**OOP in C++ : အပိုင္း (၂၈) - Functions (6)**

Function ေတြဟာ C++ ရဲ႕ အေျခခံက်တဲ႔ အစိတ္အပိုင္းေတြ ျဖစ္တဲ႔ အတြက္ အေသးစိတ္ သိရွိထားဖို႔လည္း လိုအပ္ပါတယ္။ ဒါေၾကာင္႔ ဒီအပတ္မွာလည္း function ေတြ အေၾကာင္းကိုပဲ ဆက္လက္ေဆြးေႏြးသြားမွာ ျဖစ္ပါတယ္။

**Reference Arguments**

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

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

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

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

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

**Passing Simple Data Types by Reference**

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

// ref.cpp

// demonstrates passing by reference

#include <iostream>

using namespace std;

int main()

{

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

float number, intpart, fracpart; //float variables

do {

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

cin >> number;

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

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

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

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

return 0;

}

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

// intfrac()

// finds integer and fractional parts of real number

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

{

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

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

fracp = n - intp; //subtract integer part

}

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

Enter a real number: 99.44

Integer part is 99, fractional part is 0.44

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

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

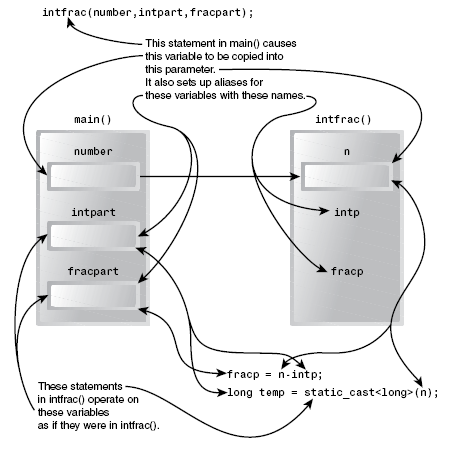
integer type ဟာ ကိန္းျပည္႔ပိုင္းေတြကိုပဲ သိမ္းဆည္းႏိုင္တဲ႔ အတြက္ ဒီနည္းကိုသံုးၿပီး ဒႆမဂဏန္းေတြကို ထိထိေရာက္ေရာက္ ျဖတ္ခ်ထားခဲ႔ႏိုင္ပါတယ္။ ရလာတဲ႔ ရလဒ္ကို ေနာက္တစ္ခါ cast ျပန္လုပ္ၿပီး ေအာက္ပါအတိုင္း float ျပန္ေျပာင္းလိုက္ပါတယ္-

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

ဒႆမေနာက္က တန္ဖိုးကို ရွာဖို႔အတြက္ကေတာ႔ မူရင္းတန္ဖိုးကို ကိန္းျပည္႔တန္ဖိုး ႏုတ္ေပးၿပီး အလြယ္တကူ ရွာႏိုင္ပါတယ္။

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

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



Refrence arguments ေတြကို ampersand (&) အသံုးျပဳရပါတယ္။ ဥပမာ -

float& intp

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

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

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

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

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

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

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

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

**A More Complex Pass by Reference**

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

// reforder.cpp

// orders two arguments passed by reference

#include <iostream>

using namespace std;

int main()

{

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

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

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

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

order(n3, n4);

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

cout << “n2=” << n2 << endl;

cout << “n3=” << n3 << endl;

cout << “n4=” << n4 << endl;

return 0;

}

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

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

{

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

{

int temp = numb1; //swap them

numb1 = numb2;

numb2 = temp;

}

}

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

n1=11

n2=99

n3=22

n4=88

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

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

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

**Passing Structures by Reference**

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

// referst.cpp

// demonstrates passing structure by reference

#include <iostream>

using namespace std;

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

struct Distance //English distance

{

int feet;

float inches;

};

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

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

void engldisp( Distance ); //declarations

int main()

{

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

Distance d2 = { 10, 5.5 };

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

cout << “\nd2 = “; engldisp(d2);

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

scale(d2, 0.25);

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

cout << “\nd2 = “; engldisp(d2);

cout << endl;

return 0;

}

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

// scale()

// scales value of type Distance by factor

void scale( Distance& dd, float factor)

{

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

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

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

}

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

// engldisp()

// display structure of type Distance in feet and inches

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

{

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

}

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

d1 = 12’-6.5”

d2 = 10’-5.5”

d1 = 6’-3.25”

d2 = 2’-7.375”

function scale() ကို ေခၚယူတဲ႔ ကုဒ္ႏွစ္ေၾကာင္းကေတာ႔ ေအာက္ပါအတိုင္း ျဖစ္ပါတယ္-

scale(d1, 0.5);

scale(d2, 0.25);

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

d1 = scale(d1, 0.5);

**Notes on Passing by Reference**

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

ေနာက္အပတ္ေတြမွာလဲ က်န္ရွိေနတဲ႔ function အေၾကာင္းကိုပဲ 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://engineer4myanmar.blogspot.com

ေတာ္ဝင္ ကြန္ပ်ဴတာ စင္တာ

၁၇၉ စ၊ သုမဂၤလာ၊ ေစ်းေလး အေနာက္ဘက္၊ ျပင္ဦးလြင္ၿမိဳ႕