**OOP in C++ : အပိုင္း (၇) - Operators**

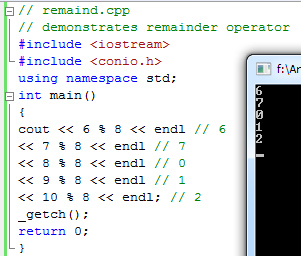
ယခင္ သင္ခန္းစာမ်ားမွာ data types conversion ျပဳလုပ္ပံုမ်ားကို ရွင္းျပခဲ႔ပါတယ္။ ယခု သင္ခန္းစာမွာေတာ႔ Operators မ်ားအေၾကာင္း ေဆြးေႏြး တင္ျပသြားမွာပါ။

**Arithmetic Operators**

C++ မွာ ပံုမွန္ arithmetic operator ၄ မ်ိဳးျဖစ္တဲ႔ +. - .\*. / တို႔ကို အသံုးျပဳၿပီး ေပါင္းျခင္း၊ ႏုတ္ျခင္း၊ ေျမွာက္ျခင္း၊ စားျခင္းမ်ားကို ျပဳလုပ္ေပးပါတယ္။ အဲဒီ arithmetic operators ေတြဟာ အျခား ပရိုဂရမ္းမင္း ဘာသာစကား ေတြမွာလိုပဲ algebra သခၤ်ာမွာ သံုးတဲ႔ပံုစံမ်ိဳးကို data types အမ်ိဳးမ်ိဳးမွာ အသံုးခ်ႏိုင္သလို အျခား arithmetic operators မ်ားလည္း ရွိပါေသးတယ္။

**The Remainder Operator**

Remainder Operator (%) ဟာ integer အမ်ိဳးအစား variable (char, short, int, long) ေတြအတြက္ပဲ အသံုးခ်လို႔ ရပါတယ္။ တခါတေလမွာ modulus operator လို႔လည္း ေခၚတတ္ၾကပါတယ္။ ၄င္းဟာ ဂဏန္းတစ္ခုကို အျခား ဂဏန္းတစ္ခုနဲ႔ စားလို႔ ရလာတဲ႔ အၾကြင္း တန္ဖိုးကို ရွာေပးပါတယ္။ ေအာက္မွာ ေဖာ္ျပထားတဲ႔ remaind.cpp ကို ေလ႔လာၾကည္႔ရေအာင္။



အထက္ပါ ပရိုဂရမ္ေလးမွာ ၆ ကေန ၁၀ အထိ ဂဏန္းေတြကို remainder operator ကို သံုးၿပီး ၈ နဲ႔စားလို႔ရလာတဲ႔ အၾကြင္းေတြကို ရွာထားပါတယ္။ အေျဖေတြကေတာ႔ ၆၊ ၇၊ ၀၊ ၁ နဲ႔ ၂ တို႔ ျဖစ္ပါတယ္။ Remainder operator ကို အေျခအေန အမ်ိဳးမ်ိဳးမွာ သံုးႏိုင္ပါတယ္။ ေနာက္ပိုင္း အလ်င္းသင္႔သလို ဥပမာမ်ားနဲ႔ ရွင္းလင္းသြားပါဦးမယ္။ ramaind.cpp မွာပါတဲ႔ **cout<<6%8** ဆိုတဲ႔ ကုဒ္မွာ remainder operator ကို ပထမဦးဆံုး အလုပ္ စလုပ္ပါတယ္။ ဘာျဖစ္လို႔လဲ ဆိုေတာ႔ **%** operator က **<<** operator ထက္ precedence ပိုျမင္႔လို႔ပဲ ျဖစ္ပါတယ္။ တကယ္လို႔ ပိုေသခ်ာ ေစခ်င္တယ္ ဆိုရင္ေတာ႔ လက္သဲကြင္း အဖြင္႔အပိတ္ တစ္စံုၾကားမွာ **(6%8)** ဆိုၿပီး ေရးေပးရမွာ ျဖစ္ပါတယ္။

