမဆံုး အလုပ္လုပ္တဲ႔ function ကေတာ႔ main() ပဲ ျဖစ္ပါတယ္။ function တစ္ခုကို statements ေတြနဲ႔ ဖြဲ႕စည္းထားတာ ျဖစ္ၿပီး ၄င္းတို႔က ကြန္ပ်ဴတာကို ညႊန္ၾကား ခိုင္းေစေနတာပဲ ျဖစ္ပါတယ္။ Statements တိုင္းဟာ semicolon (;) နဲ႔ အဆံုးသတ္ေလ႔ ရွိပါတယ္။ Statement တစ္ခုမွာ expressions ေတြ တစ္ခုထက္ မက ပိုၿပီး ပါႏိုင္ပါတယ္။ Expression ဆိုတာကေတာ႔ တန္ဖိုးတစ္ခု ထြက္လာေအာင္ variables ေတြနဲ႔ operators ေတြ စုစည္းၿပီး ေရးသားထားတာပဲ ျဖစ္ပါတယ္။

C++ မွာ output value ေတြကို screen မွာ ျပသေပးဖို႔ cout object နဲ႔ insertion operator (<<) ကို အသံုးျပဳေလ႔ ရွိပါတယ္။ Input ေတြကို ဖတ္ယူဖို႔ အတြက္ေတာ႔ cin နဲ႔ extraction operator (>>) ကို အသံုးျပဳၿပီး standard input device (keyboard) ကေန ထည္႔သြင္းေပးလိုက္တဲ႔ တန္ဖိုးေတြကို variable ေတြထဲကို ထည္႔သြင္းေပးပါတယ္။

C++ မွာ built in data types ေတြ အမ်ားအျပားရွိပါတယ္။ char,int,long နဲ႔ short တို႔ဟာ integer data type ေတြျဖစ္ၾကၿပီး float, double နဲ႔ long double တို႔ကေတာ႔ floating-point types ေတြပဲ ျဖစ္ပါတယ္။ အဲဒီ types အားလံုးဟာ အေပါင္းနဲ႔ အႏုတ္ တန္ဖိုး ႏွစ္ခုလံုး ထည္႔သြင္းလို႔ရတဲ႔ (signed) types ေတြပဲ ျဖစ္ပါတယ္။ integer types ေတြကို အႏုတ္တန္ဖိုး ထည္႔သြင္းဖို႔ မလိုအပ္တဲ႔ အေျခအေနမ်ိဳးမွာ unsigned ဆိုတဲ႔ keyword ကို အသံုးျပဳျခင္းျဖင္႔ အရြယ္အစား ႏွစ္ဆ ပိုႀကီးလာေစပါတယ္။ true နဲ႔ false တန္ဖိုး ႏွစ္ခုသာ ပါဝင္တဲ႔ Boolean variables ေတြကို သိမ္းဆည္းဖို႔ bool type အသံုးျပဳႏိုင္ပါတယ္။

const ဆိုတဲ႔ keyword ေလးကေတာ႔ variable တစ္ခုကို constant အသြင္ေျပာင္းပစ္ၿပီး ၄င္းထဲမွာ ထည္႔သြင္းထားတဲ႔ တန္ဖိုးကို ေျပာင္းလဲခြင္႔ မေပးေတာ႔ပါဘူး။

Variable တစ္ခုဟာ မတူညီတဲ႔ data types ေတြ ေရာေႏွာေရးထားတဲ႔ expressions ေတြထဲမွာ အလိုအေလ်ာက္ conversion ျပဳလုပ္ခံရေလ႔ ရွိၿပီး အကယ္၍ programmer က သတ္မွတ္ေပးထားတဲ႔ data type သို႔စိတ္ႀကိဳက္ conversion ျပဳလုပ္ခ်င္ရင္ေတာ႔ casting လုပ္ေပးရမွာပါ။

C++ မွာ သံုးေနက် +, -, \*, / ေတြ အျပင္ အၾကြင္းရွာဖို႔အတြက္ remainder operator (%) ကိုပါ ထည္႔သြင္းေပးထားပါတယ္။

Arithmetic assignment operator ေတြျဖစ္တဲ႔ +=, -= စတာေတြဟာ arithmetic operation နဲ႔ assignment operation ေတြကို တစ္ဆက္တည္း လုပ္ေဆာင္ေပးပါတယ္။ (ဥပမာ - operator ညာဘက္က တန္ဖိုးကို ဘယ္ဘက္က variable နဲ႔ ေပါင္း/ႏုတ္/ေျမွာက္/စား ၿပီး ရလာဒ္ကို အဲဒီ variable ထဲသို႔ ျပန္ထည္႔ေပးလိုက္တာ)။ Increment and decrement operators (++,--) ေတြကေတာ႔ variable ထဲကို ၁ ေပါင္းျခင္း သို႔မဟုတ္ ႏုတ္ျခင္း ျပဳလုပ္ေပးမွာ ျဖစ္ပါတယ္။

Preprocessor directives ေတြကေတာ႔ statement ေတြလို မဟုတ္ပါဘူး။ ၄င္းတို႔က computer ကို ညႊန္ၾကားခ်က္မေပးဘဲ compiler ကို ညႊန္ၾကားခ်က္ေပးတာပဲ ျဖစ္ပါတယ္။ #include ဆိုတဲ႔ directive ကေတာ႔ လက္ရွိ source ဖိုင္ ထဲကို အျခား ဖိုင္တစ္ခု ထည္႔သြင္းေပးဖို႔ compiler ကို ညႊန္ၾကားတာပဲ ျဖစ္ပါတယ္။ #define ဆိုတဲ႔ directive ကေတာ႔ ညႊန္ၾကားခ်က္ တစ္ခုကို ေနာက္ညႊန္ၾကားခ်က္ တစ္ခုနဲ႔ အစားထိုးဖို႔ ေစခိုင္းတာပါ။ using ဆိုတဲ႔ directive ကေတာ႔ compiler ကို namespace တစ္ခုရဲ႕ နာမည္ေတြ မွတ္သားထားေစတာပဲ ျဖစ္ပါတယ္။

ကၽြန္ေတာ္တို႔ ပရိုဂရမ္ေတြထဲမွာ library function ေတြကို ေခၚယူအသံုးျပဳခဲ႔မယ္ ဆိုရင္ အဲဒီ function ရဲ႕ တကယ္႔ machine-code ေတြက သက္ဆိုင္ရာ library file ေတြထဲမွာ ရွိေနၿပီး အလိုအေလ်ာက္ ခ်ိတ္ဆက္ အသံုးျပဳမွာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ အဲဒီ function ရဲ႕ declaration ေတြပါဝင္တဲ႔ header file ကိုေတာ႔ ကၽြန္ေတာ္တို႔ source file ထဲမွာ #include ကို အသံုးျပဳၿပီး ထည္႔သြင္းေပးရမွာ ျဖစ္ပါတယ္ခင္ဗ်ာ။

**အခန္း(၂)**

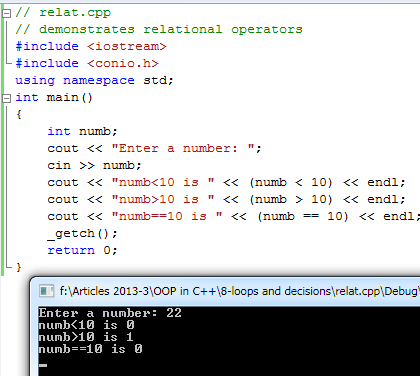
Loops and Decisions

**Relational Operators**

ပရိုဂရမ္ အမ်ားစုသည္ အစမွ အဆံုးအထိ အစဥ္လိုက္ အလုပ္လုပ္သည္႔ ပံုစံမ်ိဳး မရွိၾကပါ။ အသံုးျပဳသူ ဆံုးျဖတ္သည္႔အတိုင္း အေျခအေနေျပာင္းလဲမႈကို လိုက္ေလ်ာညီေထြျဖစ္ေအာင္ ျပဳမူသည္႔ ပံုစံမ်ိဳး ေရးသားေလ႔ ရွိၾကသည္။ ပရိုဂရမ္အတြင္းရွိ တြက္ခ်က္မႈမ်ားေပၚ တူတည္၍ Control flow ကို ပရိုဂရမ္ ၏ တစ္ေနရာမွ အျခားတစ္ေနရာသို႔ jump လုပ္ကာ ေဆာင္ရြက္ေလ႔ ရွိသည္။ ထိုသို႔ jump ျဖစ္ေစႏိုင္ေသာ statements မ်ားကို control statements မ်ားဟု ေခၚသည္။ ၄င္းတို႔ကို loops ႏွင္႔ decisions ဟူ၍ ေခါင္းစဥ္ႏွစ္ခု ခြဲျခား ေလ႔လာႏိုင္ပါသည္။

loop တစ္ခုကို အႀကိမ္မည္မွ် ပတ္မည္ သို႔မဟုတ္ decision တစ္ခုကို မည္သို႔လုပ္ေဆာင္မည္ ဆိုသည္႔ အခ်က္သည္ သက္ဆိုင္ရာ expression မ်ား မွန္သည္ မွားသည္ ဆိုသည္႔ အေပၚ မူတည္ေနေပသည္။ အဆိုပါ expression မ်ား အတြင္း၌ ရွိေသာ တန္ဖိုးႏွစ္ခုကို ႏိႈင္းယွဥ္ရန္ အသံုးျပဳသည္႔ operator မ်ားကို relational operator ဟု ေခၚဆိုၾကသည္။ Loops ႏွင္႔ decisions မ်ားကို ေလ႔လာရာတြင္ ၄င္း relational operator မ်ားအား အသံုးခ်ရသျဖင္႔ ဦးစြာ ရွင္းလင္း တင္ျပေပးပါမည္။

Relational Operators မ်ားသည္ C++ ၏ မည္သည္႔ built-in data type တန္ဖိုး ႏွစ္ခုကိုမဆို ႏိႈင္းယွဥ္ေပးႏိုင္ပါသည္။ ထို႔ျပင္ ေနာက္ပိုင္းေလ႔လာၾကရမည္႔ user-defined classes မ်ားကိုလည္း ႏိႈင္းယွဥ္ေပးႏိုင္စြမ္း ရွိပါသည္။ ထိုသို႔ ႏိႈင္းယွဥ္ရာတြင္ (==) တူညီသည္၊ (>) ႀကီးသည္၊ (<) ငယ္သည္ဟူေသာ ႏိႈင္းယွဥ္ခ်က္မ်ား ျပဳလုပ္ေလ႔ ရွိပါသည္။ ႏိႈင္းယွဥ္ခ်က္ ေဖာ္ျပခ်က္၏ ရလာဒ္မွာ မွန္လွ်င္ true ျဖစ္၍ မွားလ်င္ false ျဖစ္သည္။ ပိုမိုရွင္းလင္း လြယ္ကူစြာ နားလည္ သေဘာေပါက္ေစရန္ relat.cpp ပရိုဂရမ္ အား ေလ႔လာၾကည္႔ၾကပါစို႔။



အထက္ပါ ပရိုဂရမ္တြင္ user ထည္႔သြင္းေပးလိုက္ေသာ တန္ဖိုးႏွင္႔ ၁၀ အား ႏိႈင္းယွဥ္မႈ ၃ ခု ျပဳလုပ္ထားပါသည္။ user မွ 22 အား input အျဖစ္ ထည္႔သြင္းေပးလိုက္ခ်ိန္တြင္ ႏိႈင္းယွဥ္မႈ ၃ ခု၏ ရလာဒ္မ်ားကို အထက္ပါအတိုင္း မွန္လွ်င္ 1 မွားလွ်င္ 0 ျဖင္႔ ေဖာ္ျပထားသည္ကို ေတြ႔ရပါသည္။ ပထမ ႏိႈင္းယွဥ္ခ်က္မွာ numb (22) သည္ 10 ထက္ ငယ္သလား ဟူေသာ ႏိႈင္းယွဥ္ခ်က္ျဖစ္ၿပီး မွားယြင္းသျဖင္႔ ရလာဒ္မွာ 0 ျဖစ္ပါသည္။ ဒုတိယ ႏိႈင္းယွဥ္ခ်က္မွာ numb (22) သည္ 10 ထက္ ႀကီးသလား ဟူေသာ ႏိႈင္းယွဥ္ခ်က္ျဖစ္၍ မွန္ကန္ေသာေၾကာင္႔ ရလာဒ္မွာ 1 ျဖစ္သည္။ ေနာက္ဆံုး ႏိႈ္င္းယွဥ္ခ်က္မွာ numb (22) သည္ 10 ႏွင္႔ ညီသလား ဟူေသာ ႏိႈင္းယွဥ္ခ်က္ျဖစ္ၿပီး မွားယြင္းေသာေၾကာင္႔ ရလာဒ္မွာ 0 ျဖစ္ပါသည္။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ ရလာဒ္မ်ားကို ture, false မ်ားျဖင္႔ ေမွ်ာ္လင္႔ထားေသာ္လည္း တကယ္တမ္းတြင္ compiler က 1,0 ျဖင္႔သာ အေျဖ ထုတ္ေပးပါသည္။ သို႔ျဖစ္၍ အဆိုပါ ျပႆနာတြင္ compiler အတြက္ 1 သည္ true ျဖစ္၍ 0 သည္ false အျဖစ္ သတ္မွတ္ထားေၾကာင္း ထင္ရွားပါသည္။ အကယ္၍ ကၽြန္ေတာ္တို႔အေနျဖင္႔ bool data type ကို အသံုးျပဳခဲ႔လွ်င္ပင္ အေျဖအား အထက္ပါအတိုင္းသာ ရရွိမည္ ျဖစ္သည္။ အဘယ္ေၾကာင္႔ဆိုေသာ္ C++ စတင္ေပၚေပါက္ကာစက bool data type မရွိေသးေသာေၾကာင္႔ ျဖစ္ပါသည္။ ထို႔ေၾကာင္႔ ယခုအခါ C++ တြင္ အမွားမွန္ကို 0,1 သာမက true,false ႏွင္႔ပါ ေဖာ္ျပႏိုင္ေပသည္။

လက္ေတြ႔တြင္မူ အဆိုပါ ကြဲျပားခ်က္သည္ သိပ္အေရးမပါလွပါ။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ ture, false ျဖင္႔ ေဖာ္ျပျခင္း၊ မေဖာ္ျပျခင္းထက္ looping တစ္ခု သို႔မဟုတ္ decision တစ္ခု ဘာဆက္လုပ္ရမည္ ဆိုသည္ကို ဆံုးျဖတ္ရန္သာ လိုအပ္ေပသည္။

ေအာက္ပါ ဇယားတြင္ C++ ၏ relational operator မ်ားအားလံုးကို ေဖာ္ျပေပးထားပါသည္။

|  |  |
| --- | --- |
| Operator | Meaning |
| > | Greater than |
| < | Less than |
| == | Equal to |
| != | Not equal to |
| >= | Greater than or equal to |
| <= | Less than or equal to |

အထက္ပါ Relational operators မ်ားအား အသံုးျပဳပံုကို ေအာက္တြင္ နမူနာ expression မ်ားျဖင္႔ ေဖာ္ျပထားပါသည္။ ပထမ ႏွစ္ေၾကာင္းမွာ harry ႏွင္႔ jane ဆိုေသာ variable မ်ားထဲသို႔ တန္ဖိုးမ်ား သတ္မွတ္ ထည္႔သြင္းျခင္းျဖစ္ၿပီး က်န္စာေၾကာင္းမ်ားက ၄င္းတန္ဖိုးမ်ားကို relational operators မ်ားအသံုးျပဳ၍ အမ်ိဳးမ်ိဳး ႏိႈင္းယွဥ္ ျပထားျခင္းပင္ ျဖစ္သည္။

jane = 44; //assignment statement

harry = 12; //assignment statement

(jane == harry) //false

(harry <= 12) //true

(jane > harry) //true

(jane >= 44) //true

(harry != 12) // false

(7 < harry) //true

(0) //false (by definition)

(44) //true (since it’s not 0)

မွားေလ႔ရွိေသာ အခ်က္တစ္ခုမွာ equal operator (==) ကို equal signs ႏွစ္ခု သံုးရမည္႔အစား တစ္ခုတည္း သံုးမိျခင္းျဖစ္သည္။ ထိုအမွားမ်ိဳးသည္ compiler က မွားမွန္း မသိျခင္းေၾကာင္႔ ရွာေဖြရန္ ခက္ခဲေလ႔ ရွိသည္။

C++ အေနျဖင္႔ true ကို 1 ျဖင္႔ ကိုယ္စားျပဳေလ႔ ရွိေသာ္လည္း သုညမဟုတ္သည္႔ မည္သည္႔ ဂဏန္း (အႏုတ္တန္ဖိုးအပါအဝင္) ကိုမဆို true အျဖစ္ လက္ခံေလ႔ ရွိသည္။ false အတြက္မူ သုည 0 တစ္ခုတည္းသာ သတ္မွတ္ထားသည္။

**Loop**

Loops ဆိုသည္မွာ ပရိုဂရမ္၏ အစိတ္အပိုင္းအခ်ိဳ႕ကို အႀကိမ္အေရအတြက္ တစ္ခုအထိ အႀကိမ္ႀကိမ္ လုပ္ေဆာင္ျခင္းပင္ ျဖစ္သည္။ ထိုသို႔ ထပ္ခါတလဲလဲ လုပ္ေဆာင္ျခင္းသည္ condition အေျခအေန မွန္ကန္ေနသမွ်ပင္ ျဖစ္သည္။ အဆိုပါ condition မွားယြင္းသြားသည္ႏွင္႔တၿပိဳင္နက္ Loop ၿပီးဆံုးသြားၿပီး Loop ေနာက္က statements မ်ားကို ဆက္လက္ လုပ္ေဆာင္သြားမည္ ျဖစ္သည္။ C++ တြင္ for loop, while loop ႏွင္႔ do loop ဟူ၍ loops သံုးမ်ိဳး ရွိသည္။

**The for Loop**

**for loop** သည္ လူအမ်ားအတြက္ ေလ႔လာရာတြင္ အလြယ္ဆံုး loop ျဖစ္ေလ႔ ရွိသည္။ for loop တြင္ control elements မ်ားသည္ တစ္ေနရာထဲ၌ စုစည္းထားေလ႔ရွိၿပီး အျခား loop မ်ားတြင္မူ control မ်ားသည္ ျပန္႔က်ဲတည္ရွိသျဖင္႔ သေဘာေပါက္ရန္ ခက္ခဲတတ္ျခင္း ျဖစ္သည္။ ၄င္းသည္ ကုဒ္အစိတ္အပိုင္းမ်ားကို တိက်ေသာ အႀကိမ္အေရအတြက္အတြင္း လုပ္ေဆာင္ေလ႔ ရွိသည္။ မ်ားေသာအားျဖင္႔ for loop ကို အႀကိမ္မည္မွ် လုပ္ေဆာင္မည္ဆိုသည္႔ အခ်က္အား ႀကိဳတင္ တြက္ဆထားႏိုင္သည္႔ အေျခအေနမ်ိဳးတြင္ အသံုးျပဳေလ႔ ရွိသည္။ ေအာက္ပါ fordemo.cpp ပရိုဂရမ္တြင္ for loop ကို အသံုးျပဳ၍ 0 မွ 14 အတြင္းရွိ ဂဏန္းမ်ား၏ ႏွစ္ထပ္ကိန္းကို ရွာျပထားသည္။

// fordemo.cpp

// demonstrates simple FOR loop

#include <iostream>

using namespace std;

int main()

{

int j; //define a loop variable

for(j=0; j<15; j++) //loop from 0 to 14,

cout << j \* j << “ “; //displaying the square of j

cout << endl;

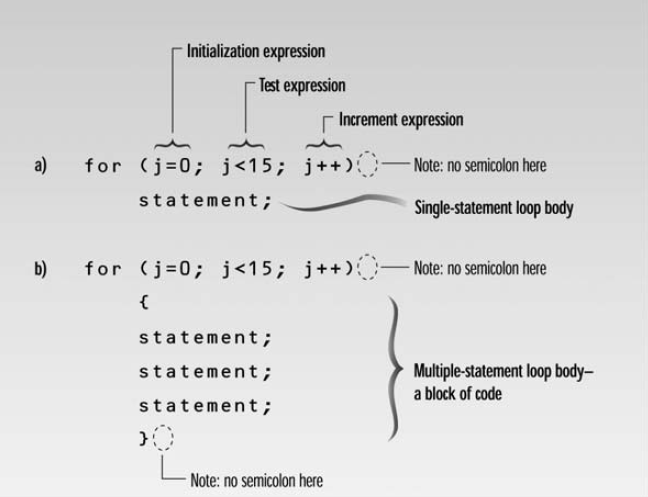
return 0;

}

ပရိုဂရမ္ကို RUN ၾကည္႔လွ်င္ ေအာက္ပါ အေျဖမ်ားကို ရရွိမည္ ျဖစ္သည္။

**0 1 4 9 16 25 36 49 64 81 100 121 144 169 196**

ဒီ အေျဖေတြ ဘယ္လိုရလာသလဲ ေလ႔လာၾကည္႔ၾကရေအာင္။ for loop မွာ for ဆိုတဲ႔ keyword ေနာက္က လက္သဲ ကြင္းစ၊ ကြင္းပိတ္ထဲမွာ expression သံုးခုကို semicolons ေတြနဲ႔ ျခားထားပါတယ္ - for(j=0; j<15; j++)။ အဲဒီ expression သံုးခုကေတာ႔ initialization expression, test expression နဲ႔ increment expression တို႔ပဲ ျဖစ္ပါတယ္။



အဆိုပါ expression မ်ားတြင္ (ျခြင္းခ်က္အခ်ိဳ႕မွအပ) loop variable သို႔မဟုတ္ looping counter ဟုေခၚေသာ variable တစ္ခုတည္းကိုသာ အသံုးျပဳေလ႔ ရွိသည္။ fordemo.cpp တြင္ အသံုးျပဳခဲ႔ေသာ loop variable မွာ j ျဖစ္သည္။ ၄င္းကို for loop မစမီ defined ျပဳလုပ္ေပးထားရန္ လိုအပ္သည္။

Loop ၏ အတြင္းရွိ statements မ်ားသည္ loop ပတ္ေနသမွ် လုပ္ေဆာင္ေနရမည္႔ ကုဒ္မ်ားပင္ ျဖစ္သည္။ fordemo.cpp တြင္မူ cout << j \* j << “ “; ကုဒ္တစ္ေၾကာင္းတည္းသာ အသံုးျပဳထားသည္။

အထက္ပါ statement သည္ j တန္ဖိုး၏ ႏွစ္ထပ္ကိန္းႏွင္႔ ေနာက္တြင္ ေနရာလြတ္ (spaces) ႏွစ္ေနရာ ကို print လုပ္ေပးပါမည္။ ႏွစ္ထပ္ကိန္းရရွိရန္ j ကို j ျဖင္႔ပင္ ေျမွာက္ေပးလိုက္မည္ ျဖစ္သည္။ loop ကို ပတ္ေနသမွ် j တန္ဖိုးသည္ 0,1,2,3, မွ 14 အထိ ေျပာင္းလဲလာမည္ ျဖစ္သည္။ ထို႔ေၾကာင္႔ ႏွစ္ထပ္ကိန္းမ်ားျဖစ္ေသာ 0, 1, 4, 9, မွ 196 အထိ ကို ရရွိမည္ ျဖစ္သည္။ မွတ္သားရမည္႔ အခ်က္တစ္ခုမွာ for statement ၏ အဆံုးတြင္ semicolon မထည္႔ရျခင္းပင္ ျဖစ္သည္။ အဘယ္ေၾကာင္႔ဆိုေသာ္ for statement ႏွင္႔ loop body အားလံုးကို ေပါင္း၍ statement တစ္ေၾကာင္းအျဖစ္ သတ္မွတ္ထားေသာေၾကာင္႔ ျဖစ္သည္။ အကယ္၍ for statement အဆံုးတြင္ semicolon ထည္႔ေပးလိုက္ပါက compiler က loop body မရွိဟု ယူဆသြားမည္ ျဖစ္ၿပီး ေမွ်ာ္လင္႔ မထားသည္႔ ရလာဒ္မ်ား ထြက္ေပၚလာမည္ ျဖစ္သည္။ ၄င္း expressions သံုးခုက loop ကို ဘယ္လို ထိန္းခ်ဳပ္သြားသည္ကို ေလ႔လာၾကရေအာင္။

**The Initialization Expression**

Initialization expression ကို loop စစခ်င္း တစ္ႀကိမ္သာ လုပ္ေဆာင္ပါသည္။ ၄င္းက loop variable ကို initial value သတ္မွတ္ေပးသည္။ fordemo.cpp တြင္မူ ၄င္းက j ၏ အစ တန္ဖိုးကို 0 သတ္မွတ္ေပးပါသည္။

**The Test Expression**

Test expression တြင္ relational operator ပါဝင္ေလ႔ ရွိသည္။ Loop ပတ္သည္႔ အႀကိမ္တိုင္း မပတ္ခင္ ၄င္းကို မွန္မမွန္ စစ္ေဆးသည္။ မွန္လွ်င္ loop ပတ္၍ မွားခဲ႔လွ်င္ loop မွ ထြက္သြားမည္ ျဖစ္သည္။

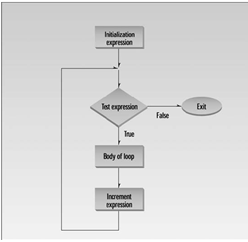
**The Increment Expression**

Increment expression ကေတာ႔ loop variable ကို loop တစ္ပတ္ ပတ္ၿပီးတိုင္း ေျပာင္းလဲေပးပါတယ္။ မ်ားေသာအားျဖင္႔ increment (တိုးေပး) လုပ္ေပးတာပါ။ ဥပမာ j++ ဆိုရင္ loop တစ္ပတ္ ပတ္ၿပီးတိုင္း j တန္ဖိုးကို 1 တိုးေပးတာပါ။ ေအာက္ကပံုမွာ for loop ရဲ႕ flowchart ကို ျပထားပါတယ္။

**How Many Times?**

fordemo.cpp ပရိုဂရမ္ေလးမွာ ပါတဲ႔ for loop ဟာ ၁၅ ႀကိမ္တိတိ အလုပ္လုပ္ပါတယ္။ ပထမဦးဆံုးမ်ာ j တန္ဖိုးက 0 ျဖစ္ပါတယ္။ ဒါကို initialization expression မွာ သတ္မွတ္ေပးပါတယ္။

ေနာက္ဆံုး ျဖစ္ရမယ္႔ j တန္ဖိုးက 14 ပါ။ ဒါကို test expression j<15 က သတ္မွတ္ေပးထားတာပါ။ တကယ္လို႔ j တန္ဖိုး 15 ျဖစ္သြားၿပီး 14 ထက္ ႀကီးသြားမယ္ဆိုရင္ အဲဒီ expression မွားသြားလို႔ for loop ကေန ထြက္သြားမွာ ျဖစ္ပါတယ္။ မ်ားေသာအားျဖင္႔ loop variable ကို 0 ထားပါတယ္။ test expression ကို less-than < operator ကိုသံုးၿပီး ေနာက္က တန္ဖိုးကို ကၽြန္ေတာ္တို႔ ပတ္ခ်င္တဲ႔ အႀကိမ္အေရအတြက္အတိုင္းထားပါတယ္။ (ဒီ ဥပမာမွာက ၁၅ ျဖစ္ပါတယ္)။ တစ္ပတ္ ပတ္ၿပီးတိုင္း increment လုပ္ပါတယ္။ fordemo.cpp မွာေတာ႔ loop body မွာ အလုပ္လုပ္ဖို႔ ကုဒ္ တစ္ေၾကာင္းသာ ေရးထားပါတယ္။ cout << j \* j << “ “;



ေနာက္ထပ္ ဥပမာေလး တစ္ခု ေရးျပပါ႔မယ္။

**For(count=0;count<100;count++)**

**//loop body**

အထက္ပါ ပရိုဂရမ္ေလးဟာ အႀကိမ္ ၁၀၀ အတိအက် လုပ္ပါလိမ္႔မယ္။ count တန္ဖိုး 0 ကေန 99 အထိ for loop ကို ပတ္ေနမွာေၾကာင္႔ပဲ ျဖစ္ပါတယ္။

**Multiple Statements in the Loop Body**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ loop body ထဲမွာ statement တစ္ေၾကာင္းမက ေရးဖို႔ လိုအပ္ေလ႔ ရွိပါတယ္။ အဲဒီလို အေျခအေနမ်ိဳးမွာ function ေတြ ေရးသလိုမ်ိဳး တြန္႔ကြင္း{} အဖြင္႔အပိတ္ၾကားမွာ ေရးေပးရမွာ ျဖစ္ပါတယ္။ ကြင္းထဲက statement တစ္ေၾကာင္းခ်င္းရဲ႕ အဆံုးမွာ semicolon ခံေပးရမွာ ျဖစ္ေပမယ္႔ ကြင္းပိတ္ေနာက္မွာေတာ႔ demicolon ထည္႔မေပးရပါဘူး။ cubelist.cpp ဆိုတဲ႔ ေနာက္ နမူနာေလး ေရးျပထားပါတယ္။ ၄င္းက ၁ ကေန ၁၀ၾကား ဂဏန္းေတြရဲ႕ သံုးထပ္ကိန္းေတြကို ေကာ္လံ ႏွစ္ခု နဲ႔ တြက္ထုတ္ေပးမွာ ျဖစ္ပါတယ္။ ေလ႔လာၾကည္႔ၾကရေအာင္..

// cubelist.cpp

// lists cubes from 1 to 10

#include <iostream>

#include <iomanip> //for setw

using namespace std;

int main()

{

int numb; //define loop variable

for(numb=1; numb<=10; numb++) //loop from 1 to 10

{

cout << setw(4) << numb; //display 1st column

int cube = numb\*numb\*numb; //calculate cube

cout << setw(6) << cube << endl; //display 2nd column

}

return 0;

}

အေျဖေတြကေတာ႔ ေအာက္ပါအတိုင္း ျဖစ္ပါတယ္-

1 1

2 8

3 27

4 64

5 125

6 216

7 343

8 512

9 729

10 1000

ဒီပရိုဂရမ္ေလးမွာ ေနာက္ထပ္ ေျပာင္းလဲမႈ အနည္းငယ္ကို ျဖည္႔စြက္ထားပါတယ္။ အရင္က loop variable ကို 0 နဲ႔ initialized လုပ္ခဲ႔ေပမယ္႔ အခုေတာ႔ 1 နဲ႔ လုပ္ထားပါတယ္။ test expression မွာလဲ 9 မသံုးဘဲ 10 ကိုသံုးထားပါတယ္။ အဲဒီလို သံုးႏိုင္ဖို႔ relational operator ကို < အစား <= (less-than-or-equal-to operator) ကို ေျပာင္းသံုးထားပါတယ္။ ရလာဒ္က loop ကို ဆယ္ပတ္ ပတ္တာပါ။ 0-9 အစား 1-10 ေျပာင္းေပးလိုက္တာပါပဲ။

ေနာက္တစ္ခုက loop body ေတြမွာ statement တစ္ေၾကာင္းတည္း ရွိေနခဲ႔မယ္ဆိုရင္ တြန္႔ကြင္းမခတ္ဘဲ ေရးေလ႔ ရွိသလို ကြင္းခတ္ၿပီးလည္း ေရးႏိုင္ပါတယ္။ တြန္႔ကြင္းကို မျဖစ္မေန မလိုအပ္ေပမယ္႔ ပရိုဂရမ္မာ အမ်ားစုကေတာ႔ ဖတ္ရတာ ရွင္းလင္းလြယ္ကူေအာင္ loop body ေရးတိုင္း ထည္႔ေပးတတ္ၾကပါတယ္။

**Blocks and Variable Visibility**

အဲဒီလို statements ေတြ အမ်ားႀကီးကို တြန္႔ကြင္းနဲ႔ ခတ္ထားတာကို block of code လို႔ ေခၚပါတယ္။ အေရးႀကီးတဲ႔ အခ်က္တစ္ခုက အဲဒီ block ထဲမွာ defined လုပ္ထားတဲ႔ variable ဟာ block အျပင္က မျမင္ႏိုင္( ယူမသံုးႏိုင္ ) တာပဲ ျဖစ္ပါတယ္။ cubelist.cpp မွာ ကၽြန္ေတာ္တို႔ အေနနဲ႔ variable cube ကို block အတြင္းမွာ define လုပ္ထားခဲ႔ပါတယ္။

**int cube = numb\*numb\*numb;**

ဒါေၾကာင္႔ သူ႔ကို block အျပင္ကေန access မလုပ္ႏိုင္ပါဘူး။ ဒါေၾကာင္႔ တကယ္လို႔ ကၽြန္ေတာ္တို႔အေနနဲ႔ cube = 10; ဆိုတဲ႔ ကုဒ္ေလးကို block အျပင္ကေန ေရးသားခဲ႔မယ္ဆိုရင္ compiler ကေန error message ေပးမွာျဖစ္ပါတယ္။ ဘာလို႔လဲဆိုေတာ႔ cube ဆိုတဲ႔ variable ဟာ block အျပင္ဘက္မွာ undefined ျဖစ္ေနလို႔ပဲ ျဖစ္ပါတယ္။

အဲဒီလို ကန္႔သတ္ခ်က္ ရွိေနျခင္းရဲ႕ အားသာခ်က္တစ္ခုကေတာ႔ မတူညီတဲ႔ block ေတြ အတြင္းမွာ နာမည္တူၿပီး တကယ္တမ္း မတူညီတဲ႔ variable ေတြ သံုးလို႔ ရတာပဲ ျဖစ္ပါတယ္။ (ဒါေပမယ္႔ စာေရးသူကေတာ႔ အဲဒီလို သံုးတာကို အားမေပးပါဘူး။ မွားသြားရင္ ေသခ်ာေပါက္ ရြာလည္သြားႏိုင္ပါတယ္။ variable ေတြကို နာမည္ သီးသန္႔စီ ေသေသခ်ာခ်ာေပးတာဟာ မျဖစ္မေန လုပ္သင္႔ပါတယ္။)

**Indentation and Loop Style**

ကၽြန္ေတာ္တို႔ ပရိုဂရမ္ေရးသားတဲ႔ ေနရာမွာ ျပန္လည္ ဖတ္ရႈရလြယ္ကူၿပီး ရွင္းလင္းတဲ႔ ေရးဟန္ ရွိဖို႔ လိုအပ္ပါတယ္။ loop ေတြကို ေရးသားတဲ႔ ေနရာမွာလဲ ဖတ္ရလြယ္ၿပီး ရွင္းေနေအာင္ ညာဘက္ကို indented လုပ္ေပးဖို႔ လိုအပ္ပါတယ္။ ဆိုလိုတာက tab ခံၿပီး ေရးသားဖို႔ကို ေျပာတာ ျဖစ္ပါတယ္။ ဒီလို ေရးသားျခင္းေၾကာင္႔ loop အစနဲ႔ အဆံုးကို အလြယ္တကူ ျမင္ႏိုင္မွာ ျဖစ္ပါတယ္။ ကြန္ပိုင္လာ ကေတာ႔ indent လုပ္၊ မလုပ္ မသိပါဘူး။ ဒါေပမယ္႔ ကၽြန္ေတာ္တို႔ ပရိုဂရမ္မာေတြ ဖတ္ရ၊ ျပင္ဆင္ရ လြယ္ေအာင္ေတာ႔ လုပ္ေပးသင္႔ပါတယ္။

ကၽြန္ေတာ္တို႔ တြန္႔ကြင္းအစကို ေရးသားတဲ႔ ေနရာမွာ တြန္႔ကြင္းအဆံုးနဲ႔ တစ္တန္းတည္း ေရးသားေလ႔ ရွိပါတယ္။ ဒါေပမယ္႔ အခ်ိဳ႕ပရိုဂရမ္မာေတြကေတာ႔ ေအာက္ပါပံုစံမ်ိဳးနဲ႔ loop statement အဆံုးမွာ တြန္႔ကြင္း အစကို ေရးတတ္ၾကပါတယ္ -

for(numb=1; numb<=10; numb++) {

cout << setw(4) << numb;

int cube = numb\*numb\*numb;

cout << setw(6) << cube << endl;

}

ဒီလိုေရးသားျခင္းအားျဖင္႔ ကုဒ္တစ္လိုင္း ပိုတိုသြားေစပါတယ္။ ဒါေပမယ္႔ တြန္႔ကြင္းအစကို မ်က္စိရွမ္းသြားတတ္ၿပီး ဖတ္ရ ပိုခက္သြားေစပါတယ္။ ဖတ္ရရွင္းလင္း လြယ္ကူတဲ႔ ေရးနည္းကေတာ႔ တြန္႔ကြင္း အဖြင္႔နဲ႔ အပိတ္ကို တစ္တန္းတည္းထားၿပီး loop body ကိုေတာ႔ indent ျပဳလုပ္ထားတဲ႔ style ပဲျဖစ္ပါတယ္။

for(numb=1; numb<=10; numb++)

{

cout << setw(4) << numb;

int cube = numb\*numb\*numb;

cout << setw(6) << cube << endl;

}

ဘယ္ style ပဲသံုးသံုး ဖတ္ေနက်မဟုတ္တဲ႔ သူအတြက္ေတာ႔ အခက္အခဲ အနည္းငယ္ ျဖစ္ႏိုင္ပါတယ္။ ဒါေၾကာင္႔ style တစ္ခုတည္းကိုပဲ စြဲစြဲျမဲျမဲ အသံုးျပဳၾကဖို႔ အၾကံျပဳလုိပါတယ္။

**Debugging Animation**

Loop operation ေတြ ဘယ္လို လုပ္ေဆာင္တယ္ဆိုတာကို အေသးစိတ္ ေလ႔လာႏိုင္ဖို႔အတြက္ compiler ေတြမွာ ပါေလ႔ရွိတဲ႔ debugging features ေတြကို အသံုးျပဳသင္႔ပါတယ္။ single-stepping ကိုသံုးၿပီး ကုဒ္တစ္ေၾကာင္းခ်င္း ဘယ္လို အလုပ္လုပ္တယ္၊ variable ေတြ ဘယ္လို ေျပာင္းလဲသြားတယ္ ဆိုတာေတြကို ေလ႔လာႏိုင္ပါတယ္။ အဲဒီအေၾကာင္းကို ေနာက္ပိုင္းမွာ အေသးစိတ္ ေရးပါဦးမယ္။

**for Loop Variations**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ increment expression ကို loop variable ေတြကို increment လုပ္ဖို႔သာမက အျခား သင္႔ေတာ္ရာ operation မ်ားကိုလည္း လုပ္ေဆာင္ဖို႔ အသံုးျပဳႏိုင္ပါတယ္။ ေနာက္ ဥပမာ တစ္ခုမွာ ၄င္းကို loop variable တန္ဖိုး decrement လုပ္ဖို႔ အသံုးျပဳထားပါတယ္။ factor.cpp ဆိုတဲ႔ ပရိုဂရမ္ေလးမွာ user ထည္႔ေပးလိုက္တဲ႔ ဂဏန္း တစ္ခုရဲ႕ factorial ကို ရွာျပထားပါတယ္။ (factorial ဆိုတာကေတာ႔ မူလဂဏန္းကို သူ႔ထက္ငယ္တဲ႔ positive integers ေတြ အားလံုးနဲ႔ ေျမွာက္ထားတာပါ။ ဒါေၾကာင္႔ 5 ရဲ႕ factorial ဟာ 5\*4\*3\*2\*1 = 210 ျဖစ္ပါတယ္)။

// factor.cpp

// calculates factorials, demonstrates FOR loop

#include <iostream>

using namespace std;

int main()

{

unsigned int numb;

unsigned long fact=1; //long for larger numbers

cout << “Enter a number: “;

cin >> numb; //get number

for(int j=numb; j>0; j--) //multiply 1 by

fact \*= j; //numb, numb-1, ..., 2, 1

cout << “Factorial is “ << fact << endl;

return 0;

}

ဒီဥပမာေလးမွာ j တန္ဖိုးကို initialization expression သံုးၿပီး user ထည္႔ေပးတဲ႔ တန္ဖိုးကို ေပးလိုက္ပါတယ္။ test expression ကေတာ႔ j တန္ဖိုး 0 ထက္ ႀကီးေနသမွ် loop ပတ္ေနေစပါတယ္။ Increment expression ကေတာပ j တန္ဖိုးကို loop တစ္ႀကိမ္ ပတ္ၿပီးတိုင္း တစ္ ေလွ်ာ႔ေပးပါတယ္။ ဒီေနရာမွာ unsigned long ကို factorial တန္ဖိုး ထည္႔သြင္းဖို႔ အသံုးျပဳထားတာ ျဖစ္ပါတယ္။ factorial တန္ဖိုးေတြက ႀကီးမားတတ္လို႔ပဲ ျဖစ္ပါတယ္။ ေအာက္မွာ ျပထားတဲ႔ factorial တန္ဖိုးေတြကို ၾကည္႔ျခင္းအားျဖင္႔ ခန္႔မွန္းႏိုင္ၾကပါတယ္။

**Enter a number: 10**

**Factorial is 3628800**

ဒီ ပရိုဂရမ္မွာ ကၽြန္ေတာ္တို႔ အႀကီးဆံုး ရွာႏိုင္တဲ႔ factorial ကေတာ႔ 12 အတြက္ပဲ ျဖစ္ပါတယ္။ အဲဒီထက္ ႀကီးတဲ႔ တန္ဖိုးေတြကို ထည္႔သြင္းခဲ႔မယ္ ဆိုရင္ေတာ႔ error message မျပဘဲ မွားယြင္းတဲ႔ ရလာဒ္ေတြ ထြက္လာမွာပဲ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ variable ေတြကို define လုပ္တဲ႔ ေနရာမွာ data type မွန္ကန္ေအာင္ သတ္မွတ္ေပးႏိုင္ဖို႔ အေရးႀကီးတာပဲ ျဖစ္ပါတယ္။

**Variables Defined in for Statements**

ေနာက္နည္းတစ္ခုကေတာ႔ loop variable (ဒီဥပမာမွာ j)ကို loop statement ထဲမွာပဲ define ျပဳလုပ္ေပးျခင္းပဲ ျဖစ္ပါတယ္။

**for(int j=numb; j>0; j--)**

အသံုးမ်ားသလို ေကာင္းမြန္တဲ႔ loop variable define ျပဳလုပ္နည္းပဲ ျဖစ္ပါတယ္။ အဲဒီလို define ျပဳလုပ္လိုက္တဲ႔ variable ကို loop body ကပဲ ျမင္ႏိုင္ (အသံုးျပဳႏိုင္) ပါတယ္။ (ကြန္ပိုင္လာ ေပၚမူတည္ၿပီး ျခြင္းခ်က္ေတာ႔ ရွိႏိုင္ပါတယ္။ ဒါေပမယ္႔ Standard C++ မွာေတာ႔ မျမင္ႏိုင္ပါဘူး)။

**Multiple Initialization and Test Expressions**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ for statement ေရးသားတဲ႔ ေနရာမွာ initialization ကို expression တစ္ခုထက္ ပိုၿပီး ေရးလို႔ ရပါတယ္။ comma နဲ႔ ျခားေပးရပါတယ္။ အဲဒီလိုပဲ increment expression ကို လဲ ေရးလို႔ရပါတယ္။ ဒါေပမယ္႔ test expression ကိုေတာ႔ တစ္ခုပဲ ေရးသားလို႔ ရပါတယ္။

**for( j=0, alpha=100; j<50; j++, beta-- )**

**{**

**// body of loop**

**}**

အထက္ပါ ဥပမာမွာေတာ႔ normal loop variable j ကို သံုးထားပါတယ္။ ေနာက္ထပ္ အျခား variable တစ္ခုျဖစ္တဲ႔ alpha ကိုလည္း initialize လုပ္ေပးထားပါတယ္။ incremental expression မွာေတာ႔ j ကို ပံုမွန္ increment လုပ္ေပးထားၿပီး တတိယ variable ျဖစ္တဲ႔ beta ကိုေတာ႔ decrement လုပ္ထားပါတယ္။

တကယ္ေတာ႔ ကၽြန္ေတာ္တို႔ လိုအပ္လာရင္ အဲဒီ expression အခ်ိဳ႕၊ သို႔မဟုတ္ အားလံုးကို မေရးဘဲ ခ်န္လွပ္ထားခဲ႔ႏိုင္ပါတယ္။ ဥပမာ for(;;) ဟာ while loop ကို test expression true ထည္႔ေပးထားတာနဲ႔ တူညီပါတယ္။ ေနာက္ပိုင္းမွာ while loop ေတြ အေၾကာင္းကို ေရးသြားေပးသြားမွာပါ။ အဲဒီလို ေရးနည္းမ်ိဳးကို အတတ္ႏိုင္ဆံုး ေရွာင္ရမွ ျဖစ္ပါတယ္။ ဖတ္ရခက္ခဲ ေစတာေၾကာင္႔ရယ္ ရလာဒ္ အတူတူရေအာင္ အျခားနည္းလမ္းေတြနဲ႔ ပိုမိုရိုးရွင္းစြာ ေရးသားႏိုင္ေသးတာေၾကာင္႔ပဲ ျဖစ္ပါတယ္။

