

OOP in C++

(အပိုင်း-၁)

Dr. အောင်ဝင်းထွန်း (BluePhoenix)

License: MIT Opensource License



စာရေးသူ၏ အမှာစာ

ကျွန်တော်အနေနဲ့ C နဲ့ C++ ကို ၁၉၉၉ ခုနှစ်လောက်မှာ စတင်လေ့လာခဲ့ပါတယ်။ အဲဒီနှစ်ထဲမှာပဲ တက္ကသိုလ်ကွန်ပျူတာခန်းထဲက ကွန်ပျူတာတွေထဲမှာ ပြန်ကြံနေတဲ့ Turbo C version 2.0 ဖိုင်လေးတွေကို စုစည်းပြီး C++ နဲ့ installer software ရေးဖြစ်ခဲ့ပါတယ်။ အဲဒီအတွေ့အကြုံကို ၂၀၀၀ ခုနှစ် အောက်တိုဘာလထုတ် ကွန်ပျူတာ ဂျာနယ် မှာ Installation software construction ဆိုတဲ့ ခေါင်းစဉ်နဲ့ ဆောင်းပါး တစ်စောင် ရေးသားခဲ့ပါတယ်။ အဲဒီဆောင်းပါးဟာ ကျွန်တော် ဘဝမှာ ပထမဆုံး ပုံနှိပ်ဖော်ပြခံခဲ့ရတဲ့ နည်းပညာ ဆောင်းပါး တစ်စောင်ပဲ ဖြစ်ခဲ့ပါတယ်။ ဒီလိုနဲ့ ကျွန်တော် ငယ်ဘဝမှာ အစွဲအလမ်းအကြီးဆုံးက C++ ဘာသာစကား ဖြစ်မှန်းမသိ ဖြစ်လာခဲ့ပါတယ်။

မိမိဘာသာ အများဆုံး လေ့လာခဲ့ရတဲ့အတွက် C++ လို့ဆိုပေမယ့် တကယ်တမ်းက ကျောင်းမှာ သင်ခဲ့ဘူးတဲ့ C language style နဲ့ပဲ ရေးဖြစ်တာ များပါတယ်။ OOP ဆိုတာကို အဲဒီတုန်းက ကြားဘူးရုံပဲ ရှိပါတယ်။ နောက်ပိုင်း နှစ်တော်တော်ကြာ အချိန်ယူပြီး ဖြည်းဖြည်းချင်း လေ့လာခဲ့ပါတယ်။

လွန်ခဲ့တဲ့ ၃ နှစ်လောက်က စပြီး ကျွန်တော်နဲ့ ရင်းနှီးတဲ့ YCC ကျောင်းသားအချို့ စာလာမေးတာကြောင့် သူတို့ လက်ရှိ အသုံးပြုနေတဲ့ သင်ရိုး စာအုပ်တွေကို ဖတ်ပြီး ပြန်ရှင်းပြ ပေးခဲ့ရပါတယ်။ အဲဒီစာအုပ်တွေထဲက စာမျက်နှာ တစ်ထောင်ကျော်ပါတဲ့ Robert Lafore ရဲ့ Object-Oriented Programming in C++ (Fourth Edition) စာအုပ်လေးဟာ C++ ကို OOP သုံးပြီး ရေးသား နည်းတွေကို အခြေခံကျကျ ရှင်းပြထားတဲ့ စာအုပ်ကောင်းတစ်အုပ် ဖြစ်တယ်ဆိုတာကို သတိထားမိခဲ့ပါတယ်။ ဒါကြောင့် အပတ်စဉ် အချိန်အနည်းငယ်ယူပြီး C++ သင်ခန်းစာလေးတွေကို အဲဒီ စာအုပ်ကနေ ကောက်နှုတ် ရေးသားဖို့ ဆုံးဖြတ်လိုက်တာ ဖြစ်ပါတယ်။ ဒီသင်ခန်းစာများဟာ C++ နဲ့ OOP ကို စတင်လေ့လာနေတဲ့ ကျောင်းသားများကိုသာ အဓိက ရည်ရွယ် ရေးသားသွားမှာ ဖြစ်ပါတယ် ခင်ဗျာ။

ဒေါက်တာ အောင်ဝင်းထွဋ်

7-5-2015

ပြင်ဦးလွင်မြို့

အခန်း(၁)

C++ Programming Basics

C++ ကို စတင်လေ့လာခြင်း

ပရိုဂရမ်းမင်း ဘာသာစကားတိုင်းမှာ အခြေခံအကျဆုံး ပရိုဂရမ် အသေးလေးတွေ ရေးဖို့အတွက်တောင် မရှိမဖြစ် လိုအပ်တဲ့ အခြေခံ စည်းမျဉ်းတွေ ရှိကြပါတယ်။ ဒီအခန်းမှာတော့ basic program construction, variables တွေနဲ့ input/output (I/O) တွေအကြောင်းကို အခြေခံကျကျ ရှင်းလင်း တင်ပြပေးသွားမှာပါ။ ဒါ့အပြင် comments ရေးသားနည်း၊ arithmetic operators များနှင့် increment operator အသုံးပြုနည်း၊ data conversion ပြုလုပ်ခြင်းနဲ့ library functions အသုံးပြုနည်းများကိုပါ ထည့်သွင်းဆွေးနွေးပေးသွားမှာပါ။ စာဖတ်သူတို့အနေနဲ့ ဘာတွေ ပြောနေတာလဲဆိုပြီး စိတ်မညစ်သွားကြပါနဲ့။ ဒီအခန်းကို ဆုံးအောင် ကြိုးစားဖတ်ကြည့်လိုက်တာနဲ့ ဒီအကြောင်းအရာတွေကို ကိုယ်တိုင် ရေးသားအသုံးပြုနိုင်အောင် ကျွမ်းကျင်သွားမှာပါ။

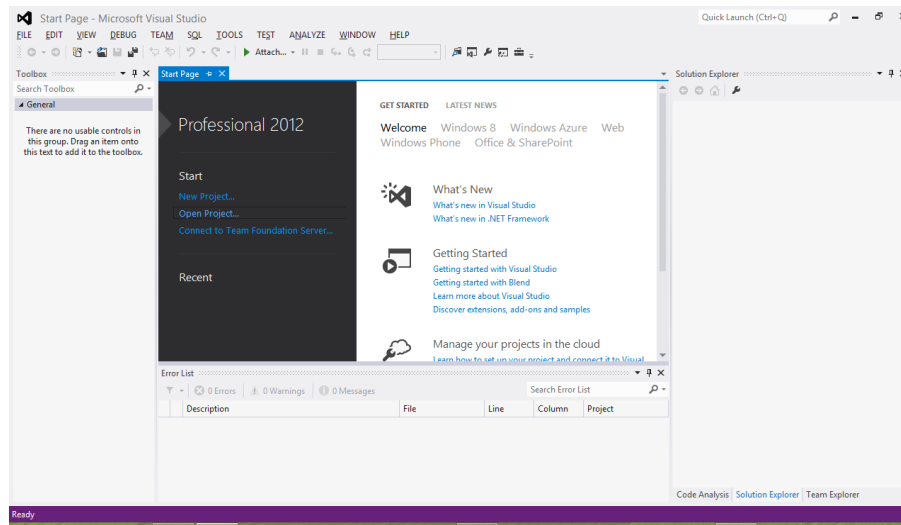
C++ ကို ဘယ်လို compiler တွေ အသုံးပြုပြီး ရေးကြမလဲ ?

ကျွန်တော် စတင်လေ့လာကာစ အချိန်များတုန်းက Turbo C++ 3.0 ကို အဓိက သုံးပြီး DOS command တွေနဲ့ အလုပ်ရှုပ်ခဲ့ကြပါတယ်။ အခု အချိန်မှာတော့ ကျောင်းသားအများစုဟာ C-Free ဆိုတဲ့ compiler ကို သုံးနေကြတာ သတိထားမိခဲ့ပါတယ်။ ဒါပေမယ့် ကျွန်တော့်အနေနဲ့ Visual Studio 2008 ဒါမှ မဟုတ် Visual Studio 2012 ကို သုံးသင့်တယ်လို့ အကြံပြုပါရစေ။ ဒီဆောင်းပါးများမှာတော့ Visual Studio 2012 ကို အသုံးပြုပြီး ရေးသားတင်ပြသွားမှာ ဖြစ်ပါတယ် ခင်ဗျာ။

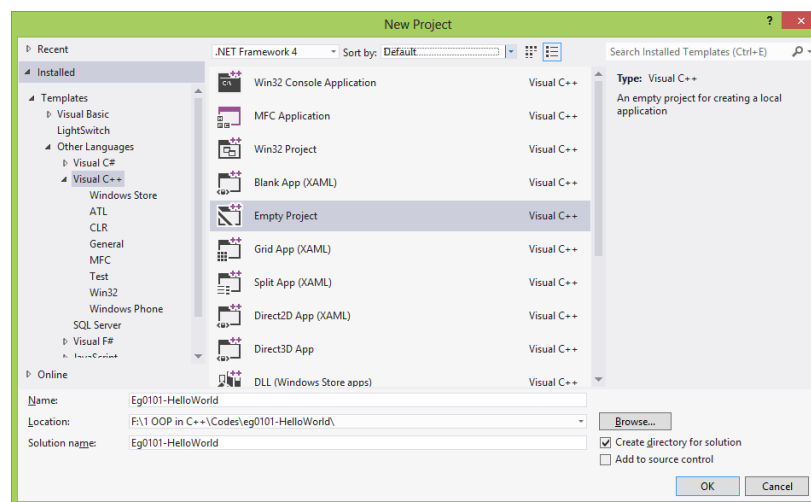
တကယ်တော့ compiler ဆိုတာက ကျွန်တော်တို့ ရေးသားထားတဲ့ source files (.CPP) ဖိုင်တွေကို Windows ပေါ်မှာ RUN လို့ ရတဲ့ executable files (.EXE) အဖြစ် ပြောင်းလဲပေးတဲ့ ပရိုဂရမ်လေး တစ်ခုပဲ ဖြစ်ပါတယ်။ အဲဒီ (.EXE) ဖိုင်လေးတွေကို compile တွေထဲကနေ တိုက်ရိုက် RUN လို့ ရသလို DOS command prompt ကလည်း RUN လို့ ရပါတယ်။

Visual Studio 2012 ကို အသုံးပြု၍ C++ program တစ်ခု ရေးသားစမ်းသပ်ခြင်း

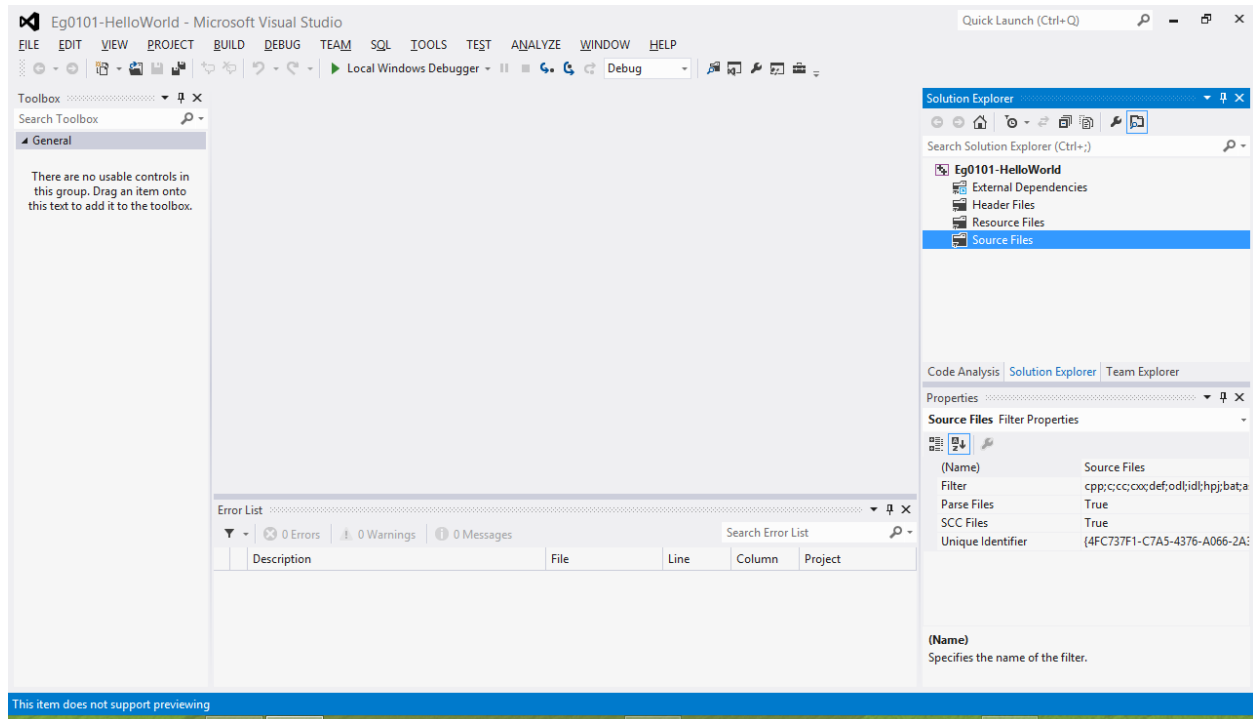
ပထမဆုံး Visual Studio 2012 ကို ဖွင့်လိုက်ပါ။ အောက်ပါ အတိုင်း မြင်ရနိုင်ပါတယ်။ ကိုယ်တင်ထားတဲ့ version အပေါ် မူတည်ပြီး အတိအကျတော့ တူချင်မှ တူပါလိမ့်မယ်။



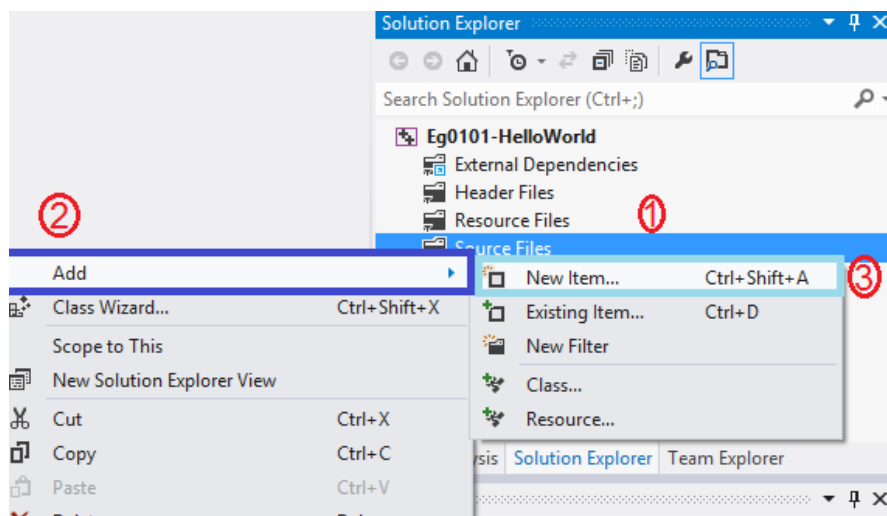
New Project... ကို နှိပ်လိုက်ပါ။ အောက်ပါအတိုင်း New Project dialog box ပေါ်လာပါလိမ့်မယ်။



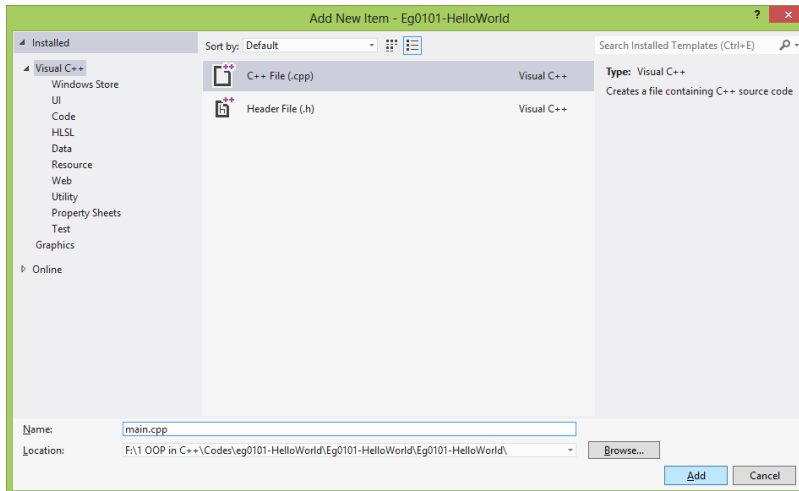
Installed->Templates->Other Languages->Visual C++ ကို ရွေးလိုက်ပါ။ ညာဘက်က ပေါ်လာတဲ့ Visual C++ project အမျိုးအစားတွေထဲက Empty Project ကို ရွေးချယ်လိုက်ပါ။ Name နေရာမှာ ကိုယ်ရေးမယ့် ပရိုဂရမ်ရဲ့ နာမည်ကို ရေးပါ။ ဖိုင်တွေ သိမ်းဖို့ နေရာကို Browse... လုပ်ပြီး ရှာပေးလိုက်ပါ။ OK ကို နှိပ်လိုက်ရင် အောက်ပါအတိုင်း မြင်ရပါလိမ့်မယ်။



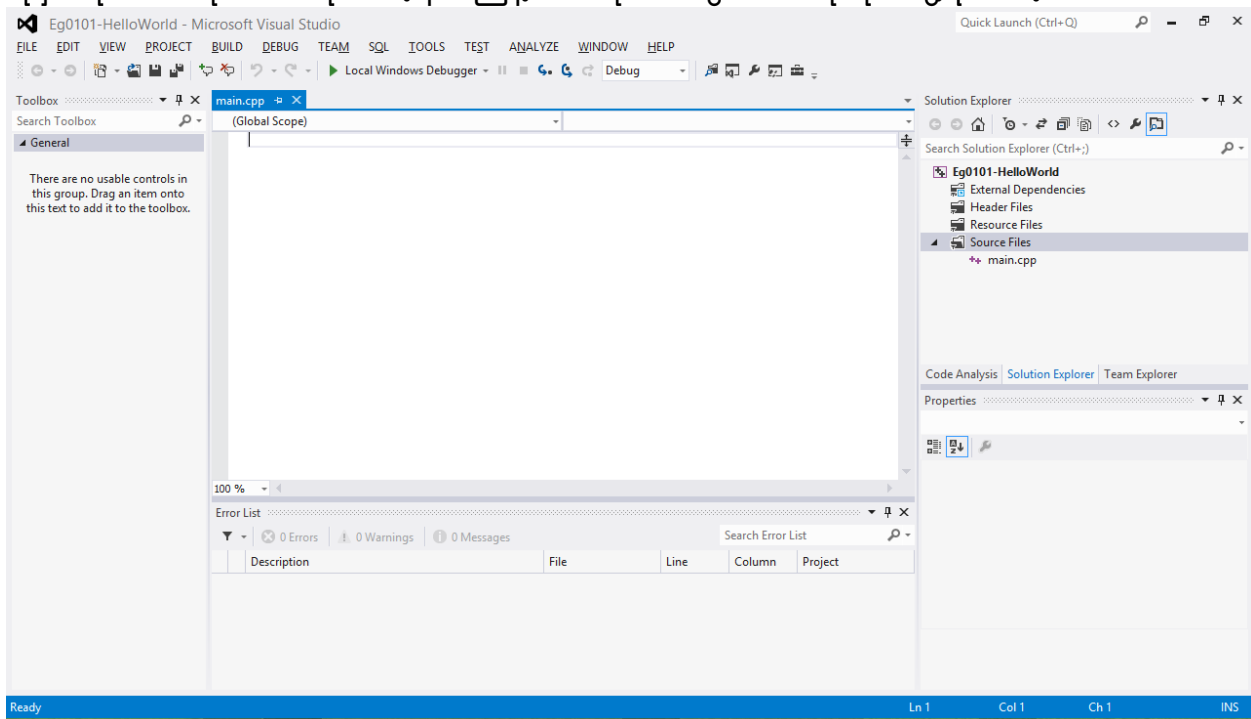
Empty Project ကို ပြုလုပ်ထားတာ ဖြစ်လို့ Source Files ဆိုတဲ့ folder ထဲမှာ ဘာဖိုင်မှ မရှိသေးပါဘူး။
 .CPP ဖိုင်တစ်ခု ထည့်သွင်းဖို့ အောက်ပါ အဆင့်များအတိုင်း လုပ်ဆောင်ပါ။



- (1) Solution Explorer ထဲက Source Files ကို right click ပြုလုပ်ပါ။
- (2) Add ကို ရွေးချယ်ပါ။
- (3) New Item.. ကို ကလစ် ပြုလုပ်ပါ။



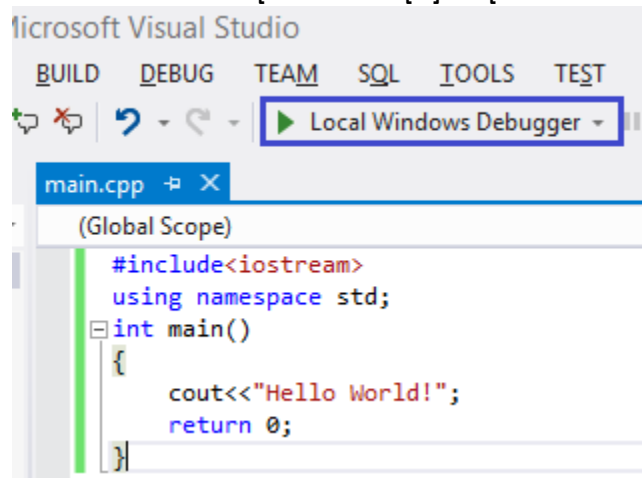
အထက်ပါ ပုံအတိုင်း Add New Item dialog box ပေါ်လာရင် C++ File(.cpp) ကို ရွေးချယ်ပါ။ Name နေရာတွင် မိမိ ရေးသားမယ့် ပရိုဂရမ် နာမည် (ဒီဥပမာမှာတော့ main.cpp) ပေးပြီး Add ကိုနှိပ်လိုက်ပါ။ ကိုယ်ပေးလိုက်တဲ့ နာမည်နဲ့ .cpp ဖိုင် အလွတ် တစ်ခုကို တွေ့ရပါလိမ့်မယ်။



အခုဆိုရင် coding များကို စတင် ဝင်ရောက် ရေးသားနိုင်ပြီ ဖြစ်ပါတယ်။ အောက်ပါ စာသားများကို ရိုက်ထည့်လိုက်ပါမယ်။

```
#include<iostream>
using namespace std;
int main()
{
    cout<<"Hello World!";
    return 0;
}
```

ဒီပရိုဂရမ်လေးဟာ အရိုးရှင်းဆုံး C++ ပရိုဂရမ်လေးပဲ ဖြစ်ပါတယ်။ Hello World! ဆိုတဲ့ စာသားလေးကို ကွန်ပိုက်တာ ဖန်သားပြင်ပေါ်မှာ ပြသပေးမယ့် ပရိုဂရမ်လေးပါ။ RUN ကြည့်ဖို့အတွက် Toolbar က Local Windows Debugger ဆိုတာလေးကို နှိပ်လိုက်ပါ။



Screen အမဲလေးတစ်ခု ပေါ်လာပြီး ချက်ချင်း ပြန်ပျောက်သွားပါလိမ့်မယ်။ အဲဒီ output ကို ကြည့်ဖို့ Visual Studio 2012 မှာ ကုဒ် အပိုလေးတွေ ထပ်ထည့်ပေးရပါမယ်။ ဒါမှ Screen ပျောက်မသွားဘဲ key တစ်ချက် အနှိပ်ကို စောင့်နေမှာ ဖြစ်ပါတယ်။ `_getch()` ဆိုတဲ့ function လေးကို `return 0;` ရဲ့ရှေ့မှာ ထပ်ထည့်ပေးရပါမယ်။ နောက်ပြီး သူ့ရဲ့ header file ဖြစ်တဲ့ `conio.h` ကို လည်း include ပြုလုပ်ပေးရမှာ ဖြစ်ပါတယ်။ ခုဆိုရင် ကုဒ်က အောက်ပါအတိုင်း ဖြစ်သွားပါပြီ။

```
main.cpp
(Global Scope)
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    cout<<"Hello World!";
    _getch();
    return 0;
}
```

အခုအချိန် ပြန် RUN ကြည့်လိုက်မယ် ဆိုရင်တော့ အောက်ပါအတိုင်း တွေ့ရမှာ ဖြစ်ပါတယ်။

ပရိုဂရမ်ကို အဆုံးသတ်ဖို့ Key တစ်ခုခု နှိပ်လိုက်ရုံပဲ ဖြစ်ပါတယ်။

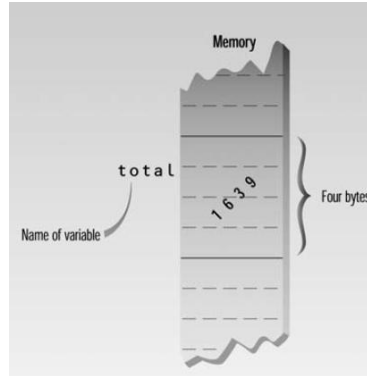
Data types and Variables

Variables တွေဟာ ပရိုဂရမ်းမင်း ဘာသာတိုင်းအတွက် အခြေခံ အကျဆုံး အစိတ်အပိုင်းများ ဖြစ်ကြပါတယ်။ အချက်အလက်တွေကို ယာယီ သိမ်းဆည်းဖို့ Variable တွေကို နာမည် သတ်မှတ်ပေးထားပြီး ဖန်တီးနိုင်သလို တန်ဖိုးတွေလဲ သတ်မှတ် ထည့်သွင်းနိုင်ပါတယ်။ Variables တွေကို ကွန်ပျူတာ မှတ်ဉာဏ် (RAM) ရဲ့ နေရာအချို့မှာ ဖန်တီးထားတာ ဖြစ်ပါတယ်။ ဒါကြောင့် Variable တစ်ခုကို တန်ဖိုးတစ်ခု ထည့်သွင်း လိုက်တယ်ဆိုတာဟာ တကယ်တော့ အဲဒီ Variable ကို သတ်မှတ်ပေးထားတဲ့ RAM ထဲမှာ အဲဒီတန်ဖိုးကို သိမ်းဆည်းထားလိုက်တာပဲ ဖြစ်ပါတယ်။ အဲဒီလို သိမ်းဆည်းထားတဲ့ နေရာမှာ အချက်အလက်ရဲ့ အမျိုးအစားပေါ်မူတည်ပြီး ခွဲခြားသိမ်းဆည်းပေးဖို့ လိုပါတယ်။ ဒါကြောင့် Variables တွေကို data types တွေ ခွဲခြားပြီး ဖန်တီးလေ့ ရှိပါတယ်။ ပရိုဂရမ်းမင်း ဘာသာစကားအများစုမှာ သုံးလေ့ ရှိတဲ့ data types တွေကတော့ integers၊ floating-point numbers တွေနဲ့ characters တွေပဲ ဖြစ်ကြပါတယ်။

Integer variable တွေဟာ 130000၊ ဒါမှ မဟုတ် -23 တို့လို အပြည့်ကိန်းတွေကို ရေတွက် ထည့်သွင်းရာမှာ အသုံးပြုပါတယ်။ floating-point numbers တွေနဲ့ မတူတဲ့ အချက်ကတော့ integers တွေမှာ ဒဿမနောက်က တန်ဖိုးတွေ (အပိုင်းကိန်း တန်ဖိုးတွေ) မပါရှိတာပဲ ဖြစ်ပါတယ်။

Defining Integer Variables

Integer variable တွေ အမျိုးအစားများစွာ ရှိပါတယ်။ ဒါပေမယ့် အသုံးအများဆုံးကတော့ int ပဲ ဖြစ်ပါတယ်။ နေရာယူတဲ့ မှတ်ဉာဏ် အရွယ်အစားကတော့ system ပေါ် မူတည်ပြီး ကွဲပြားနိုင်ပါတယ်။ ဥပမာ- 32-bit system Windows တွေမှာ int ဟာ 4 bytes (32 bits) နေရာယူပါတယ်။ ဒါကြောင့် int ဟာ -2,147,483,648 ကနေ 2,147,483,647 ကြား တန်ဖိုးရှိတဲ့ ဂဏန်းတွေကို သိမ်းဆည်းပေးနိုင်ပါတယ်။



ဒါပေမယ့် ယခင် Windows system အဟောင်းတွေနဲ့ MS-DOS စနစ်တွေမှာတော့ int ဟာ 2 bytes ပဲ နေရာယူပါတယ်။ data types တွေ နေရာယူတဲ့ သတ်မှတ်ချက်တွေကိုတော့ LIMITS ဆိုတဲ့ header file ထဲမှာ ရေးသားထားပါတယ်။ ကျွန်တော်တို့အနေနဲ့ လက်ရှိသုံးနေတဲ့ compiler ရဲ့ help ကနေလည်း ရှာဖွေ ဖတ်ရှုနိုင်ပါတယ်။

ဥပမာ-အနေနဲ့ int data type သုံးထားတဲ့ ပရိုဂရမ်လေးတစ်ခုကို ရေးပြပါမယ်။

```
//intvars.cpp
//demonstrates integer variables
#include<iostream>
using namespace std;
int main()
{
    int var1;           //define var1
    int var2;           //define var2
    var1 = 20;          //assign value to var1
    var2 = var1 + 10;    //assign value to var2
    cout << "var1+10 is "; //output text
    cout << var2 << endl; //output value of var2
    return 0;
}
```

အထက်ပါ ပရိုဂရမ်လေးမှာ **int var1; int var2;** ဆိုတဲ့ ကုဒ်နှစ်ကြောင်းဟာ integer variable var1 နဲ့ var2 တို့ကို ကြေငြာ သတ်မှတ်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ ရှေ့က int ဆိုတဲ့ keyword လေးက variable ရဲ့ type ကို သတ်မှတ်ပေးတာ ဖြစ်ပါတယ်။ အဲဒီ statement တွေကို declaration တွေလို့ ခေါ်ဆိုပြီး အခြား program statements တွေလိုပဲ စာကြောင်း အဆုံးမှာ semicolon (;) နဲ့ ပိတ်ပေးရမှာ ဖြစ်ပါတယ်။

ကျွန်တော်တို့ အနေနဲ့ variables တွေကို အသုံးပြုခင်မှာ ကြိုတင် ကြေငြာပေးထားရမှာ ဖြစ်ပါတယ်။ ဆန်သွားဝယ်မယ့်သူဟာ ဆန်အိတ်ကို ကြိုယူသွားရသလိုပဲ။ ဒါကြောင့် များသောအားဖြင့် ပရိုဂရမ်ရဲ့ အစပိုင်းမှာ variable တွေကို ကြေငြာထားလေ့ ရှိပါတယ်။ ဒါပေမယ့် တကယ်က ပရိုဂရမ်ရဲ့မည်သည့် နေရာမှာမဆို variable တွေကို ကြေငြာပေး နိုင်ပါတယ်။

Declarations and Definitions

Declaration နဲ့ Definition ကြားက ခြားနားချက်ကို ရှင်းပြချင်ပါတယ်။ Declare လုပ်တာဟာ variable ကို ဘယ်လိုခေါ်မယ်၊ ဘာ data type ဖြစ်တယ်ဆိုတာကို program အတွင်းမှာ မိတ်ဆက် ကြေငြာပေးတာ ဖြစ်ပါတယ်။ Definition ဆိုတာကတော့ memory ထဲမှာ အဲဒီ variable ကို နေရာယူလိုက်တာပဲ ဖြစ်ပါတယ်။ အထက်က ပရိုဂရမ်ထဲမှာ ရေးထားတဲ့ int var1; int var2; ဆိုတဲ့ ကုဒ်နှစ်ကြောင်းဟာ declarations တွေ ဖြစ်သလို definition တွေလည်း ဖြစ်ပါတယ်။ များသောအားဖြင့် အချို့နေရာတွေမှာ အဲဒီ အချက်နှစ်ခုက တူလေ့ ရှိပါတယ်။ ဒါပေမယ့် နောက်ပိုင်းမှာ definitions မဟုတ်တဲ့ declarations တွေ အကြောင်းကို အသေးစိတ် ရှင်းပြသွားမှာ ဖြစ်ပါတယ်။

Variable Names

Variables တွေကို နာမည်ပေးတဲ့ နေရာမှာ လိုက်နာရတဲ့ စည်းကမ်း အချို့ ရှိပါတယ်။

- နာမည်ကို စာလုံး အကြီး ဒါမှမဟုတ် အသေးနဲ့ ရေးနိုင်ပါတယ်။
- 1 ကနေ 9 အထိ ဂဏန်းတွေ ကိုလဲ ထည့်သွင်းနိုင်ပေမယ့် အစ စာလုံးကိုတော့ အကွာရန်ပဲ စရမှာ ဖြစ်ပါတယ်။
- နာမည် ကြားမှာ space ခြားလို့ မရပေမယ့် under-score (_) ကို တော့ အသုံးပြုလို့ ရပါတယ်။
- နာမည်ကို စိတ်ကြိုက် စာလုံးရေ အနည်းအများ ပေးနိုင်ပါတယ်။
- စာလုံး အကြီးနဲ့ အသေးကို compiler က ခွဲခြား သတ်မှတ်ထားတာ ဖြစ်လို့ var ဟာ VAR နဲ့ မတူနိုင်တာကို သတိပြုရမှာ ဖြစ်ပါတယ်။

နောက် ခြွင်းချက် တစ်ခုက C++ မှာ အသုံးပြုဖို့ သတ်မှတ်ထားတဲ့ keywords တွေကိုတော့ နာမည်ပေးတဲ့ အချိန်မှာ ချန်လှပ်ထားခဲ့ရမှာ ဖြစ်ပါတယ်။ (ဥပမာ- if, else, while, do, case, new, class, int, float, double, char,...)။

C++ ပရိုဂရမ်မှာ အများစုဟာ variable တွေကို နာမည်ပေးဖို့ စာလုံးအသေးတွေပဲ သုံးလေ့ရှိပါတယ်။ အချို့ကတော့ စာလုံးအကြီးနဲ့ အသေးကို ရောရေးလေ့ ရှိပါတယ်။ (ဥပမာ - IntVar, dataCount)။ const တွေကို ရေးသားတဲ့ နေရာမှာတော့ စာလုံးအကြီးတွေပဲ သုံးလေ့ ရှိကြပါတယ်။ ဘယ်လိုပုံစံမျိုးပဲရေးရေး ပရိုဂရမ်တစ်ပုဒ်မှာတော့ သတ်သတ်မှတ်မှတ် စတိုင် တစ်မျိုးနဲ့ပဲ ရေးသားသင့်ပါတယ်။ C++ မှာ class တွေနဲ့ function တွေကို နာမည်ပေးတဲ့ နေရာမှာလဲ အထက်ပါ ဥပဒေသများကိုပဲ အသုံးပြုပါတယ်။

Variable တွေကို နာမည်ပေးတဲ့ နေရာမှာ အဲဒီ variable ရဲ့ ရည်ရွယ်ချက်ကို လူတိုင်း နားလည်လွယ်မယ့် နာမည်မျိုးကို ပေးဖို့ လိုပါတယ်။ ဥပမာ ပြောရရင် bT ဒါမှမဟုတ် t ဆိုတာထက်စာရင် boilerTemperature လို့ နာမည်ပေးတာမျိုးက ပိုမိုရှင်းလင်းလွယ်ကူစေပါတယ်။

Assignment statements

အထက်ပါ ပရိုဂရမ်ထဲက `var1 = 20; var2 = var1 + 10;` ဆိုတဲ့ ကုဒ် နှစ်ကြောင်းကို လေ့လာကြည့်ကြရအောင်။ ညီမျှခြင်း သင်္ကေတ (=) ဟာ တကယ်တော့ သူ့ရဲ့ ညာဘက်က တန်ဖိုးတွေကို ဘယ်ဘက်က variable တွေထဲကို assign ပြုလုပ်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ ပထမ ကုဒ်စာကြောင်းမှာပါတဲ့ var1 ဟာ ယခင်က တန်ဖိုး မရှိသေးပေမယ့် ဒီကုဒ်ကြောင့် တန်ဖိုး 20 ဖြစ်လာပါတယ်။

Integer Constants

အဲဒီ ပထမ စာကြောင်းမှာ ပါတဲ့ 20 သည် integer constant ပဲ ဖြစ်ပါတယ်။ constants တွေဟာ ပရိုဂရမ် တစ်ပုဒ်မှာ တန်ဖိုး မပြောင်းလဲဘဲ တည်ရှိနေကြပါတယ်။ integer constant တွေကတော့ ဒဿမ တန်ဖိုး မပါရှိဘဲ integer ရဲ့ သတ်မှတ် တန်ဖိုးတွေ အတွင်းမှာ ရှိတဲ့ အပြည့်ကိန်း တန်ဖိုးတွေပဲ ဖြစ်ပါတယ်။ ကုဒ် ဒုတိယ စာကြောင်းမှာပါတဲ့ (+) သင်္ကေတကတော့ var1 ကို 10 ပေါင်းထည့်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ အဲဒီ 10 ကလဲ နောက်ထပ် integer constant တစ်ခုပဲ ဖြစ်ပါတယ်။ အဲဒီလို ပေါင်းလို့ ရလာတဲ့ တန်ဖိုးကို (=) operator က var2 ထဲကို assign ပြုလုပ်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။

Output Variations

နမူနာ ပရိုဂရမ်ထဲက `cout << "var1+10 is ";` ဆိုတဲ့ စာကြောင်းက ကျွန်တော်တို့ ယခင် သင်ခန်းစာတွေမှာ လေ့လာခဲ့သလိုမျိုး string constant တစ်ခု (`var1+10 is`) ကို display မှာ ပြသပေးမှာ ဖြစ်ပါတယ်။ နောက် စာသား တစ်ကြောင်း ဖြစ်တဲ့ `cout << var2 << endl;` ကတော့ variable `var2` ထဲမှာ သိမ်းဆည်းထားတဲ့ တန်ဖိုးကို ပြသပေးမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီပရိုဂရမ်ကို run ကြည့်လိုက်မယ်ဆိုရင် `var1+10 is 30` ဆိုတာကို မြင်ရမှာ ဖြစ်ပါတယ်။ မှတ်သားရမှာ တစ်ခုက `cout` နဲ့ `<<` operator တွေဟာ integer နဲ့ string တွေကို ခွဲခြား ကိုင်တွယ်နိုင်စွမ်း ရှိတယ် ဆိုတာပါပဲ။ ဒါကြောင့် C language မှာ `printf()` အသုံးပြုသလို (display လုပ်ရမယ့် variable သာမကပဲ) data type တွေ ခွဲခြားဖို့ `%d %c %s` တွေ ထည့်ပေးစရာမလိုဘဲ အဆင်ပြေပြေ လုပ်ဆောင်နိုင်စွမ်း ရှိပါတယ်။

ကျွန်တော်တို့ အနေနဲ့ `cout` နှစ်ကြောင်းက ထွက်လာတဲ့ အဖြေက တစ်ကြောင်းတည်း ဖြစ်နေတာကို သတိထားမိမှာပါ။ အလိုအလျောက် တစ်ကြောင်း ဆင်းသွားတာမျိုး မရှိပါဘူး။ လိုအပ်ရင် ကျွန်တော်တို့ ကိုယ်တိုင် ထည့်သွင်းပေးဖို့ လိုအပ်မှာ ဖြစ်ပါတယ်။ နောက်ပိုင်းမှာ `'\n'` လိုမျိုး escape sequence တွေနဲ့ manipulator တွေ အသုံးပြုပုံကို အသေးစိတ် ရှင်းလင်းသွားမှာ ဖြစ်ပါတယ်။

The endl Manipulator

`cout` နောက်ဆုံး စာကြောင်းထဲက နောက်ဆုံး စကားလုံးဟာ `endl` ဖြစ်ပါတယ်။ တကယ်တော့ တစ်ကြောင်း ဆင်းချင်တဲ့ နေရာမှာသုံးရတဲ့ manipulator တစ်မျိုးပဲ ဖြစ်ပါတယ်။ `'\n'` နဲ့ သဘောသဘာဝချင်း တူညီပေမယ့် အသုံးပြုရတာ အဓိပ္ပါယ် ပိုမိုရှင်းလင်းစေပါတယ်။ manipulator တွေကတော့ output stream ကို ပြုပြင်ပုံဖော်ပေးနိုင်တဲ့ ညွှန်ကြားချက်တွေပဲ ဖြစ်ပါတယ်။ `endl` အနေနဲ့ `'\n'` နဲ့ မတူတဲ့ အချက်ကတော့ output buffer ကို flush ပြုလုပ်ပေးနိုင်တာပဲ ဖြစ်ပါတယ်။

Other Integer Types

အထက်မှာ တင်ပြခဲ့သလိုပဲ int အပြင် အခြား integer data types တွေ ရှိနေပါသေးတယ်။ အသုံးအများဆုံးကတော့ long နဲ့ short ပဲ ဖြစ်ပါတယ်။ (တကယ်တော့ char ဆိုတာလဲ integer data type လို့ ပြောလို့ ရနိုင်ပါတယ်။ နောက်ပိုင်းမှာ အလျင်းသင့်သလို ရှင်းပြပေးသွားပါမယ်။) ကျွန်တော်တို့ အထက်မှာ တင်ပြခဲ့သလိုမျိုး int ဟာ system ပေါ်မူတည်ပြီး အရွယ်အစား ကွဲပြားပေမယ့် long နဲ့ short တို့ကတော့ system ပေါ် မူတည်မနေဘဲ အရွယ်အစား သတ်မှတ်ချက် ပုံသေ ဖြစ်ပါတယ်။

long အနေနဲ့ 4 bytes အမြဲနေရာယူလေ့ရှိပြီး 32-bit Windows system မှာ ရှိတဲ့ int နဲ့ အရွယ်အစားချင်း တူပါတယ်။ ဒါကြောင့် long ဟာ -2,147,483,648 ကနေ 2,147,483,647 ကြား တန်ဖိုးရှိတဲ့ ဂဏန်းတွေကို သိမ်းဆည်းပေးနိုင်ပါတယ်။ long ကို နောက်တစ်နည်း long int လို့လဲ ရေးနိုင်ပါတယ်။ အဓိပ္ပါယ်ကတော့ အတူတူပဲ ဖြစ်ပါတယ်။ 32-bit system မှာ long ကို သုံးရတာဟာ int နဲ့ တူညီနေလို့ သိပ်မထူးခြားပေမယ့် 16-bit system တွေမှာတော့ 4-bytes နေရာယူတဲ့ integer type တစ်ခုကို ဖန်တီးဖို့ အဆင်ပြေစေပါတယ်။ short ကတော့ 2-bytes နေရာယူပြီး 16-bit system က int နဲ့ အရွယ်အစား တူညီပါတယ်။ short ဟာ -32,768 ကနေ 32,767 ကြားမှာရှိတဲ့ ဂဏန်းတွေကို သိမ်းဆည်းနိုင်ပါတယ်။ short ကိုတော့ ခေတ်သစ် Windows system တွေမှာ သာမန်အားဖြင့် အသုံးပြုလေ့ မရှိဘဲ memory ကို ချွေတာဖို့ သိပ်လိုမှပဲ သုံးကြပါတော့တယ်။ int ဟာ short ထက် နှစ်ဆ အရွယ်အစား ပိုကြီးပြီး အချက်အလက်တွေကိုလဲ ပိုမိုမြန်ဆန်စွာ ရေးဖတ်နိုင်လို့ အသုံးများပါတယ်။ ကျွန်တော်တို့ အနေနဲ့ long data type အမျိုးအစား constant တစ်ခုကို ဖန်တီးချင်တယ် ဆိုရင် ဂဏန်းတန်ဖိုးရဲ့ နောက်မှာ L ကို ကပ်ထည့်ပေးလိုက်ရမှာ ဖြစ်ပါတယ်။ ဥပမာ -

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

အသုံးနည်းတဲ့ integer data types တွေ ရှိပါသေးတယ်။ compiler တွေပေါ်မူတည်ပြီး ကွဲပြားနိုင်ပါတယ်။ အဲဒီ integer တွေက သူတို့ နေရာယူတဲ့ bits အတိအကျကို သတ်မှတ်ထားပါတယ်။ (1-byte = 8-bits)။ အဲဒီ type တွေရဲ့ အမည်ရှေ့မှာ underscores နှစ်ခု ကို ရှေ့က ခံထားပါတယ်။ ဥပမာ- __int8, __int16, __int32, __int64 တို့ပဲ ဖြစ်ပါတယ်။ __int8 က 32-bit system က char နဲ့ တူညီပြီး၊ __int16 က short နဲ့ __int32 က int နဲ့ long တို့နဲ့ တူညီပါတယ်။ __int64 ကတော့ အကြီးဆုံး integer value ကို ဂဏန်း ၁၉ လုံးအထိ သိမ်းဆည်းနိုင်ပါတယ်။ ဒီ type တွေကို သုံးရာမှာ အရွယ်အစား ပုံသေဖြစ်တဲ့ အားသာချက် ရှိပေမယ့် လက်တွေ့မှာတော့ အသုံးနည်းတာကို တွေ့ရပါတယ်။

Character Variables

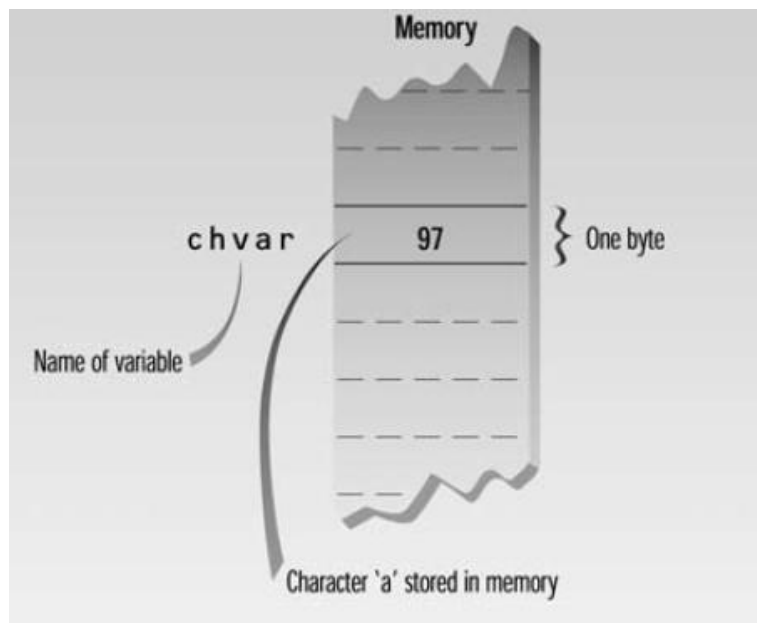
char type ဟာ -128 ကနေ 127 ကြားရှိတဲ့ integers တွေကို သိမ်းဆည်းပေးနိုင်ပါတယ်။ char variable ဟာ memory မှာ 1 byte (eight bits) ပဲ နေရာယူပါတယ်။ တခါတလေမှာ character variable တွေကို သူလက်ခံနိုင်တဲ့ တန်ဖိုးတွေကြားမှာ ဘောင်ဝင်တဲ့ ဂဏန်းတွေကို သိမ်းဖို့လဲ သုံးနိုင်ပါတယ်။ ဒါပေမယ့် များသောအားဖြင့် char variable တွေကို ASCII characters တွေ သိမ်းဆည်းဖို့ပဲ အသုံးချလေ့ ရှိပါတယ်။

တကယ်တော့ ASCII character set ဆိုတာဟာ 'a', 'B', '\$', '3' အစရှိတဲ့ characters တွေကို ဂဏန်းတွေနဲ့ ဖော်ပြတဲ့ ပုံစံပဲ ဖြစ်ပါတယ်။ အဲဒီ ဂဏန်းတွေဟာ 0 ကနေ 127 အထိ ရှိပါတယ်။ Windows system က အဲဒီ range ကို 255 လုံးအထိ တိုးချဲ့လိုက်ပြီး နိုင်ငံခြား ဘာသာစကားအချို့နဲ့ သင်္ကေတ အချို့ကို ထည့်သွင်းလိုက်ပါတယ်။

နိုင်ငံခြား ဘာသာစကားတွေကို ကိုင်တွယ်ဖြေရှင်းရတဲ့ အခါမှာ 128 နဲ့ 255 ကြား characters တွေဟာ စံသတ်မှတ်ထားတာ မရှိပါဘူး။ အရွယ်အစားကလဲ 1-byte ပဲရှိလို့ သိပ် သေးငယ်လွန်းတာကြောင့် ဂျပန်ဘာသာလိုမျိုး စကားလုံးအများကြီးပါတဲ့ ဘာသာစကားတွေကို ကိုင်တွယ်လို့ အဆင်မပြေပါဘူး။ Standard C++ မှာတော့ နိုင်ငံခြား ဘာသာစကားတွေကို ကိုင်တွယ်ဖို့ `wchar_t` ဆိုတဲ့ character type အကြီးစား တစ်ခု ပါဝင်ပါတယ်။ ဒါပေမယ့် ဒီသင်ခန်းစတွေမှာတော့ ASCII character set တွေကိုပဲ အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။

Character Constants

Character constants များသည် character တစ်လုံးကို single quotation marks (') နှစ်ခုကြားမှာ ထည့်သွင်းထားတာပဲ ဖြစ်ပါတယ်။ ဥပမာ - 'a', 'b'။ (string constant တွေကိုတော့ double quotation mark နှစ်ခုကြားမှာ ထည့်သွင်းပေးရပါတယ်)။ C++ compiler က အဲဒီ character constant တွေကို သက်ဆိုင်ရာ ASCII code တွေ အဖြစ် ပြောင်းလဲပေးပါတယ်။ ဥပမာ - 'a' ကို 97 အဖြစ် ဘာသာပြန်ယူမှာ ဖြစ်ပါတယ်။



Character variables တွေ အနေနဲ့ character constants တွေကို value အနေနဲ့ ထည့်သွင်းသိမ်းဆည်း ထားနိုင်ပါတယ်။ နမူနာ ပရိုဂရမ်လေးကို အောက်မှာ ရေးပြထားပါတယ်။

```
//charvars.cpp
//demonstrates character variables
#include <iostream>           //for cout, etc.
#include <conio.h>             //for _getch(), etc.
using namespace std;
int main()
{
    char charvar1 = 'A';      //define char variable as character
    char charvar2 = '\t';     //define char variable as tab
    cout << charvar1;         //display character
    cout << charvar2;         //display character
    charvar1 = 'B';           //set char variable to char constant
    cout << charvar1;         //display character
    cout << '\n';             //display new line character
    _getch();
    return 0;
}
```

Initialization

Variables တွေကို စတင်ဖန်တီးတဲ့ အချိန်မှာပဲ တန်ဖိုးတွေ တပါတည်း ထည့်သွင်းပေးလိုက်နိုင်ပါတယ်။ အဲဒီလို ထည့်သွင်း ပေးတာကို **initialization** လုပ်တယ်လို့ ခေါ်ပါတယ်။ အထက်က နမူနာ ပရိုဂရမ်လေးမှာ charvar1 နဲ့ charvar2 ဆိုတဲ့ char variable နှစ်ခုကို 'A' နဲ့ '\t' တို့ အသီးသီး ထည့်သွင်းပြီး initialized ပြုလုပ်ပေးထားတာပါ။

Escape Sequences

ပရိုဂရမ်လေးထဲမှာ ပါတဲ့ '\t' ဆိုတဲ့ character constant လေးဟာ အမြင်ဆန်းနေနိုင်ပါတယ်။ အဲဒီလို character constant တွေဟာ **escape sequence** တွေပဲ ဖြစ်ပါတယ်။ backslash (\) ကြောင့် ပုံမှန် characters တွေ ဘာသာပြန်တဲ့ စနစ်မျိုးကနေ escape ဖြစ်ပြီး လွတ်မြောက် နေတာကြောင့် အဲဒီလို ခေါ်ဆိုကြတာပါ။ ဒီ ဥပမာမှာတော့ t ကို character 't' လို့ ဘာသာပြန်လို့ မရနိုင်ဘဲ **tab character** အဖြစ် ပြန်ဆိုရပါမယ်။ console-mode program တွေမှာ tab တစ်ခုဟာ spaces ရှစ်နေရာ ယူမှာ ဖြစ်ပါတယ်။ နောက်ထပ် character constant တစ်ခုဖြစ်တဲ့ '\n' ကတော့ နောက်တစ်ကြောင်းဆင်းဖို့ ညွှန်ကြားချက် ပေးတာပါ။ Escape sequences တွေကို သီးခြားဖြစ်စေ၊

string constant (စာကြောင်း) တွေထဲမှာ ရောနှောထည့်သွင်းလို့ ဖြစ်စေ အသုံးပြုနိုင်ပါတယ်။ အောက်ပါ ဇယားလေးကတော့ အသုံးများတဲ့ escape sequences တွေကို ဖော်ပြပေးထားပါတယ်။

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

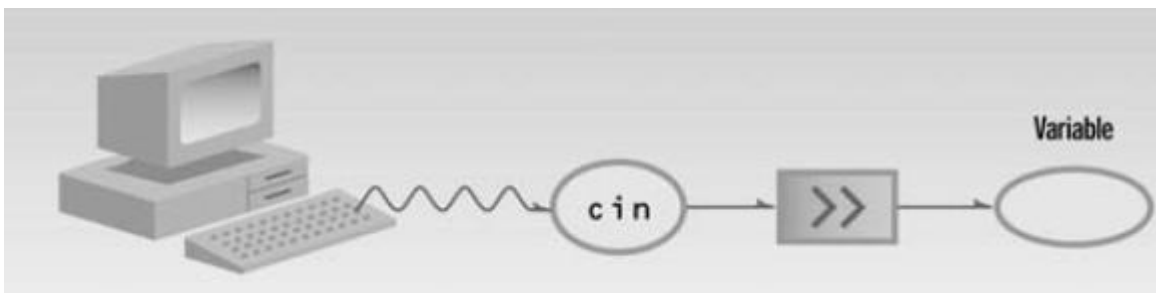
C++ မှာ backslash (\)၊ single quotation marks '၊ double quotation marks " တွေကို သီးခြား ရည်ရွယ်ချက်တွေနဲ့ အသုံးပြုထားတာပါ။ ဒါကြောင့် အဲဒီ character တွေကို constants အနေနဲ့ ဖော်ပြချင်တဲ့ အခါ အရှေ့က backslash(\) တစ်ခု ထပ်ထည့်ပြီး escape sequence အနေနဲ့ အသုံးပြုရတာ ဖြစ်ပါတယ်။ ဥပမာ - `cout << "\"Run, Spot, run, \" she said.\";` ဆိုတဲ့ ကုဒ်ကို ဘာသာပြန်လိုက်မယ်ဆိုရင် `"Run, Spot, run," she said.` ဆိုတဲ့ စာသားကို ရရှိမှာ ဖြစ်ပါတယ်။ တခါတလေ ASCII code 127 ထက် ပိုမြင့်တဲ့ keyboard ပေါ်မှာ မပါဝင်တဲ့ `graphics characters` တွေကို ဖော်ပြချင်တဲ့ အခါ `'\xdd'` ဆိုတဲ့ ပုံစံမျိုးနဲ့ ရေးသားပေးရပါတယ်။ အဲဒီပုံစံမှာပါတဲ့ `dd` နေရာမှာ ကိုယ်ရေးချင်တဲ့ character ရဲ့ သက်ဆိုင်ရာ `hexadecimal` တန်ဖိုးကို ရေးသားပေးရမှာ ဖြစ်ပါတယ်။ ဥပမာ ကိုယ်က လေးထောင့်တုံးပုံလေးကို ဖော်ပြချင်တယ်ဆိုရင် 178 နဲ့ ညီမျှတဲ့ hexadecimal တန်ဖိုး `B2` ကို သုံးပြီး `'\xB2'` လို့ ရေးသားရမှာ ဖြစ်ပါတယ်။ `charvars.cpp` ပရိုဂရမ်လေးမှာ ပထမဆုံး `charvar1` ရဲ့ တန်ဖိုး ('A') ကို ရေးသားမှာ ဖြစ်ပြီး `charvar2` ရဲ့ တန်ဖိုး အနေနဲ့ `tab` တစ်ခုစာ နေရာနေရာ `space` ခြားပြီးမှ နောက်ထပ် သတ်မှတ်ပေးလိုက်တဲ့ `charvar1` တန်ဖိုး ('B') ကို ထုတ်ပေးမှာဖြစ်ပါတယ်။ နောက်ဆုံးအနေနဲ့ `newline` ကို ဆင်းပြီး ပရိုဂရမ် ပြီးဆုံးသွားမှာ ဖြစ်ပါတယ်။

Input with cin

ကျွန်တော်တို့ အနေနဲ့ variable တွေ အသုံးပြုပုံကို လေ့လာခဲ့ကြပါတယ်။ အခုအချိန်မှာ ပရိုဂရမ် ထဲမှာ input ဘယ်လို တောင်းယူသလဲ ဆိုတာကို နမူနာ ပရိုဂရမ်လေး တစ်ပုဒ် ရေးကြည့်ကြရအောင်။ ဒီပရိုဂရမ်လေးမှာ အသုံးပြုသူကို ဖာရင်ဟိုက် အပူချိန်တန်ဖိုး ထည့်သွင်းခိုင်းပြီး စင်တီဂရိတ်ကို ပြောင်းပေးမှာ ဖြစ်ပါတယ်။ လောလောဆယ် integer variables တွေကို အသုံးပြု ရေးသားထားပါတယ်။

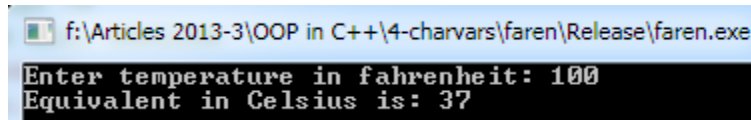
```
//faren.cpp page 72
//demonstrates cin, newline
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    int ftemp; //for temperature in fahrenheit
    cout<<"Enter temperature in fahrenheit: ";
    cin>>ftemp;
    int ctemp=(ftemp-32)*5/9; //convert to celsius
    cout<<"Equivalent in Celsius is: "<<ctemp<<'\n';
    _getch();
    return 0;
}
////////////////////////////////////
```

ဒီဥပမာလေးမှာ အသုံးပြုမယ့်သူကို ဖာရင်ဟိုက်တန်ဖိုး ထည့်ပေးဖို့ cout ကို အသုံးပြုပြီး အသိပေးပါမယ်။ နောက်တစ်ကြောင်း cin>>ftemp; ကတော့ ကီးဘုတ်ကနေ ထည့်ပေးမယ့် တန်ဖိုးကို စောင့်ပြီး ထည့်ပေးလိုက်တာနဲ့ ftemp ထဲကို ထည့်သွင်း သိမ်းဆည်းပေးမှာ ဖြစ်ပါတယ်။



cin ဆိုတဲ့ keyword ကတော့ standard input stream ကို ကိုင်တွယ်မဲ့ object တစ်ခုပဲ ဖြစ်ပါတယ်။ အဲဒီ stream ဟာ redirected မလုပ်ထားဘူးဆိုရင် keyboard ကရိုက်ထည့်လိုက်မယ့် အချက်အလက်တွေကို ကိုယ်စားပြုမှာပါ။ >> operator ကတော့ extraction ဒါမှမဟုတ် get from

operator လို့ ခေါ်ပါတယ်။ ၎င်း operator ဟာ ဘယ်ဘက်မှာရှိတဲ့ stream object (ဒီနေရာမှာ cin) ကို ညာဘက်မှာရှိတဲ့ variable ထဲကို ထည့်ပေးမှာ ဖြစ်ပါတယ်။ ပရိုဂရမ်ရဲ့ result ကို အောက်မှာ လေ့လာကြည့်နိုင်ပါတယ် ခင်ဗျာ။



```
f:\Articles 2013-3\OOP in C++\4-charvars\fare\Release\fare.exe
Enter temperature in fahrenheit: 100
Equivalent in Celsius is: 37
```