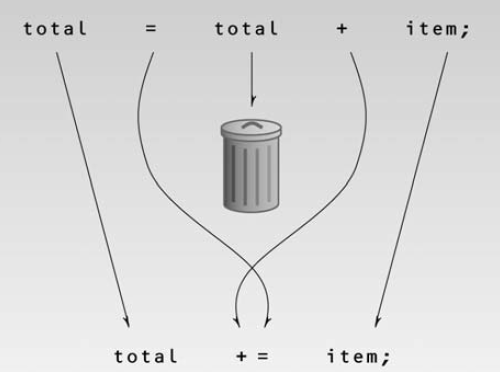
**Arithmetic Assignment Operator**

C++ ကို ေလ႔လာတဲ႔ အခါမွာ ကုဒ္ေတြကို အတိုခ်ံဳ႕ေရးႏိုင္တဲ႔ နည္းလမ္းအမ်ားအျပား ထည္႔သြင္းထားတာကို ေတြ႔ရွိရပါတယ္။ အဲဒီထဲက တစ္ခုကေတာ႔ arithmetic assignment operator ပဲ ျဖစ္ပါတယ္။ ၄င္းဟာ C++ ကို တမူထူးျခားတဲ႔ အသြင္ ျဖစ္ေစပါတယ္။

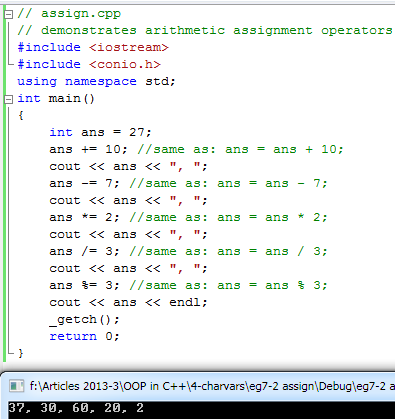
ပံုမွန္အားျဖင္႔ ပရိုဂရမ္းမင္း ဘာသာစကားေတြမွာ total တန္ဖိုးထဲကို item တန္ဖိုးတစ္ခု ထပ္ေပါင္းထည္႔ဖို႔ အတြက္ total = total + item; လို႔ ေရးေလ႔ ရွိၾကပါတယ္။ ဒါေပမယ္႔ အဲဒီ ကုဒ္ထဲမွာ total ကို ႏွစ္ခါ ထပ္ေရးရတဲ႔ အတြက္ ကုဒ္ကို တိုေအာင္ ေရးခ်င္သူမ်ား ဘဝင္မက် ျဖစ္ရပါတယ္။ ဒါေၾကာင္႔ C++ က ပိုမိုက်စ္လစ္တဲ႔ ေရးနည္းကို ပံ႔ပိုးေပးထားပါတယ္။ Arithmetic operator ကို assignment operator နဲ႔ ေပါင္းစပ္ၿပီး ႏွစ္ခါထပ္ေနတဲ႔ operand ကို ျဖဳတ္ထုတ္လိုက္ပါတယ္။ ဒီနည္းနဲ႔ အထက္ပါ ကုဒ္ကို ယခုလို တိုတိုရွင္းရွင္း နဲ႔ ျပင္ေရးလို႔ ရပါတယ္ -

Total += item;

ေအာက္မွာျပထားတဲ႔ ပံုေလးကေတာ႔ အဲဒီ ကုဒ္ ႏွစ္မ်ိဳး ထပ္တူညီပံုကို ေဖာ္ျပထားတာပဲ ျဖစ္ပါတယ္။



အထက္ပါ ကုဒ္မွာ arithmetic operator + နဲ႔ assignment operator = ကို ေပါင္းထားတဲ႔ += ကို သံုးျပသြားသလို -=, \*=, /= နဲ႔ %= စတဲ႔ operator မ်ားကိုလဲ အသံုးျပဳႏိုင္ပါတယ္။ ေအာက္ပါ assign.cpp ပရိုဂရမ္ေလးမွာ နမူနာ သံုးျပထားပါတယ္။