**The while Loop**

အထက္မွာ တိက်တဲ႔ အႀကိမ္အေရအတြက္ အတိုင္း looping ပတ္ဖို႔အတြက္ for loop ကို အသံုးနည္းမ်ားကို ေလ႔လာခဲ႔ၿပီးပါၿပီ။ ဒါေပမယ္႔ တကယ္လို႔ ကၽြန္ေတာ္တို႔ ပတ္မယ္႔ loop အေရအတြက္ကို ႀကိဳမသိႏိုင္ခဲ႔ဘူးဆိုရင္ အျခားနည္းကို သံုးရပါလိမ္႔မယ္။ အဲဒါကေတာ႔ while loop ပဲ ျဖစ္ပါတယ္။ ဥပမာအေနနဲ႔ endon0.cpp ပရိုဂရမ္ကို ေရးျပထားပါတယ္။ အဲဒီ မွာ user ကို ဂဏန္းေတြ တစ္ခုၿပီး တစ္ခု ေတာင္းပါမယ္။ တကယ္လို႔ user က 0 ကို ထည္႔သြင္းလိုက္တဲ႔ အခါမွသာ ပရိုဂရမ္ ၿပီးဆံုးသြားမွာ ျဖစ္ပါတယ္။ ဒီေနရာမွာ ပရိုဂရမ္ အေနနဲ႔ user က 0 မထည္႔သြင္းခင္ ဘယ္ႏွစ္ႀကိမ္ ဂဏန္းေတြ ထည္႔ေပးမယ္ဆိုတာကို ႀကိဳမသိပါဘူး။ user စိတ္ႀကိဳက္ ထည္႔သြင္းႏိုင္ပါလိမ္႔မယ္။

// endon0.cpp

// demonstrates WHILE loop

#include <iostream>

using namespace std;

int main()

{

int n = 99; // make sure n isn’t initialized to 0

while( n != 0 ) // loop until n is 0

cin >> n; // read a number into n

cout << endl;

return 0;

}

ေအာက္မွာေတာ႔ နမူနာအေနနဲ႔ ပရိုဂရမ္ထဲကို ဂဏန္းေတြ ထည္႔သြင္းျပထားပါတယ္။ 0 ကို ထည္႔သြင္းလိုက္တဲ႔ အခါမွာေတာ႔ loop ကေနထြက္ၿပီး ပရိုဂရမ္ ၿပီးဆံုးသြားမွာ ျဖစ္ပါတယ္။

1

27

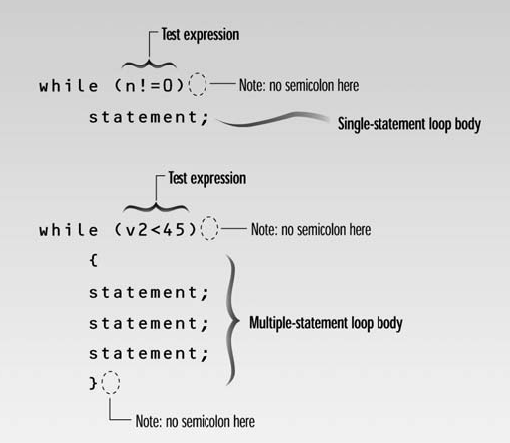
33

144

9

0

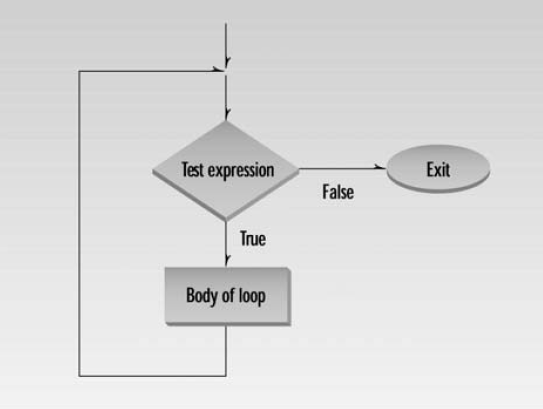
while loop ဟာ for loop ကို ပိုမိုရိုးရွင္းေအာင္ ျပဳလုပ္ထားတာနဲ႔ တူေနပါတယ္။ သူ႔မွာ test expression သာပါရွိၿပီး initialization နဲ႔ increment expression ေတြ မပါဝင္ပါဘူး။ ေအာက္က ပံုေလးမွာ while loop ရဲ႕ syntax ကို ျပထားပါတယ္။



while loop မွာ test expression မွန္ေနသမွ် ကာလပတ္လံုး loop ကို ပတ္ေနမွာ ျဖစ္ပါတယ္။ endon0.cpp မွာပါရွိတဲ႔ n!=0 (n note equal to 0) ဆိုတဲ႔ test expression ေၾကာင္႔ n တန္ဖိုး 0 မျဖစ္မခ်င္း test result မွန္ကန္ေနၿပီး loop ကို ပတ္ေနမွာ ျဖစ္ပါတယ္။ while loop မွာ initialization expression မပါေပမယ္႔ loop variable (ဒီဥပမာမွာေတာ႔ n) ကို loop မစခင္မွာ initialize လုပ္ေပးထားဖို႔ လိုအပ္မွာ ျဖစ္ပါတယ္။ ေနာက္တစ္ခုက loop body ထဲမွာ အဲဒီ loop variable ကို ေျပာင္းလဲေပးတဲ႔ statement ပါဖို႔လိုအပ္ပါတယ္။ မပါခဲ႔ရင္ loop က ထြက္ဖို႔ မျဖစ္ႏိုင္ေတာ႔ပါဘူး။ endon0.cpp မွာေတာ႔ cin<<n; ဆိုတဲ႔ ကုဒ္က n တန္ဖိုးကို ေျပာင္းလဲေစပါတယ္။

**Multiple Statements in a while Loop**

ေနာက္ထပ္ နမူနာ ပရိုဂရမ္ တစ္ခုကို ေရးျပပါဦးမယ္။ while4.cpp မွာ while loop အတြင္း multiple statements ေတြ ေရးျပထားပါတယ္။ တကယ္ေတာ႔ ဒီလို multiple statements ေတြ ေရးတဲ႔ အေၾကာင္းကို cubelist.cpp မွာတုန္းက foor loop ထဲမွာ ေရးျပခဲ႔ၿပီးပါၿပီ။ ဒီပရိုဂရမ္မွာေတာ႔ ေလးထပ္ကိန္းေတြကို ရွာျပထားပါတယ္။ ေလးထပ္ကိန္းရဲ႕ တန္ဖိုးေတြကို ဂဏန္းေလးလံုး အထိပဲ ရွာဖို႔ ဆံုးျဖတ္ထားတယ္ ဆိုၾကပါစို႔။ ဒါဆိုရင္ အမ်ားဆံုး တန္ဖိုးက 9999 ထက္ မႀကီးရပါဘူး။ အဲဒီလို အေျခအေနမ်ိဳးကို ကန္႔သတ္ဖို႔ အတြက္ဆိုရင္ တြက္မၾကည္႔ရေသးပဲနဲ႔ ဘယ္ဂဏန္းရဲ႕ ေလးထပ္ကိန္းက အဲဒီ limit ကို ေက်ာ္သြားမယ္ဆိုတာကို မသိႏိုင္ပါဘူး။ ဒါေၾကာင္႔ test expression မွာ အဲဒီ အခ်က္ကို စစ္ေဆးခိုင္းၿပီး limit ေက်ာ္သြားတာနဲ႔ loop ကို ရပ္လိုက္မွာ ျဖစ္ပါတယ္။



// while4.cpp

// prints numbers raised to fourth power

#include <iostream>

#include <iomanip> //for setw

using namespace std;

int main()

{

int pow=1; //power initially 1

int numb=1; //numb goes from 1 to ???

while( pow<10000 ) //loop while power <= 4 digits

{

cout << setw(2) << numb; //display number

cout << setw(5) << pow << endl; //display fourth power

++numb; //get ready for next power

pow = numb\*numb\*numb\*numb; //calculate fourth power

}

cout << endl;

return 0;

}

ေလးထပ္ကိန္းကို ရွာဖို႔အတြက္ကေတာ႔ အဲဒီဂဏန္းကို ေလးခါ ေျမွာက္ေပးလိုက္တာပါ။ loop တစ္ခါပတ္တိုင္း numb ဆိုတဲ႔ variable ကို 1 တိုးေပးသြားမွာပါ။ ဒါေပမယ္႔ test expression မွာေတာ႔ numb ကို မစစ္ဘဲ ေလးထပ္ကိန္းတန္ဖိုး pow ကိုပဲ စစ္ေဆးၿပီး loop ကို ဘယ္အခ်ိန္မွာ ထြက္ရမယ္ဆိုတာကို ဆံုးျဖတ္သြားမွာပါ။ ေအာက္မွာ မူရင္းဂဏန္းေတြနဲ႔ သူတို႔ရဲ႕ ေလးထပ္ကိန္းေတြကို တြက္ျပထားပါတယ္။ ေလးထပ္ကိန္းတန္ဖိုး 9999 ထက္ႀကီးသြားတာနဲ႔ ပရိုဂရမ္ၿပီးဆံုးသြားမွာ ျဖစ္ပါတယ္။

1 1

2 16

3 81

4 256

5 625

6 1296

7 2401

8 4096

9 6561

**Precedence: Arithmetic and Relational Operators**

ေနာက္ပရိုဂရမ္ တစ္ပုဒ္ကေတာ႔ Fibonacci series ကို တြက္ထုတ္ေပးမယ္႔ fibo.ccp ပဲ ျဖစ္ပါတယ္။ Fibonacci series ဆိုတာဟာ ေရွ႕ဂဏန္း ႏွစ္လံုးေပါင္းျခင္းျဖင္႔ ေနာက္ဂဏန္း ကို ရွာယူထားတဲ႔ ဂဏန္းတြဲေတြ ျဖစ္ပါတယ္။ ၄င္းရဲ႕ အစပိုင္း ဂဏန္းတစ္ခ်ိဳ႕ကေတာ႔ 1 1 2 3 5 8 13 21 34 55 တို႔ပဲ ျဖစ္ပါတယ္။ တကယ္ေတာ႔ ဒီ series ကို လက္ေတြ႔ အသံုးခ်မႈေပါင္းမ်ားစြာ ရွိပါတယ္။ ၄င္းဟာ ဗိသုကာပညာနဲ႔ အႏုပညာမွာ သံုးေလ႔ရွိတဲ႔ golden ratio နဲ႔လဲ ဆက္စပ္ေနပါေသးတယ္။ ဒီေနရာမွာေတာ႔ အဲဒီအေၾကာင္းေတြကို အက်ယ္ရွင္းမျပေတာ႔ပါဘူး။

// fibo.cpp

// demonstrates WHILE loops using fibonacci series

#include <iostream>

using namespace std;

int main()

{ //largest unsigned long

const unsigned long limit = 4294967295;

unsigned long next=0; //next-to-last term

unsigned long last=1; //last term

while( next < limit / 2 ) //don’t let results get too big

{

cout << last << “ “; //display last term

long sum = next + last; //add last two terms

next = last; //variables move forward

last = sum; // in the series

}

cout << endl;

return 0;

}

ရလာဒ္ေတြကေတာ႔ ေအာက္ပါအတိုင္းပဲ ျဖစ္ပါတယ္။

**1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987**

**1597 2584 4181 6765 10946 17711 28657 46368 75025 121393**

**196418 317811 514229 832040 1346269 2178309 3524578**

**5702887 9227465 14930352 24157817 39088169 63245986**

**102334155 165580141 267914296 433494437 701408733 1134903170**

**1836311903 2971215073**

အဲဒီရလာဒ္ေတြထဲက ေနာက္ဆံုး ဂဏန္းႏွစ္လံုးကို အခ်ိဳးခ်လိုက္မယ္ဆိုရင္ 0.618033988 ကို ရရွိမယ္ျဖစ္ၿပီး ၄င္းဟာ golden ratio ပဲျဖစ္ပါတယ္။ fibo.cpp ပရိုဂရမ္မွာ အႀကီးဆံုး positive integers ေတြ ထည္႔ထားႏိုင္ဖို႔ unsigned long data type ကို အသံုးျပဳထားပါတယ္။

Test expression မွာ အဲဒီ data type ရဲ႕ ကန္႔သတ္ခ်က္ကို ေက်ာ္မသြားတဲ႔ အခ်ိန္အထိ loop ကို ပတ္ေနေအာင္ ေရးသားထားပါတယ္။ limit တန္ဖိုးကိုေတာ႔ const သတ္မွတ္ေပးထားၿပီး မေတာ္တဆ ေျပာင္းလဲေရးသားမိျခင္းကေန ကာကြယ္ထားပါသည္။ loop ကို limit တန္ဖိုးရဲ႕ တစ္ဝက္ေရာက္လာရင္ ထြက္ဖို႔ ေရးသားရပါတယ္။ ဒါမွလဲ sum တန္ဖိုးက limit ကို ေက်ာ္မသြားမွာ ျဖစ္ပါသည္။ test expression မွာ operators ႏွစ္ခု သံုးထားပါတယ္။ (next < limit / 2)။ ကၽြန္ေတာ္တို႔ရဲ႕ ရည္ရြယ္ခ်က္က next ကို limit/2 နဲ႔ ႏိႈင္းယွဥ္ဖို႔ ျဖစ္ပါတယ္။ division ကို comparison မတိုင္မီ လုပ္ေဆာင္ဖို႔ လိုပါတယ္။ အဲဒီအခ်က္ကို ေသခ်ာေစဖို႔ လက္သဲကြင္းထဲ ထည္႔ေရးလို႔ရပါတယ္။ (next < (limit/2) )။ ဒါေပမယ္႔ လက္သဲကြင္းကို ထည္႔ေရးဖို႔ မလိုပါဘူး။ arithmetic operators ေတြဟာ relational operators ေတြထက္ precedence ပိုျမင္႔လို႔ပဲ ျဖစ္ပါတယ္။

**The do Loop**

**While loop** ကို နားလည္သြားၿပီဆိုရင္ do loop ကို သေဘာေပါက္ဖို႔ မခက္ေတာ႔ပါဘူး။ while loop မွာတုန္းက test expression မွန္မွန္ စစ္ေဆးျခင္းကို loop ရဲ႕ အစမွာ ျပဳလုပ္တာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ အဲဒီအခ်က္ မွားသြားတာနဲ႔ looping တစ္ႀကိမ္မွ မပတ္ေတာ႔ဘဲ loop ကို ေက်ာ္သြားမွာ ျဖစ္ပါတယ္။ တစ္ခါတစ္ေလမွာ test expression ကို မစစ္ေသးဘဲ loop ကို အနည္းဆံုး တစ္ႀကိမ္ေတာ႔ ပတ္ေစခ်င္တဲ႔ အခါမ်ိဳးေတြ ရွိတတ္ပါတယ္။ အဲဒီအေျခအေနမ်ိဳးမွာဆို do loop ကို while loop အစား အသံုးျပဳရေလ႔ ရွိပါတယ္။ ဒါေၾကာင္႔ do loop မွာ test expression ကို loop အဆံုးမွာ ထားထားတာျဖစ္ၿပီး အနည္းဆံုး တစ္ေခါက္ ပတ္ၿပီးမွ test expression ကို စတင္ စစ္ေဆးတာ ျဖစ္ပါတယ္။ ေအာက္မွာ ေဖာ္ျပထားတဲ႔ divdo.cpp ပရိုဂရမ္ေလးမွာေတာ႔ တည္ကိန္းနဲ႔ စားကိန္း ႏွစ္ခုကို user အား ထည္႔သြင္းေစၿပီး စားလာဒ္နဲ႔ အၾကြင္းကို / နဲ႔ % operator ေတြသံုးကာ ရွာေဖြေပးမွာ ျဖစ္ပါတယ္။ do loop အသံုးျပဳထားပံုေလးကို ဂရုစိုက္ၿပီး ေလ႔လာေစခ်င္ပါတယ္။

// divdo.cpp

// demonstrates DO loop

#include <iostream>

using namespace std;

int main()

{

long dividend, divisor;

char ch;

do //start of do loop

{ //do some processing

cout << “Enter dividend: “; cin >> dividend;

cout << “Enter divisor: “; cin >> divisor;

cout << “Quotient is “ << dividend / divisor;

cout << “, remainder is “ << dividend % divisor;

cout << “\nDo another? (y/n): “; //do it again?

cin >> ch;

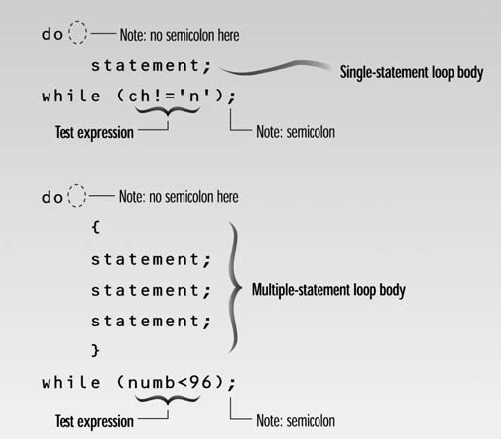
}

while( ch != ‘n’ ); //loop condition

return 0;

}

ပရိုဂရမ္ရဲ႕ ကုဒ္အမ်ားစုကို do loop ထဲမွာ ေရးသားထည္႔သြင္းထားတာကို သတိထားမိပါလိမ္႔မယ္။ do loop မွာ loop ရဲ႕ အစကို do ဆိုတဲ႔ keyword နဲ႔ သတ္မွတ္ေပးလိုက္တာ ျဖစ္ပါတယ္။ ေနာက္က တြန္႔ကြင္း အဖြင္႔ အပိတ္ၾကားမွာေတာ႔ loop body ကို ေရးသားထားတာ ျဖစ္ၿပီး အဆံုးသတ္မွာ test expression ထည္႔သြင္းထားကာ semicolon နဲ႔ ပိတ္ေပးရမွာ ျဖစ္ပါတယ္။ (semicolon ထည္႔ဖို႔ မၾကာခဏ ေမ႔တတ္တာကို သတိျပဳပါ)။ do loop ရဲ႕ syntax ကို ေအာက္မွာ ျပထားပါတယ္။



divdo.cpp ပရိုဂရမ္ေလးထဲမွာ တြက္ခ်က္မႈေတြ ၿပီးတဲ႔ ေနာက္မွာ user ကို ေနာက္ထပ္ ဆက္တြက္ခ်င္ေသးလားလို႔ ေမးပါတယ္။ တကယ္လို႔ user က ဆက္တြက္ခ်င္ေသးရင္ y ကို ရိုက္ထည္႔ရမွာ ျဖစ္ၿပီး loop ကို အၿပီးထြက္ခ်င္ရင္ေတာ႔ n ကို ရိုက္ထည္႔ရမွာပါ။ တကယ္တမ္းက test expression မွာ ch != ‘n’ လို႔ စစ္ထားတဲ႔ အတြက္ n မထည္႔သမွ် true ျဖစ္ေနၿပီး loop ကို ပတ္ေနမွာပါ။ n ထည္႔လိုက္ေတာ႔မွ false ျဖစ္သြားတဲ႔ အတြက္ loop ထဲက ထြက္သြားမွာ ျဖစ္ပါတယ္။ ေအာက္မွာ နမူနာ စမ္းသပ္ျပထားပါတယ္။

Enter dividend: 11

Enter divisor: 3

Quotient is 3, remainder is 2

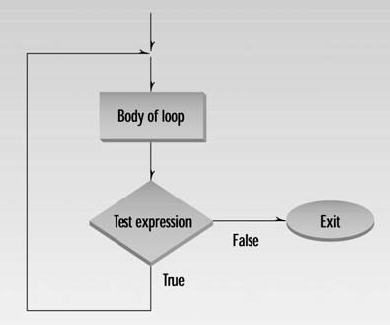
Do another? (y/n): y

Enter dividend: 222

Enter divisor: 17

Quotient is 13, remainder is 1

Do another? (y/n): n



**When to Use Which Loop**

ကၽြန္ေတာ္တို႔အေနျဖင္႔ မည္သည္႔ေနရာတြင္ မည္သည္႔ loop အမ်ိဳးအစားကို အသံုးျပဳသင္႔သည္ ဆိုသည္႔အခ်က္အား အၾကမ္းအားျဖင္႔ ခြဲျခားထားေလ႔လာႏိုင္ပါသည္။ ဥပမာအားျဖင္႔ loop ပတ္ရမည္႔ အႀကိမ္အေရအတြက္ကို ႀကိဳတင္သိေနခဲ႔ပါက for loop ကို အသံုးျပဳသင္႔ပါသည္။ while loop ႏွင္႔ do loop ကိုေတာ႔ အႀကိမ္အေရတြက္ကို ႀကိဳမသိႏိုင္ဘဲ အေျခအေနတစ္ခုခု (variable value တစ္ခုခု) ေပၚ မူတည္၍ loop မွ ထြက္ရသည္႔ အခါမ်ိဳးတြင္ test expression ျဖင္႔ စစ္ေဆးကာ အသံုးျပဳရသည္။ အဆိုပါ အေျခအေနကို စစခ်င္း စစ္ေဆးသင္႔ၿပီး မမွန္ကန္ပါက loop body အား လံုးလံုး ေက်ာ္သြားရန္ လိုအပ္သည္႔ အေျခအေနမ်ိဳးတြင္ while loop အား အသံုးျပဳရသည္။ Menu ေရးသားျခင္းကဲ႔သို႔ေသာ loop body အား အနည္းဆံုး တစ္ႀကိမ္ လုပ္ေဆာင္ၿပီးမွ လိုအပ္သည္႔ အေျခအေနအား စစ္ေဆးရသည္႔ အေျခအေနမ်ိဳးတြင္ do loop ကို အသံုးျပဳသင္႔သည္။ အထက္ပါ အခ်က္မ်ားသည္ အၾကမ္းဖ်င္း သတ္မွတ္ခ်က္မ်ားသာ ျဖစ္ၿပီး ေရးသားရာတြင္ လြယ္ကူျမန္ဆန္ေစရန္ ရည္ရြယ္သည္။ သို႔ရာတြင္ အဆိုပါ loop မ်ားအားလံုးကို အေျခအေန အားလံုးနီးပါးအတြက္ အသံုးျပဳႏိုင္ေပသည္။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ မိမိတို႔ ေရးသားေသာ ပရိုဂရမ္အား ရွင္းလင္း က်စ္လစ္ေစရန္ ဂရုျပဳ၍ loop မ်ားကို ေရြးခ်ယ္သြားၾကရမည္ ျဖစ္သည္။

**Decisions**

**loop** မ်ားတြင္ ပါဝင္ေသာ decisions မ်ားသည္ loop body အား ဆက္လက္ လုပ္ေဆာင္သင္႔ မသင္႔ကို သတ္မွတ္ စစ္ေဆးျခင္းပင္ ျဖစ္သည္။ လက္ေတြ႔ဘဝတြင္လည္း ဒီေန႔ အျပင္ထြက္သင္႔ မသင္႔၊ အကၤ်ီဘာအေရာင္ ဝယ္မလဲ? အလည္အပတ္သြားသင္႔သလား? အစရွိေသာ ဆံုးျဖတ္စရာ ကိစၥရပ္မ်ား ၾကံဳေတြ႔ေနၾကရသည္သာျဖစ္သည္။ ပရိုဂရမ္အတြင္း၌လည္း ထိုကဲ႔သို႔ေသာ one-time decision မ်ား ျပဳလုပ္ရန္ လိုအပ္ေပသည္။ test expression ၏ တန္ဖိုးေပၚ မူတည္၍ ဆံုးျဖတ္ခ်က္ခ်ကာ ပရိုဂရမ္၏ အျခားေနရာမ်ားသို႔ ကုဒ္မ်ား jump ျပဳလုပ္ရမည္ ျဖစ္သည္။ C++ တြင္ decisions မ်ားကို နည္းလမ္း အမ်ိဳးမ်ိဳးျဖင္႔ ျပဳလုပ္ႏိုင္ေသာ္လည္း အေရးအႀကီးဆံုးမွာ မတူညီေသာ အေျခအေန ႏွစ္ခုထဲမွ တစ္ခုကို ေရြးခ်ယ္ေပးႏိုင္ေသာ if…else statement ပင္ ျဖစ္သည္။ ၄င္း statement ကို else မပါဘဲ if statement အျဖစ္ ရိုးစင္းစြာ အသံုျပဳႏိုင္သည္။ အေၾကာင္းအရာ အမ်ားအျပားမွ တစ္ခုကို ေရြးထုတ္ယူရန္အတြက္မူ switch statement ကို အသံုးျပဳႏိုင္သည္။ အခ်ိဳ႕ထူးျခားသည္႔ အေျခအေနမ်ိဳးတြင္ conditional operator ကို အသံုးျပဳႏိုင္သည္။ ၄င္းတို႔ အေၾကာင္းကို တစ္ဆင္႔ခ်င္း ေလ႔လာၾကရေအာင္။

**The if Statement**

ဆံုးျဖတ္ခ်က္မ်ား ျပဳလုပ္ရာတြင္ if statement သည္ အရိုးရွင္းဆံုး ျဖစ္ပါသည္။ ၄င္းကို ifdemo.cpp တြင္ ေအာက္ပါအတိုင္ ေလ႔လာႏိုင္ပါသည္။

// ifdemo.cpp

// demonstrates IF statement

#include <iostream>

using namespace std;

int main()

{

int x;

cout << “Enter a number: “;

cin >> x;

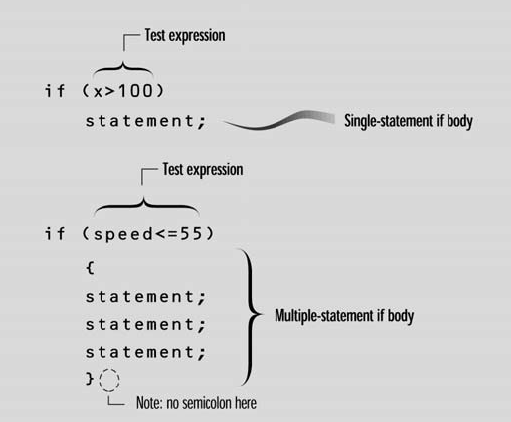
if( x > 100 )

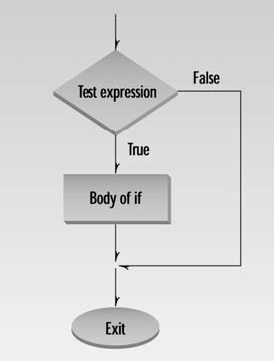
cout << “That number is greater than 100\n”;

return 0;

}

if statement ေရးတဲ႔ အခါမွာ if ဆိုတဲ႔ keyword ရဲ႕ေနာက္မွာ လက္သဲကြင္း အဖြင္႔အပိတ္နဲ႔ test expression ကို ေရးရပါတယ္။ ေအာက္ပါ if statement ရဲ႕ syntax ကို ပံုမွာ ျပထားပါတယ္။ if statement ဟာ while နဲ႔ အေတာ္ေလးတူတာကို သတိထားမိပါလိမ္႔မယ္။ ျခားနားခ်က္ကေတာ႔ if statement က while လို looping မပတ္ဘဲ test expression မွန္ခဲ႔ရင္ တစ္ႀကိမ္တည္း လုပ္ေဆာင္တာ ျဖစ္ပါတယ္။ while loop မွာေတာ႔ test expression မွန္ေနသမွ် ကာလပတ္လံုး loop ကို ပတ္ေနမွာပါ။ ေနာက္ပံုတစ္ခုမွာ if statement ရဲ႕ လုပ္ေဆာင္ပံုကို flow chat နဲ႔ ျပထားပါတယ္။





ifdemo.cpp ပရိုဂရမ္ေလးရဲ႕ ရလာဒ္ေတြကို ေအာက္မွာ ျပသထားပါတယ္။ ၁၀၀ ထက္ႀကီးတဲ႔ ဂဏန္းဆိုရင္ ပရင္႔ထုတ္ေပးေနမွာ ျဖစ္ၿပီး ၁၀၀ နဲ႔ တူသြားတာ သို႔မဟုတ္ ငယ္သြားခဲ႔ရင္ေတာ႔ ဘာမွ ထုတ္ေပးမွာ မဟုတ္ပါဘူး။

Enter a number: 2000

That number is greater than 100

**Multiple Statements in the if Body**

Loop မ်ားကို ေလ႔လာစဥ္က single statement ကို တြန္႔ကြင္း မခတ္ဘဲ ေရးလို႔ ရသလို လိုအပ္ပါက ရွင္းလင္း လြယ္ကူေစရန္ တြန္႔ကြင္းအတြင္းေရးသင္႔ေၾကာင္းႏွင္႔ multiple statements မ်ားအတြက္မူ မျဖစ္မေန တြန္႔ကြင္းအတြင္း ေရးသားရမည္ ျဖစ္ေၾကာင္း သိရွိခဲ႔ပါတယ္။ ယခု if statement မွာလဲ ထိုနည္းတူ တြန္႔ကြင္းအား အသံုးျပဳရမည္ ျဖစ္ပါတယ္။ if2.cpp မွ ေအာက္ပါအတိုင္း ေလ႔လာၾကည္႔ႏိုင္ပါတယ္။

// if2.cpp

// demonstrates IF with multiline body

#include <iostream>

using namespace std;

int main()

{

int x;

cout << “Enter a number: “;

cin >> x;

if( x > 100 )

{

cout << “The number “ << x;

cout << “ is greater than 100\n”;

}

return 0;

}

Enter a number: 12345

The number 12345 is greater than 100

**Nesting ifs Inside Loops**

Loop ေတြနဲ႔ decision ေတြဟာ တစ္ခုနဲ႔တစ္ခု nested လုပ္ၿပီး ေရးႏိုင္ၾကပါတယ္။ loop ထဲမွာ decision ေတြကို ငံုထားလို႔ ရသလို decision ေတြထဲမွာလည္း loop ေတြ ထည္႔သြင္း ထားႏိုင္ပါတယ္။ ထိုနည္းတူ loop ေတြထဲမွာ loop ေတြထပ္ငံုလို႔ရၿပီး decision ေတြထဲမွာလည္း decision ေတြ ထပ္ငံုလို႔ ရပါတယ္။ အခု သင္ခန္းစာမွာေတာ႔ for loop အတြင္းမွာ if ကို ငံုထားတဲ႔ prime.cp ဆိုတဲ႔ ပရိုဂရမ္ေလးကို ေလ႔လာၾကည္႔ၾကရေအာင္။ ဒီ ပရိုဂရမ္ေလးက prime number ေတြကို ရွာတဲ႔ ပရိုဂရမ္ေလးပါ။ prime number ဆိုတာ သူကိုယ္တိုင္ရယ္ ၁ ရယ္ကလြဲရင္ က်န္တဲ႔ ဂဏန္းနဲ႔ စားလို႔မျပတ္တဲ႔ တနည္း သုဒၶကိန္း ခြဲလို႔ မရတဲ႔ ကိန္းပဲ ျဖစ္ပါတယ္။ ဥပမာ -2, 3, 5, 7, 11, 13, 17 စတာေတြပဲ ျဖစ္ပါတယ္။

// prime.cpp

// demonstrates IF statement with prime numbers

#include <iostream>

using namespace std;

#include <process.h> //for exit()

int main()

{

unsigned long n, j;

cout << “Enter a number: “;

cin >> n; //get number to test

for(j=2; j <= n/2; j++) //divide by every integer from

if(n%j == 0) //2 on up; if remainder is 0,

{ //it’s divisible by j

cout << “It’s not prime; divisible by “ << j << endl;

exit(0); //exit from the program

}

cout << “It’s prime\n”;

return 0;

}

ဒီဥပမာေလးမွာ user ထည္႔ေပးလိုက္တဲ႔ ဂဏန္းကို n ထဲ ထည္႔လိုက္ပါတယ္။ အဲဒီေနာက္ n ကို for loop ကို အသံုးျပဳၿပီး 2 ကေန အထိ ဂဏန္းေတြနဲ႔ စားပစ္ပါတယ for(j=2; j <= n/2; j++) ္။ စားကိန္းက loop variable j ျဖစ္ပါတယ္။ အဲဒီ တန္ဖိုးထဲက တစ္ခုခုက n ကိုျပတ္ေအာင္ စားႏိုင္ခဲ႔ရင္ (တနည္း အၾကြင္း သုညျဖစ္ခဲ႔ရင္) n က prime မဟုတ္ေတာ႔ပါဘူး။ အဲဒီလို ဆံုးျဖတ္တဲ႔ အပိုင္းကို if နဲ႔ % operator သံုးၿပီး တြက္ခ်က္ ဆံုးျဖတ္ပါတယ္ if(n%j == 0)။ အကယ္၍ ထည္႔ေပးလိုက္တဲ႔ ဂဏန္းက prime number မဟုတ္ခဲ႔ရင္ user ကို ပရင္႔ထုတ္ၿပီး ျပန္အသိေပးမွာ ျဖစ္ပါတယ္။ စမ္းသပ္ျပထားပါတယ္-

Enter a number: 13

It’s prime

Enter a number: 22229

It’s prime

Enter a number: 22231

It’s not prime; divisible by 11

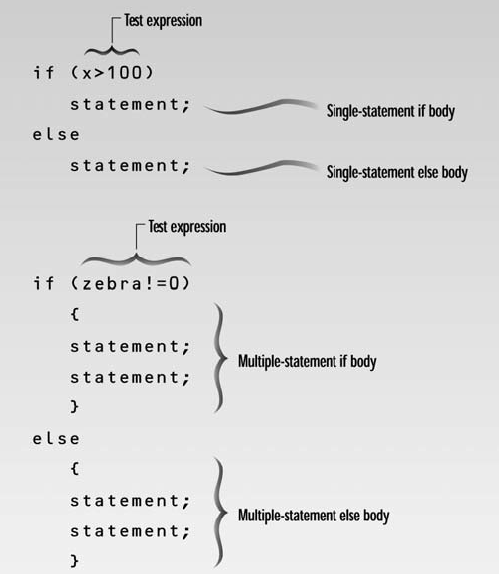
ေလာေလာဆယ္ ဥပမာမွာေတာ႔ for loop ကို တြန္႔ကြင္းနဲ႔ မခတ္ျပထားပါဘူး။ ဒါကလဲ if statement ကို statement တစ္ခုအျဖစ္သာ ယူဆေၾကာင္း ျပသခ်င္လို႔ပါ။ အရင္က ေျပာခဲ႔သလိုပဲ တြန္႔ကြင္းေတြ ထည္႔သြင္းေရးသားျခင္းက ဖတ္ရႈရလြယ္ကူေစပါတယ္။

**Library Function exit()**

**Prime** မဟုတ္မွန္းသိလို႔ program ကေန ခ်က္ျခင္း ထြက္ခ်င္တဲ႔ အခါမွာ exit() ဆိုတဲ႔ library function ကို အသံုးျပဳသင္႔ပါတယ္။ ဘယ္ေနရာကေနပဲ ျဖစ္ျဖစ္ ပရိုဂရမ္ကို ရပ္ပစ္ၿပီး ထြက္သြားမွာ ျဖစ္ပါတယ္။ သူက return value ျပန္မေပးပါဘူး။ ကြင္းထဲမွာ ထည္႔ေပးလိုက္ရတဲ႔ 0 ကေတာ႔ operating system ကို argument အေနနဲ႔ ပို႔ေပးလိုက္တာပါ။ သာမာန္အားျဖင္႔ successful termination ဆိုတဲ႔ သေဘာကို ေဆာင္ပါတယ္။ အျခားဂဏန္းေတြကေတာ႔ error ရွိေၾကာင္း ေျပာခ်င္တဲ႔ အခါ အသံုးျပဳရပါတယ္။

**The if...else Statement**

**if** statement ဟာ အေျခအေနတစ္ခု မွန္ကန္ရင္ ဘာလုပ္မယ္ဆိုတာကို ေရးသားဖို႔အတြက္ အသံုးျပဳရတာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ မွားသြားခဲ႔ရင္ေတာ႔ ဘာတစ္ခုမွ လုပ္ေပးမွာ မဟုတ္ပါဘူး။ အခ်ိဳ႕အေျခအေနေတြမွာ မွန္ရင္ တစ္ခုခုလုပ္ၿပီး မွားသြားရင္ေတာ႔ တျခားတစ္ခု လုပ္ေဆာင္ဖို႔ လိုအပ္တာတတ္ပါတယ္။ အဲဒီလို အေျခအေနမ်ိဳးအတြက္ေတာ႔ if…else statement ကို အသံုးျပဳရမွာ ျဖစ္ပါတယ္။ ေအာက္မွာျပထားသလိုပါပဲ if statement ေနာက္မွာ statement တစ္ခု ဒါမွမဟုတ္ တြန္႔ကြင္းနဲ႔ခတ္ထားတဲ႔ multiple statements ေတြ ရွိပါတယ္။ အဲဒီေနာက္မွာ else ကို ေရးၿပီး သူ႔ေနာက္ကလဲ statement တစ္ခု ဒါမွမဟုတ္ တြန္႔ကြင္းနဲ႔ခတ္ထားတဲ႔ multiple statements ေတြ ထည္႔ေရးေပးရမွာ ျဖစ္ပါတယ္။ if ေနာက္က test expression မွန္ခဲ႔မယ္ ဆိုရင္ အဲဒီေနာက္က statement(s) ေတြ အလုပ္လုပ္မွာ ျဖစ္ပါတယ္။ မွားခဲ႔ရင္ေတာ႔ else ေနာက္က statement(s) ေတြ အလုပ္လုပ္မွာ ျဖစ္ပါတယ္။



အထက္က if example ေလးကို if…else နဲ႔ ျပန္ေရးျပထားပါတယ္။

// ifelse.cpp

// demonstrates IF...ELSE statememt

#include <iostream>

using namespace std;

int main()

{

int x;

cout << “\nEnter a number: “;

cin >> x;

if( x > 100 )

cout << “That number is greater than 100\n”;

else

cout << “That number is not greater than 100\n”;

return 0;

}

ဒီဥပမာေလးမွာ user ထည္႔သြင္းေပးလိုက္တဲ႔ x တန္ဖိုးဟာ 100 ထက္ႀကီးခဲ႔ရင္ cout<< “That number is greater than 100\n”; ဆိုတာကို လုပ္ေဆာင္မွာ ျဖစ္ၿပီး မွားခဲ႔ရင္ေတာ႔ cout << “That number is not greater than 100\n”; ဆိုတာကို လုပ္ေဆာင္ေပးမွာပါ။ နမူနာအေနနဲ႔ 300 နဲ႔ 3 တန္ဖိုးႏွစ္ခု ထည္႔သြင္းေပးလိုက္လို႔ ရလာတဲ႔ ရလာဒ္ေတြကို ေအာက္မွာ ေလ႔လာ ၾကည္႔ႏိုင္ပါတယ္။

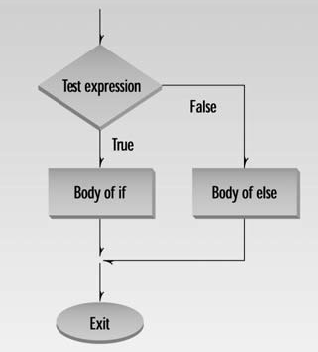
Enter a number: 300

That number is greater than 100

Enter a number: 3

That number is not greater than 100

if…else အလုပ္လုပ္ပံုကို flow chat နဲ႔ ေအာက္ပါအတိုင္း ေလ႔လာႏိုင္ပါတယ္။



**The getche() Library Function**

ေနာက္ထပ္ဥပမာ တစ္ခုအေနနဲ႔ if…else ကို while loop ထဲမွာ ထည္႔သြင္း အသံုးျပဳျပထားပါတယ္။ ေနာက္တစ္ခုက getche() ဆိုတဲ႔ library function အသစ္တစ္ခုကို သံုးျပထားပါတယ္။ အဲဒီ chcount.cpp ပရိုဂရမ္ေလးဟာ user ထည္႔သြင္းေပးလိုက္တဲ႔ စာလံုးေတြနဲ႔ အကၡရာ အေရအတြက္ေတြကို ေရတြက္ေပးမွာပါ။

// chcount.cpp

// counts characters and words typed in

#include <iostream>

using namespace std;

#include <conio.h> //for getche()

int main()

{

int chcount=0; //counts non-space characters

int wdcount=1; //counts spaces between words

char ch = ‘a’; //ensure it isn’t ‘\r’

cout << “Enter a phrase: “;

while( ch != ‘\r’ ) //loop until Enter typed

{

ch = getche(); //read one character

if( ch==’ ‘ ) //if it’s a space

wdcount++; //count a word

else //otherwise,

chcount++; //count a character

} //display results

cout << “\nWords=” << wdcount << endl

<< “Letters=” << (chcount-1) << endl;

return 0;

}

input ထည္႔သြင္းဖို႔အတြက္ cin နဲ႔ >> ကို အသံုးျပဳေလ႔ ရွိပါတယ္။ ဒီနည္းကို သံုးမယ္ဆိုရင္ user က input တစ္ခု ထည္႔သြင္းေပးလိုက္တိုင္း Enter ေခါက္ရမွာျဖစ္ပါတယ္။ ဒါေပမယ္႔ အခုလို ဥပမာ မ်ိဳးမွာ user က Enter မႏွိပ္ရဘဲ အကၡရာတိုင္းကို ဖတ္သြားဖုိ႔ လိုလာပါၿပီ။ ဒါေၾကာင္႔ getche() ဆိုတဲ႔ library function ကို အသံုးျပဳရမွာ ျဖစ္ပါတယ္။ ၄င္းကို အသံုးျပဳဖို႔ conio.h header file ကို ေၾကျငာေပးရမွာ ျဖစ္ပါတယ္။ input argument ထည္႔ေပးဖို႔ မလိုအပ္ဘဲ user ရိုက္ထည္႔လိုက္သမွ် အကၡရာေတြကို ခ်က္ခ်င္း return ျပန္ေပးမွာ ျဖစ္ပါတယ္။ chcount.cpp ထဲမွာေတာ႔ getche() နဲ႔ ဖတ္လို႔ ရလာတဲ႔ တန္ဖိုးေတြကို ch ထဲ ထည္႔ေပးလိုက္မွာျဖစ္ပါတယ္။

ေနာက္တစ္ခုက getche() function ဟာ user ရိုက္ထည္႔လိုက္သမွ် အကၡရာေတြကို ကြန္ပ်ဴတာမ်က္ႏွာျပင္ေပၚကို echo လုပ္ေပးမွာ ျဖစ္ပါတယ္။ (\_getch() function ကေတာ႔ echo ျပန္လုပ္ေပးမွာ မဟုတ္ပါဘူး။ echo ျပန္လုပ္ေပးလို႔လဲ function name ရဲ႕ ေနာက္ဆံုး စကားလံုးမွာ e ထည္႔ထားၿပီး getche ျဖစ္ေနတာပါ။ if…else statement ကေတာ႔ စကားလံုး(word) ေရတြက္ဖို႔ သံုးထားတဲ႔ wdcount ကို space ဆိုတဲ႔ character ေတြ႔တာနဲ႔ တစ္တိုးေပးမွာ ျဖစ္ပါတယ္။ အကၡရာ (character) ေတြ ေရတြက္ဖို႔ သံုးတဲ႔ chcount ကေတာ႔ space ကလြဲခို႔ ဘယ္အကၡရာကိုပဲ ရိုက္ရိုက္ တစ္တိုးေပးမွာ ျဖစ္ပါတယ္။ ဆိုလိုတာက space ကလြဲလို႔ က်န္တဲ႔ အကၡရာေတြကို character လို႔ သတ္မွတ္လိုက္တာပါ။ (ဒီလို စစ္ေဆးတဲ႔ နည္းမွာ အားနည္းခ်က္ေတြ ရွိပါတယ္။ ဒါေပမယ္႔ ဒီေနရာမွာေတာ႔ ဒီဥပမာေလးဟာ လံုေလာက္တယ္လို႔ ယူဆပါတယ္)။ ေအာက္မွာ နမူနာထည္႔သြင္းေပးလိုက္တဲ႔ စာေၾကာင္းေလးကို စကားလံုးနဲ႔ အကၡရာေတြ ေရတြက္ပံု ေလ႔လာၾကည္႔ၾကရေအာင္။

For while and do

Words=4

Letters=13

while statement မွာပါတဲ႔ test expression ကေတာ႔ ch ရဲ႕တန္ဖိုးဟာ ‘\r’ ျဖစ္မျဖစ္ စစ္ေဆးပါတယ္။ တကယ္ေတာ႔ အဲဒီ အကၡရာဟာ ကီးဘုတ္က Enter key ကို ကိုယ္စားျပဳထားတာပါ။ တကယ္လို႔ Enter ေခါက္ခဲ႔ရင္ while loop ကေန ထြက္သြားမွာ ျဖစ္ပါတယ္။

**Assignment Expressions**

chcount.cpp ကို assignment expressions နဲ႔ precedence ေတြရဲ႕ အေရးပါပံုနဲ႔ ကုဒ္လိုင္းအခ်ိဳ႕ကို ခ်ံဳ႕ပစ္ႏိုင္တာကိုျပသဖို႔ အနည္းငယ္ ျပင္ေရးျပခ်င္ပါတယ္။ အဲဒီလို ျပင္ေရးလိုက္ျခင္းအားျဖင္႔ ကုဒ္ေတြကို ဖတ္ရတာ တစ္မ်ိဳး ျဖစ္ေနႏိုင္ေပမယ္႔ C နဲ႔ C++ မွာေတာ႔ ေရးရိုးေရးစဥ္ ေရးနည္းတစ္ခုပဲ ျဖစ္ပါတယ္။ ျပင္ေရးထားတဲ႔ chcnt2.cpp ကိုေလ႔လာၾကည္႔ၾကရေအာင္။