Cascading <<

`cout` မှာ `<<` operator ကို အောက်ပါအတိုင်း အကြိမ်ကြိမ် ထပ်ပြီး သုံးပြုထားပါတယ်။

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

ပထမဆုံး "Equivalent in Celsius is: " ဆိုတဲ့ စာကြောင်းကို `cout` ကို ပေးပို့ပါတယ်။ နောက်ထပ်ပြီး `ctemp` ရဲ့ တန်ဖိုးကို ပေးပို့သလို newline character '\n' ကိုလည်း ထည့်ပေးတာ ဖြစ်ပါတယ်။ အဲဒီ output သုံးခုကို တစ်ကြောင်းချင်းစီခွဲပြီး `cout` သုံးကြောင်းနဲ့ ရေးရင်လည်း အတူတူပါပဲ။ အဲဒီလို နည်းလမ်းမျိုးကို `cin` နဲ့ extraction operator `>>` မှာလဲ သုံးလို့ ရပါတယ်။ တကယ်လို့ ထည့်သွင်းပေးရမယ့် တန်ဖိုးက သုံးခု၊ လေးခု ဖြစ်နေရင် တစ်ဆက်တည်း ထည့်ပေးလို့ ရနိုင်ပါတယ်။ ဒါပေမယ့် အသုံးပြုမယ့်သူကို အမှားနည်းစေဖို့အတွက် `cout` နဲ့ တန်ဖိုး တစ်ခုချင်း တောင်းယူပြီး `cin` နဲ့ တစ်ကြောင်းချင်း ဖတ်ယူတာမျိုးက လက်တွေ့မှာ အသုံးများပါတယ်။

Expressions

အတွက်အချက် ပြုလုပ်ဖို့အတွက် variables တွေ၊ constants တွေနဲ့ operators တွေ စီစဉ်ရေးသားထားတာကို `expression` လို့ ခေါ်ပါတယ်။ `a+12` နဲ့ `(a-37)*b/2` စတာတွေဟာ `expression` တွေပါပဲ။ အဲဒီ `expression` တွေကို run လိုက်တဲ့ အခါတိုင်းမှာ တန်ဖိုးတစ်ခု ထွက်လာမှာပါ။

`Expression` တစ်ခုရဲ့ အစိတ်အပိုင်းတွေဟာလည်း `expression` တွေ ဖြစ်နိုင်ပါတယ်။ ဒုတိယ နမူနာ စာကြောင်းမှာ `a-37` နဲ့ `b/2` နှစ်ခု စလုံးဟာ `expression` တွေပဲ ဖြစ်ပါတယ်။ `a` နဲ့ `37` တို့လို တစ်လုံးတည်း ဖြစ်နေတဲ့ variable နဲ့ constant တွေကိုလည်း `expression` တွေအဖြစ် မှတ်ယူနိုင်ပါတယ်။

Expressions တွေဟာ statements တွေနဲ့ မတူပါဘူး။ statements တွေက compiler ကို တစ်ခုခုလုပ်ဖို့ ညွှန်ကြားပြီး semicolon နဲ့ အဆုံးသတ်လေ့ ရှိပါတယ်။ expression တွေကတော့ တွက်ချက်မှုတစ်ခုကို ဖော်ပြပါတယ်။ statement တစ်ခုမှာ expression တွေ အများကြီး ပါနိုင်ပါတယ်။

Precedence

နမူနာ ပရိုဂရမ်လေးထဲက (ftemp-32)*5/9 ဆိုတဲ့ ကုဒ်လေးကို လေ့လာကြရအောင်။ အဲဒီမှာပါတဲ့ လက်သဲကွင်း () လေးသာ မပါခဲ့ဘူးဆိုရင် 32 ကို 5 နဲ့ အရင်မြှောက်မှာ ဖြစ်ပါတယ်။ ဘာလို့လဲ ဆိုတော့ * က - ထက်ပိုပြီး ဦးစားပေးအဆင့်မြင့်လို့ ဖြစ်ပါတယ်။ လက်သဲကွင်းပါလာတဲ့ အခါမှာတော့ - ကို အရင်ဆုံး လုပ်ဆောင်ပြီး နောက်မှ * ကို ဆောင်ရွက်ပါတယ်။ တကယ်တော့ လက်သဲကွင်းထဲက operation တွေကို အရင်ဆုံး လုပ်ဆောင်မှာ ဖြစ်ပါတယ်။ * နဲ့ / ကတော့ ဦးစားပေးအဆင့် တူညီပါတယ်။ ဒါဆို ဘယ်ဘက်မှာရှိတဲ့ operator ကို အရင် လုပ်ဆောင်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် * ပြီးမှ / လုပ်မှာ ဖြစ်ပါတယ်။ ဒီဦးစားပေး အဆင့်အတန်း သတ်မှတ်ပုံဟာ သင်္ချာ ဘာသာရပ်နဲ့ အခြား ပရိုဂရမ်မင်းတွေမှာလည်း အတူတူပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် C++ မှာ ဦးစားပေး အဆင့် သတ်မှတ်ချက်ဟာ အရေးကြီးတဲ့ အကြောင်းအရာ တစ်ခု ဖြစ်ပြီး operators တွေ အကြောင်း ရှင်းပြတဲ့ အခန်းရောက်ရင် အကျယ် ဆွေးနွေးပေးသွားမှာပါ။

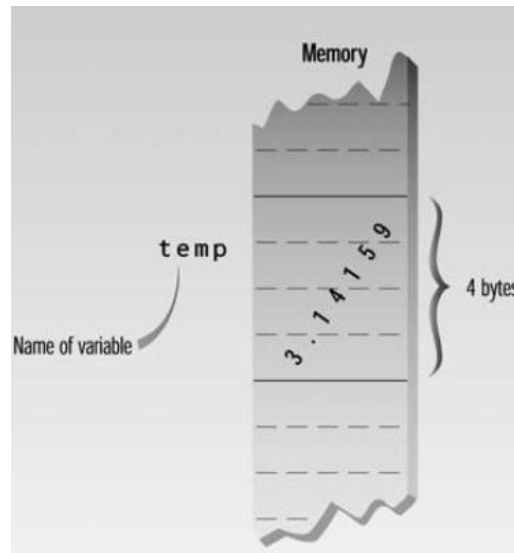
ယခင် သင်ခန်းစာများမှာ int နဲ့ char ဆိုတဲ့ type နှစ်မျိုးအကြောင်းကို ရှင်းပြခဲ့ပါတယ်။ တကယ်တော့ အဲဒီ type နှစ်ခု စလုံးက ဒဿမ မပါတဲ့ integer ဂဏန်းတွေနဲ့ ကိုယ်စားပြု ဖော်ပြခဲ့ကြတာပါ။ ယခု ဆက်လက်ပြီး floating-point variable များအကြောင်းကို ဆက်လက် ဆွေးနွေးသွားမှာ ဖြစ်ပါတယ်။

Floating Point Types

Floating-point variable တွေဟာ ဒဿမ ပါတဲ့ ဂဏန်းတွေကို ကိုယ်စားပြု သိမ်းဆည်းထားနိုင်ပါတယ်။ ဥပမာ - 3.1415927, 0.0000625, 10.2 စတာတွေပဲ ဖြစ်ပါတယ်။ ၎င်းတို့ အားလုံးမှာ decimal point (.) ရဲ့ ဘယ်ဘက်က integer part တွေ ဖြစ်ကြပြီး၊ ညာဘက်ကတော့ fractional part တွေပဲ ဖြစ်ပါတယ်။ floating-point variable တွေဟာ သင်္ချာ ပညာရှင်တွေအနေနဲ့ အကွာအဝေး၊ ဧရိယာ၊ အပူချိန် စတာတွေကို တိုင်းတာဖို့ သုံးတဲ့ real numbers တွေကို ကိုယ်စားပြုပါတယ်။ အဲဒီ တိုင်းတာမှုတွေမှာ များသောအားဖြင့် အပိုင်းကိန်း အစိတ်အပိုင်းတွေ

ပါနေလေ့ ရှိပါတယ်။ C++ မှာတော့ float, double နဲ့ long double ဆိုပြီး floating-point variable သုံးမျိုး ရှိပါတယ်။

Type float



ပုံ-၅.၁ ကွန်ပျူတာ မှတ်ဉာဏ်၌ float type variable တစ်ခု နေရာယူခြင်း

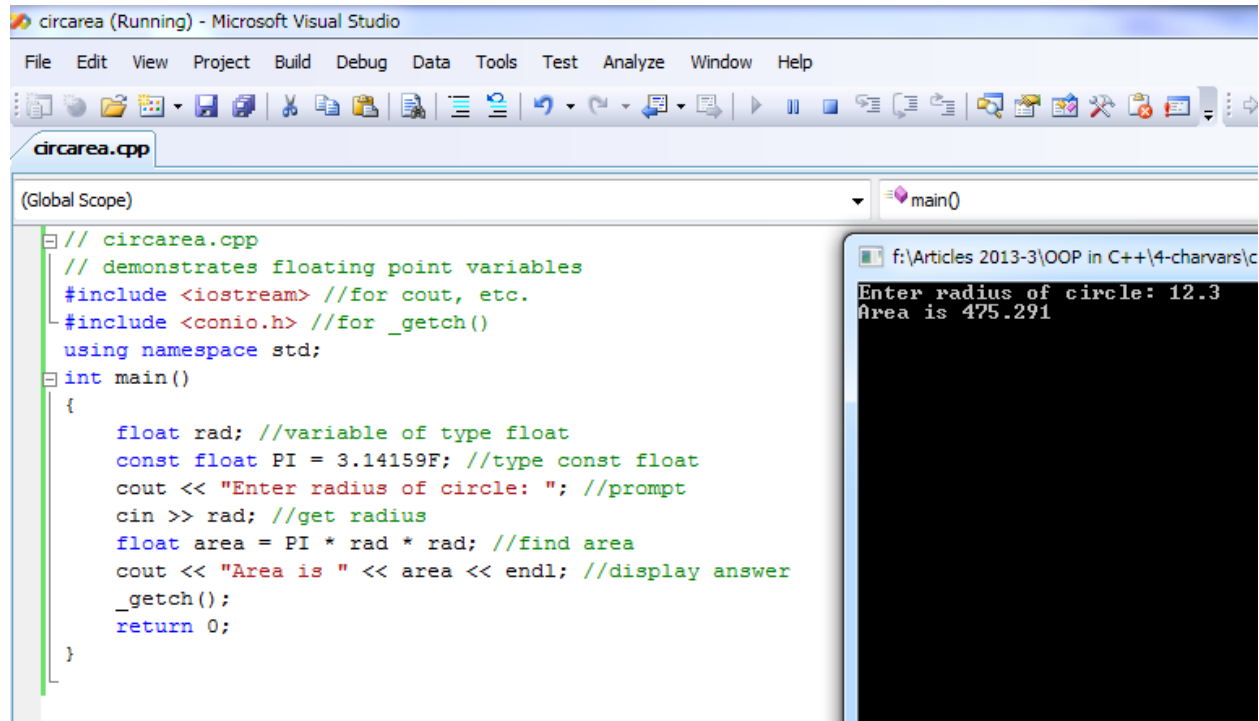
float ကတော့ အဲဒီ floating-point variable သုံးမျိုးထဲမှာ အငယ်ဆုံး ဖြစ်ပြီး 3.4×10^{-38} ကနေ 3.4×10^{38} ကြားထဲမှာ ဂဏန်း ခုနစ်လုံး precision ရှိတဲ့ အချက်အလက်တွေကို သိမ်းဆည်းနိုင်ပါတယ်။ အောက်ပါ ဥပမာလေးကတော့ floating-point number တွေကိုသုံးပြီး စက်ဝိုင်း တစ်ခုရဲ့ ဧရိယာကို ရှာတဲ့ ပရိုဂရမ်လေးပါ။

```
// 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;
}

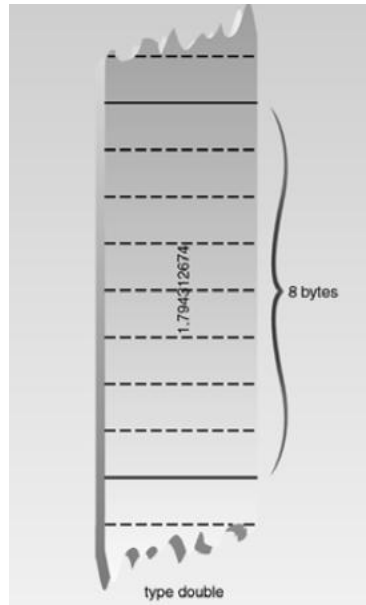
```



ပုံ-၅.၂ circarea.cpp နှင့် testing

Type double and long double

ပိုပြီး အရွယ်အစား ကြီးမားတဲ့ floating point types တွေဖြစ်တဲ့ double နဲ့ long double တွေဟာ ကွန်ပျူတာ မှတ်ဉာဏ်မှာ နေရာပိုယူပြီး တန်ဖိုးတွေ ပိုထည့်သွင်းနိုင်သလို ပိုမိုတိကျလာတာက လွဲလို့ float နဲ့ အတူတူပါ။ double type အနေနဲ့ သိမ်းဆည်းဖို့ 8 bytes လိုအပ်ပြီး 1.7×10^{-308} ကနေ 1.7×10^{308} ကြား ကိန်းဂဏန်းတွေကို သိမ်းဆည်းပေးနိုင်ပြီး ဒဿမ ၁၅ နေရာအထိ တိကျပါတယ်။ long double type ကတော့ ကွန်ပိုင်းလာ အပေါ် မူတည်နေပေမယ့် များသောအားဖြင့် double နဲ့ တူလေ့ ရှိပါတယ်။



ပုံ-၅.၃ ကွန်ပျူတာ မှတ်ဉာဏ်၌ double type variable တစ်ခု နေရာယူခြင်း

Floating-Point Constants

circarea.cpp ပရိုဂရမ်လေးမှာ အသုံးပြုခဲ့တဲ့ 3.14159F ဆိုတဲ့ ဂဏန်းလေးဟာ floating-point constant လေးပဲ ဖြစ်ပါတယ်။ decimal point ကို မြင်လိုက်တာနဲ့ integer type မဟုတ်မှန်း သိနိုင်သလို နောက်ဆုံးက F ကတော့ float type ဖြစ်ကြောင်း သတ်မှတ်ပေးလိုက်တာပါ။ double အတွက်ကတော့ နောက်ဆုံးမှာ ဘာ suffix letter မှ ထည့်ပေးလိုက်စရာ မလိုဘဲ သာမန် ဒဿမကိန်း ပုံစံ ရေးပေးဖို့ပဲ ဖြစ်ပါတယ်။ double က default type ဖြစ်နေလို့ပါပဲ။ long double အတွက်ကတော့ L ကို ထည့်ပေးဖို့ လိုမှာ ဖြစ်ပါတယ်။

ကျွန်တော်တို့ အနေနဲ့ floating-point constants တွေကို exponential notation ကိုသုံးပြီး ရေးသားနိုင်ပါတယ်။ ဒီနည်းနဲ့ သုညတွေ အများကြီး သုံးစရာ မလိုတော့ပါဘူး။ ဥပမာ 1,000,000,000 ကို 1.0E9 လို့ ဖော်ပြနိုင်ပါတယ်။ 1234.56 ကိုလည်း 1.23456E3 လို့ ရေးနိုင်ပါတယ်။ ဒီတော့ 10^9 ကို E9 နဲ့ 10^3 ကို E3 လို့ အလွယ်တကူ ရေးနိုင်ပါတယ်။ E နောက်မှာ ရေးတဲ့ ဂဏန်းကို ထပ်ညွှန်း exponent လို့ ခေါ်ကြပါတယ်။ exponent ကို ကြည့်ပြီး ဒဿမနောက် ဘယ်နေရာ ရွှေ့ပေးရမယ်ဆိုတာကို သိနိုင်ပါတယ်။ exponent တွေက positive နဲ့ negative တန်ဖိုး နှစ်ခုစလုံး ဖြစ်နိုင်ပါတယ်။ 6.35239E-5 ဆိုတာဟာ 0.0000635239 ကို ဖော်ပြတာပါ။

The const Qualifier

circarea.cpp ပရိုဂရမ်ထဲမှာ float အပြင် const ဆိုတဲ့ စကားလုံး ပါလာပါတယ်။

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

Constant အတွက်သုံးတဲ့ const ဆိုတဲ့ သော့ချက် စကားလုံးကို variable တွေကြောင့် data type တွေရဲ့ ရှေ့မှာ ရေးရပါတယ်။ အဲဒီ variable ရဲ့ တန်ဖိုးဟာ ပရိုဂရမ် တစ်ခုလုံးမှာ ပြောင်းလဲလို့ မရတော့ပါဘူး။ တကယ်လို့ အဲဒီ တန်ဖိုးကို ပရိုဂရမ်ထဲမှာ ပြောင်းလဲဖို့ ကြိုးစားခဲ့မယ် ဆိုရင် error message ရရှိမှာ ဖြစ်ပါတယ်။

တကယ်တော့ const qualifier လေးဟာ ကျွန်တော်တို့ ပုံသေ သတ်မှတ်ထားချင်တဲ့ တန်ဖိုးတစ်ခုကို မတော်တဆ ပြောင်းလဲမိခြင်းမှ ကာကွယ်ဖို့ သုံးရတာ ဖြစ်ပါတယ်။ ကုဒ် ကို ဖတ်မိတဲ့ လူတိုင်းကိုလဲ ဒီတန်ဖိုးဟာ ပြောင်းလဲလို့ မရဘူးဆိုတာ သတိပေးထားတာပဲ ဖြစ်ပါတယ်။

The #define Directive

C++ မှာ သိပ်မသုံးကြတော့တဲ့ ပုံစံ တစ်ခု ဖြစ်ပါတယ်။ constants တွေကို #define ဆိုတဲ့ preprocessor directive နဲ့ သတ်မှတ်ပေးနိုင်ပါတယ်။ ဒီနည်းနဲ့ identifier တွေနဲ့ ညီမျှ text phrase တွေကို ဖန်တီးပေးမှာပါ။ ဥပမာ - #define PI 3.14159 ဆိုတဲ့ စာကြောင်းကို ပရိုဂရမ်ရဲ့ အစမှာ ရေးသားခြင်းဖြင့် PI ဆိုတဲ့ စာသားကို 3.14159 ဖြင့် အစားထိုး ပေးမှာ ဖြစ်ပါတယ်။ ဒီလို ရေးသားနည်းကို C language မှာ အတော်လေး ခေတ်စားခဲ့ဘူးပါတယ်။ ဒါပေမယ့် ဘယ် data type ဆိုတာကို မသတ်မှတ်နိုင်တဲ့ အားနည်းချက်ကြောင့် အမှားအယွင်းတွေ ဖြစ်လာနိုင်ပါတယ်။ ဒါကြောင့် C++ နဲ့ C# မှာ const ကို သုံးပြီး ရေးသားကြတာပါ။ ဒါပေမယ့် ပရိုဂရမ် အဟောင်းတွေကို ဖတ်ကြည့်မယ် ဆိုရင်တော့ #define တွေ အများကြီး တွေ့ရနိုင်ပါတယ်။

Type bool

int data type မှာ ဖြစ်နိုင်ခြေရှိတဲ့ တန်ဖိုးတွေ ဘီလီယံနဲ့ ချီပြီး ရှိနိုင်ပါတယ်။ char မှာတော့ ၂၅၆ လုံး ဖြစ်နိုင်ပါတယ်။ bool ကတော့ true နဲ့ false ဆိုပြီး နှစ်မျိုးပဲ ဖြစ်နိုင်ပါတယ်။ သီအိုရီအရ bool ဟာ one bit ပဲ နေရာယူပါတယ်။ ဒါပေမယ့် တကယ်လက်တွေ့မှာ bit အနေနဲ့ သိမ်းခဲ့ရင် byte ကနေ ပြန်ပြီး ခွဲထုတ်ယူရတာ ဖြစ်လို့ အချိန်ပိုကြာတာကြောင့် ကွန်ပိုင်းလာက ကိုင်တွယ်ရယူဖို့ မြန်ဆန်တဲ့ byte အနေနဲ့ပဲ သိမ်းဆည်းတာ ဖြစ်ပါတယ်။ နှိုင်းယှဉ် စစ်ဆေးချက်တွေရဲ့ ရလဒ်တွေကို သိမ်းဆည်းဖို့

bool type ကို အသုံးများပါတယ်။ ဥပမာ ကိုကို ဟာ မောင်မောင်ထက် အသက်ကြီးလား ဆိုတဲ့ နှိုင်းယှဉ်ချက် မှာ မှန်ခဲ့ရင် bool value - true ကို ရရှိမှာ ဖြစ်ပြီး မှားခဲ့ပါက false ကို ရရှိမှာ ဖြစ်ပါတယ်။ bool ဆိုတဲ့ နာမည်ကို true-or-false တန်ဖိုးတွေ နဲ့ logical operators တွေကို သုံးပြီး စဉ်းစားတွေးခေါ်နည်းကို တီထွင်ခဲ့တဲ့ ၁၉ ရာစု အင်္ဂလိပ် သင်္ချာပညာရှင် George Boole အား အစွဲပြုပြီး ဂုဏ်ပြုခေါ်ဆိုကြတာပါ။

The setw Manipulator

ကျွန်တော်တို့ အနေနဲ့ endl အကြောင်းကို ရေးသားစဉ်က အချက်အလက်တွေကို screen မှာ ပြသတဲ့အခါ insertion operator (<<) နဲ့ တွဲသုံးတဲ့ manipulator ဖြစ်ကြောင်း ရှင်းပြခဲ့ဖူးပါတယ်။ အခု သင်ခန်းစာမှာတော့ output ရဲ့ field width ကို ပြောင်းလဲ ပေးနိုင်တဲ့ setw manipulator အကြောင်းပဲ ဖြစ်ပါတယ်။

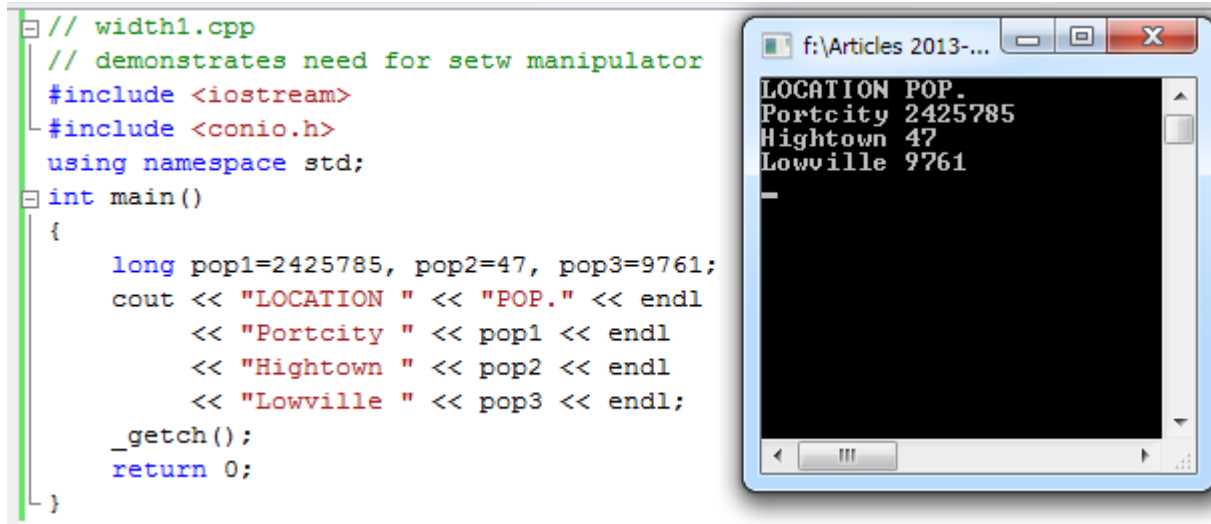
ကျွန်တော်တို့ cout ကို သုံးပြီး တန်ဖိုး တစ်ခုကို screen မှာ ပြသမယ်ဆိုရင် အဲဒီတန်ဖိုးဟာ တိကျတဲ့ width ရှိတဲ့ မမြင်ရတဲ့ လေးထောင့် နေရာတစ်ခုကို ယူထားတယ်လို့ စိတ်ကူး ကြည့်နိုင်ပါတယ်။ default field ကတော့ အဲဒီ တန်ဖိုး ဆန့်သလောက် နေရာယူထားတာပါ။ ဥပမာ- integer 567 ဟာ အကွာရာ သုံးလုံးစာ နေရာယူမှာ ဖြစ်ပြီး "pajamas" ဆိုတဲ့ string ကတော့ ခုနစ်နေရာ ယူမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် အချို့ နေရာတွေမှာတော့ ကျွန်တော်တို့ စိတ်ကြိုက် ပြင်ဆင် သတ်မှတ်ပေးဖို့ လိုလာပါတယ်။ နမူနာ အနေနဲ့ width1.cpp ပရိုဂရမ်လေးကို လေ့လာကြည့်ရအောင်။ ဒီပရိုဂရမ်လေးမှာ မြို့သုံးခုရဲ့ အမည်ကို ကော်လံ တစ်ခုမှာ ပြသထားပြီး အခြားကော်လံမှာတော့ သက်ဆိုင်ရာ လူဦးရေများကို ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။

```
// 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;
}

```



```

// 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;
}

```

Output window content:

```

LOCATION POP.
Portcity 2425785
Hightown 47
Lowville 9761

```

ပုံ-၅.၄ width1.cpp နှင့် testing

ကျွန်တော်တို့ သတိထားမိတဲ့အတိုင်း output မှာပါတဲ့ လူဦးရေ အရေအတွက် တွေဟာ နှိုင်းယှဉ်ရ ခက်ခဲနေပါတယ်။ တကယ်လို့ ညာဘက်ကို ညှိထားမယ် ဆိုရင် ဖတ်ရ ပိုမို လွယ်ကူလာမှာ ဖြစ်ပါတယ်။ မြို့နာမည်တွေ ဖော်ပြတဲ့အခါမှာလည်း နောက်က ဂဏန်းတွေကြားမှာ နေရာလွတ် (spaces) တွေ ခြားပေးနိုင်မယ်ဆိုရင် ပိုပြီး အဆင်ပြေလာမှာပါ။ အထက်က width1.cpp ကိုပဲ အနည်းငယ် ပြင်ဆင်ပြီး with2.cpp နာမည်နဲ့ setw manipulator အသုံးပြုကာ ပြန်ရေးပါမယ်။

```

// 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

```

```

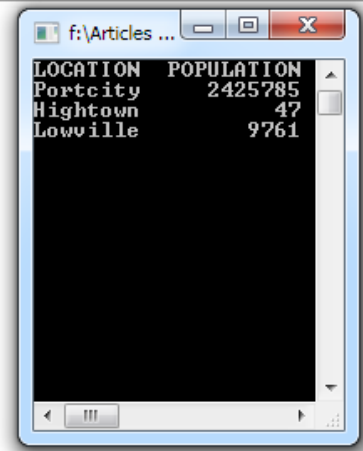
    << setw(8) << "Lowville" << setw(12) << pop3 << endl;
    _getch();
    return 0;
}

```

```

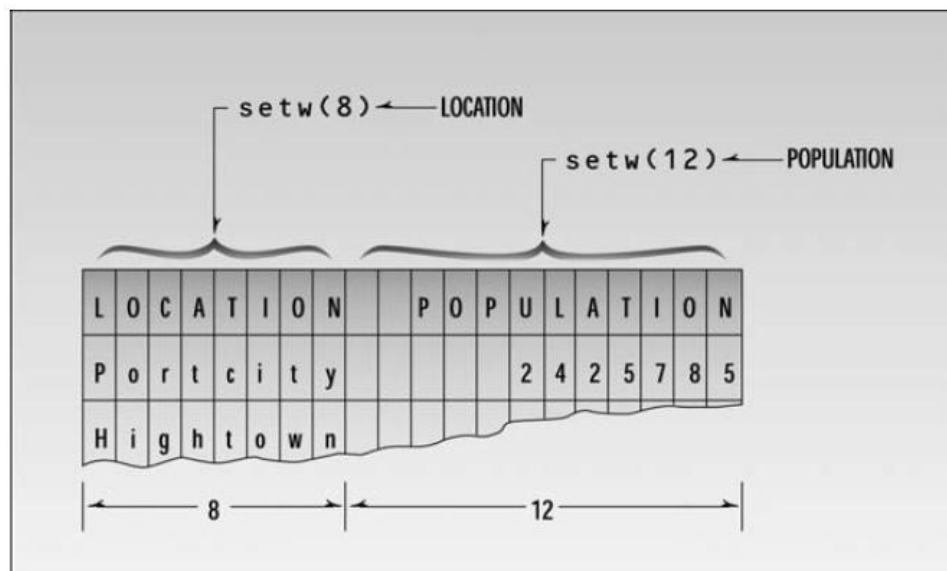
// width2.cpp
// demonstrates setw manipulator
#include <iostream>
#include <iomanip> // for setw
#include <conio.h>
using namespace std;
int main()
{
    long pop1=2425785, pop2=47, pop3=9761;
    cout << setw(8) << "LOCATION" << setw(12)
        << "POPULATION" << endl
        << setw(8) << "Portcity" << setw(12) << pop1 << endl
        << setw(8) << "Hightown" << setw(12) << pop2 << endl
        << setw(8) << "Lowville" << setw(12) << pop3 << endl;
    _getch();
    return 0;
}

```



ပုံ-၅.၅ width2.cpp နှင့် testing

setw manipulator က သူ့နောက်က ဂဏန်းတွေနဲ့ စာသားတွေကို n character အကျယ်ရှိတဲ့ field အတွင်းမှာ ဖော်ပြပေးပါတယ်။ အဲဒီနေရာမှာ n ဆိုတာက setw(n) function ကွင်းထဲက argument (n) ပဲ ဖြစ်ပါတယ်။ အဲဒီ တန်ဖိုးက field ထဲမှာ right-justified ဖြစ်နေမှာပါ။ အထက်က ပုံ-၅.၅ မှာ output မှာ မြင်ရမယ့် ပုံကို ပြထားပါတယ်။ ထူးခြားချက်အနေနဲ့ လူဦးရေတွေကို ဖော်ပြတဲ့ နေရာမှာ 2-byte နေရာယူပြီး အများဆုံး တန်ဖိုး 32767 ပဲ သိမ်းလို့ရတဲ့ အတွက် overflow ဖြစ်နိုင်ခြေရှိတဲ့ integer type တွေကို မသုံးဘဲ long ကို သုံးထားတာပါ။



ပုံ-၅.၆ width2.cpp နှင့် testing

Cascading the Insertion Operator

width1.cpp နဲ့ width2.cpp မှာ ပါရှိတဲ့ cout statement ထဲမှာ cout ကို တစ်ခါပဲ သုံးထားပြီး စာကြောင်းရေ အတော်များများ ရေးထားတာကို သတိထားမိမှာပါ။ ကွန်ပိုင်းလာ က whitespace တွေကို ထည့်မစဉ်းစားတာနဲ့ insertion operator (<<) တွေကို cascade လုပ်ပြီး ထပ်ခါ ထပ်ခါ ရေးလို့ရတဲ့ အချက်တွေကို အသုံးပြုထားတာ ဖြစ်တယ်။ cout လေးကြောင်းရေးပြီး ရလာမယ့် ရလာဒ်နဲ့ အတူတူပဲ ဖြစ်ပါတယ်။

Multiple Definitions

Variables တွေဖြစ်တဲ့ pop1, pop2 နဲ့ pop3 တို့ကို ဖန်တီးစဉ် တန်ဖိုးတွေပါ တစ်ပါတည်း သတ်မှတ်ပေးထားတာကို တွေ့ရပါမယ်။ ဒီနည်းနဲ့ data type တူညီတဲ့ variables တွေကို ကော်မာ ခံပြီး ဖန်တီးပေးခြင်း၊ တန်ဖိုး သတ်မှတ်ပေးခြင်းဖြင့် ကုဒ်များကို လျော့ချနိုင်ပါတယ်။

The IOMANIP Header File

endl, setw() စတဲ့ manipulators တွေကို အသုံးပြုဖို့ဆိုရင် ကျွန်တော်တို့ iostream header file ထဲမှာ မပါတဲ့ အတွက် iomanip header file ကို ထပ်မံ ကြေငြာပေးရမှာ ဖြစ်ပါတယ်။ width2.cpp မှာ `#include <iomanip> // for setw` ဆိုပြီး ရေးသားသွားတာပါ။

Data types Conversion

ယခင် သင်ခန်းစာများမှာ data types များ အသုံးပြုပုံကို အသေးစိတ် ရှင်းပြခဲ့ပါတယ်။ ယခု သင်ခန်းစာမှာတော့ အဲဒီ data types များကို တစ်ခုကနေ အခြားတစ်ခုသို့ ပြောင်းလဲနည်းများ အကြောင်း ဆွေးနွေး တင်ပြသွားမှာ ဖြစ်ပါတယ်။

C language မှာလိုပဲ C++ မှာ မတူညီတဲ့ data type တွေ တွဲဘက် အသုံးပြုခွင့် ပေးထားပါတယ်။ ဥပမာအနေနဲ့ အောက်ပါ mixed.cpp ပရိုဂရမ်လေးကို လေ့လာကြည့်ရအောင်။

```
// 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 ဥပမာမှာ double data type (totalWeight) ရရှိဖို့အတွက် int data type (count) ကို float data type (avgWeight) နဲ့ မြှောက်ထားတာ တွေ့ရမှာ ဖြစ်ပါတယ်။ ကျွန်တော်တို့အနေနဲ့ RUN ကြည့်တဲ့ အခါမှာ Error မပြဘဲ အလုပ်လုပ်တာကို တွေ့ရပါတယ်။ compiler အနေနဲ့ အဲဒီလို မတူညီတဲ့ data types တွေ အချင်းချင်း ပေါင်း၊ နုတ်၊ မြှောက်၊ စား အစရှိတဲ့ arithmetic operation ပြုလုပ်တာတွေကို လက်ခံပါတယ်။

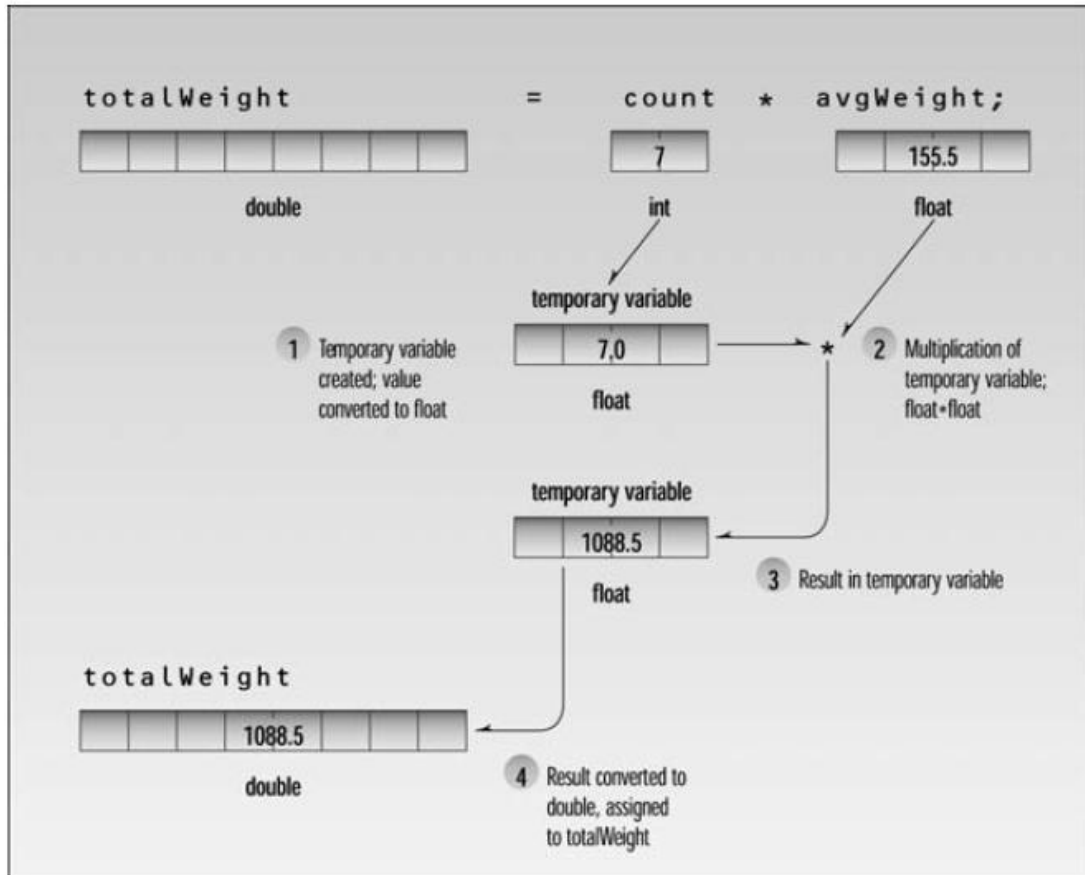
အချို့ပရိုဂရမ်းမင်း ဘာသာစကားတွေမှာ ရေးသားရတာ ဒီလောက် မလွတ်လပ်ပါဘူး။ အခုလို မတူညီတဲ့ data type တွေ ရောရေးတဲ့အခါမျိုးမှာ error ပြမှာ ဖြစ်ပါတယ်။ အဲဒီ ဘာသာစကားတွေရဲ့ compiler တွေက ကျွန်တော်တို့အနေနဲ့ မှားယွင်းရေးသားမိတယ် လို့ ယူဆပြီး အဲဒီအမှားကနေ ကယ်တင်ဖို့ ကြိုးစားကြတာပဲ ဖြစ်ပါတယ်။ C နဲ့ C++ မှာတော့ ကျွန်တော်တို့ အဲဒီလို ရောနှောရေးသားမှုကို ရည်ရွယ်ချက် တစ်ခုကြောင့်လို့ ယူဆနားလည်ပေးပြီး ကူညီ လုပ်ဆောင်ပေးမှာ ဖြစ်ပါတယ်။ အဲဒီ အချက်ဟာ C နဲ့ C++ ခေတ်စားတဲ့ အကြောင်းတစ်ခုလဲပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် ပိုပြီးလွတ်လပ်တဲ့ ဘာသာစကား ရေးသားနည်း တစ်ခုဟာ ပိုပြီး အမှား လုပ်မိစေတတ် ပါတယ်။

Automatic Conversion

compiler အနေနဲ့ အဲဒီလို ရောနှော ရေးသားထားတဲ့ expressions တွေနဲ့ ရင်ဆိုင်ရတဲ့ အခါမှာ အလုပ်လုပ်သွားပုံကို လေ့လာကြည့်ရအောင်။ အောက်မှာ ဖော်ပြထားတဲ့ Table အရ data types တွေကို higher နဲ့ lower ဆိုပြီး အကြမ်းဖျင်း ခွဲခြားထားနိုင်ပါတယ်။

Data Type	Order
long double	Highest
double	
float	
long	
int	
short	
char	Lowest

အပေါင်းနဲ့ အမြှောက် တို့လို arithmetic operators တွေဟာ တူညီတဲ့ data type ရှိတဲ့ operands တွေကိုပဲ လုပ်ဆောင်ပေးချင်ပါတယ်။ ဒါကြောင့် ရောနှောနေတဲ့ data types တွေနဲ့ ရင်ဆိုင်ရတဲ့ အခါမှာ data type အနိမ့်ကို အမြင့်သို့ အလိုအလျောက် ပြောင်းလဲပေးမှာ ဖြစ်ပါတယ်။ mixed.cpp မှာ int (count) float ကိုပြောင်းလဲပေးပြီး float (avgWeight) နဲ့ မမြှောက်ခင် tempoerary variavle တစ်ခုမှာ သိမ်းဆည်းထားပေးပါတယ်။ float နှစ်ခု မြှောက်လို့ ရလာတဲ့ ရလဒ်ကို double ပြောင်းပြီး totalWeight ထဲကို ထည့်ပေးပါတယ်။ အဲဒီ ဖြစ်စဉ်ကို အောက်မှာ ပုံနဲ့ ပြထားပါတယ်။



အဲဒီလို data types တွေ အလိုအလျောက် ပြောင်းလဲပေးသွားတာဟာ မမြင်သာပါဘူး။ ကျွန်တော်တို့ အနေနဲ့ ဖြစ်ချင်တာတွေကို C++ compiler က အလိုအလျောက် လုပ်ပေးသွားတာ ဖြစ်တဲ့အတွက် အများကြီး စဉ်းစားနေစရာလဲ မလိုအပ်ပါဘူး။ ဒါပေမယ့် တစ်ခုတစ်ရံမှာ အဲဒီလို ပြောင်းလဲတာအချို့ကို compiler အနေနဲ့ အဆင်ပြေပြေ မပြုလုပ်ပေးနိုင်ပါဘူး။ နောက်တစ်ခုက ကျွန်တော်တို့အနေနဲ့ objects တွေ ပြုလုပ်ခြင်းဖြင့် ကိုယ်ပိုင် data types တွေဖန်တီးတဲ့ အချိန်မှာ မတူညီတဲ့ object data type တွေကို ရောနှော အသုံးပြုဖို့အတွက် compiler က ပြုလုပ်မပေးနိုင်ပါဘူး။ အဲဒီအခါမှာတော့ ကိုယ်ပိုင် conversion routine တွေကို ရေးသား အသုံးပြုဖို့ လိုလာမှာ ဖြစ်ပါတယ်။ compiler ဟာ built-in data types တွေကိုပဲ အလိုအလျောက် ပြောင်းလဲ ပေးနိုင်စွမ်း ရှိပါတယ်။

Casts

တစ်ခါ တစ်ရံမှာ compiler အနေနဲ့ အလိုအလျောက် type conversion ပြုလုပ်မပေးနိုင်တဲ့ အခြေအနေမျိုးမှာ data type တစ်ခုကို ကျွန်တော်တို့ သတ်မှတ်ပေးထားတဲ့ အခြား data type တစ်ခုသို့ ပြောင်းလဲပေးခြင်းအား type casts သို့မဟုတ် Casts လုပ်တယ်လို့ ခေါ်ဆိုနိုင်ပါတယ်။

C++ မှာ static casts, dynamic casts, reinterpret casts နဲ့ const casts ဆိုပြီး casts အမျိုးအစား အများအပြားရှိပါတယ်။ ဒီသင်ခန်းစမှာတော့ static casts အကြောင်းကိုပဲ လေ့လာသွားမှာ ဖြစ်ပြီး သီးခြား အခြေအနေများမှာ အသုံးပြုရတဲ့ ကျန်ရှိနေတဲ့ casts များကိုတော့ သက်ဆိုင်ရာ သင်ခန်းစများမှာ အလျင်းသင့်သလို ထည့်သွင်း ရှင်းပြပေးသွားမှာ ဖြစ်ပါတယ်။

အောက်ပါ နမူနာသည် int data type အား char data type အဖြစ် ပြောင်းလဲပေးသော ကုဒ် ဖြစ်သည်။

```
aCharVar = static_cast<char>(anIntVar);
```

ကျွန်တော် တို့ type ပြောင်းလဲပေးချင်တဲ့ variable (anIntVar) ကို လက်သဲကွင်း () ထဲ ထည့်သွင်းထားရမှာ ဖြစ်ပြီး ပြောင်းလဲမည့် data type (char) ကိုတော့ angle brackets <> ထဲမှာ ထည့်ပေးရမှာ ဖြစ်ပါတယ်။ ဒီနည်းနဲ့ aCharVar ထဲကို တန်ဖိုး မထည့်သွင်းခင်မှာ char data type ကို ပြောင်းလဲ ပေးနိုင်ပါတယ်။

Standard C++ မတိုင်မီ C ခေတ်တုန်းက ကျွန်တော်တို့အနေနဲ့ အခြား နည်းလမ်းဟောင်းများကို cast လုပ်ဖို့ သုံးခဲ့ကြဘူးပါတယ်။ `aCharVar = (char) anIntVar;` ဒါမှ မဟုတ် `aCharVar = char(anIntVar);` ဆိုပြီး ရေးခဲ့ကြပါတယ်။ ဒါပေမယ့် အားနည်းချက်တစ်ခုက ကုဒ်တွေအများကြီးထဲမှာ casts လုပ်ထားတာကို ရှာရခက်လောက်အောင် ပျောက်နေတတ်ပါတယ်။ Find နဲ့ ရှာဖို့လည်း မလွယ်ပါဘူး။ အခု static_cast ကတော့ အဲဒီ ပြဿနာကို ဖြေရှင်းပေးနိုင်ပါတယ်။ ဒါကြောင့် ယခင် နည်းလမ်းဟောင်းတွေကို သုံးနိုင်သေးပေမယ့် static_cast နည်းလမ်းကိုပဲ အသုံးပြုကြဖို့ တိုက်တွန်းလိုပါတယ်။

Operators

ယခင် သင်ခန်းစများမှာ data types conversion ပြုလုပ်ပုံများကို ရှင်းပြခဲ့ပါတယ်။ ယခု သင်ခန်းစမှာတော့ Operators များအကြောင်း ဆွေးနွေး တင်ပြသွားမှာပါ။

Arithmetic Operators

C++ မှာ ပုံမှန် arithmetic operator ၄ မျိုးဖြစ်တဲ့ +, -, *, / တို့ကို အသုံးပြုပြီး ပေါင်းခြင်း၊ နုတ်ခြင်း၊ မြှောက်ခြင်း၊ စားခြင်းများကို ပြုလုပ်ပေးပါတယ်။ အဲဒီ arithmetic operators တွေဟာ အခြား

ပရိုဂရမ်းမင်း ဘာသာစကား တွေမှာလိုပဲ algebra သင်္ချာမှာ သုံးတဲ့ပုံစံမျိုးကို data types အမျိုးမျိုးမှာ အသုံးပြုနိုင်သလို အခြား arithmetic operators များလည်း ရှိပါသေးတယ်။

The Remainder Operator

Remainder Operator (%) ဟာ integer အမျိုးအစား variable (char, short, int, long) တွေအတွက်ပဲ အသုံးပြုလို့ ရပါတယ်။ တစ်ခုတည်းမှာ modulus operator လို့လည်း ခေါ်တတ်ကြပါတယ်။ ၎င်းဟာ ဂဏန်းတစ်ခုကို အခြား ဂဏန်းတစ်ခုနဲ့ စားလို့ ရလာတဲ့ အကြွင်း တန်ဖိုးကို ရှာပေးပါတယ်။ အောက်မှာ ဖော်ပြထားတဲ့ remaind.cpp ကို လေ့လာကြည့်ရအောင်။

```
// remaind.cpp
// demonstrates remainder operator
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    cout << 6 % 8 << endl // 6
    << 7 % 8 << endl // 7
    << 8 % 8 << endl // 0
    << 9 % 8 << endl // 1
    << 10 % 8 << endl; // 2
    _getch();
    return 0;
}
```

အထက်ပါ ပရိုဂရမ်လေးမှာ ၆ ကနေ ၁၀ အထိ ဂဏန်းတွေကို remainder operator ကို သုံးပြီး ၈ နဲ့စားလို့ရလာတဲ့ အကြွင်းတွေကို ရှာထားပါတယ်။ အဖြေတွေကတော့ ၆၊ ၇၊ ၀၊ ၁ နဲ့ ၂ တို့ ဖြစ်ပါတယ်။ Remainder operator ကို အခြေအနေ အမျိုးမျိုးမှာ သုံးနိုင်ပါတယ်။ နောက်ပိုင်း အလျင်းသင့်သလို ဥပမာများနဲ့ ရှင်းလင်းသွားပါဦးမယ်။ remaind.cpp မှာပါတဲ့ `cout<<6%8` ဆိုတဲ့ ကုဒ်မှာ remainder operator ကို ပထမဦးဆုံး အလုပ် စလုပ်ပါတယ်။ ဘာဖြစ်လို့လဲ ဆိုတော့ % operator က << operator ထက် precedence ပိုမြင့်လို့ပဲ ဖြစ်ပါတယ်။ တကယ်လို့ ပိုသေချာ စေချင်တယ် ဆိုရင်တော့ လက်သဲကွင်း အဖွင့်အပိတ် တစ်စုံကြားမှာ (6%8) ဆိုပြီး ရေးပေးရမှာ ဖြစ်ပါတယ်။

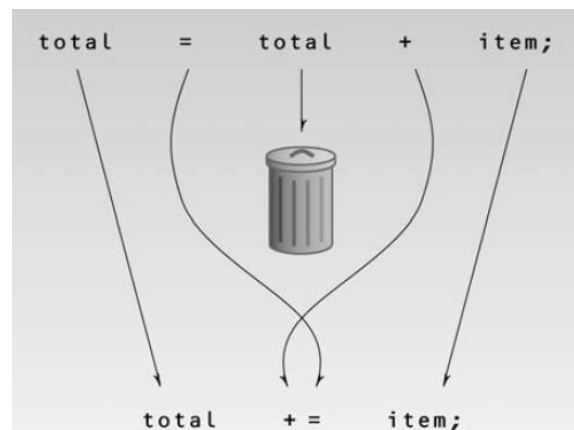
Arithmetic Assignment Operator

C++ ကို လေ့လာတဲ့ အခါမှာ ကုဒ်တွေကို အတိုချုံ့ရေးနိုင်တဲ့ နည်းလမ်းအများအပြား ထည့်သွင်းထားတာကို တွေ့ရှိရပါတယ်။ အဲဒီထဲက တစ်ခုကတော့ arithmetic assignment operator ပဲ ဖြစ်ပါတယ်။ ၎င်းဟာ C++ ကို တမူထူးခြားတဲ့ အသွင် ဖြစ်စေပါတယ်။

ပုံမှန်အားဖြင့် ပရိုဂရမ်းမင်း ဘာသာစကားတွေမှာ total တန်ဖိုးထဲကို item တန်ဖိုးတစ်ခု ထပ်ပေါင်းထည့်ဖို့ အတွက် `total = total + item;` လို့ ရေးလေ့ ရှိကြပါတယ်။ ဒါပေမယ့် အဲဒီ ကုဒ်ထဲမှာ total ကို နှစ်ခါ ထပ်ရေးရတဲ့ အတွက် ကုဒ်ကို တိုအောင် ရေးချင်သူများ ဘဝင်မကျ ဖြစ်ရပါတယ်။ ဒါကြောင့် C++ က ပိုမိုကျစ်လစ်တဲ့ ရေးနည်းကို ပံ့ပိုးပေးထားပါတယ်။ Arithmetic operator ကို assignment operator နဲ့ ပေါင်းစပ်ပြီး နှစ်ခါထပ်နေတဲ့ operand ကို ဖြုတ်ထုတ်လိုက်ပါတယ်။ ဒီနည်းနဲ့ အထက်ပါ ကုဒ်ကို ယခုလို တိုတိုရှင်းရှင်း နဲ့ ပြင်ရေးလို့ ရပါတယ် -

Total += item;

အောက်မှာပြထားတဲ့ ပုံလေးကတော့ အဲဒီ ကုဒ် နှစ်မျိုး ထပ်တူညီပုံကို ဖော်ပြထားတာပဲ ဖြစ်ပါတယ်။



အထက်ပါ ကုဒ်မှာ arithmetic operator + နဲ့ assignment operator = ကို ပေါင်းထားတဲ့ += ကို သုံးပြသွားသလို -=, *=, /= နဲ့ %= စတဲ့ operator များကိုလဲ အသုံးပြုနိုင်ပါတယ်။ အောက်ပါ assign.cpp ပရိုဂရမ်လေးမှာ နမူနာ သုံးပြထားပါတယ်။

```

// assign.cpp
// demonstrates arithmetic assignment operators
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    int ans = 27;
    ans += 10; //same as: ans = ans + 10;
    cout << ans << ", ";
    ans -= 7; //same as: ans = ans - 7;
    cout << ans << ", ";
    ans *= 2; //same as: ans = ans * 2;
    cout << ans << ", ";
    ans /= 3; //same as: ans = ans / 3;
    cout << ans << ", ";
    ans %= 3; //same as: ans = ans % 3;
    cout << ans << endl;
    _getch();
    return 0;
}

```

f:\Articles 2013-3\OOP in C++\4-charvars\eg7-2 assign\Debug\eg7-2 a
37, 30, 60, 20, 2

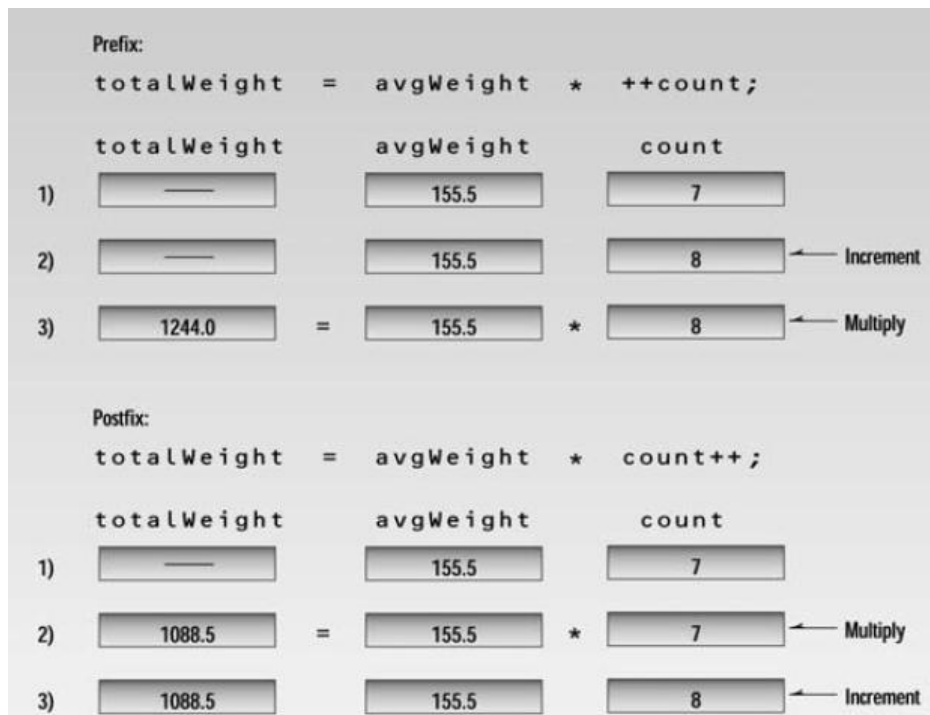
ကျွန်တော်တို့အနေနဲ့ ကုဒ်တွေ ရေးသားတဲ့ နေရာမှာ arithmetic assignment operator တွေကို မဖြစ်မနေ အသုံးပြုဖို့တော့ မလိုပါဘူး။ ဒါပေမယ့် အဲဒီ operator တွေဟာ C++ ရဲ့ common feature တွေ ဖြစ်တဲ့အပြင် ကုဒ်တွေကို တိုရင်း လိုရှင်းဖြစ်အောင် ကူညီပေးတဲ့အတွက် အလျင်းသင့်သလို အသုံးပြုစေလိုပါတယ်။ ကျွန်တော် သင်ခန်းစာတွေမှာလည်း ၎င်းတို့ကို မကြာခဏ ထည့်သွင်း အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။

Increment Operators

ကျွန်တော်တို့ အနေနဲ့ arithmetic assignment operator ကို မသိခင်တုန်းက (count ဆိုတဲ့) variable ထဲကို တန်ဖိုး တစ်ခု တိုးချင်ရင် `count = count + 1;` ဆိုပြီး ရေးခဲ့ကြပါတယ်။ arithmetic assignment operator ကို သုံးတတ်လာတော့ `count += 1;` ဆိုပြီး အတိုချုံ့ ရေးနိုင်လာပါတယ်။ ဒါပေမယ့် C++ မှာ ဒီထက်ပိုပြီး တိုအောင် ချုံ့ရေးလို့ရတဲ့ increment operator (++) ဆိုတာ ရှိပါတယ်။ ဒါကြောင့် အထက်ပါ ကုဒ်တွေကို `++count;` ဆိုပြီး အတိုဆုံးရေးလို့ ရလာပါတယ်။ ရလဒ်ကတော့ တန်ဖိုးကို ၁ တိုးပေးမှာပဲ ဖြစ်ပါတယ်။

Prefix and Postfix

Increment operator ကို အသုံးပြုတဲ့ နေရာမှာ variable ရဲ့ရှေ့ကနေ သုံးတဲ့ prefix ပုံစံနဲ့ နောက်ကနေ တွဲဘက်သုံးတဲ့ postfix ပုံစံဆိုပြီး နှစ်မျိုး သုံးလို့ ရပါတယ်။ အဲဒီ နှစ်ခုရဲ့ခြားနားချက်ကို လေ့လာကြည့်ကြရအောင်။ ကျွန်တော်တို့အနေနဲ့ ကုဒ်တွေထဲမှာ အခြား operation တွေနဲ့ ရောထွေးပြီး variable တစ်ခုကို increment လုပ်ရတဲ့ အခြေအနေမျိုးကို မကြာခဏ ကြုံရတတ်ပါတယ်။ ဥပမာ-
`totalWeight = avgWeight * ++count;` အဲဒီလို အခြေအနေမျိုးမှာ ပြဿနာက count ကို increment အရင်လုပ်မလား၊ အရင် မြှောက်မလား ဆိုတာပါပဲ။ ဒီကုဒ်အရတော့ prefix notation ကို သုံးထားတဲ့ အတွက် increment ကို အရင်လုပ်ပြီးမှ avgWeight နဲ့ မြှောက်မှာ ဖြစ်ပါတယ်။ တကယ်လို့ postfix notation ကိုသုံးပြီး `totalWeight = avgWeight * count++;` လို့ ရေးသားခဲ့မယ်ဆိုရင်တော့ avgWeight နဲ့ အရင်မြှောက်ပြီးမှ increment လုပ်မှာ ဖြစ်ပါတယ်။ ဒီဖြစ်စဉ်ကို အောက်က ပုံမှာ အသေးစိတ် ရှင်းပြထားပါတယ်။



အောက်မှာ ဖော်ပြထားတဲ့ `incrm.cpp` ပရိုဂရမ်လေးမှာတော့ prefix နဲ့ postfix နှစ်ခုလုံးကို သုံးပြုထားတာ တွေ့ရပါလိမ့်မယ်။

```

// increm.cpp
// demonstrates the increment operator
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    int count = 10;
    cout << "count=" << count << endl; //displays 10
    cout << "count=" << ++count << endl; //displays 11 (prefix)
    cout << "count=" << count << endl; //displays 11
    cout << "count=" << count++ << endl; //displays 11 (postfix)
    cout << "count=" << count << endl; //displays 12
    _getch();
    return 0;
}

```

f:\Articles 2013-3\OOP in C++\4-charvars\eg7-3 increm\Debug\eg7-3 increm.exe