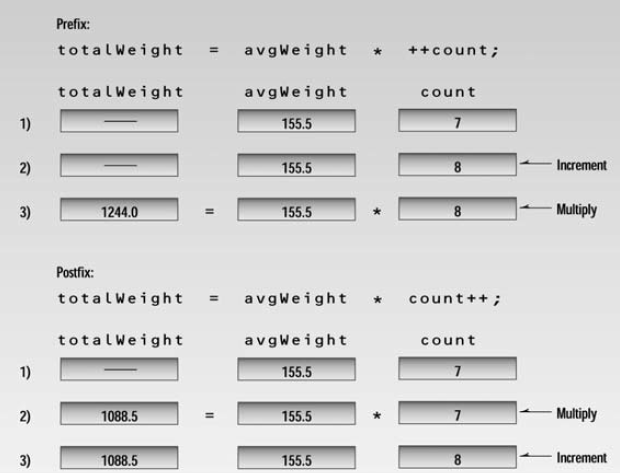
ကၽြန္ေတာ္တို႔အေနနဲ႔ ကုဒ္ေတြ ေရးသားတဲ႔ ေနရာမွာ arithmetic assignment operator ေတြကို မျဖစ္မေန အသံုးျပဳဖို႔ေတာ႔ မလိုပါဘူး။ ဒါေပမယ္႔ အဲဒီ operator ေတြဟာ C++ ရဲ႕ common feature ေတြ ျဖစ္တဲ႔အျပင္ ကုဒ္ေတြကို တိုရင္း လိုရွင္းျဖစ္ေအာင္ ကူညီေပးတဲ႔အတြက္ အလ်င္းသင္႔သလို အသံုးျပဳေစလိုပါတယ္။ ကၽြန္ေတာ္႔ သင္ခန္းစာေတြမွာလည္း ၄င္းတို႔ကို မၾကာခဏ ထည္႔သြင္း အသံုးျပဳသြားမွာ ျဖစ္ပါတယ္။

**Increment Operators**

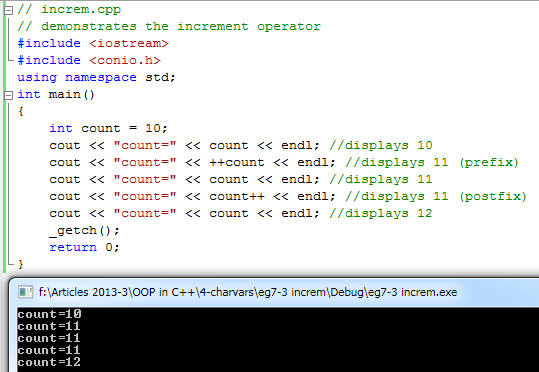
ကၽြန္ေတာ္တို႔ အေနနဲ႔ arithmetic assignment operator ကို မသိခင္တုန္းက (count ဆိုတဲ႔) variable ထဲကို တန္ဖိုး တစ္ခု တိုးခ်င္ရင္ count = count + 1; ဆိုၿပီး ေရးခဲ႔ၾကပါတယ္။ arithmetic assignment operator ကို သံုးတတ္လာေတာ႔ count += 1; ဆိုၿပီး အတိုခ်ံဳ႕ ေရးႏိုင္လာပါတယ္။ ဒါေပမယ္႔ C++ မွာ ဒီထက္ပိုၿပီး တိုေအာင္ ခ်ံဳ႕ေရးလို႔ရတဲ႔ increment operator (++) ဆိုတာ ရွိပါတယ္။ ဒါေၾကာင္႔ အထက္ပါ ကုဒ္ေတြကို ++cont; ဆိုၿပီး အတိုဆံုးေရးလို႔ ရလာပါတယ္။ ရလာဒ္ကေတာ႔ တန္ဖိုးကို ၁ တိုးေပးမွာပဲ ျဖစ္ပါတယ္။