// chcnt2.cpp

// counts characters and words typed in

#include <iostream>

using namespace std;

#include <conio.h> // for getche()

int main()

{

int chcount=0;

int wdcount=1; // space between two words

char ch;

while( (ch=getche()) != ‘\r’ ) // loop until Enter typed

{

if( ch==’ ‘ ) // if it’s a space

wdcount++; // count a word

else // otherwise,

chcount++; // count a character

} // display results

cout << “\nWords=” << wdcount << endl

<< “Letters=” << chcount << endl;

return 0;

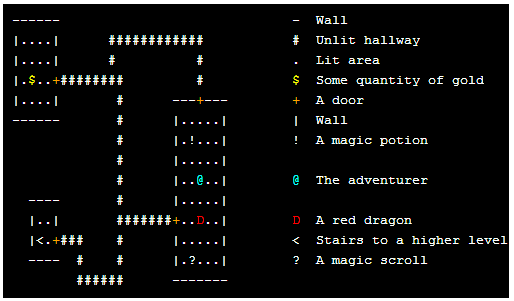
}

getche() ကေန return ျပန္ေပးလိက္တဲ႔ တန္ဖိုးကို ch ထဲ ထည္႔ေပးတဲ႔ ကုဒ္ကို အျပင္မွာ သပ္သပ္ မေရးသားေတာ႔ဘဲ while loop ရဲ႕ test expression ထဲမွာ တစ္ခါတည္း ထည္႔သြင္း ေရးသားလိုက္ျခင္းပဲ ျဖစ္ပါတယ္။ ေနာက္ထပ္ၿပီး အဲဒီ တန္ဖိုးကို ‘\r’ ဟုတ္မဟုတ္ စစ္ေဆးေပးပါတယ္။

တကယ္ေတာ႔ (ch=getche()) ဆိုတဲ႔ ကုဒ္ တစ္ခုလံုးက ch ထဲမွာ ရွိတဲ႔ တန္ဖိုးကို ကိုယ္စားျပဳေနတာ ျဖစ္တဲ႔အတြက္ တစ္ခါတည္း မွန္မမွန္ စစ္ေဆးလို႔ ရသြားတာပဲ ျဖစ္ပါတယ္။ C++ မွာ x = y = z = 0; ဆိုၿပီး ေရးသားႏိုင္ပါတယ္။ အဲဒီလို ေရးနည္းမွာ ညာဘက္အစြန္ဆံုး assignment က အရင္ စအလုပ္လုပ္္ၿပီး 0 ကို z ထဲ၊ z တန္ဖိုးကို y ထဲ၊ y တန္ဖိုးကို x ထဲထည္႔မွာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ တန္ဖိုးေတြအားလံုးက 0 ျဖစ္သြားပါလိမ္႔မယ္။ (ch=getche()) ဆိုတဲ႔ assignment expression ကို လက္သဲကြင္းနဲ႔ ခတ္ထားတာကို သတိထားမိပါလိမ္႔မယ္။ ဘာလို႔လဲဆိုေတာ႔ assignment operator ဟာ relational operator ထက္ precedence နိမ္႔တဲ႔အတြက္ အရင္အလုပ္လုပ္ေစခ်င္တဲ႔အခါ ကြင္းခတ္ေပးလိုက္ရတာပဲ ျဖစ္ပါတယ္။ တကယ္လို႔ လက္သဲကြင္းသာ မပါခဲ႔ဘူးဆိုရင္ while( ch = (getche() != ‘\r’) ) လို လုပ္ေဆာင္သြားမွာ ျဖစ္ၿပီး ကၽြန္ေတာ္တို႔ မေမွ်ာ္လင္႔ထားတဲ႔ အမွားေတြကို ရင္ဆိုင္ရမွာပါ။

အဲဒီ chcnt2.cpp ထဲမွာပါတဲ႔ while statement ဟာ ေနရာအနည္းငယ္ပဲ ယူေပမယ္႔ အသံုးတည္႔ပါတယ္။ တကယ္ေတာ႔ ၄င္းဟာ test expression တစ္ခုသာမကဘဲ keyboard က ရိုက္ထည္႔လိုက္တဲ႔ အကၡရာ တစ္လံုးကို ch ထဲကိုလည္း ထည္႔ေပးႏိုင္ခဲ႔ပါတယ္။ ဒါေပမယ္႔ ဒီလိုကုဒ္မ်ိဳးကို ပထမဆံုး ျမင္ဘူးကာစမွာေတာ႔ အဓိပၸါယ္ေဖာ္ဖို႔ ခက္ခဲႏိုင္ပါတယ္။

**Nested if...else Statements**

****

ပံု(၁၃-၁) Text-based game Roguelike

ကၽြန္ေတာ္တို႔ ငယ္ငယ္က Microsoft Windows ေတြ မေပၚေသးပါဘူး။ MS-DOS system ကိုပဲ သံုးၾကရပါတယ္။ အဲဒီေခတ္တုန္းက character-mode မွာ ကစားခဲ႔ရတဲ႔ text-based adventure စြန္႔စားခန္း ဂိမ္းေလးေတြကို အမွတ္ရမိပါေသးတယ္။ စိတ္ကူးယဥ္ ကမာၻတစ္ခုထဲမွာ ကၽြန္ေတာ္တို႔ character ေလးတစ္လံုးကို အရပ္ေလးမ်က္ႏွာ လွည္႔ပတ္သြားေစရင္းနဲ႔ ရဲတိုက္ေတြ၊ ေမွာ္ဆရာေတြ၊ ရတနာေတြ အစရွိသျဖင္႔ ရွာေဖြရတာမ်ိဳးပါ။ ဒါေပမယ္႔ အဲဒီဂိမ္းေတြထဲမွာ ရုပ္ပံုေတြ မပါေသးပါဘူး။ အဂၤလိပ္ အကၡရာေလးေတြ ကိုပဲ ပံု(၁၃-၁)မွာ ျပထားသလိုမ်ိဳး input နဲ႔ output အတြက္ သံုးခဲ႔ၾကရတာပါ။ ငယ္ရြယ္ေသးတဲ႔ စာဖတ္သူမ်ား အဖို႔ေတာ႔ အဲဒီလို ကြန္ပ်ဴတာ ေခတ္ဦးက ဂိမ္းေလးေတြကို ေတြ႔ဖူးဖို႔ မလြယ္ေတာ႔ပါဘူး။ ဒီေခတ္မွာေတာ႔ 3D graphics အေကာင္းစားေတြနဲ႔ ဂိမ္းအလန္းစားေတြ က အစားထိုး ေနရာယူ ထားလိုက္ၾကၿပီေလ။ ဒီသင္ခန္းစာမွာေတာ႔ Nested if…else statement အေၾကာင္းကို ရွင္းျပဖို႔အတြက္ အဲဒီလို စြန္႔စားခန္း ဂိမ္းေလးတစ္ခုရဲ႕ အေျခခံ အစိတ္အပိုင္းတစ္ခုကို တုပ ေရးသားျပထားပါတယ္။ adifelse.cpp ကို ေအာက္ပါအတိုင္း ေလ႔လာၾကည္႔ၾကပါစို႔-

// adifelse.cpp

// demonstrates IF...ELSE with adventure program

#include <iostream>

using namespace std;

#include <conio.h> //for getche()

int main()

{

char dir=’a’;

int x=10, y=10;

cout << “Type Enter to quit\n”;

while( dir != ‘\r’ ) //until Enter is typed

{

cout << “\nYour location is “ << x << “, “ << y;

cout << “\nPress direction key (n, s, e, w): “;

dir = getche(); //get character

if( dir==’n’) //go north

y--;

else

if( dir==’s’ ) //go south

y++;

else

if( dir==’e’ ) //go east

x++;

else

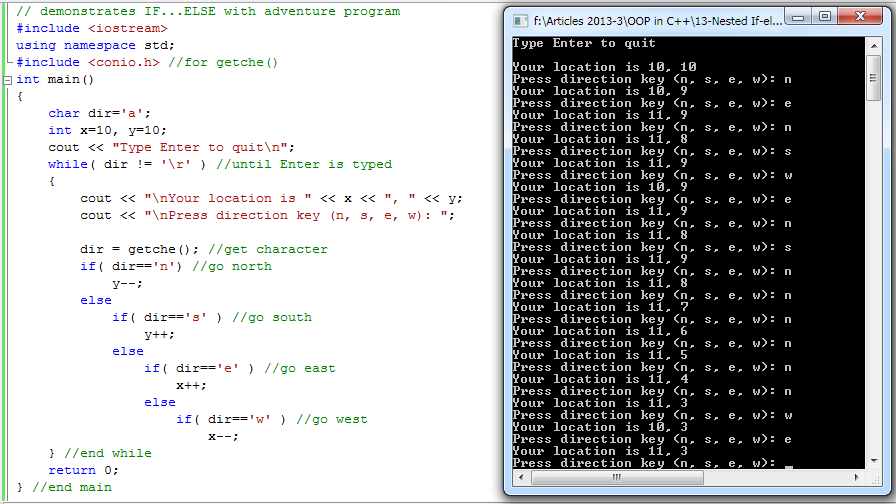
if( dir==’w’ ) //go west

x--;

} //end while

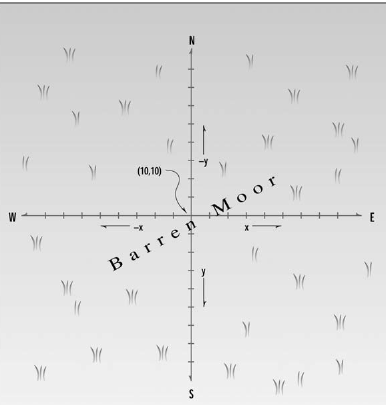
return 0;

} //end main



ပံု(၁၃-၁) Nested if…else အသံုးျပဳပံုကို ေရးသားစမ္းသပ္ျပထားပံု

ဒီဂိမ္းရဲ႕ အစမွာ ကၽြန္ေတာ္တို႔အေနနဲ႔ လြင္တီးေခါင္ တစ္ခုကို ေရာက္ေနသလို ခံစားရမွာ ျဖစ္ပါတယ္။ တစ္ႀကိမ္မွာ အရပ္ေလးမ်က္ႏွာထဲက ကိုယ္ႀကိဳက္တဲ႔ အရပ္ကို တစ္ယူနစ္ သြားလို႔ ရမွာ ျဖစ္ပါတယ္။ပရိုဂရမ္ကေတာ႔ ကၽြန္ေတာ္တို႔ ေရာက္ေနတဲ႔ ေနရာကို x,y coordinate နဲ႔ ျပေပးေနမွာပါ။ စစခ်င္း တည္ေနရာကေတာ႔ (10,10) ျဖစ္ပါတယ္။ ဒီပရိုဂရမ္ေလးမွာေတာ႔ ကၽြန္ေတာ္တို႔ ဘယ္ကိုပဲ သြားသြား ဘာမွ ေတြ႔ရဦးမွာ မဟုတ္ပါဘူး။ တည္ေနရာရဲ႕ x,y တန္ဖိုးေတြပဲ ေျပာင္းလဲ သြားမွာပါ။ ကၽြန္ေတာ္တို႔ အေနနဲ႔ ကန္႔သတ္ခ်က္ ေပးမထားတဲ႔အတြက္ ႀကိဳက္တဲ႔ အရပ္မ်က္ႏွာကို ႀကိဳက္သေလာက္ သြားခြင္႔ ရွိပါတယ္။ အထက္က ပံု (၁၃-၁) မွာ user ထည္႔ေပးလိုက္တဲ႔ အရပ္မ်က္ႏွာကို မူတည္ၿပီး xy coordinate ေတြ ေျပာင္းလဲသြားတာကို ျပထားပါတယ္။ ပရိုဂရမ္ကို အဆံုးသတ္ဖို႔႔ Enter key ကို ႏွိပ္လိုက္ရမွာ ျဖစ္ပါတယ္။ ဒီပရိုဂရမ္ေလးက တကယ္ေတာ႔ game တစ္ခုရဲ႕ အစိတ္အပိုင္း ေသးေသးေလး တစ္ခုပဲ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ multiple branches ေတြကို ကိုင္တြယ္ဖို႔ နည္းလမ္းတစ္ခုကို ေလ႔လာႏိုင္ပါတယ္။ if…else statement တစ္ခုထဲမွာ if…else statement တစ္ခု၊ အဲဒီထဲမွာ ေနာက္ထပ္ if…else statement တစ္ခု နဲ႔ ေနာက္ဆံုးမွာ if statement တစ္ခုကို အဆင္႔ဆင္႔ ထည္႔သြင္းေရးသားထားတာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ ပထမဆံုး test condition ဟာ မွားသြားခဲ႔မယ္ဆိုရင္ ဒုတိယ၊ တတိယ စသျဖင္႔ ေလးဆင္႔ စလံုး အဆင္႔ဆင္႔ စစ္ေဆး သြားမွာ ျဖစ္ပါတယ္။ တစ္ခုခု မွန္သြားခဲ႔မယ္ဆိုရင္ေတာ႔ သက္ဆိုင္ရာ x-y တန္ဖိုးမ်ားကို ေျပာင္းလဲေပးတဲ႔ ကုဒ္မ်ားကို လုပ္ေဆာင္ေပးမွာပါ။ အဲဒီလို nested group လုပ္ထားတဲ႔ if…else statements ေတြကို *decision tree* လို႔ ေခၚဆိုၾကပါတယ္။



**Matching the else**

Nested if…else ကို သံုးတဲ႔ေနရာမွာ မွားတတ္တဲ႔ ေနရာေလး တစ္ခု ရွိပါတယ္။ ဒါကေတာ႔ else ကို မွားယြင္းၿပီး မသက္ဆိုင္တဲ႔ if နဲ႔ တြဲသံုးမိတာပဲ ျဖစ္ပါတယ္။ ေအာက္က badelse.cpp ကို ေလ႔လာၾကည္႔ရေအာင္ -

// badelse.cpp

// demonstrates ELSE matched with wrong IF

#include <iostream>

using namespace std;

int main()

{

int a, b, c;

cout << “Enter three numbers, a, b, and c:\n”;

cin >> a >> b >> c;

if( a==b )

if( b==c )

cout << “a, b, and c are the same\n”;

else

cout << “a and b are different\n”;

return 0;

}

အထက္ပါ ပရိုဂရမ္ေလးမွာ cin ကိုသံုးၿပီး a,b နဲ႔ c တန္ဖိုးေတြကို ဖတ္ယူပါတယ္။ user က တန္ဖိုး သံုးခုကို ထည္႔ေပးၿပီး Enter ေခါက္ေပးရမွာပါ။ တကယ္လို႔ ကၽြန္ေတာ္တို႔က 2,3 နဲ႔ 3 ကို ထည္႔ေပးလိုက္တယ္ ဆိုၾကပါစို႔။ ဒါဆို a က 2 b က 3 နဲ႔ c က 3 အသီးသီး ျဖစ္သြားပါၿပီ။ a နဲ႔ b ဟာ မတူတဲ႔ အတြက္ ပထမ if ရဲ႕ test expression ဟာ မွားသြားပါတယ္။ ဒါေၾကာင္႔ ကၽြန္ေတာ္တို႔ အေနနဲ႔ else ကို ေခၚယူၿပီး cout << “a and b are different\n”; ဆိုတဲ႔ ကုဒ္ကို လုပ္ေဆာင္ဖို႔ ေမွ်ာ္လင္႔ထားမိမွာပါ။ ဒါေပမယ္႔ လက္ေတြ႔မွာ ဘာမွ print မထုတ္ေပးပါဘူး။ ဘာလို႔လဲဆိုေတာ႔ အဲဒီ else ကို မသက္ဆိုင္တဲ႔ if နဲ႔ မွားၿပီး တြဲထားခဲ႔မိလို႔ပါပဲ။ ရုတ္တရက္ ၾကည္႔လိုက္ရင္ else က ပထမ if နဲ႔ တြဲထားတယ္လို႔ ထင္စရာ ျဖစ္ေနပါတယ္။ တကယ္က indentation ေနရာခ်ထားမႈအရ အဲလို အထင္မွားေစတာပါ။ တကယ္က အဲဒီ else ဟာ ဒုတိယ if နဲ႔ တြဲေနတာပါ။ ဥပေဒသ ကေတာ႔ else ဟာ ေနာက္ဆံုး if မွာ သာ else မရွိခဲ႔ရင္ ၄င္းနဲ႔ တြဲေနတာ ျဖစ္ပါတယ္။ ေအာက္မွာ indentation နဲ႔ print လုပ္မယ္႔ စာသား ျပင္ထားတဲ႔ ပံုစံ အမွန္ကို ေရးျပထားပါတယ္။

if(a==b)

if(b==c)

cout << “a, b, and c are the same\n”;

else

cout << “b and c are different\n”;

ဒါေၾကာင္႔ 2,3,3 ကို ထည္႔ေပးလိုက္မယ္ ဆိုရင္ ဘာမွ print မထုတ္ေပးေပမယ္႔ 2,2,3 ကို ထည္႔ေပးလိုက္ရင္ေတာ႔ b and c are different ဆိုတာကို print ထုတ္ေပးမွာ ျဖစ္ပါတယ္။ ကၽြန္ေတာ္တို႔ အေနနဲ႔ else ကို ပထမဆံုး if နဲ႔ မျဖစ္မေန တြဲေပးခ်င္ရင္ေတာ႔ ဒုတိယ if ကို ေအာက္ပါအတိုင္း တြန္႔ကြင္းထဲ ထည္႔ေပးလိုက္ရမွာ ျဖစ္ပါတယ္။

if(a==b)

{

if(b==c)

cout << “a, b, and c are the same”;

}

else

cout << “a and b are different”;

ဒီနည္းနဲ႔ else ဟာ ပထမ if နဲ႔ တြဲသြားပါတယ္။ တြန္႔ကြင္း ခတ္ခံထားရတဲ႔ if ကို အျပင္က else က မျမင္ႏိုင္လို႔ပဲ ျဖစ္ပါတယ္။

**The else...if Construction**

if…else ေတြကို nested လုပ္တဲ႔အခါမွာ နားလည္ရ ခက္ေလ႔ရွိပါတယ္။ အထူးသျဖင္႔ ရႈပ္ရႈပ္ေထြးေထြး အဆင္႔အမ်ားႀကီး nested လုပ္ခဲ႔မိရင္ ဖတ္တဲ႔သူအေနနဲ႔ ဘာသာျပန္ဖို႔ ခက္လာပါတယ္။ ဒီျပႆနာကို ေျဖရွင္းဖို႔ ေနာက္တစ္နည္း သံုးႏိုင္ပါတယ္။ ေအာက္ပါ adelseif.cpp ပရိုဂရမ္ေလးကို ေလ႔လာၾကည္႔ၾကရေအာင္ -

// adelseif.cpp

// demonstrates ELSE...IF with adventure program

#include <iostream>

using namespace std;

#include <conio.h> //for getche()

int main()

{

char dir=’a’;

int x=10, y=10;

cout << “Type Enter to quit\n”;

while( dir != ‘\r’ ) //until Enter is typed

{

cout << “\nYour location is “ << x << “, “ << y;

cout << “\nPress direction key (n, s, e, w): “;

dir = getche(); //get character

if( dir==’n’) //go north

y--;

else if( dir==’s’ ) //go south

y++;

else if( dir==’e’ ) //go east

x++;

else if( dir==’w’ ) //go west

x--;

} //end while

return 0;

} //end main

တကယ္ေတာ႔ အထက္ပါ ကုဒ္ေလးေတြရဲ႕ ရလာဒ္ဟာ adifelse.cpp နဲ႔ အတူတူပါပဲ။ if ေတြကို else ေတြ ေနာက္ကို ေနရာေရႊ႕ေပးလိုက္တာပဲ ကြာပါတယ္။ ရုတ္တရက္ ၾကည္႔လိုက္ရင္ else if ဆိုတဲ႔ keyword အသစ္တစ္ခု သံုးထားတယ္လို႔ေတာင္ ထင္ရပါတယ္။ ဒီလိုေရးသားနည္းဟာ if…else ထက္ ဖတ္ရတာေရာ ေရးသားရတာပါ သိသိသာသာ လြယ္ကူေစပါတယ္။ ဒါေၾကာင္႔ decision tree ပံုစံမ်ိဳး ေရးသားစရာ ၾကံဳလာမယ္ဆိုရင္ if…else ပံုစံထက္ else if ကို အသံုးျပဳသင္႔ေၾကာင္း အၾကံျပဳလိုပါတယ္။

**The switch Statement**

ကၽြန္ေတာ္တို႔အေနနဲ႔ decision tree အႀကီးစားတစ္ခုကို ေျဖရွင္းဖို႔ လိုအပ္လာတဲ႔ အခါမ်ိဳးမွာ၊ ေနာက္ၿပီး အဲဒီလို ဆံုးျဖတ္ဖို႔ကို variable တစ္ခုတည္းရဲ႕ တန္ဖိုးေပၚ မူတည္ေနမယ္ ဆိုရင္ေတာ႔ if…else ဒါမွမဟုတ္ else if ေတြ အဆင္႔ဆင္႔ ေရးမယ္႔ အစား switch statement ကို အသံုးျပဳသင္႔ပါတယ္။

// platters.cpp

// demonstrates SWITCH statement

#include <iostream>

using namespace std;

int main()

{

int speed; //turntable speed

cout << “\nEnter 33, 45, or 78: “;

cin >> speed; //user enters speed

switch(speed) //selection based on speed

{

case 33: //user entered 33

cout << “LP album\n”;

break;

case 45: //user entered 45

cout << “Single selection\n”;

break;

case 78: //user entered 78

cout << “Obsolete format\n”;

break;

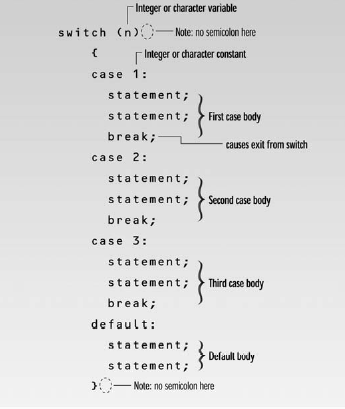
}

return 0;

}

ဒီပရိုဂရမ္ေလးဟာ တကယ္ေတာ႔ ဓာတ္ျပားေဟာင္းႀကီးမ်ားကို ဖြင္႔တဲ႔ ဓာတ္စက္ေတြရဲ႕ လည္ပတ္မႈ အျမန္ႏႈန္းကို ေရြးခ်ယ္ဖို႔ ေရးထားတာ ျဖစ္ပါတယ္။ user ထည္႔ေပးလိုက္ႏိုင္တဲ႔ 33, 45, နဲ႔ 78 ဆိုတဲ႔ ျဖစ္ႏိုင္ေျခသံုးခုအတြက္ သက္ဆိုင္ရာ messages ေတြကို print ထုတ္ေပးမွာ ျဖစ္ပါတယ္။ အဲဒီေခတ္က ဓာတ္ျပားေတြမွာ long-playing records (LPs) ေတြမွာ သီခ်င္းေတြ အမ်ားႀကီး ပါဝင္ၿပီး 33rpm (တစ္မိနစ္ကို ၃၃ပတ္ႏႈန္း) နဲ႔ လည္ပါတယ္၊ သီခ်င္းတစ္ပုဒ္စီသာပါဝင္တဲ႔ single song ေတြကိုေတာ႔ 45rpm နဲ႔ လည္ပါတယ္။ 78 ကေတာ႔ သူတို႔ LPs နဲ႔ single တို႔ထက္ ေရွးက်တဲ႔ စနစ္ေဟာင္း လည္ပတ္ႏႈန္းပဲ ျဖစ္ပါတယ္။

switch ဆိုတဲ႔ keyword ေနာက္က လက္သဲကြင္း အဖြင္႔အပိတ္ထဲမွာ variable ကို ထည္႔ေပးရမွာ ျဖစ္ပါတယ္ ( ဥပမာ- **switch(speed)** )။ တြန္႔ကြင္း အဖြင္႔အပိတ္ထဲမွာေတာ႔ case statements ေတြကို ေရးသားရမွာပါ။ case တစ္ခုစီရဲ႕ ေနာက္မွာ constant တစ္ခု ပါဝင္ရမွာ ျဖစ္ပါတယ္။ အဲဒီ constant ကို ကြင္းမခတ္ထားဘဲ ေနာက္က colon(**:**) နဲ႔ ပိတ္ေပးရမွာပါ (ဥပမာ - **case 33:**) ။ case constant မွာ အသံုးျပဳထားတဲ႔ constants ေတြဟာ switch variable နဲ႔ အမ်ိဳးအစားခ်င္း တူညီဖို႔ေတာ႔ လိုအပ္မွာ ျဖစ္ပါတယ္။ ေအာက္ကပံုမွာ switch statement ရဲ႕ syntex ကို ျပသထားပါတယ္။



switch မတိုင္ခင္မွာ switch variable ကို တန္ဖိုးတစ္ခု သတ္မွတ္ထားေပးဖို႔ လိုပါတယ္။ အဲဒီတန္ဖိုးကို case statement ေတြမွာ ရွိတဲ႔ constant တန္ဖိုးေတြနဲ႔ တူရာကို ရွာရမွာပါ။ ရည္ရြယ္ထားတဲ႔ case ကိုေတြ႔ၿပီဆိုရင္ အဲဒီေနာက္က statements ေတြကို break မေတြ႔မခ်င္း လုပ္ေဆာင္ေပးမွာ ျဖစ္ပါတယ္။ ေအာက္မွာ platter.cpp ကို run ျပထားပါတယ္။ ေလ႔လာၾကည္႔ၾကရေအာင္ -

Enter 33, 45, or 78: 45

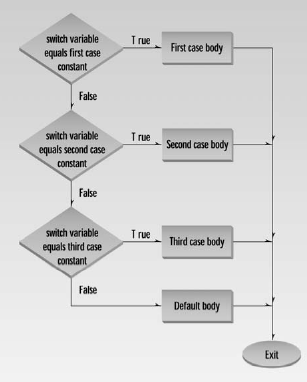
Single selection

**The break Statement**

**platters.cpp** ရဲ႕ case section အားလံုးရဲ႕ အဆံုးမွာ break statement ေရးသားထားတာကို သတိထားမိမွာပါ။ တကယ္ေတာ႔ break keyword ဟာ switch statement တစ္ခုလံုးကေန ထြက္သြားေစမွာပါ။ ပရိုဂရမ္ရဲ႕ control ဟာ break ကိုေတြ႔ရင္ switch ရဲ႕ အျပင္ဘက္ကို ထြက္သြားၿပီး သူ႔ေနာက္က statement ကို ဆက္လက္ လုပ္ေဆာင္မွာ ျဖစ္ပါတယ္။

break ကိုေရးဖို႔ မေမ႔သင္႔ပါဘူး။ break မပါခဲ႔ရင္ ေနာက္ထပ္ case ေတြကို ဆက္လက္ စစ္ေဆးေနဦးမွာပါ။ break statement ကို loop ေတြထဲက ထြက္ခ်င္တဲ႔ အခါမ်ိဳးမွာလဲ အသံုးျပဳႏိုင္ပါတယ္။ ေနာက္ပိုင္းၾကံဳလာတဲ႔ အခ်ိန္မွာ ဆက္လက္ ရွင္းျပသြားပါ႔မယ္။

တကယ္လို႔ switch variable ဟာ case constants ေတြနဲ႔ တစ္ခုမွ မတူညီခဲ႔ဘူးဆိုရင္ ဘာမွ မလုပ္ေဆာင္ဘဲ switch ထဲက ထြက္သြားမွာ ျဖစ္ပါတယ္။



**switch Statement with Character Variables**

platters.cpp မွာ အသံုးျပဳခဲ႔တဲ႔ switch variable ဟာ int data type ျဖစ္ပါတယ္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ char data type ကိုလည္း အသံုးျပဳႏိုင္ပါေသးတယ္။ ယခင္ သင္ခန္းစာက adelseif.cpp ကို ျပန္ျပင္ေရးထားတဲ႔ adswitch.cpp ကို ေလ႔လာၾကည္႔ၾကရေအာင္ -

// adswitch.cpp

// demonstrates SWITCH with adventure program

#include <iostream>

using namespace std;

#include <conio.h> //for getche()

int main()

{

char dir=’a’;

int x=10, y=10;

while( dir != ‘\r’ )

{

cout << “\nYour location is “ << x << “, “ << y;

cout << “\nEnter direction (n, s, e, w): “;

dir = getche(); //get character

switch(dir) //switch on it

{

case ‘n’: y--; break; //go north

case ‘s’: y++; break; //go south

case ‘e’: x++; break; //go east

case ‘w’: x--; break; //go west

case ‘\r’: cout << “Exiting\n”; break; //Enter key

default: cout << “Try again\n”; //unknown char

} //end switch

} //end while

return 0;

} //end main

ဒီပရိုဂရမ္ထဲမွာ switch variable အေနနဲ႔ dir ဆိုတဲ႔ character variable ကို အသံုးျပဳထားတာ ျဖစ္ပါတယ္။ char constant ေတြ အေနနဲ႔ကေတာ႔ 'n', 's', 'e', 'w' နဲ႔ '\r' တို႔ကို အသံုးျပဳထားပါတယ္။ ဒီေနရာမွာ '\r' က Enter ကို ကိုယ္စားျပဳထားတဲ႔ character ျဖစ္ပါတယ္။ (တစ္ခု မွတ္သားထားဖို႔က ကၽြန္ေတာ္တို႔အေနနဲ႔ integer နဲ႔ character variable ေတြကို switch variable အေနနဲ႔ အသံုးျပဳလို႔ ရေပမယ္႔ floating-point number ေတြကိုေတာ႔ အသံုးမျပဳႏိုင္ဘူးဆိုတာပါပဲ)။

ဒီပရိုဂရမ္မွာ ကုဒ္ေတြက တိုေတာင္းတာ ျဖစ္တဲ႔အတြက္ case ေနာက္က statements ေတြကို တစ္ေၾကာင္းတည္းမွာပဲ တန္းစီ ေရးလိုက္ျခင္းျဖင္႔ ပိုမို က်စ္လစ္သြားေစပါတယ္။ '\r' ကို သံုးၿပီး Enter ေခါက္ရင္ exit message ေပၚလာေအာင္ ေရးသားထားတာကို ေလ႔လာႏိုင္ပါတယ္။

**The default Keyword**

adswitch.cpp မွာ switch ရဲ႕ case ေတြ အဆံုးမွာ default ဆိုတဲ႔ keyword ပါေနတာကို သတိထားမိမွာပါ။ ၄င္းဟာ switch variable နဲ႔ case constant ေတြ တစ္ခုမွ မကိုက္ညီတဲ႔ အခါမ်ိဳးမွာ တစ္ခုခု လုပ္ေဆာင္ႏိုင္ဖို႔ ထည္႔ထားျခင္း ျဖစ္ပါတယ္။ ဒီ ဥပမာမွာေတာ႔ user က အျခား character တစ္ခုခုကို ထည္႔သြင္းခဲ႔ရင္ Try again! ဆိုတာကို print ထုတ္ေပးျခင္းအားျဖင္႔ ျပင္ဆင္ ထည္႔သြင္းခြင္႔ ေပးထားပါတယ္။ default ဟာ switch ရဲ႕ ေနာက္ဆံုး statement ျဖစ္တာေၾကာင္႔ break ကို ထည္႔ေပးဖို႔ မလိုေတာ႔ပါဘူး။

switch statement ကို user ထည္႔ေပးလိုက္တဲ႔ input ကို စမ္းစစ္ဖို႔ ေယဘုယ် သံုးတတ္ၾကပါတယ္။ အေျခအေနအမ်ားႀကီးထဲက တစ္ခုကို ေရြးခိုင္းတာပဲ ျဖစ္ပါတယ္။ DOS ေခတ္ေတြတုန္းက menu ေရးဖို႔ do-while နဲ႔ switch ကို တြဲသံုးေလ႔ ရွိခဲ႔ပါတယ္။ default statement ကိုေတာ႔ ကၽြန္ေတာ္တို႔ အေနနဲ႔ မလိုအပ္ဘူးလို႔ ထင္ေနရင္ေတာင္မွ မျဖစ္မေန ထည္႔ထားသင္႔ပါတယ္။ default: cout << “Error: incorrect input to switch”; break; ဆိုတဲ႔ ကုဒ္တစ္ေၾကာင္းက ပရိုဂရမ္မာ သို႔မဟုတ္ user ကို input မွားယြင္း ထည္႔ေပးေနၿပီ ဒါမွမဟုတ္ တစ္ခုခုေတာ႔ အမွားအယြင္း ရွိေနၿပီ ဆိုတာကို သတိေပးႏိုင္ပါတယ္။ ပရိုဂရမ္ အေသးေတြမွာ သိပ္မသိသာေပမယ္႔ အေရးႀကီးတဲ႔ ပရိုဂရမ္တိုင္းမွာ အမွားေတြကို တတ္ႏိုင္သမွ် ကုဒ္ေတြနဲ႔ ေထာင္ေခ်ာက္ဆင္ၿပီး ေဖာ္ထုတ္တတ္ဖို႔ လိုအပ္ပါတယ္။

**switch Versus if...else**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ switch နဲ႔ if...else (ဒါမွမဟုတ္ else if) ေတြကို ဘယ္ေနရာမွာ ဘာကို သံုးရမယ္ဆိုတာကို တိတိက်က် ဆံုးျဖတ္ရ ခက္ေနတတ္ပါတယ္။ တကယ္ေတာ႔ else if ကို variable အမ်ိဳးမ်ိဳးနဲ႔ ရႈပ္ရႈပ္ေထြးေထြး သံုးထားတဲ႔ အခါမ်ိဳးမွာ မျဖစ္မေန သံုးသင္႔ပါတယ္။ ဥပမာ -

if( SteamPressure\*Factor > 56 )

// statements

else if( VoltageIn + VoltageOut < 23000)

// statements

else if( day==Thursday )

// statements

else

// statements

switch statement မွာေတာ႔ variable တစ္ခုတည္းကိုပဲ စစ္ေဆးၿပီး ဆံုးျဖတ္တာ ျဖစ္ပါတယ္။ switch မွာေတာ႔ case a<3: // do something break; ဆိုၿပီး ေရးလို႔ မရပါဘူး။ 'a', 3 စတဲ႔ character ဒါမွမဟုတ္ integer constant နဲ႔ constant တန္ဖိုး ထြက္လာမယ္႔ 'a'+3 လို expression မ်ိဳးကိုပဲ သံုးရမွာ ျဖစ္ပါတယ္။

အဲဒီလို အေျခအေနမ်ိဳးနဲ႔ ကိုက္ညီတဲ႔အခ်ိန္မွာ swith statement ဟာ ေရးသားရတာ အင္မတန္ ရွင္းလင္း၊ လြယ္ကူၿပီး ဖတ္ရသူကိုလည္း နားလည္ရ လြယ္ေစပါတယ္။ ဒါေၾကာင္႔ decision tree မ်ားလာတာနဲ႔အမွ် switch ကို အတတ္ႏိုင္ဆံုး အသံုးျပဳေစခ်င္ပါတယ္ ခင္ဗ်ာ။

**The Conditional Operator**

Conditional Operator ဟာ အနည္းငယ္ ထူးဆန္းတဲ႔ decision operator တစ္ခုျဖစ္ပါတယ္။ ပရိုဂရမ္ ေရးသားရာမွာ ေတြ႕ၾကံဳရေလ႔ ရွိတဲ႔ - အေျခအေန တစ္ခု မွန္ရင္ variable တစ္ခုထဲ တန္ဖိုး တစ္ခု သတ္မွတ္ေပးၿပီး မွားသြားခဲ႔ရင္ အျခားတန္ဖိုးတစ္ခု သတ္မွတ္ေပးရေလ႔ ရွိတဲ႔ အခါမ်ိဳးမွာ အသံုးျပဳဖို႔အတြက္ ဖန္တီးထားတာပါ။ ဥပမာအားျဖင္႔ ငယ္တဲ႔ တန္ဖိုးရွာတဲ႔ ျပႆနာမ်ိဳးပါ။ အယ္ဖာ နဲ႔ ဘီတာမွာ အယ္ဖာက ငယ္ရင္ min ထဲကို အယ္ဖာ တန္ဖိုး ထည္႔ေပးၿပီး မွားခဲ႔ရင္ (ဘီတာက ငယ္ခဲ႔ရင္) min ထဲကို ဘီတာ တန္ဖိုး ထည္႔ေပးတဲ႔ အေျခအေနမ်ိဳး ျဖစ္ပါတယ္။ အဲဒီ အေျခအေနကို if...else သံုးၿပီး ေအာက္ပါအတိုင္း ေရးလို႔ ရပါတယ္။

if( alpha < beta )

min = alpha;

else

min = beta;

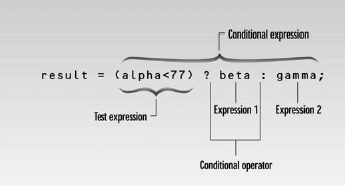
ဒီလို အေျခအေနမ်ိဳးကို မၾကာခဏ ေတြ႕ၾကံဳရတာေၾကာင္႔ C နဲ႔ C++ ကို တီထြင္ခဲ႔သူက ပိုမိုတိုေတာင္းတဲ႔ ေရးသားနည္းကို တီထြင္ရင္း conditional operator ေပၚေပါက္လာရျခင္း ျဖစ္ပါတယ္။ ဒီ operator မွာ သေကၤတ ႏွစ္ခု ပါၿပီး operand သံုးခု ပါဝင္ပါတယ္။ C++ မွာရွိတဲ႔ အျခား operator မ်ား အားလံုးဟာ operand တစ္ခု သို႔မဟုတ္ ႏွစ္ခုကို လုပ္ေဆာင္ေပးပါတယ္။ Conditional operator ကေတာ႔ တစ္ခုတည္းေသာ operand သံုးခုကို လုပ္ေဆာင္ေပးတဲ႔ operator ျဖစ္ပါတယ္။

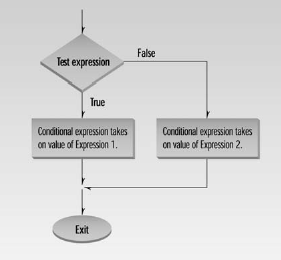
အထက္က ကုဒ္ကို conditional operator သံုးၿပီး ေရးမယ္ဆိုရင္ ေအာက္ပါအတိုင္း ရပါတယ္-

min = (alpha<beta) ? alpha : beta;

အဲဒီ statement ထဲက equal sign ညာဘက္က အျခမ္းျဖစ္တဲ႔ (alpha<beta) ? alpha : beta ဆိုတာ conditional expression ပဲ ျဖစ္ပါတယ္။ အဲဒီထဲမွာ ပါတဲ႔ question mark (?) နဲ႔ colon (:) တို႔ေပါင္းၿပီး conditional operator ျဖစ္လာတာပါ။

Question mark ေရွ႕က (alpha<beta) ဆိုတာကေတာ႔ test expression ပဲျဖစ္ပါတယ္။ အဲဒီ တစ္ခုအပါအဝင္ alpha နဲ႔ beta သံုးခုလံုးဟာ operand ေတြ ျဖစ္ၾကပါတယ္။ တကယ္လို႔ test expression မွန္ခဲ႔မယ္ဆိုရင္ conditional expression တစ္ခုလံုးရဲ႕ တန္ဖိုးဟာ question mark ေနာက္က operand ျဖစ္သြားမွာပါ။ ဒီဥပမာမွာေတာ႔ alpha ျဖစ္ပါတယ္။ Test expression မွားသြားခဲ႔ရင္ေတာ႔ colon ေနာက္က operand တန္ဖိုးျဖစ္သြားမွာပါ။ ဒီဥပမာမွာ beta ျဖစ္ပါတယ္။ Test expression မွာပါတဲ႔ လက္သဲကြင္း အဖြင္႔အပိတ္ကို မထည္႔လဲရပါတယ္။ ဒါေပမယ္႔ အစဥ္အလာ ေရးခဲ႔ၾကတာ ျဖစ္ၿပီး ဖတ္ရႈရ ပိုမိုလြယ္ကူေစပါတယ္။ ေအာက္က ပံုေတြမွာ conditional operator ရဲ႕ syntax နဲ႔ flow chat ကို ျပထားပါတယ္။





Conditional expression က ရလာတဲ႔ တန္ဖိုးကို variable တစ္ခုထဲ ထည္႔သြင္းဖို႔ပဲ ျဖစ္ျဖစ္ ဒါမွမဟုတ္ value တစ္ခု အသံုးျပဳႏိုင္တဲ႔ ေနရာတိုင္း သံုးႏိုင္ပါတယ္။ အထက္ပါ ဥပမာမွာေတာ႔ min ဆိုတဲ႔ variable ထဲကို ထည္႔သြင္းေပးလိုက္တာပါ။ ေနာက္ ဥပမာ တစ္ခုအေနနဲ႔ conditional operator ကိုသံုးၿပီး variable n ရဲ႕ absolute value ကို ရွာတဲ႔ ကုဒ္ေလး ေရးျပပါ႔မယ္။ (ဂဏန္း တစ္ခုရဲ႕ Absolute value ဆိုတာက အဲဒီ ဂဏန္းရဲ႕ အႏုတ္လကၡဏာကို ဖယ္ရွားထားတာပါ။ ဒါေၾကာင္႔ အျမဲတမ္း positive ျဖစ္ေနမွာပါ)

**absvalue = n<0 ? -n : n;**

တကယ္လို႔ n တန္ဖိုးက သုညထက္ ငယ္ခဲ႔ရင္ ? ေနာက္က -n ကို လုပ္ေဆာင္ၿပီး အႏုတ္ႏွစ္ခု ေတြ႕တဲ႔ အတြက္ အေပါင္းတန္ဖိုး ျဖစ္ေစပါမယ္။ တကယ္လို႔ မငယ္ခဲ႔ရင္ အေပါင္းတန္ဖိုး ျဖစ္ေနၿပီးသားျဖစ္လို႔ ေျပာင္းလဲမႈ မရွိဘဲ n ကိုပဲ assign လုပ္ေပးမွာ ျဖစ္ပါတယ္။

// condi.cpp

// prints ‘x’ every 8 columns

// demonstrates conditional operator

#include <iostream>

#include <conio.h>

using namespace std;

int main()

{

for(int j=0; j<80; j++) //for every column,

{ //ch is ‘x’ if column is

char ch = (j%8) ? ' ' : 'x'; //multiple of 8, and

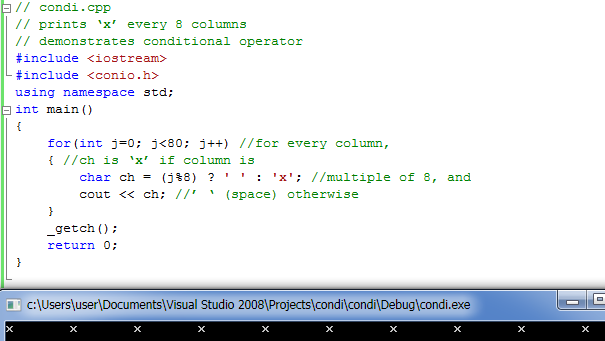
cout << ch; //’ ‘ (space) otherwise

}

\_getch();

return 0;

}



အထက္ပါပံုမွာ space ၇ ခုၿပီးတိုင္း x တစ္ခု ပရင္႔ထုတ္ျပသည္႔ ရလာဒ္ပင္ ျဖစ္သည္။ condi.cpp တြင္ j တန္ဖိုးကို 0 မွ 79 အထိ တစ္ခုခ်င္း for loop ျဖင္႔ တိုးယူလာရင္း conditional operator ကို လုပ္ေဆာင္ပါသည္။ ၄င္းတြင္ ပါဝင္ေသာ test expression (j % 8) က j တန္ဖိုးသည္ ၈ ၏ ဆတိုးကိန္းမ်ား မျဖစ္မခ်င္း (၈ ႏွင္႔စား၍ မျပတ္မခ်င္း) non zero (1-7) ျဖစ္ေနေသာေၾကာင္႔ true ျဖစ္ကာ ? ေနာက္က space ' ' ကို ပရင္႔ထုတ္ေပးေနမည္ ျဖစ္သည္။ ၈ ႏွင္႔ စား၍ ျပတ္ေသာ တန္ဖိုးမ်ားတြင္ အၾကြင္းမွာ သုည ျဖစ္ေနသျဖင္႔ test expression မွာ false ျဖစ္သြားၿပီး x ကို ပရင္႔ထုတ္ေပးျခင္းျဖင္႔ အထက္ပါ ရလာဒ္ ထြက္လာရသည္။ အထက္ပါ ကုဒ္ကို တိုေတာင္းလွၿပီ မထင္ပါႏွင္႔ဦး။ ch ကိုေဖ်ာက္၍ ေအာက္ပါအတိုင္း တစ္ေၾကာင္းတည္း ေပါင္းေရးႏိုင္ပါေသးသည္။

**cout << ( (j%8) ? ‘ ‘ : ‘x’ );**

C နဲ႔ C++ ပရိုဂရမ္မာ အေတာ္မ်ားမ်ား ဒီလို က်စ္လစ္တဲ႔ ကုဒ္မ်ိဳးကို ႏွစ္ႏွစ္ၿခိဳက္ၿခိဳက္ေရးေလ႔ ရွိပါတယ္။ ကုဒ္တိုတိုေလးနဲ႔ ထိထိေရာက္ေရာက္ ေရးႏိုင္တာ ေပ်ာ္စရာေကာင္းပါတယ္။ ဒါေပမယ္႔ မလိုအပ္တဲ႔အခါမ်ိဳးမွာ အဲလို ကုဒ္မ်ိဳးေတြကို မျဖစ္မေန အခ်ိန္ကုန္ခံေရးေနတိုင္းလဲ အက်ိဳးမရွိ ျဖစ္တတ္ပါတယ္။ ဒါေၾကာင္႔ if...else သံုးၿပီး ကုဒ္နဲနဲ ပိုေရးမလား ဒါမွမဟုတ္ conditional operator ကို သံုးမလားဆိုတာကေတာ႔ ရရွိတဲ႔ အခ်ိန္အေပၚ မူတည္မယ္လို႔ ထင္ျမင္မိပါတယ္။

**Logical Operators**

ေရွ႕သင္ခန္းစာမ်ားမွာ (Conditional Operator မပါ) operator အုပ္စု ႏွစ္ခုကို ေလ႔လာခဲ႔ၾကၿပီးပါၿပီ။ ပထမတစ္ခုကေတာ႔ arithmetic operators (+, -, \*, /, နဲ႔ %) မ်ားျဖစ္ၾကၿပီး ဒုတိယ တစ္ခုကေတာ႔ relational operators (<, >, <=, >=, ==, နဲ႔ !=)မ်ားပဲ ျဖစ္ပါတယ္။

အခု ဆက္လက္ေလ႔လာၾကရမွာကေတာ႔ တတိယအုပ္စုျဖစ္တဲ႔ *logical operators* မ်ားပဲ ျဖစ္ပါတယ္။

အဲဒီ operator မ်ားဟာ (true နဲ႔ false တန္ဖိုး ႏွစ္ခုျဖစ္ႏိုင္တဲ႔ variables) Boolean variables ေတြကို logically ေပါင္းစပ္ေပးႏိုင္စြမ္း ရွိပါတယ္။ ဥပမာအားျဖင္႔ today is a weekday ဟာ မွားလဲ မွားႏိုင္သလဲ မွန္လဲ မွန္ႏိုင္တဲ႔ အတြက္ Boolean value ရွိပါတယ္။ ေနာက္ထပ္ Boolean expression တစ္ခုက Maria took the car ျဖစ္ပါတယ္။ အဲဒီ expression ႏွစ္ခုကို logically ေပါင္းစပ္လို႔ ရပါတယ္။ If today is a weekday, and Maria took the car, then I’ll have to take the bus. ဒီေနရာမွာ သံုးသြားတဲ႔ logical connection ကေတာ႔ and ပဲ ျဖစ္ပါတယ္။ စာေၾကာင္းႏွစ္ခုကို ဆက္စပ္ၿပီးမွ true or false value ထြက္လာေအာင္ ေဆာင္ရြက္ေပးလို႔ပဲ ျဖစ္ပါတယ္။ အခ်က္ႏွစ္ခုလံုးမွန္မွ ဘတ္စ္ကားနဲ႔ သြားရမွာ ျဖစ္ပါတယ္။

**Logical AND Operator**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ Logical AND Operator ကို အသံုးျပဳၿပီး adswitch.cpp ကို ပိုၿပီး စိတ္ဝင္စားစရာ ျဖစ္ေအာင္ ေရးသားၾကည္႔ၾကရေအာင္။ ယခုေရးမယ္႔ advenand.cpp မွာ ရတနာေတြကို (7,11) ဆိုတဲ႔ coordiantes မွာ ဝွက္ထားၿပီး ကစားသူက ရွာေတြ႔မေတြ႔ ၾကည္႔ၾကရေအာင္။

// advenand.cpp

// demonstrates AND logical operator

#include <iostream>

using namespace std;

#include <process.h> //for exit()

#include <conio.h> //for getche()

int main()

{

char dir=’a’;

int x=10, y=10;

while( dir != ‘\r’ )

{

cout << “\nYour location is “ << x << “, “ << y;

cout << “\nEnter direction (n, s, e, w): “;

dir = getche(); //get direction

switch(dir)

{

case ‘n’: y--; break; //update coordinates

case ‘s’: y++; break;

case ‘e’: x++; break;

case ‘w’: x--; break;

}

if( x==7 && y==11 ) //if x is 7 and y is 11

{

cout << “\nYou found the treasure!\n”;

exit(0); //exit from program

}

} //end switch

return 0;

} //end main

ဒီပရိုဂရမ္ တစ္ခုလံုးရဲ႕ လ်ိဳ႕ဝွက္ခ်က္ကေတာ႔ if( x==7 && y==11 ) ဆိုတဲ႔ ကုဒ္ေလးပဲ ျဖစ္ပါတယ္။ Test expression ဟာ x တန္ဖိုး 7 နဲ႔ y တန္ဖိုး 11 ႏွစ္ခုလံုး မွန္မွ၊ မွန္မွာ ျဖစ္ပါတယ္။ အဲဒီ ရလာဒ္ ျဖစ္လာေအာင္ logical AND operator && က relational operator ႏွစ္ခု (x==7) နဲ႔ (y==11) တို႔ကို ေပါင္းစပ္ေပးပါတယ္။ မွတ္သားသင္႔တဲ႔ အခ်က္တစ္ခုက relational expressions ႏွစ္ခုကို လက္သဲကြင္း အဖြင္႔အပိတ္နဲ႔ ကြင္းခတ္ထားဖို႔ မလိုတာပဲ ျဖစ္ပါတယ္။

( (x==7) && (y==11) ) // inner parentheses not necessary

relational operators ေတြဟာ logical operators ေတြထက္ precedence ပိုျမင္႔ပါတယ္။

ကုဒ္ေတြကို run ၾကည္႔ရင္ ေအာက္ပါအတိုင္း ေတြ႔ရမွာပါ-

Your location is 7, 10

Enter direction (n, s, e, w): s

You found the treasure!

