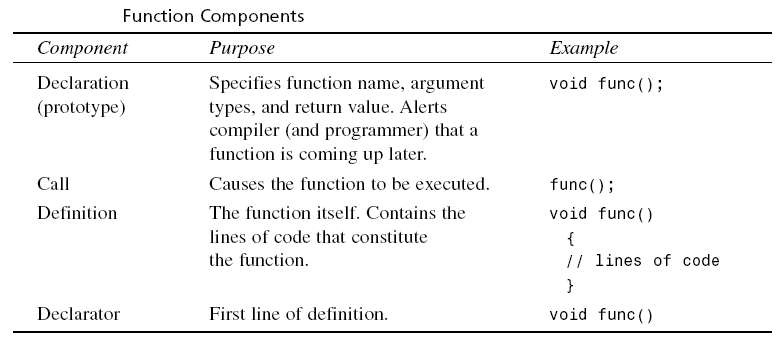
**OOP in C++ : အပိုင္း (၂၄) - Functions (2)**

ဒီအပတ္မွာလည္း function ေတြ အေၾကာင္းကို ဆက္လက္ေဆြးေႏြးသြားမွာ ျဖစ္ပါတယ္။ ေအာက္က ဇယားမွာ function တစ္ခု ေရးသားဖို႔ လိုတဲ႔ အစိတ္အပိုင္းမ်ားကို ရွင္းျပေပးထားပါတယ္။

****

**Comparison with Library Functions**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ library functions အခ်ိဳ႕ကို အေစာပိုင္း သင္ခန္းစာမ်ားမွာ အသံုးျပဳခဲ႔ၾကၿပီး ျဖစ္ပါတယ္။ ဥပမာ ch=getche(); ဆိုတဲ႔ ကုဒ္မ်ိဳးပဲ ျဖစ္ပါတယ္။ ဒါဆိုရင္ အဲဒီ library function getche() အတြက္ declaration နဲ႔ definition ေတြက ဘယ္မွာပါလဲ? တကယ္ေတာ႔ အဲဒီလို library functions ေတြ အတြက္ declaration ေတြကို ပရိုဂရမ္ေတြရဲ႕ အစမွာ ပါတဲ႔ header file (conio.h, for getche()) ေတြထဲမွာ သတ္မွတ္ေပးထားတာ ျဖစ္ပါတယ္။ သူတို႔အတြက္ definition ေတြကေတာ႔ executable code ေတြျဖစ္ေအာင္ compile လုပ္ေပးထားၿပီး library file ေတြထဲမွာ ထည္႔သြင္းထားကာ ကၽြန္ေတာ္တို႔ ေရးသားထားတဲ႔ C++ source file ကို build လုပ္တဲ႔အခ်ိန္က်မွ compiler က အလိုအေလ်ာက္ ခ်ိတ္ဆက္ၿပီး link ျပဳလုပ္ေပးမွာ ျဖစ္ပါတယ္။

ဒါေၾကာင္႔ library function ေတြကို အသံုးျပဳတဲ႔ အခါမွာ declaration နဲ႔ definition ေတြကို ကၽြန္ေတာ္တို႔ ကိုယ္တိုင္ ေရးသားေပးစရာ မလိုအပ္ဘဲ ပရိုဂရမ္ အစပိုင္းမွာ သက္ဆိုင္ရာ header file ကို ေၾကျငာေပးၿပီး သံုးစြဲႏိုင္တာ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ကၽြန္ေတာ္တို႔ ကိုယ္ပိုင္ functions ေတြ ေရးသားတဲ႔ အခါမွာေတာ႔ declaration နဲ႔ definition ေတြကို ယခင္ အပတ္က ဥပမာ ပရိုဂရမ္ေတြထဲကလိုမ်ိဳး ထည္႔သြင္းေရးသားေပးရမွာပါ။

**Eliminating the Declaration**

ေနာက္ထပ္ function ေရးနည္း တစ္မ်ိဳးကေတာ႔ function call မတိုင္မီ function declaration ကို မေရးေတာ႔ပဲ အဲဒီေနရာမွာ function definition ကို တန္းေရးလိုက္တာပဲ ျဖစ္ပါတယ္။ ေအာက္က table2.cpp ပရိုဂရမ္ေလးမွာ starline() ကို ျပန္ေရးျပထားပါတယ္။