**Prefix and Postfix**

Increment operator ကို အသံုးျပဳတဲ႔ ေနရာမွာ variable ရဲ႕ ေရွ႕ကေန သံုးတဲ႔ prefix ပံုစံနဲ႔ ေနာက္ကေန တြဲဘက္သံုးတဲ႔ postfix ပံုစံဆိုၿပီး ႏွစ္မ်ိဳး သံုးလို႔ ရပါတယ္။ အဲဒီ ႏွစ္ခုရဲ႕ ျခားနားခ်က္ကို ေလ႔လာၾကည္႔ၾကရေအာင္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ ကုဒ္ေတြထဲမွာ အျခား operation ေတြနဲ႔ ေရာေထြးၿပီး variable တစ္ခုကို increment လုပ္ရတဲ႔ အေျခအေနမ်ိဳးကို မၾကာခဏ ၾကံဳရတတ္ပါတယ္။ ဥပမာ- totalWeight = avgWeight \* ++count; အဲဒီလို အေျခအေနမ်ိဳးမွာ ျပႆနာက count ကို increment အရင္လုပ္မလား၊ အရင္ ေျမွာက္မလား ဆိုတာပါပဲ။ ဒီကုဒ္အရေတာ႔ prefix notation ကို သံုးထားတဲ႔ အတြက္ increment ကို အရင္လုပ္ၿပီးမွာ avgWeight နဲ႔ ေျမွာက္မွာ ျဖစ္ပါတယ္။ တကယ္လို႔ postfix notation ကိုသံုးၿပီး totalWeight = avgWeight \* count++; လို႔ ေရးသားခဲ႔မယ္ဆိုရင္ေတာ႔ avgWeight နဲ႔ အရင္ေျမွာက္ၿပီးမွ increment လုပ္မွာ ျဖစ္ပါတယ္။ ဒီျဖစ္စဥ္ကို ေအာက္က ပံုမွာ အေသးစိတ္ ရွင္းျပထားပါတယ္။



ေအာက္မွာ ေဖာ္ျပထားတဲ႔ increm.cpp ပရိုဂရမ္ေလးမွာေတာ႔ prefix နဲ႔ postfix ႏွစ္ခုလံုးကို သံုးျပထားတာ ေတြ႔ရပါလိမ္႔မယ္။



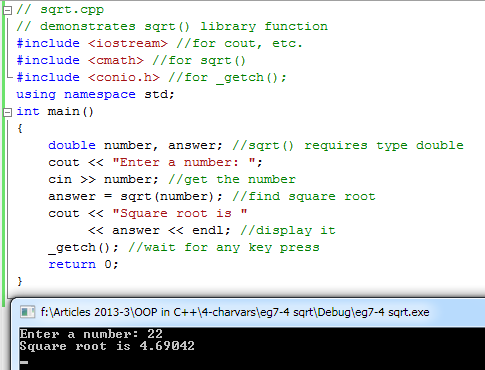
ဒီပရိုဂရမ္ေလးမွာ ပထမဆံုး count ရဲ႕ တန္ဖိုး 10 ကို cout သံုးၿပီး ထုတ္ျပပါတယ္။ ဒုတိယ စာေၾကာင္းမွာ prefix ကိုသံုးၿပီး count ကို increment လုပ္ထားတဲ႔ အတြက္ cout မထုတ္ခင္မွာ count တန္ဖိုး 10 ကေန 11 ျဖစ္သြားၿပီး အေျဖက 11 ျဖစ္လာပါတယ္။ ေနာက္တစ္ေၾကာင္းမွာ postfix ကို သံုးၿပီး increment လုပ္တာ ျဖစ္တဲ႔အတြက္ cout ထုတ္တဲ႔ အခ်ိန္မွာ increment မလုပ္ရေသးပါဘူး။ ဒါေၾကာင္႔ 11 ကိုပဲ ထုတ္ေပးတာပါ။ ဒါေပမယ္႔ cout ထုတ္ၿပီးၿပီးခ်င္း increment လုပ္ေပးမွာ ျဖစ္တဲ႔အတြက္ ေနာက္တစ္ေၾကာင္းနဲ႔ cout အထုတ္မွာ တန္ဖိုးက 12 ျဖစ္ေနတာကို ေတြ႔ရမွာ ျဖစ္ပါတယ္။