C++ မွာ Logical Operator ၃ ခု ရွိပါတယ္-

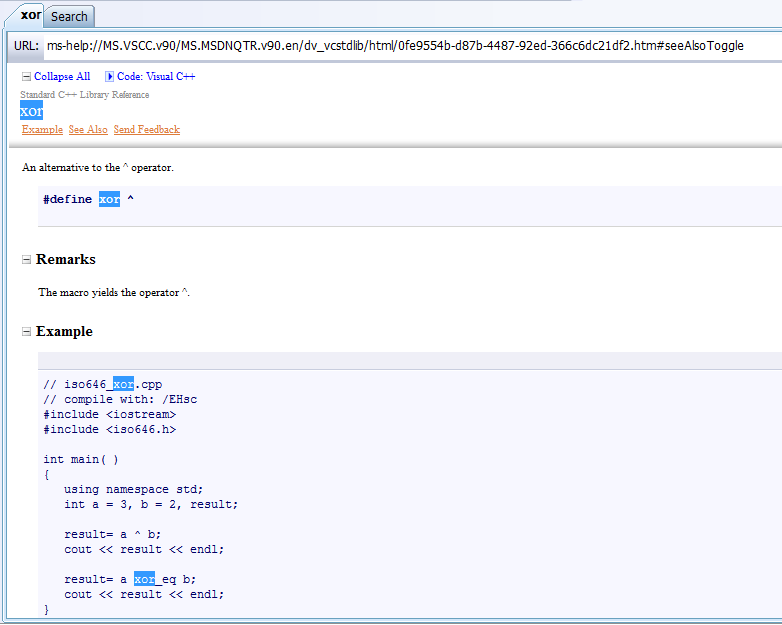
***Operator Effect***

&& Logical AND

|| Logical OR

! Logical NOT

Lafore ရဲ႕ စာအုပ္[1]၊ စာမ်က္ႏွာ ၁၁၆ မွာ ေျပာထားတာက C++ မွာ XOR (exclusive OR) logical operator မရွိဘူးလို႔ ဆိုပါတယ္။ ကၽြန္ေတာ္ကေတာ႔ C မွာတုန္းက သံုးခဲ႔တဲ႔ ^ ကို logical operator အေနနဲ႔ သံုးတာ အဆင္ေျပပါတယ္။ Visual Studio 2008 MSDN မွာေတာ႔ ေအာက္ပါအတိုင္း ႏွစ္မ်ိဳး သံုးျပထားပါတယ္။



**Logical OR Operator**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ game ကစားမယ္႔ user က အေရွ႕နဲ႔ အေနာက္ကို ခပ္ေဝးေဝးသြားမိရင္ နဂါးနဲ႔ ေတြ႔ေအာင္ ေရးခ်င္တယ္ ဆိုၾကပါစို႔။ လြတ္လြတ္လပ္လပ္ ေလွ်ာက္သြားေနတာကို ေၾကာက္စရာ အျဖစ္ေလးေတြ ထပ္ထည္႔ဖို႔ အရင္ အပတ္က advenand.cpp ကို ျပင္ေရးထားတဲ႔ advenor.cpp မွာ logical OR operator ကို အသံုးျပဳထားပါတယ္။

// advenor.cpp

// demonstrates OR logical operator

#include <iostream>

using namespace std;

#include <process.h> //for exit()

#include <conio.h> //for getche()

int main()

{

char dir=’a’;

int x=10, y=10;

while( dir != ‘\r’ ) //quit on Enter key

{

cout << “\n\nYour location is “ << x << “, “ << y;

if( x<5 || x>15 ) //if x west of 5 OR east of 15

cout << “\nBeware: dragons lurk here”;

cout << “\nEnter direction (n, s, e, w): “;

dir = getche(); //get direction

switch(dir)

{

case ‘n’: y--; break; //update coordinates

case ‘s’: y++; break;

case ‘e’: x++; break;

case ‘w’: x--; break;

} //end switch

} //end while

return 0;

} //end main()

အဲဒီ ပရိုဂရမ္ေလးထဲမွာ x<5 || x>15 ဆိုတာက x တန္ဖိုး 5 ထက္ ငယ္ၿပီး (အေနာက္ဘက္ကို ေဝးေဝးသြားမိတယ္ဆိုပါစို႔) 15 ထက္ႀကီးတဲ႔ (အေရွ႕ေရာက္လြန္းသြားတဲ႔) အေျခအေနေတြမွာ true ျဖစ္ပါလိမ္႔မယ္။ ဒီေနရာမွာ OR operator (||) က relational operators ေတြ ျဖစ္ၾကတဲ႔ < နဲ႔ > ထက္ precedence နိမ္႔တာ ျဖစ္လို႔ ကြင္းခတ္ေပးစရာ မလိုအပ္ပါဘူး။ ဒါေပမယ္႔ ကၽြန္ေတာ္႔သေဘာကေတာ႔ ျဖစ္ႏိုင္ရင္ လက္သဲကြင္းကို အျမင္ရွင္းေအာင္ ထည္႔ေပးေစလိုပါတယ္။ (x<5) || (x>15)

**Logical NOT Operator**

Logical NOT operator! ဆိုတာ Operand တစ္ခုတည္း လိုအပ္တဲ႔ unary operator တစ္ခုပဲ ျဖစ္ပါတယ္။ (C++ မွာ Operator အမ်ားစုဟာ operand ႏွစ္ခု လိုတဲ႔ binary operator မ်ားျဖစ္ၾကၿပီး ယခင္က သင္ခန္းစာမ်ားမွာ ေလ႔လာခဲ႔တဲ႔ conditional operator တစ္ခုတည္းသာ operand သံုးခုလိုတဲ႔ ternary operator ပဲ ျဖစ္ပါတယ္။ NOT operator (!) ရဲ႕ လုပ္ေဆာင္ခ်က္ကေတာ႔ operand ရဲ႕ logical value ကို ဆန္႔က်င္ဘက္ တန္ဖိုး ေျပာင္းလဲ ပစ္တာပဲ ျဖစ္ပါတယ္။ မွန္ေနတာကို မွားေအာင္ လုပ္ေပးႏိုင္ၿပီး၊ မွားေနတာကို မွန္ေအာင္ လုပ္ေပးတယ္ေပါ႔။ (ဘဝကိုသာ ဒီလို လြယ္လြယ္ကူကူ ျပဳျပင္ႏိုင္စြမ္းရွိရင္ ဘယ္ေလာက္ ေကာင္းမလဲေနာ္)။ ဥပမာအားျဖင္႔ (x==7) မွာ x တန္ဖိုးက 7 နဲ႔ တူရင္ true ျဖစ္မွာပါ။ ဒါကို!(x==7) လို႔ေျပာင္းေရးလိုက္တာနဲ႔ x ဟာ 7 နဲ႔ မတူမွ true ျဖစ္မွာပါ။ အဓိပၸါယ္က x != 7 နဲ႔ တူသြားပါတယ္။

**A True/False Value for Every Integer Variable**

ဒီလို operator ေတြအေၾကာင္းကို ေလ႔လာမိတဲ႔ အခါ expression တစ္ခုမွာ true/false တန္ဖိုး ရလာေစဖို႔ relational operator ေတြ သံုးဖို႔ လိုအပ္တယ္ဆိုတဲ႔ အေတြး ဝင္လာႏိုင္ပါတယ္။ ဒါေပမယ္႔ တကယ္တမ္းမွာ variable တစ္ခုတည္း ျဖစ္ေနပါေစ integer expression မွန္သမွ် true/false value ေတြ ရွိေနပါတယ္။ အဲဒီ integer x ရဲ႕ တန္ဖိုးဟာ 0 နဲ႔ မညီသမွ် true ျဖစ္ေနၿပီး 0 ျဖစ္ရင္ေတာ႔ false ျဖစ္သြားမွာပါ။ ဒီေနရာမွာ ! Operator ကို အသံုးျပဳလိုက္မယ္ဆိုရင္ !x ဟာ x ရဲ႕ တန္ဖိုးဟာ 0 နဲ႔ ညီသမွ် true ျဖစ္ေနၿပီး မညီ္ရင္ေတာ႔ false အျဖစ္ ေျပာင္းျပန္ ေျပာင္းလဲသြားမွာပဲ ျဖစ္ပါတယ္။

ထံုးစံအတိုင္း adventure game ေလးမွာ အထက္က idea ေတြကို အသံုးခ်ၾကည္႔ရေအာင္။ ကၽြန္ေတာ္တို႔ အေနနဲ႔ x နဲ႔ y တန္ဖိုး ႏွစ္ခုစလံုး ၇ ရဲ႕ ဆတိုးကိန္း (multiple) ေတြ ျဖစ္ေနတဲ႔ ေနရာေတြမွာ မိႈေတြ ထားခ်င္တယ္ ဆိုပါစို႔။ (အဲဒီမိႈေတြ စားမိရင္ player ကို magical power ေတြ ရေစပါတယ္) ဒါဆို x ေရာ y ပါ 7 နဲ႔ စားလို႔ ျပတ္တဲ႔ ေနရာေတြမွာ (x%7 ေရာ y%7 ပါ သုညနဲ႔ ညီတဲ႔အခါ) အဲဒီ effect ကို ေပးရမွာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ အဲဒီေနရာကို ဆံုးျဖတ္ဖို႔ if( x%7==0 && y%7==0 ) cout << “There’s a mushroom here.\n”; ဆိုၿပီး ေရးႏိုင္ပါတယ္။ ပိုၿပီး တိက်ခ်င္တယ္ ဆိုရင္ ! operator ကို အသံုးျပဳၿပိး ေအာက္ပါအတိုင္း ေရးသားလို႔ ရပါတယ္။

**if( !(x%7) && !(y%7) ) // if not x%7 and not y%7** ရလာဒ္ကေတာ႔ အတူတူပဲ ျဖစ္ပါတယ္။ အရင္ သင္ခန္းစာေတြတုန္းက logical operator && နဲ႔ || ဟာ relational operators ေတြထက္ precedence နိမ္႔တယ္ဆိုတာကို ေလ႔လာခဲ႔ၾကပါတယ္။ ဒါဆို အခု ဘာလို႔ x%7 နဲ႔ y%7 ကို လက္သဲကြင္း မျဖစ္မေန ခတ္ဖို႔ လိုလာတာလဲ? တကယ္ေတာ႔ ! ဟာ logical operator ျဖစ္ေပမယ္႔ unary operator ျဖစ္တာေၾကာင္႔ relational operators ေတြထက္ precedence ပိုျမင္႔ေနလို႔ပဲ ျဖစ္ပါတယ္။

**Precedence Summary**

Operators ေတြရဲ႕ precedence ေတြကို အႏွစ္ခ်ဳပ္ေလ႔လာၾကရေအာင္။ ဒီ စာရင္းမွာ အေပၚက operator ေတြဟာ ေအာက္က operator ေတြထက္ precedence ပိုျမင္႔ပါတယ္။ Row တူတဲ႔ operator ေတြကေတာ႔ precedence တူညီၾကပါတယ္။ လုပ္ေဆာင္တဲ႔ ေနရာမွာ precedence ပိုျမင္႔တဲ႔ operator က အရင္ လုပ္ေဆာင္မွာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ကၽြန္ေတာ္တို႔ မျဖစ္မေန အရင္လုပ္ခ်င္တဲ႔ expression ကို လက္သဲကြင္း ခတ္ေပးျခင္းျဖင္႔ force နဲ႔ လုပ္ေဆာင္ ႏိုင္ပါေသးတယ္။

***Operator type Operators Precedence***

Unary !, ++, ––, +, – Highest

Arithmetic Multiplicative \*, /, %

Additive +, –

Relational Inequality <, >, <=, >=

Equality ==, !=

Logical And &&

Or ||

Conditional ?:

Assignment =, +=, –=, \*=, /=, %= Lowest

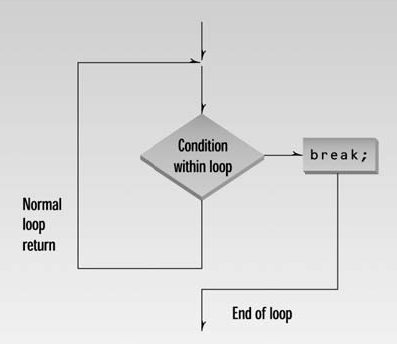
Relational expression ေတြ မ်ားျပားလာရင္ ရႈပ္ေထြးလာတတ္တာေၾကာင္႔ လိုလို မလိုလို လက္သဲကြင္းေတြကို အသံုးျပဳျခင္းအားျဖင္႔ အမွားအယြင္း နည္းသြားေစပါတယ္။ လက္သဲကြင္းပိုသြားလို႔ ဘာမွ မျဖစ္သြားပါဘူး။ precedence တြက္တာ မွားသြားခဲ႔ရင္ေတာင္ ကြင္းခတ္ထားလို႔ ျပႆနာ ရွင္းပါတယ္။ ကုဒ္ေတြကို ဖတ္ရတာလည္း ပိုမို လြယ္ကူလာေစပါတယ္။

**Other Control Statements**

C++ မွာ အျခား control statements ေတြ ရွိပါေသးတယ္။ switch ကို ေလ႔လာတုန္းက break statement ကို အသံုးျပဳနည္းကို ေတြ႔ခဲ႔ၾကပါတယ္။ ဒါေပမယ္႔ switch မွာသာမဟုတ္ဘဲ အျခားေနရာေတြမွာလဲ break ကို သံုးလို႔ ရပါေသးတယ္။ continue ကိုေတာ႔ loops ေတြမွာပဲ သံုးလို႔ ရပါတယ္။ goto ကိုေတာ႔ မသံုးသင္႔ေတာ႔ပါဘူး။ အဲဒီ statements ေတြကို အေသးစိတ္ ေလ႔လာၾကည္႔ၾကရေအာင္။

**The break Statement**

switch မွာ ေလ႔လာခဲ႔ၾကသလိုပါပဲ break statement ဟာ loop တစ္ခုကေန ထြက္သြားေစပါတယ္။ ဒါေၾကာင္႔ break ၿပီးေတာ႔ ဆက္လုပ္မယ္႔ statement ဟာ loop အျပင္ဘက္က ပထမဆံုးေတြ႔ရမယ္႔ statement ပဲျဖစ္ပါတယ္။ ပံု (၁၇-၁) မွာ break statement ရဲ႕ လုပ္ေဆာင္ပံုေတြကို flow chat နဲ႔ ရွင္းျပထားပါတယ္။



**ပံု (၁၇-၁) မွာ break statement ရဲ႕ လုပ္ေဆာင္ခ်က္ျပပံု**

To demonstrate break, here’s a program, SHOWPRIM, that displays the distribution of prime numbers in graphical form:

break အေၾကာင္းကို ရွင္းျပဖုိ႔အတြက္ ေအာက္ပါ showprim.cpp ပရိုဂရမ္ေလးကို အသံုးျပဳပါ႔မယ္။ ၄င္းဟာ prime numbers ေတြရဲ႕ distribution ကို graphical form နဲ႔ ျပသေပးမွာ ျဖစ္ပါတယ္။

// showprim.cpp

// displays prime number distribution

#include <iostream>

using namespace std;

#include <conio.h> //for getche()

int main()

{

const unsigned char WHITE = 219; //solid color (primes)

const unsigned char GRAY = 176; //gray (non primes)

unsigned char ch;

//for each screen position

for(int count=0; count<80\*25-1; count++)

{

ch = WHITE; //assume it’s prime

for(int j=2; j<count; j++) //divide by every integer from

if(count%j == 0) //2 on up; if remainder is 0,

{

ch = GRAY; //it’s not prime

break; //break out of inner loop

}

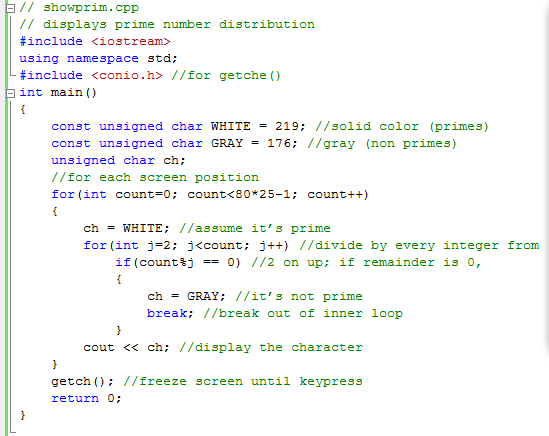
cout << ch; //display the character

}

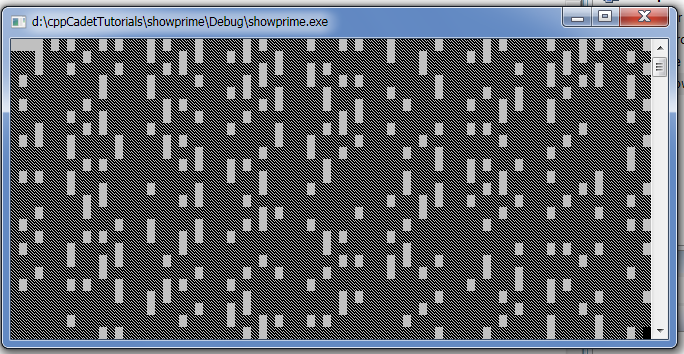
getch(); //freeze screen until keypress

return 0;

}



Console screen ရဲ႕ အက်ယ္ဟာ 80 columns နဲ႔ 25 rows (lines) ျဖစ္တဲ႔ အတြက္ ဂဏန္းအေရအတြက္အရဆိုရင္ 0-1999 (800\*25-1) ရွိပါတယ္။ သက္ဆိုင္ရာ ေနရာမွာ ရွိတဲ႔ ဂဏန္းေတြဟာ prime ျဖစ္ခဲ႔ရင္ အျဖဴေရာင္ ျခယ္ၿပီး မဟုတ္ခဲ႔ရင္ အညိဳေရာင္ျခယ္ေပးမွာပါ။ ပံု (၁၇-၂) မွာ showprime.cpp ရဲ႕ ရလာဒ္ကို ျပသေပးထားပါတယ္။ တိတိက်က်ေျပာရမယ္ဆိုရင္ သုည နဲ႔ တစ္ ဟာ prime ေတြ မဟုတ္ၾကပါဘူး။ ဒါေပမယ္႔ ပရိုဂရမ္ကို ပိုမိုရႈပ္ေထြးမသြားေအာင္ prime မ်ားနည္းတူ အျဖဴေရာင္ျခယ္ေပးထားပါတယ္။ ပထမဆံုး စာေၾကာင္းရဲ႕ ေကာ္လံေတြကို သုည ကေန ၇၉ အထိ ဂဏန္းေတြလို႔ စဥ္းစားၾကည္႔ရေအာင္။ စံုဂဏန္းေတြ ေနရာတိုင္းမွာ ၂ ကလြဲရင္ အားလံုး prime မဟုတ္ၾကပါဘူး။ ၂ နဲ႔ စားလို႔ ျပတ္ေနတာေၾကာင္႔ပါ။ ဒါဆို အျခား ဂဏန္းေတြအတြက္ ပံုေသ ပံုစံ တစ္ခုေရာ မရွိႏိုင္ဘူးလား? တကယ္လို႔သာ ေပးထားတဲ႔ ဂဏန္းေတြကို prime ဟုတ္မဟုတ္ ခန္႔မွန္းႏိုင္မယ္႔ pattern တစ္ခုမ်ား ရွာေတြ႔ႏိုင္ခဲ႔ရင္ ကမာၻ႔ သခၤ်ာ ပညာရွင္ေတြအတြက္ စိတ္လႈပ္ရွားစရာ သတင္းတစ္ခု ျဖစ္သြားႏိုင္ပါတယ္။



ပံု(၁၇-၂) showprim.cpp ၏ ရလာဒ္ကို ေတြ႔ရစဥ္

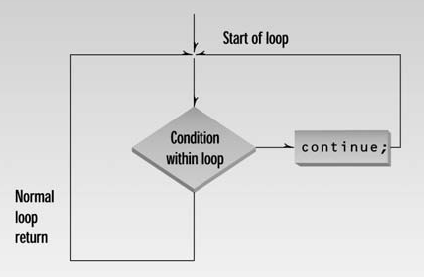
အတြင္းက loop က ဂဏန္းတစ္ခုကို prime မဟုတ္ဘူးလို႔ ဆံုးျဖတ္ခဲ႔ရင္ output ထုတ္ေပးမယ္႔ character ch ကို GRAY လို႔ သတ္မွတ္ေပးၿပီးတာနဲ႔ အဲဒီ inner loop ကေန break သံုးၿပီး ထြက္သြားမွာ ျဖစ္ပါတယ္။ (ပရိုဂရမ္ တစ္ခုလံုးက ထြက္မသြားဘဲ အျခား ဂဏန္းေတြကို စစ္ေဆးဖို႔ လိုေနပါေသးတယ္)။ အဲဒီမွာ သတိထားရမွာက break ဟာ loop ေတြ အားလံုးက ထြက္သြားေစတာ မဟုတ္ဘဲ ၄င္းရွိတဲ႔ အတြင္းအက်ဆံုး loop ကေနပဲ ထြက္သြားေစႏိုင္တာ ျဖစ္ပါတယ္။ ဘယ္လို loop အမ်ိဳးအစားပဲ ငံုထား ငံုထား အဲဒီအတိုင္းပဲ ျပဳမူမွာ ျဖစ္ပါတယ္။ ဥပမာ switch ကို loop တစ္ခုက ငံုထားခဲ႔ရင္ switch ထဲက break statement ဟာ switch ကိုပဲ ထြက္သြားေစျပီး loop ထဲကေတာ႔ ထြက္သြားမွာ မဟုတ္ပါဘူး။ ေနာက္ဆံုး cout statement ကေတာ႔ loop မၿပီးမခ်င္း prime test က သတ္မွတ္ေပးလိုက္တဲ႔ graphics character ေတြကို print ထုတ္ေပးမွာပဲ ျဖစ္ပါတယ္။

**ASCII Extended Character Set**

ဒီပရိုဂရမ္ထဲမွာ 128 ကေန 255 ရွိတဲ႔ *extended ASCII character set* ထဲက character ႏွစ္ခုကို ယူသံုးထားပါတယ္။ 219 ကေတာ႔ အျဖဴေရာင္ အတံုးေလး တစ္ခု ရဲ႕ သေကၤတ ျဖစ္ၿပီး 176 ကေတာ႔ အညိဳေရာင္ အတံုးေလးကို ကိုယ္စားျပဳထားတာပါ။ showprim.cpp နဲ႔ ယခင္က ဥပမာေတြ အေတာ္မ်ားမ်ားမွာ getch() ကို သံုးထားတာ သတိထားမိမွာပါ။ တကယ္ေတာ႔ ၄င္းဟာ DOS prompt မွာ ပရိုဂရမ္ မထြက္သြားခင္ ရလာဒ္ေတြကို ေစာင္႔ၾကည္႔ႏိုင္ဖို႔ keyboard က key တစ္ခုကို ဖတ္ဖို႔ သံုးထားတာပါ။ ဆိုလိုတာက getch() ကို ေရာက္ရင္ keyboard က key တစ္ခုခုကို မႏွိပ္မခ်င္း ပရိုဂရမ္က ရပ္ေနမွာ ျဖစ္ပါတယ္။ ေနာက္တစ္ခ်က္က character variables အတြက္ unsigned char ကို သံုးထားတာ ျဖစ္ပါတယ္။ တကယ္ေတာ႔ char က 127 အထိပဲ ရတဲ႔အတြက္ 255 အထိ သံုးလို႔ရေအာင္ unsigned char ကို သံုးထားတာပါ။

**The continue Statement**

break statement က ကၽြန္ေတာ္တို႔ကို loop ရဲ႕ ေအာက္ေျခကို ေခၚေဆာင္သြားေပးပါတယ္။ တခါတေလမွာ ဆန္႔က်င္ဘက္အေနနဲ႔ မေမွ်ာ္လင္႔တာ တစ္ခုခု ျဖစ္လာခဲ႔ရင္ loop ရဲ႕ ထပ္ဆံုးကို ျပန္သြားဖို႔ လိုအပ္လာတတ္ပါတယ္။ continue ဆိုတဲ႔ statement က အဲဒီ ရလာဒ္ကို ေပးႏိုင္ပါတယ္။ (ဒါေပမယ္႔ အတိအက်ေျပာရရင္ေတာ႔ continue ဟာ loop ရဲ႕ ကြင္းပိတ္ကို အရင္သြားၿပီးမွ ထိပ္ဆံုးကို ျပန္ခုန္တက္လိုက္တာပါ)။ ပံု(၁၇-၃) မွာ continue အလုပ္လုပ္ပံုကို flow chat နဲ႔ ျပထားပါတယ္။



ပံု(၁၇-၃) continue အလုပ္လုပ္ပံု

ယခင္ ဥပမာေတြ ေပးတုန္းက divdo.cpp ဆိုတဲ႔ ပရိုဂရမ္ေလးကို မွတ္မိဦးမယ္ ထင္ပါတယ္။ အဲဒီ ပရိုဂရမ္ေလးမွာ ကိန္းဂဏန္းေတြကို စားျပထားပါတယ္။ ဒါေပမယ္႔ သူ႔မွာ အဓိက အားနည္းခ်က္ တစ္ခု ရွိပါတယ္။ အဲဒါကေတာ႔ စားကိန္းကို သုည ေပးခဲ႔မိရင္ Divide Error ဆိုတဲ႔ runtime error message ေပၚလာၿပီး ပရိုဂရမ္ ထြက္သြားမွာပါ။ အခု ဥပမာေပးမယ္႔ divdo2.cpp မွာေတာ႔ အဲဒီ အေျခအေနကို သိမ္သိမ္ေမြ႔ေမြ႔ေလး ေျဖရွင္းေပးထားပါတယ္။

// divdo2.cpp

// demonstrates CONTINUE statement

#include <iostream>

using namespace std;

int main()

{

long dividend, divisor;

char ch;

do {

cout << “Enter dividend: “; cin >> dividend;

cout << “Enter divisor: “; cin >> divisor;

if( divisor == 0 ) //if attempt to

{ //divide by 0,

cout << “Illegal divisor\n”; //display message

continue; //go to top of loop

}

cout << “Quotient is “ << dividend / divisor;

cout << “, remainder is “ << dividend % divisor;

cout << “\nDo another? (y/n): “;

cin >> ch;

} while( ch != ‘n’ );

return 0;

}

တကယ္လို႔ user က စားကိန္းကို သုည ထည္႔ေပးခဲ႔မယ္ဆိုရင္ program က error message ထုတ္ေပးၿပီး continue အသံုးျပဳကာ loop ရဲ႕ ထိပ္ဆံုးပိုင္းကို ျပန္သြားလို႔ ဂဏန္း ထပ္ေတာင္းမွာ ျဖစ္ပါတယ္။

Enter dividend: 10

Enter divisor: 0

Illegal divisor

Enter dividend:

တကယ္လို႔ ဒီေနရာမွာ break ကို သံုးခဲ႔မိရင္ do loop ကေရာ ပရိုဂရမ္ကပါ ထြက္သြားမွာ ျဖစ္ပါတယ္။

**အခန္း(၃)**

Structures

**Structures**

ဒီအခန္းမွာေတာ႔ Loops and Desicions အေၾကာင္းကို အဆံုးသတ္တဲ႔ အေနနဲ႔ အနည္းငယ္ ေဆြးေႏြးၿပီး Structures ေတြအေၾကာင္းကို ဆက္လက္ ေလ႔လာသြားမွာ ျဖစ္ပါတယ္။ တကယ္ေတာ႔ ကၽြန္ေတာ္တို႔ အဲဒီအခန္းကို ေလ႔လာခဲ႔တာ အေတာ္ေလး ျပည္႔စံုသြားပါၿပီ။ ဒါေပမယ္႔ အသံုးမျပဳသင္႔တဲ႔ statement တစ္ခုကို ရွင္းျပဖို႔ က်န္ပါေသးတယ္။ ၄င္းကေတာ႔ goto ဆိုတဲ႔ statement ပဲ ျဖစ္ပါတယ္။ goto ကို အသံုးျပဳျခင္းအားျဖင္႔ နားလည္ရခက္ၿပီး ရႈပ္ေထြးတဲ႔ ကုဒ္ေတြ ျဖစ္လာေစႏိုင္ပါတယ္။ ဒါေၾကာင္႔ လံုးဝ အသံုးမျပဳၾကေစခ်င္ပါဘူး။ အသံုးျပဳနည္းကိုေတာ႔ တင္ျပလိုက္ပါတယ္။

ပထမဆံုး ကၽြန္ေတာ္တို႔ jump လုပ္ဖို႔ လိုအပ္တဲ႔ ကုဒ္ေနရာမွာ label တစ္ခုကို ေရးသားထားရမွာ ျဖစ္ပါတယ္။ Label ကို colon ":" နဲ႔ ပိတ္ေပးရမွာပါ။ ပရိုဂရမ္ရဲ႕ တေနရာရာမွာ goto ရဲ႕ ေနာက္ကို အဲဒီ label name ထည္႔ေပးလိုက္မယ္ဆိုရင္ runtime မွာ program control က အဲဒီေနရာေရာက္တာနဲ႔ label ထိုးထားတဲ႔ ေနရာကို jump လုပ္သြားမွာ ျဖစ္ပါတယ္။ ဥပမာ -

*goto SystemCrash;*

*// other statements*

*SystemCrash:*

*// control will begin here following goto*

ေရွ႕ပိုင္း သင္ခန္းစာမ်ားမွာ float, char, int အစရွိသျဖင္႔ ရိုးရွင္းတဲ႔ data type ေတြကို ေလ႔လာခဲ႔ၾကၿပီးပါၿပီ။ အဲဒီ data type ေတြ အားလံုးဟာ (အရပ္အျမင္႔႔၊ ကိုယ္အေလးခ်ိန္၊ ကားတန္ဖိုး စတဲ႔) သတင္းအခ်က္အလက္ တစ္ခုကိုပဲ သိမ္းဆည္းထားႏိုင္ပါတယ္။ အခ်က္အလက္ေတြကို အေျမာက္အမ်ား သိမ္းဆည္းခ်င္လာတဲ႔ အခါမွာ အဲဒီ data type ေတြဟာ မလံုေလာက္ေတာ႔ပါဘူး။ ဒါေၾကာင္႔ array, structure စတာေတြကို အသံုးခ်ၿပီး သိမ္းဆည္းဖို႔ လိုလာပါတယ္။ data type တူတဲ႔ ေဒတာ အမ်ားအျပား သိမ္းဆည္းဖို႔အတြက္ေတာ႔ ေနာက္ပိုင္းအခန္းေတြမွာ ေလ႔လာရမယ္႔ array ကို အသံုးျပဳရမွာ ျဖစ္ပါတယ္။ data type မတူညီတဲ႔ ေဒတာေတြကို သိမ္းဆည္းဖို႔အတြက္ေတာ႔ structure ကို အသံုးျပဳႏိုင္ပါတယ္။

**Structures**

Structure ဆိုတာ တကယ္ေတာ႔ ရိုးရိုး variable ေတြကို စုစည္းထားတာ ျဖစ္ပါတယ္။ အဲဒီ variables ေတြဟာ int, float, char စသျဖင္႔ types အမ်ိဳးမ်ိဳး ျဖစ္ႏိုင္ပါတယ္။ structure ထဲမွာ ပါတဲ႔ variables ေတြကို structure ရဲ႕ *members* ေတြလို႔ ေခၚပါတယ္။

C programming စာအုပ္ေတြမွာတုန္းကေတာ႔ structures ေတြကို advanced feature အျဖစ္ ယူဆတဲ႔အတြက္ စာအုပ္တိုင္းရဲ႕ ေနာက္ဆံုး အခန္းေတြမွာပဲ ေဖာ္ျပေလ႔ ရွိၾကပါတယ္။ C++ programmers ေတြအတြက္ကေတာ႔ structures ဆိုတာ objects နဲ႔ classes ေတြကို နားလည္ႏိုင္ဖို႔ အဓိက အေျခခံ အခ်က္ ႏွစ္ခုထဲက တစ္ခုပဲ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ အရင္ဆံုး မိတ္ဆက္ ေဖာ္ျပေပးထားျခင္း ျဖစ္ပါတယ္။

တကယ္ေတာ႔ class နဲ႔ structure ေတြမွာ syntax ေတြ အေတာ္ေလး တူညီေနပါတယ္။ structure က ေတာ႔ အခ်က္အလက္ေတြကို စုထားတာျဖစ္ၿပီး class က အခ်က္အလက္ေတြ အျပင္ function ေတြ စုထားတာေလးပဲ ကြာျခားမႈ ရွိပါတယ္။ ဒါေၾကာင္႔ structure ကို နားလည္ထားျခင္းျဖင္႔ class နဲ႔ object ေတြရဲ႕ သေဘာတရားကို ေလ႔လာရတာ လြယ္ကူေစမွာ ျဖစ္ပါတယ္။ Pascal နဲ႔ အျခား language အခ်ိဳ႕မွာ ရွိတဲ႔ *records* ရဲ႕ သေဘာတရားနဲ႔ ရည္ရြယ္ခ်က္က C နဲ႔ C++ မွာရွိတဲ႔ structure ရဲ႕ သေဘာတရားနဲ႔ အတူတူပဲ ျဖစ္ပါတယ္။

**A Simple Structure**

Integer variables ႏွစ္ခုနဲ႔ floating point number variable တစ္ခု ပါဝင္တဲ႔ structure တစ္ခုကို စတင္ ဖန္တီးၾကရေအာင္။ အဲဒီလို structure မ်ိဳးဟာ ကုမၸဏီ တစ္ခုရဲ႕ စတိုးထဲက အစိတ္အပိုင္း ေတြကို ကိုယ္စားျပဳထားပါတယ္။ အစိတ္္အပိုင္း တစ္ခုစီအတြက္ လိုအပ္တဲ႔ အခ်က္အလက္ေတြကို သတ္မွတ္ေပးထားတဲ႔ blueprint လိုပါပဲ။ Structure ရဲ႕ ပထမ member ကေတာ႔ အဲဒီအစိတ္အပိုင္း ပါဝင္တဲ႔ ပစၥည္းရဲ႕ model number ပဲ ျဖစ္ပါတယ္။ ဒုတိယ member ကေတာ႔ အစိတ္အပိုင္း number ျဖစ္ၿပီး ေနာက္ဆံုးတစ္ခုက က်သင္႔ေငြ ျဖစ္ပါတယ္။ ေအာက္မွာ ေဖာ္ျပထားတဲ႔ ပရိုဂရမ္ parts.cpp မွာ part ဆိုတဲ႔ structure တစ္ခုကို define ျပဳလုပ္ထားပါတယ္။ ထို႔ေနာက္ အဲဒီ structure type အမ်ိဳးအစား variable တစ္ခု (part1) ကို define ျပဳလုပ္ပါတယ္။ သူ႔ရဲ႕ members ေတြထဲကို values ေတြ ထည္႔သြင္းေပးၿပီး display ျပန္လုပ္ေပးမွာ ျဖစ္ပါတယ္။ ေလ႔လာၾကည္႔ၾကရေအာင္-

// parts.cpp

// uses parts inventory to demonstrate structures

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct part //declare a structure

{

int modelnumber; //ID number of widget

int partnumber; //ID number of widget part

float cost; //cost of part

};

////////////////////////////////////////////////////////////////

int main()

{

part part1; //define a structure variable

part1.modelnumber = 6244; //give values to structure members

part1.partnumber = 373;

part1.cost = 217.55F;

//display structure members

cout << “Model “ << part1.modelnumber;

cout << “, part “ << part1.partnumber;

cout << “, costs $” << part1.cost << endl;

return 0;

}

Run ၾကည္႔တဲ႔အခါ ေအာက္ပါ output ေတြကို ရရွိမွာ ျဖစ္ပါတယ္။

Model 6244, part 373, costs $217.55

parts.cpp ပရိုဂရမ္မွာ အဓိက အစိတ္အပိုင္း ၃ ခု ပါဝင္ပါတယ္။ ၁) structure ကို define ျပဳလုပ္ျခင္း ၂) structure variable ကို define ျပဳလုပ္ျခင္းနဲ႔ ၃) structure ရဲ႕ members ေတြကို ရယူသံုးစြဲျခင္းတို႔ပဲ ျဖစ္ပါတယ္။ တစ္ခုခ်င္းကို ေအာက္မွာ အေသးစိတ္ ေလ႔လာၾကည္႔ၾကရေအာင္။

**Defining the Structure**

structure definition ဆိုတာကေတာ႔ structure တစ္ခု ဘယ္လို ဖြဲ႕စည္းထားတယ္ ဆိုတာကို ေဖာ္ျပေပးပါတယ္။ structure မွာ ပါဝင္မယ္႔ members ေတြကို သတ္မွတ္ေပးပါတယ္။ ဥပမာ -

struct part

{

int modelnumber;

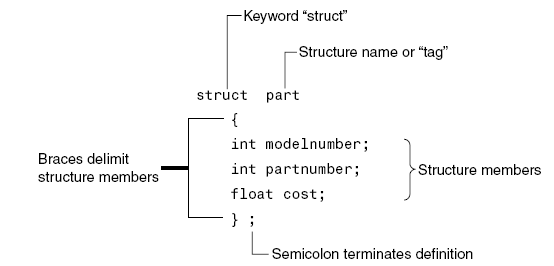
int partnumber;

float cost;

};

**Syntax of the Structure Definition**

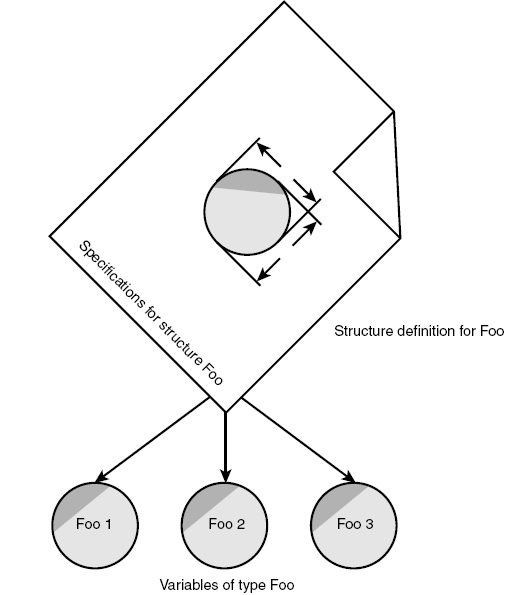
struct ဆိုတဲ႔ keyword နဲ႔ structure definition ကို အစျပဳထားပါတယ္။ ေနာက္က ကပ္ရပ္ေရးရတာက *structure name* ဒါမွမဟုတ္ *tag* ျဖစ္ပါတယ္။ ဒီဥပမာမွာေတာ႔ part လို႔ ေပးထားပါတယ္။ modelnumber, partnumber နဲ႔ cost ဆိုတဲ႔ structure members ေတြရဲ႕ declarations ကို တြန္႔ကြင္း အဖြင္႔အပိတ္ၾကားမွာ ေရးသားရပါတယ္။ တြန္႔ကြင္း အပိတ္ရဲ႕ ေနာက္မွာ semicolon နဲ႔ ပိတ္ၿပီး structure တစ္ခုလံုးကို အဆံုးသတ္ထားပါတယ္။ အျခား code block ေတြ (ဥပမာ- loop, decisions နဲ႔ functions) ကိုေတာ႔ တြန္႔ကြင္း အဖြင္႔အပိတ္နဲ႔ပဲ အစျပဳ၊ အဆံုးသတ္ေလ႔ ရွိၿပီး structure မွာေတာ႔ semicolon နဲ႔ အဆံုးသတ္တာ ျဖစ္ပါတယ္။ ေအာက္မွာ structure declaration တစ္ခုရဲ႕ syntax ကို ေဖာ္ျပေပးထားပါတယ္။



