C言語プログラマのためのC++入門

平成19年8月5日

目 次

第1章　はじめに

1.1　はじめに

　—後で書く—
ここではC++基本的な部分（ほんの一部）しか説明しない。詳しく勉強したい方は、独習C++とかそういった本を読んでね。
　あと、C++言語はC言語を拡張したものなので、基本的にC言語のコードでも問題はありません。（まあそれだったらC++使う意味ないけどね)

1.2　C言語とC++言語の違い

- #includeの違い, namespaceの存在
- printf,scanfではなくcout, cin
- 変数はどこでも宣言可能
- bool型が存在する
- string型が存在する
- 関数のオーバーロード
- malloc,freeではなくnew,delete
- 参照渡しが存在する
- クラスが存在する
- テンプレートが存在する
- オブジェクト指向プログラミング

第2章　#includeとnamespace

2.1　Hello World!

最初は画面にHelloWorld!と表示するプログラムを作ることにします。まずは何も考えず次のソースコードを入力し、実行できるかどうかのテストをしてみてください。C++のソースファイルは拡張子が.cではなく.cppである事に注意してください。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           cout << "HelloWorld!" << endl;
7           return 0;
8   }
```

このプログラムはC言語で書くと次のようになります。

```
1   #include <stdio.h>
2   int main()
3   {
4           printf("HelloWorld!");
5           return 0;
6   }
```

プログラムをコンパイルしてみてください。うまく実行できたでしょうか？実行結果はどちらも以下のようになります。

実行結果

```
HelloWorld!
```

2.2 #includeの違い

C 言語でいう `#include <stdio.h>` はC++では `#include <iostream>` となります。stdio.h とは違い.h が無い事に注意してください。iostream は InputOutputStream の略で名前の通り基本的な入出力機能を提供するものです。

昔は<iostream>ではなく<iostream.h>でした。なぜ.h が外れたかというと namespace という次の節で説明する機能をつける際にいろいろ面倒な自体が起こったためです。詳しくは namespace の節の余談で。

2.3 namespaceについて

C 言語で大規模なソフトウェアを開発したとしましょう。例えば 100 万行のコードで構成されているような大規模なプログラムを想像してください。ソースファイルも 1000 個くらいに分割されているとします。そこで、こんな感じの関数を作ったとしましょう。

```
1   //program40.c の中身
2
3   void debug()
4   {
5           printf("Debug Function");
6   }
```

```
1   //program235.c の中身
2
3   void debug()
4   {
5           printf("Hello");
6   }
```

この関数自体は全く意味の無いものです。例を簡単にするためこのようなコードとしました。本当は debug 関数にはもっと複雑なコードが書かれていると思ってください。

ここで 100 万行もコードがあると、例のように別の場所で debug 関数が作られてしまうかもしれません。もちろん例のコードは同じ関数名が二つあるのでコンパイルエラーとなってしまいます。

C 言語の場合、このようなエラーを回避するにはどのようにしたらよいでしょうか。もともと使われていた debug 関数の名前を program40_debug という名前に変更しますか？(そうなるとこの関数を呼び出しているすべての箇所を変更する必要がある)

または、program235.c の debug という関数名を debug2 としますか？

こういった問題を回避するために namespace というものが C++には存在します。使い方は簡単です。

namespace の使い方

```
namespace ネームスペースの名前 { プログラム }
```

ネームスペースの名前には好きなものをつけてもらってかまいません。プログラムと書かれた中に関数を書きます。

では、例のプログラムに namespace を適応してみましょう。

program40.cpp の中身

```
1
2
3   namespace nameA{
4           void debug()
5           {
6                   printf("Debug Function");
7           }
8   }
```

//program235.cpp の中身

```
1
2
3   namespace nameB{
4           void debug()
5           {
6                   printf("Hello");
7           }
8   }
```

さて、debug 関数を呼び出してみましょう。(#include などは省略)

```
1   int main()
2   {
3           nameA::debug(); //program40.cpp の debug 関数を呼び出す
4           nameB::debug(); //program235.cpp の debug 関数を呼び出す
5           return 0;
6   }
```

namespace 内の関数の呼び出し

namespace の名前::関数名

このようにして関数を呼び出すことができます。いちいち debug の名前をどうするか、なんて考える必要がなくなります。

けど、もしかしたら貴方はこう思うかもしれません。

「いちいち nameA::関数名 なんて長いものめんどくさいな〜」

こんな面倒くさがり屋の貴方にも便利な機能が存在します。

using の使い方

using namespace 名前空間 (namespace) の名前;

このコードを使うとある特定の名前空間のメンバ（関数や変数）だけが可視状態となります。例えば次のようなコードとなります。