// 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;

}

ဒီနည္းက function declaration ကို ျဖဳတ္ပစ္လိုက္လို႔ ပရိုဂရမ္ တိုတိုေလးေတြ ေရးဖို႔အတြက္ ရိုးရွင္းတဲ႔ နည္းလမ္း တစ္ခုပဲ ျဖစ္ပါတယ္။ ဒါေပမယ္႔ ဒီနည္းက functions ေတြ အမ်ားအျပားရွိလာတဲ႔ အခါမွာ function တစ္ခုကေန ေနာက္တစ္ခုကို ေခၚယူႏိုင္တာ ျဖစ္တဲ႔အတြက္ အစဥ္အတိုင္း ျဖစ္ေအာင္ စဥ္းစားရတဲ႔ အတြက္ အဆင္မေျပေတာ႔ပါဘူး။ တစ္ခါတေလမွာ မျဖစ္ႏိုင္တဲ႔ အေျခအေနမ်ိဳးကိုေတာင္ ၾကံဳရတတ္ပါတယ္။ ဒါ႔အျပင္ ပရိုဂရမ္မာ အမ်ားစုဟာ ပထမဆံုး အလုပ္လုပ္မယ္႔ main() function ကို အစမွာ ထားခ်င္ၾကပါတယ္။ ဒါေၾကာင္႔ declarations ေတြ အသံုးျပဳရတဲ႔ ပထမနည္းလမ္းကိုပဲ အသံုးမ်ားၾကပါတယ္။

**Passing Arguments to Functions**

An *argument* is a piece of data (an int value, for example) passed from a program to the function.

Arguments allow a function to operate with different values, or even to do different things, depending on the requirements of the program calling it.

Argument ဆိုသည္မွာ ပရိုဂရမ္မွ function သို႔ ထည္႔သြင္းေပးလိုက္ေသာ data အစိတ္အပိုင္း တစ္ခုျဖစ္သည္။ ထိုိသို႔ argument မ်ား ထည္႔သြင္းေပးျခင္းျဖင္႔ function တစ္ခုကို values အမ်ိဳးမ်ိဳး ျဖင္႔ လုပ္ေဆာင္ႏိုင္ရန္ သို႔မဟုတ္ ပရိုဂရမ္၏ လိုအပ္ခ်က္အရ မတူညီေသာ လုပ္ေဆာင္ခ်က္မ်ားကို လုပ္ေဆာင္ႏိုင္ရန္ ေဆာင္ရြက္ေပးသည္။

**Passing Constants**

အထက္က ဥပမာတြင္ အသံုးျပဳခဲ႔ေသာ starline() ဆိုသည္႔ function သည္ ၾကယ္ ၄၅ လံုးကို အျမဲတမ္း print ထုတ္ေပးသျဖင္႔ လိုသလို ေျပာင္းလဲ အသံုးခ်ႏိုင္စြမ္း နည္းပါးသည္။ ကၽြန္ေတာ္တို႔ အေနျဖင္႔ ထိုကဲ႔သို႔ အေသ ေရးသားထားေသာ function မ်ားထက္ အလိုရွိေသာ character ကို အလိုရွိေသာ အေရအတြက္ print ထုတ္ေပးမည္႔ function မ်ိဳးကို ပိုမိုလိုအပ္ေပသည္။ ေအာက္ပါ tablearg.cpp ပရိုဂရမ္ တြင္ ထိုကဲ႔သို႔ေသာ function မ်ိဳးကို ေရးျပထားပါသည္။ အလိုရွိေသာ character ႏွင္႔ အလိုရွိေသာ အႀကိမ္အေရအတြက္ကိုမူ arguments မ်ားအေနျဖင္႔ ထည္႔သြင္းေပးရမည္ ျဖစ္သည္။

// 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 ပရိုဂရမ္ေလးမွာ သံုးထားတဲ႔ function က repchar() ျဖစ္ၿပီး ေအာက္ပါအတိုင္း declare လုပ္ထားပါတယ္ -

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

Declaration မွာ လက္သဲကြင္းထဲက char နဲ႔ int ဆိုတာကေတာ႔ အဲဒီ function ထဲကို ထည္႔သြင္းေပးလိုက္မယ္႔ data ေတြရဲ႕ data types ေတြပဲ ျဖစ္ပါတယ္။