```

count=10
count=11
count=11
count=11
count=12

```

ဒီပရိုဂရမ်လေးမှာ ပထမဆုံး count ရဲ့တန်ဖိုး 10 ကို cout သုံးပြီး ထုတ်ပြပါတယ်။ ဒုတိယ စာကြောင်းမှာ prefix ကိုသုံးပြီး count ကို increment လုပ်ထားတဲ့အတွက် cout မထုတ်ခင်မှာ count တန်ဖိုး 10 ကနေ 11 ဖြစ်သွားပြီး အဖြေက 11 ဖြစ်လာပါတယ်။ နောက်တစ်ကြောင်းမှာ postfix ကို သုံးပြီး increment လုပ်တာ ဖြစ်တဲ့အတွက် cout ထုတ်တဲ့ အချိန်မှာ increment မလုပ်ရသေးပါဘူး။ ဒါကြောင့် 11 ကိုပဲ ထုတ်ပေးတာပါ။ ဒါပေမယ့် cout ထုတ်ပြီးပြီးချင်း increment လုပ်ပေးမှာ ဖြစ်တဲ့အတွက် နောက်တစ်ကြောင်းနဲ့ cout အထုတ်မှာ တန်ဖိုးက 12 ဖြစ်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

The Decrement (--) Operator

Decrement operator (--) လုပ်ဆောင်ပုံတွေက increment operator လိုပဲ ဖြစ်ပါတယ်။ တစ်ခုကွဲပြားတာက decrement operator အနေနဲ့ variable ရဲ့တန်ဖိုးထဲကနေ 1 နှုတ်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ ၎င်းကိုလဲ prefix နဲ့ postfix ပုံစံ နှစ်မျိုးနဲ့ အသုံးပြုနိုင်ပါတယ်။

Library Functions

C++ ရဲ့လုပ်ဆောင်မှု အတော်များများကို library functions တွေ ခေါ်ယူ အသုံးပြုခြင်းအားဖြင့် ရေးသားလေ့ ရှိပါတယ်။ ၎င်း functions များကို အသုံးပြုပြီး ဖိုင်များကို ရယူသုံးစွဲခြင်း၊ သင်္ချာဆိုင်ရာ တွက်ချက်မှုများ ပြုလုပ်ခြင်း၊ data များ conversion ပြုလုပ်ခြင်းနှင့် အခြားမြောက်များလှစွာသော လုပ်ဆောင်မှုများကို ဆောင်ရွက်ပေးနိုင်မှာ ဖြစ်ပါတယ်။ ကျွန်တော်တို့အနေနဲ့ ဒီအခန်းမှာတော့ library function အချို့အသုံးပြုပုံကို sqrt.cpp ဆိုတဲ့ နမူနာ ပရိုဂရမ်လေးသုံးပြီး မိတ်ဆက်ပေးချင်ပါတယ်။ ဒီပရိုဂရမ်လေးမှာ sqrt() ဆိုတဲ့ library function လေးကို သုံးပြီး user ထည့်သွင်းပေးလိုက်တဲ့ ဂဏန်းတွေကို square root ရှာပေးမှာ ဖြစ်ပါတယ်။

```
// sqrt.cpp
// demonstrates sqrt() library function
#include <iostream> //for cout, etc.
#include <cmath> //for sqrt()
#include <conio.h> //for _getch();
using namespace std;
int main()
{
    double number, answer; //sqrt() requires type double
    cout << "Enter a number: ";
    cin >> number; //get the number
    answer = sqrt(number); //find square root
    cout << "Square root is "
         << answer << endl; //display it
    _getch(); //wait for any key press
    return 0;
}
```

f:\Articles 2013-3\OOP in C++\4-charvars\eg7-4 sqrt\Debug\eg7-4 sqrt.exe

Enter a number: 22
Square root is 4.69042

ပထမဆုံး user ဆီက တောင်းလို့ ရလာတဲ့ တန်ဖိုးကို sqrt() ဆိုတဲ့ library function ရဲ့ နောက်က လက်သဲကွင်း အဖွင့်အပိတ်ထဲကို argument အနေနဲ့ ထည့်သွင်းပေးလိုက်တဲ့အခါမှာ library function က အဲဒီ argument ရဲ့ square root တန်ဖိုးကို တွက်ထုတ်ပြီး ပြန်ပေးပါတယ်။ အဲဒီ တန်ဖိုးကို assignment operator (=) သုံးပြီး answer ဆိုတဲ့ variable ထဲကို ထည့်သွင်းပေးတာ ဖြစ်ပါတယ်။ Library function တစ်ခုကို အသုံးပြုတဲ့ နေရာမှာ ထည့်သွင်းပေးရမယ့် argument နဲ့ ပြန်ရလာမယ့် return value တွေဟာ မှန်ကန်တဲ့ data type တွေ ဖြစ်နေဖို့ လိုပါတယ်။ ဒီဥပမာမှာ

ပါတဲ့ sqrt() function မှာတော့ argument ရော return value ပါ double ဖြစ်တဲ့အတွက် variables နှစ်ခုစလုံးကို double နဲ့ အသုံးပြုခဲ့တာ ဖြစ်ပါတယ်။

Header Files

ကျွန်တော်တို့ အနေနဲ့ cout, sqrt() စတဲ့ objects တွေ၊ library functions တွေကို ယူသုံးတဲ့ အခါတိုင်းမှာ သက်ဆိုင်ရာ header file တွေကို #include သုံးပြီး ကြေငြာပေးဖို့ လိုမှာ ဖြစ်ပါတယ်။ sqrt() ရဲ့ header file ဟာ cmath ဖြစ်တာကြောင့် #include<cmath> ဆိုပြီး ရေးသားပေးရပါမယ်။ အဲဒီလို ရေးသားခြင်းအားဖြင့် preprocessor က cmath ဆိုတဲ့ header file ကို ကျွန်တော်တို့ရဲ့ source file ဖြစ်တဲ့ sqrt.cpp ထဲကို ပေါင်းစပ် ထည့်သွင်းပေးမှာ ဖြစ်ပါတယ်။ တကယ်လို့ header file ကို ကြေငြာပေးဖို့ မေ့လျော့ခဲ့တယ်ဆိုရင်တော့ compiler ကနေ 'sqrt' unidentified identifier ဆိုတဲ့ error message ပေးမှာ ဖြစ်ပါတယ်။

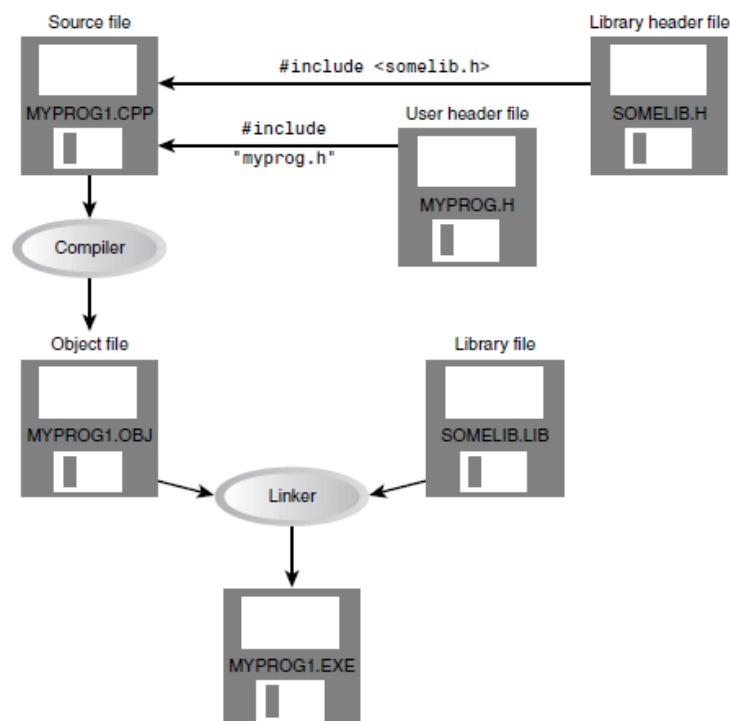
Library Files

Compiler တစ်ခုဟာ executable ဖိုင် (.exe) တစ်ခု တည်ဆောက်ဖို့အတွက် library functions တွေနဲ့ objects တွေ ပါဝင်တဲ့ ဖိုင်တွေကို ချိတ်ဆက်ပြီး ဖန်တီး ယူရတာ ဖြစ်ပါတယ်။ အဲဒီဖိုင်တွေမှာ အဲဒီ function တွေနဲ့ objects တွေရဲ့ machine-executable code တွေ ပါဝင်ပါတယ်။ Library ဖိုင်တွေဟာ များသောအားဖြင့် .lib extension နဲ့ အဆုံးသတ်လေ့ ရှိပါတယ်။ sqrt() function ကို အဲဒီလို ဖိုင်တွေထဲမှာ ထည့်သွင်းထားတာ ဖြစ်ပါတယ်။ အဲဒီ function တွေကို linker က library ဖိုင်ကနေပြီး ခွဲထုတ်ကာ ပရိုဂရမ်က ခေါ်ယူ အသုံးပြုနိုင်အောင် ချိတ်ဆက်ပေးမှာ ဖြစ်ပါတယ်။ အဲဒီ အသေးစိတ် အချက်အလက်တွေကို ကျွန်တော်တို့ ဘာမှလုပ်ပေးဖို့ မလိုအပ်ပါဘူး။ Compiler က လိုအပ်သလို ဆောင်ရွက်ပေးသွားမှာပါ။ ဒါပေမယ့် ပရိုဂရမ်မှာ တစ်ယောက် အနေနဲ့ကတော့ အဲဒီဖိုင်တွေအကြောင်းကို သိထားဖို့ လိုအပ်မှာ ဖြစ်ပါတယ်။

Header Files and Library Files

Library ဖိုင်နဲ့ header ဖိုင်တွေရဲ့ ဆက်စပ်မှုကို နားလည်ရ ခက်နေတတ်ပါတယ်။ ဒါကြောင့် အနည်းငယ် ထပ်မံ ရှင်းပြလိုပါတယ်။ ကျွန်တော်တို့ ရေးသားတဲ့ ပရိုဂရမ်ထဲမှာ sqrt() ဆိုတဲ့ library function တစ်ခုကို ထည့်သုံးချင်တယ် ဆိုပါစို့။ ဒါဆို အဲဒီ function ပါတဲ့ library ဖိုင်ကို ကျွန်တော်တို့ ပရိုဂရမ်နဲ့ ချိတ်ဆက်ပေးဖို့ လိုလာပါတယ်။ အဲဒီလို ပြုလုပ်ဖို့ linker က တာဝန်ယူ ဆောင်ရွက်ပေးပါတယ်။ ဒါပေမယ့် ဒီမှာ မပြီးသေးပါဘူး။ ကျွန်တော်တို့ ရေးထားတဲ့ source ဖိုင် ထဲမှာ

အဲဒီ library ဖိုင်ထဲမှာ ပါတဲ့ functions တွေရဲ့ နာမည်တွေ၊ အမျိုးအစားတွေနဲ့ အခြား အချက်အလက်တွေကို သိဖို့ လိုအပ်နေပါသေးတယ်။ အဲဒီအချက်အလက်တွေဟာ တကယ်တော့ header file ထဲမှာ စုစည်းထားတာ ဖြစ်ပြီး header file တစ်ခုစီမှာ သက်ဆိုင်ရာ functions အချို့ရဲ့ အချက်အလက်များကို သိမ်းဆည်းထားပါတယ်။ functions တွေကိုတော့ စုစည်းပြီး library file အနေနဲ့ စုစည်းထားလေ့ ရှိတယ်။ အဲဒီ library ဖိုင် တစ်ခုအတွက် header file အများအပြား ခွဲခြား ထားတတ်ပါတယ်။ ဥပမာ iostream header file ထဲမှာ cout အစရှိတဲ့ I/O functions အများအပြားနဲ့ objects တွေရဲ့ အချက်အလက်တွေကို စုစည်းထားသလို cmath header file ထဲမှာတော့ sqrt() အစရှိတဲ့ သင်္ချာ functions တွေရဲ့အချက်အလက်တွေ ပါဝင်ပါတယ်။ တကယ်လို့ ကျွန်တော်တို့အနေနဲ့ string နှစ်ခုကို နှိုင်းယှဉ်တဲ့ strcpy() ဆိုတဲ့ function ကို အသုံးပြုချင်တယ် ဆိုရင်တော့ string.h ကို ကြေငြာပေးဖို့ လိုမှာ ဖြစ်ပါတယ်။ C++ မှာ header file တွေ အမြဲတမ်း အသုံးပြုလေ့ ရှိပြီး ကျွန်တော်တို့ အနေနဲ့ library function တစ်ခု၊ ဒါမှမဟုတ် ကြိုတင်သတ်မှတ် ရေးသားထားတဲ့ object တွေ operator တွေကို သုံးတဲ့ အချိန်တိုင်းမှာ သက်ဆိုင်ရာ header file တွေကို ကြေငြာပေးရပါမယ်။ အောက်က ပုံလေးမှာ header file နဲ့ library ဖိုင်တွေ အသုံးပြုပုံကို ရှင်းပြထားပါတယ်။



Two Ways to Use #include

ကျွန်တော်တို့ အနေနဲ့ #include ကို နည်းလမ်း နှစ်မျိုးနဲ့ အသုံးပြုနိုင်ပါတယ်။ ပထမ တနည်းက angle brackets <> ကြားမှာ filename တွေကို ရေးတဲ့ နည်းဖြစ်ပါတယ်။ အထက်က နမူနာ ပရိုဂရမ်တွေထဲကလို #include<iostream> နဲ့ #include<cmath> တို့ကို ရေးခဲ့သလိုမျိုးပါ။ ဒီလို ရေးလိုက်တာနဲ့ compiler က အဲဒီဖိုင်တွေကို INCLUDE ဆိုတဲ့ directory ထဲမှာ သွားရှာမှာ ဖြစ်ပါတယ်။ အဲဒီ directory ထဲမှာ compiler ကို ထုတ်လုပ်သူက ထည့်ပေးလိုက်တဲ့ header files တွေ အကုန်လုံးကို စုစည်းထားတာ ဖြစ်ပါတယ်။ သာမန်အားဖြင့် ဒီနည်းကိုပဲ သုံးကြရပါတယ်။ ဒါပေမယ့် တကယ်လို့ ကျွန်တော်တို့ရဲ့ ကိုယ်ပိုင် header file တွေ ရေးသား အသုံးပြုလာကြမယ် ဆိုရင်တော့ #include"myheader.h" ဆိုတဲ့ ပုံစံမျိုးနဲ့ quotation marks "" ကြားမှာ ရေးသားပေးရမှာ ဖြစ်ပါတယ်။ အဲဒီလို ရေးသားခြင်း အားဖြင့် compiler က အဲဒီဖိုင်ကို INCLUDE directory မှာ ရှာမယ့်အစား current directory ထဲမှာ သွားရှာပါလိမ့်မယ်။ တကယ်တော့ current directory ဆိုတာ ကျွန်တော်တို့ ရေးသားနေတဲ့ ပရိုဂရမ်ရဲ့ source ဖိုင် ရှိတဲ့ နေရာဖြစ်ပါတယ်။ အဲဒီတော့ ကျွန်တော်တို့ ရေးသားထားတဲ့ header file နဲ့ source ဖိုင်တွေကို နေရာတစ်ခုထဲမှာ စုစည်းထားဖို့လိုပါတယ်။ Multifile Programs များအကြောင်း ရေးသားတဲ့ အချိန်ကျမှ အခုထက်ပိုပြီး အသေးစိတ် ဆွေးနွေးပေးပါဦးမယ်။

Summary (Chapter 1)

ဒီ Chapter မှာ C++ ရဲ့အခြေခံအကျဆုံး အစိတ်အပိုင်းက function တွေ ဖြစ်တယ်ဆိုတာကို လေ့လာခဲ့ပြီးပါပြီ။ ပရိုဂရမ် တစ်ပုဒ်မှာ ပထမဆုံး အလုပ်လုပ်တဲ့ function ကတော့ main() ပဲ ဖြစ်ပါတယ်။ function တစ်ခုကို statements တွေနဲ့ ဖွဲ့စည်းထားတာ ဖြစ်ပြီး ၎င်းတို့က ကွန်ပျူတာကို ညွှန်ကြား ခိုင်းစေနေတာပဲ ဖြစ်ပါတယ်။ Statements တိုင်းဟာ semicolon (;) နဲ့ အဆုံးသတ်လေ့ ရှိပါတယ်။ Statement တစ်ခုမှာ expressions တွေ တစ်ခုထက် မက ပိုပြီး ပါနိုင်ပါတယ်။ Expression ဆိုတာကတော့ တန်ဖိုးတစ်ခု ထွက်လာအောင် variables တွေနဲ့ operators တွေ စုစည်းပြီး ရေးသားထားတာပဲ ဖြစ်ပါတယ်။

C++ မှာ output value တွေကို screen မှာ ပြသပေးဖို့ cout object နဲ့ insertion operator (<<) ကို အသုံးပြုလေ့ ရှိပါတယ်။ Input တွေကို ဖတ်ယူဖို့ အတွက်တော့ cin နဲ့ extraction operator (>>) ကို အသုံးပြုပြီး standard input device (keyboard) ကနေ ထည့်သွင်းပေးလိုက်တဲ့ တန်ဖိုးတွေကို variable တွေထဲကို ထည့်သွင်းပေးပါတယ်။

C++ မှာ built in data types တွေ အများအပြားရှိပါတယ်။ char,int,long နဲ့ short တို့ဟာ integer data type တွေဖြစ်ကြပြီး float, double နဲ့ long double တို့ကတော့ floating-point types တွေပဲ ဖြစ်ပါတယ်။ အဲဒီ types အားလုံးဟာ အပေါင်းနဲ့ အနုတ် တန်ဖိုး နှစ်ခုလုံး ထည့်သွင်းလို့ရတဲ့ (signed) types တွေပဲ ဖြစ်ပါတယ်။ integer types တွေကို အနုတ်တန်ဖိုး ထည့်သွင်းဖို့ မလိုအပ်တဲ့ အခြေအနေမျိုးမှာ unsigned ဆိုတဲ့ keyword ကို အသုံးပြုခြင်းဖြင့် အရွယ်အစား နှစ်ဆ ပိုကြီးလာစေပါတယ်။ true နဲ့ false တန်ဖိုး နှစ်ခုသာ ပါဝင်တဲ့ Boolean variables တွေကို သိမ်းဆည်းဖို့ bool type အသုံးပြုနိုင်ပါတယ်။

const ဆိုတဲ့ keyword လေးကတော့ variable တစ်ခုကို constant အသွင်ပြောင်းပစ်ပြီး ၎င်းထဲမှာ ထည့်သွင်းထားတဲ့ တန်ဖိုးကို ပြောင်းလဲခွင့် မပေးတော့ပါဘူး။

Variable တစ်ခုဟာ မတူညီတဲ့ data types တွေ ရောနှောရေးထားတဲ့ expressions တွေထဲမှာ အလိုအလျောက် conversion ပြုလုပ်ခံရလေ့ ရှိပြီး အကယ်၍ programmer က သတ်မှတ်ပေးထားတဲ့ data type သို့စိတ်ကြိုက် conversion ပြုလုပ်ချင်ရင်တော့ casting လုပ်ပေးရမှာပါ။

C++ မှာ သုံးနေကျ +, -, *, / တွေ အပြင် အကြွင်းရှာဖို့အတွက် remainder operator (%) ကိုပါ ထည့်သွင်းပေးထားပါတယ်။

Arithmetic assignment operator တွေဖြစ်တဲ့ +=, -= စတာတွေဟာ arithmetic operation နဲ့ assignment operation တွေကို တစ်ဆက်တည်း လုပ်ဆောင်ပေးပါတယ်။ (ဥပမာ - operator ညာဘက်က တန်ဖိုးကို ဘယ်ဘက်က variable နဲ့ ပေါင်း/နုတ်/မြှောက်/စား ပြီး ရလဒ်ကို အဲဒီ variable ထဲသို့ ပြန်ထည့်ပေးလိုက်တာ)။ Increment and decrement operators (++,-) တွေကတော့ variable ထဲကို ၁ ပေါင်းခြင်း သို့မဟုတ် နုတ်ခြင်း ပြုလုပ်ပေးမှာ ဖြစ်ပါတယ်။

Preprocessor directives တွေကတော့ statement တွေလို မဟုတ်ပါဘူး။ ၎င်းတို့က computer ကို ညွှန်ကြားချက်မပေးဘဲ compiler ကို ညွှန်ကြားချက်ပေးတာပဲ ဖြစ်ပါတယ်။ #include ဆိုတဲ့ directive ကတော့ လက်ရှိ source ဖိုင် ထဲကို အခြား ဖိုင်တစ်ခု ထည့်သွင်းပေးဖို့ compiler ကို ညွှန်ကြားတာပဲ ဖြစ်ပါတယ်။ #define ဆိုတဲ့ directive ကတော့ ညွှန်ကြားချက် တစ်ခုကို နောက်ညွှန်ကြားချက် တစ်ခုနဲ့ အစားထိုးဖို့ စေခိုင်းတာပါ။ using ဆိုတဲ့ directive ကတော့ compiler ကို namespace တစ်ခုရဲ့ နာမည်တွေ မှတ်သားထားစေတာပဲ ဖြစ်ပါတယ်။

ကျွန်တော်တို့ ပရိုဂရမ်တွေထဲမှာ library function တွေကို ခေါ်ယူအသုံးပြုခဲ့မယ် ဆိုရင် အဲဒီ function ရဲ့ တကယ့် machine-code တွေက သက်ဆိုင်ရာ library file တွေထဲမှာ ရှိနေပြီး အလိုအလျောက် ချိတ်ဆက် အသုံးပြုမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် အဲဒီ function ရဲ့ declaration တွေပါဝင်တဲ့ header file ကိုတော့ ကျွန်တော်တို့ source file ထဲမှာ #include ကို အသုံးပြုပြီး ထည့်သွင်းပေးရမှာ ဖြစ်ပါတယ်ခင်ဗျာ။

အခန်း(၂)

Loops and Decisions

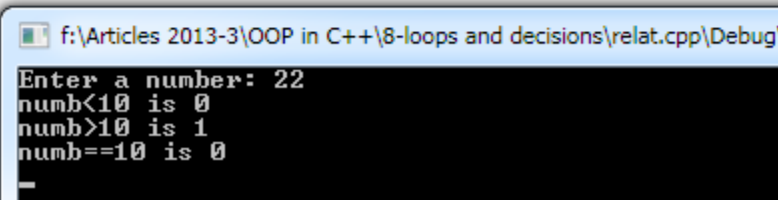
Relational Operators

ပရိုဂရမ် အများစုသည် အစမှ အဆုံးအထိ အစဉ်လိုက် အလုပ်လုပ်သည် ပုံစံမျိုး မရှိကြပါ။ အသုံးပြုသူ ဆုံးဖြတ်သည့်အတိုင်း အခြေအနေပြောင်းလဲမှုကို လိုက်လျောညီထွေဖြစ်အောင် ပြုမူသည့် ပုံစံမျိုး ရေးသားလေ့ ရှိကြသည်။ ပရိုဂရမ်အတွင်းရှိ တွက်ချက်မှုများပေါ် တူတည်၍ Control flow ကို ပရိုဂရမ် ၏ တစ်နေရာမှ အခြားတစ်နေရာသို့ jump လုပ်ကာ ဆောင်ရွက်လေ့ ရှိသည်။ ထိုသို့ jump ဖြစ်စေနိုင်သော statements များကို control statements များဟု ခေါ်သည်။ ၎င်းတို့ကို loops နှင့် decisions ဟူ၍ ခေါင်းစဉ်နှစ်ခု ခွဲခြား လေ့လာနိုင်ပါသည်။

loop တစ်ခုကို အကြိမ်မည်မျှ ပတ်မည် သို့မဟုတ် decision တစ်ခုကို မည်သို့လုပ်ဆောင်မည် ဆိုသည့် အချက်သည် သက်ဆိုင်ရာ expression များ မှန်သည် မှားသည် ဆိုသည့် အပေါ် မူတည်နေပေသည်။ အဆိုပါ expression များ အတွင်း၌ ရှိသော တန်ဖိုးနှစ်ခုကို နှိုင်းယှဉ်ရန် အသုံးပြုသည့် operator များကို relational operator ဟု ခေါ်ဆိုကြသည်။ Loops နှင့် decisions များကို လေ့လာရာတွင် ၎င်း relational operator များအား အသုံးပြုရသဖြင့် ဦးစွာ ရှင်းလင်း တင်ပြပေးပါမည်။

Relational Operators များသည် C++ ၏ မည်သည့် built-in data type တန်ဖိုး နှစ်ခုကိုမဆို နှိုင်းယှဉ်ပေးနိုင်ပါသည်။ ထို့ပြင် နောက်ပိုင်းလေ့လာကြရမည့် user-defined classes များကိုလည်း နှိုင်းယှဉ်ပေးနိုင်စွမ်း ရှိပါသည်။ ထိုသို့ နှိုင်းယှဉ်ရာတွင် (==) တူညီသည်၊ (>) ကြီးသည်၊ (<) ငယ်သည်ဟူသော နှိုင်းယှဉ်ချက်များ ပြုလုပ်လေ့ ရှိပါသည်။ နှိုင်းယှဉ်ချက် ဖော်ပြချက်၏ ရလဒ်မှာ မှန်လျှင် true ဖြစ်၍ မှားလျှင် false ဖြစ်သည်။ ပိုမိုရှင်းလင်း လွယ်ကူစွာ နားလည် သဘောပေါက်စေရန် relat.cpp ပရိုဂရမ် အား လေ့လာကြည့်ကြပါစို့။

```
// relat.cpp
// demonstrates relational operators
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    int numb;
    cout << "Enter a number: ";
    cin >> numb;
    cout << "numb<10 is " << (numb < 10) << endl;
    cout << "numb>10 is " << (numb > 10) << endl;
    cout << "numb==10 is " << (numb == 10) << endl;
    _getch();
    return 0;
}
```



အထက်ပါ ပရိုဂရမ်တွင် user ထည့်သွင်းပေးလိုက်သော တန်ဖိုးနှင့် ၁၀ အား နှိုင်းယှဉ်မှု ၃ ခု ပြုလုပ်ထားပါသည်။ user မှ 22 အား input အဖြစ် ထည့်သွင်းပေးလိုက်ချိန်တွင် နှိုင်းယှဉ်မှု ၃ ခု၏ ရလဒ်များကို အထက်ပါအတိုင်း မှန်လျှင် 1 မှားလျှင် 0 ဖြင့် ဖော်ပြထားသည်ကို တွေ့ရပါသည်။ ပထမ နှိုင်းယှဉ်ချက်မှာ numb (22) သည် 10 ထက် ငယ်သလား ဟူသော နှိုင်းယှဉ်ချက်ဖြစ်ပြီး မှားယွင်းသဖြင့် ရလဒ်မှာ 0 ဖြစ်ပါသည်။ ဒုတိယ နှိုင်းယှဉ်ချက်မှာ numb (22) သည် 10 ထက် ကြီးသလား ဟူသော နှိုင်းယှဉ်ချက်ဖြစ်၍ မှန်ကန်သောကြောင့် ရလဒ်မှာ 1 ဖြစ်သည်။ နောက်ဆုံး နှိုင်းယှဉ်ချက်မှာ numb (22) သည် 10 နှင့် ညီသလား ဟူသော နှိုင်းယှဉ်ချက်ဖြစ်ပြီး မှားယွင်းသောကြောင့် ရလဒ်မှာ 0 ဖြစ်ပါသည်။ ကျွန်တော်တို့ အနေဖြင့် ရလဒ်များကို true, false များဖြင့် မျှော်လင့်ထားသော်လည်း တကယ်တမ်းတွင် compiler က 1,0 ဖြင့်သာ အဖြေ ထုတ်ပေးပါသည်။ သို့ဖြစ်၍ အဆိုပါ ပြဿနာတွင် compiler အတွက် 1 သည် true ဖြစ်၍ 0 သည် false အဖြစ် သတ်မှတ်ထားကြောင်း ထင်ရှားပါသည်။ အကယ်၍ ကျွန်တော်တို့အနေဖြင့် bool data type ကို အသုံးပြုခဲ့လျှင်ပင် အဖြေအား အထက်ပါအတိုင်းသာ ရရှိမည် ဖြစ်သည်။ အဘယ်ကြောင့်ဆိုသော် C++ စတင်ပေါ်ပေါက်ကာစက bool data type မရှိသေးသောကြောင့် ဖြစ်ပါသည်။ ထို့ကြောင့် ယခုအခါ C++ တွင် အမှားမှန်ကို 0,1 သာမက true,false နှင့်ပါ ဖော်ပြနိုင်ပေသည်။

လက်တွေ့တွင်မူ အဆိုပါ ကွဲပြားချက်သည် သိပ်အရေးမပါလှပါ။ ကျွန်တော်တို့ အနေဖြင့် true, false ဖြင့် ဖော်ပြခြင်း၊ မဖော်ပြခြင်းထက် looping တစ်ခု သို့မဟုတ် decision တစ်ခု ဘာဆက်လုပ်ရမည် ဆိုသည်ကို ဆုံးဖြတ်ရန်သာ လိုအပ်ပေသည်။

အောက်ပါ ဇယားတွင် C++ ၏ relational operator များအားလုံးကို ဖော်ပြပေးထားပါသည်။

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

အထက်ပါ Relational operators များအား အသုံးပြုပုံကို အောက်တွင် နမူနာ expression များဖြင့် ဖော်ပြထားပါသည်။ ပထမ နှစ်ကြောင်းမှာ harry နှင့် jane ဆိုသော variable များထဲသို့ တန်ဖိုးများ သတ်မှတ် ထည့်သွင်းခြင်းဖြစ်ပြီး ကျန်စာကြောင်းများက ၎င်းတန်ဖိုးများကို relational operators များအသုံးပြု၍ အမျိုးမျိုး နှိုင်းယှဉ် ပြထားခြင်းပင် ဖြစ်သည်။

```
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)
```

မှားလေ့ရှိသော အချက်တစ်ခုမှာ equal operator (==) ကို equal signs နှစ်ခု သုံးရမည့်အစား တစ်ခုတည်း သုံးမိခြင်းဖြစ်သည်။ ထိုအမှားမျိုးသည် compiler က မှားမှန်း မသိခြင်းကြောင့် ရှာဖွေရန် ခက်ခဲလေ့ ရှိသည်။

C++ အနေဖြင့် true ကို 1 ဖြင့် ကိုယ်စားပြုလေ့ ရှိသော်လည်း သုညမဟုတ်သည့် မည်သည့် ဂဏန်း (အနုတ်တန်ဖိုးအပါအဝင်) ကိုမဆို true အဖြစ် လက်ခံလေ့ ရှိသည်။ false အတွက်မူ သုည 0 တစ်ခုတည်းသာ သတ်မှတ်ထားသည်။

Loop

Loops ဆိုသည်မှာ ပရိုဂရမ်၏ အစိတ်အပိုင်းအချို့ကို အကြိမ်အရေအတွက် တစ်ခုအထိ အကြိမ်ကြိမ် လုပ်ဆောင်ခြင်းပင် ဖြစ်သည်။ ထိုသို့ ထပ်ခါတလဲလဲ လုပ်ဆောင်ခြင်းသည် condition အခြေအနေ မှန်ကန်နေသမျှပင် ဖြစ်သည်။ အဆိုပါ condition မှားယွင်းသွားသည်နှင့်တပြိုင်နက် Loop ပြီးဆုံးသွားပြီး Loop နောက်က statements များကို ဆက်လက် လုပ်ဆောင်သွားမည် ဖြစ်သည်။ C++ တွင် for loop, while loop နှင့် do loop ဟူ၍ loops သုံးမျိုး ရှိသည်။

The for Loop

for loop သည် လူအများအတွက် လေ့လာရာတွင် အလွယ်ဆုံး loop ဖြစ်လေ့ ရှိသည်။ for loop တွင် control elements များသည် တစ်နေရာထဲ၌ စုစည်းထားလေ့ရှိပြီး အခြား loop များတွင်မူ control များသည် ပြန့်ကျဲတည်ရှိသဖြင့် သဘောပေါက်ရန် ခက်ခဲတတ်ခြင်း ဖြစ်သည်။ ၎င်းသည် ကုဒ်အစိတ်အပိုင်းများကို တိကျသော အကြိမ်အရေအတွက်အတွင်း လုပ်ဆောင်လေ့ ရှိသည်။ များသောအားဖြင့် for loop ကို အကြိမ်မည်မျှ လုပ်ဆောင်မည်ဆိုသည့် အချက်အား ကြိုတင် တွက်ဆထားနိုင်သည့် အခြေအနေမျိုးတွင် အသုံးပြုလေ့ ရှိသည်။ အောက်ပါ fordemo.cpp ပရိုဂရမ်တွင် for loop ကို အသုံးပြု၍ 0 မှ 14 အတွင်းရှိ ဂဏန်းများ၏ နှစ်ထပ်ကိန်းကို ရှာပြထားသည်။

```
// fordemo.cpp
// demonstrates simple FOR loop
#include <iostream>
using namespace std;
int main()
{
    int j; //define a loop variable
```



```

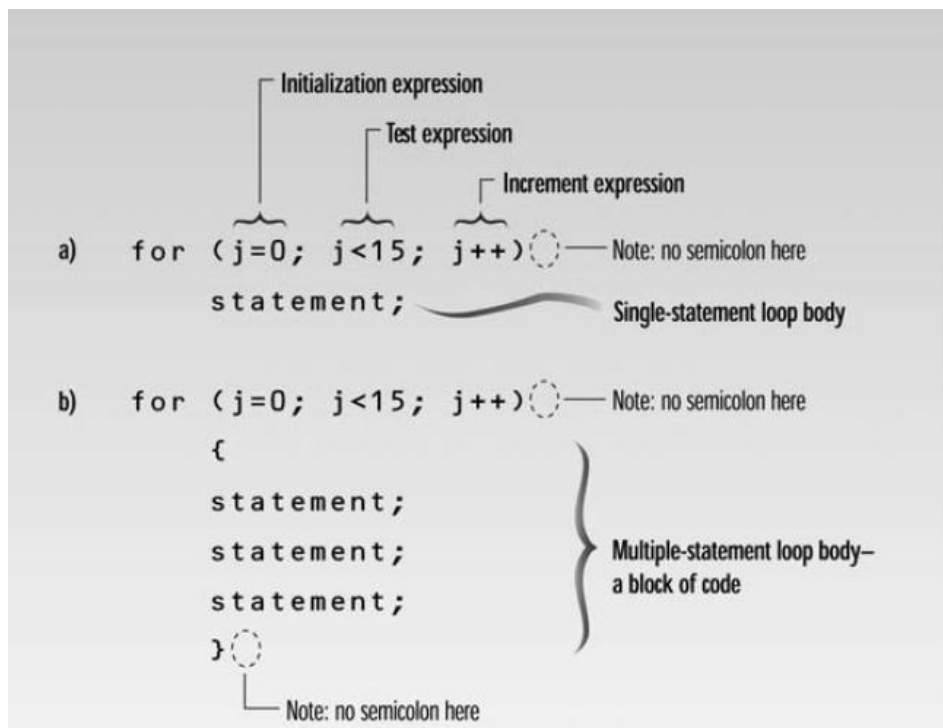
for(j=0; j<15; j++) //loop from 0 to 14,
    cout << j * j << " "; //displaying the square of j
cout << endl;
return 0;
}

```

ပရိုဂရမ်ကို RUN ကြည့်လျှင် အောက်ပါ အဖြေများကို ရရှိမည် ဖြစ်သည်။

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

ဒီ အဖြေတွေ ဘယ်လိုရလာသလဲ လေ့လာကြည့်ကြရအောင်။ for loop မှာ for ဆိုတဲ့ keyword နောက်က လက်သဲ ကွင်းစ၊ ကွင်းပိတ်ထဲမှာ expression သုံးခုကို semicolons တွေနဲ့ ခြားထားပါတယ် - for(j=0; j<15; j++)။ အဲဒီ expression သုံးခုကတော့ initialization expression, test expression နဲ့ increment expression တို့ပဲ ဖြစ်ပါတယ်။



အဆိုပါ expression များတွင် (ခြွင်းချက်အချို့မှအပ) loop variable သို့မဟုတ် looping counter ဟုခေါ်သော variable တစ်ခုတည်းကိုသာ အသုံးပြုလေ့ ရှိသည်။ fordemo.cpp တွင် အသုံးပြုခဲ့သော loop variable မှာ j ဖြစ်သည်။ ၎င်းကို for loop မစမီ defined ပြုလုပ်ပေးထားရန် လိုအပ်သည်။

Loop ၏ အတွင်းရှိ statements များသည် loop ပတ်နေသမျှ လုပ်ဆောင်နေရမည့် ကုဒ်များပင် ဖြစ်သည်။ fordemo.cpp တွင်မူ cout << j * j << " "; ကုဒ်တစ်ကြောင်းတည်းသာ အသုံးပြုထားသည်။

အထက်ပါ statement သည် j တန်ဖိုး၏ နှစ်ထပ်ကိန်းနှင့် နောက်တွင် နေရာလွတ် (spaces) နှစ်နေရာ ကို print လုပ်ပေးပါမည်။ နှစ်ထပ်ကိန်းရရှိရန် j ကို j ဖြင့်ပင် မြှောက်ပေးလိုက်မည် ဖြစ်သည်။ loop ကို ပတ်နေသမျှ j တန်ဖိုးသည် 0,1,2,3, မှ 14 အထိ ပြောင်းလဲလာမည် ဖြစ်သည်။ ထို့ကြောင့် နှစ်ထပ်ကိန်းများဖြစ်သော 0, 1, 4, 9, မှ 196 အထိ ကို ရရှိမည် ဖြစ်သည်။ မှတ်သားရမည့် အချက်တစ်ခုမှာ for statement ၏ အဆုံးတွင် semicolon မထည့်ရခြင်းပင် ဖြစ်သည်။ အဘယ်ကြောင့်ဆိုသော် for statement နှင့် loop body အားလုံးကို ပေါင်း၍ statement တစ်ကြောင်းအဖြစ် သတ်မှတ်ထားသောကြောင့် ဖြစ်သည်။ အကယ်၍ for statement အဆုံးတွင် semicolon ထည့်ပေးလိုက်ပါက compiler က loop body မရှိဟု ယူဆသွားမည် ဖြစ်ပြီး မျှော်လင့် မထားသည့် ရလဒ်များ ထွက်ပေါ်လာမည် ဖြစ်သည်။ ၎င်း expressions သုံးခုက loop ကို ဘယ်လို ထိန်းချုပ်သွားသည်ကို လေ့လာကြရအောင်။

The Initialization Expression

Initialization expression ကို loop စတင်ချင်း တစ်ကြိမ်သာ လုပ်ဆောင်ပါသည်။ ၎င်းက loop variable ကို initial value သတ်မှတ်ပေးသည်။ fordemo.cpp တွင်မူ ၎င်းက j ၏ အစတန်ဖိုးကို 0 သတ်မှတ်ပေးပါသည်။

The Test Expression

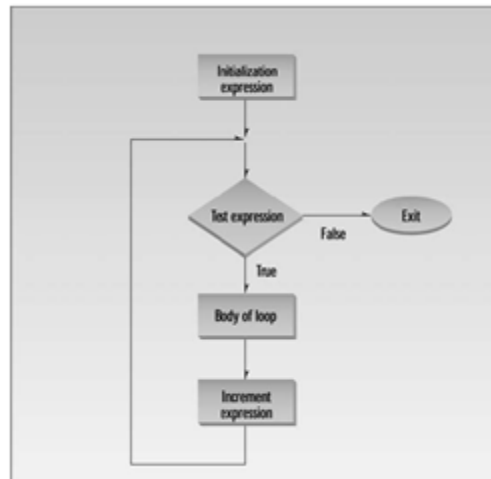
Test expression တွင် relational operator ပါဝင်လေ့ ရှိသည်။ Loop ပတ်သည့် အကြိမ်တိုင်း မပတ်ခင် ၎င်းကို မှန်မမှန် စစ်ဆေးသည်။ မှန်လျှင် loop ပတ်၍ မှားခဲ့လျှင် loop မှ ထွက်သွားမည် ဖြစ်သည်။

The Increment Expression

Increment expression ကတော့ loop variable ကို loop တစ်ပတ် ပတ်ပြီးတိုင်း ပြောင်းလဲပေးပါတယ်။ များသောအားဖြင့် increment (တိုးပေး) လုပ်ပေးတာပါ။ ဥပမာ j++ ဆိုရင် loop တစ်ပတ် ပတ်ပြီးတိုင်း j တန်ဖိုးကို 1 တိုးပေးတာပါ။ အောက်ကပုံမှာ for loop ရဲ့ flowchart ကို ပြထားပါတယ်။

How Many Times?

fordemo.cpp ပရိုဂရမ်လေးမှာ ပါတဲ့ for loop ဟာ ၁၅ ကြိမ်တိတိ အလုပ်လုပ်ပါတယ်။ ပထမဦးဆုံးမှာ j တန်ဖိုးက 0 ဖြစ်ပါတယ်။ ဒါကို initialization expression မှာ သတ်မှတ်ပေးပါတယ်။ နောက်ဆုံး ဖြစ်ရမယ့် j တန်ဖိုးက 14 ပါ။ ဒါကို test expression $j < 15$ က သတ်မှတ်ပေးထားတာပါ။ တကယ်လို့ j တန်ဖိုး 15 ဖြစ်သွားပြီး 14 ထက် ကြီးသွားမယ်ဆိုရင် အဲဒီ expression မှားသွားလို့ for loop ကနေ ထွက်သွားမှာ ဖြစ်ပါတယ်။ များသောအားဖြင့် loop variable ကို 0 ထားပါတယ်။ test expression ကို less-than < operator ကိုသုံးပြီး နောက်က တန်ဖိုးကို ကျွန်တော်တို့ ပတ်ချင်တဲ့ အကြိမ်အရေအတွက်အတိုင်းထားပါတယ်။ (ဒီ ဥပမာမှာက ၁၅ ဖြစ်ပါတယ်)။ တစ်ပတ် ပတ်ပြီးတိုင်း increment လုပ်ပါတယ်။ fordemo.cpp မှာတော့ loop body မှာ အလုပ်လုပ်ဖို့ ကုန် တစ်ကြောင်းသာ ရေးထားပါတယ်။ `cout << j * j << " ";`



နောက်ထပ် ဥပမာလေး တစ်ခု ရေးပြပါမယ်။

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

```
//loop body
```

အထက်ပါ ပရိုဂရမ်လေးဟာ အကြိမ် ၁၀၀ အတိအကျ လုပ်ပါလိမ့်မယ်။ count တန်ဖိုး 0 ကနေ 99 အထိ for loop ကို ပတ်နေမှာကြောင့်ပဲ ဖြစ်ပါတယ်။

Multiple Statements in the Loop Body

ကျွန်တော်တို့ အနေနဲ့ loop body ထဲမှာ statement တစ်ကြောင်းမက ရေးဖို့ လိုအပ်လေ့ ရှိပါတယ်။ အဲဒီလို အခြေအနေမျိုးမှာ function တွေ ရေးသလိုမျိုး တွန့်ကွင်း{} အဖွင့်အပိတ်ကြားမှာ ရေးပေးရမှာ ဖြစ်ပါတယ်။ ကွင်းထဲက statement တစ်ကြောင်းချင်းရဲ့အဆုံးမှာ semicolon ခံပေးရမှာ ဖြစ်ပေမယ့် ကွင်းပိတ်နောက်မှာတော့ demicolon ထည့်ပေးရပါဘူး။ cubelist.cpp ဆိုတဲ့ နောက်

နမူနာလေး ရေးပြထားပါတယ်။ ၎င်းက ၁ ကနေ ၁၀ ကြား ဂဏန်းတွေရဲ့ သုံးထပ်ကိန်းတွေကို ကော်လံ နှစ်ခု နဲ့ တွက်ထုတ်ပေးမှာ ဖြစ်ပါတယ်။ လေ့လာကြည့်ကြရအောင်..

```
// 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;
}
```

အဖြေတွေကတော့ အောက်ပါအတိုင်း ဖြစ်ပါတယ်-

```
1 1
2 8
3 27
4 64
5 125
6 216
7 343
8 512
9 729
```

10 1000

ဒီပရိုဂရမ်လေးမှာ နောက်ထပ် ပြောင်းလဲမှု အနည်းငယ်ကို ဖြည့်စွက်ထားပါတယ်။ အရင်က loop variable ကို 0 နဲ့ initialized လုပ်ခဲ့ပေမယ့် အခုတော့ 1 နဲ့ လုပ်ထားပါတယ်။ test expression မှာလဲ 9 မသုံးဘဲ 10 ကိုသုံးထားပါတယ်။ အဲဒီလို သုံးနိုင်ဖို့ relational operator ကို < အစား <= (less-than-or-equal-to operator) ကို ပြောင်းသုံးထားပါတယ်။ ရလဒ်က loop ကို ဆယ်ပတ် ပတ်တာပါ။ 0-9 အစား 1-10 ပြောင်းပေးလိုက်တာပါ။

နောက်တစ်ခုက loop body တွေမှာ statement တစ်ကြောင်းတည်း ရှိနေခဲ့မယ်ဆိုရင် တွန့်ကွင်းမခတ်ဘဲ ရေးလေ့ ရှိသလို ကွင်းခတ်ပြီးလည်း ရေးနိုင်ပါတယ်။ တွန့်ကွင်းကို မဖြစ်မနေ မလိုအပ်ပေမယ့် ပရိုဂရမ်မှာ အများစုကတော့ ဖတ်ရတာ ရှင်းလင်းလွယ်ကူအောင် loop body ရေးတိုင်း ထည့်ပေးတတ်ကြပါတယ်။

Blocks and Variable Visibility

အဲဒီလို statements တွေ အများကြီးကို တွန့်ကွင်းနဲ့ ခတ်ထားတာကို block of code လို့ ခေါ်ပါတယ်။ အရေးကြီးတဲ့ အချက်တစ်ခုက အဲဒီ block ထဲမှာ defined လုပ်ထားတဲ့ variable ဟာ block အပြင်က မမြင်နိုင်(ယူမသုံးနိုင်) တာပဲ ဖြစ်ပါတယ်။ cubelist.cpp မှာ ကျွန်တော်တို့ အနေနဲ့ variable cube ကို block အတွင်းမှာ define လုပ်ထားခဲ့ပါတယ်။

```
int cube = numb*numb*numb;
```

ဒါကြောင့် သူ့ကို block အပြင်ကနေ access မလုပ်နိုင်ပါဘူး။ ဒါကြောင့် တကယ်လို့ ကျွန်တော်တို့အနေနဲ့ cube = 10; ဆိုတဲ့ ကုဒ်လေးကို block အပြင်ကနေ ရေးသားခဲ့မယ်ဆိုရင် compiler ကနေ error message ပေးမှာဖြစ်ပါတယ်။ ဘာလို့လဲဆိုတော့ cube ဆိုတဲ့ variable ဟာ block အပြင်ဘက်မှာ undefined ဖြစ်နေလို့ပဲ ဖြစ်ပါတယ်။

အဲဒီလို ကန့်သတ်ချက် ရှိနေခြင်းရဲ့ အားသာချက်တစ်ခုကတော့ မတူညီတဲ့ block တွေ အတွင်းမှာ နာမည်တူပြီး တကယ်တမ်း မတူညီတဲ့ variable တွေ သုံးလို့ ရတာပဲ ဖြစ်ပါတယ်။ (ဒါပေမယ့် စာရေးသူကတော့ အဲဒီလို သုံးတာကို အားမပေးပါဘူး။ မှားသွားရင် သေချာပေါက် ရွာလည်သွားနိုင်ပါတယ်။ variable တွေကို နာမည် သီးသန့်စီ သေသေချာချာပေးတာဟာ မဖြစ်မနေ လုပ်သင့်ပါတယ်။)

Indentation and Loop Style

ကျွန်တော်တို့ ပရိုဂရမ်ရေးသားတဲ့ နေရာမှာ ပြန်လည် ဖတ်ရှုရလွယ်ကူပြီး ရှင်းလင်းတဲ့ ရေးဟန် ရှိဖို့ လိုအပ်ပါတယ်။ loop တွေကို ရေးသားတဲ့ နေရာမှာလဲ ဖတ်ရလွယ်ပြီး ရှင်းနေအောင် ညာဘက်ကို

indented လုပ်ပေးဖို့ လိုအပ်ပါတယ်။ ဆိုလိုတာက tab ခံပြီး ရေးသားဖို့ကို ပြောတာ ဖြစ်ပါတယ်။ ဒီလို ရေးသားခြင်းကြောင့် loop အစနဲ့ အဆုံးကို အလွယ်တကူ မြင်နိုင်မှာ ဖြစ်ပါတယ်။ ကွန်ပိုင်းလာ ကတော့ indent လုပ်၊ မလုပ် မသိပါဘူး။ ဒါပေမယ့် ကျွန်တော်တို့ ပရိုဂရမ်မာတွေ ဖတ်ရ၊ ပြင်ဆင်ရ လွယ်အောင်တော့ လုပ်ပေးသင့်ပါတယ်။

ကျွန်တော်တို့ တွန့်ကွင်းအစကို ရေးသားတဲ့ နေရာမှာ တွန့်ကွင်းအဆုံးနဲ့ တစ်တန်းတည်း ရေးသားလေ့ ရှိပါတယ်။ ဒါပေမယ့် အချို့ပရိုဂရမ်မာတွေကတော့ အောက်ပါပုံစံမျိုးနဲ့ loop statement အဆုံးမှာ တွန့်ကွင်း အစကို ရေးတတ်ကြပါတယ် -

```
for(numb=1; numb<=10; numb++) {
    cout << setw(4) << numb;
    int cube = numb*numb*numb;
    cout << setw(6) << cube << endl;
}
```

ဒီလိုရေးသားခြင်းအားဖြင့် ကုဒ်တစ်လိုင်း ပိုတိုသွားစေပါတယ်။ ဒါပေမယ့် တွန့်ကွင်းအစကို မျက်စိရှမ်းသွားတတ်ပြီး ဖတ်ရ ပိုခက်သွားစေပါတယ်။ ဖတ်ရရှင်းလင်း လွယ်ကူတဲ့ ရေးနည်းကတော့ တွန့်ကွင်း အဖွင့်နဲ့ အပိတ်ကို တစ်တန်းတည်းထားပြီး loop body ကိုတော့ indent ပြုလုပ်ထားတဲ့ style ပဲဖြစ်ပါတယ်။

```
for(numb=1; numb<=10; numb++)
{
    cout << setw(4) << numb;
    int cube = numb*numb*numb;
    cout << setw(6) << cube << endl;
}
```

ဘယ် style ပဲသုံးသုံး ဖတ်နေကျမဟုတ်တဲ့ သူအတွက်တော့ အခက်အခဲ အနည်းငယ် ဖြစ်နိုင်ပါတယ်။ ဒါကြောင့် style တစ်ခုတည်းကိုပဲ စွဲစွဲမြဲမြဲ အသုံးပြုကြဖို့ အကြံပြုလိုပါတယ်။

Debugging Animation

Loop operation တွေ ဘယ်လို လုပ်ဆောင်တယ်ဆိုတာကို အသေးစိတ် လေ့လာနိုင်ဖို့အတွက် compiler တွေမှာ ပါလေ့ရှိတဲ့ debugging features တွေကို အသုံးပြုသင့်ပါတယ်။ single-stepping ကိုသုံးပြီး ကုဒ်တစ်ကြောင်းချင်း ဘယ်လို အလုပ်လုပ်တယ်၊ variable တွေ ဘယ်လို

ပြောင်းလဲသွားတယ် ဆိုတာတွေကို လေ့လာနိုင်ပါတယ်။ အဲဒီအကြောင်းကို နောက်ပိုင်းမှာ အသေးစိတ် ရေးပါဦးမယ်။

for Loop Variations

ကျွန်တော်တို့ အနေနဲ့ increment expression ကို loop variable တွေကို increment လုပ်ဖို့သာမက အခြား သင့်တော်ရာ operation များကိုလည်း လုပ်ဆောင်ဖို့ အသုံးပြုနိုင်ပါတယ်။ နောက် ဥပမာ တစ်ခုမှာ ၎င်းကို loop variable တန်ဖိုး decrement လုပ်ဖို့ အသုံးပြုထားပါတယ်။ factor.cpp ဆိုတဲ့ ပရိုဂရမ်လေးမှာ user ထည့်ပေးလိုက်တဲ့ ဂဏန်း တစ်ခုရဲ့ factorial ကို ရှာပြထားပါတယ်။ (factorial ဆိုတာကတော့ မူလဂဏန်းကို သူ့ထက်ငယ်တဲ့ positive integers တွေ အားလုံးနဲ့ မြှောက်ထားတာပါ။ ဒါကြောင့် 5 ရဲ့ factorial ဟာ $5*4*3*2*1 = 210$ ဖြစ်ပါတယ်။)

```
// 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;
}
```

ဒီဥပမာလေးမှာ j တန်ဖိုးကို initialization expression သုံးပြီး user ထည့်ပေးတဲ့ တန်ဖိုးကို ပေးလိုက်ပါတယ်။ test expression ကတော့ j တန်ဖိုး 0 ထက် ကြီးနေသမျှ loop ပတ်နေစေပါတယ်။ Increment expression ကတော့ပ j တန်ဖိုးကို loop တစ်ကြိမ် ပတ်ပြီးတိုင်း တစ် လျှော့ပေးပါတယ်။ ဒီနေရာမှာ unsigned long ကို factorial တန်ဖိုး ထည့်သွင်းဖို့ အသုံးပြုထားတာ ဖြစ်ပါတယ်။ factorial

တန်ဖိုးတွေက ကြီးမားတတ်လို့ပဲ ဖြစ်ပါတယ်။ အောက်မှာ ပြထားတဲ့ factorial တန်ဖိုးတွေကို ကြည့်ခြင်းအားဖြင့် ခန့်မှန်းနိုင်ကြပါတယ်။

Enter a number: 10

Factorial is 3628800

ဒီ ပရိုဂရမ်မှာ ကျွန်တော်တို့ အကြီးဆုံး ရှာနိုင်တဲ့ factorial ကတော့ 12 အတွက်ပဲ ဖြစ်ပါတယ်။ အဲဒီထက် ကြီးတဲ့ တန်ဖိုးတွေကို ထည့်သွင်းခဲ့မယ် ဆိုရင်တော့ error message မပြဘဲ မှားယွင်းတဲ့ ရလဒ်တွေ ထွက်လာမှာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် variable တွေကို define လုပ်တဲ့ နေရာမှာ data type မှန်ကန်အောင် သတ်မှတ်ပေးနိုင်ဖို့ အရေးကြီးတာပဲ ဖြစ်ပါတယ်။

Variables Defined in for Statements

နောက်နည်းတစ်ခုကတော့ loop variable (ဒီဥပမာမှာ j)ကို loop statement ထဲမှာပဲ define ပြုလုပ်ပေးခြင်းပဲ ဖြစ်ပါတယ်။

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

အသုံးများသလို ကောင်းမွန်တဲ့ loop variable define ပြုလုပ်နည်းပဲ ဖြစ်ပါတယ်။ အဲဒီလို define ပြုလုပ်လိုက်တဲ့ variable ကို loop body ကပဲ မြင်နိုင် (အသုံးပြုနိုင်) ပါတယ်။ (ကွန်ပိုင်းလာ ပေါ်မူတည်ပြီး ခြွင်းချက်တော့ ရှိနိုင်ပါတယ်။ ဒါပေမယ့် Standard C++ မှာတော့ မမြင်နိုင်ပါဘူး။)

Multiple Initialization and Test Expressions

ကျွန်တော်တို့ အနေနဲ့ for statement ရေးသားတဲ့ နေရာမှာ initialization ကို expression တစ်ခုထက် ပိုပြီး ရေးလို့ ရပါတယ်။ comma နဲ့ ခြားပေးရပါတယ်။ အဲဒီလိုပဲ increment expression ကို လဲ ရေးလို့ရပါတယ်။ ဒါပေမယ့် test expression ကိုတော့ တစ်ခုပဲ ရေးသားလို့ ရပါတယ်။

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

```
{
```

```
    // body of loop
```

```
}
```

အထက်ပါ ဥပမာမှာတော့ normal loop variable j ကို သုံးထားပါတယ်။ နောက်ထပ် အခြား variable တစ်ခုဖြစ်တဲ့ alpha ကိုလည်း initialize လုပ်ပေးထားပါတယ်။ incremental expression မှာတော့ j ကို ပုံမှန် increment လုပ်ပေးထားပြီး တတိယ variable ဖြစ်တဲ့ beta ကိုတော့ decrement လုပ်ထားပါတယ်။

တကယ်တော့ ကျွန်တော်တို့ လိုအပ်လာရင် အဲဒီ expression အချို့ သို့မဟုတ် အားလုံးကို မရေးဘဲ ချန်လှပ်ထားခဲ့နိုင်ပါတယ်။ ဥပမာ for(;;) ဟာ while loop ကို test expression true ထည့်ပေးထားတာနဲ့ တူညီပါတယ်။ နောက်ပိုင်းမှာ while loop တွေ အကြောင်းကို ရေးသွားပေးသွားမှာပါ။ အဲဒီလို ရေးနည်းမျိုးကို အတတ်နိုင်ဆုံး ရှောင်ရမှ ဖြစ်ပါတယ်။ ဖတ်ရခက်ခဲ စေတာကြောင့်ရယ် ရလဒ် အတူတူရအောင် အခြားနည်းလမ်းတွေနဲ့ ပိုမိုရိုးရှင်းစွာ ရေးသားနိုင်သေးတာကြောင့်ပဲ ဖြစ်ပါတယ်။

The while Loop

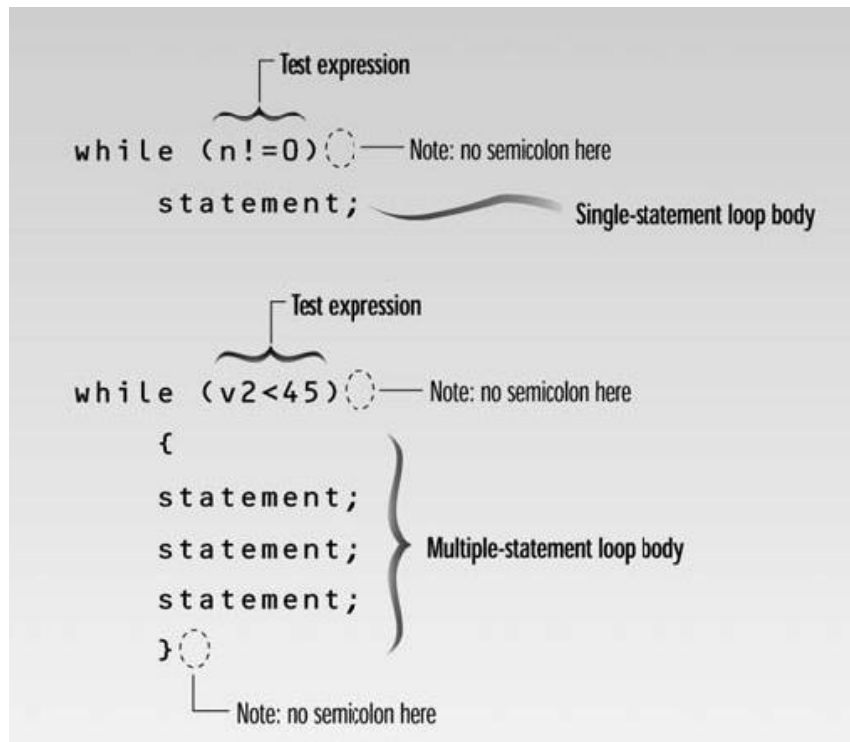
အထက်မှာ တိကျတဲ့ အကြိမ်အရေအတွက် အတိုင်း looping ပတ်ဖို့အတွက် for loop ကို အသုံးနည်းများကို လေ့လာခဲ့ပြီးပါပြီ။ ဒါပေမယ့် တကယ်လို့ ကျွန်တော်တို့ ပတ်မယ့် loop အရေအတွက်ကို ကြိုမသိနိုင်ခဲ့ဘူးဆိုရင် အခြားနည်းကို သုံးရပါလိမ့်မယ်။ အဲဒါကတော့ while loop ပဲ ဖြစ်ပါတယ်။ ဥပမာအနေနဲ့ endon0.cpp ပရိုဂရမ်ကို ရေးပြထားပါတယ်။ အဲဒီ မှာ user ကို ဂဏန်းတွေ တစ်ခုပြီး တစ်ခု တောင်းပါမယ်။ တကယ်လို့ user က 0 ကို ထည့်သွင်းလိုက်တဲ့ အခါမှသာ ပရိုဂရမ် ပြီးဆုံးသွားမှာ ဖြစ်ပါတယ်။ ဒီနေရာမှာ ပရိုဂရမ် အနေနဲ့ user က 0 မထည့်သွင်းခင် ဘယ်နှစ်ကြိမ် ဂဏန်းတွေ ထည့်ပေးမယ်ဆိုတာကို ကြိုမသိပါဘူး။ user စိတ်ကြိုက် ထည့်သွင်းနိုင်ပါလိမ့်မယ်။