**The Decrement (--) Operator**

Decrement operator (--) လုပ္ေဆာင္ပံုေတြက increment operator လိုပဲ ျဖစ္ပါတယ္။ တစ္ခုကြဲျပားတာက decrement operator အေနနဲ႔ variable ရဲ႕ တန္ဖိုးထဲကေန 1 ႏုုတ္ေပးလိုက္တာပဲ ျဖစ္ပါတယ္။ ၄င္းကိုလဲ prefix နဲ႔ postfix ပံုစံ ႏွစ္မ်ိဳးနဲ႔ အသံုးျပဳႏိုင္ပါတယ္။

**Library Functions**

C++ ရဲ႕ လုပ္ေဆာင္မႈ အေတာ္မ်ားမ်ားကို library functions ေတြ ေခၚယူ အသံုးျပဳျခင္းအားျဖင္႔ ေရးသားေလ႔ ရွိပါတယ္။ ၄င္း functions မ်ားကို အသံုးျပဳၿပီး ဖိုင္မ်ားကို ရယူသံုးစြဲျခင္း၊ သခၤ်ာဆိုင္ရာ တြက္ခ်က္မႈမ်ား ျပဳလုပ္ျခင္း၊ data မ်ား conversion ျပဳလုပ္ျခင္းႏွင္႔ အျခားေျမာက္မ်ားလွစြာေသာ လုပ္ေဆာင္မႈမ်ားကို ေဆာင္ရြက္ေပးႏိုင္မွာ ျဖစ္ပါတယ္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ ဒီအခန္းမွာေတာ႔ library function အခ်ိဳ႕ အသံုးျပဳပံုကို sqrt.cpp ဆိုတဲ႔ နမူနာ ပရိုဂရမ္ေလးသံုးၿပီး မိတ္ဆက္ေပးခ်င္ပါတယ္။ ဒီပရိုဂရမ္ေလးမွာ sqrt() ဆိုတဲ႔ library function ေလးကို သံုးၿပီး user ထည္႔သြင္းေပးလိုက္တဲ႔ ဂဏန္းေတြကို square root ရွာေပးမွာ ျဖစ္ပါတယ္။



ပထမဆံုး user ဆီက ေတာင္းလို႔ ရလာတဲ႔ တန္ဖိုးကို sqrt() ဆိုတဲ႔ library function ရဲ႕ ေနာက္က လက္သဲကြင္း အဖြင္႔အပိတ္ထဲကို argument အေနနဲ႔ ထည္႔သြင္းေပးလိုက္တဲ႔အခါမွာ library function က အဲဒီ argument ရဲ႕ square root တန္ဖိုးကို တြက္ထုတ္ၿပီး ျပန္ေပးပါတယ္။ အဲဒီ တန္ဖိုးကို assignment operator (=) သံုးၿပီး answer ဆိုတဲ႔ variable ထဲကို ထည္႔သြင္းေပးတာ ျဖစ္ပါတယ္။ Library function တစ္ခုကို အသံုးျပဳတဲ႔ ေနရာမွာ ထည္႔သြင္းေပးရမယ္႔ argument နဲ႔ ျပန္ရလာမယ္႔ return value ေတြဟာ မွန္ကန္တဲ႔ data type ေတြ ျဖစ္ေနဖို႔ လိုပါတယ္။ ဒီဥပမာမွာ ပါတဲ႔ sqrt() function မွာေတာ႔ argument ေရာ return value ပါ double ျဖစ္တဲ႔အတြက္ variables ႏွစ္ခုစလံုးကို double နဲ႔ အသံုးျပဳခဲ႔တာ ျဖစ္ပါတယ္။