```
1   int main()
2   {
3           using namespace nameA;
4           debug(); //program40.cpp の debug 関数を呼び出す
5           nameB::debug(); //program235.cpp の debug 関数を呼び出す
6           return 0;
7   }
```

また、特定の名前空間の特定の関数だけを可視状態にすることもできます。

using の使い方

using 名前空間の名前::関数または変数

```
1   int main()
2   {
3           using nameA::debug
4           debug(); //program40.cpp の debug 関数を呼び出す
5           nameB::debug(); //program235.cpp の debug 関数を呼び出す
6           return 0;
7   }
```

では、一番初めの HelloWorld プログラムに戻ってみましょう。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           cout << "HelloWorld!" << endl;
7           return 0;
8   }
```

ここではstd という名前空間の中のメンバを使っています。cout は文字を表示する関数（正確に言うとちょっと違うけど）ですが、これはstd という名前空間の中に定義されています。cout なんて短い名前の関数名、もしかしたら自分で作りたいと思うことがあるかも（あるのか？）しれません。そういった場合、標準関数であることを明確にするため、std という名前の名前空間に関数が作られています。

もちろん、

```
1   #include <iostream>
2
3   int main()
4   {
5           std::cout << "HelloWorld!" << endl;
6           return 0;
7   }
```

または、

```
1   #include <iostream>
2   using std::cout;
```

```
3   int main()
4   {
5           cout << "HelloWorld!" << endl;
6           return 0;
7   }
```

とすることもできます。

<iostream.h>の.h が外れた理由は namespace にあります。昔の C++には namespace というものは存在していませんでした。しかし、大規模なプログラムを作っていると、どうしても標準関数の関数名と自分が作ったプログラムの関数名が同じになってしまうという事態が頻繁に（？）発生してしまいました。
この問題を回避するため、namespace が作られ標準関数は標準関数専用の名前空間に入れられました。この名前空間が std です。
<iostream>と<iostream.h>の違いは標準関数が std 名前空間に囲まれているかいないかの違いです。標準関数なんて良く使うもの、いきなり仕様が変えられては困ったものです。そこで、昔の C++コードもちゃんと動くように未だに古い<iostream.h>が残されているわけです。

2.4 演習問題

演習 2.1 次のプログラムはコンパイルすることができない。正しいコードに書き直しなさい。

```
1  #include <stdio.h>
2
3  namespace nameA{
4          int function(int x){
5                  return x+1;
6          }
7  }
8  int main()
9  {
10         printf("value :%d",function(2) );
11         return 0;
12 }
```

演習 2.2 次のプログラムはコンパイルすることができない。正しいコードに書き直しなさい。

```
1  #include <stdio.h>
2
3  namespace nameA{
4          int hoge(int x){
5                  return x+1;
6          }
7
8          int piyo(int y){
9                  return y+2;
10         }
11 }
12
13 int main()
14 {
15         using nameA::hoge;
16         printf("value :%d",hoge(2) );
17         printf("value :%d",piyo(2) );
```

```
18          return 0;
19  }
```

演習 2.3 次のプログラムはコンパイルすることができない。正しいコードに書き直しなさい。

```
1   #include <stdio.h>
2
3   namespace nameA{
4           int hoge(int x){
5                   return x+1;
6           }
7   }
8   void foo()
9   {
10          using namespace nameA;
11          printf("value :%d",hoge(2) );
12  }
13  int main()
14  {
15          foo();
16          printf("value :%d",hoge(2) );
17          return 0;
18  }
```

演習 2.4 次のプログラムはコンパイルすることができない。動作しない理由を答えなさい。

```
1   #include <stdio.h>
2
3   namespace nameA{
4           int hoge(int x){
5                   return x+1;
6           }
7   }
8   namespace nameB{
9           int hoge(int x){
10                  return x+2;
```

```
11          }
12  }
13  int main()
14  {
15          using namespace nameA;
16          using namespace nameB;
17          printf("value :%d",hoge(2) );
18          return 0;
19  }
```