```
// 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;
}
```

အောက်မှာတော့ နမူနာအနေနဲ့ ပရိုဂရမ်ထဲကို ဂဏန်းတွေ ထည့်သွင်းပြထားပါတယ်။ 0 ကို ထည့်သွင်းလိုက်တဲ့ အခါမှာတော့ loop ကနေထွက်ပြီး ပရိုဂရမ် ပြီးဆုံးသွားမှာ ဖြစ်ပါတယ်။

1
27
33
144
9
0

while loop ဟာ for loop ကို ပိုမိုရိုးရှင်းအောင် ပြုလုပ်ထားတာနဲ့ တူနေပါတယ်။ သူ့မှာ test expression သာပါရှိပြီး initialization နဲ့ increment expression တွေ မပါဝင်ပါဘူး။ အောက်က ပုံလေးမှာ while loop ရဲ့ syntax ကို ပြထားပါတယ်။

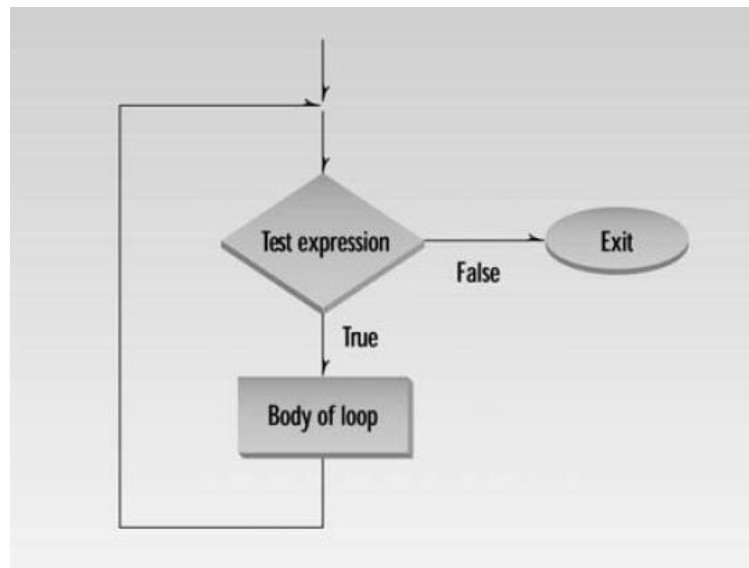


while loop မှာ test expression မှန်နေသမျှ ကာလပတ်လုံး loop ကို ပတ်နေမှာ ဖြစ်ပါတယ်။ endon0.cpp မှာပါရှိတဲ့ $n \neq 0$ (n not equal to 0) ဆိုတဲ့ test expression ကြောင့် n တန်ဖိုး 0 မဖြစ်မချင်း test result မှန်ကန်နေပြီး loop ကို ပတ်နေမှာ ဖြစ်ပါတယ်။ while loop မှာ initialization expression မပါပေမယ့် loop variable (ဒီဥပမာမှာတော့ n) ကို loop မစခင်မှာ initialize လုပ်ပေးထားဖို့ လိုအပ်မှာ ဖြစ်ပါတယ်။ နောက်တစ်ခုက loop body ထဲမှာ အဲဒီ loop variable ကို

ပြောင်းလဲပေးတဲ့ statement ပါဖို့လိုအပ်ပါတယ်။ မပါခဲ့ရင် loop က ထွက်ဖို့ မဖြစ်နိုင်တော့ပါဘူး။
endon0.cpp မှာတော့ `cin<<n;` ဆိုတဲ့ ကုဒ်က `n` တန်ဖိုးကို ပြောင်းလဲစေပါတယ်။

Multiple Statements in a while Loop

နောက်ထပ် နမူနာ ပရိုဂရမ် တစ်ခုကို ရေးပြပါဦးမယ်။ while4.cpp မှာ while loop အတွင်း multiple statements တွေ ရေးပြထားပါတယ်။ တကယ်တော့ ဒီလို multiple statements တွေ ရေးတဲ့ အကြောင်းကို cubelist.cpp မှာတုန်းက for loop ထဲမှာ ရေးပြခဲ့ပြီးပါပြီ။ ဒီပရိုဂရမ်မှာတော့ လေးထပ်ကိန်းတွေကို ရှာပြထားပါတယ်။ လေးထပ်ကိန်းရဲ့တန်ဖိုးတွေကို ဂဏန်းလေးလုံး အထိပဲ ရှာဖို့ ဆုံးဖြတ်ထားတယ် ဆိုကြပါစို့။ ဒါဆိုရင် အများဆုံး တန်ဖိုးက 9999 ထက် မကြီးရပါဘူး။ အဲဒီလို အခြေအနေမျိုးကို ကန့်သတ်ဖို့ အတွက်ဆိုရင် တွက်မကြည့်ရသေးပဲနဲ့ ဘယ်ဂဏန်းရဲ့လေးထပ်ကိန်းက အဲဒီ limit ကို ကျော်သွားမယ်ဆိုတာကို မသိနိုင်ပါဘူး။ ဒါကြောင့် test expression မှာ အဲဒီ အချက်ကို စစ်ဆေးခိုင်းပြီး limit ကျော်သွားတာနဲ့ loop ကို ရပ်လိုက်မှာ ဖြစ်ပါတယ်။



```
// 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;
}

```

လေးထပ်ကိန်းကို ရှာဖွေအတွက်ကတော့ အဲဒီဂဏန်းကို လေးခါ မြှောက်ပေးလိုက်တာပါ။ loop တစ်ခါပတ်တိုင်း numb ဆိုတဲ့ variable ကို 1 တိုးပေးသွားမှာပါ။ ဒါပေမယ့် test expression မှာတော့ numb ကို မစစ်ဘဲ လေးထပ်ကိန်းတန်ဖိုး pow ကိုပဲ စစ်ဆေးပြီး loop ကို ဘယ်အချိန်မှာ ထွက်ရမယ်ဆိုတာကို ဆုံးဖြတ်သွားမှာပါ။ အောက်မှာ မူရင်းဂဏန်းတွေနဲ့ သူတို့ရဲ့ လေးထပ်ကိန်းတွေကို တွက်ပြထားပါတယ်။ လေးထပ်ကိန်းတန်ဖိုး 9999 ထက်ကြီးသွားတာနဲ့ ပရိုဂရမ်ပြီးဆုံးသွားမှာ ဖြစ်ပါတယ်။

```

1 1
2 16
3 81
4 256
5 625
6 1296
7 2401
8 4096
9 6561

```

Precedence: Arithmetic and Relational Operators

နောက်ပရိုဂရမ် တစ်ပုဒ်ကတော့ Fibonacci series ကို တွက်ထုတ်ပေးမယ့် fibo.cpp ပဲ ဖြစ်ပါတယ်။ Fibonacci series ဆိုတာဟာ ရှေ့ဂဏန်း နှစ်လုံးပေါင်းခြင်းဖြင့် နောက်ဂဏန်း ကို ရှာယူထားတဲ့ ဂဏန်းတွေ ဖြစ်ပါတယ်။ ၎င်းရဲ့အစပိုင်း ဂဏန်းတစ်ချို့ကတော့ 1 1 2 3 5 8 13 21 34 55 တို့ပဲ ဖြစ်ပါတယ်။ တကယ်တော့ ဒီ series ကို လက်တွေ့ အသုံးချမှုပေါင်းများစွာ ရှိပါတယ်။ ၎င်းဟာ ဗိသုကာပညာနဲ့ အနုပညာမှာ သုံးလေ့ရှိတဲ့ golden ratio နဲ့လဲ ဆက်စပ်နေပါသေးတယ်။ ဒီနေရာမှာတော့ အဲဒီအကြောင်းတွေကို အကျယ်ရှင်းမပြတော့ပါဘူး။

```
// 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;
}
```

ရလဒ်တွေကတော့ အောက်ပါအတိုင်းပဲ ဖြစ်ပါတယ်။

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

အဲဒီရလဒ်တွေထဲက နောက်ဆုံး ဂဏန်းနှစ်လုံးကို အချိုးချလိုက်မယ်ဆိုရင် 0.618033988 ကို ရရှိမယ်ဖြစ်ပြီး ၎င်းဟာ golden ratio ပဲဖြစ်ပါတယ်။ fibo.cpp ပရိုဂရမ်မှာ အကြီးဆုံး positive integers တွေ ထည့်ထားနိုင်ဖို့ unsigned long data type ကို အသုံးပြုထားပါတယ်။

Test expression မှာ အဲဒီ data type ရဲ့ကန့်သတ်ချက်ကို ကျော်မသွားတဲ့ အချိန်အထိ loop ကို ပတ်နေအောင် ရေးသားထားပါတယ်။ limit တန်ဖိုးကိုတော့ const သတ်မှတ်ပေးထားပြီး မတော်တဆ ပြောင်းလဲရေးသားမိခြင်းကနေ ကာကွယ်ထားပါသည်။ loop ကို limit တန်ဖိုးရဲ့ တစ်ဝက်ရောက်လာရင် ထွက်ဖို့ ရေးသားရပါတယ်။ ဒါမှလဲ sum တန်ဖိုးက limit ကို ကျော်မသွားမှာ ဖြစ်ပါသည်။ test expression မှာ operators နှစ်ခု သုံးထားပါတယ်။ (next < limit / 2)။ ကျွန်တော်တို့ရဲ့ရည်ရွယ်ချက်က next ကို limit/2 နဲ့ နှိုင်းယှဉ်ဖို့ ဖြစ်ပါတယ်။ division ကို comparison မတိုင်မီ လုပ်ဆောင်ဖို့ လိုပါတယ်။ အဲဒီအချက်ကို သေချာစေဖို့ လက်သဲကွင်းထဲ ထည့်ရေးလို့ရပါတယ်။ (next < (limit/2))။ ဒါပေမယ့် လက်သဲကွင်းကို ထည့်ရေးဖို့ မလိုပါဘူး။ arithmetic operators တွေဟာ relational operators တွေထက် precedence ပိုမြင့်လို့ပဲ ဖြစ်ပါတယ်။

The do Loop

While loop ကို နားလည်သွားပြီဆိုရင် do loop ကို သဘောပေါက်ဖို့ မခက်တော့ပါဘူး။ while loop မှာတုန်းက test expression မှန်မှန် စစ်ဆေးခြင်းကို loop ရဲ့ အစမှာ ပြုလုပ်တာ ဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီအချက် မှားသွားတာနဲ့ looping တစ်ကြိမ်မှ မပတ်တော့ဘဲ loop ကို ကျော်သွားမှာ ဖြစ်ပါတယ်။ တစ်ခါတစ်လေမှာ test expression ကို မစစ်သေးဘဲ loop ကို အနည်းဆုံး တစ်ကြိမ်တော့ ပတ်စေချင်တဲ့ အခါမျိုးတွေ ရှိတတ်ပါတယ်။ အဲဒီအခြေအနေမျိုးမှာဆို do loop ကို while loop အစား အသုံးပြုရလေ့ ရှိပါတယ်။ ဒါကြောင့် do loop မှာ test expression ကို loop အဆုံးမှာ ထားထားတာဖြစ်ပြီး အနည်းဆုံး တစ်ခေါက် ပတ်ပြီးမှ test expression ကို စတင် စစ်ဆေးတာ ဖြစ်ပါတယ်။ အောက်မှာ ဖော်ပြထားတဲ့ divdo.cpp ပရိုဂရမ်လေးမှာတော့ တည်ကိန်းနဲ့ စားကိန်း နှစ်ခုကို user အား ထည့်သွင်းစေပြီး စားလဒ်နဲ့ အကြွင်းကို / နဲ့ % operator တွေသုံးကာ ရှာဖွေပေးမှာ ဖြစ်ပါတယ်။ do loop အသုံးပြုထားပုံလေးကို ဂရုစိုက်ပြီး လေ့လာစေချင်ပါတယ်။

```
// divdo.cpp
```

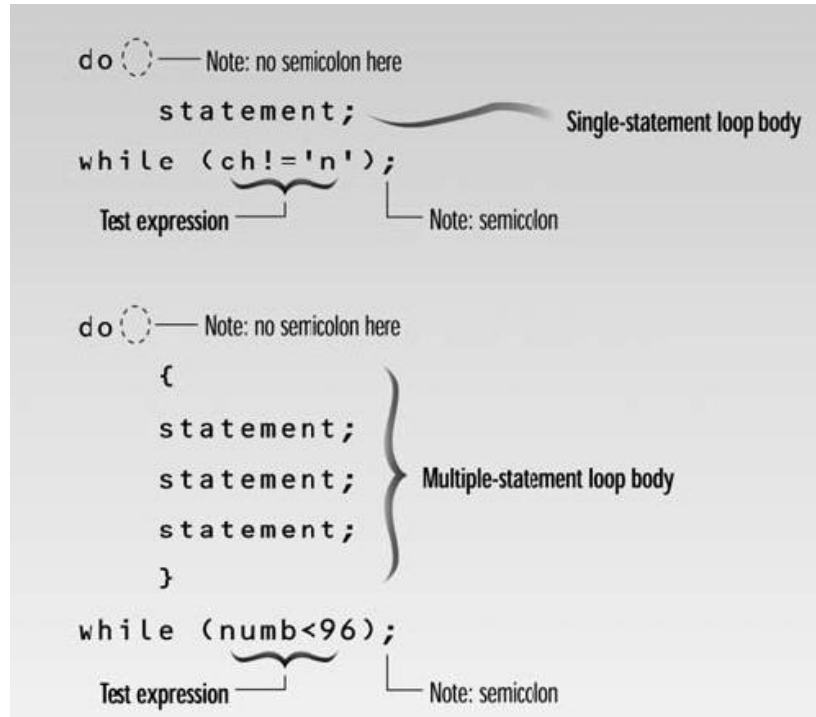
```
// demonstrates DO loop
```

```

#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;
}

```

ပရိုဂရမ်ရဲ့ ကုဒ်အများစုကို do loop ထဲမှာ ရေးသားထည့်သွင်းထားတာကို သတိထားမိပါလိမ့်မယ်။ do loop မှာ loop ရဲ့အစကို do ဆိုတဲ့ keyword နဲ့ သတ်မှတ်ပေးလိုက်တာ ဖြစ်ပါတယ်။ နောက်က တွန့်ကွင်း အဖွင့် အပိတ်ကြားမှာတော့ loop body ကို ရေးသားထားတာ ဖြစ်ပြီး အဆုံးသတ်မှာ test expression ထည့်သွင်းထားကာ semicolon နဲ့ ပိတ်ပေးရမှာ ဖြစ်ပါတယ်။ (semicolon ထည့်ဖို့ မကြာခဏ မေ့တတ်တာကို သတိပြုပါ)။ do loop ရဲ့ syntax ကို အောက်မှာ ပြထားပါတယ်။



divdo.cpp ပရိုဂရမ်လေးထဲမှာ တွက်ချက်မှုတွေ ပြီးတဲ့ နောက်မှာ user ကို နောက်ထပ် ဆက်တွက်ချင်သေးလားလို့ မေးပါတယ်။ တကယ်လို့ user က ဆက်တွက်ချင်သေးရင် y ကို ရိုက်ထည့်ရမှာ ဖြစ်ပြီး loop ကို အပြီးထွက်ချင်ရင်တော့ n ကို ရိုက်ထည့်ရမှာပါ။ တကယ်တမ်းက test expression မှာ `ch != 'n'` လို့ စစ်ထားတဲ့ အတွက် n မထည့်သမျှ true ဖြစ်နေပြီး loop ကို ပတ်နေမှာပါ။ n ထည့်လိုက်တော့မှ false ဖြစ်သွားတဲ့ အတွက် loop ထဲက ထွက်သွားမှာ ဖြစ်ပါတယ်။ အောက်မှာ နမူနာ စမ်းသပ်ပြထားပါတယ်။

Enter dividend: 11

Enter divisor: 3

Quotient is 3, remainder is 2

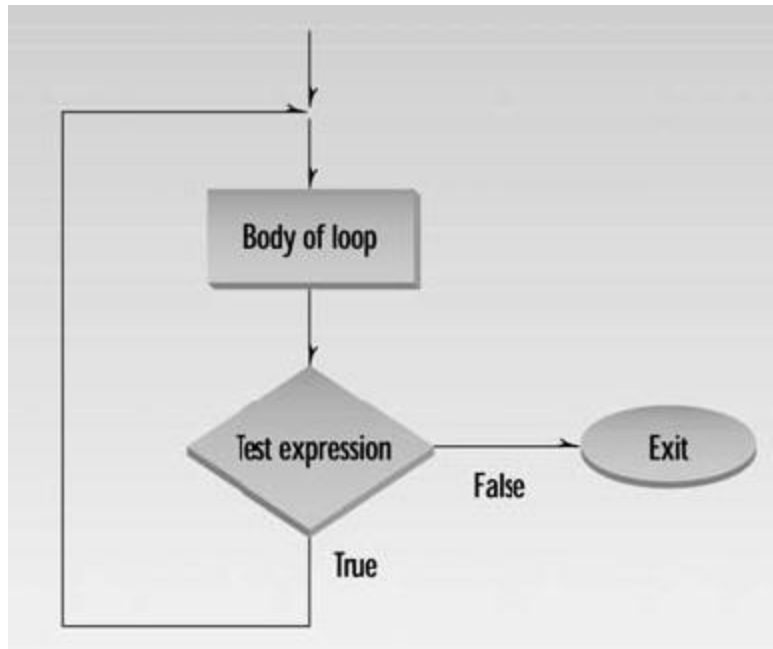
Do another? (y/n): y

Enter dividend: 222

Enter divisor: 17

Quotient is 13, remainder is 1

Do another? (y/n): n



When to Use Which Loop

ကျွန်တော်တို့အနေဖြင့် မည်သည့်နေရာတွင် မည်သည့် loop အမျိုးအစားကို အသုံးပြုသင့်သည် ဆိုသည့်အချက်အား အကြမ်းအားဖြင့် ခွဲခြားထားလေ့လာနိုင်ပါသည်။ ဥပမာအားဖြင့် loop ပတ်ရမည့် အကြိမ်အရေအတွက်ကို ကြိုတင်သိနေခဲ့ပါက for loop ကို အသုံးပြုသင့်ပါသည်။ while loop နှင့် do loop ကိုတော့ အကြိမ်အရေအတွက်ကို ကြိုမသိနိုင်ဘဲ အခြေအနေတစ်ခုခု (variable value တစ်ခုခု) ပေါ် မူတည်၍ loop မှ ထွက်ရမည့် အခါမျိုးတွင် test expression ဖြင့် စစ်ဆေးကာ အသုံးပြုရသည်။ အဆိုပါ အခြေအနေကို စစ်ချင်း စစ်ဆေးသင့်ပြီး မမှန်ကန်ပါက loop body အား လုံးလုံး ကျော်သွားရန် လိုအပ်သည့် အခြေအနေမျိုးတွင် while loop အား အသုံးပြုရသည်။ Menu ရေးသားခြင်းကဲ့သို့သော loop body အား အနည်းဆုံး တစ်ကြိမ် လုပ်ဆောင်ပြီးမှ လိုအပ်သည့် အခြေအနေအား စစ်ဆေးရမည့် အခြေအနေမျိုးတွင် do loop ကို အသုံးပြုသင့်သည်။ အထက်ပါ အချက်များသည် အကြမ်းဖျင်း သတ်မှတ်ချက်များသာ ဖြစ်ပြီး ရေးသားရာတွင် လွယ်ကူမြန်ဆန်စေရန် ရည်ရွယ်သည်။ သို့ရာတွင် အဆိုပါ loop များအားလုံးကို အခြေအနေ အားလုံးနီးပါးအတွက် အသုံးပြုနိုင်ပေသည်။ ကျွန်တော်တို့ အနေဖြင့် မိမိတို့ ရေးသားသော ပရိုဂရမ်အား ရှင်းလင်း ကျစ်လစ်စေရန် ဂရုပြု၍ loop များကို ရွေးချယ်သွားကြရမည် ဖြစ်သည်။

Decisions

loop များတွင် ပါဝင်သော decisions များသည် loop body အား ဆက်လက် လုပ်ဆောင်သင့် မသင့်ကို သတ်မှတ် စစ်ဆေးခြင်းပင် ဖြစ်သည်။ လက်တွေ့ဘဝတွင်လည်း ဒီနေ့ အပြင်ထွက်သင့် မသင့်၊ အင်္ကျီဘာအရောင် ဝယ်မလဲ? အလည်အပတ်သွားသင့်သလား? အစရှိသော ဆုံးဖြတ်စရာ ကိစ္စရပ်များ ကြုံတွေ့နေကြရသည်သာဖြစ်သည်။ ပရိုဂရမ်အတွင်း၌လည်း ထိုကဲ့သို့သော one-time decision များ ပြုလုပ်ရန် လိုအပ်ပေသည်။ test expression ၏ တန်ဖိုးပေါ် မူတည်၍ ဆုံးဖြတ်ချက်ချကာ ပရိုဂရမ်၏

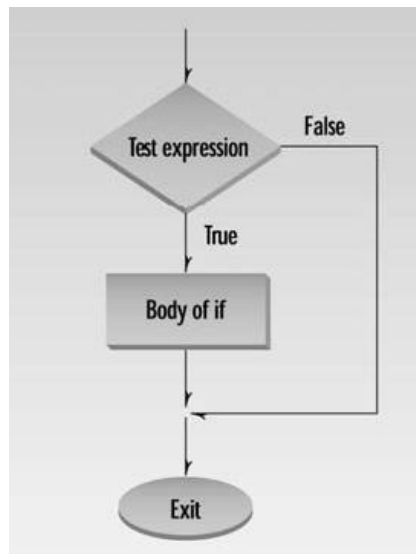
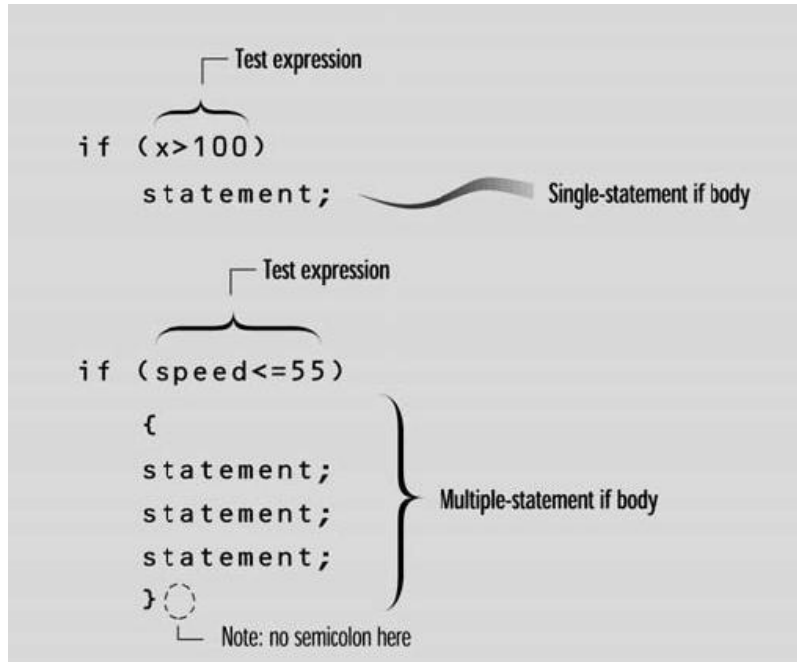
အခြားနေရာများသို့ ကုန်များ jump ပြုလုပ်ရမည် ဖြစ်သည်။ C++ တွင် decisions များကို နည်းလမ်း အမျိုးမျိုးဖြင့် ပြုလုပ်နိုင်သော်လည်း အရေးအကြီးဆုံးမှာ မတူညီသော အခြေအနေ နှစ်ခုထဲမှ တစ်ခုကို ရွေးချယ်ပေးနိုင်သော if...else statement ပင် ဖြစ်သည်။ ၎င်း statement ကို else မပါဘဲ if statement အဖြစ် ရိုးစင်းစွာ အသုံးပြုနိုင်သည်။ အကြောင်းအရာ အများအပြားမှ တစ်ခုကို ရွေးထုတ်ယူရန်အတွက်မူ switch statement ကို အသုံးပြုနိုင်သည်။ အချို့ထူးခြားသည့် အခြေအနေမျိုးတွင် conditional operator ကို အသုံးပြုနိုင်သည်။ ၎င်းတို့ အကြောင်းကို တစ်ဆင့်ချင်း လေ့လာကြရအောင်။

The if Statement

ဆုံးဖြတ်ချက်များ ပြုလုပ်ရာတွင် if statement သည် အရိုးရှင်းဆုံး ဖြစ်ပါသည်။ ၎င်းကို ifdemo.cpp တွင် အောက်ပါအတိုင်း လေ့လာနိုင်ပါသည်။

```
// 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 ရေးတဲ့ အခါမှာ if ဆိုတဲ့ keyword ရဲ့နောက်မှာ လက်သဲကွင်း အဖွင့်အပိတ်နဲ့ test expression ကို ရေးရပါတယ်။ အောက်ပါ if statement ရဲ့ syntax ကို ပုံမှာ ပြထားပါတယ်။ if statement ဟာ while နဲ့ အတော်လေးတူတာကို သတိထားမိပါလိမ့်မယ်။ ခြားနားချက်ကတော့ if statement က while လို looping မပတ်ဘဲ test expression မှန်ခဲ့ရင် တစ်ကြိမ်တည်း လုပ်ဆောင်တာ ဖြစ်ပါတယ်။ while loop မှာတော့ test expression မှန်နေသမျှ ကာလပတ်လုံး loop ကို ပတ်နေမှာပါ။ နောက်ပိုတစ်ခုမှာ if statement ရဲ့လုပ်ဆောင်ပုံကို flow chat နဲ့ ပြထားပါတယ်။



ifdemo.cpp ပရိုဂရမ်လေးရဲ့ ရလဒ်တွေကို အောက်မှာ ပြသထားပါတယ်။ ၁၀၀ ထက်ကြီးတဲ့ ဂဏန်းဆိုရင် ပရင့်ထုတ်ပေးနေမှာ ဖြစ်ပြီး ၁၀၀ နဲ့ တူသွားတာ သို့မဟုတ် ငယ်သွားခဲ့ရင်တော့ ဘာမှ ထုတ်ပေးမှာ မဟုတ်ပါဘူး။

Enter a number: 2000

That number is greater than 100

Multiple Statements in the if Body

Loop များကို လေ့လာစဉ်က single statement ကို တွန့်ကွင်း မခတ်ဘဲ ရေးလို့ ရသလို လိုအပ်ပါက ရှင်းလင်း လွယ်ကူစေရန် တွန့်ကွင်းအတွင်းရေးသင့်ကြောင်းနှင့် multiple statements များအတွက်မူ မဖြစ်မနေ တွန့်ကွင်းအတွင်း ရေးသားရမည် ဖြစ်ကြောင်း သိရှိခဲ့ပါတယ်။ ယခု if statement မှာလဲ ထိုနည်းတူ တွန့်ကွင်းအား အသုံးပြုရမည် ဖြစ်ပါတယ်။ if2.cpp မှ အောက်ပါအတိုင်း လေ့လာကြည့်နိုင်ပါတယ်။

```
// 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 တွေနှင့် decision တွေဟာ တစ်ခုနဲ့တစ်ခု nested လုပ်ပြီး ရေးနိုင်ကြပါတယ်။ loop ထဲမှာ decision တွေကို ငုံထားလို့ ရသလို decision တွေထဲမှာလည်း loop တွေ ထည့်သွင်း ထားနိုင်ပါတယ်။ ထိုနည်းတူ loop တွေထဲမှာ loop တွေထပ်ငုံလို့ရပြီး decision တွေထဲမှာလည်း decision တွေ ထပ်ငုံလို့ ရပါတယ်။ အခု သင်ခန်းစာမှာတော့ for loop အတွင်းမှာ if ကို ငုံထားတဲ့ prime.cp ဆိုတဲ့ ပရိုဂရမ်လေးကို လေ့လာကြည့်ကြရအောင်။ ဒီ ပရိုဂရမ်လေးက prime number တွေကို ရှာတဲ့ ပရိုဂရမ်လေးပါ။ prime number ဆိုတာ သူကိုယ်တိုင်ရယ် ၁ ရယ်ကလွဲရင် ကျန်တဲ့ ဂဏန်းနဲ့

စားလို့မပြတ်တဲ့ တနည်း သုဒ္ဓကိန်း ခွဲလို့ မရတဲ့ ကိန်းပဲ ဖြစ်ပါတယ်။ ဥပမာ -2, 3, 5, 7, 11, 13, 17 စတာတွေပဲ ဖြစ်ပါတယ်။

```
// 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;
}
```

ဒီဥပမာလေးမှာ user ထည့်ပေးလိုက်တဲ့ ဂဏန်းကို n ထဲ ထည့်လိုက်ပါတယ်။ အဲဒီနောက် n ကို for loop ကို အသုံးပြုပြီး 2 ကနေ $\frac{n}{2}$ အထိ ဂဏန်းတွေနဲ့ စားပစ်ပါတယ်။ for(j=2; j <= n/2; j++)။ စားကိန်းက loop variable j ဖြစ်ပါတယ်။ အဲဒီ တန်ဖိုးထဲက တစ်ခုခုက n ကိုပြတ်အောင် စားနိုင်ခဲ့ရင် (တနည်း အကြွင်း သုညဖြစ်ခဲ့ရင်) n က prime မဟုတ်တော့ပါဘူး။ အဲဒီလို ဆုံးဖြတ်တဲ့ အပိုင်းကို if နဲ့ % operator သုံးပြီး တွက်ချက် ဆုံးဖြတ်ပါတယ် if(n%j == 0)။ အကယ်၍ ထည့်ပေးလိုက်တဲ့ ဂဏန်းက prime number မဟုတ်ခဲ့ရင် user ကို ပရင့်ထုတ်ပြီး ပြန်အသိပေးမှာ ဖြစ်ပါတယ်။ စမ်းသပ်ပြထားပါတယ်-

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

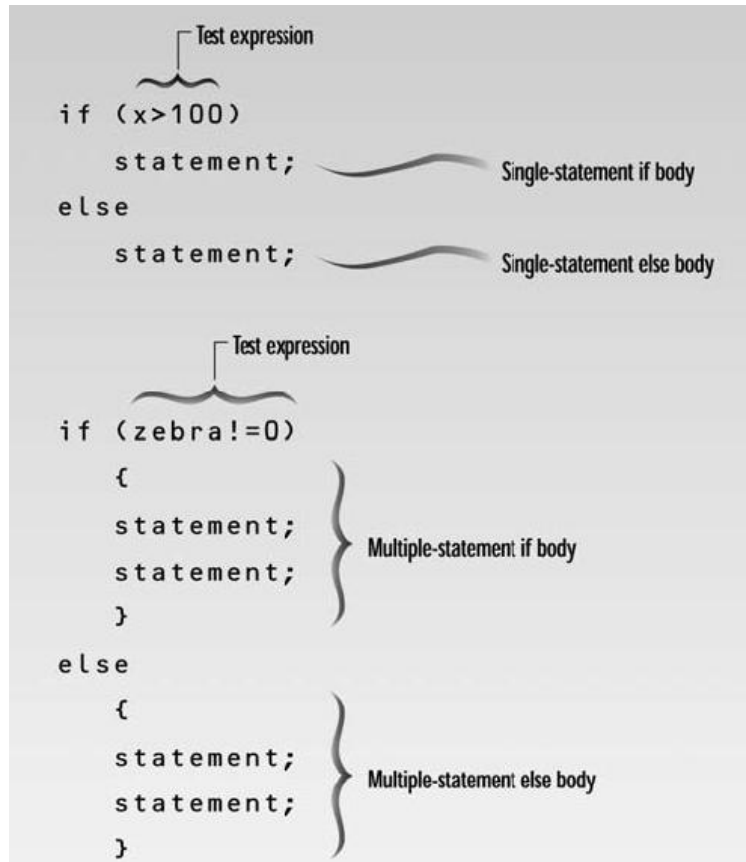
လောလောဆယ် ဥပမာမှာတော့ for loop ကို တွန့်ကွင်းနဲ့ မခတ်ပြထားပါဘူး။ ဒါကလဲ if statement ကို statement တစ်ခုအဖြစ်သာ ယူဆကြောင်း ပြသချင်လို့ပါ။ အရင်က ပြောခဲ့သလိုပဲ တွန့်ကွင်းတွေ ထည့်သွင်းရေးသားခြင်းက ဖတ်ရှုရလွယ်ကူစေပါတယ်။

Library Function exit()

Prime မဟုတ်မှန်းသိလို့ program ကနေ ချက်ခြင်း ထွက်ချင်တဲ့ အခါမှာ exit() ဆိုတဲ့ library function ကို အသုံးပြုသင့်ပါတယ်။ ဘယ်နေရာကနေပဲ ဖြစ်ဖြစ် ပရိုဂရမ်ကို ရပ်ပစ်ပြီး ထွက်သွားမှာ ဖြစ်ပါတယ်။ သူက return value ပြန်မပေးပါဘူး။ ကွင်းထဲမှာ ထည့်ပေးလိုက်ရတဲ့ 0 ကတော့ operating system ကို argument အနေနဲ့ ပို့ပေးလိုက်တာပါ။ သာမန်အားဖြင့် successful termination ဆိုတဲ့ သဘောကို ဆောင်ပါတယ်။ အခြားဂဏန်းတွေကတော့ error ရှိကြောင်း ပြောချင်တဲ့ အခါ အသုံးပြုရပါတယ်။

The if...else Statement

if statement ဟာ အခြေအနေတစ်ခု မှန်ကန်ရင် ဘာလုပ်မယ်ဆိုတာကို ရေးသားဖို့အတွက် အသုံးပြုရတာ ဖြစ်ပါတယ်။ ဒါပေမယ့် မှားသွားခဲ့ရင်တော့ ဘာတစ်ခုမှ လုပ်ပေးမှာ မဟုတ်ပါဘူး။ အချို့အခြေအနေတွေမှာ မှန်ရင် တစ်ခုခုလုပ်ပြီး မှားသွားရင်တော့ တခြားတစ်ခု လုပ်ဆောင်ဖို့ လိုအပ်တာတတ်ပါတယ်။ အဲဒီလို အခြေအနေမျိုးအတွက်တော့ if...else statement ကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။ အောက်မှာပြထားသလိုပါပဲ if statement နောက်မှာ statement တစ်ခု ဒါမှမဟုတ် တွန့်ကွင်းနဲ့ခတ်ထားတဲ့ multiple statements တွေ ရှိပါတယ်။ အဲဒီနောက်မှာ else ကို ရေးပြီး သူ့နောက်ကလဲ statement တစ်ခု ဒါမှမဟုတ် တွန့်ကွင်းနဲ့ခတ်ထားတဲ့ multiple statements တွေ ထည့်ရေးပေးရမှာ ဖြစ်ပါတယ်။ if နောက်က test expression မှန်ခဲ့မယ် ဆိုရင် အဲဒီနောက်က statement(s) တွေ အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ မှားခဲ့ရင်တော့ else နောက်က statement(s) တွေ အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။



အထက်က if example လေးကို if...else နဲ့ ပြန်ရေးပြထားပါတယ်။

```
// ifelse.cpp
```

```
// demonstrates IF...ELSE statement
```

```
#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;
```

```

}

```

ဒီဥပမာလေးမှာ user ထည့်သွင်းပေးလိုက်တဲ့ x တန်ဖိုးဟာ 100 ထက်ကြီးခဲ့ရင် cout<< "That number is greater than 100\n"; ဆိုတာကို လုပ်ဆောင်မှာ ဖြစ်ပြီး မှားခဲ့ရင်တော့ cout << "That number is not greater than 100\n"; ဆိုတာကို လုပ်ဆောင်ပေးမှာပါ။ နမူနာအနေနဲ့ 300 နဲ့ 3 တန်ဖိုးနှစ်ခု ထည့်သွင်းပေးလိုက်လို့ ရလာတဲ့ ရလဒ်တွေကို အောက်မှာ လေ့လာ ကြည့်နိုင်ပါတယ်။

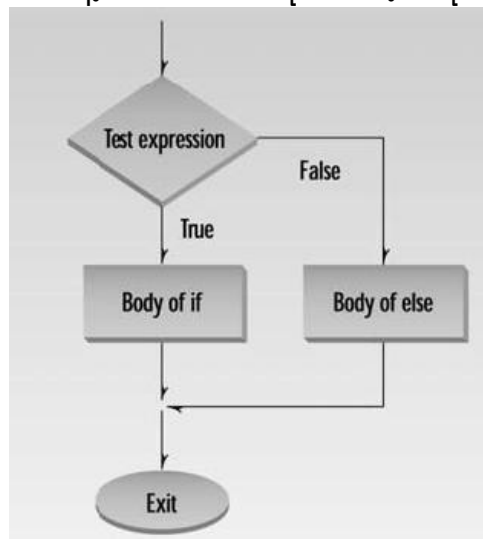
Enter a number: 300

That number is greater than 100

Enter a number: 3

That number is not greater than 100

if...else အလုပ်လုပ်ပုံကို flow chat နဲ့ အောက်ပါအတိုင်း လေ့လာနိုင်ပါတယ်။



The getch() Library Function

နောက်ထပ်ဥပမာ တစ်ခုအနေနဲ့ if...else ကို while loop ထဲမှာ ထည့်သွင်း အသုံးပြုပြထားပါတယ်။ နောက်တစ်ခုက getch() ဆိုတဲ့ library function အသစ်တစ်ခုကို သုံးပြုထားပါတယ်။ အဲဒီ chcount.cpp ပရိုဂရမ်လေးဟာ user ထည့်သွင်းပေးလိုက်တဲ့ စာလုံးတွေနဲ့ အက္ခရာ အရေအတွက်တွေကို ရေတွက်ပေးမှာပါ။

```
// chcount.cpp
```

```
// counts characters and words typed in
```

```
#include <iostream>
```

```
using namespace std;
```

```
#include <conio.h> //for getch()
```



```

int main()
{
    int chcount=0; //counts non-space characters
    int wdcoun=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 = getch(); //read one character
        if( ch==' ' ) //if it's a space
            wdcoun++; //count a word
        else //otherwise,
            chcount++; //count a character
    } //display results
    cout << "\nWords=" << wdcoun << endl
    << "Letters=" << (chcount-1) << endl;
    return 0;
}

```

input ထည့်သွင်းဖို့အတွက် cin နဲ့ >> ကို အသုံးပြုလေ့ ရှိပါတယ်။ ဒီနည်းကို သုံးမယ်ဆိုရင် user က input တစ်ခု ထည့်သွင်းပေးလိုက်တိုင်း Enter ခေါက်ရမှာဖြစ်ပါတယ်။ ဒါပေမယ့် အခုလို ဥပမာ မျိုးမှာ user က Enter မနှိပ်ရဘဲ အကွာရာတိုင်းကို ဖတ်သွားဖို့ လိုလာပါပြီ။ ဒါကြောင့် getch() ဆိုတဲ့ library function ကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။ ၎င်းကို အသုံးပြုဖို့ conio.h header file ကို ကြေငြာပေးရမှာ ဖြစ်ပါတယ်။ input argument ထည့်ပေးဖို့ မလိုအပ်ဘဲ user ရိုက်ထည့်လိုက်သမျှ အကွာရာတွေကို ချက်ချင်း return ပြန်ပေးမှာ ဖြစ်ပါတယ်။ chcount.cpp ထဲမှာတော့ getch() နဲ့ ဖတ်လို့ ရလာတဲ့ တန်ဖိုးတွေကို ch ထဲ ထည့်ပေးလိုက်မှာဖြစ်ပါတယ်။

နောက်တစ်ခုက getch() function ဟာ user ရိုက်ထည့်လိုက်သမျှ အကွာရာတွေကို ကွန်ပျူတာမျက်နှာပြင်ပေါ်ကို echo လုပ်ပေးမှာ ဖြစ်ပါတယ်။ (_getch() function ကတော့ echo ပြန်လုပ်ပေးမှာ မဟုတ်ပါဘူး။ echo ပြန်လုပ်ပေးလို့လဲ function name ရဲ့နောက်ဆုံး စကားလုံးမှာ e ထည့်ထားပြီး getch ဖြစ်နေတာပါ။ if...else statement ကတော့ စကားလုံး(word) ရေတွက်ဖို့ သုံးထားတဲ့ wdcoun ကို space ဆိုတဲ့ character တွေတာနဲ့ တစ်တိုးပေးမှာ ဖြစ်ပါတယ်။ အကွာရာ (character) တွေ ရေတွက်ဖို့ သုံးတဲ့ chcount ကတော့ space ကလွဲရှိ ဘယ်အကွာရာကိုပဲ ရိုက်ရိုက် တစ်တိုးပေးမှာ ဖြစ်ပါတယ်။ ဆိုလိုတာက space ကလွဲလို့ ကျန်တဲ့ အကွာရာတွေကို character လို့ သတ်မှတ်လိုက်တာပါ။ (ဒီလို စစ်ဆေးတဲ့ နည်းမှာ အားနည်းချက်တွေ ရှိပါတယ်။ ဒါပေမယ့်

ဒီနေရာမှာတော့ ဒီဥပမာလေးဟာ လုံလောက်တယ်လို့ ယူဆပါတယ်။ အောက်မှာ နမူနာထည့်သွင်းပေးလိုက်တဲ့ စာကြောင်းလေးကို စကားလုံးနဲ့ အက္ခရာတွေ ရေတွက်ပုံ လေ့လာကြည့်ကြရအောင်။

For while and do

Words=4

Letters=13

while statement မှာပါတဲ့ test expression ကတော့ ch ရဲ့တန်ဖိုးဟာ '\r' ဖြစ်မဖြစ် စစ်ဆေးပါတယ်။ တကယ်တော့ အဲဒီ အက္ခရာဟာ ကီးဘုတ်က Enter key ကို ကိုယ်စားပြုထားတာပါ။ တကယ်လို့ Enter ခေါက်ခဲ့ရင် while loop ကနေ ထွက်သွားမှာ ဖြစ်ပါတယ်။

Assignment Expressions

chcount.cpp ကို assignment expressions နဲ့ precedence တွေရဲ့ အရေးပါပုံနဲ့ ကုန်လိုင်းအချို့ကို ချုံ့ပစ်နိုင်တာကိုပြသဖို့ အနည်းငယ် ပြင်ရေးပြချင်ပါတယ်။ အဲဒီလို ပြင်ရေးလိုက်ခြင်းအားဖြင့် ကုန်တွေကို ဖတ်ရတာ တစ်မျိုး ဖြစ်နေနိုင်ပေမယ့် C နဲ့ C++ မှာတော့ ရေးရိုးရေးစဉ် ရေးနည်းတစ်ခုပဲ ဖြစ်ပါတယ်။ ပြင်ရေးထားတဲ့ chcnt2.cpp ကိုလေ့လာကြည့်ကြရအောင်။

```
// chcnt2.cpp
```

```
// counts characters and words typed in
```

```
#include <iostream>
```

```
using namespace std;
```

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

```
int main()
```

```
{
```

```
    int chcount=0;
```

```
    int wdcnt=1; // space between two words
```

```
    char ch;
```

```
    while( (ch=getche()) != '\r' ) // loop until Enter typed
```

```
    {
```

```
        if( ch==' ' ) // if it's a space
```

```
            wdcnt++; // count a word
```

```
        else // otherwise,
```

```
            chcount++; // count a character
```

```

    } // display results
    cout << "\nWords=" << wdcount << endl
    << "Letters=" << chcount << endl;
    return 0;
}

```

getche() ကနေ return ပြန်ပေးလိုက်တဲ့ တန်ဖိုးကို ch ထဲ ထည့်ပေးတဲ့ ကုဒ်ကို အပြင်မှာ သပ်သပ် မရေးသားတော့ဘဲ while loop ရဲ့ test expression ထဲမှာ တစ်ခါတည်း ထည့်သွင်း ရေးသားလိုက်ခြင်းပဲ ဖြစ်ပါတယ်။ နောက်ထပ်ပြီး အဲဒီ တန်ဖိုးကို '\r' ဟုတ်မဟုတ် စစ်ဆေးပေးပါတယ်။

တကယ်တော့ (ch=getche()) ဆိုတဲ့ ကုဒ် တစ်ခုလုံးက ch ထဲမှာ ရှိတဲ့ တန်ဖိုးကို ကိုယ်စားပြုနေတာ ဖြစ်တဲ့အတွက် တစ်ခါတည်း မှန်မမှန် စစ်ဆေးလို့ ရသွားတာပဲ ဖြစ်ပါတယ်။ C++ မှာ x = y = z = 0; ဆိုပြီး ရေးသားနိုင်ပါတယ်။ အဲဒီလို ရေးနည်းမှာ ညာဘက်အစွန်ဆုံး assignment က အရင် စအလုပ်လုပ်ပြီး 0 ကို z ထဲ၊ z တန်ဖိုးကို y ထဲ၊ y တန်ဖိုးကို x ထဲထည့်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် တန်ဖိုးတွေအားလုံးက 0 ဖြစ်သွားပါလိမ့်မယ်။ (ch=getche()) ဆိုတဲ့ assignment expression ကို လက်သဲကွင်းနဲ့ ခတ်ထားတာကို သတိထားမိပါလိမ့်မယ်။ ဘာလို့လဲဆိုတော့ assignment operator ဟာ relational operator ထက် precedence နိမ့်တဲ့အတွက် အရင်အလုပ်လုပ်စေချင်တဲ့အခါ ကွင်းခတ်ပေးလိုက်ရတာပဲ ဖြစ်ပါတယ်။ တကယ်လို့ လက်သဲကွင်းသာ မပါခဲ့ဘူးဆိုရင် while(ch = (getche() != '\r')) လို့ လုပ်ဆောင်သွားမှာ ဖြစ်ပြီး ကျွန်တော်တို့ မမျှော်လင့်ထားတဲ့ အမှားတွေကို ရင်ဆိုင်ရမှာပါ။

အဲဒီ chcnt2.cpp ထဲမှာပါတဲ့ while statement ဟာ နေရာအနည်းငယ်ပဲ ယူပေမယ့် အသုံးတည့်ပါတယ်။ တကယ်တော့ ၎င်းဟာ test expression တစ်ခုသာမကဘဲ keyboard က ရိုက်ထည့်လိုက်တဲ့ အက္ခရာ တစ်လုံးကို ch ထဲကိုလည်း ထည့်ပေးနိုင်ခဲ့ပါတယ်။ ဒါပေမယ့် ဒီလိုကုဒ်မျိုးကို ပထမဆုံး မြင်ဘူးကာစမှာတော့ အဓိပ္ပါယ်ဖော်ဖို့ ခက်ခဲနိုင်ပါတယ်။

Nested if...else Statements

```

-----
|. . . .|      #####
|. . . .|      #          #
|. $. . +#####          #
|. . . .|      #          ---+---
-----        #          |. . . .|
                #          |. ! . . .|
                #          |. . . .|
                #          |. @ . .|
                #          |. . . .|
-----        #          |. . . .|
|. .|          #####+. . D. .|
|<. +###      #          |. . . .|
---- #        #          |. ? . . .|
      #####          -----

```

- Wall
 # Unlit hallway
 . Lit area
 \$ Some quantity of gold
 + A door
 | Wall
 ! A magic potion
 @ The adventurer
 D A red dragon
 < Stairs to a higher level
 ? A magic scroll

ပုံ(၁၃-၁) Text-based game Roguelike

ကျွန်တော်တို့ ငယ်ငယ်က Microsoft Windows တွေ မပေါ်သေးပါဘူး။ MS-DOS system ကိုပဲ သုံးကြရပါတယ်။ အဲဒီခေတ်တုန်းက character-mode မှာ ကစားခဲ့ရတဲ့ text-based adventure စွန့်စားခန်း ဂိမ်းလေးတွေကို အမှတ်ရမိပါသေးတယ်။ စိတ်ကူးယဉ် ကမ္ဘာတစ်ခုထဲမှာ ကျွန်တော်တို့ character လေးတစ်လုံးကို အရပ်လေးမျက်နှာ လှည့်ပတ်သွားစေရင်းနဲ့ ရဲတိုက်တွေ၊ မှော်ဆရာတွေ၊ ရတနာတွေ အစရှိသဖြင့် ရှာဖွေရတာမျိုးပါ။ ဒါပေမယ့် အဲဒီဂိမ်းတွေထဲမှာ ရုပ်ပုံတွေ မပါသေးပါဘူး။ အင်္ဂလိပ် အက္ခရာလေးတွေ ကိုပဲ ပုံ(၁၃-၁)မှာ ပြထားသလိုမျိုး input နဲ့ output အတွက် သုံးခဲ့ကြရတာပါ။ ငယ်ရွယ်သေးတဲ့ စာဖတ်သူများ အဖို့တော့ အဲဒီလို ကွန်ပျူတာ ခေတ်ဦးက ဂိမ်းလေးတွေကို တွေ့ဖူးဖို့ မလွယ်တော့ပါဘူး။ ဒီခေတ်မှာတော့ 3D graphics အကောင်းစားတွေနဲ့ ဂိမ်းအလန်းစားတွေ က အစားထိုး နေရာယူ ထားလိုက်ကြပြီလေ။ ဒီသင်ခန်းစာမှာတော့ Nested if...else statement အကြောင်းကို ရှင်းပြဖို့အတွက် အဲဒီလို စွန့်စားခန်း ဂိမ်းလေးတစ်ခုရဲ့ အခြေခံ အစိတ်အပိုင်းတစ်ခုကို တုပ ရေးသားပြထားပါတယ်။ adifelse.cpp ကို အောက်ပါအတိုင်း လေ့လာကြည့်ကြပါစို့-

```

// adifelse.cpp
// demonstrates IF...ELSE with adventure program
#include <iostream>
using namespace std;
#include <conio.h> //for getch()

```

```

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 = getch(); //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

```

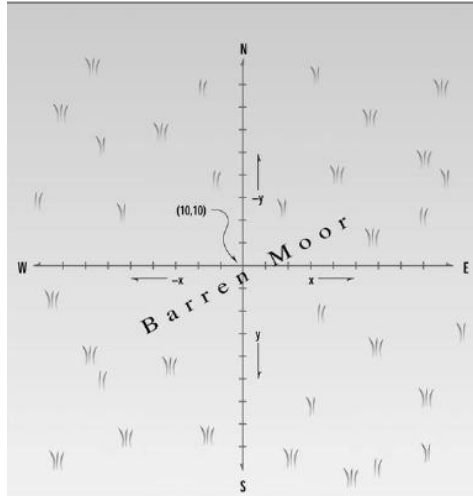
```
// demonstrates IF...ELSE with adventure program
#include <iostream>
using namespace std;
#include <conio.h> //for getch()
int main()
{
    char dir='a';
    int x=10, y=10;
    cout << "Type Enter to quit\n";
    while( dir != '\r' ) //until Enter is typed
    {
        cout << "\nYour location is " << x << ", " << y;
        cout << "\nPress direction key (n, s, e, w): ";

        dir = getch(); //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
```

The terminal window shows the program's execution. It starts with "Type Enter to quit". After pressing Enter, it displays "Your location is 10, 10" and prompts for a direction key. The user enters 'n', and the location changes to 10, 9. This process continues with 's', 'e', 'w', and 'n' keys, showing the location changing to 11, 9, 11, 8, 10, 9, 11, 7, 11, 6, 11, 5, 11, 4, 11, 3, and finally 10, 3 after pressing 'w'. The program ends with "Press direction key (n, s, e, w): " and a blank line.

ပုံ(၁၃-၁) Nested if...else အသုံးပြုပုံကို ရေးသားစမ်းသပ်ပြထားပုံ

ဒီဂိမ်းရဲ့ အစမှာ ကျွန်တော်တို့အနေနဲ့ လွင်တီးခေါင် တစ်ခုကို ရောက်နေသလို ခံစားရမှာ ဖြစ်ပါတယ်။ တစ်ကြိမ်မှာ အရပ်လေးမျက်နှာထဲက ကိုယ်ကြိုက်တဲ့ အရပ်ကို တစ်ယူနစ် သွားလို့ ရမှာ ဖြစ်ပါတယ်။ ပရိုဂရမ်ကတော့ ကျွန်တော်တို့ ရောက်နေတဲ့ နေရာကို x,y coordinate နဲ့ ပြပေးနေမှာပါ။ စတင်ချင်း တည်နေရာကတော့ (10,10) ဖြစ်ပါတယ်။ ဒီပရိုဂရမ်လေးမှာတော့ ကျွန်တော်တို့ ဘယ်ကိုပဲ သွားသွား ဘာမှ တွေ့ရဦးမှာ မဟုတ်ပါဘူး။ တည်နေရာရဲ့ x,y တန်ဖိုးတွေပဲ ပြောင်းလဲ သွားမှာပါ။ ကျွန်တော်တို့ အနေနဲ့ ကန့်သတ်ချက် ပေးမထားတဲ့အတွက် ကြိုက်တဲ့ အရပ်မျက်နှာကို ကြိုက်သလောက် သွားခွင့် ရှိပါတယ်။ အထက်က ပုံ (၁၃-၁) မှာ user ထည့်ပေးလိုက်တဲ့ အရပ်မျက်နှာကို မူတည်ပြီး xy coordinate တွေ ပြောင်းလဲသွားတာကို ပြထားပါတယ်။ ပရိုဂရမ်ကို အဆုံးသတ်ဖို့ Enter key ကို နှိပ်လိုက်ရမှာ ဖြစ်ပါတယ်။ ဒီပရိုဂရမ်လေးက တကယ်တော့ game တစ်ခုရဲ့ အစိတ်အပိုင်း သေးသေးလေး တစ်ခုပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် multiple branches တွေကို ကိုင်တွယ်ဖို့ နည်းလမ်းတစ်ခုကို လေ့လာနိုင်ပါတယ်။ if...else statement တစ်ခုထဲမှာ if...else statement တစ်ခု၊ အဲဒီထဲမှာ နောက်ထပ် if...else statement တစ်ခု နဲ့ နောက်ဆုံးမှာ if statement တစ်ခုကို အဆင့်ဆင့် ထည့်သွင်းရေးသားထားတာ ဖြစ်ပါတယ်။ ဒါကြောင့် ပထမဆုံး test condition ဟာ မှားသွားခဲ့မယ်ဆိုရင် ဒုတိယ၊ တတိယ စသဖြင့် လေးဆင့် စလုံး အဆင့်ဆင့် စစ်ဆေး သွားမှာ ဖြစ်ပါတယ်။ တစ်ခုခု မှန်သွားခဲ့မယ်ဆိုရင်တော့ သက်ဆိုင်ရာ x-y တန်ဖိုးများကို ပြောင်းလဲပေးတဲ့ ကုဒ်များကို လုပ်ဆောင်ပေးမှာပါ။ အဲဒီလို nested group လုပ်ထားတဲ့ if...else statements တွေကို *decision tree* လို့ ခေါ်ဆိုကြပါတယ်။



Matching the else

Nested if...else ကို သုံးတဲ့နေရာမှာ မှားတတ်တဲ့ နေရာလေး တစ်ခု ရှိပါတယ်။ ဒါကတော့ else ကို မှားယွင်းပြီး မသက်ဆိုင်တဲ့ if နဲ့ တွဲသုံးမိတာပဲ ဖြစ်ပါတယ်။ အောက်က badelse.cpp ကို လေ့လာကြည့်ရအောင် -

// 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;
```

```
}
```

အထက်ပါ ပရိုဂရမ်လေးမှာ cin ကိုသုံးပြီး a,b နဲ့ c တန်ဖိုးတွေကို ဖတ်ယူပါတယ်။ user က တန်ဖိုး သုံးခုကို ထည့်ပေးပြီး Enter ခေါက်ပေးရမှာပါ။ တကယ်လို့ ကျွန်တော်တို့က 2,3 နဲ့ 3 ကို ထည့်ပေးလိုက်တယ် ဆိုကြပါစို့။ ဒါဆို a က 2 b က 3 နဲ့ c က 3 အသီးသီး ဖြစ်သွားပါပြီ။ a နဲ့ b ဟာ မတူတဲ့ အတွက် ပထမ if ရဲ့ test expression ဟာ မှားသွားပါတယ်။ ဒါကြောင့် ကျွန်တော်တို့ အနေနဲ့ else ကို ခေါ်ယူပြီး cout << "a and b are different\n"; ဆိုတဲ့ ကုဒ်ကို လုပ်ဆောင်ဖို့ မျှော်လင့်ထားမိမှာပါ။ ဒါပေမယ့် လက်တွေ့မှာ ဘာမှ print မထုတ်ပေးပါဘူး။ ဘာလို့လဲဆိုတော့ အဲဒီ else ကို မသက်ဆိုင်တဲ့ if နဲ့ မှားပြီး တွဲထားခဲ့မိလို့ပါပဲ။ ရုတ်တရက် ကြည့်လိုက်ရင် else က ပထမ if နဲ့ တွဲထားတယ်လို့ ထင်စရာ ဖြစ်နေပါတယ်။ တကယ်က indentation နေရာချထားမှုအရ အဲလို အထင်မှားစေတာပါ။ တကယ်က အဲဒီ else ဟာ ဒုတိယ if နဲ့ တွဲနေတာပါ။ ဥပဒေသ ကတော့ else ဟာ နောက်ဆုံး if မှာ သာ else မရှိခဲ့ရင် ၎င်းနဲ့ တွဲနေတာ ဖြစ်ပါတယ်။ အောက်မှာ indentation နဲ့ print လုပ်မယ့် စာသား ပြင်ထားတဲ့ ပုံစံ အမှန်ကို ရေးပြထားပါတယ်။

```
if(a==b)
    if(b==c)
        cout << "a, b, and c are the same\n";
    else
        cout << "b and c are different\n";
```