**Header Files**

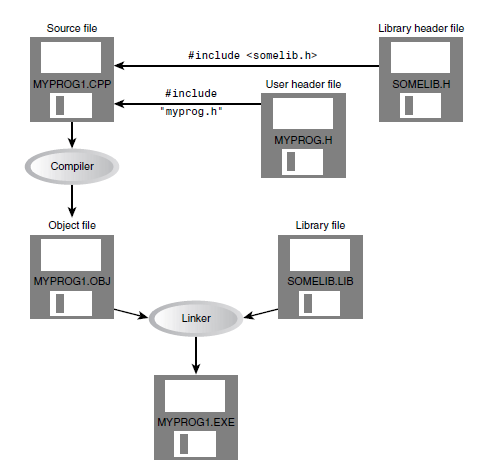
ကၽြန္ေတာ္တို႔ အေနနဲ႔ cout, sqrt() စတဲ႔ objects ေတြ၊ library functions ေတြကို ယူသံုးတဲ႔ အခါတိုင္းမွာ သက္ဆိုင္ရာ header file ေတြကို #include သံုးၿပီး ေၾကျငာေပးဖို႔ လိုမွာ ျဖစ္ပါတယ္။ sqrt() ရဲ႕ header file ဟာ cmath ျဖစ္တာေၾကာင္႔ #include<cmath> ဆိုၿပီး ေရးသားေပးရပါမယ္။ အဲဒီလို ေရးသားျခင္းအားျဖင္႔ preprocessor က cmath ဆိုတဲ႔ header file ကို ကၽြန္ေတာ္တို႔ရဲ႕ source file ျဖစ္တဲ႔ sqrt.cpp ထဲကို ေပါင္းစပ္ ထည္႔သြင္းေပးမွာ ျဖစ္ပါတယ္။ တကယ္လို႔ header file ကို ေၾကျငာေပးဖို႔ ေမ႔ေလ်ာ႔ခဲ႔တယ္ဆိုရင္ေတာ႔ compiler ကေန ‘sqrt’ unidentified identifier ဆိုတဲ႔ error message ေပးမွာ ျဖစ္ပါတယ္။

**Library Files**

Compiler တစ္ခုဟာ executable ဖိုင္ (.exe) တစ္ခု တည္ေဆာက္ဖို႔အတြက္ library functions ေတြနဲ႔ objects ေတြ ပါဝင္တဲ႔ ဖိုင္ေတြကို ခ်ိတ္ဆက္ၿပီး ဖန္တီး ယူရတာ ျဖစ္ပါတယ္။ အဲဒီဖိုင္ေတြမွာ အဲဒီ function ေတြနဲ႔ objects ေတြရဲ႕ machine-executable code ေတြ ပါဝင္ပါတယ္။ Library ဖိုင္ေတြဟာ မ်ားေသာအားျဖင္႔ .lib extension နဲ႔ အဆံုးသတ္ေလ႔ ရွိပါတယ္။ sqrt() function ကို အဲဒီလို ဖိုင္ေတြထဲမွာ ထည္႔သြင္းထားတာ ျဖစ္ပါတယ္။ အဲဒီ function ေတြကို linker က library ဖိုင္ကေနၿပီး ခြဲထုတ္ကာ ပရိုဂရမ္က ေခၚယူ အသံုးျပဳႏိုင္ေအာင္ ခ်ိတ္ဆက္ေပးမွာ ျဖစ္ပါတယ္။ အဲဒီ အေသးစိတ္ အခ်က္အလက္ေတြကို ကၽြန္ေတာ္တို႔ ဘာမွလုပ္ေပးဖို႔ မလိုအပ္ပါဘူး။ Compiler က လိုအပ္သလို ေဆာင္ရြက္ေပးသြားမွာပါ။ ဒါေပမယ္႔ ပရိုဂရမ္မာ တစ္ေယာက္ အေနနဲ႔ကေတာ႔ အဲဒီဖိုင္ေတြအေၾကာင္းကို သိထားဖို႔ လိုအပ္မွာ ျဖစ္ပါတယ္။