第3章　標準入出力

3.1　cout

またまた HelloWorld プログラムの登場です。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           cout << "HelloWorld!" << endl;
7           return 0;
8   }
```

次は cout の説明をしていきたいと思います。cout は c 言語でいう printf にあたり、画面へ文字を出力することができます。 << は今の段階では cout の中に文字列を放り込むものだと思ってください。

画面への文字の出力

```
cout << "表示したい文字";
```

複数の文字を放り込むこともできます。

```
1   int main()
2   {
3           cout << "HelloWorld!" << "\n" << "goodbye" << endl;
4           return 0;
5   }
```

また、endl というものを cout に放り込んでいますが、これは改行という意味です。[1]

また、文字だけでなく変数や数字を放り込むこともできます。

[1]正確に言うと改行だけではないのですが、今の段階では同じと思ってもらって結構です

```
1   int main()
2   {
3           int i=5;
4           cout << "Value is " << i << endl;
5           return 0;
6   }
```

3.2 cin

cin は C 言語でいう scanf にあたります。

cin の使い方

```
cin >> 代入する変数名;
```

cin から >> を使って変数の中に値を放り込んでいます。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           int i;
7           cout << "何か文字を入力してね :";
8           cin >> i;
9           cout << "入力された文字は" << i << "です。" << endl;
10          return 0;
11  }
```

この例ではキーボードから入力された値を画面に出力します。

第4章　変数について

4.1　変数の宣言について

C++言語ではC言語とは違い変数をどこでも宣言することができます。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           cout << "なにか数字を入力してね：" << endl;
7           int i;
8           cin >> i;
9           cout << "値は" << i << "です" << endl;
10          return 0;
11  }
```

4.2　bool型について

C++では新しい型としてboolというものが存在します。boolはtrueかfalseかを保持する変数です。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           bool hoge = true;
7           if( hoge == false ) {
8                   cout << "false." << endl;
```

```
9           } else {
10                  cout << "true." << endl;
11          }
12          return 0;
13  }
```

例では true. が画面に出力されます。

4.3 string 型について

C 言語では文字列を保持するには char 型の配列を使用しました。しかし、char 型にはいろいろと面倒な問題が含まれていました。

```
1   #include <stdio.h>
2
3   int main()
4   {
5           char buffer[100];
6           strcpy( buffer , "Hello World!" );
7           printf("%s",buffer);
8           return 0;
9   }
```

まず、char 型 100 個の配列となっていますが、もし 100 文字以上の文字が入力された場合どうなるでしょうか？また、文字列をコピーするのにいちいち strcpy 関数を使わなければならないのは非常に面倒です。

これらの問題を回避するため、C++では新たに string 型というものが用意されました。

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7           string buffer;
8           buffer = "Hello World!";
```

```
 9          cout << buffer << endl;
10          return 0;
11  }
```

string 型を使用するためには `#include <string>` を定義する必要があります。buffer に文字列を渡す場合は、int や double の時の値の代入と同じように文字列を代入することができます。また、プログラマが char 型の時のように容量を心配する必要がなくなります。

また、文字列の結合も簡単にできます。

```
 1  #include <iostream>
 2  #include <string>
 3  using namespace std;
 4
 5  int main()
 6  {
 7          string buffer = "Hello ";
 8          string hoge = "World";
 9          buffer = buffer + hoge;
10          cout << buffer << endl;
11  }
```

このプログラムは Hello World を出力します。

また、文字列の比較も簡単です。

```
 1  #include <iostream>
 2  #include <string>
 3  using namespace std;
 4
 5  int main()
 6  {
 7          string buffer = "Hello";
 8
 9          if( hoge == "Hello" ) {
10                  cout << buffer << endl;
11          }
12          return 0;
13  }
```

このプログラムは Hello を出力します。

第5章　OverloadとTemplate

5.1　関数のオーバーロードとは

C 言語で、二つの値を比較して大きいほうを返却する関数を書いたとします。

```
1   int max(int x, int y)
2   {
3           if( x > y )
4                   return x;
5           else
6                   return y;
7   }
```

関数を呼び出す時は、

```
        int i = max( 1 , 2 );
```

などとすればよいでしょう。

さて、ここで問題となるのが、引数に double 型を渡したい場合です。max 関数は引数に int を取っているので、

