**OOP in C++ : အပိုင္း (၉) – for loop**

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

**The for Loop**

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

// fordemo.cpp

// demonstrates simple FOR loop

#include <iostream>

using namespace std;

int main()

{

int j; //define a loop variable

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

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

cout << endl;

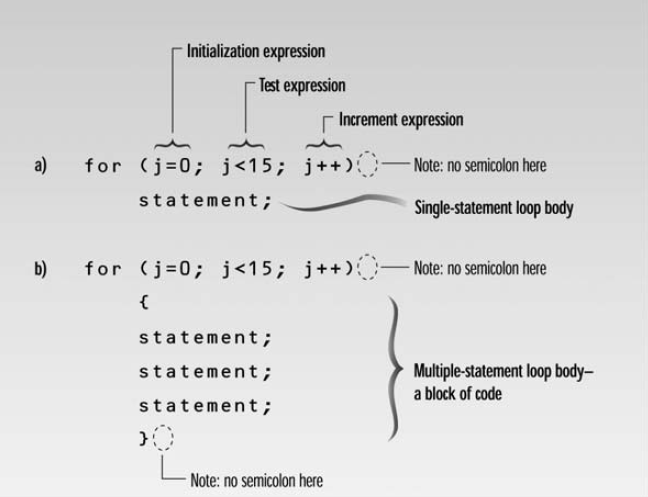
return 0;

}

ပရိုဂရမ္ကို RUN ၾကည္႔လွ်င္ ေအာက္ပါ အေျဖမ်ားကို ရရွိမည္ ျဖစ္သည္။

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

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



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

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

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

**The Initialization Expression**

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

**The Test Expression**

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

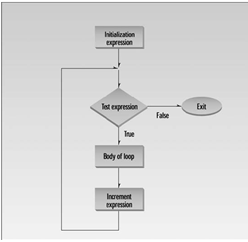
**The Increment Expression**

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

**How Many Times?**

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

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



ေနာက္ထပ္ ဥပမာေလး တစ္ခု ေရးျပပါ႔မယ္။

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

//loop body

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

**Multiple Statements in the Loop Body**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ loop body ထဲမွာ statement တစ္ေၾကာင္းမက ေရးဖို႔ လိုအပ္ေလ႔ ရွိပါတယ္။

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

// cubelist.cpp

// lists cubes from 1 to 10

#include <iostream>

#include <iomanip> //for setw

using namespace std;

int main()

{

int numb; //define loop variable

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

{

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

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

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

}

return 0;

}

အေျဖေတြကေတာ႔ ေအာက္ပါအတိုင္း ျဖစ္ပါတယ္-

1 1

2 8

3 27

4 64

5 125

6 216

7 343

8 512

9 729

10 1000

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

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

**Blocks and Variable Visibility**

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

int cube = numb\*numb\*numb;

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

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

**Indentation and Loop Style**

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

There is a common variation on the style we use for loops in this book. We show the braces aligned vertically, but some programmers prefer to place the opening brace just after the loop statement, like this:

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

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

cout << setw(4) << numb;

int cube = numb\*numb\*numb;

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

}

This saves a line in the listing but makes it more difficult to read, since the opening brace isharder to see and harder to match with the corresponding closing brace. Another style is to indent the body but not the braces:

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

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

{

cout << setw(4) << numb;

int cube = numb\*numb\*numb;

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

}

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

**Debugging Animation**

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

**for Loop Variations**

The increment expression doesn’t need to increment the loop variable; it can perform any operation it likes. In the next example it *decrements* the loop variable. This program, FACTOR, asks the user to type in a number, and then calculates the factorial of this number. (The factorial is calculated by multiplying the original number by all the positive integers smaller than itself. Thus the factorial of 5 is 5\*4\*3\*2\*1, or 120.)

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

// factor.cpp

// calculates factorials, demonstrates FOR loop

#include <iostream>

using namespace std;

int main()

{

unsigned int numb;

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

cout << “Enter a number: “;

cin >> numb; //get number

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

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

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

return 0;

}

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

Enter a number: 10

Factorial is 3628800

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

**Variables Defined in for Statements**

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

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

အသံုးမ်ားသလို ေကာင္းမြန္တဲ႔ loop variable define ျပဳလုပ္နည္းပဲ ျဖစ္ပါတယ္။

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

**Multiple Initialization and Test Expressions**

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

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

{

// body of loop

}

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

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

ေနာက္အပတ္ေတြမွာေတာ႔ က်န္ရွိေနေသးတဲ႔ looping ေတြအေၾကာင္းကို ဆက္လက္ ေဆြးေႏြးသြားပါမယ္ခင္ဗ်ာ။ ဤအခန္းဆက္ ေဆာင္းပါးမ်ားဟာ IT ေက်ာင္းသားမ်ားအတြက္ အထူးရည္ရြယ္ၿပီး ေရးသားထားတာ ျဖစ္ပါတယ္။ သိလိုသည္မ်ားကို aungwh2013@gmail.com ကို ဆက္သြယ္ၿပီး ေမးျမန္းႏိုင္သလို YCC ေက်ာင္းသားမ်ား အတြက္လည္း ေအာက္ေဖာ္ျပပါ လိပ္စာရွိ ေတာ္ဝင္ ကြန္ပ်ဴတာ စင္တာ၌လည္း ေလ႔လာ စံုစမ္းႏိုင္ပါတယ္ခင္ဗ်ာ။

**အကိုးအကား**

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

Dr. ေအာင္ဝင္းထြဋ္ (bluephoenix)

http://www.tech4mm.com

ေတာ္ဝင္ ကြန္ပ်ဴတာ စင္တာ

၁၇၉ စ၊ သုမဂၤလာ၊ ေစ်းေလး အေနာက္ဘက္၊ ျပင္ဦးလြင္ၿမိဳ႕