**ပံု (၁၈-၁)** *Syntax of the structure definition.*

**Use of the Structure Definition**

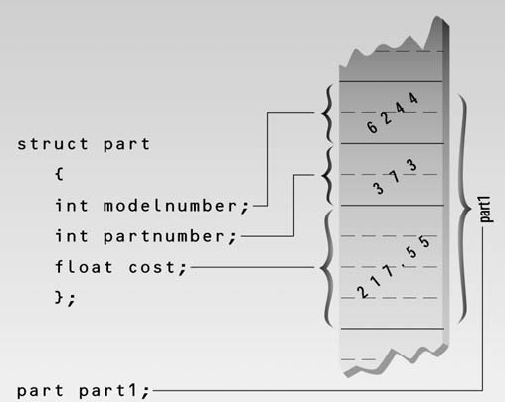
Structure definition သည္ အဆိုပါ Structure အမ်ိဳးအစား variables ကို ဖန္တီးရန္ blueprint အေနျဖင္႔သာ အသံုးျပဳထားသည္။ ဆိုလိုသည္႔အခ်က္မွာ structure definition သည္ အျခား သာမာန္ variables မ်ား၏ definition မ်ားလို memoryအတြင္း ေနရာယူူျခင္း မရွိသလို variable မ်ားကို အမည္ေပးျခင္းလည္း မဟုတ္ေပ။ ၄င္းသည္ structure variable တစ္ခုကို define ျပဳလုပ္ၿပီးလွ်င္ မည္သည္႔ ပံုစံ ရွိႏိုင္မည္ဟူေသာ ေဖာ္ျပခ်က္ သက္သက္သာ ျဖစ္သည္။ ေနာက္ပိုင္း class ကို ေလ႔လာသည္႔အခါ အလားတူ သေဘာတရားမ်ိဳးကို အသံုးျပဳထားျခင္းျဖစ္ေၾကာင္း ေတြ႔ရွိရမည္ ျဖစ္သည္။



ပံု (၁၈-၂) structure ႏွင္႔ structure variables

**Defining a Structure Variable**

main() ထဲရွိ ပထမဆံုး ေဖာ္ျပခ်က္ျဖစ္ေသာ part part1; သည္ structure part type ၏ part1 ဟူေသာ variable တစ္ခုကို defines ျပဳလုပ္ျခင္းပင္ ျဖစ္သည္။ အဆိုပါ ကုဒ္ေၾကာင္႔ memory ထဲတြင္ part1 ၏ members မ်ားကို ဖန္တီးမည္ ျဖစ္သည္။ မည္မွ် ေနရာယူမည္ဆိုသည္ကို part1 အတြင္းရွိ members မ်ားျဖစ္ေသာ modelnumber, partnumber ႏွင္႔ cost တို႔က သတ္မွတ္ေပးသည္။ အထက္ပါ ဥပမာတြင္ integer ႏွစ္ခုအတြက္ 4 bytes (32-bit) ႏွင္႔ float အတြက္ 4 bytes စုစုေပါင္း 8 bytes ေနရာယူမည္ ျဖစ္သည္။ ပံု (၁၈-၃) ကိုၾကည္႔ပါ။



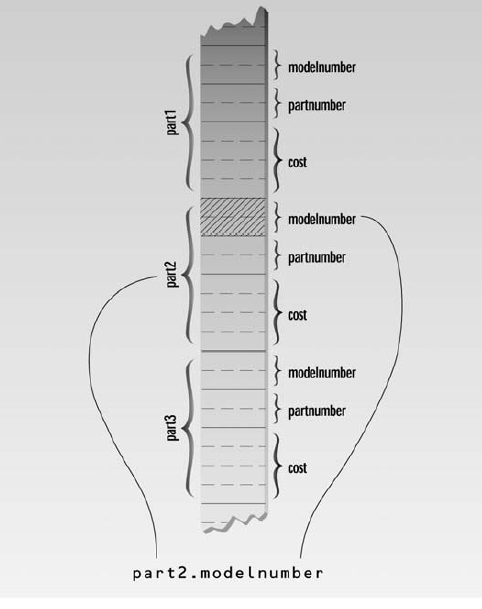
ပံု (၁၈-၃) memory အတြင္း structure member မ်ား ေနရာယူပံု

**Accessing Structure Members**

Structure variable ေတြကို defined လုပ္ၿပီးတာနဲ႔ သူရဲ႕ member ေတြကို *dot operator* အသံုးျပဳၿပီး access လုပ္ႏိုင္ပါၿပီ။ ေအာက္မွာ အဲဒီလို ေရးသားနည္းကို နမူနာျပထားပါတယ္။

part1.modelnumber = 6244;

Structure member ေတြကို အသံုးျပဳဖို႔ အပိုင္း (၃) ပိုင္း ေရးသားေပးရမွာ ျဖစ္ပါတယ္။ ပထမ အပိုင္းကေတာ႔ structure variable ရဲ႕ name ျဖစ္ပါတယ္။ အဲဒီေနာက္မွာ ဒုတိယအပိုင္းအေနနဲ႔ period (.) ကိုေရးသားရမွာျဖစ္ၿပီး ေနာက္ဆံုး တတိယ အပိုင္းအေနနဲ႔ member name (ဒီဥပမာမွာ- modelnumber)ကို ေရးသားရမွာပါ။ အဲဒီအဓိပၸါယ္ကေတာ႔ “the modelnumber member of part1.” ဒါမွမဟုတ္ part1 ရဲ႕ member တစ္ခုျဖစ္တဲ႔ modelnumber လို႔ ဆိုလိုတာပဲ ျဖစ္ပါတယ္။ Dot operator ရဲ႕ အေခၚအေဝၚ အျပည္႔အစံု က *member access operator*  ျဖစ္ေပမယ္႔ ရွည္တဲ႔အတြက္ အသံုးမျပဳခ်င္ၾကတာ ျဖစ္ပါတယ္။ အဲဒီမွာ သတိျပဳရမယ္႔ အခ်က္ကေတာ႔ ပထမဆံုး ေရးသားတဲ႔ အပိုင္းဟာ structure definition (part) မဟုတ္ဘဲ specific structure variable (part1) ျဖစ္တယ္ဆိုတာပါပဲ။ Variable name ေတြဟာ part1, part2 စသျဖင္႔ တစ္ခုနဲ႔ တစ္ခု ခြဲျခားဖုိ႔အတြက္ အသံုးျပဳတာပဲ ျဖစ္ပါတယ္။ ေအာက္က ပံု(၁၉-၁) မွာ ေလ႔လာၾကည္႔ႏိုင္ပါတယ္။



ပံု(၁၉-၁) Dot operator အသံုးျပပံု

Structure members ေတြကို အျခား variables ေတြလိုပဲ လုပ္ေဆာင္ေစပါတယ္။ ဥပမာ part1.modelnumber = 6244; ဆိုတဲ႔ statement မွာ သာမန္ assignment operator ကိုသံုးၿပီး အဲဒီ member ထဲကို တန္ဖိုး 6244 ထည္႔သြင္းေပးလိုက္တာ ျဖစ္ပါတယ္။ cout statements ေတြကို သံုးၿပီးေတာ႔လည္း members ေတြရဲ႕ တန္ဖိုးေတြကို print ထုတ္ေပးလို႔ ရပါတယ္။

**cout << “\nModel “ << part1.;**

အထက္ပါ statement ဟာ structure member ျဖစ္တဲ႔ modelnumber ရဲ႕ value ကို output ထုတ္ေပးမွာ ျဖစ္ပါတယ္။

**Other Structure Features**

Structures ေတြဟာ အံ႔ၾသစရာ ေကာင္းေလာက္ေအာင္ စြယ္စံု အသံုးဝင္ပါတယ္။ ဒါေၾကာင္႔ ေနာက္ထပ္ အသံုးတည္႔တဲ႔ structure ေတြရဲ႕ features ေတြကို ေလ႔လာၾကည္႔ၾကရေအာင္။

**Initializing Structure Members**

ေနာက္လာမယ္႔ ဥပမာ ပရိုဂရမ္ေလးမွာ structure variable ေတြကို define ျပဳလုပ္ရင္း structure members ေတြကို initialize ျပဳလုပ္နည္းေတြကို ျပသထားပါတယ္။ ဒီဥပမာမွာပဲ structure type တစ္ခုကို variable တစ္ခုထက္ပိုၿပီး အသံုးခ်ႏိုင္တာကိုလဲ ေလ႔လာႏိုင္ပါတယ္။

// partinit.cpp

// shows initialization of structure variables

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct part //specify a structure

{

int modelnumber; //ID number of widget

int partnumber; //ID number of widget part

float cost; //cost of part

};

////////////////////////////////////////////////////////////////

int main()

{ //initialize variable

part part1 = { 6244, 373, 217.55F };

part part2; //define variable

//display first variable

cout << “Model “ << part1.modelnumber;

cout << “, part “ << part1.partnumber;

cout << “, costs $” << part1.cost << endl;

part2 = part1; //assign first variable to second

//display second variable

cout << “Model “ << part2.modelnumber;

cout << “, part “ << part2.partnumber;

cout << “, costs $” << part2.cost << endl;

return 0;

}

ဒီပရိုဂရမ္ေလးမွာ part ရဲ႕ type အမ်ိဳးအစား variables ႏွစ္ခု (part1, part2) ကို define ျပဳလုပ္ပါတယ္။ part1 ကိုေတာ႔ define ျပဳလုပ္ခ်ိန္မွာပဲ initialize လုပ္ေပးလိုက္ပါတယ္။ တနည္းအားျဖင္႔ တန္ဖိုးေတြ ထည္႔သြင္းေပးလိုက္တာပါ။ ၿပီးေတာ႔ part1 ထဲက members ေတြရဲ႕ တန္ဖိုးေတြကို print ထုတ္ေပးပါတယ္။ part2 ထဲကို part1 assign လုပ္ေပးပါတယ္။ ေနာက္ဆံုးမွာ part2 ထဲက members ေတြရဲ႕ တန္ဖိုးေတြကို ေအာက္ပါအတိုင္း print ထုတ္ေပးလုိက္ပါတယ္။

Model 6244, part 373, costs $217.55

Model 6244, part 373, costs $217.55

တန္ဖိုးႏွစ္ခု တူေနၾကတာကေတာ႔ variable ႏွစ္ခုဟာ တူညီေနလို႔ပဲ ျဖစ္ပါတယ္။

**part part1 = { 6244, 373, 217.55 };** ဆိုတဲ႔ statement ကေတာ႔ part1 ကိုေတာ႔ define ျပဳလုပ္ခ်ိန္မွာinitialize လုပ္ေပးလိုက္တာပါ။ structure members ေတြကို assign လုပ္မယ္႔ တန္ဖိုးေတြကို တြန္႔ကြင္းနဲ႔ ခတ္ၿပီး တစ္ခုနဲ႔ တစ္ခုၾကား ေကာ္မာခံထားဖို႔ လိုပါတယ္။ ပထမဆံုး တန္ဖိုးကို ပထမ member၊ ဒုတိယ တန္ဖိုးကို ဒုတိယ member စသျဖင္႔ ထည္႔သြင္းေပးသြားမွာ ျဖစ္ပါတယ္။

**Structure Variables in Assignment Statements**

ကၽြန္ေတာ္တို႔ partinit.cpp မွာ ေတြ႔ခဲ႔ၾကတဲ႔ အတိုင္း structure variable ေတြကို တစ္ခုနဲ႔ တစ္ခု ေအာက္ပါအတိုင္း assign ျပဳလုပ္ႏိုင္ပါတယ္။

**part2 = part1;**

အဲဒီအခါမွာ part1 ရဲ႕ member အားလံုးဟာ part2 ရဲ႕ သက္ဆိုင္ရာ member ေတြအားလံုးကို assign ျပဳလုပ္ေပးပါတယ္။ Structure တစ္ခုမွာ member ေတြ ဒါဇင္နဲ႔ခ်ီၿပီး ပါဝင္ႏိုင္တာေၾကာင္႔ ကြန္ပ်ဴတာအေနနဲ႔ ဒီ assignment ျဖစ္စဥ္ကို ေဆာင္ရြက္ဖို႔ အလုပ္ေတာ္ေတာ္မ်ားမ်ား လုပ္ေပးရမွာ ျဖစ္ပါတယ္။

မွတ္သားရမွာ တစ္ခုကေတာ႔ အဲဒီလို assign ျပဳလုပ္မယ္႔ structures ေတြဟာ structure type တူညီဖို႔ လိုအပ္ပါတယ္။ တကယ္လို႔ မတူညီတဲ႔ structures ႏွစ္ခုကို assign ျပဳလုပ္မယ္ဆိုရင္ error message ျပမွာ ျဖစ္ပါတယ္။

**A Measurement Example**

ေနာက္ထပ္ ဥပမာတစ္ခုအေနနဲ႔ မတူညီတဲ႔ သတင္းအခ်က္အလက္ ေတြကို အုပ္စုဖြဲ႔ၿပီး အသံုးခ်ဖို႔ structure တစ္ခု တည္ေဆာက္ျပပါမယ္။ အေဆာက္အဦ တစ္ခုရဲ႕ ပံုၾကမ္းကို ေလ႔လာၾကည္႔မယ္ဆိုရင္ အကြာအေဝးေတြဟာ ေပေတြ လက္မေတြနဲ႔ တိုင္းတာထားတာကို ေတြ႔ရွိရမွာပါ။ (တစ္ေပမွာ ၁၂ လက္မ ရွိတယ္ဆိုေတာ႔ အားလံုး သိၿပီးသားျဖစ္ပါတယ္)။ ဧည္႔ခန္းတစ္ခုရဲ႕ အလ်ားကို 15’–8” လို႔ ျပထားမယ္ဆိုရင္ အဲဒီမွာ သံုးထားတဲ႔ hyphen ဟာ အႏုတ္ လကၡဏာ မဟုတ္ဘဲ ေပနဲ႔ လက္မကို ျခားထားေပးတာပါ။ ၁၅ ေပ နဲ႔ ၈ လက္မလို႔ ဆိုလိုပါတယ္။ အဲဒီလို စနစ္မ်ိဳးအတြက္ ဆိုရင္ ေပအတြက္ variable တစ္ခုနဲ႔ လက္မအတြက္ variable တစ္ခုဆိုၿပီး ႏွစ္ခု သံုးရပါလိမ္႔မယ္။ ေအာက္က englstrc.cpp ပရိုဂရမ္ေလးမွာ အဲဒီျပႆနာကို structure အသံုးခ်ၿပီး ေျဖရွင္းျပထားပါတယ္။ အဲဒီ structure ထဲမွာ မတူညီတဲ႔ Distance type ႏွစ္ခုကို အတူတူ ထည္႔သြင္းေရးသား အသံုးျပဳထားတာ ျဖစ္ပါတယ္။

// englstrc.cpp

// demonstrates structures using English measurements

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct Distance //English distance

{

int feet;

float inches;

};

////////////////////////////////////////////////////////////////

int main()

{

Distance d1, d3; //define two lengths

Distance d2 = { 11, 6.25 }; //define & initialize one length

//get length d1 from user

cout << “\nEnter feet: “; cin >> d1.feet;

cout << “Enter inches: “; cin >> d1.inches;

//add lengths d1 and d2 to get d3

d3.inches = d1.inches + d2.inches; //add the inches

d3.feet = 0; //(for possible carry)

if(d3.inches >= 12.0) //if total exceeds 12.0,

{ //then decrease inches by 12.0

d3.inches -= 12.0; //and

d3.feet++; //increase feet by 1

}

d3.feet += d1.feet + d2.feet; //add the feet

//display all lengths

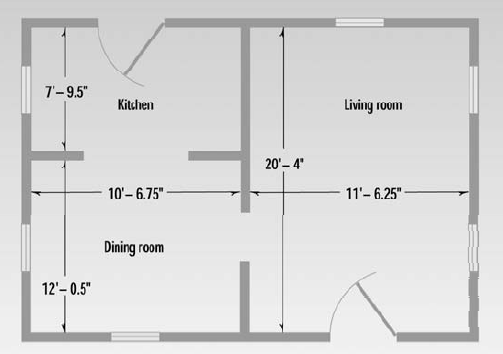
cout << d1.feet << “\’-” << d1.inches << “\” + “;

cout << d2.feet << “\’-” << d2.inches << “\” = “;

cout << d3.feet << “\’-” << d3.inches << “\”\n”;

return 0;

}



ပံု(၁၉-၂) အိမ္ပံုၾကမ္း ႏွင္႔ အတိုင္းအတာမ်ား ျပပံု

အထက္ပါ ပရိုဂရမ္ထဲမွာ feet နဲ႔ inches ဆိုတဲ႔ members ႏွစ္ခု ပါဝင္တဲ႔ Distance structure ကို အသံုးျပဳထားပါတယ္။ လက္မေတြကို မ်ားေသာအားျဖင္႔ ဒႆမကိန္းနဲ႔ ေဖာ္ျပႏိုင္တာေၾကာင္႔ inches အတြက္ float type ကို သံုးထားပါတယ္။ ေပ ကေတာ႔ အျမဲတမ္း ကိန္းျပည္႔ျဖစ္ေနတာေၾကာင္႔ feet ကို int type အသံုးျပဳထားတာပါ။ အတိုင္းအတာေတြကို မွတ္သားဖို႔ structure variable သံုးခု (d1, d2, d3) ကို define ျပဳလုပ္ပါတယ္။ ဒါေပမယ္႔ d2 ကိုပဲ initialize ျပဳလုပ္ေပးပါတယ္။

**Distance d2 = { 11, 6.25 }; //define & initialize one length**

ပရုိဂရမ္က user ကို အတိုင္းအတာတစ္ခုရဲ႕ ေပ နဲ႔ လက္မ တန္ဖိုးေတြကို ထည္႔သြင္းေပးဖို႔ ေတာင္းဆိုပါတယ္။ (လက္မ တန္ဖိုးဟာ ၁၂ ေအာက္ ငယ္ဖို႔ လိုပါတယ္)။ အဲဒီေနာက္မွာေတာ႔ d1 နဲ႔ d2 ကို ေပါင္းၿပီး d3 ထဲမွာ ရလဒ္ကို သိမ္းလိုက္ပါတယ္။ ေနာက္ဆံုးမွာေတာ႔ အတိုင္းအတာ သံုးခုလံုးရဲ႕ တန္ဖိုးေတြကို print ျပန္ထုတ္ေပးပါတယ္။ ေအာက္မွာ နမူနာ ျပထားပါတယ္။

Enter feet: 10

Enter inches: 6.75

10’-6.75” + 11’-6.25” = 22’-1”

အဲဒီမွာ သတိထားစရာ အခ်က္ကေတာ႔ အဲဒီ structure variable ေတြကို ေအာက္ပါအတိုင္း တန္းေပါင္းလို႔ မရတာပါပဲ။

~~d3 = d1 + d2;~~ // can’t do this in ENGLSTRC

ဘာေၾကာင္႔လဲဆိုေတာ႔ C++ မွာ Distance type variables ေတြကို ေပါင္းေပးႏိုင္တဲ႔ routine ေတြ မရွိလို႔ဘဲ ျဖစ္ပါတယ္။ + operator ဟာ float, int စတဲ႔ built-in types ေတြကို ေပါင္းေပးႏိုင္ပါတယ္။ ဒါေပမယ္႔ ကၽြန္ေတာ္တို႔ ကိုယ္တိုင္ define လုပ္လိုက္တဲ႔ type ေတြကိုေတာ႔ ေပါင္းေပးႏိုင္စြမ္း မရွိပါဘူး။ (Class ကို ေလ႔လာၾကတဲ႔ အခါမွာေတာ႔ “Operator Overloading,” အေၾကာင္းကို ရွင္းျပေပးပါ႔မယ္။)

**Structures Within Structures**

structures ေတြထဲမွာ ေနာက္ထပ္ structures ေတြ nested လုပ္ၿပီး ထည္႔ႏိုင္ပါတယ္။ နမူနာအေနနဲ႔ englstrc.cpp ကို ျပင္ေရးျပပါ႔မယ္။ ဒီ ဥပမာထဲမွာ အခန္းတစ္ခုရဲ႕ အလ်ားနဲ႔ အနံ အရြယ္အစားေတြကို သိမ္းထားမယ္႔ data structure တစ္ခုကို ဖန္တီးျပပါ႔မယ္။ အဂၤလိပ္ အတိုင္းအတာေတြကို သံုးထားတာျဖစ္လို႔ ေပ နဲ႔ လက္မကိုုပဲ ဆက္သံုးသြားမွာပါ။ ေအာက္မွာ Room ဆိုတဲ႔ structre တစ္ခုကို Distance type variable lenght နဲ႔ width ကို အသံုးျပဳၿပီး တည္ေဆာက္ျပထားပါတယ္။

struct Room

{

Distance length;

Distance width;

}

ေအာက္က englarea.cpp မွာေတာ႔ အခန္းတစ္ခန္းကို ကိုယ္စားျပဳဖို႔ Room structure ကို သံုးထားပါတယ္။

// englarea.cpp

// demonstrates nested structures

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct Distance //English distance

{

int feet;

float inches;

};

////////////////////////////////////////////////////////////////

struct Room //rectangular area

{

Distance length; //length of rectangle

Distance width; //width of rectangle

};

////////////////////////////////////////////////////////////////

int main()

{

Room dining; //define a room

dining.length.feet = 13; //assign values to room

dining.length.inches = 6.5;

dining.width.feet = 10;

dining.width.inches = 0.0;

//convert length & width

float l = dining.length.feet + dining.length.inches/12;

float w = dining.width.feet + dining.width.inches/12;

//find area and display it

cout << “Dining room area is “ << l \* w

<< “ square feet\n” ;

return 0;

}

ဒီပရိုဂရမ္ထဲမွာ Room type - dining ဆိုတဲ႔ variable တစ္ခုကို ေအာက္ပါအတိုင္း define ျပဳလုပ္ထားပါတယ္။

**Room dining; // variable dining of type Room**

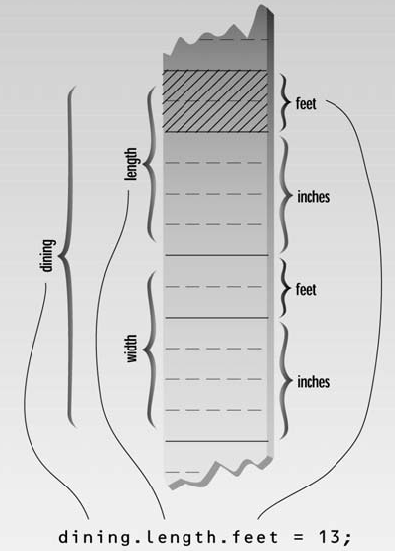
ေနာက္ပိုင္းမွာေတာ႔ အဲဒီ structure ရဲ႕ member ေတြကို တန္ဖိုးေတြ assign ျပဳလုပ္ေပးပါတယ္။

**Accessing Nested Structure Members**

Structure တစ္ခုထဲမွာ အျခား structure တစ္ခုကို ျပဳလုပ္ထားတာေၾကာင္႔ သူတို႔ရဲ႕ member ေတြကို ရယူဖို႔ dot operator ႏွစ္ခုကို အသံုးျပဳရမွာပါ။ ဥပမာ-

**dining.length.feet = 13;**

အဲဒီ statement မွာဆိုရင္ dining က structure variable ရဲ႕ name ျဖစ္ပါတယ္။ length က အျပင္က structure (Room)ရဲ႕ member name ျဖစ္ၿပီး feet ကေတာ႔ အတြင္းက structure (Distance) ရဲ႕ member name ပါ။ ကုဒ္ရဲ႕ ဆိုလိုခ်က္ အျပည္႔အစံုက dining variable ရဲ႕ length member ထဲက feet member ထဲကို တန္ဖိုး ၁၃ ထည္႔ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။ ေအာက္က ပံု (၂၀-၁) မွာ ေလ႔လာၾကည္႔ပါ။



ပံု(၂၀-၁) nested structures မ်ားအတြင္း dot operator အသံုးျပဳပံု

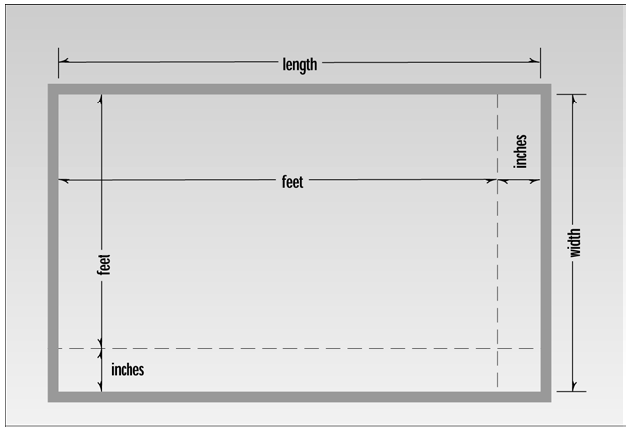
dining ရဲ႕ members ေတြကို တန္ဖိုးေတြ assign လုပ္ေပးၿပီးတာနဲ႔ ပရိုဂရမ္က ပံု(၂၀-၂) မွာ ျပထားသလို အဲဒီအခန္းရဲ႕ ၾကမ္းခင္း ဧရိယာကို တြက္ေပးမွာ ျဖစ္ပါတယ္။ ဧရိယာကို မတြက္ခင္ ပထမဆံုး Distance type ေတြျဖစ္တဲ႔ length နဲ႔ width ေတြကို feet နဲ႔ တိုင္းတာတဲ႔ float variables ေတြ ျဖစ္တဲ႔ l နဲ႔ w အျဖစ္ ေျပာင္းလဲေပးမွာ ျဖစ္ပါတယ္။ အဲဒီလို ေျပာင္းလဲဖို႔အတြက္ inches member တန္ဖိုးေတြကို 12 နဲ႔ စားၿပီး feet member ေတြထဲ ေပါင္းထည္႔ေပးရမွာပါ။

**//convert length & width**

**float l = dining.length.feet + dining.length.inches/12;**

**float w = dining.width.feet + dining.width.inches/12;**

အဲဒီလို ေပါင္းၿပီးသြားတာနဲ႔ ရလာတဲ႔ ရလဒ္ေတြဟာ float type ျဖစ္သြားပါတယ္။ ဧရိယတန္ဖိုး ရွာဖို႔အတြက္ကေတာ႔ l နဲ႔ w ကို ေျမွာက္ေပးလိုက္ရံုပဲ ျဖစ္ပါတယ္။



ပံု(၂၀-၂) အခန္းတစ္ခု၏ အတိုင္းအတာမ်ားျပပံု

**User-Defined Type Conversions**

မွတ္သားရမယ္႔ အခ်က္တစ္ခုက ပရိုဂရမ္ဟာ Distance type အတိုင္းအတာ ႏွစ္ခုကို float type variable l နဲ႔ w အျဖစ္ ေျပာင္းလဲလိုက္တာပဲ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ Room type structure ထဲမွာ သိမ္းဆည္းထားတဲ႔ (ဒီဥပမာမွာ ေနာက္ထပ္ Distance type structure ႏွစ္ခုနဲ႔ သိမ္းထားတဲ႔) အတိုင္းအတာေတြကို square feet နဲ႔ ကိုယ္စားျပဳတဲ႔ ၾကမ္းခင္းဧရိယာအျဖစ္ floating-point number တစ္ခုတည္းဘဝကို ေျပာင္းလဲေပးခဲ႔ပါတယ္။ ပရိုဂရမ္ရဲ႕ ရလဒ္ေတြကေတာ႔ ေအာက္ပါအတိုင္းပဲ ျဖစ္ပါတယ္ -

**Dining room area is 135.416672 square feet**

User-defined data types ေတြ အသံုးျပဳထားတဲ႔ ပရိုဂရမ္ေတြထဲမွာ data type ေျပာင္းလဲေပးျခင္းဟာ အေရးႀကီးတဲ႔ အခ်က္တစ္ခု ျဖစ္ပါတယ္။

**Initializing Nested Structures**

အခုလိုမ်ိဳး structure တစ္ခုကို ငံုထားတဲ႔ structure တစ္ခုထဲကို initialize လုပ္ၿပီး တန္ဖိုးေတြကို ဘယ္လို ထည္႔သြင္းၾကမွာလဲ? ေအာက္ပါ statement ေလးကေတာ႔ englarea.cpp ထဲက တန္ဖိုးေတြကို တစ္ခါတည္း initialize လုပ္ၿပီး ထည္႔သြင္းတဲ႔ နည္းလမ္း ျဖစ္ပါတယ္။

**Room dining = { {13, 6.5}, {10, 0.0} };**

Room structure ထဲမွာ ျမွပ္ႏွံထားတဲ႔ Distance type structure တစ္ခုစီကို သီးျခား initialized လုပ္ေပးဖို႔ လိုပါတယ္။ အဲဒီလို သီးျခားျပဳလုပ္ဖို႔အတြက္ တြန္႔ကြင္းေတြနဲ႔ ေကာ္မာေတြကို အသံုးျပဳရမွာပဲ ျဖစ္ပါတယ္။

ပထမ Distance ကို {13, 6.5} နဲ႔ ဒုတိယ တစ္ခုကိုေတာ႔ {10, 0.0} initialized လုပ္ပါတယ္။ အဲဒီေနာက္ ရလာတဲ႔ Distance value ႏွစ္ခုကို Room variable ထဲကို ထပ္မံ initialize လုပ္ေပးပါတယ္။ တြန္႔ကြင္းနဲ႔ ေကာ္မာေတြပဲ သံုးတာ ျဖစ္ပါတယ္။

**Depth of Nesting**

သီအိုရီအရ structures ေတြထဲမွာ အျခား structures ေတြ အဆင္႔ဆင္႔ အကန္႔အသတ္မရွိ nested လုပ္လို႔ ရပါတယ္။ အေဆာက္အဦေတြကို ဒီဇိုင္းျပလုပ္တဲ႔ ပရိုဂရမ္မ်ိဳးမွာ ေအာက္ပါ ကုဒ္မ်ိဳးေတြကို ျမင္ေတြ႔ရႏိုင္ပါတယ္။

**apartment1.laundry\_room.washing\_machine.width.feet**

**A Card Game Example**

ေနာက္ထပ္ ဥပမာေလး တစ္ခုကို သံုးၿပီး structures ေတြကို ေလ႔လာၾကည္႔ၾကရေအာင္။ ဖဲသံုးခ်ပ္ ကစားနည္းကို ျမင္ဖူးၾကမယ္ ထင္ပါတယ္။ မ်ားေသာအားျဖင္႔ ဘုရားပြဲေတြမွာ ေတြ႔ရတတ္ပါတယ္။ အဲဒီကစားနည္းမွာ cardsharp လို႔ ေခၚတဲ႔ ဖဲဝိဇၨာက ဖဲခ်ပ္ သံုးခ်ပ္ကို ျပထားၿပီး စားပြဲေပၚေမွာက္ထားလိုက္ပါတယ္။ ကၽြန္ေတာ္တို႔ မ်က္ေစ႔ေရွ႕မွာပဲ အဲဒီဖဲသံုးခ်ပ္ကို ေနရာအႀကိမ္ႀကိမ္ေျပာင္းလိုက္ပါတယ္။ ၿပီးတဲ႔အခါမွာ ကၽြန္ေတာ္တို႔ကို ဖဲတစ္ခ်ပ္ကို ေရြးခိုင္းပါတယ္။ သတ္မွတ္ထားတဲ႔ ဖဲျဖစ္ရင္ ကၽြန္ေတာ္တို႔ ႏိုင္ပါတယ္။ မွားခဲ႔ရင္ေတာ႔ ရွံႈးသြားမွာပါ။ ဖဲဝိဇၨာက ဖဲကို ေရႊ႕တာ အရမ္းကို ျမန္ၿပီး ရႈပ္ေထြးတာေၾကာင္႔ ကစားသူက မ်က္စိနဲ႔ လိုက္မမွတ္ႏိုင္ပါဘူး။ ဒါေၾကာင္႔ မွန္းဆဖို႔ ခက္ခဲပါတယ္။ အဲဒီကစားနည္းကို ေအာက္မွာ structure အသံုးျပဳထားတဲ႔ ပရိုဂရမ္ေလးနဲ႔ ေရးျပထားပါတယ္။ ပထမဆံုး ဖဲခ်ပ္ေတြကို ကိုယ္စားျပဳဖို႔ card ဆိုတဲ႔ structure တစ္ခုကို ဖန္တီးလိုက္ပါတယ္။

struct card

{

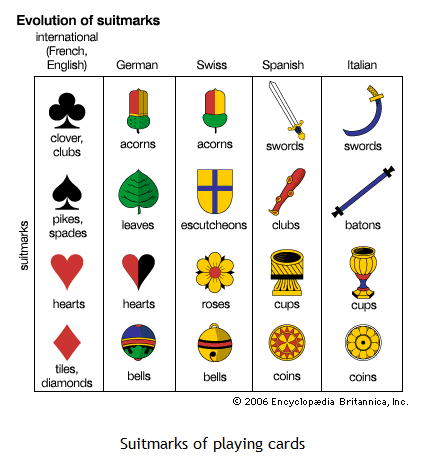
int number;

int suit;

};

အဲဒီ structure က ဖဲတစ္ခ်ပ္ကို ကိုယ္စားျပဳပါတယ္။ သူ႔ထဲမွာပါတဲ႔ number ဆိုတဲ႔ member ကေတာ႔ 2 ကေန 14 အထိ ဂဏန္းေတြကို သိမ္းေပးမွာပါ။ ဒီေနရာမွာ ဖဲထုပ္ထဲမွာ 2 ကေန 10 အထိပဲ ဂဏန္းရွိတာေၾကာင္႔ 11,12,13 နဲ႔ 14 ေတြဟာ ဂ်က္၊ ကြင္း၊ ကင္းနဲ႔ ေအ႔ ေတြကို အသီးသီး ကိုယ္စားျပဳေပးမွာပါ။ suit ကေတာ႔ 0 ကေန 3 အထိ ပါဝင္မွာ ျဖစ္ၿပီး ကလပ္၊ ဒိုင္းမြန္း၊ ဟတ္ နဲ႔ စပိတ္ ေတြကို ကိုယ္စားျပဳမွာပါ။ စာဖတ္သူတို႔ အေထြေထြ ဗဟုသုတ ရႏိုင္ေစဖို႔အတြက္ ၿဗိတိသွ်စြယ္စံုက်မ္း Encyclopædia Britannica ထဲက ေရွးေဟာင္းဖဲခ်ပ္ေလးေတြကို ကူးယူေဖာ္ျပေပးလိုက္ပါတယ္။





cards.cpp ကို ေလ႔လာၾကည္႔ပါ။

// cards.cpp

// demonstrates structures using playing cards

#include <iostream>

using namespace std;

const int clubs = 0; //suits

const int diamonds = 1;

const int hearts = 2;

const int spades = 3;

const int jack = 11; //face cards

const int queen = 12;

const int king = 13;

const int ace = 14;

////////////////////////////////////////////////////////////////

struct card

{

int number; //2 to 10, jack, queen, king, ace

int suit; //clubs, diamonds, hearts, spades

};

////////////////////////////////////////////////////////////////

int main()

{

card temp, chosen, prize; //define cards

int position;

card card1 = { 7, clubs }; //initialize card1

cout << “Card 1 is the 7 of clubs\n”;

card card2 = { jack, hearts }; //initialize card2

cout << “Card 2 is the jack of hearts\n”;

card card3 = { ace, spades }; //initialize card3

cout << “Card 3 is the ace of spades\n”;

prize = card3; //copy this card, to remember it

cout << “I’m swapping card 1 and card 3\n”;

temp = card3; card3 = card1; card1 = temp;

cout << “I’m swapping card 2 and card 3\n”;

temp = card3; card3 = card2; card2 = temp;

cout << “I’m swapping card 1 and card 2\n”;

temp = card2; card2 = card1; card1 = temp;

cout << “Now, where (1, 2, or 3) is the ace of spades? “;

cin >> position;

switch (position)

{

case 1: chosen = card1; break;

case 2: chosen = card2; break;

case 3: chosen = card3; break;

}

if(chosen.number == prize.number && // compare cards

chosen.suit == prize.suit)

cout << “That’s right! You win!\n”;

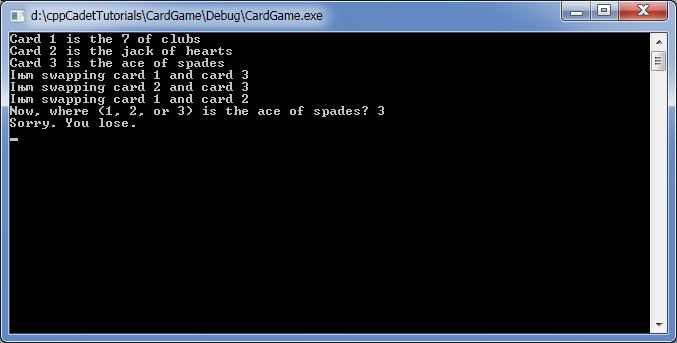
else

cout << “Sorry. You lose.\n”;

return 0;

}

ေအာက္မွာ ပရိုဂရမ္ရဲ႕ output ကို ျပထားပါတယ္-



အဲဒီ ပရိုဂရမ္မွာ အေျဖမွန္က ၂ ျဖစ္ပါတယ္။

ပရိုဂရမ္ရဲ႕ အစမွာ face card ေတြနဲ႔ suit တန္ဖိုးေတြ ထည္႔သြင္းဖို႔ const int variables ေတြ ဖန္တီးပါတယ္။ အဲဒီ variables ေတြ အကုန္လံုးကိုေတာ႔ ဒီပရိုဂရမ္ထဲမွာ အသံုးမျပဳထားပါဘူး။ ဒါေပမယ္႔ ျပည္႔စံုေအာင္ ထည္႔ေပးထားတဲ႔ သေဘာပါ။ card structure ကို သတ္မွတ္ၿပီးတဲ႔ အခါမွာ အဲဒီ card structure အမ်ိဳးအစား variable သံုးခုျဖစ္တဲ႔ temp, chosen နဲ႔ prize တို႔ကို initialize မလုပ္ဘဲ ဖန္တီးလိုက္ပါတယ္။ ေနာက္ပိုင္း ဖဲခ်ပ္ေတြ ေရႊ႕ဖို႔နဲ႔ ကစားသမား ေရြးခ်ယ္တဲ႔ ဖဲခ်ပ္မွန္မမွန္ တိုက္ဆိုင္ စစ္ေဆးတဲ႔ ေနရာမွာ အသံုးျပဳဖို႔ပဲ ျဖစ္ပါတယ္။

ကစားတဲ႔ ေနရာမွာ အသံုးျပဳတဲ႔ ဖဲသံုးခ်ပ္ အတြက္ကေတာ႔ ေနာက္ထပ္ card1, card2 နဲ႔ card3 ဆိုတဲ႔ card structure သံုးခုကို တန္ဖိုးေတြနဲ႔ initialized လုပ္ေပးလိုက္ပါတယ္။ အဲဒီေနာက္မွာ ကစားသမား သိရွိႏိုင္ေစရန္ ရည္ရြယ္ၿပီး အဲဒီ ဖဲခ်ပ္ေတြကို print out ထုတ္ေပးလိုက္ပါတယ္။ ကစားသူမွ ခန္႔မွန္း ေရြးခ်ယ္ရမယ္႔ ဖဲခ်ပ္ကို မွတ္သားထားဖို႔ အတြက္ prize ဆိုတဲ႔ variable ကို အသံုးျပဳပါတယ္။

ပရိုဂရမ္က ဖဲခ်ပ္ေတြကို ေနရာေရႊ႕ေျပာင္းပါတယ္။ ပထမဖဲကို တတိယဖဲနဲ႔ လဲပါတယ္။ ေနာက္ ဒုတိယဖဲကို တတိယဖဲနဲ႔ လဲပါတယ္။ ေနာက္တစ္ခါမွာေတာ႔ ပထမဆံုးဖဲကို ဒုတိယဖဲ နဲ႔ ေနရာလဲလိုက္ပါတယ္။ ေနရာလဲသမွ်ကိုလည္း ကစားသူကို အသိေပးေနမွာပါ။ (ဒီပရိုဂရမ္ေလးမွာေတာ႔ နမူနာအေနနဲ႔ ေရးျပထားတာျဖစ္လို႔ လြယ္ကူစြာ ခန္႔မွန္းႏိုင္ေပမယ္႔ တကယ္တမ္း ရာနဲ႔ခ်ီၿပီး ေနရာလဲလိုက္မယ္ အသိေပးခ်က္ေတြကိုလဲ အခ်ိန္အနည္းငယ္သာ ျပသေပးလိုက္မယ္ဆိုရင္ ခန္႔မွန္းဖို႔ ခက္သြားမွာ ျဖစ္ပါတယ္။)

ေနာက္ဆံုးမွာ ကစားသူကို သတ္မွတ္ထားတဲ႔ဖဲခ်ပ္ ရွိတဲ႔ ေနရာကို ခန္႔မွန္းခိုင္းပါတယ္။ အဲဒီတန္ဖိုးကို chosen ထဲကို ထည္႔ပါတယ္။ မွန္မွန္ကို သိဖို႔ prize card နဲ႔ ႏိႈင္းယွဥ္ စစ္ေဆးပါတယ္။ တကယ္လို႔ chosen နဲ႔ prize တန္ဖိုးေတြ တူခဲ႔ရင္ ကစားသူ အႏိုင္ရရွိမွာ ျဖစ္ၿပီး မတူခဲ႔ရင္ေတာ႔ ရႈံးနိမ္႔မွာ ျဖစ္ပါတယ္။

ဖဲခ်ပ္ေတြ လဲလွယ္ဖို႔အတြက္ကေတာ႔ structures ေတြ ျဖစ္ေပမယ္႔ assignment operator ကိုပဲ သံုးၿပီး အလြယ္တကူ ျပဳလုပ္ထားပါတယ္။ temp = card3; card3 = card1; card1 = temp;

ဒီေနရာမွာ ျပႆနာတစ္ခုက structure ေတြကို ေပါင္းလို႔မရသလို == operator ကိုသံုးၿပီး ( chosen == prize ဆိုတဲ႔ ပံုစံမ်ိဳး) အလြယ္တကူ ႏိႈင္းယွဥ္လို႔ မရႏိုင္တာ ျဖစ္ပါတယ္။ ဒီျပႆနာကို Class အေၾကာင္း ေလ႔လာတဲ႔အခါမွာ ေတြ႔ရွိရမယ္႔ operator overloading နည္းလမ္းနဲ႔ ေျဖရွင္းပံုကို ေနာက္ပိုင္းမွာ ဆက္လက္ ရွင္းျပေပးသြားမွာ ျဖစ္ပါတယ္။

**Structures and Classes**

ၾကံဳႀကိဳက္လာတဲ႔အတြက္ structure နဲ႔ class အေၾကာင္း အနည္းငယ္ ရွင္းျပလိုပါတယ္။ sturcture ေတြကို data ေတြ စုစည္းဖို႔ အသံုးျပဳၿပီး class ေတြကိုေတာ႔ data ေတြနဲ႔ function ေတြ စုစည္းထားဖို႔ အသံုးျပဳတယ္လို႔ ေျပာခဲ႔ပါတယ္။ C မွာေတာ႔ structure ထဲမွာ data ပဲ ထည္႔လို႔ ရပါတယ္။ ဒါေပမယ္႔ C++ မွာ structures ေတြကို data ေတြသာမကာ function ေတြကိုပါ ထည္႔သြင္းႏိုင္ပါတယ္။ သို႔ေသာ္ C++ ပရိုဂရမ္မာ အမ်ားစုဟာ structures ေတြကို data ေတြအတြက္ပဲ သီးသန္႔သံုးေလ႔ ရွိၿပီး function ေတြ ထည္႔သံုးဖို႔လိုလာတိုင္း classes ေတြကိုသာ အသံုးျပဳတတ္ၾကပါတယ္။ ဒါေၾကာင္႔ စာေရးသူအေနနဲ႔လဲ structures ေတြကို data ေတြ စုစည္းဖို႔ အသံုးျပဳနည္းမ်ားကိုသာ ဆက္လက္တင္ျပေပးသြားမွာ ျဖစ္ပါတယ္။

**Enumerations**

ၿပီးခဲ႔တဲ႔ သင္ခန္းစာေတြမွာ user-defined data types အျဖစ္ structures မ်ားအား အသံုးျပဳနည္းကို ေလ႔လာခဲ႔ၾကပါတယ္။ ယခု အပတ္မွာေတာ႔ emumeration ကို အသံုးျပဳၿပီး ကိုယ္ပိုင္ data types မ်ား ဖန္တီးပံုကို ေနာက္ထပ္ နည္းလမ္းတစ္ခုအေနနဲ႔ ေဆြးေႏြးလိုပါတယ္။ ဒီနည္းလမ္းက structures ေတြေလာက္ေတာ႔ အေရးမႀကီးပါဘူး။ C++ ပရိုဂရမ္ေတြကို enumerations အေၾကာင္းမသိပဲ ေကာင္းေကာင္းေရးႏိုင္ပါတယ္။ ဒါေပမယ္႔ enumerations ေတြအေၾကာင္းကို သိရွိ အသံုးခ်ႏိုင္မယ္ဆိုရင္ ပိုမိုရွင္းလင္း လြယ္ကူၿပီး C++ ပိုဆန္တဲ႔ ပရိုဂရမ္ေတြကို ေရးသားႏိုင္လာမွာ ျဖစ္ပါတယ္။