ဒါကြောင့် 2,3,3 ကို ထည့်ပေးလိုက်မယ် ဆိုရင် ဘာမှ print မထုတ်ပေးပေမယ့် 2,2,3 ကို ထည့်ပေးလိုက်ရင်တော့ b and c are different ဆိုတာကို print ထုတ်ပေးမှာ ဖြစ်ပါတယ်။ ကျွန်တော်တို့ အနေနဲ့ else ကို ပထမဆုံး if နဲ့ မဖြစ်မနေ တွဲပေးချင်ရင်တော့ ဒုတိယ if ကို အောက်ပါအတိုင်း တွန့်ကွင်းထဲ ထည့်ပေးလိုက်ရမှာ ဖြစ်ပါတယ်။

```
if(a==b)
{
    if(b==c)
        cout << "a, b, and c are the same";
}
else
    cout << "a and b are different";
```

ဒီနည်းနဲ့ else ဟာ ပထမ if နဲ့ တွဲသွားပါတယ်။ တွန့်ကွင်း ခတ်ခံထားရတဲ့ if ကို အပြင်က else က မမြင်နိုင်လို့ပဲ ဖြစ်ပါတယ်။

The else...if Construction

if...else တွေကို nested လုပ်တဲ့အခါမှာ နားလည်ရ ခက်လေ့ရှိပါတယ်။ အထူးသဖြင့် ရှုပ်ရှုပ်ထွေးထွေး အဆင့်အများကြီး nested လုပ်ခဲ့မိရင် ဖတ်တဲ့သူအနေနဲ့ ဘာသာပြန်ဖို့ ခက်လာပါတယ်။ ဒီပြဿနာကို ဖြေရှင်းဖို့ နောက်တစ်နည်း သုံးနိုင်ပါတယ်။ အောက်ပါ adelseif.cpp ပရိုဂရမ်လေးကို လေ့လာကြည့်ကြရအောင် -

```
// adelseif.cpp
// demonstrates ELSE...IF with adventure program
#include <iostream>
using namespace std;
#include <conio.h> //for getch()
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 = getch(); //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
```

တကယ်တော့ အထက်ပါ ကုဒ်လေးတွေရဲ့ ရလဒ်ဟာ adifelse.cpp နဲ့ အတူတူပါပဲ။ if တွေကို else တွေ နောက်ကို နေရာရွှေ့ပေးလိုက်တာပဲ ကွာပါတယ်။ ရုတ်တရက် ကြည့်လိုက်ရင် else if ဆိုတဲ့ keyword အသစ်တစ်ခု သုံးထားတယ်လို့တောင် ထင်ရပါတယ်။ ဒီလိုရေးသားနည်းဟာ if...else ထက် ဖတ်ရတာရော ရေးသားရတာပါ သိသိသာသာ လွယ်ကူစေပါတယ်။ ဒါကြောင့် decision tree ပုံစံမျိုး ရေးသားစရာ ကြုံလာမယ်ဆိုရင် if...else ပုံစံထက် else if ကို အသုံးပြုသင့်ကြောင်း အကြံပြုလိုပါတယ်။

The switch Statement

ကျွန်တော်တို့အနေနဲ့ decision tree အကြီးစားတစ်ခုကို ဖြေရှင်းဖို့ လိုအပ်လာတဲ့ အခါမျိုးမှာ နောက်ပြီး အဲဒီလို ဆုံးဖြတ်ဖို့ကို variable တစ်ခုတည်းရဲ့ တန်ဖိုးပေါ် မူတည်နေမယ် ဆိုရင်တော့ if...else ဒါမှမဟုတ် else if တွေ အဆင့်ဆင့် ရေးမယ့် အစား switch statement ကို အသုံးပြုသင့်ပါတယ်။

```
// 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;
}

```



ဒီပရိုဂရမ်လေးဟာ တကယ်တော့ ဓာတ်ပြားဟောင်းကြီးများကို ဖွင့်တဲ့ ဓာတ်စက်တွေရဲ့ လည်ပတ်မှု အမြန်နှုန်းကို ရွေးချယ်ဖို့ ရေးထားတာ ဖြစ်ပါတယ်။ user ထည့်ပေးလိုက်နိုင်တဲ့ 33, 45, နဲ့ 78 ဆိုတဲ့ ဖြစ်နိုင်ခြေသုံးခုအတွက် သက်ဆိုင်ရာ messages တွေကို print ထုတ်ပေးမှာ ဖြစ်ပါတယ်။ အဲဒီခေတ်က ဓာတ်ပြားတွေမှာ long-playing records (LPs) တွေမှာ သီချင်းတွေ အများကြီး ပါဝင်ပြီး 33rpm (တစ်မိနစ်ကို ၃၃ပတ်နှုန်း) နဲ့ လည်ပါတယ်။ သီချင်းတစ်ပုဒ်စီသာပါဝင်တဲ့ single song တွေကိုတော့ 45rpm နဲ့ လည်ပါတယ်။ 78 ကတော့ သူတို့ LPs နဲ့ single တို့ထက် ရှေးကျတဲ့ စနစ်ဟောင်း လည်ပတ်နှုန်းပဲ ဖြစ်ပါတယ်။

switch ဆိုတဲ့ keyword နောက်က လက်သဲကွင်း အဖွင့်အပိတ်ထဲမှာ variable ကို ထည့်ပေးရမှာ ဖြစ်ပါတယ် (ဥပမာ- **switch(speed)**)။ တွန့်ကွင်း အဖွင့်အပိတ်ထဲမှာတော့ case statements တွေကို ရေးသားရမှာပါ။ case တစ်ခုစီရဲ့ နောက်မှာ constant တစ်ခု ပါဝင်ရမှာ ဖြစ်ပါတယ်။ အဲဒီ constant ကို ကွင်းမခတ်ထားဘဲ နောက်က colon(:) နဲ့ ပိတ်ပေးရမှာပါ (ဥပမာ - **case 33:**) ။ case constant မှာ အသုံးပြုထားတဲ့ constants တွေဟာ switch variable နဲ့ အမျိုးအစားချင်း တူညီဖို့တော့ လိုအပ်မှာ ဖြစ်ပါတယ်။ အောက်ကပုံမှာ switch statement ရဲ့ syntax ကို ပြသထားပါတယ်။

```

      Integer or character variable
switch (n) {
      Integer or character constant
case 1:
    statement;
    statement; } First case body
    break;      causes exit from switch
case 2:
    statement;
    statement; } Second case body
    break;
case 3:
    statement;
    statement; } Third case body
    break;
default:
    statement;
    statement; } Default body
}
      Note: no semicolon here

```

switch မတိုင်ခင်မှာ switch variable ကို တန်ဖိုးတစ်ခု သတ်မှတ်ထားပေးဖို့ လိုပါတယ်။ အဲဒီတန်ဖိုးကို case statement တွေမှာ ရှိတဲ့ constant တန်ဖိုးတွေနဲ့ တူရာကို ရှာရမှာပါ။ ရည်ရွယ်ထားတဲ့ case ကိုတွေ့ပြီဆိုရင် အဲဒီနောက်က statements တွေကို break မတွေ့မချင်း လုပ်ဆောင်ပေးမှာ ဖြစ်ပါတယ်။ အောက်မှာ platter.cpp ကို run ပြထားပါတယ်။ လေ့လာကြည့်ကြရအောင် -

Enter 33, 45, or 78: 45

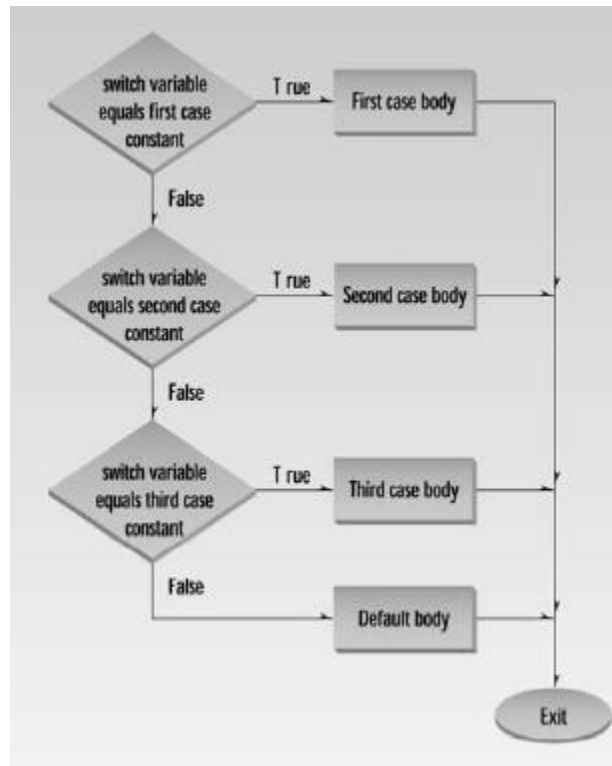
Single selection

The break Statement

platters.cpp ရဲ့ case section အားလုံးရဲ့အဆုံးမှာ break statement ရေးသားထားတာကို သတိထားမိမှာပါ။ တကယ်တော့ break keyword ဟာ switch statement တစ်ခုလုံးကနေ ထွက်သွားစေမှာပါ။ ပရိုဂရမ်ရဲ့ control ဟာ break ကိုတွေ့ရင် switch ရဲ့အပြင်ဘက်ကို ထွက်သွားပြီး သူ့နောက်က statement ကို ဆက်လက် လုပ်ဆောင်မှာ ဖြစ်ပါတယ်။

break ကိုရေးဖို့ မမေ့သင့်ပါဘူး။ break မပါခဲ့ရင် နောက်ထပ် case တွေကို ဆက်လက် စစ်ဆေးနေဦးမှာပါ။ break statement ကို loop တွေထဲက ထွက်ချင်တဲ့ အခါမျိုးမှာလဲ အသုံးပြုနိုင်ပါတယ်။ နောက်ပိုင်းကြိုလာတဲ့ အချိန်မှာ ဆက်လက် ရှင်းပြသွားပါမယ်။

တကယ်လို့ switch variable ဟာ case constants တွေနဲ့ တစ်ခုမှ မတူညီခဲ့ဘူးဆိုရင် ဘာမှ မလုပ်ဆောင်ဘဲ switch ထဲက ထွက်သွားမှာ ဖြစ်ပါတယ်။



switch Statement with Character Variables

platters.cpp မှာ အသုံးပြုခဲ့တဲ့ switch variable ဟာ int data type ဖြစ်ပါတယ်။ ကျွန်တော်တို့အနေနဲ့ char data type ကိုလည်း အသုံးပြုနိုင်ပါသေးတယ်။ ယခင် သင်ခန်းစာက adelseif.cpp ကို ပြန်ပြင်ရေးထားတဲ့ adswitch.cpp ကို လေ့လာကြည့်ကြရအောင် -

```
// adswitch.cpp
// demonstrates SWITCH with adventure program
#include <iostream>
using namespace std;
#include <conio.h> //for getch()
```

```

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

```

ဒီပရိုဂရမ်ထဲမှာ switch variable အနေနဲ့ dir ဆိုတဲ့ character variable ကို အသုံးပြုထားတာ ဖြစ်ပါတယ်။ char constant တွေ အနေနဲ့ကတော့ 'n', 's', 'e', 'w' နဲ့ '\r' တို့ကို အသုံးပြုထားပါတယ်။ ဒီနေရာမှာ '\r' က Enter ကို ကိုယ်စားပြုထားတဲ့ character ဖြစ်ပါတယ်။ (တစ်ခု မှတ်သားထားဖို့က ကျွန်တော်တို့အနေနဲ့ integer နဲ့ character variable တွေကို switch variable အနေနဲ့ အသုံးပြုလို့ ရပေမယ့် floating-point number တွေကိုတော့ အသုံးမပြုနိုင်ဘူးဆိုတာပါပဲ)။

ဒီပရိုဂရမ်မှာ ကုဒ်တွေက တိုတောင်းတာ ဖြစ်တဲ့အတွက် case နောက်က statements တွေကို တစ်ကြောင်းတည်းမှာပဲ တန်းစီ ရေးလိုက်ခြင်းဖြင့် ပိုမို ကျစ်လစ်သွားစေပါတယ်။ '\r' ကို သုံးပြီး Enter ခေါက်ရင် exit message ပေါ်လာအောင် ရေးသားထားတာကို လေ့လာနိုင်ပါတယ်။

The default Keyword

adswitch.cpp မှာ switch ရဲ့ case တွေ အဆုံးမှာ default ဆိုတဲ့ keyword ပါနေတာကို သတိထားမိမှာပါ။ ၎င်းဟာ switch variable နဲ့ case constant တွေ တစ်ခုမှ မကိုက်ညီတဲ့ အခါမျိုးမှာ တစ်ခုခု လုပ်ဆောင်နိုင်ဖို့ ထည့်ထားခြင်း ဖြစ်ပါတယ်။ ဒီ ဥပမာမှာတော့ user က အခြား character တစ်ခုခုကို ထည့်သွင်းခဲ့ရင် Try again! ဆိုတာကို print ထုတ်ပေးခြင်းအားဖြင့် ပြင်ဆင် ထည့်သွင်းခွင့် ပေးထားပါတယ်။ default ဟာ switch ရဲ့ နောက်ဆုံး statement ဖြစ်တာကြောင့် break ကို ထည့်ပေးဖို့ မလိုတော့ပါဘူး။

switch statement ကို user ထည့်ပေးလိုက်တဲ့ input ကို စမ်းစစ်ဖို့ ယေဘုယျ သုံးတတ်ကြပါတယ်။ အခြေအနေအများကြီးထဲက တစ်ခုကို ရွေးခိုင်းတာပဲ ဖြစ်ပါတယ်။ DOS ခေတ်တွေတုန်းက menu ရေးဖို့ do-while နဲ့ switch ကို တွဲသုံးလေ့ ရှိခဲ့ပါတယ်။ default statement ကိုတော့ ကျွန်တော်တို့ အနေနဲ့ မလိုအပ်ဘူးလို့ ထင်နေရင်တောင်မှ မဖြစ်မနေ ထည့်ထားသင့်ပါတယ်။ default: cout << "Error: incorrect input to switch"; break; ဆိုတဲ့ ကုဒ်တစ်ကြောင်းက ပရိုဂရမ်မာ သို့မဟုတ် user ကို input မှားယွင်း ထည့်ပေးနေပြီ ဒါမှမဟုတ် တစ်ခုခုတော့ အမှားအယွင်း ရှိနေပြီ ဆိုတာကို သတိပေးနိုင်ပါတယ်။ ပရိုဂရမ် အသေးတွေမှာ သိပ်မသိသာပေမယ့် အရေးကြီးတဲ့ ပရိုဂရမ်တိုင်းမှာ အမှားတွေကို တတ်နိုင်သမျှ ကုဒ်တွေနဲ့ ထောင်ချောက်ဆင်ပြီး ဖော်ထုတ်တတ်ဖို့ လိုအပ်ပါတယ်။

switch Versus if...else

ကျွန်တော်တို့ အနေနဲ့ switch နဲ့ if...else (ဒါမှမဟုတ် else if) တွေကို ဘယ်နေရာမှာ ဘာကို သုံးရမယ်ဆိုတာကို တိတိကျကျ ဆုံးဖြတ်ရ ခက်နေတတ်ပါတယ်။ တကယ်တော့ else if ကို variable အမျိုးမျိုးနဲ့ ရှုပ်ရှုပ်ထွေးထွေး သုံးထားတဲ့ အခါမျိုးမှာ မဖြစ်မနေ သုံးသင့်ပါတယ်။ ဥပမာ -

```
if( SteamPressure*Factor > 56 )
// statements
else if( VoltageIn + VoltageOut < 23000)
// statements
else if( day==Thursday )
```

```
// statements
```

```
else
```

```
// statements
```

switch statement မှာတော့ variable တစ်ခုတည်းကိုပဲ စစ်ဆေးပြီး ဆုံးဖြတ်တာ ဖြစ်ပါတယ်။ switch မှာတော့ case a<3: // do something break; ဆိုပြီး ရေးလို့ မရပါဘူး။ 'a', 3 စတဲ့ character ဒါမှမဟုတ် integer constant နဲ့ constant တန်ဖိုး ထွက်လာမယ့် 'a'+3 လို expression မျိုးကိုပဲ သုံးရမှာ ဖြစ်ပါတယ်။

အဲဒီလို အခြေအနေမျိုးနဲ့ ကိုက်ညီတဲ့အချိန်မှာ switch statement ဟာ ရေးသားရတာ အင်မတန် ရှင်းလင်း၊ လွယ်ကူပြီး ဖတ်ရသူကိုလည်း နားလည်ရ လွယ်စေပါတယ်။ ဒါကြောင့် decision tree များလာတာနဲ့အမျှ switch ကို အတတ်နိုင်ဆုံး အသုံးပြုစေချင်ပါတယ် ခင်ဗျာ။

The Conditional Operator

Conditional Operator ဟာ အနည်းငယ် ထူးဆန်းတဲ့ decision operator တစ်ခုဖြစ်ပါတယ်။ ပရိုဂရမ် ရေးသားရာမှာ တွေ့ကြုံရလေ့ ရှိတဲ့ - အခြေအနေ တစ်ခု မှန်ရင် variable တစ်ခုထဲ တန်ဖိုး တစ်ခု သတ်မှတ်ပေးပြီး မှားသွားခဲ့ရင် အခြားတန်ဖိုးတစ်ခု သတ်မှတ်ပေးရလေ့ ရှိတဲ့ အခါမျိုးမှာ အသုံးပြုဖို့အတွက် ဖန်တီးထားတာပါ။ ဥပမာအားဖြင့် ငယ်တဲ့ တန်ဖိုးရှာတဲ့ ပြဿနာမျိုးပါ။ အယ်ဖာ နဲ့ ဘီတာမှာ အယ်ဖာက ငယ်ရင် min ထဲကို အယ်ဖာ တန်ဖိုး ထည့်ပေးပြီး မှားခဲ့ရင် (ဘီတာက ငယ်ခဲ့ရင်) min ထဲကို ဘီတာ တန်ဖိုး ထည့်ပေးတဲ့ အခြေအနေမျိုး ဖြစ်ပါတယ်။ အဲဒီ အခြေအနေကို if...else သုံးပြီး အောက်ပါအတိုင်း ရေးလို့ ရပါတယ်။

```
if( alpha < beta )
```

```
    min = alpha;
```

```
else
```

```
    min = beta;
```

ဒီလို အခြေအနေမျိုးကို မကြာခဏ တွေ့ကြုံရတာကြောင့် C နဲ့ C++ ကို တီထွင်ခဲ့သူက ပိုမိုတိုတောင်းတဲ့ ရေးသားနည်းကို တီထွင်ရင်း conditional operator ပေါ်ပေါက်လာရခြင်း ဖြစ်ပါတယ်။ ဒီ operator မှာ သင်္ကေတ နှစ်ခု ပါပြီး operand သုံးခု ပါဝင်ပါတယ်။ C++ မှာရှိတဲ့ အခြား operator များ အားလုံးဟာ operand တစ်ခု သို့မဟုတ် နှစ်ခုကို လုပ်ဆောင်ပေးပါတယ်။ Conditional operator ကတော့ တစ်ခုတည်းသော operand သုံးခုကို လုပ်ဆောင်ပေးတဲ့ operator ဖြစ်ပါတယ်။

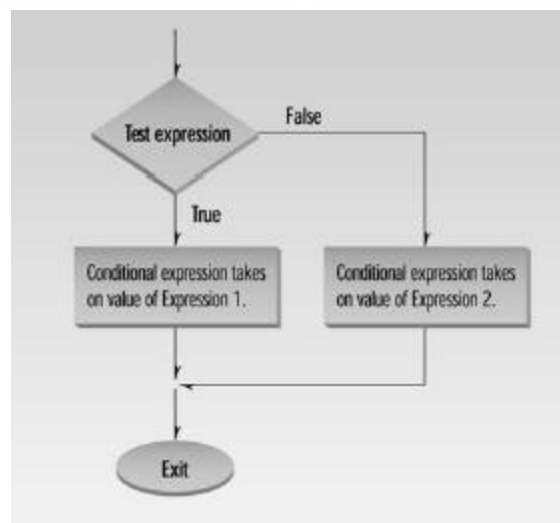
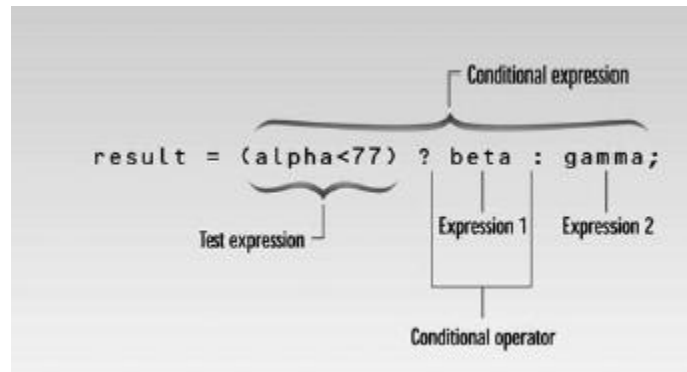
အထက်က ကုဒ်ကို conditional operator သုံးပြီး ရေးမယ်ဆိုရင် အောက်ပါအတိုင်း ရပါတယ်-


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

အဲဒီ statement ထဲက equal sign ညာဘက်က အခြမ်းဖြစ်တဲ့ (alpha < beta) ? alpha : beta ဆိုတာ conditional expression ဝဲ ဖြစ်ပါတယ်။ အဲဒီထဲမှာ ပါတဲ့ question mark (?) နဲ့ colon (:) တို့ပေါင်းပြီး conditional operator ဖြစ်လာတာပါ။

Question mark ရှေ့က (alpha < beta) ဆိုတာကတော့ test expression ဝဲဖြစ်ပါတယ်။ အဲဒီ တစ်ခုအပါအဝင် alpha နဲ့ beta သုံးခုလုံးဟာ operand တွေ ဖြစ်ကြပါတယ်။ တကယ်လို့ test expression မှန်ခဲ့မယ်ဆိုရင် conditional expression တစ်ခုလုံးရဲ့ တန်ဖိုးဟာ question mark နောက်က operand ဖြစ်သွားမှာပါ။ ဒီဥပမာမှာတော့ alpha ဖြစ်ပါတယ်။ Test expression မှားသွားခဲ့ရင်တော့ colon နောက်က operand တန်ဖိုးဖြစ်သွားမှာပါ။ ဒီဥပမာမှာ beta ဖြစ်ပါတယ်။

Test expression မှာပါတဲ့ လက်သဲကွင်း အဖွင့်အပိတ်ကို မထည့်လဲရပါတယ်။ ဒါပေမယ့် အစဉ်အလာ ရေးခဲ့ကြတာ ဖြစ်ပြီး ဖတ်ရှုရ ပိုမိုလွယ်ကူစေပါတယ်။ အောက်က ပုံတွေမှာ conditional operator ရဲ့ syntax နဲ့ flow chat ကို ပြထားပါတယ်။



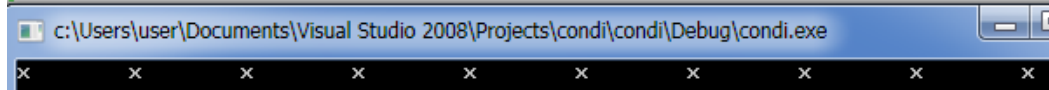
Conditional expression က ရလာတဲ့ တန်ဖိုးကို variable တစ်ခုထဲ ထည့်သွင်းဖို့ပဲ ဖြစ်ဖြစ် ဒါမှမဟုတ် value တစ်ခု အသုံးပြုနိုင်တဲ့ နေရာတိုင်း သုံးနိုင်ပါတယ်။ အထက်ပါ ဥပမာမှာတော့ min ဆိုတဲ့ variable ထဲကို ထည့်သွင်းပေးလိုက်တာပါ။ နောက် ဥပမာ တစ်ခုအနေနဲ့ conditional operator ကိုသုံးပြီး variable n ရဲ့ absolute value ကို ရှာတဲ့ ကုဒ်လေး ရေးပြပါမယ်။ (ဂဏန်း တစ်ခုရဲ့ Absolute value ဆိုတာက အဲဒီ ဂဏန်းရဲ့ အနုတ်လက္ခဏာကို ဖယ်ရှားထားတာပါ။ ဒါကြောင့် အမြဲတမ်း positive ဖြစ်နေမှာပါ)

absvalue = n<0 ? -n : n;

တကယ်လို့ n တန်ဖိုးက သုညထက် ငယ်ခဲ့ရင် ? နောက်က -n ကို လုပ်ဆောင်ပြီး အနုတ်နှစ်ခု တွေ့တဲ့ အတွက် အပေါင်းတန်ဖိုး ဖြစ်စေပါမယ်။ တကယ်လို့ မငယ်ခဲ့ရင် အပေါင်းတန်ဖိုး ဖြစ်နေပြီးသားဖြစ်လို့ ပြောင်းလဲမှု မရှိဘဲ n ကိုပဲ assign လုပ်ပေးမှာ ဖြစ်ပါတယ်။

```
// 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;
}
```

```
// 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;
}
```



အထက်ပါပုံမှာ space ၇ ခုပြီးတိုင်း x တစ်ခု ပရင့်ထုတ်ပြသည့် ရလဒ်ပင် ဖြစ်သည်။ condi.cpp တွင် j တန်ဖိုးကို 0 မှ 79 အထိ တစ်ခုချင်း for loop ဖြင့် တိုးယူလာရင်း conditional operator ကို လုပ်ဆောင်ပါသည်။ ၎င်းတွင် ပါဝင်သော test expression (j % 8) က j တန်ဖိုးသည် ၈ ၏ ဆတိုးကိန်းများ မဖြစ်မချင်း (၈ နှင့်စား၍ မပြတ်မချင်း) non zero (1-7) ဖြစ်နေသောကြောင့် true ဖြစ်ကာ ? နောက်က space ' ' ကို ပရင့်ထုတ်ပေးနေမည် ဖြစ်သည်။ ၈ နှင့် စား၍ ပြတ်သော တန်ဖိုးများတွင် အကြွင်းမှာ သုည ဖြစ်နေသဖြင့် test expression မှာ false ဖြစ်သွားပြီး x ကို ပရင့်ထုတ်ပေးခြင်းဖြင့် အထက်ပါ ရလဒ် ထွက်လာရသည်။ အထက်ပါ ကုဒ်ကို တိုတောင်းလှပြီ မထင်ပါနှင့်ဦး။ ch ကိုဖျောက်၍ အောက်ပါအတိုင်း တစ်ကြောင်းတည်း ပေါင်းရေးနိုင်ပါသေးသည်။

cout << (j%8) ? ' ' : 'x');

C နဲ့ C++ ပရိုဂရမ်မာ အတော်များများ ဒီလို ကျစ်လစ်တဲ့ ကုဒ်မျိုးကို နှစ်နှစ်မြိုက်မြိုက်ရေးလေ့ ရှိပါတယ်။ ကုဒ်တိုတိုလေးနဲ့ ထိထိရောက်ရောက် ရေးနိုင်တာ ပျော်စရာကောင်းပါတယ်။ ဒါပေမယ့် မလိုအပ်တဲ့အခါမျိုးမှာ အဲလို ကုဒ်မျိုးတွေကို မဖြစ်မနေ အချိန်ကုန်ခံနေရတိုင်းလဲ အကျိုးမရှိ ဖြစ်တတ်ပါတယ်။ ဒါကြောင့် if...else သုံးပြီး ကုဒ်နဲ့နဲ့ ပိုရေးမလား ဒါမှမဟုတ် conditional operator ကို သုံးမလားဆိုတာကတော့ ရရှိတဲ့ အချိန်အပေါ် မူတည်မယ်လို့ ထင်မြင်မိပါတယ်။

Logical Operators

ရှေ့သင်ခန်းစာများမှာ (Conditional Operator မပါ) operator အုပ်စု နှစ်ခုကို လေ့လာခဲ့ကြပြီးပါပြီ။ ပထမတစ်ခုကတော့ arithmetic operators (+, -, *, /, နဲ့ %) များဖြစ်ကြပြီး ဒုတိယ တစ်ခုကတော့ relational operators (<, >, <=, >=, ==, နဲ့ !=) များပဲ ဖြစ်ပါတယ်။

အခု ဆက်လက်လေ့လာကြရမှာကတော့ တတိယအုပ်စုဖြစ်တဲ့ *logical operators* များပဲ ဖြစ်ပါတယ်။ အဲဒီ operator များဟာ (true နဲ့ false တန်ဖိုး နှစ်ခုဖြစ်နိုင်တဲ့ variables) Boolean variables တွေကို logically ပေါင်းစပ်ပေးနိုင်စွမ်း ရှိပါတယ်။ ဥပမာအားဖြင့် *today is a weekday* ဟာ မှားလဲ မှားနိုင်သလဲ မှန်လဲ မှန်နိုင်တဲ့ အတွက် Boolean value ရှိပါတယ်။ နောက်ထပ် Boolean expression တစ်ခုက *Maria took the car* ဖြစ်ပါတယ်။ အဲဒီ expression နှစ်ခုကို logically ပေါင်းစပ်လို့ ရပါတယ်။ *If today is a weekday, and Maria took the car, then I'll have to take the bus.* ဒီနေရာမှာ သုံးသွားတဲ့ logical connection ကတော့ **and** ပဲ ဖြစ်ပါတယ်။ စာကြောင်းနှစ်ခုကို ဆက်စပ်ပြီးမှ true or false value ထွက်လာအောင် ဆောင်ရွက်ပေးလို့ပဲ ဖြစ်ပါတယ်။ အချက်နှစ်ခုလုံးမှန်မှ ဘတ်စ်ကားနဲ့ သွားရမှာ ဖြစ်ပါတယ်။

Logical AND Operator

ကျွန်တော်တို့ အနေနဲ့ Logical AND Operator ကို အသုံးပြုပြီး *adswitch.cpp* ကို ပိုပြီး စိတ်ဝင်စားစရာ ဖြစ်အောင် ရေးသားကြည့်ကြရအောင်။ ယခုရေးမယ့် *advenand.cpp* မှာ ရတနာတွေကို (7,11) ဆိုတဲ့ coordinates မှာ ဝှက်ထားပြီး ကစားသူက ရှာတွေ့မတွေ့ ကြည့်ကြရအောင်။

```
// advenand.cpp
```

```
// demonstrates AND logical operator
```

```
#include <iostream>
```

```
using namespace std;
```

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

```
#include <conio.h> //for getch()
```

```
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

```

ဒီပရိုဂရမ် တစ်ခုလုံးရဲ့ လျှို့ဝှက်ချက်ကတော့ if(x==7 && y==11) ဆိုတဲ့ ကုဒ်လေးပဲ ဖြစ်ပါတယ်။ Test expression ဟာ x တန်ဖိုး 7 နဲ့ y တန်ဖိုး 11 နှစ်ခုလုံး မှန်မှ၊ မှန်မှာ ဖြစ်ပါတယ်။ အဲဒီ ရလဒ် ဖြစ်လာအောင် logical AND operator && က relational operator နှစ်ခု (x==7) နဲ့ (y==11) တို့ကို ပေါင်းစပ်ပေးပါတယ်။ မှတ်သားသင့်တဲ့ အချက်တစ်ခုက relational expressions နှစ်ခုကို လက်သဲကွင်း အဖွင့်အပိတ်နဲ့ ကွင်းခတ်ထားဖို့ မလိုတာပဲ ဖြစ်ပါတယ်။

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

relational operators တွေဟာ logical operators တွေထက် precedence ပိုမြင့်ပါတယ်။

ကုဒ်တွေကို run ကြည့်ရင် အောက်ပါအတိုင်း တွေ့ရမှာပါ-

Your location is 7, 10

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

You found the treasure!

C++ မှာ Logical Operator ၃ ခု ရှိပါတယ်-

<i>Operator</i>	<i>Effect</i>
&&	Logical AND
	Logical OR
!	Logical NOT

Lafore ရဲ့ စာအုပ်[1]၊ စာမျက်နှာ ၁၁၆ မှာ ပြောထားတာက C++ မှာ XOR (exclusive OR) logical operator မရှိဘူးလို့ ဆိုပါတယ်။ ကျွန်တော်ကတော့ C မှာတုန်းက သုံးခဲ့တဲ့ ^ ကို logical operator အနေနဲ့ သုံးတာ အဆင်ပြေပါတယ်။ Visual Studio 2008 MSDN မှာတော့ အောက်ပါအတိုင်း နှစ်မျိုး သုံးပြုထားပါတယ်။

xor

Search

URL: ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/dv_vcstdlib/html/0fe9554b-d87b-4487-92ed-366c6dc21df2.htm#seeAlsoToggle

Collapse All

Code: Visual C++

Standard C++ Library Reference

Xor

Example See Also Send Feedback

An alternative to the ^ operator.

```
#define xor ^
```

Remarks

The macro yields the operator ^.

Example

```
// iso646_xor.cpp
// compile with: /EHsc
#include <iostream>
#include <iso646.h>

int main( )
{
    using namespace std;
    int a = 3, b = 2, result;

    result= a ^ b;
    cout << result << endl;

    result= a xor b;
    cout << result << endl;
}
```

Logical OR Operator

ကျွန်တော်တို့ အနေနဲ့ game ကစားမယ့် user က အရှေ့နဲ့ အနောက်ကို ခပ်ဝေးဝေးသွားမိရင် နဂါးနဲ့ တွေ့အောင် ရေးချင်တယ် ဆိုကြပါစို့။ လွတ်လွတ်လပ်လပ် လျှောက်သွားနေတာကို ကြောက်စရာ အဖြစ်လေးတွေ ထပ်ထည့်ဖို့ အရင် အပတ်က advenand.cpp ကို ပြင်ရေးထားတဲ့ advenor.cpp မှာ logical OR operator ကို အသုံးပြုထားပါတယ်။

```
// 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 ပရိုဂရမ်လေးထဲမှာ $x < 5$ || $x > 15$ ဆိုတာက x တန်ဖိုး 5 ထက် ငယ်ပြီး (အနောက်ဘက်ကို ဝေးဝေးသွားမိတယ်ဆိုပါစို့) 15 ထက်ကြီးတဲ့ (အရှေ့ရောက်လွန်းသွားတဲ့) အခြေအနေတွေမှာ true ဖြစ်ပါလိမ့်မယ်။ ဒီနေရာမှာ OR operator (||) က relational operators တွေ ဖြစ်ကြတဲ့ $<$ နဲ့ $>$ ထက် precedence နိမ့်တာ ဖြစ်လို့ ကွင်းဆတ်ပေးစရာ မလိုအပ်ပါဘူး။ ဒါပေမယ့် ကျွန်တော့်သဘောကတော့ ဖြစ်နိုင်ရင် လက်သဲကွင်းကို အမြင်ရှင်းအောင် ထည့်ပေးစေလိုပါတယ်။ $(x < 5) || (x > 15)$

Logical NOT Operator

Logical NOT operator! ဆိုတာ Operand တစ်ခုတည်း လိုအပ်တဲ့ unary operator တစ်ခုပဲ ဖြစ်ပါတယ်။ (C++ မှာ Operator အများစုဟာ operand နှစ်ခု လိုတဲ့ binary operator များဖြစ်ကြပြီး ယခင်က သင်ခန်းစာများမှာ လေ့လာခဲ့တဲ့ conditional operator တစ်ခုတည်းသာ operand သုံးခုလိုတဲ့ ternary operator ပဲ ဖြစ်ပါတယ်။ NOT operator (!) ရဲ့ လုပ်ဆောင်ချက်ကတော့ operand ရဲ့ logical value ကို ဆန့်ကျင်ဘက် တန်ဖိုး ပြောင်းလဲ ပစ်တာပဲ ဖြစ်ပါတယ်။ မှန်နေတာကို မှားအောင် လုပ်ပေးနိုင်ပြီး၊ မှားနေတာကို မှန်အောင် လုပ်ပေးတယ်ပေါ့။ (ဘဝကိုသာ ဒီလို လွယ်လွယ်ကူကူ ပြုပြင်နိုင်စွမ်းရှိရင် ဘယ်လောက် ကောင်းမလဲနော်)။ ဥပမာအားဖြင့် $(x == 7)$ မှာ x တန်ဖိုးက 7 နဲ့ တူရင် true ဖြစ်မှာပါ။ ဒါကို $!(x == 7)$ လို့ပြောင်းရေးလိုက်တာနဲ့ x ဟာ 7 နဲ့ မတူမှ true ဖြစ်မှာပါ။ အဓိပ္ပါယ်က $x != 7$ နဲ့ တူသွားပါတယ်။

A True/False Value for Every Integer Variable

ဒီလို operator တွေအကြောင်းကို လေ့လာမိတဲ့ အခါ expression တစ်ခုမှာ true/false တန်ဖိုး ရလာစေဖို့ relational operator တွေ သုံးဖို့ လိုအပ်တယ်ဆိုတဲ့ အတွေး ဝင်လာနိုင်ပါတယ်။ ဒါပေမယ့် တကယ်တမ်းမှာ variable တစ်ခုတည်း ဖြစ်နေပါစေ integer expression မှန်သမျှ true/false value တွေ ရှိနေပါတယ်။ အဲဒီ integer x ရဲ့တန်ဖိုးဟာ 0 နဲ့ မညီသမျှ true ဖြစ်နေပြီး 0 ဖြစ်ရင်တော့ false ဖြစ်သွားမှာပါ။ ဒီနေရာမှာ ! Operator ကို အသုံးပြုလိုက်မယ်ဆိုရင် $!x$ ဟာ x ရဲ့တန်ဖိုးဟာ 0 နဲ့ ညီသမျှ true ဖြစ်နေပြီး မညီရင်တော့ false အဖြစ် ပြောင်းပြန် ပြောင်းလဲသွားမှာပဲ ဖြစ်ပါတယ်။

ထုံးစံအတိုင်း adventure game လေးမှာ အထက်က idea တွေကို အသုံးပြုကြည့်ရအောင်။ ကျွန်တော်တို့ အနေနဲ့ x နဲ့ y တန်ဖိုး နှစ်ခုစလုံး ၇ ရဲ့ ဆတိုးကိန်း (multiple) တွေ ဖြစ်နေတဲ့

နေရာတွေမှာ မှီတွေ ထားချင်တယ် ဆိုပါစို့။ (အဲဒီမှီတွေ စားမိရင် player ကို magical power တွေ ရစေပါတယ်) ဒါဆို x ရော y ပါ 7 နဲ့ စားလို့ ပြတ်တဲ့ နေရာတွေမှာ (x%7 ရော y%7 ပါ သုညနဲ့ ညီတဲ့အခါ) အဲဒီ effect ကို ပေးရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီနေရာကို ဆုံးဖြတ်ဖို့ if(x%7==0 && y%7==0) cout << "There's a mushroom here.\n"; ဆိုပြီး ရေးနိုင်ပါတယ်။ ပိုပြီး တိကျချင်တယ် ဆိုရင် ! operator ကို အသုံးပြုပြီး အောက်ပါအတိုင်း ရေးသားလို့ ရပါတယ်။

if(!(x%7) && !(y%7)) // if not x%7 and not y%7 ရလဒ်ကတော့ အတူတူပဲ ဖြစ်ပါတယ်။ အရင် သင်ခန်းစာတွေတုန်းက logical operator && နဲ့ || ဟာ relational operators တွေထက် precedence နိမ့်တယ်ဆိုတာကို လေ့လာခဲ့ကြပါတယ်။ ဒါဆို အခု ဘာလို့ x%7 နဲ့ y%7 ကို လက်သဲကွင်း မဖြစ်မနေ ခတ်ဖို့ လိုလာတာလဲ? တကယ်တော့ ! ဟာ logical operator ဖြစ်ပေမယ့် unary operator ဖြစ်တာကြောင့် relational operators တွေထက် precedence ပိုမြင့်နေလို့ပဲ ဖြစ်ပါတယ်။

Precedence Summary

Operators တွေရဲ့ precedence တွေကို အနှစ်ချုပ်လေ့လာကြရအောင်။ ဒီ စာရင်းမှာ အပေါ်က operator တွေဟာ အောက်က operator တွေထက် precedence ပိုမြင့်ပါတယ်။ Row တူတဲ့ operator တွေကတော့ precedence တူညီကြပါတယ်။ လုပ်ဆောင်တဲ့ နေရာမှာ precedence ပိုမြင့်တဲ့ operator က အရင် လုပ်ဆောင်မှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် ကျွန်တော်တို့ မဖြစ်မနေ အရင်လုပ်ချင်တဲ့ expression ကို လက်သဲကွင်း ခတ်ပေးခြင်းဖြင့် force နဲ့ လုပ်ဆောင် နိုင်ပါသေးတယ်။

<i>Operator type</i>	<i>Operators Precedence</i>
Unary !, ++, --, +, -	Highest
Arithmetic Multiplicative *, /, %	
Additive +, -	
Relational Inequality <, >, <=, >=	
Equality ==, !=	
Logical And &&	
Or	
Conditional ?:	

Assignment =, +=, -=, *=, /=, %= **Lowest**

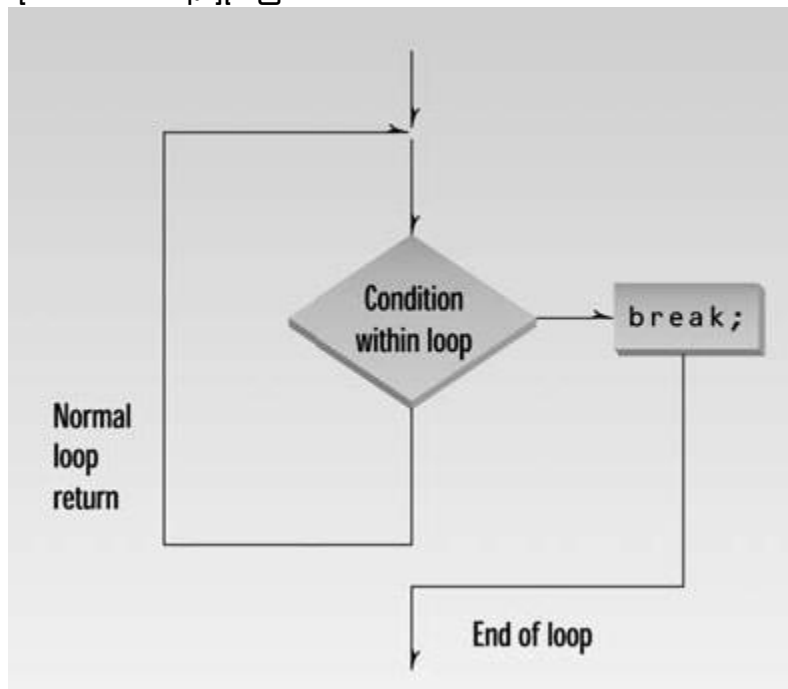
Relational expression တွေ များပြားလာရင် ရှုပ်ထွေးလာတတ်တာကြောင့် လိုလို မလိုလို လက်သဲကွင်းတွေကို အသုံးပြုခြင်းအားဖြင့် အမှားအယွင်း နည်းသွားစေပါတယ်။ လက်သဲကွင်းပိုသွားလို့ ဘာမှ မဖြစ်သွားပါဘူး။ precedence တွက်တာ မှားသွားခဲ့ရင်တောင် ကွင်းခတ်ထားလို့ ပြဿနာ ရှင်းပါတယ်။ ကုန်တွေကို ဖတ်ရတာလည်း ပိုမို လွယ်ကူလာစေပါတယ်။

Other Control Statements

C++ မှာ အခြား control statements တွေ ရှိပါသေးတယ်။ switch ကို လေ့လာတုန်းက break statement ကို အသုံးပြုနည်းကို တွေ့ခဲ့ကြပါတယ်။ ဒါပေမယ့် switch မှာသာမဟုတ်ဘဲ အခြားနေရာတွေမှာလဲ break ကို သုံးလို့ ရပါသေးတယ်။ continue ကိုတော့ loops တွေမှာပဲ သုံးလို့ ရပါတယ်။ goto ကိုတော့ မသုံးသင့်တော့ပါဘူး။ အဲဒီ statements တွေကို အသေးစိတ် လေ့လာကြည့်ကြရအောင်။

The break Statement

switch မှာ လေ့လာခဲ့ကြသလိုပါပဲ break statement ဟာ loop တစ်ခုကနေ ထွက်သွားစေပါတယ်။ ဒါကြောင့် break ပြီးတော့ ဆက်လုပ်မယ့် statement ဟာ loop အပြင်ဘက်က ပထမဆုံးတွေရမယ့် statement ပဲဖြစ်ပါတယ်။ ပုံ (၁၇-၁) မှာ break statement ရဲ့ လုပ်ဆောင်ပုံတွေကို flow chat နဲ့ ရှင်းပြထားပါတယ်။



ပုံ (၁၇-၁) မှာ break statement ရဲ့ လုပ်ဆောင်ချက်ပြပုံ

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

break အကြောင်းကို ရှင်းပြဖို့အတွက် အောက်ပါ showprim.cpp ပရိုဂရမ်လေးကို အသုံးပြုပါမယ်။
 ၎င်းဟာ prime numbers တွေရဲ့ distribution ကို graphical form နဲ့ ပြသပေးမှာ ဖြစ်ပါတယ်။

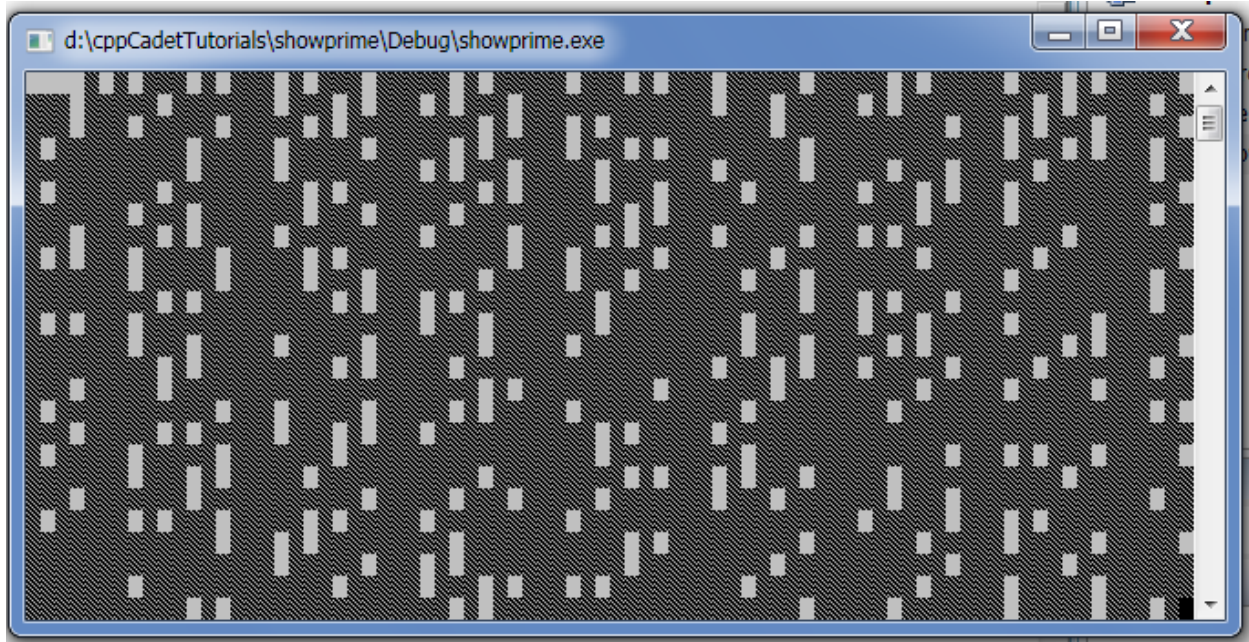
```
// showprim.cpp
// displays prime number distribution
#include <iostream>
using namespace std;
#include <conio.h> //for getch()
int main()
{
    const unsigned char WHITE = 219; //solid color (primes)
    const unsigned char GRAY = 176; //gray (non primes)
    unsigned char ch;
    //for each screen 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;
}
```

```

// showprim.cpp
// displays prime number distribution
#include <iostream>
using namespace std;
#include <conio.h> //for getch()
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 ရဲ့ အကျယ်ဟာ 80 columns နဲ့ 25 rows (lines) ဖြစ်တဲ့ အတွက် ဂဏန်းအရေအတွက်အရဆိုရင် 0-1999 (800*25-1) ရှိပါတယ်။ သက်ဆိုင်ရာ နေရာမှာ ရှိတဲ့ ဂဏန်းတွေဟာ prime ဖြစ်ခဲ့ရင် အဖြူရောင် ခြယ်ပြီး မဟုတ်ခဲ့ရင် အညိုရောင်ခြယ်ပေးမှာပါ။ ပုံ (၁၇-၂) မှာ showprime.cpp ရဲ့ ရလဒ်ကို ပြသပေးထားပါတယ်။ တိတိကျကျပြောရမယ်ဆိုရင် သူညီ နဲ့ တစ် ဟာ prime တွေ မဟုတ်ကြပါဘူး။ ဒါပေမယ့် ပရိုဂရမ်ကို ပိုမိုရှုပ်ထွေးသွားအောင် prime များနည်းတူ အဖြူရောင်ခြယ်ပေးထားပါတယ်။ ပထမဆုံး စာကြောင်းရဲ့ ကော်လံတွေကို သူညီ ကနေ ၇၉ အထိ ဂဏန်းတွေလို့ စဉ်းစားကြည့်ရအောင်။ စုံဂဏန်းတွေ နေရာတိုင်းမှာ ၂ ကလွဲရင် အားလုံး prime မဟုတ်ကြပါဘူး။ ၂ နဲ့ စားလို့ ပြတ်နေတာကြောင့်ပါ။ ဒါဆို အခြား ဂဏန်းတွေအတွက် ပုံသေ ပုံစံ တစ်ခုရော မရှိနိုင်ဘူးလား? တကယ်လို့သာ ပေးထားတဲ့ ဂဏန်းတွေကို prime ဟုတ်မဟုတ် ခန့်မှန်းနိုင်မယ့် pattern တစ်ခုများ ရှာတွေ့နိုင်ခဲ့ရင် ကမ္ဘာ့ သင်္ချာ ပညာရှင်တွေအတွက် စိတ်လှုပ်ရှားစရာ သတင်းတစ်ခု ဖြစ်သွားနိုင်ပါတယ်။



ပုံ(၁၇-၂) showprim.cpp ၏ ရလဒ်ကို တွေ့ရစဉ်

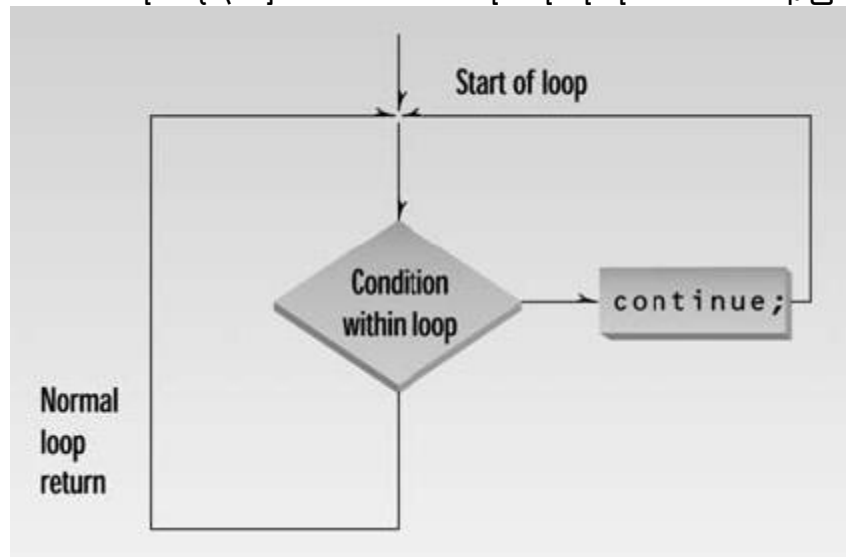
အတွင်းက loop က ဂဏန်းတစ်ခုကို prime မဟုတ်ဘူးလို့ ဆုံးဖြတ်ခဲ့ရင် output ထုတ်ပေးမယ့် character ch ကို GRAY လို့ သတ်မှတ်ပေးပြီးတာနဲ့ အဲဒီ inner loop ကနေ break သုံးပြီး ထွက်သွားမှာ ဖြစ်ပါတယ်။ (ပရိုဂရမ် တစ်ခုလုံးက ထွက်မသွားဘဲ အခြား ဂဏန်းတွေကို စစ်ဆေးဖို့ လိုနေပါသေးတယ်။) အဲဒီမှာ သတိထားရမှာက break ဟာ loop တွေ အားလုံးက ထွက်သွားစေတာ မဟုတ်ဘဲ ၎င်းရှိတဲ့ အတွင်းအကျဆုံး loop ကနေပဲ ထွက်သွားစေနိုင်တာ ဖြစ်ပါတယ်။ ဘယ်လို loop အမျိုးအစားပဲ ငုံထား ငုံထား အဲဒီအတိုင်းပဲ ပြုမူမှာ ဖြစ်ပါတယ်။ ဥပမာ switch ကို loop တစ်ခုက ငုံထားခဲ့ရင် switch ထဲက break statement ဟာ switch ကိုပဲ ထွက်သွားစေပြီး loop ထဲကတော့ ထွက်သွားမှာ မဟုတ်ပါဘူး။ နောက်ဆုံး cout statement ကတော့ loop မပြီးမချင်း prime test က သတ်မှတ်ပေးလိုက်တဲ့ graphics character တွေကို print ထုတ်ပေးမှာပဲ ဖြစ်ပါတယ်။

ASCII Extended Character Set

ဒီပရိုဂရမ်ထဲမှာ 128 ကနေ 255 ရှိတဲ့ *extended ASCII character set* ထဲက character နှစ်ခုကို ယူသုံးထားပါတယ်။ 219 ကတော့ အဖြူရောင် အတုံးလေး တစ်ခု ရဲ့သင်္ကေတ ဖြစ်ပြီး 176 ကတော့ အညိုရောင် အတုံးလေးကို ကိုယ်စားပြုထားတာပါ။ showprim.cpp နဲ့ ယခင်က ဥပမာတွေ အတော်များများမှာ getch() ကို သုံးထားတာ သတိထားမိမှာပါ။ တကယ်တော့ ၎င်းဟာ DOS prompt မှာ ပရိုဂရမ် မထွက်သွားခင် ရလဒ်တွေကို စောင့်ကြည့်နိုင်ဖို့ keyboard က key တစ်ခုကို ဖတ်ဖို့ သုံးထားတာပါ။ ဆိုလိုတာက getch() ကို ရောက်ရင် keyboard က key တစ်ခုကို မနိပ်မချင်း ပရိုဂရမ်က ရပ်နေမှာ ဖြစ်ပါတယ်။ နောက်တစ်ချက်က character variables အတွက် unsigned char ကို သုံးထားတာ ဖြစ်ပါတယ်။ တကယ်တော့ char က 127 အထိပဲ ရတဲ့အတွက် 255 အထိ သုံးလို့ရအောင် unsigned char ကို သုံးထားတာပါ။

The continue Statement

break statement က ကျွန်တော်တို့ကို loop ရဲ့ အောက်ခြေကို ခေါ်ဆောင်သွားပေးပါတယ်။ တခါတလေမှာ ဆန့်ကျင်ဘက်အနေနဲ့ မမျှော်လင့်တာ တစ်ခုခု ဖြစ်လာခဲ့ရင် loop ရဲ့ ထပ်ဆုံးကို ပြန်သွားဖို့ လိုအပ်လာတတ်ပါတယ်။ continue ဆိုတဲ့ statement က အဲဒီ ရလဒ်ကို ပေးနိုင်ပါတယ်။ (ဒါပေမယ့် အတိအကျပြောရရင်တော့ continue ဟာ loop ရဲ့ ကွင်းပိတ်ကို အရင်သွားပြီးမှ ထပ်ဆုံးကို ပြန်ခုန်တက်လိုက်တာပါ။) ပုံ(၁၇-၃) မှာ continue အလုပ်လုပ်ပုံကို flow chat နဲ့ ပြထားပါတယ်။



ပုံ(၁၇-၃) continue အလုပ်လုပ်ပုံ

ယခင် ဥပမာတွေ ပေးတုန်းက divdo.cpp ဆိုတဲ့ ပရိုဂရမ်လေးကို မှတ်မိဦးမယ် ထင်ပါတယ်။ အဲဒီ ပရိုဂရမ်လေးမှာ ကိန်းဂဏန်းတွေကို စားပြထားပါတယ်။ ဒါပေမယ့် သူ့မှာ အဓိက အားနည်းချက် တစ်ခု ရှိပါတယ်။ အဲဒါကတော့ စားကိန်းကို သုည ပေးခဲ့မိရင် Divide Error ဆိုတဲ့ runtime error message ပေါ်လာပြီး ပရိုဂရမ် ထွက်သွားမှာပါ။ အခု ဥပမာပေးမယ့် divdo2.cpp မှာတော့ အဲဒီ အခြေအနေကို သိမ်သိမ်မွေမွေလေး ဖြေရှင်းပေးထားပါတယ်။

```
// 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;
}

```

တကယ်လို့ user က စားကိန်းကို သုည ထည့်ပေးခဲ့မယ်ဆိုရင် program က error message ထုတ်ပေးပြီး continue အသုံးပြုကာ loop ရဲ့ ထိပ်ဆုံးပိုင်းကို ပြန်သွားလို့ ဂဏန်း ထပ်တောင်းမှာ ဖြစ်ပါတယ်။

Enter dividend: 10

Enter divisor: 0

Illegal divisor

Enter dividend:

တကယ်လို့ ဒီနေရာမှာ break ကို သုံးခဲ့မိရင် do loop ကရော ပရိုဂရမ်ကပါ ထွက်သွားမှာ ဖြစ်ပါတယ်။

အခန်း(၃)

Structures

Structures

ဒီအခန်းမှာတော့ Loops and Desicions အကြောင်းကို အဆုံးသတ်တဲ့ အနေနဲ့ အနည်းငယ် ဆွေးနွေးပြီး Structures တွေအကြောင်းကို ဆက်လက် လေ့လာသွားမှာ ဖြစ်ပါတယ်။ တကယ်တော့ ကျွန်တော်တို့ အဲဒီအခန်းကို လေ့လာခဲ့တာ အတော်လေး ပြည့်စုံသွားပါပြီ။ ဒါပေမယ့် အသုံးမပြုသင့်တဲ့ statement တစ်ခုကို ရှင်းပြဖို့ ကျန်ပါသေးတယ်။ ၎င်းကတော့ goto ဆိုတဲ့ statement ပဲ ဖြစ်ပါတယ်။ goto ကို အသုံးပြုခြင်းအားဖြင့် နားလည်ရခက်ပြီး ရှုပ်ထွေးတဲ့ ကုဒ်တွေ ဖြစ်လာစေနိုင်ပါတယ်။ ဒါကြောင့် လုံးဝ အသုံးမပြုကြစေချင်ပါဘူး။ အသုံးပြုနည်းကိုတော့ တင်ပြလိုက်ပါတယ်။