**Header Files and Library Files**

Library ဖိုင္နဲ႔ header ဖိုင္ေတြရဲ႕ ဆက္စပ္မႈကို နားလည္ရ ခက္ေနတတ္ပါတယ္။ ဒါေၾကာင္႔ အနည္းငယ္ ထပ္မံ ရွင္းျပလိုပါတယ္။ ကၽြန္ေတာ္တို႔ ေရးသားတဲ႔ ပရိုဂရမ္ထဲမွာ sqrt() ဆိုတဲ႔ library function တစ္ခုကို ထည္႔သံုးခ်င္တယ္ ဆိုပါစို႔။ ဒါဆို အဲဒီ function ပါတဲ႔ library ဖိုင္ကို ကၽြန္ေတာ္တို႔ ပရိုဂရမ္နဲ႔ ခ်ိတ္ဆက္ေပးဖို႔ လိုလာပါတယ္။ အဲဒီလို ျပဳလုပ္ဖို႔ linker က တာဝန္ယူ ေဆာင္ရြက္ေပးပါတယ္။ ဒါေပမယ္႔ ဒီမွာ မၿပီးေသးပါဘူး။ ကၽြန္ေတာ္တို႔ ေရးထားတဲ႔ source ဖိုင္ ထဲမွာ အဲဒီ library ဖိုင္ထဲမွာ ပါတဲ႔ functions ေတြရဲ႕ နာမည္ေတြ၊ အမ်ိဳးအစားေတြနဲ႔ အျခား အခ်က္အလက္ေတြကို သိဖို႔ လိုအပ္ေနပါေသးတယ္။ အဲဒီအခ်က္အလက္ေတြဟာ တကယ္ေတာ႔ header file ထဲမွာ စုစည္းထားတာ ျဖစ္ၿပီး header file တစ္ခုစီမွာ သက္ဆိုင္ရာ functions အခ်ိဳ႕ရဲ႕ အခ်က္အလက္မ်ားကို သိမ္းဆည္းထားပါတယ္။ functions ေတြကိုေတာ႔ စုစည္းၿပီး library file အေနနဲ႔ စုစည္းထားေလ႔ ရွိတယ္။ အဲဒီ library ဖိုင္ တစ္ခုအတြက္ header file အမ်ားအျပား ခြဲျခား ထားတတ္ပါတယ္။ ဥပမာ iostream header file ထဲမွာ cout အစရွိတဲ႔ I/O functions အမ်ားအျပားနဲ႔ objects ေတြရဲ႕ အခ်က္အလက္ေတြကို စုစည္းထားသလို cmath header file ထဲမွာေတာ႔ sqrt() အစရွိတဲ႔ သခၤ်ာ functions ေတြရဲ႕ အခ်က္အလက္ေတြ ပါဝင္ပါတယ္။ တကယ္လို႔ ကၽြန္ေတာ္တို႔အေနနဲ႔ string ႏွစ္ခုကို ႏိႈင္းယွဥ္တဲ႔ strcpy() ဆိုတဲ႔ function ကို အသံုးျပဳခ်င္တယ္ ဆိုရင္ေတာ႔ string.h ကို ေၾကျငာေပးဖို႔ လိုမွာ ျဖစ္ပါတယ္။ C++ မွာ header file ေတြ အျမဲတမ္း အသံုးျပဳေလ႔ ရွိၿပီး ကၽြန္ေတာ္တို႔ အေနနဲ႔ library function တစ္ခု၊ ဒါမွမဟုတ္ ႀကိဳတင္သတ္မွတ္ ေရးသားထားတဲ႔ object ေတြ operator ေတြကို သံုးတဲ႔ အခ်ိန္တိုင္းမွာ သက္ဆိုင္ရာ header file ေတြကို ေၾကျငာေပးရပါမယ္။ ေအာက္က ပံုေလးမွာ header file နဲ႔ library ဖိုင္ေတြ အသံုးျပဳပံုကို ရွင္းျပထားပါတယ္။