Function call ျပဳလုပ္တဲ႔ အခါမွာေတာ႔ သတ္မွတ္ထားတဲ႔ တန္ဖိုးေတြ (ဒီေနရာမွာေတာ႔ constatns ေတြ) ကို လက္သဲကြင္းရဲ႕ သက္ဆိုင္ရာေနရာေတြမွာ ေကာ္မာ ခံၿပီး ထည္႔သြင္းေပးလိုက္မွာပါ။ ဥပမာ-

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

ဒီ statement ေလးက repchar() ကို dash အလံုးေရ ၄၃ လံုး print ထုတ္ေပးဖို႔ ညႊန္ၾကားလိုက္တာပဲ ျဖစ္ပါတယ္။ Function call မွာ ထည္႔သြင္းေပးလိုက္တဲ႔ data ေတြရဲ႕ type ဟာ declaration မွာ သတ္မွတ္ေပးထားတဲ႔ အတိုင္း ျဖစ္ဖို႔ေတာ႔ လိုပါတယ္။ ဒါေၾကာင္႔ ပထမဆံုး ထည္႔သြင္းေပးရမယ္႔ argument ဟာ char ျဖစ္ရမွာ ျဖစ္ၿပီး ေနာက္တစ္လံုးကေတာ႔ int ျဖစ္ရပါမယ္။ ေနာက္ထပ္ function call တစ္ခု ျဖစ္တဲ႔ repchar(‘=’, 23); ကေတာ႔ equal signs ၂၃ ခု ထုတ္ေပးဖို႔ ညႊန္ၾကားခ်က္ ျဖစ္ပါတယ္။ tablearg.cpp ရဲ႕ output ကို ေလ႔လာၾကည္႔ၾကရေအာင္။

-------------------------------------------

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 ေတြကို ေခၚယူတဲ႔ ပရိုဂရမ္က လိုအပ္တဲ႔ arguments ေတြ (ဥပမာ '-' နဲ႔ 43) ေတြကို ထည္႔သြင္းေပးလိုက္ရမွာ ျဖစ္ပါတယ္။ ၄င္းတန္ဖိုးေတြကို function ထဲက အသံုးျပဳထားတဲ႔ parameters ေတြလို႔ ေခၚတဲ႔ variables ေတြ (ဥပမာ ch နဲ႔ n) ထဲကို ကူးယူလိုက္မွာ ျဖစ္ပါတယ္။ (ပရိုဂရမ္မာ အမ်ားစုဟာ argument နဲ႔ parameter ေတြကို အလဲအလွယ္လုပ္ၿပီး ေျပာဆို သံုးစြဲတတ္ၾကပါတယ္။)

Function definition မွာေတာ႔ လက္သဲကြင္းထဲမွာ parameters ေတြရဲ႕ data types ေတြအျပင္ names ေတြကိုပါ ထည္႔သြင္းေပးရမွာ ျဖစ္ပါတယ္။ ဥပမာ-

void repchar(char ch, int n) //declarator specifies parameter

//names and data types

အဲဒီ parameters ေတြ (ch နဲ႔ n) ကို function ထဲမွာ သာမန္ variable ေတြ သံုးသလိုပဲ သံုးစြဲႏိုင္ပါတယ္။ ဆိုလိုတာက အဲဒီလို လက္သဲကြင္းထဲမွာ ေရးလိုက္ျခင္းဟာ ေအာက္ပါအတိုင္း ေၾကျငာလိုက္တာနဲ႔ အတူတူပဲ ျဖစ္ပါတယ္။

char ch;

int n;

Funciton ကို call လုပ္တဲ႔အခါမွာေတာ႔ အဲဒီ parameters ေတြထဲကို calling program က ထည္႔သြင္းေပးလိုက္တဲ႔ တန္ဖိုးေတြ အလိုအေလ်ာက္ initialized လုပ္ေပးမွာ ျဖစ္ပါတယ္။

ေနာက္အပတ္ေတြမွာလဲ function 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://engineer4myanmar.blogspot.com

ေတာ္ဝင္ ကြန္ပ်ဴတာ စင္တာ

၁၇၉ စ၊ သုမဂၤလာ၊ ေစ်းေလး အေနာက္ဘက္၊ ျပင္ဦးလြင္ၿမိဳ႕