**Days of the Week**

မ်ားေသာအားျဖင္႔ အတိအက် သိရွိထားၿပီးသား data အနည္းငယ္သာ ပါဝင္တဲ႔ data type တစ္ခုကို ဖန္တီးဖို႔ enumeration ကို အသံုးျပဳေလ႔ ရွိၾကပါတယ္။ ဥပမာအေနနဲ႔ တစ္ပတ္စာ ေန႔ေတြကို enumeration လုပ္ထားတဲ႔ dayenum.cpp ကို ေလ႔လာၾကရေအာင္ -

// dayenum.cpp

// demonstrates enum types

#include <iostream>

using namespace std;

//specify enum type

enum days\_of\_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

int main()

{

days\_of\_week day1, day2; //define variables

//of type days\_of\_week

day1 = Mon; //give values to

day2 = Thu; //variables

int diff = day2 - day1; //can do integer arithmetic

cout << “Days between = “ << diff << endl;

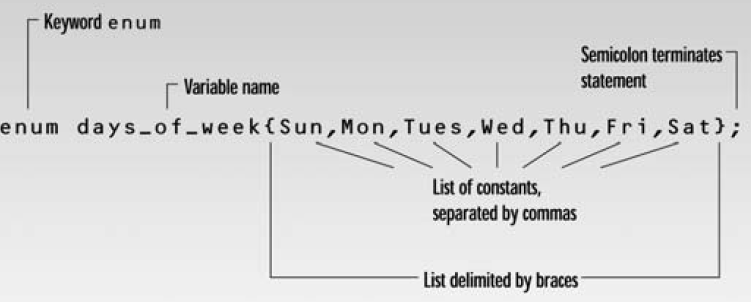
if(day1 < day2) //can do comparisons

cout << “day1 comes before day2\n”;

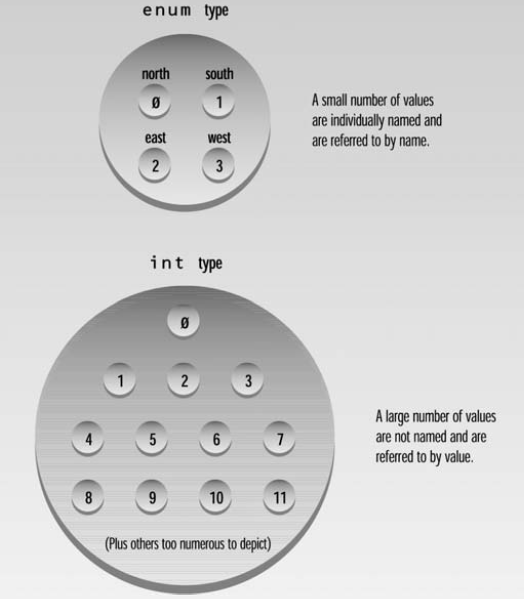
return 0;

}

ပရိုဂရမ္းမင္းတိုင္းမွာ ကၽြန္ေတာ္တို႔ သတ္မွတ္ခ်င္တဲ႔အရာေတြကို မ်ားေသာအားျဖင္႔ int တန္ဖိုး တစ္ခုခု သတ္မွတ္ေပးၿပီး ကြန္ပ်ဴတာကို ေျပာျပေလ႔ ရွိပါတယ္။ ဥပမာ Sunday ဆိုတာကို ကြန္ပ်ဴတာ နားလည္ႏိုင္ဖို႔ int Sunday=0; စသျဖင္႔ သတ္မွတ္ေလ႔ ရွိပါတယ္။ ဒါေၾကာင္႔ ေနာက္ပိုင္း 0 လို႔ ေျပာတိုင္း ကြန္ပ်ဴတာက Sunday လို႔ လက္ခံၿပီး အလုပ္လုပ္ေပးပါတယ္။ ဒါေပမယ္႔ အဲဒီလို အခ်က္အလက္ေတြ မ်ားလာရင္ ကြန္ပ်ဴတာက မွတ္မိေပမယ္႔ လူကေတာ႔ လိုက္မမွတ္ႏိုင္ေတာ႔ပါဘူး။ တခါတေလ အယူအဆေတြ လြဲမွားသြားႏိုင္ပါတယ္။ Enumeration ကို အသံုးျပဳျခင္းအားျဖင္႔ ကၽြန္ေတာ္တို႔ ထားခ်င္တဲ႔ သတ္မွတ္ခ်က္ေတြကို မူရင္း အေခၚအေဝၚအတိုင္း ေရးသား အသံုးျပဳႏိုင္တာေၾကာင္႔ အဲဒီ အခက္အခဲကို ေက်ာ္လႊားႏိုင္ပါတယ္။ အထက္ပါ ပရိုဂရမ္ေလးမွာပါတဲ႔ days\_of\_week ဆိုတဲ႔ enum type မွာ Sun, Mon, Tues, Wed, Thu, Fri, Sat စသျဖင္႔ enumerators (၇) ခု ပါဝင္ၿပီး သူတို႔ရဲ႕ syntax ကို ေအာက္မွာ ေလ႔လာႏိုင္ပါတယ္။



Enumeration ဆိုတာ ျဖစ္ႏိုင္ေျခ ရွိတဲ႔ တန္ဖိုးအားလံုးကို စုစည္းၿပီး စာရင္းျပဳစုထားတာပဲ ျဖစ္ပါတယ္။ အဲဒီ တန္ဖိုး တစ္ခုခ်င္းစီကိုလည္း သီးသန္႔ အမည္မ်ား ေပးထားရပါတယ္။ အထက္မွာ ေဖာ္ျပထားသလို enum တစ္ခုကို declare လုပ္ၿပီးတာနဲ႔ အဲဒီ type အမ်ိဳးအစား variables ေတြကို define လုပ္ႏိုင္ၿပီ ျဖစ္ပါတယ္။ dayenum.cpp မွာ days\_of\_week day1, day2; ဆိုၿပီး day1 နဲ႔ day2 variables ေတြကို define လုပ္ျပထားပါတယ္။ ( C မွာတုန္းကေတာ႔ enum ဆိုတဲ႔ keyword ကို define လုပ္တဲ႔ အခါမွာလည္း ထည္႔ေပးရပါတယ္။ ဥပမာ- enum days\_of\_week day1, day2; ဒါေပမယ္႔ C++ မွာေတာ႔ ထည္႔ေပးစရာ မလိုေတာ႔ပါဘူး။)



day1 နဲ႔ day2 လိုမ်ိဳး enumerated type ေတြကို သက္ဆိုင္ရာ enum declaration မွာ ပါဝင္တဲ႔ တန္ဖိုးေတြထဲက တစ္ခုခုကို သတ္မွတ္ေပးႏိုင္ပါတယ္။ ဒီဥပမာမွာေတာ႔ Mon နဲ႔ Thu ဆိုၿပီး ေပးထားခဲ႔ပါတယ္။ Declaration ထဲမွာ မပါတဲ႔ တန္ဖိုးေတြကိုေတာ႔ သတ္မွတ္ မေပးႏိုင္ပါဘူး။ ဥပမာ day1 = halloween; ဆိုၿပီး ေပးလို႔ မရပါဘူး။ ကၽြန္ေတာ္တို႔အေနနဲ႔ enumerated type ေတြကို standared arithmetic operators (+,-,\*,/) ေတြကို သံုးၿပီး အေပါင္း၊ အႏုတ္၊ အေျမာက္၊ အစား လုပ္ႏိုင္ပါတယ္။ ပရိုဂရမ္ထဲမွာ တန္ဖိုးႏွစ္ခုရဲ႕ ျခားနားျခင္းကို ရွာျပထားပါတယ္။ Comparison operators (<, >, <=, >=, !) ေတြကို အသံုးျပဳၿပီး တစ္ခုနဲ႔ တစ္ခုကိုလည္း ႏိႈင္းယွဥ္ႏိုင္ပါတယ္။ ပရိုဂရမ္ကို run ၾကည္႔မယ္ဆိုရင္ ေအာက္ပါ ရလဒ္ေတြကို ေတြ႔ရမွာ ျဖစ္ပါတယ္။

Days between = 3

day1 comes before day2

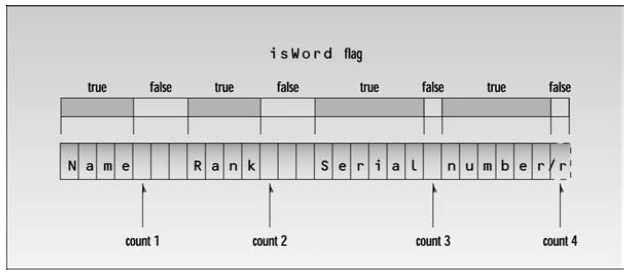
Enum types ေတြကို Arithmetic နဲ႔ relational operators ေတြသံုးလို႔ရေပမယ္႔ တစ္ခ်ိဳ႕ေနရာေတြမွာ အဓိပၸါယ္မရွိတာေၾကာင္႔ အသံုးမျပဳသင္႔ပါဘူး။ ဥပမာ အေနနဲ႔ ေအာက္က enum type declaration မွာ enum pets { cat, dog, hamster, canary, ocelot }; ဆိုၿပီး ေၾကျငာထားတယ္ ဆိုပါစို႔။ ဒီေနရာမွာ (dog + canary) နဲ႔ (cat < hamster) ဆိုတဲ႔ statements ေတြဟာ အဓိပၸါယ္ မရွိပါဘူး။

Enum types ေတြကို arithmetic နဲ႔ relational operations ေတြ လုပ္ေဆာင္ႏိုင္တာဟာ တကယ္ေတာ႔ သူတို႔ကို integers ေတြ အေနနဲ႔ internally သတ္မွတ္ယူဆထားလို႔ပဲ ျဖစ္ပါတယ္။ ဘာမွ သတ္မွတ္ မေပးခဲ႔ဘူးဆိုရင္ ပထမဆံုး တန္ဖိုးကို 0 သတ္မွတ္ေပးမွာ ျဖစ္ပါတယ္။ ေနာက္က တန္ဖိုးေတြကို တစ္ တိုးၿပီး 1, 2, 3, .. စသျဖင္႔ သတ္မွတ္ေပးမွာပါ။ ဒါေၾကာင္႔ dayenum.cpp ထဲက Sun ကေန Sat အထိ တန္ဖိုးေတြဟာ တကယ္ေတာ႔ integer တန္ဖိုး 0-6 ပဲ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ enum types ေတြကို arithmetic operation ေတြ ျပဳလုပ္တဲ႔ အခါမွာ အဲဒီ integer တန္ဖိုးေတြကုိ အသံုးျပဳၿပီး လုပ္ေဆာင္တာ ျဖစ္ပါတယ္။

ကြန္ပ်ဴတာက enum ေတြကို integers ေတြ အေနနဲ႔ သိေပမယ္႔ ကၽြန္ေတာ္တို႔ကေတာ႔ အဲဒီအခ်က္ကို သတိထားၿပီး အသံုးျပဳရမွာ ျဖစ္ပါတယ္။ day1 = 5; ဆိုၿပီး ေရးခဲ႔မိရင္ compiler က compile လုပ္ေပးေပမယ္႔ warning message ထုတ္ေပးမွာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ တတ္ႏိုင္သမွ် enums ေတြဟာ (တကယ္ေတာ႔) integers ေတြ ျဖစ္တယ္ဆိုတဲ႔ အခ်က္ကို ေမ႔ထားရမွာပါ။

**One Thing or Another**

ေနာက္ထပ္ ဥပမာအေနနဲ႔ user မွ ရိုက္ထည္႔လိုက္တဲ႔ စကားလံုးေတြကို ေရတြက္တဲ႔ ပရိုဂရမ္ေလး တစ္ပုဒ္ကို ထပ္မံေလ႔လာမွာ ျဖစ္ပါတယ္။ ယခင္ ဥပမာ တစ္ခုျဖစ္တဲ႔ chcount.cpp မွာ spaces မ်ားကို ေရတြက္ျခင္းျဖင္႔ စကားလံုး အေရအတြက္ကို ခန္႔မွန္းတြက္ခ်က္ခဲ႔ပါတယ္။ တကယ္လို႔ user က space တစ္ခုထက္ပိုၿပီး ေရးသားခဲ႔မယ္ဆိုရင္ စာလံုးအေရအတြက္ မွားယြင္းသြားမွာ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ ယခု ပရိုဂရမ္မွာ space မဟုတ္တဲ႔ အကၡရာေတြအဆံုးမွာ space ကို ေတြ႔မွ စကားလံုးကို ေရတြက္တဲ႔ စနစ္ကို အသံုးျပဳထားပါတယ္။ အဲဒီနည္းစနစ္ကို ေအာက္က ပံုေလးမွာ ရွင္းျပထားပါတယ္။



ဒါေၾကာင္႔ စကားလံုးေတြၾကားမွာ spaces ေတြ အေျမာက္အမ်ားပါေနတာေၾကာင္႔ ေရတြက္မႈ မွားယြင္းတာမ်ိဳး ရွိေတာ႔မွာ မဟုတ္ပါဘူး။ wdcount.cpp မွာ အဲဒီလို ျပဳလုပ္ဖို႔ enumerators ႏွစ္ခု (NO, YES) သာပါတဲ႔ enum (itsaWord) ကို အသံုးျပဳထားပါတယ္။ ေလ႔လာၾကည္႔ၾကရေအာင္ -

// wdcount.cpp

// demonstrates enums, counts words in phrase

#include <iostream>

#include <conio.h>

using namespace std;

#include <conio.h> //for getche()

enum itsaWord { NO, YES }; //NO=0, YES=1

int main()

{

itsaWord isWord = NO; //YES when in a word,

//NO when in whitespace

char ch = ‘a’; //character read from keyboard

int wordcount = 0; //number of words read

cout << “Enter a phrase:\n”;

do {

ch = getche(); //get character

if(ch==’ ‘ || ch==’\r’) //if white space,

{

if( isWord == YES ) //and doing a word,

{ //then it’s end of word

wordcount++; //count the word

isWord = NO; //reset flag

}

} //otherwise, it’s

else //normal character

if( isWord == NO ) //if start of word,

isWord = YES; //then set flag

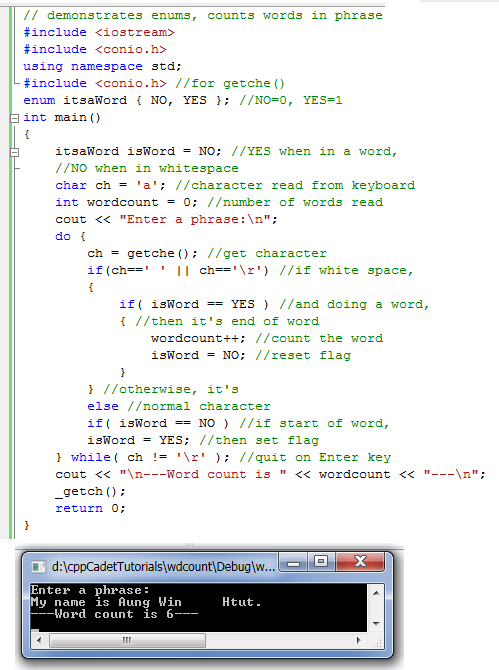
} while( ch != ‘\r’ ); //quit on Enter key

cout << “\n---Word count is “ << wordcount << “---\n”;

\_getch();

return 0;

}



ပရိုဂရမ္ အစမွာ do loop ကို အသံုးျပဳၿပီး keyboard က အကၡရာေတြကို ဖတ္ရႈပါတယ္။ space မဟုတ္တဲ႔ အကၡရာေတြကို ေက်ာ္သြားၿပီး space ကို စေတြ႔တာနဲ႔ အေရွ႕က ဖတ္ခဲ႔တဲ႔ အကၡရာဟာ စကားလံုး ဟုတ္မဟုတ္ကို enum variable - isWord ကို YES ျဖစ္မျဖစ္နဲ႔ စစ္ၿပီး ဟုတ္ခဲ႔ရင္ cont ကို ၁ တိုးလိုက္ပါတယ္။ အဲဒီေနာက္ isWord ကို NO ေျပာင္းလိုက္ပါတယ္။ ဒါမွ ေနာက္ထပ္ space ေတြ ကပ္လ်က္ ရွိေနခဲ႔ရင္ count မလုပ္မွာ ျဖစ္ပါတယ္။ enum - itsaWord မွာ ျဖစ္ႏိုင္ေျခရွိတဲ႔ တန္ဖိုး ႏွစ္ခု (NO နဲ႔ YES) ပဲ ရွိပါတယ္။ NO ကိုေရွ႕ကထားတာ သတိထားမိပါလိမ္႔မယ္။ ဒါေၾကာင္႔ သူ႔ရဲ႕တန္ဖိုးဟာ 0 (false ရဲ႕ သေကၤတ) ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ if(isWord == YES) အစား if(isWord) ဆိုၿပီး စစ္ေဆးႏိုင္ပါတယ္။ တခါတရံ အေျခအေနေပၚ မူတည္ၿပီး အဲဒီေနရာမွာ enum အစား bool variable ကို သံုးႏိုင္ပါေသးတယ္။

**Organizing the Cards**

အခု enum types ေတြကို ရွင္းျပဖို႔ ေနာက္ဆံုး ဥပမာေလး တစ္ခုကို ေလ႔လာၾကရေအာင္။ ယခင္ သင္ခန္းစာမွာ ပါတဲ႔ cards.cpp ပရိုဂရမ္ေလးကို မွတ္မိဦးမယ္ ထင္ပါတယ္။ အဲဒီမွာ card suits ေတြကို const int ေတြသံုးၿပီး ေအာက္ပါအတိုင္း သတ္မွတ္ခဲ႔ပါတယ္။

const int clubs = 0;

const int diamonds = 1;

const int hearts = 2;

const int spades = 3;

ဒီလို ေရးနည္းထက္ ပိုေကာင္းတဲ႔ နည္းလမ္းကို ေအာက္က cardenum.cpp မွာ ေလ႔လာၾကရေအာင္ -

// cardenum.cpp

// demonstrates enumerations

#include <iostream>

using namespace std;

const int jack = 11; //2 through 10 are unnamed integers

const int queen = 12;

const int king = 13;

const int ace = 14;

enum Suit { clubs, diamonds, hearts, spades };

////////////////////////////////////////////////////////////////

struct card

{

int number; //2 to 10, jack, queen, king, ace

Suit suit; //clubs, diamonds, hearts, spades

};

////////////////////////////////////////////////////////////////

int main()

{

card temp, chosen, prize; //define cards

int position;

card card1 = { 7, clubs }; //initialize card1

cout << “Card 1 is the seven of clubs\n”;

card card2 = { jack, hearts }; //initialize card2

cout << “Card 2 is the jack of hearts\n”;

card card3 = { ace, spades }; //initialize card3

cout << “Card 3 is the ace of spades\n”;

prize = card3; //copy this card, to remember it

cout << “I’m swapping card 1 and card 3\n”;

temp = card3; card3 = card1; card1 = temp;

cout << “I’m swapping card 2 and card 3\n”;

temp = card3; card3 = card2; card2 = temp;

cout << “I’m swapping card 1 and card 2\n”;

temp = card2; card2 = card1; card1 = temp;

cout << “Now, where (1, 2, or 3) is the ace of spades? “;

cin >> position;

switch (position)

{

case 1: chosen = card1; break;

case 2: chosen = card2; break;

case 3: chosen = card3; break;

}

if(chosen.number == prize.number && //compare cards

chosen.suit == prize.suit)

cout << “That’s right! You win!\n”;

else

cout << “Sorry. You lose.\n”;

return 0;

}

ဒီပရိုဂရမ္မွာ suits ေတြအတြက္ definitions ေတြကို enum အသံုးျပဳၿပီး ေအာက္ပါအတိုင္း declare လုပ္ထားပါတယ္။

**enum Suit { clubs, diamonds, hearts, spades };**

ဒီနည္းဟာ const variables ေတြ အသံုးျပဳတာထက္ ပိုၿပီး ရွင္းလင္းပါတယ္။ တကယ္လို႔ card1.suit = 5; ဆိုၿပီး သံုးခဲ႔ရင္ေတာ႔ ကြန္ပိုင္လာက သတိေပးတာ ခံရမွာ ျဖစ္ၿပီး မသံုးသင္႔ပါဘူး။

**Specifying Integer Values**

ကၽြန္ေတာ္တို႔အေနနဲ႔ အေစာပိုင္းမွာ ပထမဆံုး enumerator ကို 0 သတ္မွတ္ထားၿပီး ေနာက္ထပ္ enumerator ေတြကို တစ္တိုးၿပီး 1, 2, 3,.. စသျဖင္႔ သတ္မွတ္ေပးလိမ္႔မယ္လို႔ ေျပာခဲ႔ပါတယ္။ ဒါေပမယ္႔ ကိုယ္ႀကိဳက္ႏွစ္သက္ရာ တန္ဖိုးနဲ႔ အစျပဳဖို႔အတြက္လည္း equal sign (=) ကို အသံုးျပဳႏိုင္ပါတယ္။ ဥပမာ suits ေတြရဲ႕ တန္ဖိုးကို 0 နဲ႔ အစမျပဳဘဲ 1 နဲ႔ စခ်င္တယ္ဆိုပါစို႔။

enum Suit { clubs=1, diamonds, hearts, spades }; ဆိုၿပီး ေရးသားႏိုင္ပါတယ္။ ေနာက္က လာတဲ႔ တန္ဖိုးေတြကေတာ႔ ထံုးစံအတိုင္း တစ္တိုးၿပီး 2, 3, 4,.. စသျဖင္႔ ျဖစ္သြားမွာပါ။ တကယ္တမ္းမွာ တန္ဖိုးတစ္ခုခ်င္းစီကို equal sign (=) အသံုးျပဳၿပီး ထည္႔သြင္းေပးလိုက္ႏိုင္ပါတယ္။

**Not Perfect**

eunm ရဲ႕ အားနည္းခ်က္ကေတာ႔ ၄င္းကို C++ ရဲ႕ I/O statements ေတြက နားမလည္တာပဲ ျဖစ္ပါတယ္။ ဥပမာ ေအာက္က ကုဒ္ေတြကို run မယ္ဆိုပါစို႔

enum direction { north, south, east, west };

direction dir1 = south;

cout << dir1;

ကၽြန္ေတာ္တို႔ကေတာ႔ output ဟာ south ျဖစ္လိမ္႔မယ္လို႔ ထင္ၾကမွာပါပဲ။ ဒါေပမယ္႔ C++ I/O က enum variables ေတြကို integers ေတြအေနနဲ႔ပဲ ကိုင္တြယ္တာ ျဖစ္လို႔ output ကို 1 အျဖစ္ပဲ ထုတ္ေပးမွာပါ။

**Other Examples**

ေနာက္ထပ္ enum အသံုးျပဳနည္းမ်ားကို ေအာက္မွာ ဥပမာေလးမ်ားနဲ႔ ျပသေပးထားပါတယ္။ ေလ႔လာၾကည္႔ၾကပါ။

enum months { Jan, Feb, Mar, Apr, May, Jun,

Jul, Aug, Sep, Oct, Nov, Dec };

enum switch { off, on };

enum meridian { am, pm };

enum chess { pawn, knight, bishop, rook, queen, king };

enum coins { penny, nickel, dime, quarter, half-dollar, dollar };

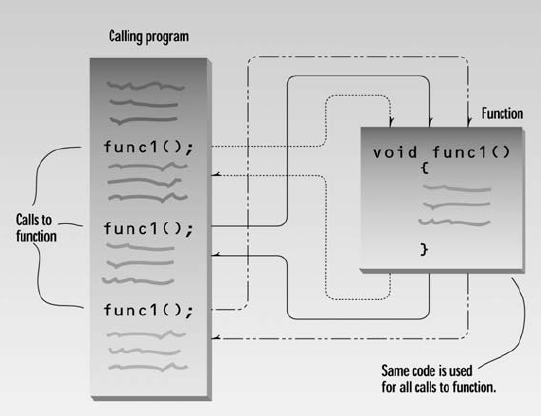
**အခန္း(၄)**

Functions

**Functions**

Function ဆိုသည္မွာ ပရိုဂရမ္ statements ကုဒ္ အခ်ိဳ႕ကို ယူနစ္ တစ္ခုအျဖစ္ စုစည္းၿပီး နာမည္တစ္ခု ေပးထားျခင္းပင္ ျဖစ္သည္။ အဆိုပါ ယူနစ္မ်ားကို ပရိုဂရမ္၏ အျခားအစိတ္အပိုင္းမ်ားမွ ေခၚယူ အသံုးျပဳႏိုင္ပါသည္။ Functions မ်ားကို အသံုးျပဳရျခင္း၏ အဓိက အေၾကာင္းရင္းမွာ ပရိုဂရမ္တစ္ပုဒ္အား conceptual organization ျပဳလုပ္ရာတြင္ အေထာက္အကူျပဳရန္ပင္ ျဖစ္သည္။ ထို႔ျပင္ ပရိုဂရမ္ တစ္ပုဒ္အား functions မ်ားအျဖစ္ ခြဲျခား ေရးသားျခင္းသည္ structured programming ၏ အဓိက ဥပေဒသ တစ္ခုလည္း ျဖစ္ပါသည္။ (သို႔ရာတြင္ object-oriented programming တြင္ ၄င္းထက္ ပိုမို အားေကာင္းေသာ နည္းလမ္းမ်ားကို အသံုးျပဳႏိုင္ပါသည္။)

ေနာက္ထပ္ ရည္ရြယ္ခ်က္ တစ္ခုမွာ ပရိုဂရမ္၏ အရြယ္အစားကို ေလ်ာ႔ခ်ရန္ ျဖစ္သည္။ တစ္ႀကိမ္ထက္ ပိုသံုးထားေသာ ကုဒ္မ်ားကို function အျဖစ္ ေျပာင္းလဲ အသံုးျပဳသင္႔ေပသည္။ Functions မ်ားကို ပရိုဂရမ္အတြင္း အႀကိမ္မ်ားစြာ ေခၚယူ အသံုးျပဳေလ႔ ရွိေသာ္လည္း function ၏ ကုဒ္မ်ားကိုမူ memory ၏ ေနရာ တစ္ခုတည္းတြင္သာ သိမ္းဆည္းထားျခင္း ျဖစ္သည္။ ပရိုဂရမ္၏ အစိတ္အပိုင္း အသီးသီးမွ function တစ္ခုအား ေခၚယူ အသံုးျပဳပံုကို ေအာက္ပါအတိုင္း သရုပ္ေဖာ္ျပသထားပါသည္။



C++ ႏွင္႔ C တြင္ အသံုးျပဳရေသာ functions မ်ားသည္ အျခား ပရိုဂရမ္းမင္း ဘာသာစကားမ်ားရွိ subroutines၊ procedures မ်ားႏွင္႔ သေဘာတရားခ်င္း တူညီၿပီး ေရးသားပံု ဆင္တူေပသည္။

**Simple Functions**

ပထမဦးစြာ ၾကယ္ ၄၅ လံုး print ထုတ္ေပးေသာ function တစ္ခုကို စတင္ ေလ႔လာၾကရေအာင္။ ေအာက္ပါ table.cpp ပရိုဂရမ္သည္ ဇယားတစ္ခုကို ဖန္တီးေပးၿပီး အဆိုပါ ဇယားကို ပိုမို ဖတ္ရႈရလြယ္ကူေစရန္ ၾကယ္မ်ားျဖင္႔ ျပဳလုပ္ထားေသာ လိုင္းမ်ားျဖင္႔ တားေပးထား ပါသည္။

// table.cpp

// demonstrates simple function

#include <iostream>

#include <conio.h>

using namespace std;

void starline(); //function declaration

// (prototype)

int main()

{

starline(); //call to function

cout << “Data type Range” << endl;

starline(); //call to function

cout << “char -128 to 127” << endl

<< “short -32,768 to 32,767” << endl

<< “int System dependent” << endl

<< “long -2,147,483,648 to 2,147,483,647” << endl;

starline(); //call to function

\_getch();

return 0;

}

//--------------------------------------------------------------

// starline()

// function definition

void starline() //function declarator

{

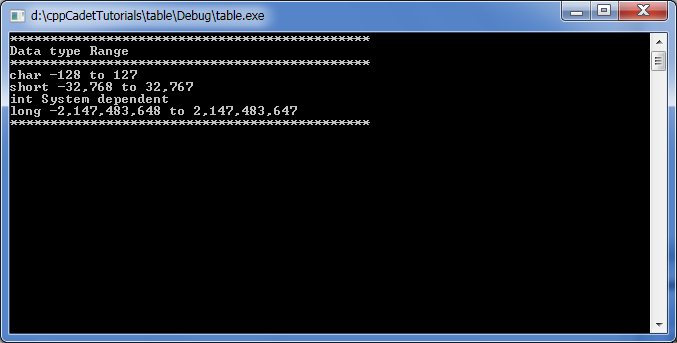
for(int j=0; j<45; j++) //function body

cout << ‘\*’;

cout << endl;

}

ပရိုဂရမ္၏ output မွာ ေအာက္ပါအတိုင္း ျဖစ္ပါသည္။



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Data type Range

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

char -128 to 127

short -32,768 to 32,767

int System dependent

long -2,147,483,648 to 2,147,483,647

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

အထက္ပါ ပရိုဂရမ္တြင္ main() ႏွင္႔ starline() ဟူေသာ functions ႏွစ္ခုပါဝင္ပါသည္။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ ယခင္ သင္ခန္းစာမ်ားတြင္ main() function တစ္ခုတည္းကိုသာ ႀကိမ္ဖန္မ်ားစြာ ေတြ႔ျမင္လာခဲ႔ၾကရပါသည္။ ထိုသို႔ ပရိုဂရမ္ တစ္ပုဒ္တြင္ functions မ်ား ထပ္မံေပါင္းထည္႔ရန္ လိုသည္႔ အခ်က္ သံုးခ်က္ ရွိပါသည္။ ၄င္းတို႔မွာ function declaration, calls to the function ႏွင္႔ function definition မ်ား ျဖစ္ၾကပါသည္။

**The Function Declaration**

ကၽြန္ေတာ္တို႔အေနျဖင္႔ compiler အား variable တစ္ခုကို ႀကိဳတင္ေၾကျငာထားျခင္း မရွိဘဲ အသံုးမျပဳႏိုင္သကဲ႔သို႔ function တစ္ခုကိုလည္း ႀကိဳတင္ေၾကျငာေပးထားရန္ လိုအပ္ေပသည္။ ထိုသို႔ ျပဳလုပ္ႏိုင္သည္႔ နည္းလမ္း ႏွစ္ခု ရွိပါသည္။ ပထမတစ္နည္းမွာ function အား မေခၚယူမီ declare ျပဳလုပ္ျခင္းျဖစ္ပါသည္။ (ဒုတိယနည္းလမ္းမွာ function အားမေခၚယူမီ define ျပဳလုပ္ျခင္း ျဖစ္ၿပီး ၄င္းကို ေနာက္ပိုင္းတြင္ ဆက္လက္ေလ႔လာသြားမည္။)

table.cpp ပရိုဂရမ္တြင္ starline() ဆိုေသာ function ကို void starline(); ဆိုေသာ ကုဒ္ျဖင္႔ declare ေၾကျငာေပးထားသည္။ ဤနည္းျဖင္႔ ေနာင္တစ္ခ်ိန္တြင္ starline ဟုေခၚေသာ function တစ္ခုကို ေခၚယူရန္ စီစဥ္ထားေၾကာင္း compiler အား အသိေပးလိုက္ျခင္းပင္ ျဖစ္သည္။

ထိုသို႔ေၾကျငာရာတြင္ ပါဝင္ေသာ void ဆိုသည္႔ keyword သည္ function မွ return value ျပန္ေပးမည္ မဟုတ္ေၾကာင္း သတ္မွတ္ေပးထားျခင္းျဖစ္ၿပီး လက္သဲကြင္းထဲတြင္ အလြတ္ထားျခင္းသည္ arguments ထည္႔သြင္းေပးရန္ မလိုေၾကာင္း (လက္ခံမည္မဟုတ္ေၾကာင္း) အသိေပးထားျခင္းပင္ ျဖစ္သည္။ (ေနာက္တစ္နည္းမွာ လက္သဲကြင္းထဲတြင္ void ထည္႔ေပးျခင္းျဖစ္ၿပီး C တြင္ အသံုးမ်ားသည္။ သို႔ရာတြင္ C++ ၌မူ လက္သဲကြင္း အလြတ္ထားေလ႔ရွိသည္။) ေနာက္ပိုင္းတြင္ arguments မ်ားနွင္႔ return values မ်ား အေၾကာင္းကို ေလ႔လာသြားမည္ျဖစ္သည္။ function declaration ကို semicolon (;) ျဖင္႔ အဆံုးသတ္ေလ႔ရွိၿပီး ၄င္းစာေၾကာင္းသည္ ၿပီးျပည္႔စံုသည္႔ statement တစ္ခုပင္ ျဖစ္သည္။

Function declarations မ်ားသည္ functions မ်ား၏ model သို႔မဟုတ္ blueprint သဖြယ္ ျဖစ္ေနသျခင္႔ ၄င္းတို႔ကို prototypes မ်ားဟုလည္း ေခၚဆိုၾကသည္။ ၄င္းတို႔သည္ ဤကဲ႔သို႔ ပံုစံရွိေသာ function တစ္ခုသည္ ပရိုဂရမ္ ၏ တစ္ေနရာတြင္ ေရးသားထားမည္ ျဖစ္ေသာေၾကာင္႔ ၄င္းအား ေခၚဆို အသံုးခ်မႈမ်ားကို အဆိုပါ function အားေရးသားထားခ်က္ မတိုင္မီ ေတြ႔ရွိရပါက နားလည္ေစရန္ compiler အား အေၾကာင္းၾကားလိုက္ျခင္းပင္ ျဖစ္သည္။ အဆိုပါ declaration တြင္ ပါဝင္ေသာ return type၊ arguments အေရအတြက္ႏွင္႔ types မ်ား စသည္႔ အခ်က္အလက္မ်ားကို function ၏ signature မ်ားဟုလည္း ေခၚဆိုေလ႔ ရွိသည္။

**Calling the Function**

အထက္ပါ ပရိုဂရမ္တြင္ starline() function အား main() function အတြင္း သံုးႀကိမ္ေခၚယူ အသံုးျပဳထားေပသည္။ ထိုသို႔ ေခၚယူရန္ starline(); ဟူေသာ ကုဒ္ကို အသံုးျပဳခဲ႔သည္။ Function တစ္ခုအား ေခၚယူရန္မွာ အဆိုပါ function ၏ အမည္ေနာက္တြင္ လက္သဲကြင္းတစ္စံု ႏွင္႔ ၄င္းတို႔ကို semicolon ျဖင္႔ ပိတ္ေပးရန္သာ ျဖစ္သည္။ Function call ၏ syntax သည္ function declaration ၏ syntax ႏွင္႔ ဆင္တူၿပီး return type မပါျခင္းသာ ကြာျခားမႈ ရွိသည္။

**void starline(); //function declaration**

**starline(); //function calling**

Call statement ကို လုပ္ေဆာင္သည္႔အခါ control သည္ function သို႔ ေရာက္ရွိသြားၿပီး (ေအာက္တြင္ ဆက္လက္ ရွင္းျပသြားမည္႔) function definition ရွိ ကုဒ္မ်ားကို လုပ္ေဆာင္မည္ျဖစ္သည္။ ထုိ႔ေနာက္ control သည္ function call ၏ ေနာက္တြင္ ရွိေသာ ကုဒ္မ်ားသို႔ ျပန္လည္ေရာက္ရွိသြားမည္ ျဖစ္သည္။

**The Function Definition**

Function တစ္ခုအတြက္ တကယ္႔ကုဒ္မ်ား ေရးသားသည္႔ ေနရာကို function definition ဟုေခၚသည္။ ေအာက္တြင္ starline() function ၏ definition ကို ေလ႔လာၾကည္႔ရေအာင္-

void starline() //declarator

{

for(int j=0; j<45; j++) //function body

cout << ‘\*’;

cout << endl;

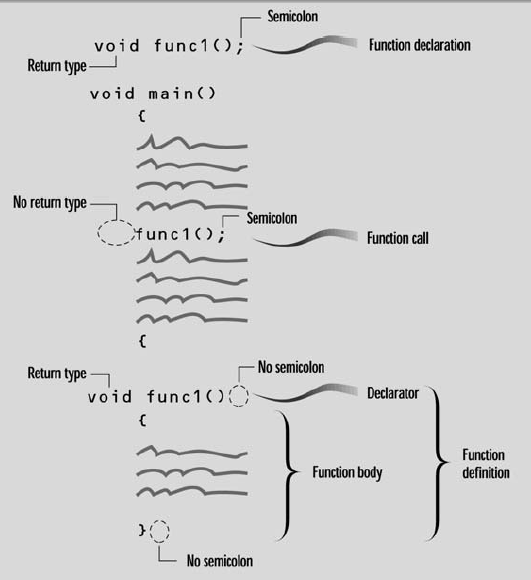
}

Function definition တြင္ declarator ဟုေခၚေသာ စာေၾကာင္းႏွင္႔ function body ဟူ၍ ပါဝင္သည္။ Function body သည္ function ၏ ကုဒ္မ်ား ပါဝင္ၿပီး တြန္႔ကြင္း အဖြင္႔အပိတ္ျဖင္႔ သတ္မွတ္ထားသည္။

The declarator တြင္ ပါဝင္ေသာ အခ်က္မ်ားသည္ declaration ႏွင္႔ ကိုက္ညီမႈ ရွိရမည္ ျဖစ္သည္။ ေအာက္ပါအခ်က္မ်ား တူညီရမည္ျဖစ္သည္-

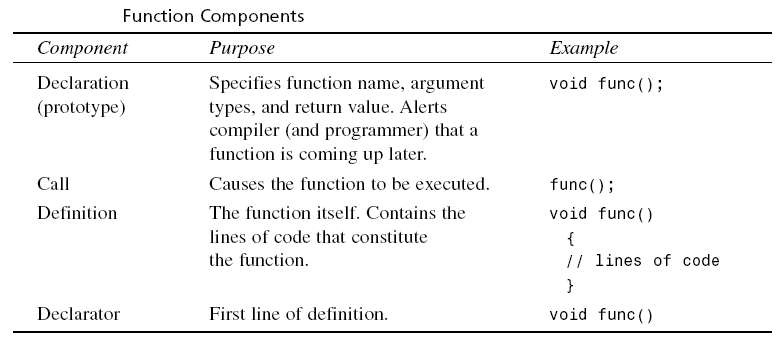
1. function name
2. same argument types in the same order (if there are arguments)
3. return type.

declarator အားsemicolon လံုးဝ မထည္႔ေပးရေပ။ ေအာက္ပါပံုတြင္ function declaration, function call ႏွင္႔ function definition တို႔၏ syntax ကို ေလ႔လာႏိုင္ပါသည္။



Function တစ္ခုအား ေခၚယူလိုက္သည္႔အခါ control သည္ function body ရွိ ပထမဆံုး ကုဒ္ကို ကူးေျပာင္းသြားၿပီး စတင္လုပ္ေဆာင္မည္ ျဖစ္သည္။ function body ရွိ ကုဒ္မ်ား အားလံုး လုပ္ေဆာင္ၿပီး၍ တြန္႔ကြင္း အပိတ္အား ေတြ႔လိုက္သည္႔အခ်ိန္တြင္ control သည္ calling ျပဳလုပ္သည္႔ ေနရာသို႔ ျပန္ေရာက္သြားမည္ ျဖစ္သည္။

ေအာက္က ဇယားမွာ function တစ္ခု ေရးသားဖို႔ လိုတဲ႔ အစိတ္အပိုင္းမ်ားကို ရွင္းျပေပးထားပါတယ္။

****

**Comparison with Library Functions**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ library functions အခ်ိဳ႕ကို အေစာပိုင္း သင္ခန္းစာမ်ားမွာ အသံုးျပဳခဲ႔ၾကၿပီး ျဖစ္ပါတယ္။ ဥပမာ ch=getche(); ဆိုတဲ႔ ကုဒ္မ်ိဳးပဲ ျဖစ္ပါတယ္။ ဒါဆိုရင္ အဲဒီ library function getche() အတြက္ declaration နဲ႔ definition ေတြက ဘယ္မွာပါလဲ? တကယ္ေတာ႔ အဲဒီလို library functions ေတြ အတြက္ declaration ေတြကို ပရိုဂရမ္ေတြရဲ႕ အစမွာ ပါတဲ႔ header file (conio.h, for getche()) ေတြထဲမွာ သတ္မွတ္ေပးထားတာ ျဖစ္ပါတယ္။ သူတို႔အတြက္ definition ေတြကေတာ႔ executable code ေတြျဖစ္ေအာင္ compile လုပ္ေပးထားၿပီး library file ေတြထဲမွာ ထည္႔သြင္းထားကာ ကၽြန္ေတာ္တို႔ ေရးသားထားတဲ႔ C++ source file ကို build လုပ္တဲ႔အခ်ိန္က်မွ compiler က အလိုအေလ်ာက္ ခ်ိတ္ဆက္ၿပီး link ျပဳလုပ္ေပးမွာ ျဖစ္ပါတယ္။

ဒါေၾကာင္႔ library function ေတြကို အသံုးျပဳတဲ႔ အခါမွာ declaration နဲ႔ definition ေတြကို ကၽြန္ေတာ္တို႔ ကိုယ္တိုင္ ေရးသားေပးစရာ မလိုအပ္ဘဲ ပရိုဂရမ္ အစပိုင္းမွာ သက္ဆိုင္ရာ header file ကို ေၾကျငာေပးၿပီး သံုးစြဲႏိုင္တာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ကၽြန္ေတာ္တို႔ ကိုယ္ပိုင္ functions ေတြ ေရးသားတဲ႔ အခါမွာေတာ႔ declaration နဲ႔ definition ေတြကို ယခင္ အပတ္က ဥပမာ ပရိုဂရမ္ေတြထဲကလိုမ်ိဳး ထည္႔သြင္းေရးသားေပးရမွာပါ။

**Eliminating the Declaration**

ေနာက္ထပ္ function ေရးနည္း တစ္မ်ိဳးကေတာ႔ function call မတိုင္မီ function declaration ကို မေရးေတာ႔ပဲ အဲဒီေနရာမွာ function definition ကို တန္းေရးလိုက္တာပဲ ျဖစ္ပါတယ္။ ေအာက္က table2.cpp ပရိုဂရမ္ေလးမွာ starline() ကို ျပန္ေရးျပထားပါတယ္။

// table2.cpp

// demonstrates function definition preceding function calls

#include <iostream>

using namespace std; //no function declaration

//--------------------------------------------------------------

// starline() //function definition omit here

void starline()

{

for(int j=0; j<45; j++)

cout << ‘\*’;

cout << endl;

}

//--------------------------------------------------------------

int main() //main() follows function

{

starline(); //call to function

cout << “Data type Range” << endl;

starline(); //call to function

cout << “char -128 to 127” << endl

<< “short -32,768 to 32,767” << endl

<< “int System dependent” << endl

<< “long -2,147,483,648 to 2,147,483,647” << endl;

starline(); //call to function

return 0;

}

ဒီနည္းက function declaration ကို ျဖဳတ္ပစ္လိုက္လို႔ ပရိုဂရမ္ တိုတိုေလးေတြ ေရးဖို႔အတြက္ ရိုးရွင္းတဲ႔ နည္းလမ္း တစ္ခုပဲ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ဒီနည္းက functions ေတြ အမ်ားအျပားရွိလာတဲ႔ အခါမွာ function တစ္ခုကေန ေနာက္တစ္ခုကို ေခၚယူႏိုင္တာ ျဖစ္တဲ႔အတြက္ အစဥ္အတိုင္း ျဖစ္ေအာင္ စဥ္းစားရတဲ႔ အတြက္ အဆင္မေျပေတာ႔ပါဘူး။ တစ္ခါတေလမွာ မျဖစ္ႏိုင္တဲ႔ အေျခအေနမ်ိဳးကိုေတာင္ ၾကံဳရတတ္ပါတယ္။ ဒါ႔အျပင္ ပရိုဂရမ္မာ အမ်ားစုဟာ ပထမဆံုး အလုပ္လုပ္မယ္႔ main() function ကို အစမွာ ထားခ်င္ၾကပါတယ္။ ဒါေၾကာင္႔ declarations ေတြ အသံုးျပဳရတဲ႔ ပထမနည္းလမ္းကိုပဲ အသံုးမ်ားၾကပါတယ္။

**Passing Arguments to Functions**

Argument ဆိုသည္မွာ ပရိုဂရမ္မွ function သို႔ ထည္႔သြင္းေပးလိုက္ေသာ data အစိတ္အပိုင္း တစ္ခုျဖစ္သည္။ ထိုိသို႔ argument မ်ား ထည္႔သြင္းေပးျခင္းျဖင္႔ function တစ္ခုကို values အမ်ိဳးမ်ိဳး ျဖင္႔ လုပ္ေဆာင္ႏိုင္ရန္ သို႔မဟုတ္ ပရိုဂရမ္၏ လိုအပ္ခ်က္အရ မတူညီေသာ လုပ္ေဆာင္ခ်က္မ်ားကို လုပ္ေဆာင္ႏိုင္ရန္ ေဆာင္ရြက္ေပးသည္။