**Two Ways to Use #include**

ကၽြန္ေတာ္တို႔ အေနနဲ႔ #include ကို နည္းလမ္း ႏွစ္မ်ိဳးနဲ႔ အသံုးျပဳႏိုင္ပါတယ္။ ပထမ တနည္းက angle brackets <> ၾကားမွာ filename ေတြကို ေရးတဲ႔ နည္းျဖစ္ပါတယ္။ အထက္က နမူနာ ပရိုဂရမ္ေတြထဲကလို #include<iostream> နဲ႔ #include<cmath> တို႔ကို ေရးခဲ႔သလိုမ်ိုးပါ။ ဒီလို ေရးလိုက္တာနဲ႔ compiler က အဲဒီဖိုင္ေတြကို INCLUDE ဆိုတဲ႔ directory ထဲမွာ သြားရွာမွာ ျဖစ္ပါတယ္။ အဲဒီ directory ထဲမွာ compiler ကို ထုတ္လုပ္သူက ထည္႔ေပးလိုက္တဲ႔ header files ေတြ အကုန္လံုးကို စုစည္းထားတာ ျဖစ္ပါတယ္။ သာမန္အားျဖင္႔ ဒီနည္းကိုပဲ သံုးၾကရပါတယ္။ ဒါေပမယ္႔ တကယ္လို႔ ကၽြန္ေတာ္တို႔ရဲ႕ ကိုယ္ပိုင္ header file ေတြ ေရးသား အသံုးျပဳလာၾကမယ္ ဆိုရင္ေတာ႔ #include”myheader.h” ဆိုတဲ႔ ပံုစံမ်ိဳးနဲ႔ quotation marks “” ၾကားမွာ ေရးသားေပးရမွာ ျဖစ္ပါတယ္။ အဲဒီလို ေရးသားျခင္း အားျဖင္႔ compiler က အဲဒီဖိုင္ကို INCLUDE directory မွာ ရွာမယ္႔အစား current directory ထဲမွာ သြားရွာပါလိမ္႔မယ္။ တကယ္ေတာ႔ current directory ဆိုတာ ကၽြန္ေတာ္တို႔ ေရးသားေနတဲ႔ ပရိုဂရမ္ရဲ႕ source ဖိုင္ ရွိတဲ႔ ေနရာျဖစ္ပါတယ္။ အဲဒီေတာ႔ ကၽြန္ေတာ္တို႔ ေရးသားထားတဲ႔ header file နဲ႔ source ဖိုင္ေတြကို ေနရာတစ္ခုထဲမွာ စုစည္းထားဖို႔လိုပါတယ္။ Multifile Programs မ်ားအေၾကာင္း ေရးသားတဲ႔ အခ်ိန္က်မွ အခုထက္ပိုၿပီး အေသးစိတ္ ေဆြးေႏြးေပးပါဦးမယ္။

ေနာက္အပတ္ေတြမွာေတာ႔ looping ေတြအေၾကာင္းကို ဆက္လက္ ေဆြးေႏြး သြားပါမယ္ခင္ဗ်ာ။ ဤအခန္းဆက္ ေဆာင္းပါးမ်ားဟာ IT ေက်ာင္းသားမ်ားအတြက္ အထူးရည္ရြယ္ၿပီး ေရးသားထားတာ ျဖစ္ပါတယ္။ သိလိုသည္မ်ား ကို 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