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  }
```

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.