**Passing Constants**

အထက္က ဥပမာတြင္ အသံုးျပဳခဲ႔ေသာ starline() ဆိုသည္႔ function သည္ ၾကယ္ ၄၅ လံုးကို အျမဲတမ္း print ထုတ္ေပးသျဖင္႔ လိုသလို ေျပာင္းလဲ အသံုးခ်ႏိုင္စြမ္း နည္းပါးသည္။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ ထိုကဲ႔သို႔ အေသ ေရးသားထားေသာ function မ်ားထက္ အလိုရွိေသာ character ကို အလိုရွိေသာ အေရအတြက္ print ထုတ္ေပးမည္႔ function မ်ိဳးကို ပိုမိုလိုအပ္ေပသည္။ ေအာက္ပါ tablearg.cpp ပရိုဂရမ္ တြင္ ထိုကဲ႔သို႔ေသာ function မ်ိဳးကို ေရးျပထားပါသည္။ အလိုရွိေသာ character ႏွင္႔ အလိုရွိေသာ အႀကိမ္အေရအတြက္ကိုမူ arguments မ်ားအေနျဖင္႔ ထည္႔သြင္းေပးရမည္ ျဖစ္သည္။

// tablearg.cpp

// demonstrates function arguments

#include <iostream>

using namespace std;

void repchar(char, int); //function declaration

int main()

{

repchar(‘-’, 43); //call to function

cout << “Data type Range” << endl;

repchar(‘=’, 23); //call to function

cout << “char -128 to 127” << endl

<< “short -32,768 to 32,767” << endl

<< “int System dependent” << endl

<< “double -2,147,483,648 to 2,147,483,647” << endl;

repchar(‘-’, 43); //call to function

return 0;

}

//--------------------------------------------------------------

// repchar()

// function definition

void repchar(char ch, int n) //function declarator

{

for(int j=0; j<n; j++) //function body

cout << ch;

cout << endl;

}

အဲဒီ ပရိုဂရမ္ေလးမွာ သံုးထားတဲ႔ function က repchar() ျဖစ္ၿပီး ေအာက္ပါအတိုင္း declare လုပ္ထားပါတယ္ -

**void repchar(char, int); // declaration specifies data types**

Declaration မွာ လက္သဲကြင္းထဲက char နဲ႔ int ဆိုတာကေတာ႔ အဲဒီ function ထဲကို ထည္႔သြင္းေပးလိုက္မယ္႔ data ေတြရဲ႕ data types ေတြပဲ ျဖစ္ပါတယ္။

Function call ျပဳလုပ္တဲ႔ အခါမွာေတာ႔ သတ္မွတ္ထားတဲ႔ တန္ဖိုးေတြ (ဒီေနရာမွာေတာ႔ constatns ေတြ) ကို လက္သဲကြင္းရဲ႕ သက္ဆိုင္ရာေနရာေတြမွာ ေကာ္မာ ခံၿပီး ထည္႔သြင္းေပးလိုက္မွာပါ။ ဥပမာ-

**repchar(‘-’, 43); // function call specifies actual values**

ဒီ statement ေလးက repchar() ကို dash အလံုးေရ ၄၃ လံုး print ထုတ္ေပးဖို႔ ညႊန္ၾကားလိုက္တာပဲ ျဖစ္ပါတယ္။ Function call မွာ ထည္႔သြင္းေပးလိုက္တဲ႔ data ေတြရဲ႕ type ဟာ declaration မွာ သတ္မွတ္ေပးထားတဲ႔ အတိုင္း ျဖစ္ဖို႔ေတာ႔ လိုပါတယ္။ ဒါေၾကာင္႔ ပထမဆံုး ထည္႔သြင္းေပးရမယ္႔ argument ဟာ char ျဖစ္ရမွာ ျဖစ္ၿပီး ေနာက္တစ္လံုးကေတာ႔ int ျဖစ္ရပါမယ္။ ေနာက္ထပ္ function call တစ္ခု ျဖစ္တဲ႔ repchar(‘=’, 23); ကေတာ႔ equal signs ၂၃ ခု ထုတ္ေပးဖို႔ ညႊန္ၾကားခ်က္ ျဖစ္ပါတယ္။ tablearg.cpp ရဲ႕ output ကို ေလ႔လာၾကည္႔ၾကရေအာင္။

-------------------------------------------

Data type Range

=======================

char -128 to 127

short -32,768 to 32,767

int System dependent

long -2,147,483,648 to 2,147,483,647

-------------------------------------------

Function ေတြကို ေခၚယူတဲ႔ ပရိုဂရမ္က လိုအပ္တဲ႔ arguments ေတြ (ဥပမာ '-' နဲ႔ 43) ေတြကို ထည္႔သြင္းေပးလိုက္ရမွာ ျဖစ္ပါတယ္။ ၄င္းတန္ဖိုးေတြကို function ထဲက အသံုးျပဳထားတဲ႔ parameters ေတြလို႔ ေခၚတဲ႔ variables ေတြ (ဥပမာ ch နဲ႔ n) ထဲကို ကူးယူလိုက္မွာ ျဖစ္ပါတယ္။ (ပရိုဂရမ္မာ အမ်ားစုဟာ argument နဲ႔ parameter ေတြကို အလဲအလွယ္လုပ္ၿပီး ေျပာဆို သံုးစြဲတတ္ၾကပါတယ္။)

Function definition မွာေတာ႔ လက္သဲကြင္းထဲမွာ parameters ေတြရဲ႕ data types ေတြအျပင္ names ေတြကိုပါ ထည္႔သြင္းေပးရမွာ ျဖစ္ပါတယ္။ ဥပမာ-

void repchar(char ch, int n) //declarator specifies parameter

//names and data types

အဲဒီ parameters ေတြ (ch နဲ႔ n) ကို function ထဲမွာ သာမန္ variable ေတြ သံုးသလိုပဲ သံုးစြဲႏိုင္ပါတယ္။ ဆိုလိုတာက အဲဒီလို လက္သဲကြင္းထဲမွာ ေရးလိုက္ျခင္းဟာ ေအာက္ပါအတိုင္း ေၾကျငာလိုက္တာနဲ႔ အတူတူပဲ ျဖစ္ပါတယ္။

char ch;

int n;

Funciton ကို call လုပ္တဲ႔အခါမွာေတာ႔ အဲဒီ parameters ေတြထဲကို calling program က ထည္႔သြင္းေပးလိုက္တဲ႔ တန္ဖိုးေတြ အလိုအေလ်ာက္ initialized လုပ္ေပးမွာ ျဖစ္ပါတယ္။

**Passing Variables**

ယခင္ အပတ္မွာ ေလ႔လာခဲ႔ၾကတဲ႔ tablearg.cpp ပရိုဂရမ္ထဲမွာ constant arguments (‘–’, 43) ေတြကို အသံုးျပဳခဲ႔ပါတယ္။ ယခုအပတ္မွာေတာ႔ constants ေတြအစား variables ေတြကို arguments မ်ား အျဖစ္ အသံုးျပဳျခင္းကို ေလ႔လာသြားမွာပါ။ ေအာက္ပါ vararg.cpp ပရိုဂရမ္ေလးမွာေတာ႔ tablearg.cpp မွာတုန္းကလိုပဲ repchar() function ပါဝင္မွာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ တစ္ခု ပိုထူးျခားလာတာကေတာ႔ user က ေရြးခ်ယ္ေပးလိုက္တဲ႔ character နဲ႔ အႀကိမ္အေရအတြက္ကို အသံုးျပဳသြားမွာပဲ ျဖစ္ပါတယ္။

// vararg.cpp

// demonstrates variable arguments

#include <iostream>

using namespace std;

void repchar(char, int); //function declaration

int main()

{

char chin;

int nin;

cout << “Enter a character: “;

cin >> chin;

cout << “Enter number of times to repeat it: “;

cin >> nin;

repchar(chin, nin);

return 0;

}

//--------------------------------------------------------------

// repchar()

// function definition

void repchar(char ch, int n) //function declarator

{

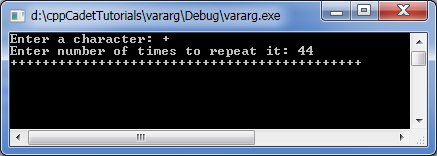
for(int j=0; j<n; j++) //function body

cout << ch;

cout << endl;

}

vararg.cpp ရဲ႕ နမူနာ output ကို ေအာက္မွာ ေလ႔လာႏိုင္ပါတယ္။



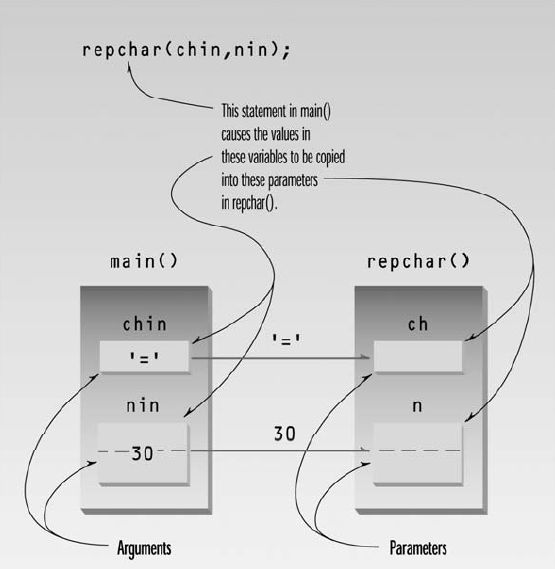
repchar() အတြက္ input arguments မ်ားအေနနဲ႔ main() ထဲက chin နဲ႔ nin တို႔ကို ေအာက္ပါအတိုင္း အသံုးျပဳထားပါတယ္။

repchar(chin, nin); // function call

Variables ေတြအတြက္ အသံုးျပဳထားတဲ႔ arguments ေတြရဲ႕ data types ေတြဟာ constants ေတြ အသံုးျပဳစဥ္ကလိုပဲ function declaration နဲ႔ definition ေတြမွာ တူညီေနဖို႔ လိုအပ္ပါတယ္။ ဒါေၾကာင္႔ chin ဟာ char ျဖစ္ဖို႔လိုၿပီး nin ကေတာ႔ int ျဖစ္ဖို႔ လိုပါတယ္။

**Passing by Value**

vararg.cpp ပရိုဂရမ္မွာ chin နဲ႔ nin အတြင္း ရွိတဲ႔ တန္ဖိုးမ်ားကို function call ျပဳလုပ္တဲ႔ အခါမွာ function ထဲကို pass ျပဳလုပ္ေပးမွာ ျဖစ္ပါတယ္။ function ထဲတြင္ ၄င္း တန္ဖိုးမ်ားကို သိမ္းဆည္းအသံုးျပဳရန္ variables အသစ္မ်ား ဖန္တီးေပးမည္ ျဖစ္သည္။ ထိုသို႔ ျပဳလုပ္ရန္ function ၏ declarator ျဖစ္ေသာ void repchar(char ch, int n) ထဲတြင္ သတ္မွတ္ေပးထားေသာ data type မ်ားႏွင္႔ အမည္မ်ား (char ch, int n) ကို အသံုးျပဳထားျခင္းပင္ ျဖစ္သည္။ function အတြင္း pass ျပဳလုပ္လိုက္ေသာ တန္ဖိုးမ်ားကို အဆိုပါ parameters မ်ားအတြင္း initialize ျပဳလုပ္၍ ထည္႔သြင္းေပးလိုက္ျခင္းပင္ ျဖစ္သည္။ ထို႔ေနာက္တြင္ေတာ႔ ၄င္း တန္ဖိုးမ်ားကို ပံုမွန္ variables မ်ား အသံုးျပဳသည္႔အတိုင္း function body အတြင္း အသံုးျပဳႏိုင္ၿပီ ျဖစ္သည္။ ထိုကဲ႔သို႔ passing ျပဳလုပ္လိုက္ေသာ arguments မ်ားကို function အတြင္း ကူးယူဖန္တီးသည္႔ နည္းလမ္းကို passing by value ဟု ေခၚဆိုၾကသည္။ ေနာက္ထပ္ နည္းလမ္းတစ္ခုမွာ passing by reference ဟု ေခၚၿပီး ေနာက္ပိုင္း သင္ခန္းစာမ်ားတြင္ ဆက္လက္ ေလ႔လာသြားၾကမည္ ျဖစ္သည္။ Passing by value ျဖစ္စဥ္ကို ေအာက္ပါပံုတြင္ အေသးစိတ္ ေလ႔လာႏိုင္ပါသည္။



**Structures as Arguments**

function အတြင္းသို႔ တန္ဖိုးမ်ားကို တစ္ခုခ်င္း ထည္႔သြင္းေပးႏိုင္သလို structures မ်ားကို အသံုးျပဳ၍ အစုလိုက္ အျပံဳလိုက္လည္း ထည္႔သြင္းေပးႏိုင္ေပသည္။ နမူနာ ပရိုဂရမ္မ်ားျဖင္႔ ဆက္လက္၍ ေလ႔လာၾကရေအာင္။

**Passing a Distance Structure**

ေအာက္ပါ ဥပမာတြင္ ယခင္က ေလ႔လာခဲ႔ဘူးေသာ Distance structure type ကို input argument အေနျဖင္႔ အသံုးျပဳထားသည္။ engldisp.cpp ကို ေလ႔လာၾကည္႔ၾကရေအာင္ -

// engldisp.cpp

// demonstrates passing structure as argument

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct Distance //English distance

{

int feet;

float inches;

};

////////////////////////////////////////////////////////////////

void engldisp( Distance ); //declaration

int main()

{

Distance d1, d2; //define two lengths

//get length d1 from user

cout << “Enter feet: “; cin >> d1.feet;

cout << “Enter inches: “; cin >> d1.inches;

//get length d2 from user

cout << “\nEnter feet: “; cin >> d2.feet;

cout << “Enter inches: “; cin >> d2.inches;

cout << “\nd1 = “;

engldisp(d1); //display length 1

cout << “\nd2 = “;

engldisp(d2); //display length 2

cout << endl;

return 0;

}

//--------------------------------------------------------------

// engldisp()

// display structure of type Distance in feet and inches

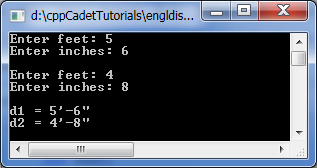
void engldisp( Distance dd ) //parameter dd of type Distance

{

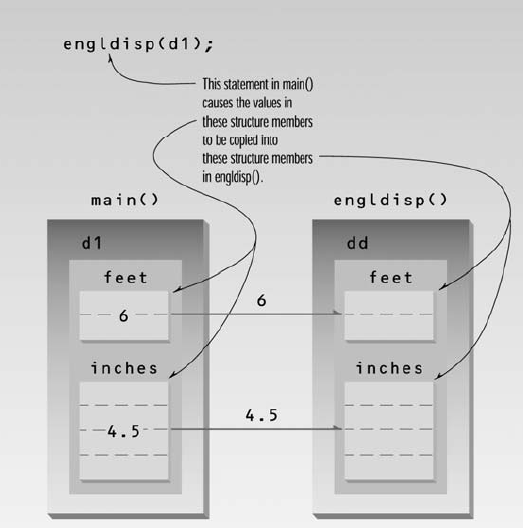
cout << dd.feet << “\’-” << dd.inches << “\””;

}

main() function ထဲတြင္ user ထည္႔သြင္းလိုက္ေသာ distances တန္ဖိုးမ်ားကို feet-and-inches ပံုစံျဖင္႔ လက္ခံၿပီး d1 ႏွင္႔ d2 ဟူေသာ structure ႏွစ္ခုအတြင္း ထည္႔သြင္းလိုက္သည္။ ထို႔ေနာက္ engldisp() function ကို ေခၚယူကာ d1 ႏွင္႔ d2 ကို input argument မ်ားအျဖစ္ ထည္႔သြင္းေပးလိုက္သည္။ ၄င္း function ၏ ရည္ရြယ္ခ်က္မွာ စံျပဳပံုစံ ျဖစ္ေသာ 10'–2.25'' ကဲ႔သို႔ ျပသေပးရန္ ျဖစ္သည္။ ေအာက္တြင္ အဆိုပါ ပရိုဂရမ္၏ output ကို နမူနာ ျပသထားသည္။



main() အတြင္းရွိ function calls မ်ား၊ function declaration မ်ား ႏွင္႔ function body ရွိ declarator မ်ားသည္ input ထည္႔သြင္းေပးလိုက္ေသာ Distance type structure variables မ်ားကို အျခား int, char တို႔ကဲ႔သို႔ေသာ argument variables မ်ားနည္းတူ ျပဳမူေဆာင္ရြက္ေပးသည္။ main() function ထဲတြင္ engldisp() function ကို ႏွစ္ႀကိမ္ေခၚယူထားသည္။ ပထမတစ္ႀကိမ္တြင္ structure d1 အား ထည္႔သြင္းေပးၿပီး ဒုတိယအႀကိမ္တြင္ d2 အားထည္႔သြင္းေပးသည္။ engldisp() function ထဲတြင္ Distance type structure - dd ကို parameter အျဖစ္ အသံုးျပဳထားၿပီး main() function မွ ထည္႔သြင္းေပးလိုက္ေသာ structure value မ်ားကို သာမာန္ argument value မ်ားကဲ႔သို႔ပင္ dd ထဲသို႔ အလိုအေလ်ာက္ ထည္႔သြင္းေပးသည္။ ထို႔ေၾကာင္႔ engldisp() function ထဲမွ statements မ်ားသည္ dd.feet ႏွင္႔ dd.inches ဆိုသည္႔ ပံုမွန္ structure အသံုးျပဳပံု နည္းလမ္းမ်ားအတိုင္း အဆိုပါ value မ်ားကို ရယူ အသံုးျပဳႏိုင္ေပသည္။ ေအာက္ပါပံုတြင္ function တစ္ခုအတြင္းသို႔ structure တစ္ခုအား argument အေနျဖင္႔ ထည္႔သြင္းအသံုးျပဳပံုကို ရွင္းလင္းထားသည္။



ပံုမွန္ variables မ်ားကဲ႔သို႔ပင္ engldisp() function အတြင္းရွိ structure parameter - dd ကို ေျပာင္းလဲျခင္းသည္ ၄င္းအား main() function မွ ထည္႔သြင္းေပးလိုက္ေသာ structures -d1 ႏွင္႔ d2 တို႔အား ေျပာင္းလဲေစျခင္း မရွိေပ။ သီးသန္႔ ျဖစ္ေနေပသည္။ ဥပမာ -

dd.feet = 2;

dd.inches = 3.25;

တို႔သည္ main() အတြင္းရွိ d1 ႏွင္႔ d2 တို႔၏ တန္ဖိုးမ်ားအား မေျပာင္းလဲ ေစႏိုင္ေပ။

**Passing a circle Structure**

အခုတစ္ပတ္မွာေတာ႔ Console Graphics Lite functions မ်ားကို အသံုးျပဳၿပီး ေရးသားထားတဲ႔ ဥပမာေလးကို ေလ႔လာသြားမွာ ျဖစ္ပါတယ္။ Console Graphics Lite functions ရဲ႕ ကုဒ္မ်ားကို ဂရပ္ဖစ္ပိုင္းဆိုင္ရာ နမူနာ ပရိုဂရမ္မ်ားမွာ ထည္႔သြင္းအသံုးျပဳႏိုင္ေအာင္ ဆိုတဲ႔ ရည္ရြယ္ခ်က္နဲ႔ Robert Lafore ေရးသားတဲ႔ Object-Oriented Programming in C++(4th edition) ရဲ႕ Appendix E မွာ ထည္႔သြင္းေပးထားတာပဲ ျဖစ္ပါတယ္။ ေနာက္ပိုင္းမွာေတာ႔ အဲဒီ function မ်ားကို အေသးစိတ္ ရွင္းလင္းေပးမွာ ျဖစ္ေပမယ္႔ အခုအခန္းမွာေတာ႔ function ေတြနဲ႔ ပက္သက္ၿပီး အဓိကထားေလ႔လာေနတာ ျဖစ္တဲ႔အတြက္ ဂရပ္ဖစ္အပိုင္းေရးသားထားခ်က္မ်ားကို အနည္းငယ္ပဲ ေဆြးေႏြးသြားမွာပါ။ အဲဒီ Console Graphics Lite functions ေတြကို အသံုးျပဳဖို႔ အတြက္ေတာ႔ အသံုးျပဳတဲ႔ compiler အေပၚမူတည္ၿပီး msoftcon.h ဒါမွမဟုတ္ borlancon.h တစ္ခုခုကို include ျပဳလုပ္ေပးဖို႔ လိုအပ္ပါတယ္။ ေအာက္ပါ နမူနာ ပရိုဂရမ္ေလးမွာေတာ႔ function တစ္ခုထဲကို structure တစ္ခု ထည္႔သြင္း အသံုးျပဳပံုကို Console Graphics Lite functions ေတြကို အသံုးျပဳထားၿပီး ေရးသားျပထားပါတယ္။ အဲဒီ နမူနာထဲမွာ စက္ဝိုင္းတစ္ခု ကို ကိုယ္စားျပဳတဲ႔ circle structure တစ္ခုကို အသံုးျပဳထားပါတယ္။ စက္ဝိုင္းတစ္ခုကို ေဖာ္ျပႏိုင္ဖို႔ အလယ္မွတ္နဲ႔ radius ေတြကို ထည္႔သြင္းေပးရမွာပါ။ ဒါ႔အျပင္ စက္ဝိုင္းအတြက္ အေရာင္နဲ႔ fill pattern ေတြကိုလည္း ထည္႔သြင္းထားႏိုင္ပါတယ္။ ေအာက္ပါ circstrc.cpp ကို ေလ႔လာၾကည္႔ႏိုင္ပါတယ္။

// circstrc.cpp

// circles as graphics objects

#include “msoftcon.h” // for graphics functions

////////////////////////////////////////////////////////////////

struct circle //graphics circle

{

int xCo, yCo; //coordinates of center

int radius;

color fillcolor; //color

fstyle fillstyle; //fill pattern

};

////////////////////////////////////////////////////////////////

void circ\_draw(circle c)

{

set\_color(c.fillcolor); //set color

set\_fill\_style(c.fillstyle); //set fill pattern

draw\_circle(c.xCo, c.yCo, c.radius); //draw solid circle

}

//--------------------------------------------------------------

int main()

{

init\_graphics(); //initialize graphics system

//create circles

circle c1 = { 15, 7, 5, cBLUE, X\_FILL };

circle c2 = { 41, 12, 7, cRED, O\_FILL };

circle c3 = { 65, 18, 4, cGREEN, MEDIUM\_FILL };

circ\_draw(c1); //draw circles

circ\_draw(c2);

circ\_draw(c3);

set\_cursor\_pos(1, 25); //cursor to lower left corner

return 0;

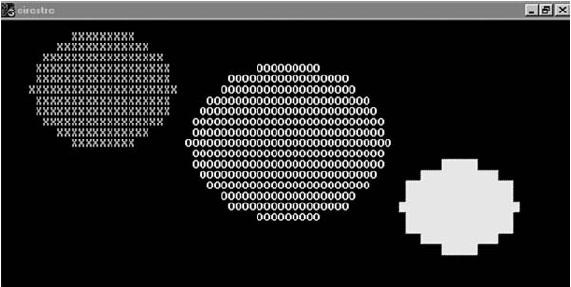
}

အထက္ပါ ဥပမာမွာ circle structure type အမ်ိဳးအစား variables c1, c2 နဲ႔ c3 ေတြကို မတူညီတဲ႔ တန္ဖိုးေတြ ထည္႔သြင္းၿပီး initialized လုပ္ထားပါတယ္။ ဥပမာအေနနဲ႔ c1 ကို ေအာက္ပါအတိုင္း ဖန္တီးလိုက္ပါတယ္-

circle c1 = { 15, 7, 5, cBLUE, X\_FILL };

ကၽြန္္ေတာ္တို႔အေနနဲ႔ console screen ကို 80 columns နဲ႔ 25 rows ရွိတယ္လို႔ သတ္မွတ္လိုက္ပါတယ္။ အထက္က 15 ဟာ column number ဒါမွမဟုတ္ x coordinate ျဖစ္ၿပီး 7 ကေတာ႔ row number ဒါမွမဟုတ္ screen ရဲ႕ ထိပ္ဆံုးက ေနအစျပဳတဲ႔ y coordinate ျဖစ္ပါတယ္။ အဲဒီ တန္ဖိုးႏွစ္ခုက စက္ဝိုင္းရဲ႕ အလယ္မွတ္ကို ေဖာ္ျပေပးတာျဖစ္ပါတယ္။ ေနာက္က 5 ကေတာ႔ စက္ဝိုင္းရဲ႕ အခ်င္းဝက္ တန္ဖိုး ျဖစ္ပါတယ္။ cBLUE ကေတာ႔ စက္ဝိုင္းရဲ႕ အေရာင္ကို ကိုယ္စားျပဳထားၿပီး X\_FILL ကေတာ႔ စက္ဝိုင္းကို ျဖည္႔ေပးမယ္႔ fill pattern ပဲ ျဖစ္ပါတယ္။ စက္ဝိုင္းႏွစ္ခုကို တန္ဖိုး အတူတူေပးၿပီး ဖန္တီးလိုက္ပါတယ္။

စက္ဝိုင္းေတြ အားလံုးကို တန္ဖိုးေတြ သတ္မွတ္ေပးၿပီးတာနဲ႔ circ\_draw() function ကို သံုးႀကိမ္ေခၚယူၿပီး စက္ဝိုင္းေတြကို Console မွာ ဆြဲေပးမွာ ျဖစ္ပါတယ္။ ေအာက္က ပံုေလးမွာေတာ႔ အဲဒီ result ေတြကို ျပသထားပါတယ္။ ပံုေတြက ခပ္ၾကမ္းၾကမ္းျဖစ္ေနေပမယ္႔ အေျခခံ ဂရပ္ဖစ္ နမူနာေလး အျဖစ္ေတာ႔ အသံုးက်ပါတယ္။



ဒီနမူနာေလးမွာ စက္ဝိုင္းတစ္ခုရဲ႕ သြင္ျပင္လကၡဏာေတြကို structure ထဲမွာ စုစည္းထားတာကို သတိထားမိမွာပါ။ ၄င္းကို circ\_draw() function ထဲကို input အေနနဲ႔ ထည္႔သြင္းေပးလိုက္ၿပီး ေရးဆြဲလိုက္တာပဲ ျဖစ္ပါတယ္။ circ\_draw() ကို ဘယ္လို ေရးသားထားသလဲ ဆိုတာကိုေတာ႔ function အေၾကာင္း ၿပီးတဲ႔ အခ်ိန္မွာ Console Graphics Lite functions မ်ားအေၾကာင္း အေသးစိတ္ ရွင္းျပတဲ႔ အထဲမွာ ထည္႔သြင္းရွင္းျပေပးသြားမွာပါ။

**Names in the Declaration**

Function declarations ေတြကို ေၾကျငာတဲ႔ ေနရာမွာ ပိုမိုရွင္းလင္းေအာင္ ေရးသားႏိုင္ပါေသးတယ္။ Input arguments အေနနဲ႔ ထည္႔ေပးရမယ္႔ data type ေတြနဲ႔ အတူ အဓိပၸါယ္ရွိတဲ႔ variable name ေတြ ထည္႔ေပးလိုက္ဖို႔ပဲ ျဖစ္ပါတယ္။ စာေရးသူအေနနဲ႔ကေတာ႔ function declarations ေတြျပဳလုပ္ဖို႔ ဒီနည္းကို အျမဲအသံုးျပဳပါတယ္။ ဥပမာအေနနဲ႔ ကြန္ပ်ဴတာ ဖန္သားျပင္ေပၚမွာ အမွတ္တစ္ခုကို ျပသေပးတဲ႔ function ေလးတစ္ခုကို အသံုးျပဳေနတယ္ ဆိုၾကပါစို႔။ ယခင္ ဥပမာမ်ားအရ ေအာက္ပါအတိုင္း data typed ေတြကို ထည္႔သြင္းၿပီး ေရးသားႏိုင္ပါတယ္။

**void display\_point(int, int); //declaration**

အထက္ပါ function declaration ေလးကို ေအာက္ပါအတိုင္း ပိုမိုရွင္းလင္းေအာင္ ျပင္ေရးႏိုင္ပါတယ္။

**void display\_point(int horiz, int vert); //declaration**

Compiler အတြက္ေတာ႔ အထက္ပါ declarations ႏွစ္ခုစလံုးကဟာ အတူတူပါပဲ။ ဒါေပမယ္႔ ပထမ declaration မွာ ပါတဲ႔ (int, int) ဟာ ထည္႔ေပးရမယ္႔ arguments ေတြနဲ႔ ပက္သက္ၿပီး ဘာမွ မေျပာထားတဲ႔ အတြက္ ဘယ္တစ္ခုက vertical coordinate အတြက္ ျဖစ္ၿပီး၊ ဘယ္တစ္ခုက horizontal coordinate အတြက္ ျဖစ္တယ္ဆိုတာကို မသိႏိုင္ပါဘူး။

ဒုတိယနည္းလမ္းကို အသံုးျပဳၿပီး ေၾကျငာလိုက္တဲ႔ အခါမွာ ပရိုဂရမ္မာေတြ အတြက္ ပိုမိုရွင္းလင္း လြယ္ကူေစပါတယ္။ ဒီလို declaration မ်ိဳးကို ဖတ္မိတဲ႔ မည္သူမဆို အဲဒီ function ကို ေခၚယူတဲ႔ ေနရာမွာ arguments ေတြကို မွန္မွန္ကန္ကန္ ထည္႔သြင္းမိဖို႔မ်ားပါတယ္။ မွတ္သားရမယ္႔ အခ်က္ တစ္ခုကေတာ႔ အဲဒီ declarations ေတြထဲမွာ အသံုးျပဳထားတဲ႔ variable names ေတြဟာ function တစ္ခုကို ေခၚယူတဲ႔ အခါမွာ အသံုးျပဳမယ္႔ variable names ေတြအေပၚမ်ာ အက်ိဳးသက္ေရာက္မႈ မရွိတာပဲ ျဖစ္ပါတယ္။ ဒါေၾကာင္႔ အဲဒီလို ေၾကျငာဖို႔ ႀကိဳက္ႏွစ္သက္ရာ variable names ေတြကို လြတ္လြတ္လပ္လပ္ အသံုးျပဳႏိုင္ပါတယ္။ ဥပမာ-

**display\_point(x, y); // function call**

ဒါေၾကာင္႔ ကၽြန္ေတာ္ေရးသားမယ္႔ ေနာက္ပိုင္း သင္ခန္းစာမ်ားမွာ စာဖတ္သူမ်ား လြယ္ကူစြာ နားလည္ႏိုင္ေစဖို႔ ရည္ရြယ္ၿပီး ဒုတိယနည္းလမ္းျဖစ္တဲ႔ variable name နဲ႔ data type တြဲသံုးၿပီး ေရးသားျပသြားမွာ ျဖစ္ပါတယ္။

**Returning Values from Functions**

Function တစ္ခုကို ေဆာင္ရြက္ၿပီးသြားတဲ႔ အခါမွာ တန္ဖိုးတစ္ခုကို ေခၚယူတဲ႔ ပရိုဂရမ္ကို ျပန္ပို႔ေပးပါတယ္။ မ်ားေသာအားျဖင္႔ အဲဒီ တန္ဖိုးဟာ function က သက္ဆိုင္ရာ ျပႆနာ တစ္ခုကို ေျဖရွင္းေပးလိုက္တဲ႔ အေျဖေတြ ပါဝင္ေလ႔ ရွိပါတယ္။ ေအာက္ပါ ဥပမာမွာ အေလးခ်ိန္တစ္ခုကို ေပါင္ကေန ကီလိုဂရမ္ ေျပာင္းေပး မွာ ျဖစ္ပါတယ္။ convert.cpp ေလးကို ေလ႔လာၾကည္႔ၾကရေအာင္-

// convert.cpp

// demonstrates return values, converts pounds to kg

#include <iostream>

using namespace std;

float lbstokg(float); //declaration

int main()

{

float lbs, kgs;

cout << “\nEnter your weight in pounds: “;

cin >> lbs;

kgs = lbstokg(lbs);

cout << “Your weight in kilograms is “ << kgs << endl;

return 0;

}

//--------------------------------------------------------------

// lbstokg()

// converts pounds to kilograms

float lbstokg(float pounds)

{

float kilograms = 0.453592 \* pounds;

return kilograms;

}

ေအာက္မွာ အဲဒီ ပရိုဂရမ္ကို နမူနာ စမ္းသပ္ျပထားပါတယ္။ output ေတြကို ေလ႔လာၾကည္႔ပါ-

Enter your weight in pounds: 182

Your weight in kilograms is 82.553741

Function တစ္ခုမွ တန္ဖိုးတစ္ခုကို return ျပန္ေပးႏိုင္ရန္အတြက္ အဆိုပါ တန္ဖိုး၏ data type ကို သတ္မွတ္ေပးထားရမည္ ျဖစ္သည္။ ထိုသို႔သတ္မွတ္ျခင္းကို function declaration အတြင္း ထည္႔သြင္းေရးသားရမည္ ျဖစ္သည္။ အထက္ပါ ဥပမာျပထားတဲ႔ အတိုင္း function declaration နဲ႔ definition နွစ္ခုစလံုးမွာ (float) data type ကို function name ရဲ႕ အေရွ႕မွာ ေရးသားသတ္မွတ္ရမွာ ျဖစ္ပါတယ္။

ယခင္က ဥပမာေပး ေရးသားျပထားတဲ႔ နမူနာ ပရိုဂရမ္မ်ားမွာ တန္ဖိုးျပန္ေပးတာ မရွိခဲ႔ပါဘူး။ ဒါေၾကာင္႔ return type ကို void လို႔ ေရးသားေပးရပါတယ္။ convert.cpp မွာ ေရးသားထားတဲ႔ lbstokg() မွာေတာ႔ float type ကို return ျပန္ေပးထားပါတယ္။ ဒါေၾကာင္႔ သူရဲ႕ declaration က ေအာက္ပါအတိုင္း ျဖစ္ပါတယ္-

**float lbstokg(float);**

ေရွ႕ဆံုးမွာ ရွိတဲ႔ float က return type ကို သတ္မွတ္ေပးတာ ျဖစ္ပါတယ္။ ေနာက္က လဲသဲကြင္းထဲမွာ ရွိတဲ႔ float ကေတာ႔ အဲဒီ function (lbstokg()) ကို ထည္႔သြင္းေပးရမယ္႔ data type ကို သတ္မွတ္ေပးထားတာ ျဖစ္ပါတယ္။

Function တစ္ခုက တန္ဖိုးတစ္ခုကို return ျပန္ေပးတဲ႔ အခါမွာ lbstokg(lbs) ဆိုတဲ႔ function call ကို အဲဒီ function က ျပန္ေပးတဲ႔ တန္ဖိုးအေနနဲ႔ ယူဆၿပီး အသံုးျပဳမွာ ျဖစ္ပါတယ္။ ဥပမာအားျဖင္႔ အဲဒီက ထြက္လာတဲ႔ တန္ဖိုးကို အျခား variable ေတြကဲ႔သို႔ ယူဆၿပီး assignment statement မွာ

**kgs = lbstokg(lbs);**

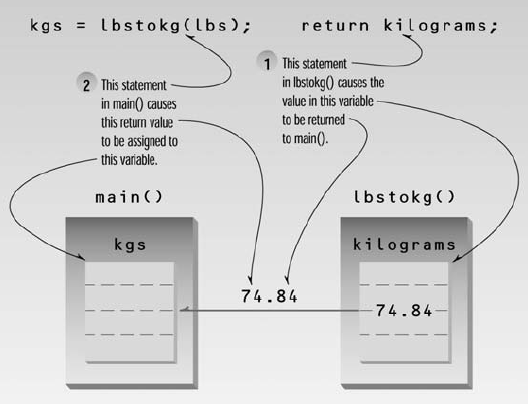
ဒီကုဒ္က lbstokg() ကေန return ျပန္ေပးလိုက္တဲ႔ တန္ဖိုးကို kgs ဆိုတဲ႔ variable ထဲ assign လုပ္ၿပီး ထည္႔ေပးသြားမွာ ျဖစ္ပါတယ္။

**The return Statement**

Function lbstokg() ထဲကို အေလးခ်ိန္ ေပါင္တန္ဖိုးေတြကို ကိုယ္စားျပဳတဲ႔ argument တစ္ခုထည္႔သြင္းေပးတဲ႔ အခါမွာ pounds ဆိုတဲ႔ parameter ထဲမွာ သိမ္းဆည္းထားလိုက္မွာ ျဖစ္ပါတယ္။ အဲဒီ ေပါင္တန္ဖိုးေတြကို ကီလိုဂရမ္ေျပာင္းလဲဖို႔ ကိန္းေသတစ္ခုနဲ႔ ေျမွာက္ၿပီး kilograms ဆိုတဲ႔ variable ထဲမွာ သိမ္းထားလိုက္ပါတယ္။ အဲဒီ variable ထဲက တန္ဖိုးကို ေအာက္ပါအတိုင္း return statement အသံုးျပဳၿပီး ေခၚယူတဲ႔ ပရိုဂရမ္ကို ျပန္ေပးမွာ ျဖစ္ပါတယ္။

**return kilograms;**

သတိထားရမယ္႔ အခ်က္တစ္ခုကေတာ႔ ကီလိုဂရမ္ တန္ဖိုးကို သိမ္းဆည္းဖို႔ main() function ထဲမွာ kgs ဆိုတဲ႔ variable ကို အသံုးျပဳထားၿပီး lbstokg() ထဲမွာ kilograms ဆိုတဲ႔ variable ကို အသံုးျပဳထားပါတယ္။ Calling program ျဖစ္တဲ႔ main() function က lbstokg() function ထဲက kilograms ဆိုတဲ႔ variable ကို ရယူသံုးစြဲျခင္း မျပဳဘဲ (သံုးလို႔လဲ မရပါဘူး) အဲဒီ lbstokg() function ကေန ျပန္ေပးတဲ႔ တန္ဖိုးကိုပဲ ရယူသံုးစြဲထားတာ ျဖစ္ပါတယ္။ အဲဒီ ျဖစ္စဥ္ကို ေအာက္ပါ ပံုမွာ ရွင္းလင္း တင္ျပထားပါတယ္။



ကၽြန္ေတာ္တို႔ အေနနဲ႔ function တစ္ခုကို arguments ေတြအမ်ားႀကီး ထည္႔သြင္းေပးလို႔ ရေပမယ္႔ return ျပန္ေပးတဲ႔ အခါမွာေတာ႔ argument တစ္ခုပဲ ျပန္ေပးႏိုင္ပါတယ္။ တကယ္လို႔ function တစ္ခုကေန အခ်က္အလက္အမ်ားအျပား ေပးခ်င္လာတဲ႔ အခါမွာ အဲဒီ ကန္႔သတ္ခ်က္ ေၾကာင္႔ အခက္အခဲ ျဖစ္လာႏိုင္ပါတယ္။ ဒါေၾကာင္႔ အဲဒီအခက္အခဲကို ေက်ာ္လႊားဖို႔အတြက္ arguments ေတြကို pass by reference နည္းလမ္းနဲ႔ေသာ္ လည္းေကာင္း၊ အခ်က္အလက္ အမ်ားအျပားကို စုစည္းၿပီး structure တစ္ခုတည္ေဆာက္ကာ return ျပန္ေပးျခင္းျဖင္႔ ေသာ္လည္းေကာင္း ေျဖရွင္းႏိုင္ပါတယ္။

ကၽြန္ေတာ္တို႔ အေနျဖင္႔ function တစ္ခု၏ return type ကို function declaration မွာ မျဖစ္မေန ထည္႔သြင္းေပးရမွာ ျဖစ္ပါတယ္။ အကယ္၍ function အေနျဖင္႔ return ျပန္ေပးရန္ မလိုအပ္ပါက void ကို ထည္႔သြင္းေပးျခင္းျဖင္႔ ညႊန္ျပႏိုင္ပါသည္။ function declaration တြင္ return type ထည္႔သြင္းမေပးပါက compiler အေနျဖင္႔ int value ျပန္ေပးလိမ္႔မည္ဟု ယူဆမည္ ျဖစ္သည္။ ဥပမာအေနျဖင္႔ ေအာက္ပါ declaration ကို ေလ႔လာၾကည္႔ပါ-

somefunc(); // declaration -- assumes return type is int tells the compiler that somefunc() တြင္ return type ကို သတ္မွတ္ေပးထားျခင္းမရွိ၍ int ဟု ယူဆမည္ ျဖစ္သည္။ ဤသို႔ ယူဆရျခင္းမွာ အေစာပိုင္း C versions မ်ား အသံုးျပဳပံုကို အေျခခံထားျခင္းေၾကာင္႔ ပင္ျဖစ္သည္။ သို႔ရာတြင္ ၄င္းနည္းလမ္းကို လက္ေတြ႔တြင္ အသံုးမျပဳသင္႔ေပ။ retun type သည္ int ျဖစ္ေနရင္ေတာင္မွ declaration တြင္ မျဖစ္မေန သတ္မွတ္ေပးသင္႔ေပသည္။ သို႔မွသာ ကုဒ္မ်ား တစ္ပံုစံတည္း ျဖစ္ၿပီး ဖတ္ရလြယ္မည္ ျဖစ္သည္။

**Eliminating Unnecessary Variables**

convert.cpp ပရိုဂရမ္တြင္ တကယ္တမ္း မလိုအပ္ဘဲ ဖတ္ရႈရ ရွင္းလင္းေစရန္ ထည္႔သြင္း အသံုးျပဳထားေသာ variables အမ်ားအျပား ပါဝင္သည္။ ေအာက္ပါ convert2.cpp ပရိုဂရမ္တြင္ မလိုအပ္ေသာ variables မ်ားကို ျဖဳတ္၍ ေရးသားျပထားသည္။

// convert2.cpp

// eliminates unnecessary variables

#include <iostream>

using namespace std;

float lbstokg(float); //declaration

int main()

{

float lbs;

cout << “\nEnter your weight in pounds: “;

cin >> lbs;

cout << “Your weight in kilograms is “ << lbstokg(lbs)

<< endl;

return 0;

}

//--------------------------------------------------------------

// lbstokg()

// converts pounds to kilograms

float lbstokg(float pounds)

{

return 0.453592 \* pounds;

}

main() function တြင္ convert.cpp တြင္ အသံုးျပဳထားေသာ variable kgs ကို ျဖဳတ္ပစ္ၿပီး function call - lbstokg(lbs) ကို cout statement အတြင္းတြင္ ေအာက္ပါအတိုင္း တိုက္ရိုက္ အသံုးျပဳထားသည္။

cout << “Your weight in kilograms is “ << lbstokg(lbs) << endl;

ထိုနည္းတူ lbstokg() function တြင္လည္း variable kilograms ကို အသံုးမျပဳေတာ႔ဘဲ 0.453592\*pounds ဟူသည္႔ expression ကို return statement တြင္ တိုက္ရိုက္ထည္႔သြင္းထားသည္။

return 0.453592 \* pounds;

ယခင္ နည္းလမ္းအရ တြက္ခ်က္မႈမွ ရရွိေသာ တန္ဖိုးကို variable အတြင္း ထည္႔သြင္းကာ calling program ကို return ျပန္ေပးမည္ျဖစ္သည္။ ယခု နည္း၌မူ တြက္ခ်က္ရေသာ တန္ဖိုးကို calling program သို႔ တိုက္ရိုက္ return ျပန္ေပးမည္ျဖစ္သည္။ ပိုမိုရွင္းလင္းေစရန္အတြက္ ပရိုဂရမ္မာ အမ်ားစုက expression ကို လက္သဲကြင္းျဖင္႔ ပိတ္၍ ေအာက္ပါအတိုင္း ေရးသားေလ႔ ရွိသည္။

return (0.453592 \* pounds);

Compiler အေနျဖင္႔ အဆိုပါ လက္သဲကြင္းမ်ားကို မလိုအပ္ေသာ္လည္း ကၽြန္ေတာ္တို႔ ပရိုဂရမ္မာမ်ား ဖတ္ရႈရ လြယ္ကူေစရန္ ထည္႔သြင္းေပးသင္႔သည္။ ကၽြမ္းက်င္ေသာ C++ (သို႔မဟုတ္ C) ပရိုဂရမ္မာမ်ား အေနျဖင္႔ ပိုမိုရွည္လ်ားေသာ convert.cpp ပံုစံမ်ိဳးထက္ တိုေတာင္းေသာ convert2.cpp ပံုစံမ်ိဳးကို ႀကိဳက္ႏွစ္သက္ေလ႔ ရွိၾကသည္။ သို႔ရာတြင္ convert2.cpp ပံုစံသည္ နားလည္ရန္ ခက္ခဲႏိုင္ေသာေၾကာင္႔ မကၽြမ္းက်င္ေသးသူမ်ားအဖို႔ အသံုးမျပဳသင္႔ေပ။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ တိုေတာင္းမႈ ႏွင္႔ ရွင္းလင္းမႈကို ေရြးခ်ယ္ရသည္႔အခါတြင္ ကၽြန္ေတာ္တို႔၏ ေရးသားမႈစတိုင္ႏွင္႔ အဆိုပါ ကုဒ္မ်ားကို မည္သူက ဖတ္ရႈမည္ဟူေသာ အခ်က္ကို အေျခခံ၍ ေရြးခ်ယ္ၾကရမည္ ျဖစ္ပါသည္။