```
        int i = max( 1.5 , 2.0 );
```

等とすることはできません。こういった場合、C 言語では max の名前を変更するしかありませんでした。

```
1   int max_int(int x, int y)
2   {
3           if( x > y )
4                   return x;
5           else
6                   return y;
7   }
```

```
 8  double max_double(double x, double y)
 9  {
10          if( x > y )
11                  return x;
12          else
13                  return y;
14  }
```

この問題を回避する為に C++言語では同じ関数名でも引数が違えば違う関数として扱うことができるようになりました。このことを関数のオーバーロード (Overload) といいます。

```
 1  int max(int x, int y)
 2  {
 3          if( x > y )
 4                  return x;
 5          else
 6                  return y;
 7  }
 8  double max(double x, double y)
 9  {
10          if( x > y )
11                  return x;
12          else
13                  return y;
14  }
```

呼び出す時は、

```
int i = max( 1,2 );  //max(int,int) が呼ばれる
double j = max( 1.5,2.2 ); //max(double,double) が呼ばれる
```

とします。

5.2 Template とは

5.1 節の max 関数は引数が違うだけで変数の中身は全く同じでした。この場合、わざわざ引数だけを変えた関数をたくさん作るのは面倒です。そこで登場するのが Template です。次のコードを見てください。

```
1   template<typename T>
2   T my_max(T x,T y)
3   {
4           if( x > y )
5                   return x;
6           else
7                   return y;
8   }
```

T 型という新しい型を作り出しています。この型は呼び出し時にコンパイラが自動的に型を判断して処理を行ってくれます。

関数の呼び出しは

```
int i = my_max( 1,2 );
double j = my_max( 1.5,2.2 );
```

となります。引数に 1 を渡した場合、T 型は int 型に置き換えられます。同様に 1.5 を渡した場合、T 型は double 型に置き換えられます。

template の使い方

```
template<typename 新しく定義する型の名前>
```

第6章　メモリ管理

6.1　newとdeleteの基本的な使い方

C 言語で動的なメモリを確保する場合、malloc を使います。メモリ開放は free です。

しかし、C++言語では new と delete を使います。C 言語で次のようなコードを書いたとします。

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   int main()
4   {
5           int* p = (int*)malloc( sizeof(int) );
6           if( !p ){
7                   printf("%s\n","memory allocation error.");
8                   return 1;
9           }
10          *p = 10;
11          printf("value : %d",*p);
12          free(p);
13          return 0;
14  }
```

これを new/delete を使って書くと次のようなコードとなります。

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5           int* p = new int;
6           if( !p ) {
```

```
7                   cout << "memory allocation error." << endl;
8                   return 1;
9           }
10          *p = 10;
11          cout << "value : " << *p << endl;
12          delete p;
13          return 0;
14  }
```

new は確保したメモリのポインタを返します。

new の使い方

```
new 型;
```

確保したメモリは delete で削除することができます。

delete の使い方

```
delete 削除する変数;
```

malloc で確保したものを delete で削除や、new で確保したメモリを free で開放等はできません。詳しくは説明しませんが、malloc と new ではメモリを確保する場所が違います。malloc ではヒープ (heap) という場所に確保され、new ではフリーストア (Free Store) という場所に確保されます。

6.2 配列のメモリ割り当て

配列のメモリを確保する場合次のようなコードを書きます。

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6           int* p = new int[3];
7           if( !p ) {
```

```
 8                   cout << "memory allocation error." << endl;
 9                   return 1;
10           }
11           for(int i=0; i < 3; i++ )
12                   p[i] = i;
13
14           for(int i=0; i < 3; i++ )
15                   cout << "value [" << i << "] :" << p[i] << "\n";
16           delete [] p;
17           return 0;
18   }
```

配列の確保

```
new 型 [要素数];
```

要素数 3 個の int 型の配列を確保する場合、例のように `new int[3]` とします。注意してもらいたいのは、delete の方です。

配列の開放

```
delete [] 削除する変数;
```

配列を確保した場合、delete は `delete [] p;` となります。ここで、`delete p;` としてはいけません。こうするとメモリリーク[1]の原因となります。

[1] new したものを delete しない事

参考文献

[1] ハーバート・シルト著, トップスタジオ訳, 独習 C++改訂版.