**OOP in C++ : အပိုင္း (၁၁) - do loop**

ဒီအပတ္ သင္ခန္းစာမွာေတာ႔ က်န္ရွိေနေသးတဲ႔ do loop နဲ႔ အတူ decision လုပ္တဲ႔ အေၾကာင္းေလးေတြကို ဆက္လက္ ေဆြးေႏြးသြားမွာ ျဖစ္ပါတယ္။

**The do Loop**

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

// divdo.cpp

// demonstrates DO loop

#include <iostream>

using namespace std;

int main()

{

long dividend, divisor;

char ch;

do //start of do loop

{ //do some processing

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

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

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

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

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

cin >> ch;

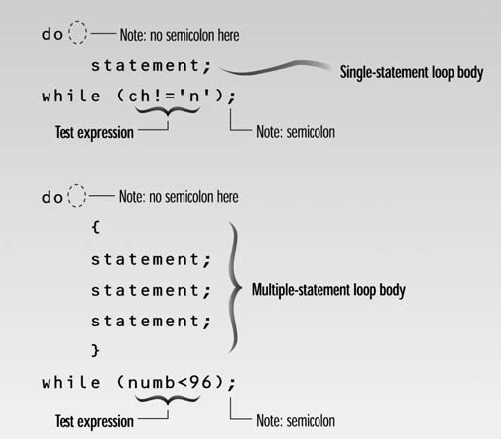
}

while( ch != ‘n’ ); //loop condition

return 0;

}

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



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

Enter dividend: 11

Enter divisor: 3

Quotient is 3, remainder is 2

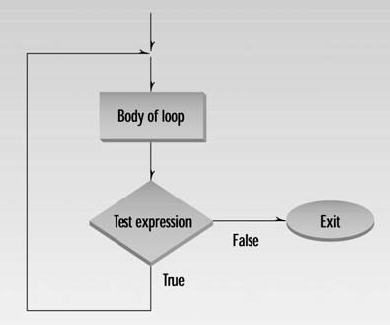
Do another? (y/n): y

Enter dividend: 222

Enter divisor: 17

Quotient is 13, remainder is 1

Do another? (y/n): n



**When to Use Which Loop**

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

**Decisions**

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

**The if Statement**

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

// ifdemo.cpp

// demonstrates IF statement

#include <iostream>

using namespace std;

int main()

{

int x;

cout << “Enter a number: “;

cin >> x;

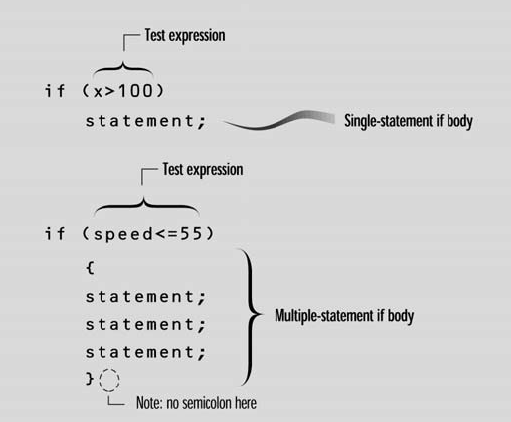
if( x > 100 )

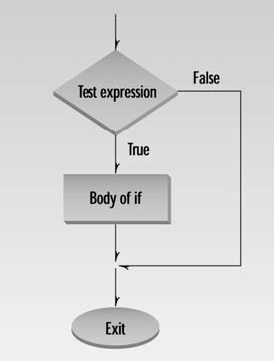
cout << “That number is greater than 100\n”;

return 0;

}

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





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

Enter a number: 2000

That number is greater than 100

**Multiple Statements in the if Body**

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

// if2.cpp

// demonstrates IF with multiline body

#include <iostream>

using namespace std;

int main()

{

int x;

cout << “Enter a number: “;

cin >> x;

if( x > 100 )

{

cout << “The number “ << x;

cout << “ is greater than 100\n”;

}

return 0;

}

Enter a number: 12345

The number 12345 is greater than 100

**Nesting ifs Inside Loops**

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

// prime.cpp

// demonstrates IF statement with prime numbers

#include <iostream>

using namespace std;

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

int main()

{

unsigned long n, j;

cout << “Enter a number: “;

cin >> n; //get number to test

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

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

{ //it’s divisible by j

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

exit(0); //exit from the program

}

cout << “It’s prime\n”;

return 0;

}

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

Enter a number: 13

It’s prime

Enter a number: 22229

It’s prime

Enter a number: 22231

It’s not prime; divisible by 11

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

**Library Function exit()**

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

ေနာက္အပတ္ေတြမွာေတာ႔ က်န္ရွိေနေသးတဲ႔ if-else အေၾကာင္းကို ဆက္လက္ ေဆြးေႏြးသြားပါမယ္ခင္ဗ်ာ။ ဤအခန္းဆက္ ေဆာင္းပါးမ်ားဟာ IT ေက်ာင္းသားမ်ားအတြက္ အထူးရည္ရြယ္ၿပီး ေရးသားထားတာ ျဖစ္ပါတယ္။ သင္ရိုးကို ေရးသားျခင္း ျဖစ္တဲ႔အတြက္ တိုက္ရိုက္ ဘာသာျပန္ဆိုျခင္းနည္းကို ေနရာေတာ္ေတာ္မ်ားမ်ားမွာ သံုးထားပါတယ္။ 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