**Returning Structure Variables**

ေရွ႕ပိုင္း သင္ခန္းစာမ်ားတြင္ functions မ်ားအတြင္း arguments မ်ားအေနျဖင္႔ structures မ်ားကို ထည္႔သြင္းေပးႏိုင္ေၾကာင္း ေလ႔လာခဲ႔ၾကပါသည္။ ထိုနည္းတူစြာ structure တစ္ခုကို return value အျဖစ္လည္း အသံုးျပဳႏိုင္ပါသည္။ retstrc.cpp တြင္ Distance variables မ်ားကို ေပါင္းကာ ရရွိလာေသာ Distance data type တန္ဖိုးကို return ျပန္ေပးမည္ ျဖစ္ပါသည္။

// retstrc.cpp

// demonstrates returning a structure

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct Distance //English distance

{

int feet;

float inches;

};

////////////////////////////////////////////////////////////////

Distance addengl(Distance, Distance); //declarations

void engldisp(Distance);

int main()

{

Distance d1, d2, d3; //define three lengths

//get length d1 from user

cout << “\nEnter feet: “; cin >> d1.feet;

cout << “Enter inches: “; cin >> d1.inches;

//get length d2 from user

cout << “\nEnter feet: “; cin >> d2.feet;

cout << “Enter inches: “; cin >> d2.inches;

d3 = addengl(d1, d2); //d3 is sum of d1 and d2

cout << endl;

engldisp(d1); cout << “ + “; //display all lengths

engldisp(d2); cout << “ = “;

engldisp(d3); cout << endl;

return 0;

}

//--------------------------------------------------------------

// addengl()

// adds two structures of type Distance, returns sum

Distance addengl( Distance dd1, Distance dd2 )

{

Distance dd3; //define a new structure for sum

dd3.inches = dd1.inches + dd2.inches; //add the inches

dd3.feet = 0; //(for possible carry)

if(dd3.inches >= 12.0) //if inches >= 12.0,

{ //then decrease inches

dd3.inches -= 12.0; //by 12.0 and

dd3.feet++; //increase feet

} //by 1

dd3.feet += dd1.feet + dd2.feet; //add the feet

return dd3; //return structure

}

//--------------------------------------------------------------

// engldisp()

// display structure of type Distance in feet and inches

void engldisp( Distance dd )

{

cout << dd.feet << “\’-” << dd.inches << “\””;

}

အသံုးျပဳသူမ်ားအေနျဖင္႔ ေပ ႏွင္႔ လက္မ မ်ားကို အလ်ား ႏွစ္စံုအတြက္ ထည္႔သြင္းေပးရမည္ ျဖစ္ၿပီး addengl() function ကို အသံုးျပဳ၍ တန္ဖိုးႏွစ္ခုကို ေပါင္းေပးမည္ ျဖစ္သည္။ ထို႔ေနာက္ ရလာေသာ တန္ဖိုးမ်ားကို engldisp() function ကို အသံုးျပဳ၍ ျပသေပးမည္ ျဖစ္သည္။ ေအာက္မွာ အဲဒီ ပရိုဂရမ္ကို နမူနာ စမ္းသပ္ျပထားပါတယ္။ output ေတြကို ေလ႔လာၾကည္႔ပါ-

Enter feet: 4

Enter inches: 5.5

Enter feet: 5

Enter inches: 6.5

4’-5.5” + 5’-6.5” = 10’-0”

Distance type အလ်ားႏွစ္ခုကို ေပါင္းဖို႔ addengl() function ကို ေခၚယူၿပီး ရလာတဲ႔ တန္ဖိုး ကို ေနာက္ထပ္ Distance type ျဖစ္တဲ႔ d3 ထဲကို ေအာက္ပါအတိုင္း ထည္႔သြင္းေပးလိုက္ပါတယ္။

d3 = addengl(d1, d2);

ဒီေန႔ သင္ခန္းစာေလးမွာ structure တစ္ခုကို function ရဲ႕ return value အျဖစ္ ျပန္ေပးႏိုင္တဲ႔အေၾကာင္း ေလ႔လာရသလို အထက္ပါ ပရိုဂရမ္ေလးမွာ function ႏွစ္ခုကို သံုးျပထားပါတယ္။ အဲဒီ function ေတြရဲ႕ definitions ေတြကို ေရွ႕ေနာက္ စိတ္ႀကိဳက္ စီစဥ္ထားႏိုင္ပါတယ္။ တစ္ခုတည္းေသာ စည္းမ်ဥ္းကေတာ႔ function call ေတြ မတိုင္ခင္မွာ အဲဒီ function ေတြကို declare လုပ္ထားရမယ္ဆိုတဲ႔ အခ်က္ပဲ ျဖစ္ပါတယ္ခင္ဗ်ာ။

**Reference Arguments**

Reference တစ္ခုဟာ variable တစ္ခုကို ကိုင္တြယ္ဖို႔ ေနာက္ထပ္ နည္းလမ္းတစ္ခုကို ပံ႔ပိုးေပးပါတယ္။ အသံုးအဝင္ဆံုး နည္းလမ္းကေတာ႔ function ေတြကို argument ေတြ pass လုပ္ေပးတဲ႔ ေနရာမွာပဲ ျဖစ္ပါတယ္။

ယခင္ သင္ခန္းစာမ်ားမွာ functin တစ္ခုကို arguments ေတြကို passed by value နည္းနဲ႔ ပို႔ေပးခဲ႔တာကို ေလ႔လာခဲ႔ၿပီးပါၿပီ။ အဲဒီလိုျပဳလုပ္တဲ႔ ေနရာမွာ ေခၚယူခံရတဲ႔ function က အဲဒီ argument နဲ႔ data type တူညီတဲ႔ variable အသစ္တစ္ခုကို ဖန္တီးၿပီး အဲဒီ argument ထဲက တန္ဖိုးကို copy ကူးၿပီး ထည္႔ေပးလိုက္ပါတယ္။

Function ေတြဟာ သူ႔ကို ေခၚယူတဲ႔ ပရိုဂရမ္ထဲက မူရင္း တန္ဖိုးေတြကို ရယူႏိုင္ျခငး္ မရွိပါဘူး။ တန္ဖိုးေတြကို ကူးယူၿပီး အသစ္ဖန္တီးရတာ ျဖစ္ပါတယ္။ အဲဒီ နည္းလမ္းဟာ တကယ္လို႔ function ေတြက original variable ေတြရဲ႕ တန္ဖိုးေတြကို ျပင္ဆင္ဖို႔ မလိုအပ္တဲ႔ အခ်ိန္မ်ိဳးမွာ အသံုးဝင္ပါတယ္။ တစ္နည္းအားျဖင္႔ ၄င္းနည္းလမ္းဟာ မူရင္းတန္ဖိုးေတြကို ျပင္ဆင္ျခင္း မခံရေအာင္ အာမခံခ်က္ ေပးႏိုင္ပါတယ္။

ဒါေပမယ္႔ passing arguments by reference ျပဳလုပ္တဲ႔ အခါမွာေတာ႔ ကြဲျပားျခားနားစြာ ျပဳမူေဆာင္ရြက္ပါတယ္။ ဒီနည္းလမ္းမွာေတာ႔ function ထဲကို တန္ဖိုးေတြ ထည္႔သြင္းေပးလိုက္မယ္႔ အစား မူရင္းတန္ဖိုးကို ရည္ညႊန္းေနတဲ႔ reference တစ္ခု (တကယ္ေတာ႔ memory address တစ္ခု) ကို pass ျပဳလုပ္ေပးမွာ ျဖစ္ပါတယ္။

Passing by reference လုပ္တဲ႔ နည္းလမ္းရဲ႕ အေရးပါတဲ႔ အားသာခ်က္တစ္ခုကေတာ႔ function အေနနဲ႔ ေခၚယူလိုက္တဲ႔ ပရိုဂရမ္ထဲက တကယ္႔ variable ေတြကို ရယူျပင္ဆင္ႏိုင္တာပဲ ျဖစ္ပါတယ္။ အဲဒီ အားသာခ်က္ေတြကို တကယ္တမ္း အသံုးျပဳတဲ႔ ေနရာကေတာ႔ function တစ္ခုကေန အခ်က္အလက္ အမ်ားအျပားကို ျပန္ပို႔ေပးႏိုင္ဖို႔ အသံုးျပဳတာပဲ ျဖစ္ပါတယ္။

**Passing Simple Data Types by Reference**

ref.cpp မွာ passed by reference နည္းလမ္းကို အသံုးျပဳၿပီး variable တစ္ခုကို pass လုပ္ျပထားပါတယ္။

// ref.cpp

// demonstrates passing by reference

#include <iostream>

using namespace std;

int main()

{

void intfrac(float, float&, float&); //declaration

float number, intpart, fracpart; //float variables

do {

cout << “\nEnter a real number: “; //number from user

cin >> number;

intfrac(number, intpart, fracpart); //find int and frac

cout << “Integer part is “ << intpart //print them

<< “, fraction part is “ << fracpart << endl;

} while( number != 0.0 ); //exit loop on 0.0

return 0;

}

//--------------------------------------------------------------

// intfrac()

// finds integer and fractional parts of real number

void intfrac(float n, float& intp, float& fracp)

{

long temp = static\_cast<long>(n); //convert to long,

intp = static\_cast<float>(temp); //back to float

fracp = n - intp; //subtract integer part

}

ပထမဦးစြာ main() ပရိုဂရမ္မွ user ကို float တန္ဖိုး တစ္ခု ထည္႔သြင္းေစပါတယ္။ ပရိုဂရမ္က ထို တန္ဖိုးကို integer နဲ႔ fractional part ဆိုၿပီး ကိန္းျပည္႔နဲ႔ ဒႆမ ဂဏန္းေတြ အျဖစ္ ခြဲထုတ္လိုက္ပါတယ္။ ျမင္သာေအာင္ ဥပမာေပးရမယ္ဆိုရင္ ၁၂.၄၅၆ ကို ၁၂.၀ နဲ႔ ၀.၄၅၆ ဆိုၿပီး ခြဲထုတ္လိုက္တာ ျဖစ္ပါတယ္။ အဲဒီလို ခြဲထုတ္ႏိုင္ဖို႔အတြက္ main() က intfrac() function ကို ေခၚယူရပါတယ္။ ေအာက္မွာ ေတာ႔ အဲဒီ ပရိုဂရမ္ကို စမ္းသပ္လို႔ ထြက္လာတဲ႔ အေျဖေတြကို ျပသထားပါတယ္။

Enter a real number: 99.44

Integer part is 99, fractional part is 0.44

အခ်ိဳ႕ compiler ေတြဟာ ဒႆမကိန္းကို ဖန္တီးတဲ႔ အခါမွာ အပို ဒီဂ်စ္ေတြ ထုတ္ေပးတတ္ပါတယ္။ ဥပမာ ၀.၄၄၀၀၀၂ ဆိုတဲ႔ ဂဏန္းမ်ိဳးျဖစ္ပါတယ္။ တကယ္ေတာ႔ ဒါဟာ compiler မွာပါတဲ႔ conversion routine ရဲ႕ အမွားျဖစ္ပါတယ္။ intfrac() function ဟာ ထည္႔သြင္းေပးလိုက္တဲ႔ တန္ဖိုး (n)ကို cast လုပ္ၿပီး long type ေျပာင္းလဲကာ ကိန္းျပည္႔တန္ဖိုးကို ေအာက္ပါအတိုင္း ရွာလိုက္ပါတယ္။

long temp = static\_cast<long>(n);

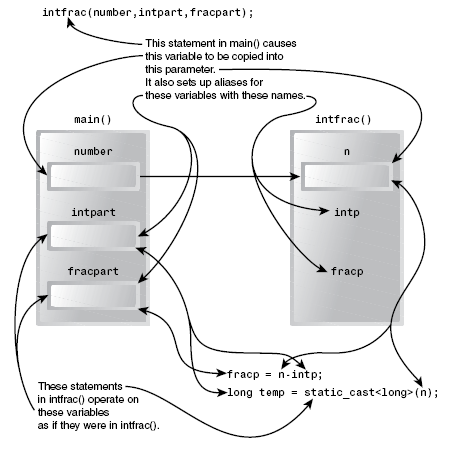
integer type ဟာ ကိန္းျပည္႔ပိုင္းေတြကိုပဲ သိမ္းဆည္းႏိုင္တဲ႔ အတြက္ ဒီနည္းကိုသံုးၿပီး ဒႆမဂဏန္းေတြကို ထိထိေရာက္ေရာက္ ျဖတ္ခ်ထားခဲ႔ႏိုင္ပါတယ္။ ရလာတဲ႔ ရလဒ္ကို ေနာက္တစ္ခါ cast ျပန္လုပ္ၿပီး ေအာက္ပါအတိုင္း float ျပန္ေျပာင္းလိုက္ပါတယ္-

intp = static\_cast<float>(temp);

ဒႆမေနာက္က တန္ဖိုးကို ရွာဖို႔အတြက္ကေတာ႔ မူရင္းတန္ဖိုးကို ကိန္းျပည္႔တန္ဖိုး ႏုတ္ေပးၿပီး အလြယ္တကူ ရွာႏိုင္ပါတယ္။

intfrac() function အေနနဲ႔ ကိန္းျပည္႔နဲ႔ ဒႆမေနာက္က ဂဏန္း ႏွစ္ခုစလံုးကို ရွာလို႔ရပါၿပီ။ ဒါေပမယ္႔ function တစ္ခုဟာ တန္ဖိုး တစ္ခုကိုပဲ return ျပန္ေပးလို႔ ရတဲ႔အတြက္ ႏွစ္ခုစလံုးကို ျပန္ေပးဖို႔ ဘယ္လိုလုပ္ၾကမလဲ? ဒီျပႆနာကို reference arguments ေတြ အသံုးျပဳၿပီး ေျဖရွင္းႏိုင္ပါတယ္။ ေအာက္မွာ ေဖာ္ျပထားတဲ႔ intfrac() function ရဲ႕ declarator ေလးကို ေလ႔လာၾကည္႔ရေအာင္-

**void intfrac(float n, float& intp, float& fracp)**



Refrence arguments ေတြကို ampersand (&) အသံုးျပဳရပါတယ္။ ဥပမာ -

**float& intp**

ဒီေနရာမွာ ampersand (&) ဟာ intp သည္ pass လုပ္လိုက္တဲ႔ argument အတြက္ alias ဒါမွမဟုတ္ အျခားအမည္တစ္ခု ျဖစ္တယ္ဆိုတာကို ေဖာ္ျပေပးပါတယ္။ နားလည္ရလြယ္ကူေအာင္ ရွင္းျပရမယ္ ဆိုရင္ေတာ႔ intfrac() function ထဲက intp ကို အသံုးျပဳတိုင္း တကယ္တမ္းက main() ထဲက intpart ကို ရည္ညႊန္းသံုးစြဲေနတာပဲ ျဖစ္ပါတယ္။

ampersand (&) ဟာ reference လို႔ ဆိုလိုပါတယ္။ ဒါေၾကာင္႔ float& intp ဟာ intp သည္ သူ႔ကို pass လုပ္မယ္႔ variable ရဲ႕ reference ျဖစ္တယ္လို႔ ဆိုလိုပါတယ္။ ထိုနည္းတူပဲ fracp ဟာ fracpart ရဲ႕ alias တစ္ခု ျဖစ္ပါတယ္။ Function ရဲ႕ declaration ကို ေရးသားတဲ႔ အခါမွာလည္း definition ထဲကလို & သံုးၿပီး ေအာက္ပါအတိုင္း ေရးရပါမယ္။

**void intfrac(float, float&, float&); // ampersands**

Function definition မွာလိုပဲ passed by reference ျပဳလုပ္မယ္႔ arguments ေတြေနာက္မွာ ampersnad & ထည္႔ေပးရပါမယ္။ ဒါေပမယ္႔ function call ထဲမွာေတာ႔ ampersnad & ကို ထည္႔ေပးစရာ မလိုေတာ႔ပါဘူး။

**intfrac(number, intpart, fracpart); // no ampersands**

ဒါေၾကာင္႔ function call ကို ၾကည္႔လိုက္ရံုနဲ႔ ဘယ္ argument က passed by reference ဒါမွမဟုတ္ value ျဖစ္တယ္ဆိုတာကို မသိႏိုင္ပါဘူး။ ဒီဥပမာမွာေတာ႔ intpart နဲ႔ fracpart ေတြဟာ passed by reference ျဖစ္ၿပီး၊ variable number ကေတာ႔ passed by value ျဖစ္ပါတယ္။

intp နဲ႔ intpart ေတြဟာ fracp နဲ႔ fracpart ဆိုတဲ႔ အျခားနာမည္ေတြရွိတဲ႔ memory ထဲက ေနရာတစ္ခုသာ ျဖစ္ပါတယ္။ အမည္မတူေပမယ္႔ ေနရာကေတာ႔ တစ္ခုထဲပါ။ ဘယ္တစ္ခုကိုျပင္ျပင္ memory ထဲမွာ ျပင္ၿပီးသားျဖစ္ၿပီး ႏွစ္ခုစလံုး တန္ဖိုးေတြ အတူတူေျပာင္းလဲသြားမွာ ျဖစ္ပါတယ္။

ေနာက္ထပ္ intfrac() function ထဲက parameter တစ္ခုျဖစ္တဲ႔ n ကေတာ႔ main() function ထဲက number နဲ႔ မတူညီဘဲ သီးျခား variable တစ္ခုျဖစ္ပါတယ္။ ဆိုလိုတာက အမည္လည္း မတူသလို memory ထဲက ေနရာလည္း မတူညီပါဘူး။ number ထဲက တန္ဖိုးကို n ထဲကို ေကာ္ပီကူးထည္႔ ေပးလိုက္တာပါ။ number ကို တန္ဖိုးေျပာင္းထည္႔ဖို႔ မလိုတဲ႔အတြက္ passed by reference မျပဳလုပ္တာပဲျဖစ္ပါတယ္။ C programmer ေတြအေနနဲ႔ကေတာ႔ ampersand ကို address of သေကၤတနဲ႔ မွားတတ္တဲ႔အတြက္ သတိျပဳေစလိုပါတယ္။

**A More Complex Pass by Reference**

ေနာက္ထပ္ pass by reference ျပဳလုပ္နည္းေတြကို ဆက္လက္ေလ႔လာပါ႔မယ္။ ဒီဥပမာမွာေတာ႔ အနည္းငယ္ ပိုမိုရႈပ္ေထြးတဲ႔ နည္းလမ္းမ်ားပါဝင္လာပါတယ္။ ကၽြန္ေတာ္တို႔မွာ ဂဏန္းတြဲေတြ ရွိၿပီး ငယ္တဲ႔ တစ္ခုကို အျမဲေရွ႕က ထားခ်င္တယ္ဆိုၾကပါစို႔။ အဲဒီလို ျပဳလုပ္ဖို႔အတြက္ order() ဆိုတဲ႔ function ေလးကို ေရးပါမယ္။ အဲဒီ ထဲမွာ ဂဏန္းႏွစ္ခုကို pass by reference ျပဳလုပ္ၿပီး ထည္႔သြင္းလိုက္ပါမယ္။ ၄င္းတို႔ကို စစ္ေဆးၿပီး ေရွ႕က ဂဏန္းက ႀကီးေနခဲ႔ရင္ ေနရာခ်င္း လဲေပးလိုက္မွာ ျဖစ္ပါတယ္။ reforder.cpp ကို ေလ႔လာၾကည္႔ၾကရေအာင္-

// reforder.cpp

// orders two arguments passed by reference

#include <iostream>

using namespace std;

int main()

{

void order(int&, int&); //prototype

int n1=99, n2=11; //this pair not ordered

int n3=22, n4=88; //this pair ordered

order(n1, n2); //order each pair of numbers

order(n3, n4);

cout << “n1=” << n1 << endl; //print out all numbers

cout << “n2=” << n2 << endl;

cout << “n3=” << n3 << endl;

cout << “n4=” << n4 << endl;

return 0;

}

//--------------------------------------------------------------

void order(int& numb1, int& numb2) //orders two numbers

{

if(numb1 > numb2) //if 1st larger than 2nd,

{

int temp = numb1; //swap them

numb1 = numb2;

numb2 = temp;

}

}

main() ထဲမွာ ဂဏန္းႏွစ္စံု ရွိပါတယ္။ ပထမစံုတြဲက အစီအစဥ္မက်ဘဲ ဒုတိယ စံုတြဲက အစီအစဥ္က်ေနပါတယ္။ ဂဏန္းႏွစ္စံု စလံုးကို order() function ေခၚယူၿပီး ထည္႔သြင္းေပးလိုက္ပါတယ္။ အဲဒီအခါမွာ အစီအစဥ္မက်တဲ႔ ပထမ စံုတြဲကို ေနရာလဲၿပီး အစီအစဥ္က်ေအာင္ ေျပာင္းလဲေပးလိုက္ပါတယ္။ နဂိုကထဲက အစီအစဥ္က်ေနတဲ႔ ဒုတိယ စံုတြဲကိုေတာ႔ မေျပာင္းလဲေပးပါဘူး။

n1=11

n2=99

n3=22

n4=88

order() function အတြင္းမွာ ပထမ variable ကို numb1 လို႔ ေခၚၿပီး ဒုတိယ variable ကို numb2 လို႔ ေခၚပါတယ္။ တကယ္လို႔ numb1 က numb2 ထက္ ႀကီးေနခဲ႔မယ္ဆိုရင္ numb1 တန္ဖိုးကို temp ထဲ ခဏ ထည္႔ထားလိုက္ပါတယ္။ ေနာက္ numb2 ကို numb1 ထဲ ထည္႔လိုက္ၿပီး ေနာက္ဆံုးမွာေတာ႔ temp ထဲခဏထည္႔ထားတဲ႔ တန္ဖိုးကို numb2 ထဲ ေျပာင္းထည္႔ေပးလိုက္မွာ ျဖစ္ပါတယ္။

ဒီေနရာမွာ မွတ္သားရမယ္႔ အခ်က္ကေတာ႔ numb1 နဲ႔ numb2 ဟာ ထည္႔ေပးလိုက္မယ္႔ pass by reference arguments ေတြအတြက္ ေနာက္ထပ္ နာမည္တစ္မ်ိဳး ျဖစ္လာမွာပါ။ ဒီဥပမာမွာေတာ႔ n1 နဲ႔ n2 ကို function ပထမအႀကိမ္ေခၚစဥ္ ထည္႔သြင္းေပးလိုက္မွာ ျဖစ္ၿပီး n3 နဲ႔ n4 ကို ဒုတိယအႀကိမ္ ေခၚယူစဥ္မွာ ထည္႔သြင္းေပးရမွာ ျဖစ္ပါတယ္။ အစီအစဥ္မက်တဲ႔ ဂဏန္းစံုတြဲ ျဖစ္ေနရင္ေတာ႔ ဂဏန္းခ်င္း လဲေပးလိုက္မွာပါ။

Reference arguments ေတြကို အသံုးျပဳရတာဟာ အေဝးထိန္းစနစ္တစ္ခုကို အသံုးျပဳရသလိုပါပဲ။ ေခၚယူတဲ႔ ပရိုဂရမ္က function ကို သူ႔ထဲက ဘယ္ variables ေတြကို လုပ္ေဆာင္ရမယ္ဆိုတာကို ေျပာျပလိုက္ရံုနဲ႔ အဲဒီ function က ေခၚယူတဲ႔ ပရိုဂရမ္ထဲက variables ေတြကို သူတို႔ရဲ႕ နာမည္ေတာင္ မသိဘဲနဲ႔ ေျပာင္းလဲေပးလိုက္မွာပဲ ျဖစ္ပါတယ္။ ျမင္သာေအာင္ ဥပမာေပးရမယ္ဆိုရင္ အိမ္ကို ေဆးသုတ္ဖို႔ အျပင္က ကုမၸဏီ တစ္ခုကို ဖုန္းဆက္ၿပီး မွာလိုက္တဲ႔အခါ အဲဒီလူေတြ တကယ္ေရာက္မလာပဲနဲ႔ ကၽြန္ေတာ္တို႔ အိမ္ေလး ေမွာ္ဆန္ဆန္ ေဆးေရာင္ေတြ ေျပာင္းသြားသလိုပါပဲ ခင္ဗ်ာ။

**Passing Structures by Reference**

ကၽြန္ေတာ္တို႔အေနနဲ႔ သာမန္ data type ေတြကို pass by reference လုပ္လို႔ ရသလို structures ေတြကိုလဲ pass by reference လုပ္လို႔ရပါတယ္။ ေအာက္ပါ referst.cpp မွာ type Distance ရဲ႕ တန္ဖိုးေတြကို scale ေျပာင္းလဲေပးတဲ႔ အေၾကာင္း pass by reference လုပ္ၿပီး ေရးသားထားပါတယ္။ ဒီနည္းဟာ လက္ေတြ႔မွာ အသံုးတည္႔ပါတယ္။ အေဆာက္အအံု တစ္ခုရဲ႕ အရြယ္အစားအကုန္လံုးကို scale factor ေျပာင္းလဲေပးလိုက္တာနဲ႔ အရြယ္အစားကို အခ်ိဳးက် ခ်ံဳ႕ၿပီး၊ ခ်ဲ႕ၿပီး ရရွိမွာ ျဖစ္ပါတယ္။ referst.cpp ကို ေလ႔လာၾကည္႔ၾကရေအာင္-

// referst.cpp

// demonstrates passing structure by reference

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct Distance //English distance

{

int feet;

float inches;

};

////////////////////////////////////////////////////////////////

void scale( Distance&, float ); //function

void engldisp( Distance ); //declarations

int main()

{

Distance d1 = { 12, 6.5 }; //initialize d1 and d2

Distance d2 = { 10, 5.5 };

cout << “d1 = “; engldisp(d1); //display old d1 and d2

cout << “\nd2 = “; engldisp(d2);

scale(d1, 0.5); //scale d1 and d2

scale(d2, 0.25);

cout << “\nd1 = “; engldisp(d1); //display new d1 and d2

cout << “\nd2 = “; engldisp(d2);

cout << endl;

return 0;

}

//--------------------------------------------------------------

// scale()

// scales value of type Distance by factor

void scale( Distance& dd, float factor)

{

float inches = (dd.feet\*12 + dd.inches) \* factor;

dd.feet = static\_cast<int>(inches / 12);

dd.inches = inches - dd.feet \* 12;

}

//--------------------------------------------------------------

// engldisp()

// display structure of type Distance in feet and inches

void engldisp( Distance dd ) //parameter dd of type Distance

{

cout << dd.feet << “\’-” << dd.inches << “\””;

}

referst.cpp မွာ Distance variables—d1 နဲ႔ d2 ကို initialize ျပဳလုပ္ၿပီး တန္ဖိုးေတြ သတ္မွတ္ေပးပါတယ္။ ထို႔ေနာက္မွာ scale() function ကို ေခၚယူၿပီး d1 ကို 0.5 နဲ႔ d2 ကို 0.25 နဲ႔ အသီးသီး ေျမွာက္ပါတယ္။ ရလာတဲ႔ အေျဖေတြကို ေနာက္ဆံုးမွာ ထုတ္ေဖာ္ျပသေပးပါတယ္။ ေအာက္မွာ နမူနာ စမ္းသပ္ျပထားပါတယ္။

d1 = 12’-6.5”

d2 = 10’-5.5”

d1 = 6’-3.25”

d2 = 2’-7.375”

function scale() ကို ေခၚယူတဲ႔ ကုဒ္ႏွစ္ေၾကာင္းကေတာ႔ ေအာက္ပါအတိုင္း ျဖစ္ပါတယ္-

scale(d1, 0.5);

scale(d2, 0.25);

ပထမကုဒ္မွာ d1 ကို 0.5 နဲ႔ ေျမွာက္ေစၿပီး ဒုတိယကုဒ္မွာေတာ႔ d2 ကို 0.25 နဲ႔ ေျမွာက္ေစမွာ ျဖစ္ပါတယ္။ မွတ္သားသင္႔တဲ႔ အခ်က္ကေတာ႔ အဲဒီလိုျပဳလုပ္ရာမွာ d1 နဲ႔ d2 တန္ဖိုးေတြ တိုက္ရိုက္ ေျပာင္းလဲသြားတာပဲ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ဒီဥပမာမွာ တန္ဖိုးတစ္ခုကိုပဲ ေျပာင္းလဲခ်င္တာ ျဖစ္တဲ႔အတြက္ pass by reference ကို မသံုးဘဲ pass by value ကိုသံုးၿပီး return နဲ႔ျပန္ေပးနိုင္ဖို႔ ျပင္ေရးလို႔ရပါတယ္။ အဲဒီလို function မ်ိဳးကို ေခၚယူဖို႔အတြက္ဆိုရင္ ေအာက္ပါအတိုင္း ျပဳလုပ္ႏိုင္ပါတယ္-

d1 = scale(d1, 0.5);

**Notes on Passing by Reference**

C programming language မွာတုန္းကေတာ႔ References ေတြ မရွိေသးပါဘူး။ အဲဒီအစား pointers ေတြကို အသံုးျပဳေနၾကတာ ျဖစ္ပါတယ္။ C++ မွာေတာ႔ သာမန္ variables ေတြသာမက objects ေတြအတြက္ပါ ပိုမိုေကာင္းမြန္စြာနဲ႔ အဆင္ေျပေျပေရးသားႏိုင္ေအာင္ references ေတြကို ထည္႔သြင္းေပးထားတာ ျဖစ္ပါတယ္။ Function ေတြထဲကို arguments ေတြထည္႔သြင္းႏိုင္တဲ႔ တတိယ နည္းလမ္းကိုေတာ႔ ေနာက္ပိုင္း Pointer အခန္းက်မွ ဆက္လက္ရွင္းျပေပးသြားမွာပါ။

**Overloaded Functions**

Overloaded function မ်ားသည္ ၄င္းတို႔အား ေပးပို႔လိုက္ေသာ အခ်က္အလက္မ်ား (input parameter(s)) မ်ားအေပၚ မူတည္၍ မတူညီေသာ လုပ္ေဆာင္ခ်က္မ်ားကို ျပဳလုပ္ေပးသည္။ Oveloading သည္ ကၽြန္ေတာ္ ၾကားဘူးသည္႔ ဟာသ တစ္ခုႏွင္႔ တူညီပါသည္။ ေက်ာ္ၾကားေသာ သိပၸံပညာရွင္ႀကီး တစ္ေယာက္က ဓာတ္ဗူးကို အႀကီးက်ယ္ဆံုးေသာ တီထြင္မႈဟု အေၾကာက္အကန္ ျငင္းခံုပါသည္။ ၄င္း၏ အဆိုမွာ ဓာတ္ဗူးသည္ ပူေသာ အရာမ်ားကို ပူေအာင္ ထိန္းထားႏိုင္ၿပီး ေအးေသာအရာမ်ားကို ဆက္ေအးေနေအာင္ ထိန္းထားႏိုင္စြမ္း ရွိေသာေၾကာင္႔ အလြန္ထူးဆန္း အံ႔ၾသဖြယ္ရာပင္ ျဖစ္သည္ ဟုဆိုသည္။ ဓာတ္ဗူးက ဘယ္လိုလုပ္ၿပီး သိသလဲဟု ၄င္းက ေစာဒက တက္ခဲ႔ပါသည္။ overloaded function သည္လည္း ထိုကဲ႔သို႔ပင္ အံ႔ၾသဖြယ္ရာ သိရွိၿပီး လုပ္ေဆာင္ေပးႏိုင္ပါသည္။ ၄င္းသည္ ပထမ အခ်က္အလက္ တစ္ခုအတြက္ လုပ္ေဆာင္မႈ တစ္ခုကို လုပ္ေပးၿပီး အျခားအခ်က္အလက္အတြက္ သက္ဆိုင္ရာ ေနာက္လုပ္ေဆာင္မႈ တစ္ခုကို ျပဳလုပ္ေပးႏိုင္ပါသည္။ အဆိုပါ သေဘာသဘာဝကို ဥပမာမ်ားျဖင္႔ ရွင္းလင္းသြားမည္ ျဖစ္ပါသည္။

**Different Numbers of Arguments**

ယခင္ ဥပမာမ်ား (table.cpp ႏွင္႔ tablearg.cpp) တြင္ အသံုးျပဳခဲ႔ေသာ functions မ်ားကို ျပန္လည္ ေလ႔လာၾကည္႔ၾကပါစို႔။ starline() function သည္ ၾကယ္ ၄၅ လံုးကို print ထုတ္ေပးသည္။ repchar() function တြင္မူ ပံုေသ သတ္မွတ္ထားျခင္း မရွိဘဲ ေခၚယူေသာ program မွ ထည္႔သြင္းေပးလိုက္သည္႔ character ႏွင္႔ အလံုးအေရအတြက္ကို print ထုတ္ေပးမည္ျဖစ္သည္။ အကယ္၍ charline() function ကို ေရးသားကာ calling program မွ ထည္႔သြင္းေပးလိုက္ေသာ character အလံုးေရ ၄၅ လံုးကို print ထုတ္ေပးမည္ ဆိုၾကပါစို႔။ အဆိုပါ starline(), repchar(), ႏွင္႔ charline() function သံုးမ်ိဳးစလံုးသည္ ဆင္တူေသာ လုပ္ေဆာင္ခ်က္ကို လုပ္ေဆာင္ၾကၿပီး အမည္ မတူညီသည္ကို သတိထားမိပါလိမ္႔မည္။ ပရိုဂရမ္မာမ်ား အေနျဖင္႔ အဆိုပါ functions မ်ားကို အသံုးျပဳရန္ နာမည္ သံုးခုကို မွတ္သားထားရန္ လိုၿပီး ေရးသားေနေသာ ပရိုဂရမ္အတြက္ စုစည္းထားေသာ *Function Reference* documentation တြင္ အကၡစဥ္အလိုက္ စီစဥ္ထားေသာ အထဲမွ အမည္သံုးခုစာ သံုးေနရာ လိုက္လံ ရွာေဖြေနရမည္ ျဖစ္သည္။ အကယ္၍ လုပ္ေဆာင္ခ်က္ ဆင္တူေသာ functions မ်ားကို input arguments အေရအတြက္ႏွင္႔ data types မ်ား မတူညီသည္႔တိုင္ေအာင္ နာမည္တူ ေပးခဲ႔လွ်င္ အသံုးျပဳမည္႔ ပရိုဂရမ္မာအတြက္ ပိုမို အဆင္ေျပႏိုင္မည္ ျဖစ္သည္။ ေအာက္ပါ overload.cpp ပရိုဂရမ္တြင္ ေလ႔လာၾကည္႔ၾကပါစို႔။

// overload.cpp

// demonstrates function overloading

#include <iostream>

using namespace std;

void repchar(); //declarations

void repchar(char);

void repchar(char, int);

int main()

{

repchar();

repchar(‘=’);

repchar(‘+’, 30);

return 0;

}

//--------------------------------------------------------------

// repchar()

// displays 45 asterisks

void repchar()

{

for(int j=0; j<45; j++) // always loops 45 times

cout << ‘\*’; // always prints asterisk

cout << endl;

}

//--------------------------------------------------------------

// repchar()

// displays 45 copies of specified character

void repchar(char ch)

{

for(int j=0; j<45; j++) // always loops 45 times

cout << ch; // prints specified character

cout << endl;

}

//--------------------------------------------------------------

// repchar()

// displays specified number of copies of specified character

void repchar(char ch, int n)

{

for(int j=0; j<n; j++) // loops n times

cout << ch; // prints specified character

cout << endl;

}

၄င္းပရိုဂရမ္ကို စမ္းသပ္ၾကည္႔လွ်င္ ေအာက္ပါအတိုင္း အကၡရာမ်ဥ္း သံုးေၾကာင္းကို ေတြ႔ရမည္ ျဖစ္ပါသည္။

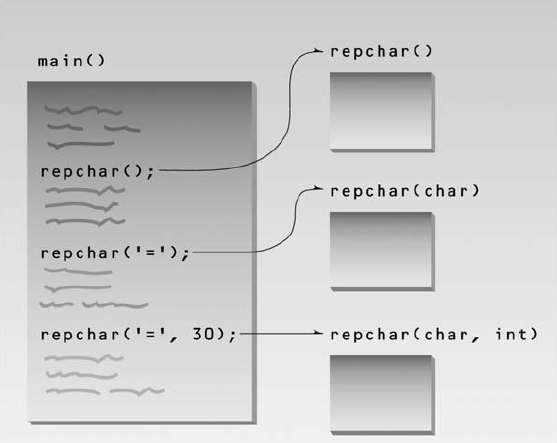
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=============================================

++++++++++++++++++++++++++++++

ပထမ ႏွစ္ေၾကာင္းသည္ အကၡရာ ၄၅ လံုးရွည္လ်ားၿပီး ေနာက္ဆံုး တတိယ အေၾကာင္းတြင္ အလံုးေရ ၃၀ ပါဝင္ပါသည္။ အဆိုပါ ပရိုဂရမ္တြင္ အမည္တူေသာ functions သံုးခု ပါဝင္ပါသည္။ function declarations သံုးခု၊ function calls သံုးခုႏွင္႔ function definitons သံုးခုတို႔ ပါဝင္ပါသည္။ ၄င္းတို႔ကို compiler က ခြဲျခားသိႏိုင္ျခင္းမွာ ၄င္းတို႔တြင္ ပါဝင္ေသာ မတူညီသည္႔ arguments အေရအတြက္ ႏွင္႔ မတူညီေသာ data types မ်ားပင္ ျဖစ္ပါသည္။

ေနာက္တစ္နည္းဆိုရလွ်င္ void repchar(); သည္ argument လံုးဝ မရွိဘဲ၊ char type argument တစ္ခုပါဝင္ေသာ void repchar(char); ႏွင္႔ char type argument တစ္ခုႏွင္႔ int type argument တစ္ခု ပါဝင္ေသာ void repchar(char, int); functions မ်ားႏွင္႔ မူညီဘဲ ကြဲျပားျခားနားေပသည္။ ၄င္းအခ်က္ကို compiler က ခြဲျခားႏိုင္စြမ္းရွိသျဖင္႔ အမည္တူေသာ္လည္း သက္ဆိုင္ရာ function ကိုသာ ေခၚယူ လုပ္ေဆာင္ေပးမည္ ျဖစ္သည္။



(ပံု-၁) function overloading ျပဳလုပ္ပံု

**Different Kinds of Arguments**

overload.cpp ပရိုဂရမ္တြင္ input arguments အေရအတြက္ မတူေသာ္လည္း အမည္တူသည္႔ functions သံုးခုကို ေရးသားထားပါသည္။ Compiler အေနျဖင္႔ arguments အေရအတြက္ မတူညီမႈျဖင္႔ ၄င္းတို႔အား ခြဲျခားၿပီး လုပ္ေဆာင္ေပးႏိုင္ခဲ႔ပါသည္။ သို႔ရာတြင္ Compiler အေနျဖင္႔ argument အေရအတြက္ တူညီေသာ္လည္း data type မတူညီျခင္းကို အေျခခံၿပီး ခြဲျခားေပးႏိုင္စြမ္းလည္း ရွိပါသည္။ overengl.cpp တြင္ ေလ႔လာၾကည္႔ႏိုင္ပါသည္။ ၄င္းတြင္ function အတြင္း ထည္႔သြင္း ေပးရမည္႔ တစ္ခုတည္းေသာ argument ကို structure type Distance ျဖစ္ေစ သို႔မဟုတ္ simple variable float type ျဖစ္ေစ အသံုးျပဳႏိုင္ပါသည္။ Compiler မွ ထည္႔သြင္းေပးလိုက္သည္႔ argument ၏ type ေပၚ မူတည္ၿပီး သက္ဆိုင္ရာ function ကို ေခၚယူေပးမည္ ျဖစ္ပါသည္။

// overengl.cpp

// demonstrates overloaded functions

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

struct Distance //English distance

{

int feet;

float inches;

};

////////////////////////////////////////////////////////////////

void engldisp( Distance ); //declarations

void engldisp( float );

int main()

{

Distance d1; //distance of type Distance

float d2; //distance of type float

//get length d1 from user

cout << “\nEnter feet: “; cin >> d1.feet;

cout << “Enter inches: “; cin >> d1.inches;

//get length d2 from user

cout << “Enter entire distance in inches: “; cin >> d2;

cout << “\nd1 = “;

engldisp(d1); //display length 1

cout << “\nd2 = “;

engldisp(d2); //display length 2

cout << endl;

return 0;

}

//--------------------------------------------------------------

// engldisp()

// display structure of type Distance in feet and inches

void engldisp( Distance dd ) //parameter dd of type Distance

{

cout << dd.feet << “\’-” << dd.inches << “\””;

}

//--------------------------------------------------------------

// engldisp()

// display variable of type float in feet and inches

void engldisp( float dd ) //parameter dd of type float

{

int feet = static\_cast<int>(dd / 12);

float inches = dd - feet\*12;

cout << feet << “\’-” << inches << “\””;

}

အထက္ပါ ပရိုဂရမ္တြင္ user အား distances တန္ဖိုး ႏွစ္ခုကို ထည္႔သြင္းေစပါသည္။ ပထမ အႀကိမ္တြင္ ေပ ႏွင္႔ လက္မ ကို ခြဲျခား ထည္႔သြင္းေစၿပီး ဒုတိယ အႀကိမ္တြင္ လက္မဖြဲ႕ၿပီးသား အတိုင္းအတာကိုသာ ထည္႔သြင္းေစပါသည္(ဥပမာ - 9'–1.5'' အစား 109.5 inches ဟု ထည္႔သြင္းေပးရန္ ျဖစ္သည္)။ ပရိုဂရမ္မွ ပထမအႀကိမ္တြင္ Distance type argument အသံုးျပဳထားသည္႔ engldisp(Distance) function ကို ေခၚယူၿပီး ဒုတိယအႀကိမ္တြင္ float type argument ကို အသံုးျပဳထားေသာ engldisp(float) function ကို ေခၚယူမည္ ျဖစ္ပါသည္။ ေအာက္ပါအတိုင္း နမူနာ စမ္းသပ္ အသံုးျပဳျပထားပါသည္။

Enter feet: 5

Enter inches: 10.5

Enter entire distance in inches: 76.5

d1 = 5’-10.5”

d2 = 6’-4.5”

သတိျပဳရမည္႔ အခ်က္တစ္ခုမွာ engldisp() functions ႏွစ္ခုစလံုးသည္ ဆင္တူေသာ လုပ္ေဆာင္မႈ ရွိေသာ္လည္း ကုဒ္မ်ားမွာ လံုးဝ ကြဲျပားျခားနားေပသည္။ လက္မဖြဲ႕ၿပီးသား တန္ဖိုးကို အသံုးျပဳသည္႔ function သည္ ေပ ႏွင္႔ လက္မ ျပန္ဖြဲ႕ၿပီးမွ ရလဒ္ကို ျပသေပးမည္ ျဖစ္ပါသည္။

Overloaded functions မ်ားကို အသံုးျပဳျခင္းျဖင္႔ ပရိုဂရမ္မာမ်ားသည္ function အမည္မ်ားစြာကို မွတ္သားရန္ မလိုေတာ႔သျဖင္႔ ေရးသားရသည္မွာ ပိုမိုလြယ္ကူလာေပသည္။ အကယ္၍ overloaded function အသံုးမျပဳခဲ႔လွ်င္ function နာမည္မ်ားစြာကို မွတ္သားရသျဖင္႔ မွားယြင္းႏိုင္ေပသည္။ ဥပမာအားျဖင္႔ absolute value ကို ရွာရန္ ေရးသားထားသည္႔ functions မ်ားသည္ C programming အတြက္ အမ်ိဳးမ်ိဳး ကြဲျပားေနေပသည္။ int data type အတြက္ abs()၊ complex numbers အတြက္ cabs()၊ type double အတြက္ fabs() ႏွင္႔ type long အတြက္ labs() ဟူ၍ အသီးသီး ခြဲျခားေရးသားထားရေပသည္။ C++ တြင္မူ abs() တစ္ခုတည္းျဖင္႔ ႀကိဳက္ႏွစ္သက္ရာ data type ကို ထည္႔သြင္းၿပီး အသံုးျပဳႏိုင္ပါသည္။ Function overloading ကို objects မ်ားအေၾကာင္း ေလ႔လာရာတြင္ ဆက္လက္ ရွင္းျပပါဦးမည္။

**နိဂံုး**

တကယ္ေတာ႔ function ေတြအေၾကာင္းကို ရွင္းျပဖို႔ အနည္းငယ္ က်န္ေနပါေသးတယ္။ ဒါေပမယ္႔ အေျခအေနအရ OOP in C++ အပိုင္း (၁) ကို ဒီေနရာမွာပဲ အဆံုးသတ္လိုက္ပါတယ္။ သိလိုသည္မ်ားကို aungwh2013@gmail.com ကို ဆက္သြယ္ၿပီး ေမးျမန္းႏိုင္သလို YCC ေက်ာင္းသားမ်ား အတြက္လည္း ေအာက္ေဖာ္ျပပါ လိပ္စာရွိ ေတာ္ဝင္ ကြန္ပ်ဴတာ စင္တာ၌လည္း ေလ႔လာ စံုစမ္းႏိုင္ပါတယ္ခင္ဗ်ာ။

**မွီျငမ္း**

* Object-Oriented Programming in C++(4th edition), Robert Lafore, Copyright©2002 by Sams Publishing: ISBN 0-672-32308-7

Dr. ေအာင္ဝင္းထြဋ္ (bluephoenix)

http://engineer4myanmar.blogspot.com

ေတာ္ဝင္ ကြန္ပ်ဴတာ စင္တာ

၁၇၉ စ၊ သုမဂၤလာ၊ ေစ်းေလး အေနာက္ဘက္၊ ျပင္ဦးလြင္ၿမိဳ႕