ပထမဆုံး ကျွန်တော်တို့ jump လုပ်ဖို့ လိုအပ်တဲ့ ကုဒ်နေရာမှာ label တစ်ခုကို ရေးသားထားရမှာ ဖြစ်ပါတယ်။ Label ကို colon ":" နဲ့ ပိတ်ပေးရမှာပါ။ ပရိုဂရမ်ရဲ့တနေရာရာမှာ goto ရဲ့ နောက်ကို အဲဒီ label name ထည့်ပေးလိုက်မယ်ဆိုရင် runtime မှာ program control က အဲဒီနေရာရောက်တာနဲ့ label ထိုးထားတဲ့ နေရာကို jump လုပ်သွားမှာ ဖြစ်ပါတယ်။ ဥပမာ -

```
goto SystemCrash;
```

```
// other statements
```

```
SystemCrash:
```

```
// control will begin here following goto
```

ရှေ့ပိုင်း သင်ခန်းစာများမှာ float, char, int အစရှိသဖြင့် ရိုးရှင်းတဲ့ data type တွေကို လေ့လာခဲ့ကြပြီးပါပြီ။ အဲဒီ data type တွေ အားလုံးဟာ (အရပ်အမြင့်၊ ကိုယ်အလေးချိန်၊ ကားတန်ဖိုး စတဲ့) သတင်းအချက်အလက် တစ်ခုကိုပဲ သိမ်းဆည်းထားနိုင်ပါတယ်။ အချက်အလက်တွေကို အမြောက်အများ သိမ်းဆည်းချင်လာတဲ့ အခါမှာ အဲဒီ data type တွေဟာ မလုံလောက်တော့ပါဘူး။ ဒါကြောင့် array, structure စတာတွေကို အသုံးပြုပြီး သိမ်းဆည်းဖို့ လိုလာပါတယ်။ data type တူတဲ့ ဒေတာ အများအပြား သိမ်းဆည်းဖို့အတွက်တော့ နောက်ပိုင်းအခန်းတွေမှာ လေ့လာရမယ့် array ကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။ data type မတူညီတဲ့ ဒေတာတွေကို သိမ်းဆည်းဖို့အတွက်တော့ structure ကို အသုံးပြုနိုင်ပါတယ်။

Structures

Structure ဆိုတာ တကယ်တော့ ရိုးရိုး variable တွေကို စုစည်းထားတာ ဖြစ်ပါတယ်။ အဲဒီ variables တွေဟာ int, float, char စသဖြင့် types အမျိုးမျိုး ဖြစ်နိုင်ပါတယ်။ structure ထဲမှာ ပါတဲ့ variables တွေကို structure ရဲ့ members တွေလို့ ခေါ်ပါတယ်။

C programming စာအုပ်တွေမှာတုန်းကတော့ structures တွေကို advanced feature အဖြစ် ယူဆတဲ့အတွက် စာအုပ်တိုင်းရဲ့ နောက်ဆုံး အခန်းတွေမှာပဲ ဖော်ပြလေ့ ရှိကြပါတယ်။ C++ programmers တွေအတွက်ကတော့ structures ဆိုတာ objects နဲ့ classes တွေကို နားလည်နိုင်ဖို့

အဓိက အခြေခံ အချက် နှစ်ခုထဲက တစ်ခုပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် အရင်ဆုံး မိတ်ဆက် ဖော်ပြပေးထားခြင်း ဖြစ်ပါတယ်။

တကယ်တော့ class နဲ့ structure တွေမှာ syntax တွေ အတော်လေး တူညီနေပါတယ်။ structure က တော့ အချက်အလက်တွေကို စုထားတာဖြစ်ပြီး class က အချက်အလက်တွေ အပြင် function တွေ စုထားတာလေးပဲ ကွာခြားမှု ရှိပါတယ်။ ဒါကြောင့် structure ကို နားလည်ထားခြင်းဖြင့် class နဲ့ object တွေရဲ့သဘောတရားကို လေ့လာရတာ လွယ်ကူစေမှာ ဖြစ်ပါတယ်။ Pascal နဲ့ အခြား language အချို့မှာ ရှိတဲ့ *records* ရဲ့သဘောတရားနဲ့ ရည်ရွယ်ချက်က C နဲ့ C++ မှာရှိတဲ့ structure ရဲ့သဘောတရားနဲ့ အတူတူပဲ ဖြစ်ပါတယ်။

A Simple Structure

Integer variables နှစ်ခုနဲ့ floating point number variable တစ်ခု ပါဝင်တဲ့ structure တစ်ခုကို စတင် ဖန်တီးကြရအောင်။ အဲဒီလို structure မျိုးဟာ ကုမ္ပဏီ တစ်ခုရဲ့ စတိုးထဲက အစိတ်အပိုင်း တွေကို ကိုယ်စားပြုထားပါတယ်။ အစိတ်အပိုင်း တစ်ခုစီအတွက် လိုအပ်တဲ့ အချက်အလက်တွေကို သတ်မှတ်ပေးထားတဲ့ blueprint လိုပါပဲ။ Structure ရဲ့ ပထမ member ကတော့ အဲဒီအစိတ်အပိုင်း ပါဝင်တဲ့ ပစ္စည်းရဲ့ model number ပဲ ဖြစ်ပါတယ်။ ဒုတိယ member ကတော့ အစိတ်အပိုင်း number ဖြစ်ပြီး နောက်ဆုံးတစ်ခုက ကျသင့်ငွေ ဖြစ်ပါတယ်။ အောက်မှာ ဖော်ပြထားတဲ့ ပရိုဂရမ် parts.cpp မှာ part ဆိုတဲ့ structure တစ်ခုကို define ပြုလုပ်ထားပါတယ်။ တို့နောက် အဲဒီ structure type အမျိုးအစား variable တစ်ခု (part1) ကို define ပြုလုပ်ပါတယ်။ သူ့ရဲ့ members တွေထဲကို values တွေ ထည့်သွင်းပေးပြီး display ပြန်လုပ်ပေးမှာ ဖြစ်ပါတယ်။ လေ့လာကြည့်ကြရအောင်-

```
// 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 ကြည့်တဲ့အခါ အောက်ပါ output တွေကို ရရှိမှာ ဖြစ်ပါတယ်။

Model 6244, part 373, costs \$217.55

parts.cpp ပရိုဂရမ်မှာ အဓိက အစိတ်အပိုင်း ၃ ခု ပါဝင်ပါတယ်။ ၁) structure ကို define ပြုလုပ်ခြင်း ၂) structure variable ကို define ပြုလုပ်ခြင်းနဲ့ ၃) structure ရဲ့ members တွေကို ရယူသုံးစွဲခြင်းတို့ပဲ ဖြစ်ပါတယ်။ တစ်ခုချင်းကို အောက်မှာ အသေးစိတ် လေ့လာကြည့်ကြရအောင်။

Defining the Structure

structure definition ဆိုတာကတော့ structure တစ်ခု ဘယ်လို ဖွဲ့စည်းထားတယ် ဆိုတာကို ဖော်ပြပေးပါတယ်။ structure မှာ ပါဝင်မယ့် members တွေကို သတ်မှတ်ပေးပါတယ်။ ဥပမာ -

```

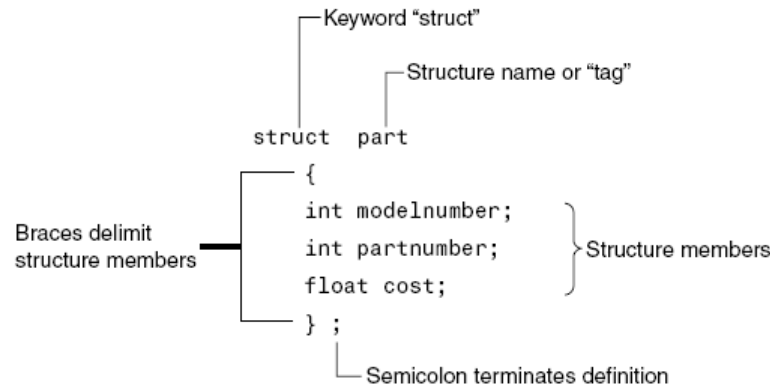
struct part
{
    int modelnumber;
    int partnumber;
    float cost;
};

```

Syntax of the Structure Definition

struct ဆိုတဲ့ keyword နဲ့ structure definition ကို အစပြုထားပါတယ်။ နောက်က ကပ်ရပ်ရေးရတာက *structure name* ဒါမှမဟုတ် *tag* ဖြစ်ပါတယ်။ ဒီဥပမာမှာတော့ part လို့ ပေးထားပါတယ်။ modelnumber, partnumber နဲ့ cost ဆိုတဲ့ structure members တွေရဲ့ declarations ကို တွန့်ကွင်း အဖွင့်အပိတ်ကြားမှာ ရေးသားရပါတယ်။ တွန့်ကွင်း အပိတ်ရဲ့ နောက်မှာ

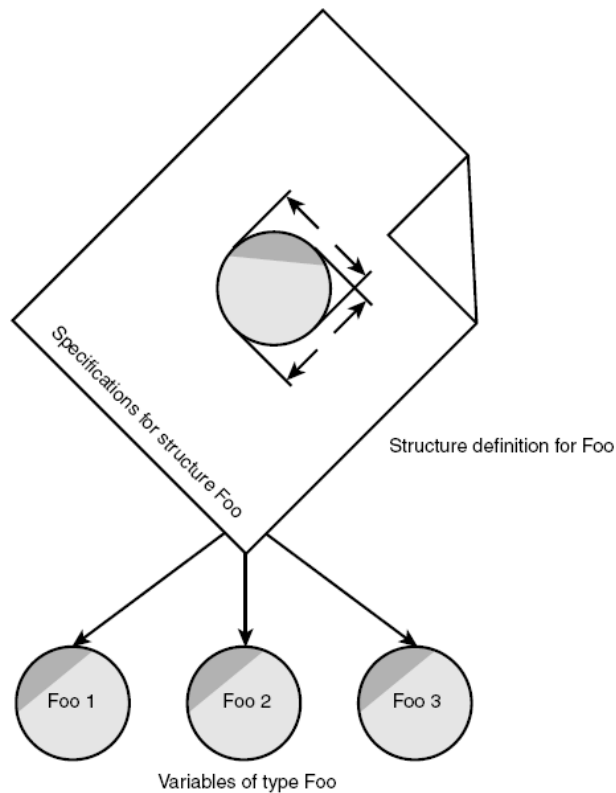
semicolon နဲ့ ပိတ်ပြီး structure တစ်ခုလုံးကို အဆုံးသတ်ထားပါတယ်။ အခြား code block တွေ (ဥပမာ- loop, decisions နဲ့ functions) ကိုတော့ တွန့်ကွင်း အဖွင့်အပိတ်နဲ့ပဲ အစပြု။ အဆုံးသတ်လေ့ရှိပြီး structure မှာတော့ semicolon နဲ့ အဆုံးသတ်တာ ဖြစ်ပါတယ်။ အောက်မှာ structure declaration တစ်ခုရဲ့ syntax ကို ဖော်ပြပေးထားပါတယ်။



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

Use of the Structure Definition

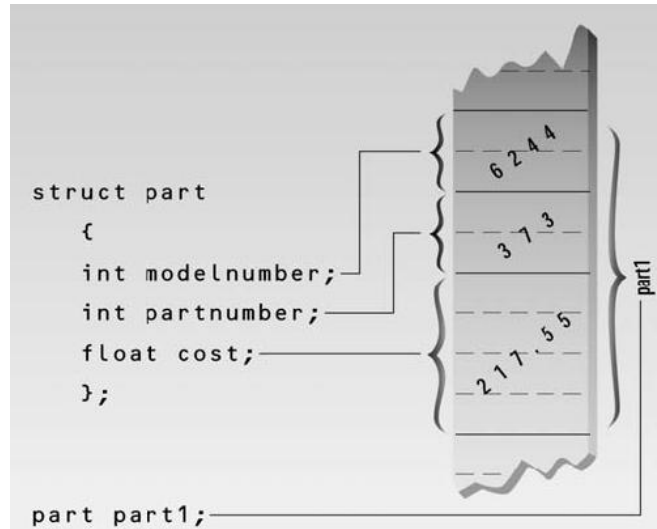
Structure definition သည် အဆိုပါ Structure အမျိုးအစား variables ကို ဖန်တီးရန် blueprint အနေဖြင့်သာ အသုံးပြုထားသည်။ ဆိုလိုသည်မှာ structure definition သည် အခြား သာမန် variables များ၏ definition များလို memoryအတွင်း နေရာယူခြင်း မရှိသလို variable များကို အမည်ပေးခြင်းလည်း မဟုတ်ပေ။ ၎င်းသည် structure variable တစ်ခုကို define ပြုလုပ်ပြီးလျှင် မည်သည့် ပုံစံ ရှိနိုင်မည်ဟူသော ဖော်ပြချက် သက်သက်သာ ဖြစ်သည်။ နောက်ပိုင်း class ကို လေ့လာသည့်အခါ အလားတူ သဘောတရားမျိုးကို အသုံးပြုထားခြင်းဖြစ်ကြောင်း တွေ့ရှိရမည် ဖြစ်သည်။



ပုံ (၁၈-၂) structure နှင့် structure variables

Defining a Structure Variable

main() ထဲရှိ ပထမဆုံး ဖော်ပြချက်ဖြစ်သော **part part1;** သည် structure part type ၏ part1 ဟူသော variable တစ်ခုကို defines ပြုလုပ်ခြင်းပင် ဖြစ်သည်။ အဆိုပါ ကုဒ်ကြောင့် memory ထဲတွင် part1 ၏ members များကို ဖန်တီးမည် ဖြစ်သည်။ မည်မျှ နေရာယူမည်ဆိုသည်ကို part1 အတွင်းရှိ members များဖြစ်သော modelnumber, partnumber နှင့် cost တို့က သတ်မှတ်ပေးသည်။ အထက်ပါ ဥပမာတွင် integer နှစ်ခုအတွက် 4 bytes (32-bit) နှင့် float အတွက် 4 bytes စုစုပေါင်း 8 bytes နေရာယူမည် ဖြစ်သည်။ ပုံ (၁၈-၃) ကိုကြည့်ပါ။



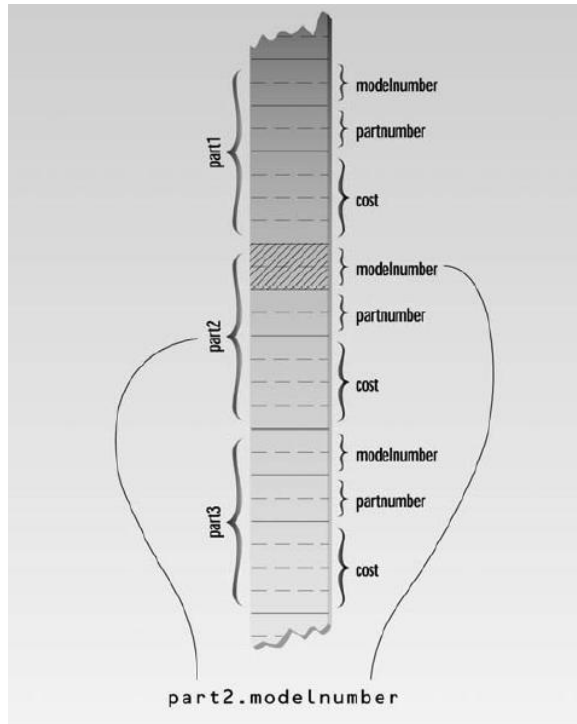
ပုံ (၁၈-၃) memory အတွင်း structure member များ နေရာယူပုံ

Accessing Structure Members

Structure variable တွေကို defined လုပ်ပြီးတာနဲ့ သူ့ရဲ့ member တွေကို *dot operator* အသုံးပြုပြီး access လုပ်နိုင်ပါပြီ။ အောက်မှာ အဲဒီလို ရေးသားနည်းကို နမူနာပြထားပါတယ်။

```
part1.modelnumber = 6244;
```

Structure member တွေကို အသုံးပြုဖို့ အပိုင်း (၃) ပိုင်း ရေးသားပေးရမှာ ဖြစ်ပါတယ်။ ပထမ အပိုင်းကတော့ structure variable ရဲ့ name ဖြစ်ပါတယ်။ အဲဒီနောက်မှာ ဒုတိယအပိုင်းအနေနဲ့ period (.) ကိုရေးသားရမှာဖြစ်ပြီး နောက်ဆုံး တတိယ အပိုင်းအနေနဲ့ member name (ဒီဥပမာမှာ- modelnumber) ကို ရေးသားရမှာပါ။ အဲဒီအဓိပ္ပါယ်ကတော့ "the modelnumber member of part1." ဒါမှမဟုတ် part1 ရဲ့ member တစ်ခုဖြစ်တဲ့ modelnumber လို့ ဆိုလိုတာပဲ ဖြစ်ပါတယ်။ Dot operator ရဲ့အခေါ်အဝေါ်အပြည့်အစုံ က *member access operator* ဖြစ်ပေမယ့် ရှည်တဲ့အတွက် အသုံးမပြုချင်ကြတာ ဖြစ်ပါတယ်။ အဲဒီမှာ သတိပြုရမယ့် အချက်ကတော့ ပထမဆုံး ရေးသားတဲ့ အပိုင်းဟာ structure definition (part) မဟုတ်ဘဲ specific structure variable (part1) ဖြစ်တယ်ဆိုတာပါပဲ။ Variable name တွေဟာ part1, part2 စသဖြင့် တစ်ခုနဲ့ တစ်ခု ခွဲခြားဖို့အတွက် အသုံးပြုတာပဲ ဖြစ်ပါတယ်။ အောက်က ပုံ(၁၉-၁) မှာ လေ့လာကြည့်နိုင်ပါတယ်။



ပုံ(၁၉-၁) Dot operator အသုံးပြုပုံ

Structure members တွေကို အခြား variables တွေလိုပဲ လုပ်ဆောင်စေပါတယ်။ ဥပမာ `part1.modelnumber = 6244;` ဆိုတဲ့ statement မှာ သာမန် assignment operator ကိုသုံးပြီး အဲဒီ member ထဲကို တန်ဖိုး 6244 ထည့်သွင်းပေးလိုက်တာ ဖြစ်ပါတယ်။ `cout` statements တွေကို သုံးပြီးတော့လည်း members တွေရဲ့တန်ဖိုးတွေကို print ထုတ်ပေးလို့ ရပါတယ်။

```
cout << "\nModel " << part1.;
```

အထက်ပါ statement ဟာ structure member ဖြစ်တဲ့ `modelnumber` ရဲ့ value ကို output ထုတ်ပေးမှာ ဖြစ်ပါတယ်။

Other Structure Features

Structures တွေဟာ အံ့သြစရာ ကောင်းလောက်အောင် စွယ်စုံ အသုံးဝင်ပါတယ်။ ဒါကြောင့် နောက်ထပ် အသုံးတည့်တဲ့ structure တွေရဲ့ features တွေကို လေ့လာကြည့်ကြရအောင်။

Initializing Structure Members

နောက်လာမယ့် ဥပမာ ပရိုဂရမ်လေးမှာ structure variable တွေကို define ပြုလုပ်ရင်း structure members တွေကို initialize ပြုလုပ်နည်းတွေကို ပြသထားပါတယ်။ ဒီဥပမာမှာပဲ structure type တစ်ခုကို variable တစ်ခုထက်ပိုပြီး အသုံးပြုနိုင်တာကိုလဲ လေ့လာနိုင်ပါတယ်။

```
// 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;
}

```

ဒီပရိုဂရမ်လေးမှာ part ရဲ့ type အမျိုးအစား variables နှစ်ခု (part1, part2) ကို define ပြုလုပ်ပါတယ်။ part1 ကိုတော့ define ပြုလုပ်ချိန်မှာပဲ initialize လုပ်ပေးလိုက်ပါတယ်။ တနည်းအားဖြင့် တန်ဖိုးတွေ ထည့်သွင်းပေးလိုက်တာပါ။ ပြီးတော့ part1 ထဲက members တွေရဲ့ တန်ဖိုးတွေကို print ထုတ်ပေးပါတယ်။ part2 ထဲကို part1 assign လုပ်ပေးပါတယ်။ နောက်ဆုံးမှာ part2 ထဲက members တွေရဲ့ တန်ဖိုးတွေကို အောက်ပါအတိုင်း print ထုတ်ပေးလိုက်ပါတယ်။

Model 6244, part 373, costs \$217.55

Model 6244, part 373, costs \$217.55

တန်ဖိုးနှစ်ခု တူနေကြတာကတော့ variable နှစ်ခုဟာ တူညီနေလို့ပဲ ဖြစ်ပါတယ်။

part part1 = { 6244, 373, 217.55 }; ဆိုတဲ့ statement ကတော့ part1 ကိုတော့ define ပြုလုပ်ချိန်မှာ initialize လုပ်ပေးလိုက်တာပါ။ structure members တွေကို assign လုပ်မယ့် တန်ဖိုးတွေကို

တွန့်ကွင်းနဲ့ ခတ်ပြီး တစ်ခုနဲ့ တစ်ခုကြား ကော်မာခံထားဖို့ လိုပါတယ်။ ပထမဆုံး တန်ဖိုးကို ပထမ member ဒုတိယ တန်ဖိုးကို ဒုတိယ member စသဖြင့် ထည့်သွင်းပေးသွားမှာ ဖြစ်ပါတယ်။

Structure Variables in Assignment Statements

ကျွန်တော်တို့ partinit.cpp မှာ တွေ့ခဲ့ကြတဲ့ အတိုင်း structure variable တွေကို တစ်ခုနဲ့ တစ်ခု အောက်ပါအတိုင်း assign ပြုလုပ်နိုင်ပါတယ်။

```
part2 = part1;
```

အဲဒီအခါမှာ part1 ရဲ့ member အားလုံးဟာ part2 ရဲ့ သက်ဆိုင်ရာ member တွေအားလုံးကို assign ပြုလုပ်ပေးပါတယ်။ Structure တစ်ခုမှာ member တွေ ဒါဇင်နဲ့ချီပြီး ပါဝင်နိုင်တာကြောင့် ကွန်ပျူတာအနေနဲ့ ဒီ assignment ဖြစ်စဉ်ကို ဆောင်ရွက်ဖို့ အလုပ်တော်တော်များများ လုပ်ပေးရမှာ ဖြစ်ပါတယ်။

မှတ်သားရမှာ တစ်ခုကတော့ အဲဒီလို assign ပြုလုပ်မယ့် structures တွေဟာ structure type တူညီဖို့ လိုအပ်ပါတယ်။ တကယ်လို့ မတူညီတဲ့ structures နှစ်ခုကို assign ပြုလုပ်မယ်ဆိုရင် error message ပြမှာ ဖြစ်ပါတယ်။

A Measurement Example

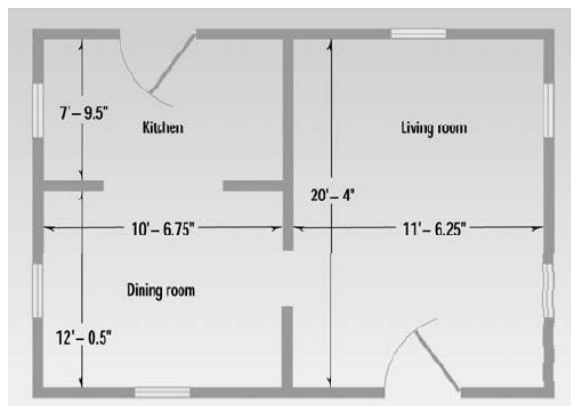
နောက်ထပ် ဥပမာတစ်ခုအနေနဲ့ မတူညီတဲ့ သတင်းအချက်အလက် တွေကို အုပ်စုဖွဲ့ပြီး အသုံးပြုဖို့ structure တစ်ခု တည်ဆောက်ပြပါမယ်။ အဆောက်အဦ တစ်ခုရဲ့ ပုံကြမ်းကို လေ့လာကြည့်မယ်ဆိုရင် အကွာအဝေးတွေဟာ ပေတွေ လက်မတွေနဲ့ တိုင်းတာထားတာကို တွေ့ရှိရမှာပါ။ (တစ်ပေမှာ ၁၂ လက်မ ရှိတယ်ဆိုတော့ အားလုံး သိပြီးသားဖြစ်ပါတယ်။) ဧည့်ခန်းတစ်ခုရဲ့ အလျားကို 15'-8" လို့ ပြထားမယ်ဆိုရင် အဲဒီမှာ သုံးထားတဲ့ hyphen ဟာ အနုတ် လက္ခဏာ မဟုတ်ဘဲ ပေနဲ့ လက်မကို ခြားထားပေးတာပါ။ ၁၅ ပေ နဲ့ ၈ လက်မလို့ ဆိုလိုပါတယ်။ အဲဒီလို စနစ်မျိုးအတွက် ဆိုရင် ပေအတွက် variable တစ်ခုနဲ့ လက်မအတွက် variable တစ်ခုဆိုပြီး နှစ်ခု သုံးရပါလိမ့်မယ်။ အောက်က englstrc.cpp ပရိုဂရမ်လေးမှာ အဲဒီပြဿနာကို structure အသုံးပြုပြီး ဖြေရှင်းပြထားပါတယ်။ အဲဒီ structure ထဲမှာ မတူညီတဲ့ Distance type နှစ်ခုကို အတူတူ ထည့်သွင်းရေးသား အသုံးပြုထားတာ ဖြစ်ပါတယ်။

```
// 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;
}

```



ပုံ(၁၉-၂) အိမ်ပုံကြမ်း နှင့် အတိုင်းအတာများ ပြပုံ

အထက်ပါ ပရိုဂရမ်ထဲမှာ feet နဲ့ inches ဆိုတဲ့ members နှစ်ခု ပါဝင်တဲ့ Distance structure ကို အသုံးပြုထားပါတယ်။ လက်မတွေကို များသောအားဖြင့် ဒဿမကိန်းနဲ့ ဖော်ပြနိုင်တာကြောင့် inches အတွက်

float type ကို သုံးထားပါတယ်။ ပေ ကတော့ အမြဲတမ်း ကိန်းပြည့်ဖြစ်နေတာကြောင့် feet ကို int type အသုံးပြုထားတာပါ။ အတိုင်းအတာတွေကို မှတ်သားဖို့ structure variable သုံးခု (d1, d2, d3) ကို define ပြုလုပ်ပါတယ်။ ဒါပေမယ့် d2 ကိုပဲ initialize ပြုလုပ်ပေးပါတယ်။

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

ပရိုဂရမ်က user ကို အတိုင်းအတာတစ်ခုရဲ့ ပေ နဲ့ လက်မ တန်ဖိုးတွေကို ထည့်သွင်းပေးဖို့ တောင်းဆိုပါတယ်။ (လက်မ တန်ဖိုးဟာ ၁၂ အောက် ငယ်ဖို့ လိုပါတယ်)။ အဲဒီနောက်မှာတော့ d1 နဲ့ d2 ကို ပေါင်းပြီး d3 ထဲမှာ ရလဒ်ကို သိမ်းလိုက်ပါတယ်။ နောက်ဆုံးမှာတော့ အတိုင်းအတာ သုံးခုလုံးရဲ့ တန်ဖိုးတွေကို print ပြန်ထုတ်ပေးပါတယ်။ အောက်မှာ နမူနာ ပြထားပါတယ်။

Enter feet: 10

Enter inches: 6.75

10'-6.75" + 11'-6.25" = 22'-1"

အဲဒီမှာ သတိထားစရာ အချက်ကတော့ အဲဒီ structure variable တွေကို အောက်ပါအတိုင်း တန်းပေါင်းလို့ မရတာပါ။

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

ဘာကြောင့်လဲဆိုတော့ C++ မှာ Distance type variables တွေကို ပေါင်းပေးနိုင်တဲ့ routine တွေ မရှိလို့ဘဲ ဖြစ်ပါတယ်။ + operator ဟာ float, int စတဲ့ built-in types တွေကို ပေါင်းပေးနိုင်ပါတယ်။ ဒါပေမယ့် ကျွန်တော်တို့ ကိုယ်တိုင် define လုပ်လိုက်တဲ့ type တွေကိုတော့ ပေါင်းပေးနိုင်စွမ်း မရှိပါဘူး။ (Class ကို လေ့လာကြတဲ့ အခါမှာတော့ "Operator Overloading," အကြောင်းကို ရှင်းပြပေးပါမယ်။)

Structures Within Structures

structures တွေထဲမှာ နောက်ထပ် structures တွေ nested လုပ်ပြီး ထည့်နိုင်ပါတယ်။ နမူနာအနေနဲ့ englstrc.cpp ကို ပြင်ရေးပြပါမယ်။ ဒီ ဥပမာထဲမှာ အခန်းတစ်ခုရဲ့ အလျားနဲ့ အနံ အရွယ်အစားတွေကို သိမ်းထားမယ့် data structure တစ်ခုကို ဖန်တီးပြပါမယ်။ အင်္ဂလိပ် အတိုင်းအတာတွေကို သုံးထားတာဖြစ်လို့ ပေ နဲ့ လက်မကိုပဲ ဆက်သုံးသွားမှာပါ။ အောက်မှာ Room ဆိုတဲ့ structre တစ်ခုကို Distance type variable length နဲ့ width ကို အသုံးပြုပြီး တည်ဆောက်ပြထားပါတယ်။

struct Room

```
{
    Distance length;
    Distance width;
}
```

အောက်က englarea.cpp မှာတော့ အခန်းတစ်ခန်းကို ကိုယ်စားပြုဖို့ Room structure ကို သုံးထားပါတယ်။

```
// 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;
}

```

ဒီပရိုဂရမ်ထဲမှာ Room type - dining ဆိုတဲ့ variable တစ်ခုကို အောက်ပါအတိုင်း define ပြုလုပ်ထားပါတယ်။

Room dining; // variable dining of type Room

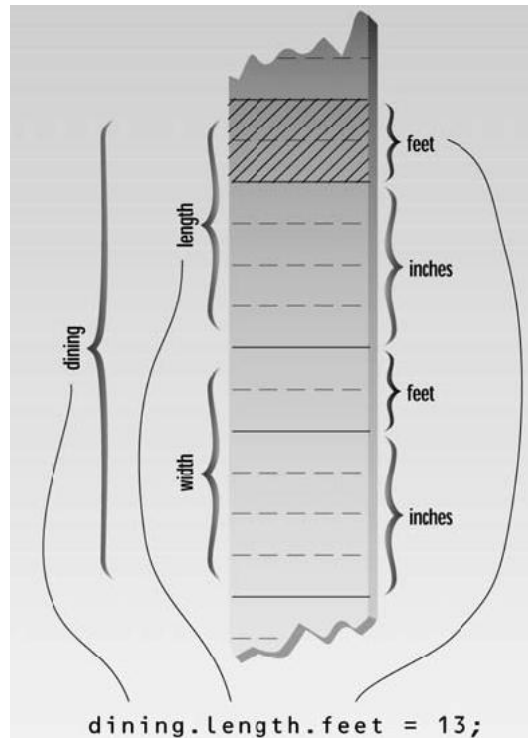
နောက်ပိုင်းမှာတော့ အဲဒီ structure ရဲ့ member တွေကို တန်ဖိုးတွေ assign ပြုလုပ်ပေးပါတယ်။

Accessing Nested Structure Members

Structure တစ်ခုထဲမှာ အခြား structure တစ်ခုကို ပြုလုပ်ထားတာကြောင့် သူတို့ရဲ့ member တွေကို ရယူဖို့ dot operator နှစ်ခုကို အသုံးပြုရမှာပါ။ ဥပမာ-

dining.length.feet = 13;

အဲဒီ statement မှာဆိုရင် dining က structure variable ရဲ့ name ဖြစ်ပါတယ်။ length က အပြင်က structure (Room)ရဲ့ member name ဖြစ်ပြီး feet ကတော့ အတွင်းက structure (Distance) ရဲ့ member name ပါ။ ကုန်ရဲ့ ဆိုလိုချက် အပြည့်အစုံက dining variable ရဲ့ length member ထဲက feet member ထဲကို တန်ဖိုး ၁၃ ထည့်ပေးလိုက်တာပဲ ဖြစ်ပါတယ်။ အောက်က ပုံ (၂၀-၁) မှာ လေ့လာကြည့်ပါ။



ပုံ(၂၀-၁) nested structures များအတွင်း dot operator အသုံးပြုပုံ

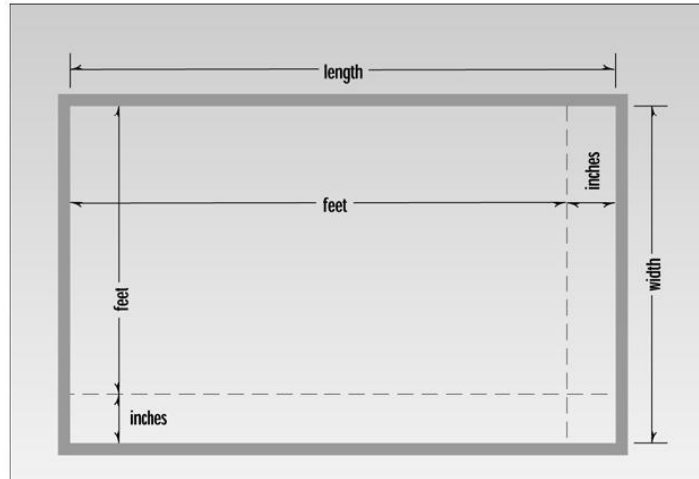
dining ရဲ့ members တွေကို တန်ဖိုးတွေ assign လုပ်ပေးပြီးတာနဲ့ ပရိုဂရမ်က ပုံ(၂၀-၂) မှာ ပြထားသလို အဲဒီအခန်းရဲ့ ကြမ်းခင်း ဧရိယာကို တွက်ပေးမှာ ဖြစ်ပါတယ်။ ဧရိယာကို မတွက်ခင် ပထမဆုံး Distance type တွေဖြစ်တဲ့ length နဲ့ width တွေကို feet နဲ့ တိုင်းတာတဲ့ float variables တွေ ဖြစ်တဲ့ l နဲ့ w အဖြစ် ပြောင်းလဲပေးမှာ ဖြစ်ပါတယ်။ အဲဒီလို ပြောင်းလဲဖို့အတွက် inches member တန်ဖိုးတွေကို 12 နဲ့ စားပြီး feet member တွေထဲ ပေါင်းထည့်ပေးရမှာပါ။

//convert length & width

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

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

အဲဒီလို ပေါင်းပြီးသွားတာနဲ့ ရလာတဲ့ ရလဒ်တွေဟာ float type ဖြစ်သွားပါတယ်။ ဧရိယာတန်ဖိုး ရှာဖို့အတွက်ကတော့ l နဲ့ w ကို မြှောက်ပေးလိုက်ရုံပဲ ဖြစ်ပါတယ်။



ပုံ(၂၀-၂) အခန်းတစ်ခု၏ အတိုင်းအတာများပြပုံ

User-Defined Type Conversions

မှတ်သားရမယ့် အချက်တစ်ခုက ပရိုဂရမ်ဟာ Distance type အတိုင်းအတာ နှစ်ခုကို float type variable l နဲ့ w အဖြစ် ပြောင်းလဲလိုက်တာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် Room type structure ထဲမှာ သိမ်းဆည်းထားတဲ့ (ဒီဥပမာမှာ နောက်ထပ် Distance type structure နှစ်ခုနဲ့ သိမ်းထားတဲ့) အတိုင်းအတာတွေကို square feet နဲ့ ကိုယ်စားပြုတဲ့ ကြမ်းခင်းဧရိယာအဖြစ် floating-point number တစ်ခုတည်းဘဝကို ပြောင်းလဲပေးခဲ့ပါတယ်။ ပရိုဂရမ်ရဲ့ ရလဒ်တွေကတော့ အောက်ပါအတိုင်းပဲ ဖြစ်ပါတယ် -

Dining room area is 135.416672 square feet

User-defined data types တွေ အသုံးပြုထားတဲ့ ပရိုဂရမ်တွေထဲမှာ data type ပြောင်းလဲပေးခြင်းဟာ အရေးကြီးတဲ့ အချက်တစ်ခု ဖြစ်ပါတယ်။

Initializing Nested Structures

အခုလိုမျိုး structure တစ်ခုကို ငုံထားတဲ့ structure တစ်ခုထဲကို initialize လုပ်ပြီး တန်ဖိုးတွေကို ဘယ်လို ထည့်သွင်းကြမှာလဲ? အောက်ပါ statement လေးကတော့ englarea.cpp ထဲက တန်ဖိုးတွေကို တစ်ခါတည်း initialize လုပ်ပြီး ထည့်သွင်းတဲ့ နည်းလမ်း ဖြစ်ပါတယ်။

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

Room structure ထဲမှာ မြှုပ်နှံထားတဲ့ Distance type structure တစ်ခုကို သီးခြား initialized လုပ်ပေးဖို့ လိုပါတယ်။ အဲဒီလို သီးခြားပြုလုပ်ဖို့အတွက် တွန့်ကွင်းတွေနဲ့ ကော်မာတွေကို အသုံးပြုရမှာပဲ ဖြစ်ပါတယ်။

ပထမ Distance ကို {13, 6.5} နဲ့ ဒုတိယ တစ်ခုကိုတော့ {10, 0.0} initialized လုပ်ပါတယ်။ အဲဒီနောက် ရလာတဲ့ Distance value နှစ်ခုကို Room variable ထဲကို ထပ်မံ initialize လုပ်ပေးပါတယ်။ တွန့်ကွင်းနဲ့ ကော်မာတွေပဲ သုံးတာ ဖြစ်ပါတယ်။

Depth of Nesting

သီအိုရီအရ structures တွေထဲမှာ အခြား structures တွေ အဆင့်ဆင့် အကန့်အသတ်မရှိ nested လုပ်လို့ ရပါတယ်။ အဆောက်အဦတွေကို ဒီဇိုင်းပြုလုပ်တဲ့ ပရိုဂရမ်မျိုးမှာ အောက်ပါ ကုဒ်မျိုးတွေကို မြင်တွေ့ရနိုင်ပါတယ်။

apartment1.laundry_room.washing_machine.width.feet

A Card Game Example

နောက်ထပ် ဥပမာလေး တစ်ခုကို သုံးပြီး structures တွေကို လေ့လာကြည့်ကြရအောင်။ ဖဲသုံးချပ် ကစားနည်းကို မြင်ဖူးကြမယ် ထင်ပါတယ်။ များသောအားဖြင့် ဘုရားပွဲတွေမှာ တွေ့ရတတ်ပါတယ်။ အဲဒီကစားနည်းမှာ cardsharp လို့ ခေါ်တဲ့ ဖဲဝိဇ္ဇာက ဖဲချပ် သုံးချပ်ကို ပြထားပြီး စားပွဲပေါ်မှောက်ထားလိုက်ပါတယ်။ ကျွန်တော်တို့ မျက်စေ့ရှေ့မှာပဲ အဲဒီဖဲသုံးချပ်ကို နေရာအကြိမ်ကြိမ်ပြောင်းလိုက်ပါတယ်။ ပြီးတဲ့အခါမှာ ကျွန်တော်တို့ကို ဖဲတစ်ချပ်ကို ရွေးခိုင်းပါတယ်။ သတ်မှတ်ထားတဲ့ ဖဲဖြစ်ရင် ကျွန်တော်တို့ နိုင်ပါတယ်။ မှားခဲ့ရင်တော့ ရှုံးသွားမှာပါ။ ဖဲဝိဇ္ဇာက ဖဲကို ရွှေ့တာ အရမ်းကို မြန်ပြီး ရှုပ်ထွေးတာကြောင့် ကစားသူက မျက်စိနဲ့ လိုက်မမှတ်နိုင်ပါဘူး။ ဒါကြောင့် မှန်းဆဖို့ ခက်ခဲပါတယ်။ အဲဒီကစားနည်းကို အောက်မှာ structure အသုံးပြုထားတဲ့ ပရိုဂရမ်လေးနဲ့ ရေးပြထားပါတယ်။ ပထမဆုံး ဖဲချပ်တွေကို ကိုယ်စားပြုဖို့ card ဆိုတဲ့ structure တစ်ခုကို ဖန်တီးလိုက်ပါတယ်။

```
struct card
```

```
{
```

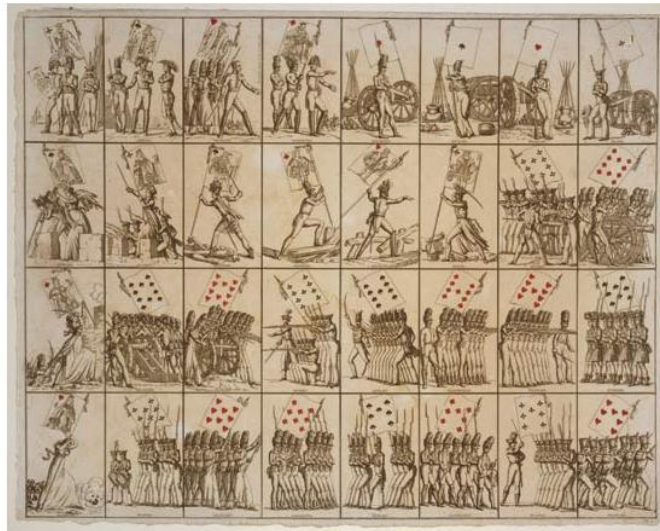
```
    int number;
```

```
    int suit;
```

```
};
```





















အဲဒီ structure က ဖဲတစ်ချပ်ကို ကိုယ်စားပြုပါတယ်။ သူ့ထဲမှာပါတဲ့ number ဆိုတဲ့ member ကတော့ 2 ကနေ 14 အထိ ဂဏန်းတွေကို သိမ်းပေးမှာပါ။ ဒီနေရာမှာ ဖဲထုပ်ထဲမှာ 2 ကနေ 10 အထိပဲ ဂဏန်းရှိတာကြောင့် 11,12,13 နဲ့ 14 တွေဟာ ဂျက်၊ ကွင်း၊ ကင်းနဲ့ အေ တွေကို အသီးသီး ကိုယ်စားပြုပေးမှာပါ။ suit ကတော့ 0 ကနေ 3 အထိ ပါဝင်မှာ ဖြစ်ပြီး ကလပ်၊ ဒိုင်းမွန်း၊ ဟတ် နဲ့ စပိတ်

တွေကို ကိုယ်စားပြုမှာပါ။ စာဖတ်သူတို့ အတွေ့အထူး ဗဟုသုတ ရနိုင်စေဖို့အတွက် ပြိတ်သူ့စွယ်စုံကျမ်း
Encyclopædia Britannica ထဲက ရှေးဟောင်းဖဲချပ်လေးတွေကို ကူးယူဖော်ပြပေးလိုက်ပါတယ်။



Sheet of French playing cards, c. 1800. Soldiers bear a flag that shows the card's suit and rank.

Evolution of suitmarks

	international (French, English)	German	Swiss	Spanish	Italian
suitmarks	 clover, clubs	 acorns	 acorns	 swords	 swords
	 pikes, spades	 leaves	 escutcheons	 clubs	 batons
	 hearts	 hearts	 roses	 cups	 cups
	 tiles, diamonds	 bells	 bells	 coins	 coins

© 2006 Encyclopædia Britannica, Inc.

Suitmarks of playing cards

cards.cpp ကို လေ့လာကြည့်ပါ။

```
// 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;

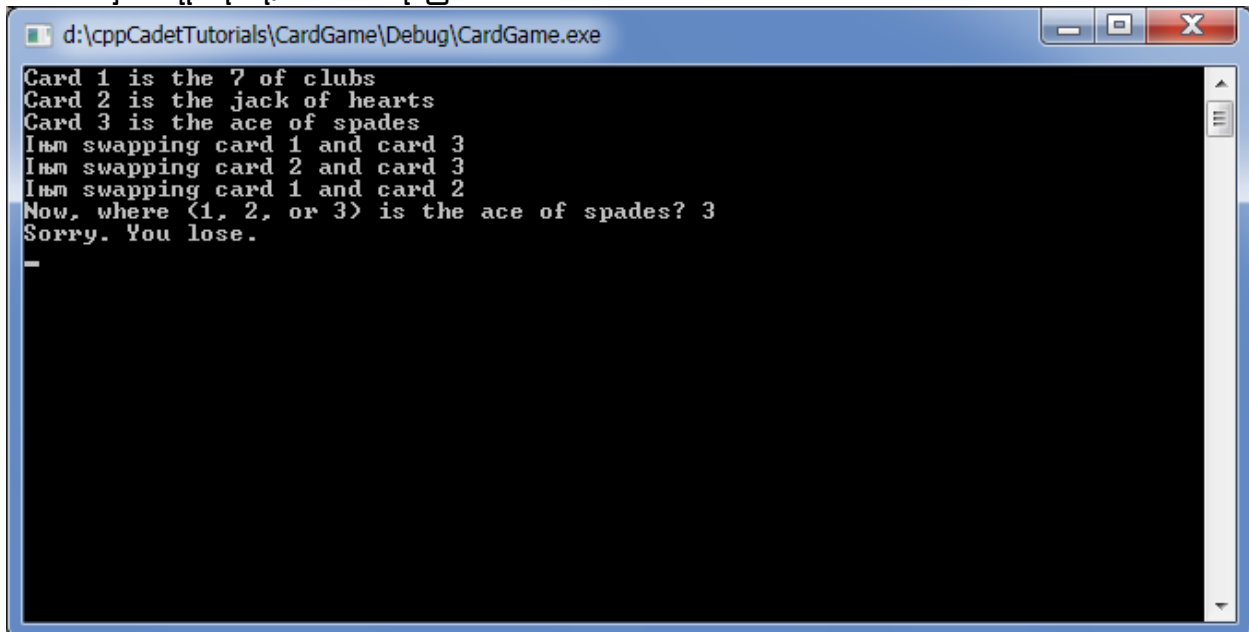
```

```

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;
}

```

အောက်မှာ ပရိုဂရမ်ရဲ့ output ကို ပြထားပါတယ်-



```

d:\cppCadetTutorials\CardGame\Debug\CardGame.exe
Card 1 is the 7 of clubs
Card 2 is the jack of hearts
Card 3 is the ace of spades
I'm swapping card 1 and card 3
I'm swapping card 2 and card 3
I'm swapping card 1 and card 2
Now, where (1, 2, or 3) is the ace of spades? 3
Sorry. You lose.
_

```

အဲဒီ ပရိုဂရမ်မှာ အဖြေမှန်က ၂ ဖြစ်ပါတယ်။

ပရိုဂရမ်ရဲ့အစမှာ face card တွေနဲ့ suit တန်ဖိုးတွေ ထည့်သွင်းဖို့ const int variables တွေ ဖန်တီးပါတယ်။ အဲဒီ variables တွေ အကုန်လုံးကိုတော့ ဒီပရိုဂရမ်ထဲမှာ အသုံးမပြုထားပါဘူး။ ဒါပေမယ့် ပြည့်စုံအောင် ထည့်ပေးထားတဲ့ သဘောပါ။ card structure ကို သတ်မှတ်ပြီးတဲ့ အခါမှာ အဲဒီ card structure အမျိုးအစား variable သုံးခုဖြစ်တဲ့ temp, chosen နဲ့ prize တို့ကို initialize

မလုပ်ဘဲ ဖန်တီးလိုက်ပါတယ်။ နောက်ပိုင်း ဖဲချပ်တွေ ရွှေ့ဖို့နဲ့ ကစားသမား ရွေးချယ်တဲ့ ဖဲချပ်မှန်မမှန် တိုက်ဆိုင် စစ်ဆေးတဲ့ နေရာမှာ အသုံးပြုဖို့ပဲ ဖြစ်ပါတယ်။

ကစားတဲ့ နေရာမှာ အသုံးပြုတဲ့ ဖဲသုံးချပ် အတွက်ကတော့ နောက်ထပ် card1, card2 နဲ့ card3 ဆိုတဲ့ card structure သုံးခုကို တန်ဖိုးတွေနဲ့ initialized လုပ်ပေးလိုက်ပါတယ်။ အဲဒီနောက်မှာ ကစားသမား သိရှိနိုင်စေရန် ရည်ရွယ်ပြီး အဲဒီ ဖဲချပ်တွေကို print out ထုတ်ပေးလိုက်ပါတယ်။ ကစားသူမှ ခန့်မှန်း ရွေးချယ်ရမယ့် ဖဲချပ်ကို မှတ်သားထားဖို့ အတွက် prize ဆိုတဲ့ variable ကို အသုံးပြုပါတယ်။

ပရိုဂရမ်က ဖဲချပ်တွေကို နေရာရွှေ့ပြောင်းပါတယ်။ ပထမဖဲကို တတိယဖဲနဲ့ လဲပါတယ်။ နောက် ဒုတိယဖဲကို တတိယဖဲနဲ့ လဲပါတယ်။ နောက်တစ်ခါမှာတော့ ပထမဆုံးဖဲကို ဒုတိယဖဲ နဲ့ နေရာလဲလိုက်ပါတယ်။ နေရာလဲသမျှကိုလည်း ကစားသူကို အသိပေးနေမှာပါ။ (ဒီပရိုဂရမ်လေးမှာတော့ နမူနာအနေနဲ့ ရေးပြထားတာဖြစ်လို့ လွယ်ကူစွာ ခန့်မှန်းနိုင်ပေမယ့် တကယ်တမ်း ရာနဲ့ချီပြီး နေရာလဲလိုက်မယ် အသိပေးချက်တွေကိုလဲ အချိန်အနည်းငယ်သာ ပြသပေးလိုက်မယ်ဆိုရင် ခန့်မှန်းဖို့ ခက်သွားမှာ ဖြစ်ပါတယ်။)

နောက်ဆုံးမှာ ကစားသူကို သတ်မှတ်ထားတဲ့ဖဲချပ် ရှိတဲ့ နေရာကို ခန့်မှန်းခိုင်းပါတယ်။ အဲဒီတန်ဖိုးကို chosen ထဲကို ထည့်ပါတယ်။ မှန်မှန်ကို သိဖို့ prize card နဲ့ နှိုင်းယှဉ် စစ်ဆေးပါတယ်။ တကယ်လို့ chosen နဲ့ prize တန်ဖိုးတွေ တူခဲ့ရင် ကစားသူ အနိုင်ရရှိမှာ ဖြစ်ပြီး မတူခဲ့ရင်တော့ ရှုံးနိမ့်မှာ ဖြစ်ပါတယ်။

ဖဲချပ်တွေ လဲလှယ်ဖို့အတွက်ကတော့ structures တွေ ဖြစ်ပေမယ့် assignment operator ကိုပဲ သုံးပြီး အလွယ်တကူ ပြုလုပ်ထားပါတယ်။ temp = card3; card3 = card1; card1 = temp; ဒီနေရာမှာ ပြဿနာတစ်ခုက structure တွေကို ပေါင်းလို့မရသလို == operator ကိုသုံးပြီး (chosen == prize ဆိုတဲ့ ပုံစံမျိုး) အလွယ်တကူ နှိုင်းယှဉ်လို့ မရနိုင်တာ ဖြစ်ပါတယ်။ ဒီပြဿနာကို Class အကြောင်း လေ့လာတဲ့အခါမှာ တွေ့ရှိရမယ့် operator overloading နည်းလမ်းနဲ့ ဖြေရှင်းပုံကို နောက်ပိုင်းမှာ ဆက်လက် ရှင်းပြပေးသွားမှာ ဖြစ်ပါတယ်။

Structures and Classes

ကြိုကြိုက်လာတဲ့အတွက် structure နဲ့ class အကြောင်း အနည်းငယ် ရှင်းပြလိုပါတယ်။ structure တွေကို data တွေ စုစည်းဖို့ အသုံးပြုပြီး class တွေကိုတော့ data တွေနဲ့ function တွေ စုစည်းထားဖို့ အသုံးပြုတယ်လို့ ပြောခဲ့ပါတယ်။ C မှာတော့ structure ထဲမှာ data ပဲ ထည့်လို့ ရပါတယ်။ ဒါပေမယ့် C++ မှာ structures တွေကို data တွေသာမက function တွေကိုပါ ထည့်သွင်းနိုင်ပါတယ်။ သို့သော် C++ ပရိုဂရမ်မာ အများစုဟာ structures တွေကို data တွေအတွက်ပဲ သီးသန့်သုံးလေ့ ရှိပြီး function တွေ ထည့်သုံးဖို့လိုလာတိုင်း classes တွေကိုသာ အသုံးပြုတတ်ကြပါတယ်။ ဒါကြောင့် စာရေးသူအနေနဲ့လဲ structures တွေကို data တွေ စုစည်းဖို့ အသုံးပြုနည်းများကိုသာ ဆက်လက်တင်ပြပေးသွားမှာ ဖြစ်ပါတယ်။

Enumerations

ပြီးခဲ့တဲ့ သင်ခန်းစာတွေမှာ user-defined data types အဖြစ် structures များအား အသုံးပြုနည်းကို လေ့လာခဲ့ကြပါတယ်။ ယခု အပတ်မှာတော့ enumeration ကို အသုံးပြုပြီး ကိုယ်ပိုင် data types များ ဖန်တီးပုံကို နောက်ထပ် နည်းလမ်းတစ်ခုအနေနဲ့ ဆွေးနွေးလိုပါတယ်။ ဒီနည်းလမ်းက structures တွေလောက်တော့ အရေးမကြီးပါဘူး။ C++ ပရိုဂရမ်တွေကို enumerations အကြောင်းမသိပဲ ကောင်းကောင်းရေးနိုင်ပါတယ်။ ဒါပေမယ့် enumerations တွေအကြောင်းကို သိရှိ အသုံးပြုနိုင်မယ်ဆိုရင် ပိုမိုရှင်းလင်း လွယ်ကူပြီး C++ ပိုဆန်တဲ့ ပရိုဂရမ်တွေကို ရေးသားနိုင်လာမှာ ဖြစ်ပါတယ်။

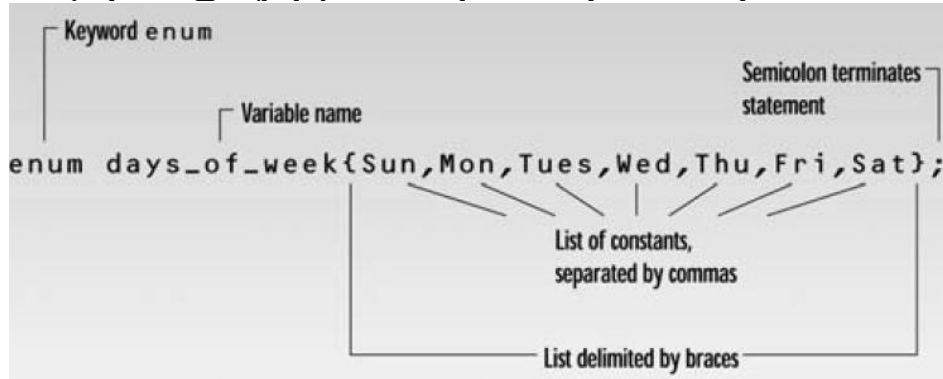
Days of the Week

များသောအားဖြင့် အတိအကျ သိရှိထားပြီးသား data အနည်းငယ်သာ ပါဝင်တဲ့ data type တစ်ခုကို ဖန်တီးဖို့ enumeration ကို အသုံးပြုလေ့ ရှိကြပါတယ်။ ဥပမာအနေနဲ့ တစ်ပတ်စာ နေ့တွေကို enumeration လုပ်ထားတဲ့ dayenum.cpp ကို လေ့လာကြရအောင် -

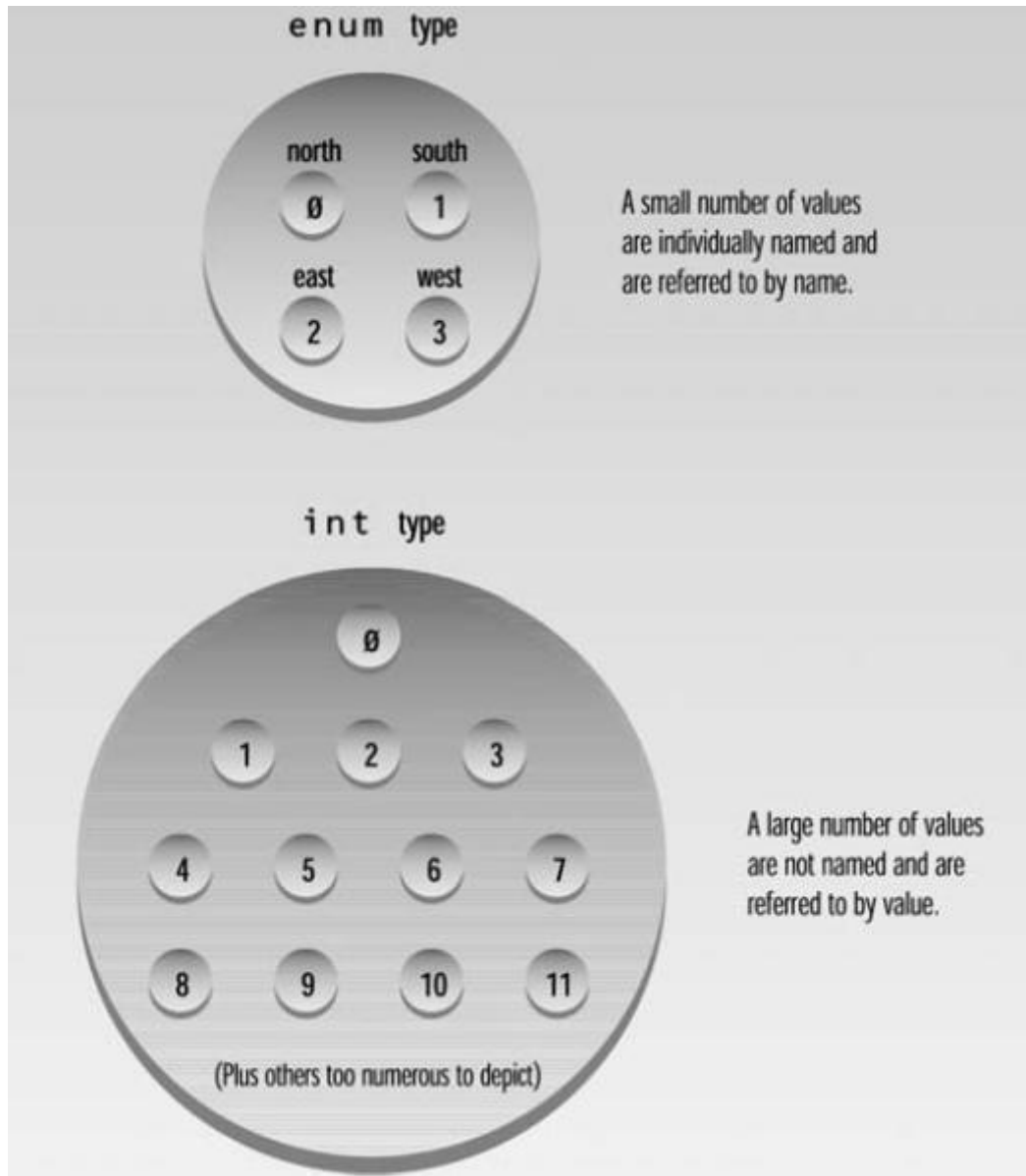
```
// dayenum.cpp
// demonstrates enum types
#include <iostream>
using namespace std;
//specify enum type
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
int main()
{
    days_of_week day1, day2; //define variables
    //of type days_of_week
    day1 = Mon; //give values to
    day2 = Thu; //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;
}
```

ပရိုဂရမ်းမင်းတိုင်းမှာ ကျွန်တော်တို့ သတ်မှတ်ချင်တဲ့အရာတွေကို များသောအားဖြင့် int တန်ဖိုး တစ်ခုခု သတ်မှတ်ပေးပြီး ကွန်ပျူတာကို ပြောပြလေ့ ရှိပါတယ်။ ဥပမာ Sunday ဆိုတာကို ကွန်ပျူတာ နားလည်နိုင်ဖို့ int Sunday=0; စသဖြင့် သတ်မှတ်လေ့ ရှိပါတယ်။ ဒါကြောင့် နောက်ပိုင်း 0 လို့

ပြောတိုင်း ကွန်ပျူတာက Sunday လို့ လက်ခံပြီး အလုပ်လုပ်ပေးပါတယ်။ ဒါပေမယ့် အဲဒီလို အချက်အလက်တွေ များလာရင် ကွန်ပျူတာက မှတ်မိပေမယ့် လူကတော့ လိုက်မမှတ်နိုင်တော့ပါဘူး။ တခါတလေ အယူအဆတွေ လွဲမှားသွားနိုင်ပါတယ်။ Enumeration ကို အသုံးပြုခြင်းအားဖြင့် ကျွန်တော်တို့ ထားချင်တဲ့ သတ်မှတ်ချက်တွေကို မူရင်း အခေါ်အဝေါ်အတိုင်း ရေးသား အသုံးပြုနိုင်တာကြောင့် အဲဒီ အခက်အခဲကို ကျော်လွှားနိုင်ပါတယ်။ အထက်ပါ ပရိုဂရမ်လေးမှာပါတဲ့ days_of_week ဆိုတဲ့ enum type မှာ Sun, Mon, Tues, Wed, Thu, Fri, Sat စသဖြင့် enumerators (၇) ခု ပါဝင်ပြီး သူတို့ရဲ့ syntax ကို အောက်မှာ လေ့လာနိုင်ပါတယ်။



Enumeration ဆိုတာ ဖြစ်နိုင်ခြေ ရှိတဲ့ တန်ဖိုးအားလုံးကို စုစည်းပြီး စာရင်းပြုစုထားတာပဲ ဖြစ်ပါတယ်။ အဲဒီ တန်ဖိုး တစ်ခုချင်းစီကိုလည်း သီးသန့် အမည်များ ပေးထားရပါတယ်။ အထက်မှာ ဖော်ပြထားသလို enum တစ်ခုကို declare လုပ်ပြီးတာနဲ့ အဲဒီ type အမျိုးအစား variables တွေကို define လုပ်နိုင်ပြီ ဖြစ်ပါတယ်။ dayenum.cpp မှာ days_of_week day1, day2; ဆိုပြီး day1 နဲ့ day2 variables တွေကို define လုပ်ပြထားပါတယ်။ (C မှာတုန်းကတော့ enum ဆိုတဲ့ keyword ကို define လုပ်တဲ့ အခါမှာလည်း ထည့်ပေးရပါတယ်။ ဥပမာ- enum days_of_week day1, day2; ဒါပေမယ့် C++ မှာတော့ ထည့်ပေးစရာ မလိုတော့ပါဘူး။)



day1 နဲ့ day2 လိုမျိုး enumerated type တွေကို သက်ဆိုင်ရာ enum declaration မှာ ပါဝင်တဲ့ တန်ဖိုးတွေထဲက တစ်ခုခုကို သတ်မှတ်ပေးနိုင်ပါတယ်။ ဒီဥပမာမှာတော့ Mon နဲ့ Thu ဆိုပြီး ပေးထားခဲ့ပါတယ်။ Declaration ထဲမှာ မပါတဲ့ တန်ဖိုးတွေကိုတော့ သတ်မှတ် မပေးနိုင်ပါဘူး။ ဥပမာ day1 = halloween; ဆိုပြီး ပေးလို့ မရပါဘူး။ ကျွန်တော်တို့အနေနဲ့ enumerated type တွေကို standard arithmetic operators (+, -, *, /) တွေကို သုံးပြီး အပေါင်း၊ အနုတ်၊ အမြောက်၊ အစား လုပ်နိုင်ပါတယ်။ ပရိုဂရမ်ထဲမှာ တန်ဖိုးနှစ်ခုရဲ့ ခြားနားခြင်းကို ရှာပြထားပါတယ်။ Comparison operators (<, >, <=, >=, !) တွေကို အသုံးပြုပြီး တစ်ခုနဲ့ တစ်ခုကိုလည်း နှိုင်းယှဉ်နိုင်ပါတယ်။ ပရိုဂရမ်ကို run ကြည့်မယ်ဆိုရင် အောက်ပါ ရလဒ်တွေကို တွေ့ရမှာ ဖြစ်ပါတယ်။

Days between = 3

day1 comes before day2

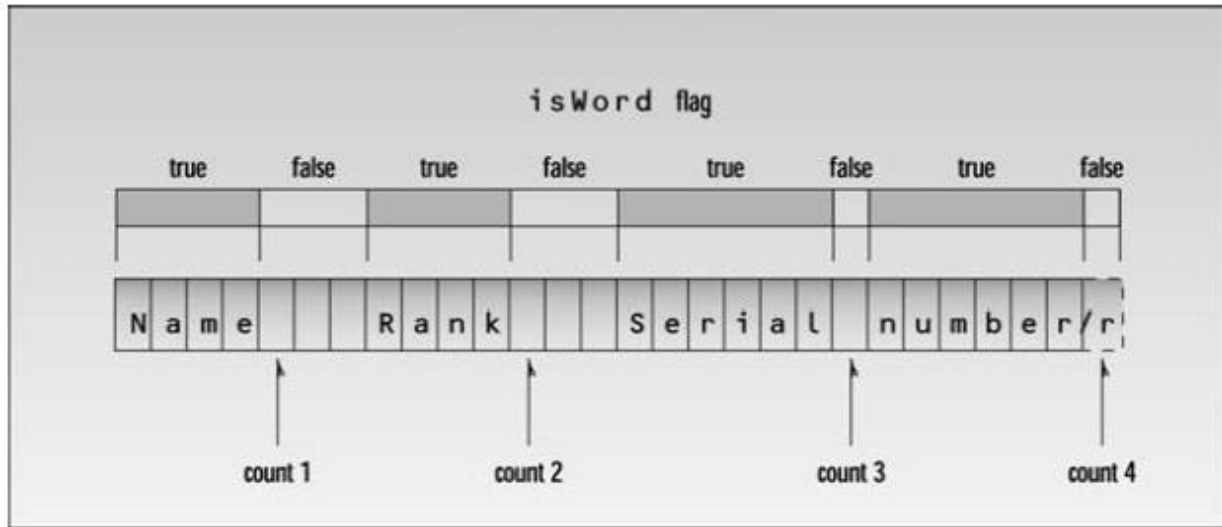
Enum types တွေကို Arithmetic နဲ့ relational operators တွေသုံးလို့ရပေမယ့် တစ်ချို့နေရာတွေမှာ အဓိပ္ပါယ်မရှိတာကြောင့် အသုံးမပြုသင့်ပါဘူး။ ဥပမာ အနေနဲ့ အောက်က enum type declaration မှာ enum pets { cat, dog, hamster, canary, ocelot }; ဆိုပြီး ကြေငြာထားတယ် ဆိုပါစို့။ ဒီနေရာမှာ (dog + canary) နဲ့ (cat < hamster) ဆိုတဲ့ statements တွေဟာ အဓိပ္ပါယ် မရှိပါဘူး။

Enum types တွေကို arithmetic နဲ့ relational operations တွေ လုပ်ဆောင်နိုင်တာဟာ တကယ်တော့ သူတို့ကို integers တွေ အနေနဲ့ internally သတ်မှတ်ယူဆထားလို့ပဲ ဖြစ်ပါတယ်။ ဘာမှ သတ်မှတ် မပေးခဲ့ဘူးဆိုရင် ပထမဆုံး တန်ဖိုးကို 0 သတ်မှတ်ပေးမှာ ဖြစ်ပါတယ်။ နောက်က တန်ဖိုးတွေကို တစ် တိုးပြီး 1, 2, 3, .. စသဖြင့် သတ်မှတ်ပေးမှာပါ။ ဒါကြောင့် dayenum.cpp ထဲက Sun ကနေ Sat အထိ တန်ဖိုးတွေဟာ တကယ်တော့ integer တန်ဖိုး 0-6 ပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် enum types တွေကို arithmetic operation တွေ ပြုလုပ်တဲ့ အခါမှာ အဲဒီ integer တန်ဖိုးတွေကို အသုံးပြုပြီး လုပ်ဆောင်တာ ဖြစ်ပါတယ်။

ကွန်ပျူတာက enum တွေကို integers တွေ အနေနဲ့ သိပေမယ့် ကျွန်တော်တို့ကတော့ အဲဒီအချက်ကို သတိထားပြီး အသုံးပြုရမှာ ဖြစ်ပါတယ်။ day1 = 5; ဆိုပြီး ရေးခဲ့မိရင် compiler က compile လုပ်ပေးပေမယ့် warning message ထုတ်ပေးမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် တတ်နိုင်သမျှ enums တွေဟာ (တကယ်တော့) integers တွေ ဖြစ်တယ်ဆိုတဲ့ အချက်ကို မေ့ထားရမှာပါ။

One Thing or Another

နောက်ထပ် ဥပမာအနေနဲ့ user မှ ရိုက်ထည့်လိုက်တဲ့ စကားလုံးတွေကို ရေတွက်တဲ့ ပရိုဂရမ်လေး တစ်ပုဒ်ကို ထပ်မံလေ့လာမှာ ဖြစ်ပါတယ်။ ယခင် ဥပမာ တစ်ခုဖြစ်တဲ့ chcount.cpp မှာ spaces များကို ရေတွက်ခြင်းဖြင့် စကားလုံး အရေအတွက်ကို ခန့်မှန်းတွက်ချက်ခဲ့ပါတယ်။ တကယ်လို့ user က space တစ်ခုထက်ပိုပြီး ရေးသားခဲ့မယ်ဆိုရင် စာလုံးအရေအတွက် မှားယွင်းသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ယခု ပရိုဂရမ်မှာ space မဟုတ်တဲ့ အက္ခရာတွေအဆုံးမှာ space ကို တွေ့မှ စကားလုံးကို ရေတွက်တဲ့ စနစ်ကို အသုံးပြုထားပါတယ်။ အဲဒီနည်းစနစ်ကို အောက်က ပုံလေးမှာ ရှင်းပြထားပါတယ်။



ဒါကြောင့် စကားလုံးတွေကြားမှာ spaces တွေ အမြောက်အများပါနေတာကြောင့် ရေတွက်မှု မှားယွင်းတာမျိုး ရှိတော့မှာ မဟုတ်ပါဘူး။ wdcount.cpp မှာ အဲဒီလို ပြုလုပ်ဖို့ enumerators နှစ်ခု (NO, YES) သာပါတဲ့ enum (itsaWord) ကို အသုံးပြုထားပါတယ်။ လေ့လာကြည့်ကြရအောင် -

