**OOP in C++ : အပိုင္း (၁၆) - Conditional Operator and Logical Operators အပိုင္း (၂)**

ဒီအပတ္ သင္ခန္းစာမွာေတာ႔ Logical Operators ေတြအေၾကာင္းကို ေလ႔လာၾကရေအာင္။ ထံုးစံအတိုင္း မၿပီးဆံုးေသးတဲ႔ adventure game ေလးကို feature အသစ္ေလးေတြ ထပ္ထည္႔ၾကမယ္ေလ။

**Logical OR Operator**

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

// advenor.cpp

// demonstrates OR logical operator

#include <iostream>

using namespace std;

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

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

int main()

{

char dir=’a’;

int x=10, y=10;

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

{

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

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

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

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

dir = getche(); //get direction

switch(dir)

{

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

case ‘s’: y++; break;

case ‘e’: x++; break;

case ‘w’: x--; break;

} //end switch

} //end while

return 0;

} //end main()

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

**Logical NOT Operator**

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

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

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

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

if( !(x%7) && !(y%7) ) // if not x%7 and not y%7 ရလာဒ္ကေတာ႔ အတူတူပဲ ျဖစ္ပါတယ္။

အရင္ သင္ခန္းစာေတြတုန္းက logical operator && နဲ႔ || ဟာ relational operators ေတြထက္ precedence နိမ္႔တယ္ဆိုတာကို ေလ႔လာခဲ႔ၾကပါတယ္။ ဒါဆို အခု ဘာလို႔ x%7 နဲ႔ y%7 ကို လက္သဲကြင္း မျဖစ္မေန ခတ္ဖို႔ လိုလာတာလဲ? တကယ္ေတာ႔ ! ဟာ logical operator ျဖစ္ေပမယ္႔ unary operator ျဖစ္တာေၾကာင္႔ relational operators ေတြထက္ precedence ပိုျမင္႔ေနလို႔ပဲ ျဖစ္ပါတယ္။

**Precedence Summary**

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

***Operator type Operators Precedence***

Unary !, ++, ––, +, – Highest

Arithmetic Multiplicative \*, /, %

Additive +, –

Relational Inequality <, >, <=, >=

Equality ==, !=

Logical And &&

Or ||

Conditional ?:

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

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

ေနာက္အပတ္ေတြမွာေတာ႔ အျခား **Control Statements** ေတြ 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