```
// wdcount.cpp
// demonstrates enums, counts words in phrase
#include <iostream>
#include <conio.h>
using namespace std;
#include <conio.h> //for getch()
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 = getch(); //get character
        if(ch==' ' || ch=='\r') //if white space,
        {
            if( isWord == YES ) //and doing a word,
```

```

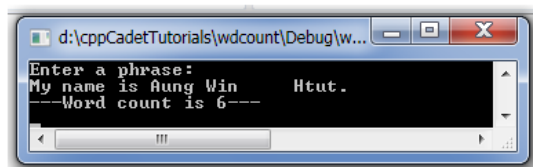
        { //then it's end of word
            wordcount++; //count the word
            isWord = NO; //reset flag
        }
    } //otherwise, it's
    else //normal character
    if( isWord == NO ) //if start of word,
        isWord = YES; //then set flag
} while( ch != '\r' ); //quit on Enter key
cout << "\n---Word count is " << wordcount << "---\n";
_getch();
return 0;
}

```

```

// demonstrates enums, counts words in phrase
#include <iostream>
#include <conio.h>
using namespace std;
#include <conio.h> //for getch()
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 = getch(); //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;
}

```



ပရိုဂရမ် အစမှာ do loop ကို အသုံးပြုပြီး keyboard က အက္ခရာတွေကို ဖတ်ရှုပါတယ်။ space မဟုတ်တဲ့ အက္ခရာတွေကို ကျော်သွားပြီး space ကို စတွေ့တာနဲ့ အရှေ့က ဖတ်ခဲ့တဲ့ အက္ခရာဟာ စကားလုံး ဟုတ်မဟုတ်ကို enum variable - isWord ကို YES ဖြစ်မဖြစ်နဲ့ စစ်ပြီး ဟုတ်ခဲ့ရင် cont ကို ၁ တိုးလိုက်ပါတယ်။ အဲဒီနောက် isWord ကို NO ပြောင်းလိုက်ပါတယ်။ ဒါမှ နောက်ထပ် space တွေ ကပ်လျက် ရှိနေခဲ့ရင် count မလုပ်မှာ ဖြစ်ပါတယ်။ enum - itsaWord မှာ ဖြစ်နိုင်ခြေရှိတဲ့ တန်ဖိုး နှစ်ခု (NO နဲ့ YES) ပဲ ရှိပါတယ်။ NO ကိုရှေ့ကထားတာ သတိထားမိပါလိမ့်မယ်။ ဒါကြောင့် သူ့ရဲ့တန်ဖိုးဟာ 0 (false ရဲ့ သင်္ကေတ) ဖြစ်ပါတယ်။ ဒါကြောင့် if(isWord == YES) အစား if(isWord) ဆိုပြီး စစ်ဆေးနိုင်ပါတယ်။ တခါတရံ အခြေအနေပေါ် မူတည်ပြီး အဲဒီနေရာမှာ enum အစား bool variable ကို သုံးနိုင်ပါသေးတယ်။

Organizing the Cards

အခု enum types တွေကို ရှင်းပြဖို့ နောက်ဆုံး ဥပမာလေး တစ်ခုကို လေ့လာကြရအောင်။ ယခင် သင်ခန်းစမှာ ပါတဲ့ cards.cpp ပရိုဂရမ်လေးကို မှတ်မိဦးမယ် ထင်ပါတယ်။ အဲဒီမှာ card suits တွေကို const int တွေသုံးပြီး အောက်ပါအတိုင်း သတ်မှတ်ခဲ့ပါတယ်။

```
const int clubs = 0;
```

```
const int diamonds = 1;
```

```
const int hearts = 2;
```

```
const int spades = 3;
```

ဒီလို ရေးနည်းထက် ပိုကောင်းတဲ့ နည်းလမ်းကို အောက်က cardenum.cpp မှာ လေ့လာကြရအောင် -

```
// 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;
}

```

}

ဒီပရိုဂရမ်မှာ suits တွေအတွက် definitions တွေကို enum အသုံးပြုပြီး အောက်ပါအတိုင်း declare လုပ်ထားပါတယ်။

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

ဒီနည်းဟာ const variables တွေ အသုံးပြုတာထက် ပိုပြီး ရှင်းလင်းပါတယ်။ တကယ်လို့ card1.suit = 5; ဆိုပြီး သုံးခဲ့ရင်တော့ ကွန်ပိုင်းလာက သတိပေးတာ ခံရမှာ ဖြစ်ပြီး မသုံးသင့်ပါဘူး။

Specifying Integer Values

ကျွန်တော်တို့အနေနဲ့ အစောပိုင်းမှာ ပထမဆုံး enumerator ကို 0 သတ်မှတ်ထားပြီး နောက်ထပ် enumerator တွေကို တစ်တိုးပြီး 1, 2, 3,.. စသဖြင့် သတ်မှတ်ပေးလိမ့်မယ်လို့ ပြောခဲ့ပါတယ်။ ဒါပေမယ့် ကိုယ်ကြိုက်နှစ်သက်ရာ တန်ဖိုးနဲ့ အစပြုဖို့အတွက်လည်း equal sign (=) ကို အသုံးပြုနိုင်ပါတယ်။ ဥပမာ suits တွေရဲ့တန်ဖိုးကို 0 နဲ့ အစပြုဘဲ 1 နဲ့ စချင်တယ်ဆိုပါစို့။

```
enum Suit { clubs=1, diamonds, hearts, spades };
```

ဆိုပြီး ရေးသားနိုင်ပါတယ်။ နောက်က လာတဲ့ တန်ဖိုးတွေကတော့ ထုံးစံအတိုင်း တစ်တိုးပြီး 2, 3, 4,.. စသဖြင့် ဖြစ်သွားမှာပါ။ တကယ်တမ်းမှာ တန်ဖိုးတစ်ခုချင်းစီကို equal sign (=) အသုံးပြုပြီး ထည့်သွင်းပေးလိုက်နိုင်ပါတယ်။

Not Perfect

enum ရဲ့အားနည်းချက်ကတော့ ၎င်းကို C++ ရဲ့ I/O statements တွေက နားမလည်တာပဲ ဖြစ်ပါတယ်။ ဥပမာ အောက်က ကုဒ်တွေကို run မယ်ဆိုပါစို့

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

```
direction dir1 = south;
```

```
cout << dir1;
```

ကျွန်တော်တို့ကတော့ output ဟာ south ဖြစ်လိမ့်မယ်လို့ ထင်ကြမှာပါပဲ။ ဒါပေမယ့် C++ I/O က enum variables တွေကို integers တွေအနေနဲ့ပဲ ကိုင်တွယ်တာ ဖြစ်လို့ output ကို 1 အဖြစ်ပဲ ထုတ်ပေးမှာပါ။

Other Examples

နောက်ထပ် enum အသုံးပြုနည်းများကို အောက်မှာ ဥပမာလေးများနဲ့ ပြသပေးထားပါတယ်။ လေ့လာကြည့်ကြပါ။

```
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 };
```

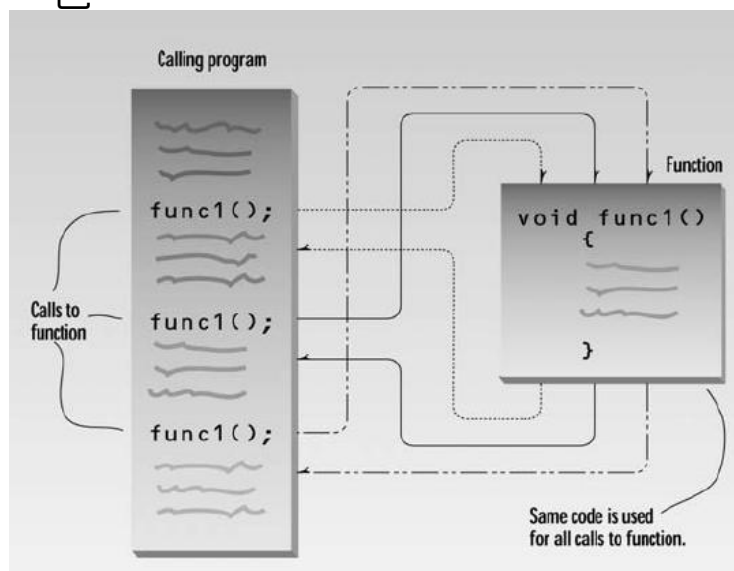
အခန်း(၄)

Functions

Functions

Function ဆိုသည်မှာ ပရိုဂရမ် statements ကုဒ် အချို့ကို ယူနစ် တစ်ခုအဖြစ် စုစည်းပြီး နာမည်တစ်ခု ပေးထားခြင်းပင် ဖြစ်သည်။ အဆိုပါ ယူနစ်များကို ပရိုဂရမ်၏ အခြားအစိတ်အပိုင်းများမှ ခေါ်ယူ အသုံးပြုနိုင်ပါသည်။ Functions များကို အသုံးပြုခြင်း၏ အဓိက အကြောင်းရင်းမှာ ပရိုဂရမ်တစ်ခုအား conceptual organization ပြုလုပ်ရာတွင် အထောက်အကူပြုရန်ပင် ဖြစ်သည်။ ထို့ပြင် ပရိုဂရမ် တစ်ခုအား functions များအဖြစ် ခွဲခြား ရေးသားခြင်းသည် structured programming ၏ အဓိက ဥပဒေ တစ်ခုလည်း ဖြစ်ပါသည်။ (သို့ရာတွင် object-oriented programming တွင် ၎င်းထက် ပိုမို အားကောင်းသော နည်းလမ်းများကို အသုံးပြုနိုင်ပါသည်။)

နောက်ထပ် ရည်ရွယ်ချက် တစ်ခုမှာ ပရိုဂရမ်၏ အရွယ်အစားကို လျော့ချရန် ဖြစ်သည်။ တစ်ကြိမ်ထက် ပိုသုံးထားသော ကုဒ်များကို function အဖြစ် ပြောင်းလဲ အသုံးပြုသင့်ပေသည်။ Functions များကို ပရိုဂရမ်အတွင်း အကြိမ်များစွာ ခေါ်ယူ အသုံးပြုလေ့ ရှိသော်လည်း function ၏ ကုဒ်များကိုမူ memory ၏ နေရာ တစ်ခုတည်းတွင်သာ သိမ်းဆည်းထားခြင်း ဖြစ်သည်။ ပရိုဂရမ်၏ အစိတ်အပိုင်း အသီးသီးမှ function တစ်ခုအား ခေါ်ယူ အသုံးပြုပုံကို အောက်ပါအတိုင်း သရုပ်ဖော်ပြသထားပါသည်။



C++ နှင့် C တွင် အသုံးပြုရသော functions များသည် အခြား ပရိုဂရမ်းမင်း ဘာသာစကားများရှိ subroutines၊ procedures များနှင့် သဘောတရားချင်း တူညီပြီး ရေးသားပုံ ဆင်တူပေသည်။

Simple Functions

ပထမဦးစွာ ကြယ် ၄၅ လုံး print ထုတ်ပေးသော function တစ်ခုကို စတင် လေ့လာကြရအောင်။ အောက်ပါ table.cpp ပရိုဂရမ်သည် ဇယားတစ်ခုကို ဖန်တီးပေးပြီး အဆိုပါ

ဇယားကို ပိုမို ဖတ်ရှုရလွယ်ကူစေရန် ကြယ်များဖြင့် ပြုလုပ်ထားသော လိုင်းများဖြင့် တားပေးထားပါသည်။

```
// 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;
}
```

ပရိုဂရမ်၏ output မှာ အောက်ပါအတိုင်း ဖြစ်ပါသည်။


```

*****
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
*****

```

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

အထက်ပါ ပရိုဂရမ်တွင် main() နှင့် starline() ဟူသော functions နှစ်ခုပါဝင်ပါသည်။ ကျွန်တော်တို့ အနေဖြင့် ယခင် သင်ခန်းစာများတွင် main() function တစ်ခုတည်းကိုသာ ကြိမ်ဖန်များစွာ တွေ့မြင်လာခဲ့ကြရပါသည်။ ထိုသို့ ပရိုဂရမ် တစ်ခုတွင် functions များ ထပ်မံပေါင်းထည့်ရန် လိုသည့် အချက် သုံးချက် ရှိပါသည်။ ၎င်းတို့မှာ function declaration, calls to the function နှင့် function definition များ ဖြစ်ကြပါသည်။

The Function Declaration

ကျွန်တော်တို့အနေဖြင့် compiler အား variable တစ်ခုကို ကြိုတင်ကြေငြာထားခြင်း မရှိဘဲ အသုံးပြုနိုင်သကဲ့သို့ function တစ်ခုကိုလည်း ကြိုတင်ကြေငြာပေးထားရန် လိုအပ်ပေသည်။ ထိုသို့ ပြုလုပ်နိုင်သည့် နည်းလမ်း နှစ်ခု ရှိပါသည်။ ပထမတစ်နည်းမှာ function အား မခေါ်ယူမီ declare ပြုလုပ်ခြင်းဖြစ်ပါသည်။ (ဒုတိယနည်းလမ်းမှာ function အားမခေါ်ယူမီ define ပြုလုပ်ခြင်း ဖြစ်ပြီး ၎င်းကို နောက်ပိုင်းတွင် ဆက်လက်လေ့လာသွားမည်။)

table.cpp ပရိုဂရမ်တွင် starline() ဆိုသော function ကို void starline(); ဆိုသော ကုဒ်ဖြင့် declare ကြေငြာပေးထားသည်။ ဤနည်းဖြင့် နောက်တစ်ချိန်တွင် starline ဟုခေါ်သော function တစ်ခုကို ခေါ်ယူရန် စီစဉ်ထားကြောင်း compiler အား အသိပေးလိုက်ခြင်းပင် ဖြစ်သည်။

ထိုသို့ကြေငြာရာတွင် ပါဝင်သော void ဆိုသည့် keyword သည် function မှ return value ပြန်ပေးမည် မဟုတ်ကြောင်း သတ်မှတ်ပေးထားခြင်းဖြစ်ပြီး လက်သဲကွင်းထဲတွင် အလွတ်ထားခြင်းသည် arguments ထည့်သွင်းပေးရန် မလိုကြောင်း (လက်ခံမည်မဟုတ်ကြောင်း) အသိပေးထားခြင်းပင် ဖြစ်သည်။ (နောက်တစ်နည်းမှာ လက်သဲကွင်းထဲတွင် void ထည့်ပေးခြင်းဖြစ်ပြီး C တွင် အသုံးများသည်။ သို့ရာတွင် C++ ၌မူ လက်သဲကွင်း အလွတ်ထားလေ့ရှိသည်။) နောက်ပိုင်းတွင် arguments များနှင့် return values များ အကြောင်းကို လေ့လာသွားမည်ဖြစ်သည်။ function declaration ကို semicolon (;) ဖြင့် အဆုံးသတ်လေ့ရှိပြီး ၎င်းစာကြောင်းသည် ပြီးပြည့်စုံသည့် statement တစ်ခုပင် ဖြစ်သည်။

Function declarations များသည် functions များ၏ model သို့မဟုတ် blueprint သဖွယ် ဖြစ်နေသဖြင့် ၎င်းတို့ကို prototypes များဟုလည်း ခေါ်ဆိုကြသည်။ ၎င်းတို့သည် ဤကဲ့သို့ ပုံစံရှိသော function တစ်ခုသည် ပရိုဂရမ် ၏ တစ်နေရာတွင် ရေးသားထားမည် ဖြစ်သောကြောင့် ၎င်းအား ခေါ်ဆို အသုံးချမှုများကို အဆိုပါ function အားရေးသားထားချက် မတိုင်မီ တွေ့ရှိရပါက နားလည်စေရန် compiler အား အကြောင်းကြားလိုက်ခြင်းပင် ဖြစ်သည်။ အဆိုပါ declaration တွင် ပါဝင်သော return type၊ arguments အရေအတွက်နှင့် types များ စသည့် အချက်အလက်များကို function ၏ signature များဟုလည်း ခေါ်ဆိုလေ့ ရှိသည်။

Calling the Function

အထက်ပါ ပရိုဂရမ်တွင် starline() function အား main() function အတွင်း သုံးကြိမ်ခေါ်ယူ အသုံးပြုထားပေသည်။ ထိုသို့ ခေါ်ယူရန် starline(); ဟူသော ကုဒ်ကို အသုံးပြုခဲ့သည်။ Function တစ်ခုအား ခေါ်ယူရန်မှာ အဆိုပါ function ၏ အမည်နောက်တွင် လက်သဲကွင်းတစ်ခု နှင့် ၎င်းတို့ကို semicolon ဖြင့် ပိတ်ပေးရန်သာ ဖြစ်သည်။ Function call ၏ syntax သည် function declaration ၏ syntax နှင့် ဆင်တူပြီး return type မပါခြင်းသာ ကွာခြားမှု ရှိသည်။

void starline(); //function declaration

starline(); //function calling

Call statement ကို လုပ်ဆောင်သည့်အခါ control သည် function သို့ ရောက်ရှိသွားပြီး (အောက်တွင် ဆက်လက် ရှင်းပြသွားမည့်) function definition ရှိ ကုဒ်များကို လုပ်ဆောင်မည်ဖြစ်သည်။ ထို့နောက် control သည် function call ၏ နောက်တွင် ရှိသော ကုဒ်များသို့ ပြန်လည်ရောက်ရှိသွားမည် ဖြစ်သည်။

The Function Definition

Function တစ်ခုအတွက် တကယ့်ကုဒ်များ ရေးသားသည့် နေရာကို function definition ဟုခေါ်သည်။ အောက်တွင် starline() function ၏ definition ကို လေ့လာကြည့်ရအောင်-

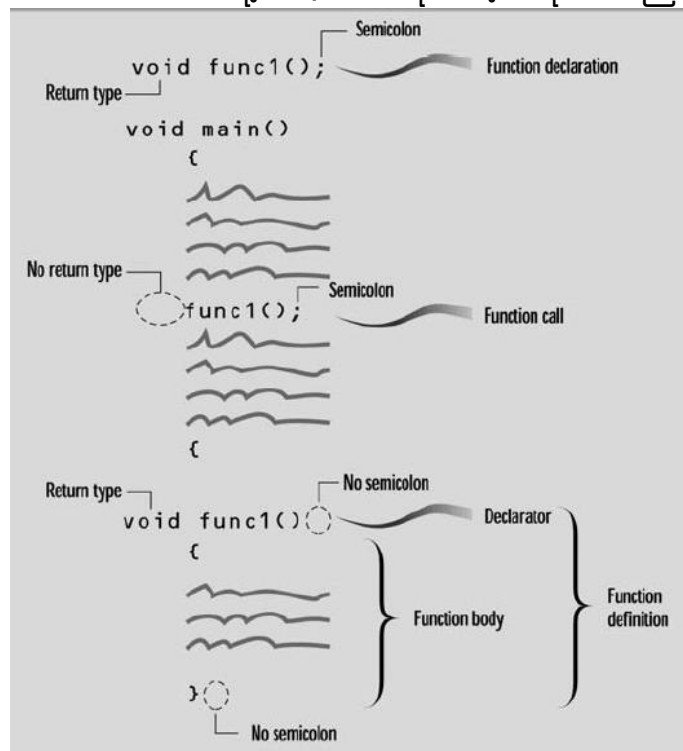
```
void starline() //declarator
{
    for(int j=0; j<45; j++) //function body
        cout << '*';
    cout << endl;
}
```

Function definition တွင် declarator ဟုခေါ်သော စာကြောင်းနှင့် function body ဟူ၍ ပါဝင်သည်။ Function body သည် function ၏ ကုဒ်များ ပါဝင်ပြီး တွန့်ကွင်း အဖွင့်အပိတ်ဖြင့် သတ်မှတ်ထားသည်။

The declarator တွင် ပါဝင်သော အချက်များသည် declaration နှင့် ကိုက်ညီမှု ရှိရမည် ဖြစ်သည်။ အောက်ပါအချက်များ တူညီရမည်ဖြစ်သည်-

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

declarator အား semicolon လုံးဝ မထည့်ပေးရပေ။ အောက်ပါပုံတွင် function declaration, function call နှင့် function definition တို့၏ syntax ကို လေ့လာနိုင်ပါသည်။



Function တစ်ခုအား ခေါ်ယူလိုက်သည့်အခါ control သည် function body ရှိ ပထမဆုံး ကုဒ်ကို ကူးပြောင်းသွားပြီး စတင်လုပ်ဆောင်မည် ဖြစ်သည်။ function body ရှိ ကုဒ်များ အားလုံး

လုပ်ဆောင်ပြီး၍ တွန့်ကွင်း အပိတ်အား တွေ့လိုက်သည့်အချိန်တွင် control သည် calling ပြုလုပ်သည့် နေရာသို့ ပြန်ရောက်သွားမည် ဖြစ်သည်။

အောက်က ဇယားမှာ function တစ်ခု ရေးသားဖို့ လိုတဲ့ အစိတ်အပိုင်းများကို ရှင်းပြပေးထားပါတယ်။

Function Components		
Component	Purpose	Example
Declaration (prototype)	Specifies function name, argument types, and return value. Alerts compiler (and programmer) that a function is coming up later.	<code>void func();</code>
Call	Causes the function to be executed.	<code>func();</code>
Definition	The function itself. Contains the lines of code that constitute the function.	<code>void func() { // lines of code }</code>
Declarator	First line of definition.	<code>void func()</code>

Comparison with Library Functions

ကျွန်တော်တို့ အနေနဲ့ library functions အချို့ကို အစောပိုင်း သင်ခန်းစာများမှာ အသုံးပြုခဲ့ကြပြီး ဖြစ်ပါတယ်။ ဥပမာ `ch=getche();` ဆိုတဲ့ ကုဒ်မျိုးပဲ ဖြစ်ပါတယ်။ ဒါဆိုရင် အဲဒီ library function `getche()` အတွက် declaration နဲ့ definition တွေက ဘယ်မှာပါလဲ? တကယ်တော့ အဲဒီလို library functions တွေ အတွက် declaration တွေကို ပရိုဂရမ်တွေရဲ့ အစမှာ ပါတဲ့ header file (`conio.h`, for `getche()`) တွေထဲမှာ သတ်မှတ်ပေးထားတာ ဖြစ်ပါတယ်။ သူတို့အတွက် definition တွေကတော့ executable code တွေဖြစ်အောင် compile လုပ်ပေးထားပြီး library file တွေထဲမှာ ထည့်သွင်းထားကာ ကျွန်တော်တို့ ရေးသားထားတဲ့ C++ source file ကို build လုပ်တဲ့အချိန်ကျမှ compiler က အလိုအလျောက် ချိတ်ဆက်ပြီး link ပြုလုပ်ပေးမှာ ဖြစ်ပါတယ်။

ဒါကြောင့် library function တွေကို အသုံးပြုတဲ့ အခါမှာ declaration နဲ့ definition တွေကို ကျွန်တော်တို့ ကိုယ်တိုင် ရေးသားပေးစရာ မလိုအပ်ဘဲ ပရိုဂရမ် အစပိုင်းမှာ သက်ဆိုင်ရာ header file ကို ကြေငြာပေးပြီး သုံးစွဲနိုင်တာ ဖြစ်ပါတယ်။ ဒါပေမယ့် ကျွန်တော်တို့ ကိုယ်ပိုင် functions တွေ ရေးသားတဲ့ အခါမှာတော့ declaration နဲ့ definition တွေကို ယခင် အပတ်က ဥပမာ ပရိုဂရမ်တွေထဲကလိုမျိုး ထည့်သွင်းရေးသားပေးရမှာပါ။

Eliminating the Declaration

နောက်ထပ် function ရေးနည်း တစ်မျိုးကတော့ function call မတိုင်မီ function declaration ကို မရေးတော့ပဲ အဲဒီနေရာမှာ function definition ကို တန်းရေးလိုက်တာပဲ ဖြစ်ပါတယ်။ အောက်က `table2.cpp` ပရိုဂရမ်လေးမှာ `starline()` ကို ပြန်ရေးပြထားပါတယ်။

```
// 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;
}

```

ဒီနည်းက function declaration ကို ဖြုတ်ပစ်လိုက်လို့ ပရိုဂရမ် တိုတိုလေးတွေ ရေးဖို့အတွက် ရိုးရှင်းတဲ့ နည်းလမ်း တစ်ခုပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် ဒီနည်းက functions တွေ အများအပြားရှိလာတဲ့ အခါမှာ function တစ်ခုကနေ နောက်တစ်ခုကို ခေါ်ယူနိုင်တာ ဖြစ်တဲ့အတွက် အစဉ်အတိုင်း ဖြစ်အောင် စဉ်းစားရတဲ့ အတွက် အဆင်မပြေတော့ပါဘူး။ တစ်ခါတလေမှာ မဖြစ်နိုင်တဲ့ အခြေအနေမျိုးကိုတောင် ကြုံရတတ်ပါတယ်။ ဒါ့အပြင် ပရိုဂရမ်မှာ အများစုဟာ ပထမဆုံး အလုပ်လုပ်မယ့် main() function ကို အစမှာ ထားချင်ကြပါတယ်။ ဒါကြောင့် declarations တွေ အသုံးပြုရတဲ့ ပထမနည်းလမ်းကိုပဲ အသုံးများကြပါတယ်။

Passing Arguments to Functions

Argument ဆိုသည်မှာ ပရိုဂရမ်မှ function သို့ ထည့်သွင်းပေးလိုက်သော data အစိတ်အပိုင်း တစ်ခုဖြစ်သည်။ ထိုသို့ argument များ ထည့်သွင်းပေးခြင်းဖြင့် function တစ်ခုကို values အမျိုးမျိုး ဖြင့် လုပ်ဆောင်နိုင်ရန် သို့မဟုတ် ပရိုဂရမ်၏ လိုအပ်ချက်အရ မတူညီသော လုပ်ဆောင်ချက်များကို လုပ်ဆောင်နိုင်ရန် ဆောင်ရွက်ပေးသည်။

Passing Constants

အထက်က ဥပမာတွင် အသုံးပြုခဲ့သော starline() ဆိုသည့် function သည် ကြယ် ၄၅ လုံးကို အမြဲတမ်း print ထုတ်ပေးသဖြင့် လိုသလို ပြောင်းလဲ အသုံးပြုနိုင်စွမ်း နည်းပါးသည်။ ကျွန်တော်တို့ အနေဖြင့် ထိုကဲ့သို့ အသေ ရေးသားထားသော function များထက် အလိုရှိသော character ကို အလိုရှိသော အရေအတွက် print ထုတ်ပေးမည့် function မျိုးကို ပိုမိုလိုအပ်ပေသည်။ အောက်ပါ tablearg.cpp ပရိုဂရမ်တွင် ထိုကဲ့သို့သော function မျိုးကို ရေးပြထားပါသည်။ အလိုရှိသော character နှင့် အလိုရှိသော အကြိမ်အရေအတွက်ကိုမူ arguments များအနေဖြင့် ထည့်သွင်းပေးရမည် ဖြစ်သည်။

```
// 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 ပရိုဂရမ်လေးမှာ သုံးထားတဲ့ function က repchar() ဖြစ်ပြီး အောက်ပါအတိုင်း declare လုပ်ထားပါတယ် -

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

Declaration မှာ လက်သဲကွင်းထဲက char နဲ့ int ဆိုတာကတော့ အဲဒီ function ထဲကို ထည့်သွင်းပေးလိုက်မယ့် data တွေရဲ့ data types တွေပဲ ဖြစ်ပါတယ်။

Function call ပြုလုပ်တဲ့ အခါမှာတော့ သတ်မှတ်ထားတဲ့ တန်ဖိုးတွေ (ဒီနေရာမှာတော့ constans တွေ) ကို လက်သဲကွင်းရဲ့ သက်ဆိုင်ရာနေရာတွေမှာ ကော်မာ ခံပြီး ထည့်သွင်းပေးလိုက်မှာပါ။ ဥပမာ-

repchar('-', 43); // function call specifies actual values

ဒီ statement လေးက repchar() ကို dash အလုံးရေ ၄၃ လုံး print ထုတ်ပေးဖို့ ညွှန်ကြားလိုက်တာပဲ ဖြစ်ပါတယ်။ Function call မှာ ထည့်သွင်းပေးလိုက်တဲ့ data တွေရဲ့ type ဟာ declaration မှာ သတ်မှတ်ပေးထားတဲ့ အတိုင်း ဖြစ်ဖို့တော့ လိုပါတယ်။ ဒါကြောင့် ပထမဆုံး ထည့်သွင်းပေးရမယ့် argument ဟာ char ဖြစ်ရမှာ ဖြစ်ပြီး နောက်တစ်လုံးကတော့ int ဖြစ်ရပါမယ်။ နောက်ထပ် function call တစ်ခု ဖြစ်တဲ့ repchar('=', 23); ကတော့ equal signs ၂၃ ခု ထုတ်ပေးဖို့ ညွှန်ကြားချက် ဖြစ်ပါတယ်။ tablearg.cpp ရဲ့ output ကို လေ့လာကြည့်ကြရအောင်။

```
-----
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 တွေကို ခေါ်ယူတဲ့ ပရိုဂရမ်က လိုအပ်တဲ့ arguments တွေ (ဥပမာ '-' နဲ့ 43) တွေကို ထည့်သွင်းပေးလိုက်ရမှာ ဖြစ်ပါတယ်။ ၎င်းတန်ဖိုးတွေကို function ထဲက အသုံးပြုထားတဲ့ parameters တွေလို့ ခေါ်တဲ့ variables တွေ (ဥပမာ ch နဲ့ n) ထဲကို ကူးယူလိုက်မှာ ဖြစ်ပါတယ်။ (ပရိုဂရမ်မှာ အများစုဟာ argument နဲ့ parameter တွေကို အလဲအလှယ်လုပ်ပြီး ပြောဆို သုံးစွဲတတ်ကြပါတယ်။)

Function definition မှာတော့ လက်သဲကွင်းထဲမှာ parameters တွေရဲ့ data types တွေအပြင် names တွေကိုပါ ထည့်သွင်းပေးရမှာ ဖြစ်ပါတယ်။ ဥပမာ-

```
void repchar(char ch, int n) //declarator specifies parameter
//names and data types
```

အဲဒီ parameters တွေ (ch နဲ့ n) ကို function ထဲမှာ သာမန် variable တွေ သုံးသလိုပဲ သုံးစွဲနိုင်ပါတယ်။ ဆိုလိုတာက အဲဒီလို လက်သဲကွင်းထဲမှာ ရေးလိုက်ခြင်းဟာ အောက်ပါအတိုင်း ကြေငြာလိုက်တာနဲ့ အတူတူပဲ ဖြစ်ပါတယ်။

```
char ch;
```

```
int n;
```

Function ကို call လုပ်တဲ့အခါမှာတော့ အဲဒီ parameters တွေထဲကို calling program က ထည့်သွင်းပေးလိုက်တဲ့ တန်ဖိုးတွေ အလိုအလျောက် initialized လုပ်ပေးမှာ ဖြစ်ပါတယ်။

Passing Variables

ယခင် အပတ်မှာ လေ့လာခဲ့ကြတဲ့ tablearg.cpp ပရိုဂရမ်ထဲမှာ constant arguments ('-', 43) တွေကို အသုံးပြုခဲ့ပါတယ်။ ယခုအပတ်မှာတော့ constants တွေအစား variables တွေကို arguments များ အဖြစ် အသုံးပြုခြင်းကို လေ့လာသွားမှာပါ။ အောက်ပါ vararg.cpp ပရိုဂရမ်လေးမှာတော့ tablearg.cpp မှာတုန်းကလိုပဲ repchar() function ပါဝင်မှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် တစ်ခု ပိုထူးခြားလာတာကတော့ user က ရွေးချယ်ပေးလိုက်တဲ့ character နဲ့ အကြိမ်အရေအတွက်ကို အသုံးပြုသွားမှာပဲ ဖြစ်ပါတယ်။

```
// vararg.cpp
```

```
// demonstrates variable arguments
```

```
#include <iostream>
```

```
using namespace std;
```

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

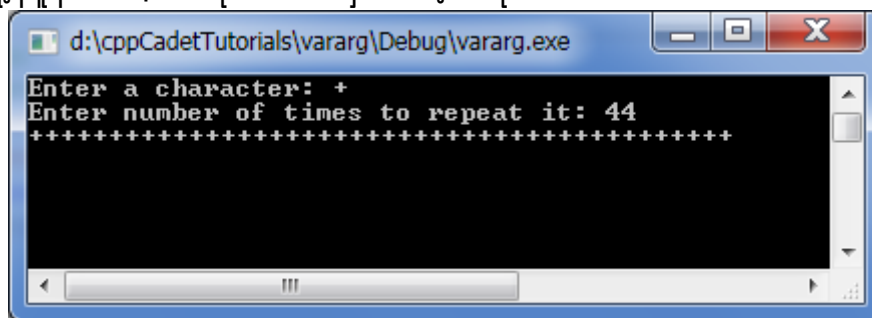


```

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 ရဲ့နမူနာ output ကို အောက်မှာ လေ့လာနိုင်ပါတယ်။



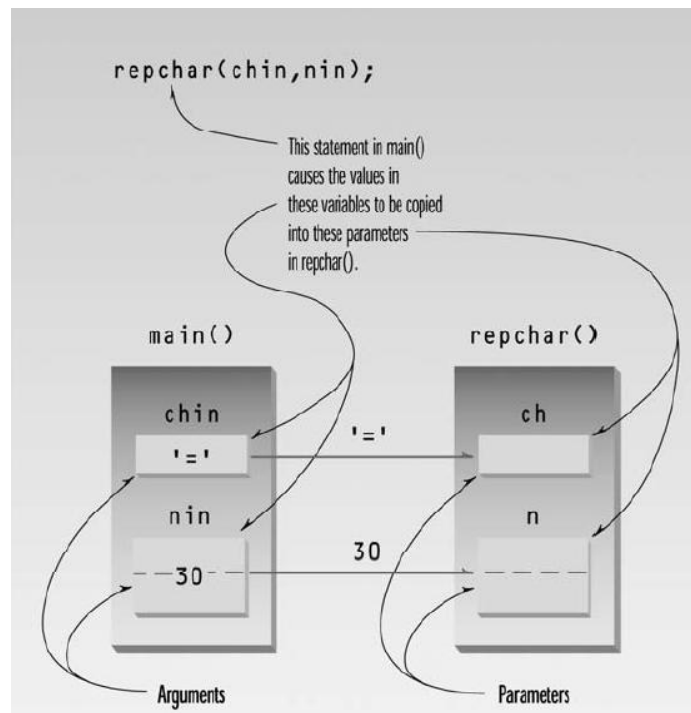
repchar() အတွက် input arguments များအနေနဲ့ main() ထဲက chin နဲ့ nin တို့ကို အောက်ပါအတိုင်း အသုံးပြုထားပါတယ်။

repchar(chin, nin); // function call

Variables တွေအတွက် အသုံးပြုထားတဲ့ arguments တွေရဲ့ data types တွေဟာ constants တွေ အသုံးပြုစဉ်ကလိုပဲ function declaration နဲ့ definition တွေမှာ တူညီနေဖို့ လိုအပ်ပါတယ်။ ဒါကြောင့် chin ဟာ char ဖြစ်ဖို့လိုပြီး nin ကတော့ int ဖြစ်ဖို့ လိုပါတယ်။

Passing by Value

vararg.cpp ပရိုဂရမ်မှာ chin နဲ့ nin အတွင်း ရှိတဲ့ တန်ဖိုးများကို function call ပြုလုပ်တဲ့ အခါမှာ function ထဲကို pass ပြုလုပ်ပေးမှာ ဖြစ်ပါတယ်။ function ထဲတွင် ၎င်း တန်ဖိုးများကို သိမ်းဆည်းအသုံးပြုရန် variables အသစ်များ ဖန်တီးပေးမည် ဖြစ်သည်။ ထိုသို့ ပြုလုပ်ရန် function ၏ declarator ဖြစ်သော void repchar(char ch, int n) ထဲတွင် သတ်မှတ်ပေးထားသော data type များနှင့် အမည်များ (char ch, int n) ကို အသုံးပြုထားခြင်းပင် ဖြစ်သည်။ function အတွင်း pass ပြုလုပ်လိုက်သော တန်ဖိုးများကို အဆိုပါ parameters များအတွင်း initialize ပြုလုပ်၍ ထည့်သွင်းပေးလိုက်ခြင်းပင် ဖြစ်သည်။ ထို့နောက်တွင်တော့ ၎င်း တန်ဖိုးများကို ပုံမှန် variables များ အသုံးပြုသည့်အတိုင်း function body အတွင်း အသုံးပြုနိုင်ပြီ ဖြစ်သည်။ ထိုကဲ့သို့ passing ပြုလုပ်လိုက်သော arguments များကို function အတွင်း ကူးယူဖန်တီးသည့် နည်းလမ်းကို passing by value ဟု ခေါ်ဆိုကြသည်။ နောက်ထပ် နည်းလမ်းတစ်ခုမှာ passing by reference ဟု ခေါ်ပြီး နောက်ပိုင်း သင်ခန်းစာများတွင် ဆက်လက် လေ့လာသွားကြမည် ဖြစ်သည်။ Passing by value ဖြစ်စဉ်ကို အောက်ပါပုံတွင် အသေးစိတ် လေ့လာနိုင်ပါသည်။



Structures as Arguments

function အတွင်းသို့ တန်ဖိုးများကို တစ်ခုချင်း ထည့်သွင်းပေးနိုင်သလို structures များကို အသုံးပြု၍ အစုလိုက် အပြုံလိုက်လည်း ထည့်သွင်းပေးနိုင်ပေသည်။ နမူနာ ပရိုဂရမ်များဖြင့် ဆက်လက်၍ လေ့လာကြရအောင်။

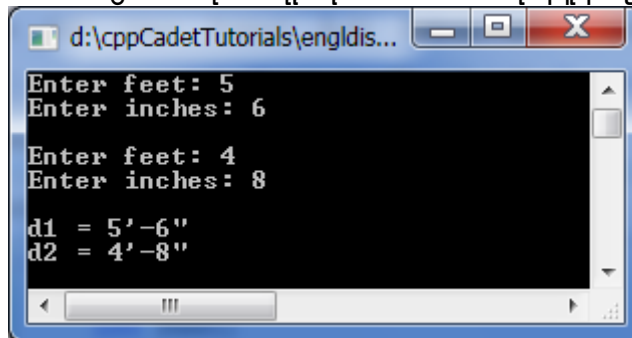
Passing a Distance Structure

အောက်ပါ ဥပမာတွင် ယခင်က လေ့လာခဲ့ဖူးသော Distance structure type ကို input argument အနေဖြင့် အသုံးပြုထားသည်။ engldisp.cpp ကို လေ့လာကြည့်ကြရအောင် -

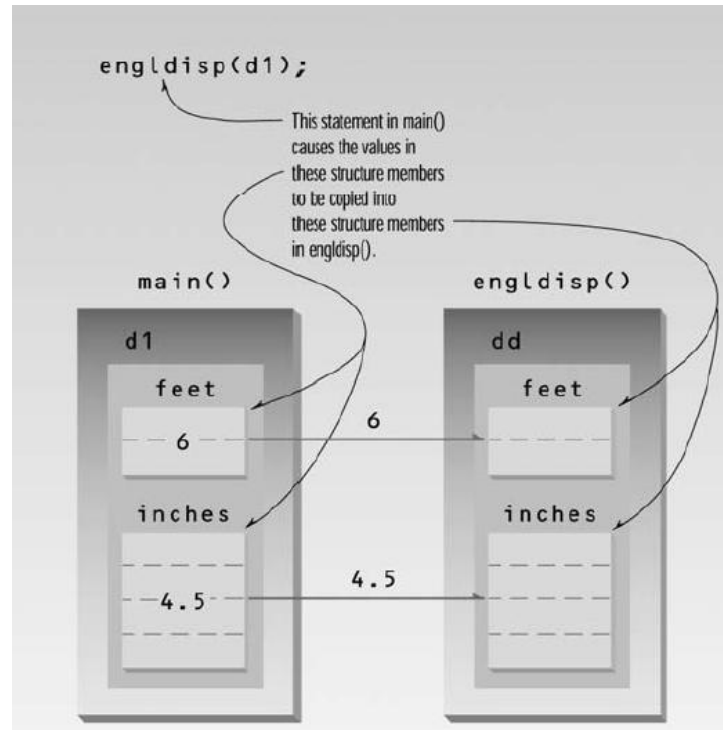
```
// engldisp.cpp
// demonstrates passing structure as argument
#include <iostream>
using namespace std;
/////////////////////////////////////////////////////////////////
struct Distance //English distance
{
    int feet;
    float inches;
};
/////////////////////////////////////////////////////////////////
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 ထဲတွင် user ထည့်သွင်းလိုက်သော distances တန်ဖိုးများကို feet-and-inches ပုံစံဖြင့် လက်ခံပြီး d1 နှင့် d2 ဟူသော structure နှစ်ခုအတွင်း ထည့်သွင်းလိုက်သည်။ ထို့နောက် engldisp() function ကို ခေါ်ယူကာ d1 နှင့် d2 ကို input argument များအဖြစ် ထည့်သွင်းပေးလိုက်သည်။ ၎င်း function ၏ ရည်ရွယ်ချက်မှာ စံပြပုံစံ ဖြစ်သော 10'-2.25" ကဲ့သို့ ပြသပေးရန် ဖြစ်သည်။ အောက်တွင် အဆိုပါ ပရိုဂရမ်၏ output ကို နမူနာ ပြသထားသည်။



main() အတွင်းရှိ function calls များ၊ function declaration များ နှင့် function body ရှိ declarator များသည် input ထည့်သွင်းပေးလိုက်သော Distance type structure variables များကို အခြား int, char တို့ကဲ့သို့သော argument variables များနည်းတူ ပြုမူဆောင်ရွက်ပေးသည်။ main() function ထဲတွင် engldisp() function ကို နှစ်ကြိမ်ခေါ်ယူထားသည်။ ပထမတစ်ကြိမ်တွင် structure d1 အား ထည့်သွင်းပေးပြီး ဒုတိယအကြိမ်တွင် d2 အားထည့်သွင်းပေးသည်။ engldisp() function ထဲတွင် Distance type structure - dd ကို parameter အဖြစ် အသုံးပြုထားပြီး main() function မှ ထည့်သွင်းပေးလိုက်သော structure value များကို သာမန် argument value များကဲ့သို့ပင် dd ထဲသို့ အလိုအလျောက် ထည့်သွင်းပေးသည်။ ထို့ကြောင့် engldisp() function ထဲမှ statements များသည် dd.feet နှင့် dd.inches ဆိုသည့် ပုံမှန် structure အသုံးပြုပုံ နည်းလမ်းများအတိုင်း အဆိုပါ value များကို ရယူ အသုံးပြုနိုင်ပေသည်။ အောက်ပါပုံတွင် function တစ်ခုအတွင်းသို့ structure တစ်ခုအား argument အနေဖြင့် ထည့်သွင်းအသုံးပြုပုံကို ရှင်းလင်းထားသည်။



ပုံမှန် variables များကဲ့သို့ပင် `engldisp()` function အတွင်းရှိ structure parameter - `dd` ကို ပြောင်းလဲခြင်းသည် ၎င်းအား `main()` function မှ ထည့်သွင်းပေးလိုက်သော structures - `d1` နှင့် `d2` တို့အား ပြောင်းလဲစေခြင်း မရှိပေ။ သီးသန့် ဖြစ်နေပေသည်။ ဥပမာ -

`dd.feet = 2;`

`dd.inches = 3.25;`

တို့သည် `main()` အတွင်းရှိ `d1` နှင့် `d2` တို့၏ တန်ဖိုးများအား မပြောင်းလဲ စေနိုင်ပေ။

Passing a circle Structure

အခုတစ်ပတ်မှာတော့ Console Graphics Lite functions များကို အသုံးပြုပြီး ရေးသားထားတဲ့ ဥပမာလေးကို လေ့လာသွားမှာ ဖြစ်ပါတယ်။ Console Graphics Lite functions ရဲ့ ကုဒ်များကို ဂရပ်ဖစ်ပိုင်းဆိုင်ရာ နမူနာ ပရိုဂရမ်များမှာ ထည့်သွင်းအသုံးပြုနိုင်အောင် ဆိုတဲ့ ရည်ရွယ်ချက်နဲ့ Robert Lafore ရေးသားတဲ့ **Object-Oriented Programming in C++(4th edition)** ရဲ့ Appendix E မှာ ထည့်သွင်းပေးထားတာပဲ ဖြစ်ပါတယ်။ နောက်ပိုင်းမှာတော့ အဲဒီ function များကို အသေးစိတ် ရှင်းလင်းပေးမှာ ဖြစ်ပေမယ့် အခုအခန်းမှာတော့ function တွေနဲ့ ပတ်သက်ပြီး အဓိကထားလေ့လာနေတာ ဖြစ်တဲ့အတွက် ဂရပ်ဖစ်အပိုင်းရေးသားထားချက်များကို အနည်းငယ်ပဲ ဆွေးနွေးသွားမှာပါ။ အဲဒီ Console Graphics Lite functions တွေကို အသုံးပြုဖို့ အတွက်တော့ အသုံးပြုတဲ့ compiler အပေါ်မူတည်ပြီး `msoftcon.h` ဒါမှမဟုတ် `borlancon.h` တစ်ခုခုကို include ပြုလုပ်ပေးဖို့ လိုအပ်ပါတယ်။ အောက်ပါ နမူနာ ပရိုဂရမ်လေးမှာတော့ function တစ်ခုထဲကို structure တစ်ခု ထည့်သွင်း အသုံးပြုပုံကို Console Graphics Lite functions တွေကို အသုံးပြုထားပြီး ရေးသားပြထားပါတယ်။ အဲဒီ နမူနာထဲမှာ စက်ဝိုင်းတစ်ခု ကို ကိုယ်စားပြုတဲ့ circle structure တစ်ခုကို

အသုံးပြုထားပါတယ်။ စက်ဝိုင်းတစ်ခုကို ဖော်ပြနိုင်ဖို့ အလယ်မှတ်နဲ့ radius တွေကို ထည့်သွင်းပေးရမှာပါ။ ဒါ့အပြင် စက်ဝိုင်းအတွက် အရောင်နဲ့ fill pattern တွေကိုလည်း ထည့်သွင်းထားနိုင်ပါတယ်။ အောက်ပါ circstrc.cpp ကို လေ့လာကြည့်နိုင်ပါတယ်။

```
// 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;
}

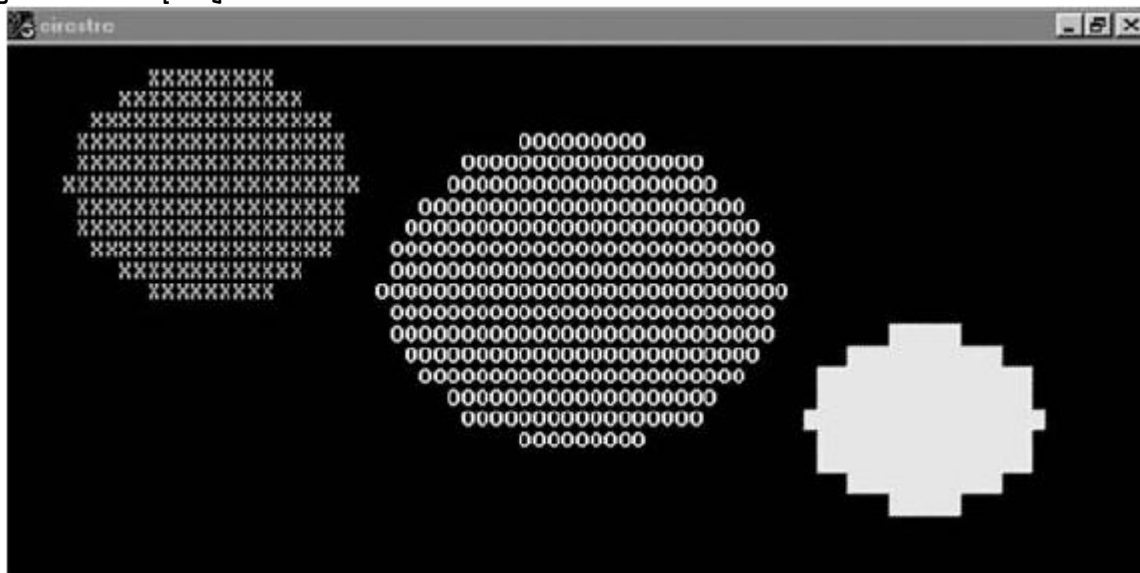
```

အထက်ပါ ဥပမာမှာ circle structure type အမျိုးအစား variables c1, c2 နဲ့ c3 တွေကို မတူညီတဲ့ တန်ဖိုးတွေ ထည့်သွင်းပြီး initialized လုပ်ထားပါတယ်။ ဥပမာအနေနဲ့ c1 ကို အောက်ပါအတိုင်း ဖန်တီးလိုက်ပါတယ်-

```
circle c1 = { 15, 7, 5, cBLUE, X_FILL };
```

ကျွန်တော်တို့အနေနဲ့ console screen ကို 80 columns နဲ့ 25 rows ရှိတယ်လို့ သတ်မှတ်လိုက်ပါတယ်။ အထက်က 15 ဟာ column number ဒါမှမဟုတ် x coordinate ဖြစ်ပြီး 7 ကတော့ row number ဒါမှမဟုတ် screen ရဲ့ထိပ်ဆုံးက နေအစပြုတဲ့ y coordinate ဖြစ်ပါတယ်။ အဲဒီ တန်ဖိုးနှစ်ခုက စက်ဝိုင်းရဲ့အလယ်မှတ်ကို ဖော်ပြပေးတာဖြစ်ပါတယ်။ နောက်က 5 ကတော့ စက်ဝိုင်းရဲ့ အချင်းဝက် တန်ဖိုး ဖြစ်ပါတယ်။ cBLUE ကတော့ စက်ဝိုင်းရဲ့အရောင်ကို ကိုယ်စားပြုထားပြီး X_FILL ကတော့ စက်ဝိုင်းကို ဖြည့်ပေးမယ့် fill pattern ပဲ ဖြစ်ပါတယ်။ စက်ဝိုင်းနှစ်ခုကို တန်ဖိုး အတူတူပေးပြီး ဖန်တီးလိုက်ပါတယ်။

စက်ဝိုင်းတွေ အားလုံးကို တန်ဖိုးတွေ သတ်မှတ်ပေးပြီးတာနဲ့ circ_draw() function ကို သုံးကြိမ်ခေါ်ယူပြီး စက်ဝိုင်းတွေကို Console မှာ ဆွဲပေးမှာ ဖြစ်ပါတယ်။ အောက်က ပုံလေးမှာတော့ အဲဒီ result တွေကို ပြသထားပါတယ်။ ပုံတွေက ခပ်ကြမ်းကြမ်းဖြစ်နေပေမယ့် အခြေခံ ဂရပ်ဖစ် နမူနာလေး အဖြစ်တော့ အသုံးကျပါတယ်။



ဒီနမူနာလေးမှာ စက်ဝိုင်းတစ်ခုရဲ့ သွင်ပြင်လက္ခဏာတွေကို structure ထဲမှာ စုစည်းထားတာကို သတိထားမိမှာပါ။ ၎င်းကို circ_draw() function ထဲကို input အနေနဲ့ ထည့်သွင်းပေးလိုက်ပြီး ရေးဆွဲလိုက်တာပဲ ဖြစ်ပါတယ်။ circ_draw() ကို ဘယ်လို ရေးသားထားသလဲ

ဆိုတာကိုတော့ function အကြောင်း ပြီးတဲ့ အချိန်မှာ Console Graphics Lite functions များအကြောင်း အသေးစိတ် ရှင်းပြတဲ့ အထဲမှာ ထည့်သွင်းရှင်းပြပေးသွားမှာပါ။

Names in the Declaration

Function declarations တွေကို ကြေငြာတဲ့ နေရာမှာ ပိုမိုရှင်းလင်းအောင် ရေးသားနိုင်ပါသေးတယ်။ Input arguments အနေနဲ့ ထည့်ပေးရမယ့် data type တွေနဲ့ အတူ အဓိပ္ပါယ်ရှိတဲ့ variable name တွေ ထည့်ပေးလိုက်ဖို့ပဲ ဖြစ်ပါတယ်။ စာရေးသူအနေနဲ့ကတော့ function declarations တွေပြုလုပ်ဖို့ ဒီနည်းကို အမြဲအသုံးပြုပါတယ်။ ဥပမာအနေနဲ့ ကွန်ပျူတာ ဖန်သားပြင်ပေါ်မှာ အမှတ်တစ်ခုကို ပြသပေးတဲ့ function လေးတစ်ခုကို အသုံးပြုနေတယ် ဆိုကြပါစို့။ ယခင် ဥပမာများအရ အောက်ပါအတိုင်း data typed တွေကို ထည့်သွင်းပြီး ရေးသားနိုင်ပါတယ်။

```
void display_point(int, int); //declaration
```

အထက်ပါ function declaration လေးကို အောက်ပါအတိုင်း ပိုမိုရှင်းလင်းအောင် ပြင်ရေးနိုင်ပါတယ်။

```
void display_point(int horiz, int vert); //declaration
```

Compiler အတွက်တော့ အထက်ပါ declarations နှစ်ခုစလုံးကဟာ အတူတူပါပဲ။ ဒါပေမယ့် ပထမ declaration မှာ ပါတဲ့ (int, int) ဟာ ထည့်ပေးရမယ့် arguments တွေနဲ့ ပတ်သက်ပြီး ဘာမှ မပြောထားတဲ့ အတွက် ဘယ်တစ်ခုက vertical coordinate အတွက် ဖြစ်ပြီး၊ ဘယ်တစ်ခုက horizontal coordinate အတွက် ဖြစ်တယ်ဆိုတာကို မသိနိုင်ပါဘူး။

ဒုတိယနည်းလမ်းကို အသုံးပြုပြီး ကြေငြာလိုက်တဲ့ အခါမှာ ပရိုဂရမ်မာတွေ အတွက် ပိုမိုရှင်းလင်း လွယ်ကူစေပါတယ်။ ဒီလို declaration မျိုးကို ဖတ်မိတဲ့ မည်သူမဆို အဲဒီ function ကို ခေါ်ယူတဲ့ နေရာမှာ arguments တွေကို မှန်မှန်ကန်ကန် ထည့်သွင်းမိဖို့များပါတယ်။ မှတ်သားရမယ့် အချက် တစ်ခုကတော့ အဲဒီ declarations တွေထဲမှာ အသုံးပြုထားတဲ့ variable names တွေဟာ function တစ်ခုကို ခေါ်ယူတဲ့ အခါမှာ အသုံးပြုမယ့် variable names တွေအပေါ်မျှ အကျိုးသက်ရောက်မှု မရှိတာပဲ ဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီလို ကြေငြာဖို့ ကြိုက်နှစ်သက်ရာ variable names တွေကို လွတ်လွတ်လပ်လပ် အသုံးပြုနိုင်ပါတယ်။ ဥပမာ-

```
display_point(x, y); // function call
```

ဒါကြောင့် ကျွန်တော်ရေးသားမယ့် နောက်ပိုင်း သင်ခန်းစာများမှာ စာဖတ်သူများ လွယ်ကူစွာ နားလည်နိုင်စေဖို့ ရည်ရွယ်ပြီး ဒုတိယနည်းလမ်းဖြစ်တဲ့ variable name နဲ့ data type တွဲသုံးပြီး ရေးသားပြသွားမှာ ဖြစ်ပါတယ်။

Returning Values from Functions

Function တစ်ခုကို ဆောင်ရွက်ပြီးသွားတဲ့ အခါမှာ တန်ဖိုးတစ်ခုကို ခေါ်ယူတဲ့ ပရိုဂရမ်ကို ပြန်ပို့ပေးပါတယ်။ များသောအားဖြင့် အဲဒီ တန်ဖိုးဟာ function က သက်ဆိုင်ရာ ပြဿနာ တစ်ခုကို ဖြေရှင်းပေးလိုက်တဲ့ အဖြေတွေ ပါဝင်လေ့ ရှိပါတယ်။ အောက်ပါ ဥပမာမှာ အလေးချိန်တစ်ခုကို ပေါင်ကနေ ကီလိုဂရမ် ပြောင်းပေး မှာ ဖြစ်ပါတယ်။ convert.cpp လေးကို လေ့လာကြည့်ကြရအောင်-


```
// 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;
}
```

အောက်မှာ အဲဒီ ပရိုဂရမ်ကို နမူနာ စမ်းသပ်ပြထားပါတယ်။ output တွေကို လေ့လာကြည့်ပါ-

Enter your weight in pounds: 182

Your weight in kilograms is 82.553741

Function တစ်ခုမှ တန်ဖိုးတစ်ခုကို return ပြန်ပေးနိုင်ရန်အတွက် အဆိုပါ တန်ဖိုး၏ data type ကို သတ်မှတ်ပေးထားရမည် ဖြစ်သည်။ ထိုသို့သတ်မှတ်ခြင်းကို function declaration အတွင်း ထည့်သွင်းရေးသားရမည် ဖြစ်သည်။ အထက်ပါ ဥပမာပြထားတဲ့ အတိုင်း function declaration နဲ့ definition နှစ်ခုစလုံးမှာ (float) data type ကို function name ရဲ့အရှေ့မှာ ရေးသားသတ်မှတ်ရမှာ ဖြစ်ပါတယ်။

ယခင်က ဥပမာပေး ရေးသားပြထားတဲ့ နမူနာ ပရိုဂရမ်များမှာ တန်ဖိုးပြန်ပေးတာ မရှိခဲ့ပါဘူး။ ဒါကြောင့် return type ကို void လို့ ရေးသားပေးရပါတယ်။ convert.cpp မှာ ရေးသားထားတဲ့

lbstokg() မှာတော့ float type ကို return ပြန်ပေးထားပါတယ်။ ဒါကြောင့် သူရဲ့ declaration က အောက်ပါအတိုင်း ဖြစ်ပါတယ်-

float lbstokg(float);

ရှေ့ဆုံးမှာ ရှိတဲ့ float က return type ကို သတ်မှတ်ပေးတာ ဖြစ်ပါတယ်။ နောက်က လဲသဲကွင်းထဲမှာ ရှိတဲ့ float ကတော့ အဲဒီ function (lbstokg()) ကို ထည့်သွင်းပေးရမယ့် data type ကို သတ်မှတ်ပေးထားတာ ဖြစ်ပါတယ်။

Function တစ်ခုက တန်ဖိုးတစ်ခုကို return ပြန်ပေးတဲ့ အခါမှာ lbstokg(lbs) ဆိုတဲ့ function call ကို အဲဒီ function က ပြန်ပေးတဲ့ တန်ဖိုးအနေနဲ့ ယူဆပြီး အသုံးပြုမှာ ဖြစ်ပါတယ်။ ဥပမာအားဖြင့် အဲဒီက ထွက်လာတဲ့ တန်ဖိုးကို အခြား variable တွေကဲ့သို့ ယူဆပြီး assignment statement မှာ

kgs = lbstokg(lbs);

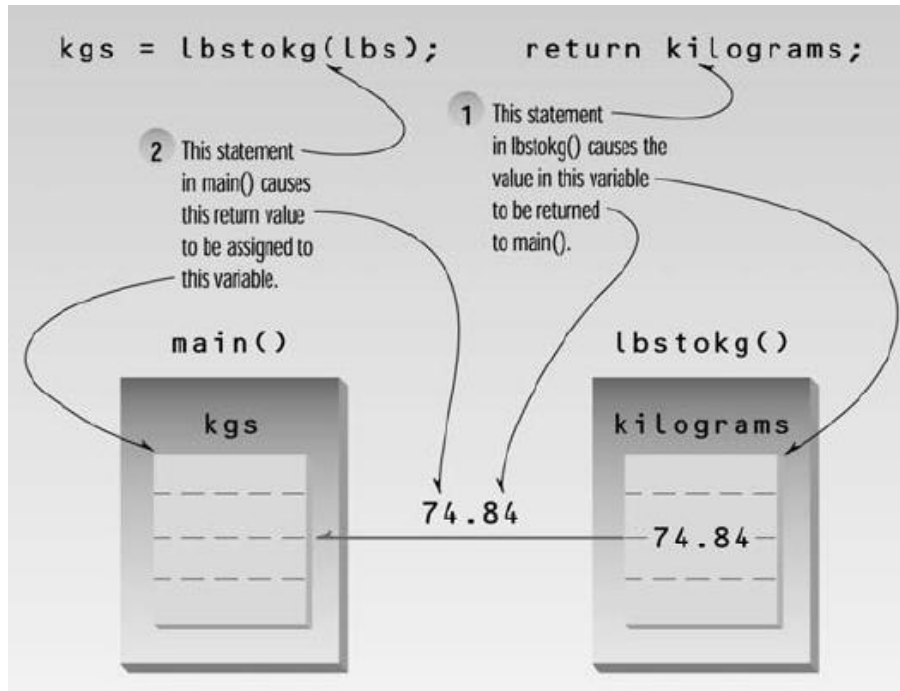
ဒီကုန်က lbstokg() ကနေ return ပြန်ပေးလိုက်တဲ့ တန်ဖိုးကို kgs ဆိုတဲ့ variable ထဲ assign လုပ်ပြီး ထည့်ပေးသွားမှာ ဖြစ်ပါတယ်။

The return Statement

Function lbstokg() ထဲကို အလေးချိန် ပေါင်တန်ဖိုးတွေကို ကိုယ်စားပြုတဲ့ argument တစ်ခုထည့်သွင်းပေးတဲ့ အခါမှာ pounds ဆိုတဲ့ parameter ထဲမှာ သိမ်းဆည်းထားလိုက်မှာ ဖြစ်ပါတယ်။ အဲဒီ ပေါင်တန်ဖိုးတွေကို ကီလိုဂရမ်ပြောင်းလဲဖို့ ကိန်းသေတစ်ခုနဲ့ မြှောက်ပြီး kilograms ဆိုတဲ့ variable ထဲမှာ သိမ်းထားလိုက်ပါတယ်။ အဲဒီ variable ထဲက တန်ဖိုးကို အောက်ပါအတိုင်း return statement အသုံးပြုပြီး ခေါ်ယူတဲ့ ပရိုဂရမ်ကို ပြန်ပေးမှာ ဖြစ်ပါတယ်။

return kilograms;

သတိထားရမယ့် အချက်တစ်ခုကတော့ ကီလိုဂရမ် တန်ဖိုးကို သိမ်းဆည်းဖို့ main() function ထဲမှာ kgs ဆိုတဲ့ variable ကို အသုံးပြုထားပြီး lbstokg() ထဲမှာ kilograms ဆိုတဲ့ variable ကို အသုံးပြုထားပါတယ်။ Calling program ဖြစ်တဲ့ main() function က lbstokg() function ထဲက kilograms ဆိုတဲ့ variable ကို ရယူသုံးစွဲခြင်း မပြုဘဲ (သုံးလိုလဲ မရပါဘူး) အဲဒီ lbstokg() function ကနေ ပြန်ပေးတဲ့ တန်ဖိုးကိုပဲ ရယူသုံးစွဲထားတာ ဖြစ်ပါတယ်။ အဲဒီ ဖြစ်စဉ်ကို အောက်ပါ ပုံမှာ ရှင်းလင်း တင်ပြထားပါတယ်။



ကျွန်တော်တို့ အနေနဲ့ function တစ်ခုကို arguments တွေအများကြီး ထည့်သွင်းပေးလို့ ရပေမယ့် return ပြန်ပေးတဲ့ အခါမှာတော့ argument တစ်ခုပဲ ပြန်ပေးနိုင်ပါတယ်။ တကယ်လို့ function တစ်ခုကနေ အချက်အလက်အများအပြား ပေးချင်လာတဲ့ အခါမှာ အဲဒီ ကန့်သတ်ချက်ကြောင့် အခက်အခဲ ဖြစ်လာနိုင်ပါတယ်။ ဒါကြောင့် အဲဒီအခက်အခဲကို ကျော်လွှားဖို့အတွက် arguments တွေကို pass by reference နည်းလမ်းနဲ့သော် လည်းကောင်း၊ အချက်အလက်အများအပြားကို စုစည်းပြီး structure တစ်ခုတည်ဆောက်ကာ return ပြန်ပေးခြင်းဖြင့် သော်လည်းကောင်း ဖြေရှင်းနိုင်ပါတယ်။

ကျွန်တော်တို့ အနေဖြင့် function တစ်ခု၏ return type ကို function declaration မှာ မဖြစ်မနေ ထည့်သွင်းပေးရမှာ ဖြစ်ပါတယ်။ အကယ်၍ function အနေဖြင့် return ပြန်ပေးရန် မလိုအပ်ပါက void ကို ထည့်သွင်းပေးခြင်းဖြင့် ညွှန်ပြနိုင်ပါသည်။ function declaration တွင် return type ထည့်သွင်းမပေးပါက compiler အနေဖြင့် int value ပြန်ပေးလိမ့်မည်ဟု ယူဆမည် ဖြစ်သည်။ ဥပမာအနေဖြင့် အောက်ပါ declaration ကို လေ့လာကြည့်ပါ-

somefunc(); // declaration -- assumes return type is int tells the compiler that

somefunc() တွင် return type ကို သတ်မှတ်ပေးထားခြင်းမရှိ၍ int ဟု ယူဆမည် ဖြစ်သည်။ ဤသို့ ယူဆရခြင်းမှာ အစောပိုင်း C versions များ အသုံးပြုပုံကို အခြေခံထားခြင်းကြောင့် ပင်ဖြစ်သည်။ သို့ရာတွင် ၎င်းနည်းလမ်းကို လက်တွေ့တွင် အသုံးမပြုသင့်ပေ။ return type သည် int ဖြစ်နေရင်တောင်မှ declaration တွင် မဖြစ်မနေ သတ်မှတ်ပေးသင့်ပေသည်။ သို့မှသာ ကုဒ်များ တစ်ပုံတည်း ဖြစ်ပြီး ဖတ်ရလွယ်မည် ဖြစ်သည်။

Eliminating Unnecessary Variables

convert.cpp ပရိုဂရမ်တွင် တကယ်တမ်း မလိုအပ်ဘဲ ဖတ်ရှုရ ရှင်းလင်းစေရန် ထည့်သွင်း အသုံးပြုထားသော variables အများအပြား ပါဝင်သည်။ အောက်ပါ convert2.cpp ပရိုဂရမ်တွင် မလိုအပ်သော variables များကို ဖြုတ်၍ ရေးသားပြထားသည်။

```
// 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 တွင် convert.cpp တွင် အသုံးပြုထားသော variable kgs ကို ဖြုတ်ပစ်ပြီး function call - lbstokg(lbs) ကို cout statement အတွင်းတွင် အောက်ပါအတိုင်း တိုက်ရိုက် အသုံးပြုထားသည်။

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

ထိုနည်းတူ lbstokg() function တွင်လည်း variable kilograms ကို အသုံးမပြုတော့ဘဲ 0.453592*pounds ဟူသည့် expression ကို return statement တွင် တိုက်ရိုက်ထည့်သွင်းထားသည်။

```
return 0.453592 * pounds;
```

ယခင် နည်းလမ်းအရ တွက်ချက်မှုမှ ရရှိသော တန်ဖိုးကို variable အတွင်း ထည့်သွင်းကာ calling program ကို return ပြန်ပေးမည်ဖြစ်သည်။ ယခု နည်း၌မူ တွက်ချက်ရသော တန်ဖိုးကို calling program သို့ တိုက်ရိုက် return ပြန်ပေးမည်ဖြစ်သည်။ ပိုမိုရှင်းလင်းစေရန်အတွက် ပရိုဂရမ်မာ အများစုက expression ကို လက်သဲကွင်းဖြင့် ပိတ်၍ အောက်ပါအတိုင်း ရေးသားလေ့ ရှိသည်။

```
return (0.453592 * pounds);
```

Compiler အနေဖြင့် အဆိုပါ လက်သဲကွင်းများကို မလိုအပ်သော်လည်း ကျွန်တော်တို့ ပရိုဂရမ်မာများ ဖတ်ရှုရ လွယ်ကူစေရန် ထည့်သွင်းပေးသင့်သည်။ ကျွမ်းကျင်သော C++ (သို့မဟုတ် C) ပရိုဂရမ်မာများ အနေဖြင့် ပိုမိုရှည်လျားသော convert.cpp ပုံစံမျိုးထက် တိုတောင်းသော convert2.cpp ပုံစံမျိုးကို ကြိုက်နှစ်သက်လေ့ ရှိကြသည်။ သို့ရာတွင် convert2.cpp ပုံစံသည် နားလည်ရန် ခက်ခဲနိုင်သောကြောင့် မကျွမ်းကျင်သေးသူများအဖို့ အသုံးမပြုသင့်ပေ။ ကျွန်တော်တို့ အနေဖြင့် တိုတောင်းမှု နှင့် ရှင်းလင်းမှုကို ရွေးချယ်ရသည့်အခါတွင် ကျွန်တော်တို့၏ ရေးသားမှုစတိုင်နှင့် အဆိုပါ ကုဒ်များကို မည်သူက ဖတ်ရှုမည်ဟူသော အချက်ကို အခြေခံ၍ ရွေးချယ်ကြရမည် ဖြစ်ပါသည်။

Returning Structure Variables

ရှေ့ပိုင်း သင်ခန်းစာများတွင် functions များအတွင်း arguments များအနေဖြင့် structures များကို ထည့်သွင်းပေးနိုင်ကြောင်း လေ့လာခဲ့ကြပါသည်။ ထိုနည်းတူစွာ structure တစ်ခုကို return value အဖြစ်လည်း အသုံးပြုနိုင်ပါသည်။ retstrc.cpp တွင် Distance variables များကို ပေါင်းကာ ရရှိလာသော Distance data type တန်ဖိုးကို return ပြန်ပေးမည် ဖြစ်ပါသည်။

```
// 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 << "'";
}
```

အသုံးပြုသူများအနေဖြင့် ပေ နှင့် လက်မ များကို အလျား နှစ်စုံအတွက် ထည့်သွင်းပေးရမည် ဖြစ်ပြီး addengl() function ကို အသုံးပြု၍ တန်ဖိုးနှစ်ခုကို ပေါင်းပေးမည် ဖြစ်သည်။ ထို့နောက် ရလာသော တန်ဖိုးများကို engldisp() function ကို အသုံးပြု၍ ပြသပေးမည် ဖြစ်သည်။ အောက်မှာ အဲဒီ ပရိုဂရမ်ကို နမူနာ စမ်းသပ်ပြထားပါတယ်။ output တွေကို လေ့လာကြည့်ပါ-

Enter feet: 4

Enter inches: 5.5

Enter feet: 5

Enter inches: 6.5

4'-5.5" + 5'-6.5" = 10'-0"

Distance type အလျားနှစ်ခုကို ပေါင်းဖို့ addengl() function ကို ခေါ်ယူပြီး ရလာတဲ့ တန်ဖိုး ကို နောက်ထပ် Distance type ဖြစ်တဲ့ d3 ထဲကို အောက်ပါအတိုင်း ထည့်သွင်းပေးလိုက်ပါတယ်။

```
d3 = addengl(d1, d2);
```

ဒီနေ့ သင်ခန်းစာလေးမှာ structure တစ်ခုကို function ရဲ့ return value အဖြစ် ပြန်ပေးနိုင်တဲ့အကြောင်း လေ့လာရသလို အထက်ပါ ပရိုဂရမ်လေးမှာ function နှစ်ခုကို သုံးပြုထားပါတယ်။ အဲဒီ function တွေရဲ့ definitions တွေကို ရှေ့နောက် စိတ်ကြိုက် စီစဉ်ထားနိုင်ပါတယ်။ တစ်ခုတည်းသော စည်းမျဉ်းကတော့ function call တွေ မတိုင်ခင်မှာ အဲဒီ function တွေကို declare လုပ်ထားရမယ်ဆိုတဲ့ အချက်ပဲ ဖြစ်ပါတယ်ခင်ဗျာ။

Reference Arguments

Reference တစ်ခုဟာ variable တစ်ခုကို ကိုင်တွယ်ဖို့ နောက်ထပ် နည်းလမ်းတစ်ခုကို ပံ့ပိုးပေးပါတယ်။ အသုံးအဝင်ဆုံး နည်းလမ်းကတော့ function တွေကို argument တွေ pass လုပ်ပေးတဲ့ နေရာမှာပဲ ဖြစ်ပါတယ်။

ယခင် သင်ခန်းစာများမှာ function တစ်ခုကို arguments တွေကို passed by value နည်းနဲ့ ပို့ပေးခဲ့တာကို လေ့လာခဲ့ပြီးပါပြီ။ အဲဒီလိုပြုလုပ်တဲ့ နေရာမှာ ခေါ်ယူခံရတဲ့ function က အဲဒီ argument နဲ့ data type တူညီတဲ့ variable အသစ်တစ်ခုကို ဖန်တီးပြီး အဲဒီ argument ထဲက တန်ဖိုးကို copy ကူးပြီး ထည့်ပေးလိုက်ပါတယ်။

Function တွေဟာ သူ့ကို ခေါ်ယူတဲ့ ပရိုဂရမ်ထဲက မူရင်း တန်ဖိုးတွေကို ရယူနိုင်ခြင်း မရှိပါဘူး။ တန်ဖိုးတွေကို ကူးယူပြီး အသစ်ဖန်တီးရတာ ဖြစ်ပါတယ်။ အဲဒီ နည်းလမ်းဟာ တကယ်လို့ function

တွေက original variable တွေရဲ့တန်ဖိုးတွေကို ပြင်ဆင်ဖို့ မလိုအပ်တဲ့ အချိန်မျိုးမှာ အသုံးဝင်ပါတယ်။ တစ်နည်းအားဖြင့် ၎င်းနည်းလမ်းဟာ မူရင်းတန်ဖိုးတွေကို ပြင်ဆင်ခြင်း မခံရအောင် အာမခံချက် ပေးနိုင်ပါတယ်။

ဒါပေမယ့် passing arguments by reference ပြုလုပ်တဲ့ အခါမှာတော့ ကွဲပြားခြားနားစွာ ပြုမူဆောင်ရွက်ပါတယ်။ ဒီနည်းလမ်းမှာတော့ function ထဲကို တန်ဖိုးတွေ ထည့်သွင်းပေးလိုက်မယ့် အစား မူရင်းတန်ဖိုးကို ရည်ညွှန်းနေတဲ့ reference တစ်ခု (တကယ်တော့ memory address တစ်ခု) ကို pass ပြုလုပ်ပေးမှာ ဖြစ်ပါတယ်။

Passing by reference လုပ်တဲ့ နည်းလမ်းရဲ့ အရေးပါတဲ့ အားသာချက်တစ်ခုကတော့ function အနေနဲ့ ခေါ်ယူလိုက်တဲ့ ပရိုဂရမ်ထဲက တကယ် variable တွေကို ရယူပြင်ဆင်နိုင်တာပဲ ဖြစ်ပါတယ်။ အဲဒီ အားသာချက်တွေကို တကယ်တမ်း အသုံးပြုတဲ့ နေရာကတော့ function တစ်ခုကနေ အချက်အလက် အများအပြားကို ပြန်ပို့ပေးနိုင်ဖို့ အသုံးပြုတာပဲ ဖြစ်ပါတယ်။

Passing Simple Data Types by Reference

ref.cpp မှာ passed by reference နည်းလမ်းကို အသုံးပြုပြီး variable တစ်ခုကို pass လုပ်ပြထားပါတယ်။

```
// 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
}
```

ပထမဦးစွာ main() ပရိုဂရမ်မှ user ကို float တန်ဖိုး တစ်ခု ထည့်သွင်းစေပါတယ်။ ပရိုဂရမ်က ထို တန်ဖိုးကို integer နဲ့ fractional part ဆိုပြီး ကိန်းပြည့်နဲ့ ဒဿမ ဂဏန်းတွေ အဖြစ် ခွဲထုတ်လိုက်ပါတယ်။ မြင်သာအောင် ဥပမာပေးရမယ်ဆိုရင် ၁၂.၄၅၆ ကို ၁၂.၀ နဲ့ ၀.၄၅၆ ဆိုပြီး ခွဲထုတ်လိုက်တာ ဖြစ်ပါတယ်။ အဲဒီလို ခွဲထုတ်နိုင်ဖို့အတွက် main() က intfrac() function ကို ခေါ်ယူရပါတယ်။ အောက်မှာ တော့ အဲဒီ ပရိုဂရမ်ကို စမ်းသပ်လို့ ထွက်လာတဲ့ အဖြေတွေကို ပြသထားပါတယ်။

Enter a real number: 99.44

Integer part is 99, fractional part is 0.44

အချို့ compiler တွေဟာ ဒဿမကိန်းကို ဖန်တီးတဲ့ အခါမှာ အပို ဒီဂျစ်တွေ ထုတ်ပေးတတ်ပါတယ်။ ဥပမာ ၀.၄၄၀၀၀၂ ဆိုတဲ့ ဂဏန်းမျိုးဖြစ်ပါတယ်။ တကယ်တော့ ဒါဟာ compiler မှာပါတဲ့ conversion routine ရဲ့ အမှားဖြစ်ပါတယ်။ intfrac() function ဟာ ထည့်သွင်းပေးလိုက်တဲ့ တန်ဖိုး (n)ကို cast လုပ်ပြီး long type ပြောင်းလဲကာ ကိန်းပြည့်တန်ဖိုးကို အောက်ပါအတိုင်း ရှာလိုက်ပါတယ်။

```
long temp = static_cast<long>(n);
```

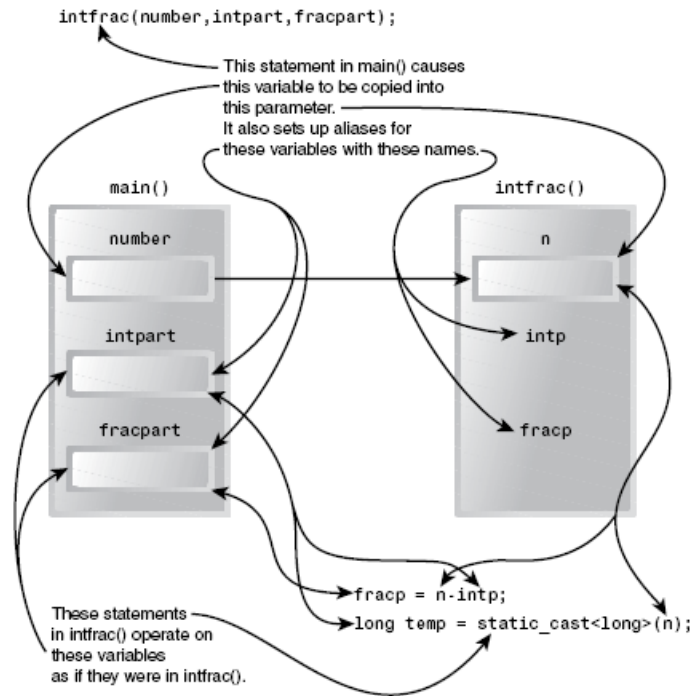
integer type ဟာ ကိန်းပြည့်ပိုင်းတွေကိုပဲ သိမ်းဆည်းနိုင်တဲ့ အတွက် ဒီနည်းကိုသုံးပြီး ဒဿမဂဏန်းတွေကို ထိထိရောက်ရောက် ဖြတ်ချထားခဲ့နိုင်ပါတယ်။ ရလာတဲ့ ရလဒ်ကို နောက်တစ်ခါ cast ပြန်လုပ်ပြီး အောက်ပါအတိုင်း float ပြန်ပြောင်းလိုက်ပါတယ်-

```
intp = static_cast<float>(temp);
```

ဒဿမနောက်က တန်ဖိုးကို ရှာဖို့အတွက်ကတော့ မူရင်းတန်ဖိုးကို ကိန်းပြည့်တန်ဖိုး နုတ်ပေးပြီး အလွယ်တကူ ရှာနိုင်ပါတယ်။

intfrac() function အနေနဲ့ ကိန်းပြည့်နဲ့ ဒဿမနောက်က ဂဏန်း နှစ်ခုစလုံးကို ရှာလို့ရပါပြီ။ ဒါပေမယ့် function တစ်ခုဟာ တန်ဖိုး တစ်ခုကိုပဲ return ပြန်ပေးလို့ ရတဲ့အတွက် နှစ်ခုစလုံးကို ပြန်ပေးဖို့ ဘယ်လိုလုပ်ကြမလဲ? ဒီပြဿနာကို reference arguments တွေ အသုံးပြုပြီး ဖြေရှင်းနိုင်ပါတယ်။ အောက်မှာ ဖော်ပြထားတဲ့ intfrac() function ရဲ့ declarator လေးကို လေ့လာကြည့်ရအောင်-

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



Reference arguments တွေကို ampersand (&) အသုံးပြုရပါတယ်။ ဥပမာ -

float& intp

ဒီနေရာမှာ ampersand (&) ဟာ `intp` သည် `pass` လုပ်လိုက်တဲ့ `argument` အတွက် `alias` ဒါမှမဟုတ် အခြားအမည်တစ်ခု ဖြစ်တယ်ဆိုတာကို ဖော်ပြပေးပါတယ်။ နားလည်ရလွယ်ကူအောင် ရှင်းပြရမယ် ဆိုရင်တော့ `intfrac()` function ထဲက `intp` ကို အသုံးပြုတိုင်း တကယ်တမ်းက `main()` ထဲက `intpart` ကို ရည်ညွှန်းသုံးစွဲနေတာပဲ ဖြစ်ပါတယ်။

ampersand (&) ဟာ `reference` လို့ ဆိုလိုပါတယ်။ ဒါကြောင့် `float& intp` ဟာ `intp` သည် သူ့ကို `pass` လုပ်မယ့် `variable` ရဲ့ `reference` ဖြစ်တယ်လို့ ဆိုလိုပါတယ်။ ထိုနည်းတူပဲ `fracp` ဟာ `fracpart` ရဲ့ `alias` တစ်ခု ဖြစ်ပါတယ်။ Function ရဲ့ `declaration` ကို ရေးသားတဲ့ အခါမှာလည်း `definition` ထဲကလို & သုံးပြီး အောက်ပါအတိုင်း ရေးရပါမယ်။

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

Function definition မှာလိုပဲ `passed by reference` ပြုလုပ်မယ့် `arguments` တွေနှောက်မှာ ampersand & ထည့်ပေးရပါမယ်။ ဒါပေမယ့် function call ထဲမှာတော့ ampersand & ကို ထည့်ပေးစရာ မလိုတော့ပါဘူး။

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

ဒါကြောင့် function call ကို ကြည့်လိုက်ရုံနဲ့ ဘယ် argument က passed by reference ဒါမှမဟုတ် value ဖြစ်တယ်ဆိုတာကို မသိနိုင်ပါဘူး။ ဒီဥပမာမှာတော့ intpart နဲ့ fracpart တွေဟာ passed by reference ဖြစ်ပြီး၊ variable number ကတော့ passed by value ဖြစ်ပါတယ်။

intp နဲ့ intpart တွေဟာ fracp နဲ့ fracpart ဆိုတဲ့ အခြားနာမည်တွေရှိတဲ့ memory ထဲက နေရာတစ်ခုသာ ဖြစ်ပါတယ်။ အမည်မတူပေမယ့် နေရာကတော့ တစ်ခုထဲပါ။ ဘယ်တစ်ခုကိုပြင်ပြင် memory ထဲမှာ ပြင်ပြီးသားဖြစ်ပြီး နှစ်ခုစလုံး တန်ဖိုးတွေ အတူတူပြောင်းလဲသွားမှာ ဖြစ်ပါတယ်။

နောက်ထပ် intfrac() function ထဲက parameter တစ်ခုဖြစ်တဲ့ n ကတော့ main() function ထဲက number နဲ့ မတူညီဘဲ သီးခြား variable တစ်ခုဖြစ်ပါတယ်။ ဆိုလိုတာက အမည်လည်း မတူသလို memory ထဲက နေရာလည်း မတူညီပါဘူး။ number ထဲက တန်ဖိုးကို n ထဲကို ကော်ပီကူးထည့် ပေးလိုက်တာပါ။ number ကို တန်ဖိုးပြောင်းထည့်ဖို့ မလိုတဲ့အတွက် passed by reference မပြုလုပ်တာပဲဖြစ်ပါတယ်။ C programmer တွေအနေနဲ့ကတော့ ampersand ကို address of သင်္ကေတနဲ့ မှားတတ်တဲ့အတွက် သတိပြုစေလိုပါတယ်။

A More Complex Pass by Reference

နောက်ထပ် pass by reference ပြုလုပ်နည်းတွေကို ဆက်လက်လေ့လာပါမယ်။ ဒီဥပမာမှာတော့ အနည်းငယ် ပိုမိုရှုပ်ထွေးတဲ့ နည်းလမ်းများပါဝင်လာပါတယ်။ ကျွန်တော်တို့မှာ ဂဏန်းတွဲတွေ ရှိပြီး ငယ်တဲ့ တစ်ခုကို အမြဲရှေ့က ထားချင်တယ်ဆိုကြပါစို့။ အဲဒီလို ပြုလုပ်ဖို့အတွက် order() ဆိုတဲ့ function လေးကို ရေးပါမယ်။ အဲဒီ ထဲမှာ ဂဏန်းနှစ်ခုကို pass by reference ပြုလုပ်ပြီး ထည့်သွင်းလိုက်ပါမယ်။ ၎င်းတို့ကို စစ်ဆေးပြီး ရှေ့က ဂဏန်းက ကြီးနေခဲ့ရင် နေရာချင်း လဲပေးလိုက်မှာ ဖြစ်ပါတယ်။ reorder.cpp ကို လေ့လာကြည့်ကြရအောင်-

```
// reorder.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() ထဲမှာ ဂဏန်းနှစ်စုံ ရှိပါတယ်။ ပထမစုံတွဲက အစီအစဉ်မကျဘဲ ဒုတိယ စုံတွဲက အစီအစဉ်ကျနေပါတယ်။ ဂဏန်းနှစ်စုံ စလုံးကို order() function ခေါ်ယူပြီး ထည့်သွင်းပေးလိုက်ပါတယ်။ အဲဒီအခါမှာ အစီအစဉ်မကျတဲ့ ပထမ စုံတွဲကို နေရာလဲပြီး အစီအစဉ်ကျအောင် ပြောင်းလဲပေးလိုက်ပါတယ်။ နဂိုကထဲက အစီအစဉ်ကျနေတဲ့ ဒုတိယ စုံတွဲကိုတော့ မပြောင်းလဲပေးပါဘူး။

n1=11

n2=99

n3=22

n4=88

order() function အတွင်းမှာ ပထမ variable ကို numb1 လို့ ခေါ်ပြီး ဒုတိယ variable ကို numb2 လို့ ခေါ်ပါတယ်။ တကယ်လို့ numb1 က numb2 ထက် ကြီးနေခဲ့မယ်ဆိုရင် numb1 တန်ဖိုးကို temp ထဲ ခဏ ထည့်ထားလိုက်ပါတယ်။ နောက် numb2 ကို numb1 ထဲ ထည့်လိုက်ပြီး နောက်ဆုံးမှာတော့ temp ထဲခဏထည့်ထားတဲ့ တန်ဖိုးကို numb2 ထဲ ပြောင်းထည့်ပေးလိုက်မှာ ဖြစ်ပါတယ်။

ဒီနေရာမှာ မှတ်သားရမယ့် အချက်ကတော့ numb1 နဲ့ numb2 ဟာ ထည့်ပေးလိုက်မယ့် pass by reference arguments တွေအတွက် နောက်ထပ် နာမည်တစ်မျိုး ဖြစ်လာမှာပါ။ ဒီဥပမာမှာတော့ n1 နဲ့ n2 ကို function ပထမအကြိမ်ခေါ်စဉ် ထည့်သွင်းပေးလိုက်မှာ ဖြစ်ပြီး n3 နဲ့ n4 ကို ဒုတိယအကြိမ်

ခေါ်ယူစဉ်မှာ ထည့်သွင်းပေးရမှာ ဖြစ်ပါတယ်။ အစီအစဉ်မကျတဲ့ ဂဏန်းစုံတွဲ ဖြစ်နေရင်တော့ ဂဏန်းချင်း လဲပေးလိုက်မှာပါ။

Reference arguments တွေကို အသုံးပြုရတာဟာ အဝေးထိန်းစနစ်တစ်ခုကို အသုံးပြုရသလိုပါပဲ။ ခေါ်ယူတဲ့ ပရိုဂရမ်က function ကို သူထဲက ဘယ် variables တွေကို လုပ်ဆောင်ရမယ်ဆိုတာကို ပြောပြလိုက်ရုံနဲ့ အဲဒီ function က ခေါ်ယူတဲ့ ပရိုဂရမ်ထဲက variables တွေကို သူတို့ရဲ့ နာမည်တောင် မသိဘဲနဲ့ ပြောင်းလဲပေးလိုက်မှာပဲ ဖြစ်ပါတယ်။ မြင်သာအောင် ဥပမာပေးရမယ်ဆိုရင် အိမ်ကို ဆေးသုတ်ဖို့ အပြင်က ကုမ္ပဏီ တစ်ခုကို ဖုန်းဆက်ပြီး မှာလိုက်တဲ့အခါ အဲဒီလူတွေ တကယ်ရောက်မလာပဲနဲ့ ကျွန်တော်တို့ အိမ်လေး မှော်ဆန်ဆန် ဆေးရောင်တွေ ပြောင်းသွားသလိုပါပဲ ခင်ဗျာ။

Passing Structures by Reference

ကျွန်တော်တို့အနေနဲ့ သာမန် data type တွေကို pass by reference လုပ်လို့ ရသလို structures တွေကိုလဲ pass by reference လုပ်လို့ရပါတယ်။ အောက်ပါ referst.cpp မှာ type Distance ရဲ့ တန်ဖိုးတွေကို scale ပြောင်းလဲပေးတဲ့ အကြောင်း pass by reference လုပ်ပြီး ရေးသားထားပါတယ်။ ဒီနည်းဟာ လက်တွေ့မှာ အသုံးတည့်ပါတယ်။ အဆောက်အအုံ တစ်ခုရဲ့ အရွယ်အစားအကုန်လုံးကို scale factor ပြောင်းလဲပေးလိုက်တာနဲ့ အရွယ်အစားကို အချိုးကျ ချုံပြီး၊ ချဲ့ပြီး ရရှိမှာ ဖြစ်ပါတယ်။ referst.cpp ကို လေ့လာကြည့်ကြရအောင်-

```
// 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 မှာ Distance variables—d1 နဲ့ d2 ကို initialize ပြုလုပ်ပြီး တန်ဖိုးတွေ သတ်မှတ်ပေးပါတယ်။ ထို့နောက်မှာ scale() function ကို ခေါ်ယူပြီး d1 ကို 0.5 နဲ့ d2 ကို 0.25 နဲ့

အသီးသီး မြှောက်ပါတယ်။ ရလာတဲ့ အဖြေတွေကို နောက်ဆုံးမှာ ထုတ်ဖော်ပြသပေးပါတယ်။
အောက်မှာ နမူနာ စမ်းသပ်ပြထားပါတယ်။

d1 = 12'-6.5"

d2 = 10'-5.5"

d1 = 6'-3.25"

d2 = 2'-7.375"

function scale() ကို ခေါ်ယူတဲ့ ကုဒ်နှစ်ကြောင်းကတော့ အောက်ပါအတိုင်း ဖြစ်ပါတယ်-

scale(d1, 0.5);

scale(d2, 0.25);

ပထမကုဒ်မှာ d1 ကို 0.5 နဲ့ မြှောက်စေပြီး ဒုတိယကုဒ်မှာတော့ d2 ကို 0.25 နဲ့ မြှောက်စေမှာ ဖြစ်ပါတယ်။ မှတ်သားသင့်တဲ့ အချက်ကတော့ အဲဒီလိုပြုလုပ်ရာမှာ d1 နဲ့ d2 တန်ဖိုးတွေ တိုက်ရိုက် ပြောင်းလဲသွားတာပဲ ဖြစ်ပါတယ်။ ဒါပေမယ့် ဒီဥပမာမှာ တန်ဖိုးတစ်ခုကိုပဲ ပြောင်းလဲချင်တာ ဖြစ်တဲ့အတွက် pass by reference ကို မသုံးဘဲ pass by value ကိုသုံးပြီး return နဲ့ပြန်ပေးနိုင်ဖို့ ပြင်ရေးလို့ရပါတယ်။ အဲဒီလို function မျိုးကို ခေါ်ယူဖို့အတွက်ဆိုရင် အောက်ပါအတိုင်း ပြုလုပ်နိုင်ပါတယ်-

d1 = scale(d1, 0.5);

Notes on Passing by Reference

C programming language မှာတုန်းကတော့ References တွေ မရှိသေးပါဘူး။ အဲဒီအစား pointers တွေကို အသုံးပြုနေကြတာ ဖြစ်ပါတယ်။ C++ မှာတော့ သာမန် variables တွေသာမက objects တွေအတွက်ပါ ပိုမိုကောင်းမွန်စွာနဲ့ အဆင်ပြေပြေရေးသားနိုင်အောင် references တွေကို ထည့်သွင်းပေးထားတာ ဖြစ်ပါတယ်။ Function တွေထဲကို arguments တွေထည့်သွင်းနိုင်တဲ့ တတိယ နည်းလမ်းကိုတော့ နောက်ပိုင်း Pointer အခန်းကျမှ ဆက်လက်ရှင်းပြပေးသွားမှာပါ။

Overloaded Functions

Overloaded function များသည် ၎င်းတို့အား ပေးပို့လိုက်သော အချက်အလက်များ (input parameter(s)) များအပေါ် မူတည်၍ မတူညီသော လုပ်ဆောင်ချက်များကို ပြုလုပ်ပေးသည်။ Overloading သည် ကျွန်တော် ကြားဘူးသည့် ဟာသ တစ်ခုနှင့် တူညီပါသည်။ ကျော်ကြားသော သိပ္ပံပညာရှင်ကြီး တစ်ယောက်က ဓာတ်ဗူးကို အကြီးကျယ်ဆုံးသော တီထွင်မှုဟု အကြောက်အကန် ငြင်းခုံပါသည်။ ၎င်း၏ အဆိုမှာ ဓာတ်ဗူးသည် ပူသော အရာများကို ပူအောင် ထိန်းထားနိုင်ပြီး အေးသောအရာများကို ဆက်အေးနေအောင် ထိန်းထားနိုင်စွမ်း ရှိသောကြောင့် အလွန်ထူးဆန်း အံ့ဩဖွယ်ရာပင် ဖြစ်သည် ဟုဆိုသည်။ ဓာတ်ဗူးက ဘယ်လိုလုပ်ပြီး သိသလဲဟု ၎င်းက စောဒက တက်ခဲ့ပါသည်။ overloaded function သည်လည်း ထိုကဲ့သို့ပင် အံ့ဩဖွယ်ရာ သိရှိပြီး လုပ်ဆောင်ပေးနိုင်ပါသည်။ ၎င်းသည် ပထမ အချက်အလက် တစ်ခုအတွက်

လုပ်ဆောင်မှု တစ်ခုကို လုပ်ပေးပြီး အခြားအချက်အလက်အတွက် သက်ဆိုင်ရာ နောက်လုပ်ဆောင်မှု တစ်ခုကို ပြုလုပ်ပေးနိုင်ပါသည်။ အဆိုပါ သဘောသဘာဝကို ဥပမာများဖြင့် ရှင်းလင်းသွားမည် ဖြစ်ပါသည်။

Different Numbers of Arguments

ယခင် ဥပမာများ (table.cpp နှင့် tablearg.cpp) တွင် အသုံးပြုခဲ့သော functions များကို ပြန်လည် လေ့လာကြည့်ကြပါစို့။ starline() function သည် ကြယ် ၄၅ လုံးကို print ထုတ်ပေးသည်။ repchar() function တွင်မူ ပုံသေ သတ်မှတ်ထားခြင်း မရှိဘဲ ခေါ်ယူသော program မှ ထည့်သွင်းပေးလိုက်သည့် character နှင့် အလုံးအရေအတွက်ကို print ထုတ်ပေးမည်ဖြစ်သည်။ အကယ်၍ charline() function ကို ရေးသားကာ calling program မှ ထည့်သွင်းပေးလိုက်သော character အလုံးရေ ၄၅ လုံးကို print ထုတ်ပေးမည် ဆိုကြပါစို့။ အဆိုပါ starline(), repchar(), နှင့် charline() function သုံးမျိုးစလုံးသည် ဆင်တူသော လုပ်ဆောင်ချက်ကို လုပ်ဆောင်ကြပြီး အမည် မတူညီသည်ကို သတိထားမိပါလိမ့်မည်။ ပရိုဂရမ်များ အနေဖြင့် အဆိုပါ functions များကို အသုံးပြုရန် နာမည် သုံးခုကို မှတ်သားထားရန် လိုပြီး ရေးသားနေသော ပရိုဂရမ်အတွက် စုစည်းထားသော *Function Reference* documentation တွင် အကွစဉ်အလိုက် စီစဉ်ထားသော အထဲမှ အမည်သုံးခုစာ သုံးနေရာ လိုက်လံ ရှာဖွေနေရမည် ဖြစ်သည်။ အကယ်၍ လုပ်ဆောင်ချက် ဆင်တူသော functions များကို input arguments အရေအတွက်နှင့် data types များ မတူညီသည့်တိုင်အောင် နာမည်တူ ပေးခဲ့လျှင် အသုံးပြုမည့် ပရိုဂရမ်အတွက် ပိုမို အဆင်ပြေနိုင်မည် ဖြစ်သည်။ အောက်ပါ overload.cpp ပရိုဂရမ်တွင် လေ့လာကြည့်ကြပါစို့။

```
// 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;
}

```

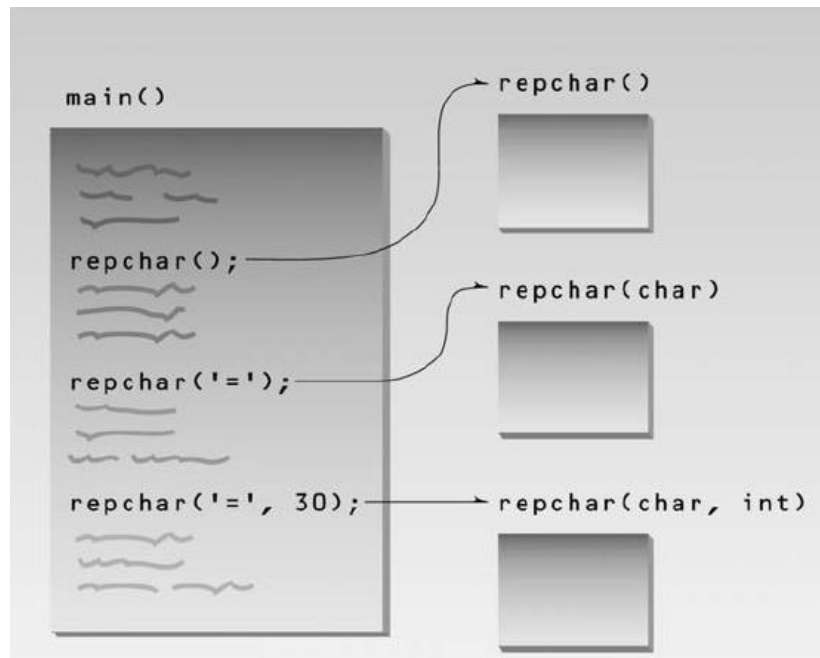
၎င်းပရိုဂရမ်ကို စမ်းသပ်ကြည့်လျှင် အောက်ပါအတိုင်း အကွေ့ရာမျဉ်း သုံးကြောင်းကို တွေ့ရမည် ဖြစ်ပါသည်။

=====

+++++

ပထမ နှစ်ကြောင်းသည် အကွာရာ ၄၅ လုံးရှည်လျားပြီး နောက်ဆုံး တတိယ အကြောင်းတွင် အလုံးရေ ၃၀ ပါဝင်ပါသည်။ အဆိုပါ ပရိုဂရမ်တွင် အမည်တူသော functions သုံးခု ပါဝင်ပါသည်။ function declarations သုံးခု၊ function calls သုံးခုနှင့် function definitions သုံးခုတို့ ပါဝင်ပါသည်။ ၎င်းတို့ကို compiler က ခွဲခြားသိနိုင်ခြင်းမှာ ၎င်းတို့တွင် ပါဝင်သော မတူညီသည့် arguments အရေအတွက် နှင့် မတူညီသော data types များပင် ဖြစ်ပါသည်။

နောက်တစ်နည်းဆိုရလျှင် void repchar(); သည် argument လုံးဝ မရှိဘဲ၊ char type argument တစ်ခုပါဝင်သော void repchar(char); နှင့် char type argument တစ်ခုနှင့် int type argument တစ်ခု ပါဝင်သော void repchar(char, int); functions များနှင့် မူညီဘဲ ကွဲပြားခြားနားပေသည်။ ၎င်းအချက်ကို compiler က ခွဲခြားနိုင်စွမ်းရှိသဖြင့် အမည်တူသော်လည်း သက်ဆိုင်ရာ function ကိုသာ ခေါ်ယူ လုပ်ဆောင်ပေးမည် ဖြစ်သည်။



(ပုံ-၁) function overloading ပြုလုပ်ပုံ

Different Kinds of Arguments

overload.cpp ပရိုဂရမ်တွင် input arguments အရေအတွက် မတူသော်လည်း အမည်တူသည့် functions သုံးခုကို ရေးသားထားပါသည်။ Compiler အနေဖြင့် arguments အရေအတွက် မတူညီမှုဖြင့် ၎င်းတို့အား ခွဲခြားပြီး လုပ်ဆောင်ပေးနိုင်ခဲ့ပါသည်။ သို့ရာတွင် Compiler အနေဖြင့် argument အရေအတွက် တူညီသော်လည်း data type မတူညီခြင်းကို အခြေခံပြီး ခွဲခြားပေးနိုင်စွမ်းလည်း ရှိပါသည်။ overengl.cpp တွင် လေ့လာကြည့်နိုင်ပါသည်။ ၎င်းတွင် function အတွင်း ထည့်သွင်း ပေးရမည့် တစ်ခုတည်းသော argument ကို structure type Distance ဖြစ်စေ သို့မဟုတ် simple variable float type ဖြစ်စေ အသုံးပြုနိုင်ပါသည်။

Compiler မှ ထည့်သွင်းပေးလိုက်သည့် argument ၏ type ပေါ် မူတည်ပြီး သက်ဆိုင်ရာ function ကို ခေါ်ယူပေးမည် ဖြစ်ပါသည်။

```
// 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 << "\"";
}

```

အထက်ပါ ပရိုဂရမ်တွင် user အား distances တန်ဖိုး နှစ်ခုကို ထည့်သွင်းစေပါသည်။ ပထမ အကြိမ်တွင် ပေ နှင့် လက်မ ကို ခွဲခြား ထည့်သွင်းစေပြီး ဒုတိယ အကြိမ်တွင် လက်မဖွဲ့ပြီးသား အတိုင်းအတာကိုသာ ထည့်သွင်းစေပါသည်(ဥပမာ - 9'-1.5" အစား 109.5 inches ဟု ထည့်သွင်းပေးရန် ဖြစ်သည်)။ ပရိုဂရမ်မှ ပထမအကြိမ်တွင် Distance type argument အသုံးပြုထားသည့် engldisp(Distance) function ကို ခေါ်ယူပြီး ဒုတိယအကြိမ်တွင် float type argument ကို အသုံးပြုထားသော engldisp(float) function ကို ခေါ်ယူမည် ဖြစ်ပါသည်။ အောက်ပါအတိုင်း နမူနာ စမ်းသပ် အသုံးပြုပြထားပါသည်။

Enter feet: 5

Enter inches: 10.5

Enter entire distance in inches: 76.5

d1 = 5'-10.5"

d2 = 6'-4.5"

သတိပြုရမည့် အချက်တစ်ခုမှာ engldisp() functions နှစ်ခုစလုံးသည် ဆင်တူသော လုပ်ဆောင်မှု ရှိသော်လည်း ကုဒ်များမှာ လုံးဝ ကွဲပြားခြားနားပေသည်။ လက်မဖွဲ့ပြီးသား တန်ဖိုးကို အသုံးပြုသည့် function သည် ပေ နှင့် လက်မ ပြန်ဖွဲ့ပြီးမှ ရလဒ်ကို ပြသပေးမည် ဖြစ်ပါသည်။

Overloaded functions များကို အသုံးပြုခြင်းဖြင့် ပရိုဂရမ်များသည် function အမည်များစွာကို မှတ်သားရန် မလိုတော့သဖြင့် ရေးသားရသည်မှာ ပိုမိုလွယ်ကူလာပေသည်။ အကယ်၍ overloaded function အသုံးမပြုခဲ့လျှင် function နာမည်များစွာကို မှတ်သားရသဖြင့် မှားယွင်းနိုင်ပေသည်။ ဥပမာအားဖြင့် absolute value ကို ရှာရန် ရေးသားထားသည့် functions များသည် C programming အတွက် အမျိုးမျိုး ကွဲပြားနေပေသည်။ int data type အတွက် abs()၊ complex numbers အတွက် cabs()၊ type double အတွက် fabs() နှင့် type long အတွက် labs() ဟူ၍ အသီးသီး ခွဲခြားရေးသားထားရပေသည်။ C++ တွင်မူ abs() တစ်ခုတည်းဖြင့် ကြိုက်နှစ်သက်ရာ data type ကို ထည့်သွင်းပြီး အသုံးပြုနိုင်ပါသည်။ Function overloading ကို objects များအကြောင်း လေ့လာရာတွင် ဆက်လက် ရှင်းပြပါဦးမည်။

နိဂုံး

တကယ်တော့ function တွေအကြောင်းကို ရှင်းပြဖို့ အနည်းငယ် ကျန်နေပါသေးတယ်။ ဒါပေမယ့် အခြေအနေအရ OOP in C++ အပိုင်း (၁) ကို ဒီနေရာမှာပဲ အဆုံးသတ်လိုက်ပါတယ်။ သိလိုသည်များကို aungwh2013@gmail.com ကို ဆက်သွယ်ပြီး မေးမြန်းနိုင်သလို YCC ကျောင်းသားများ အတွက်လည်း အောက်ဖော်ပြပါ လိပ်စာရှိ **တော်ဝင် ကွန်ပျူတာ စင်တာ**၌လည်း လေ့လာ စုံစမ်းနိုင်ပါတယ်ခင်ဗျာ။

မှီငြမ်း

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

Dr. အောင်ဝင်းထွဋ် (bluephoenix)

<http://engineer4myanmar.blogspot.com>

တော်ဝင် ကွန်ပျူတာ စင်တာ

၁၇၉ စ၊ သုမင်္ဂလာ၊ ဈေးလေး အနောက်ဘက်၊ ပြင်ဦးလွင်မြို့။