C++ Basics
Part One

Sai-Keung Wong

National Yang Ming Chiao Tung University

Hsinchu, Taiwan

Intended Learning Outcomes

- Display the message “Hello World!” on the console.
- Describe the purpose of namespace.
- Implement a program with multiple header files.
- List the essential good programming styles
- List different types of errors.

Display a message

Hello World!

A C++ Program

Display the message “Hello World!” on the console window.

```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

namespace

A **namespace** is a set of signs (*names*) that are used to identify and refer to objects of various kinds.

```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

The functions **cout, endl, and operator <<** are defined in namespace **std.**

The header file **iostream** defines the standard input/output stream objects.

A C++ Program

Display the message “Hello World!” on the console window.

```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

cout
endl, and
operator <<
are defined in namespace **std.**

The functions **cout, endl, and operator <<** are defined in namespace **std.**
The header file **iostream** defines the standard input/output stream objects.

namespace

A **namespace** is a set of signs (*names*) that are used to identify and refer to objects of various kinds.

```
#include <iostream>
using namespace std;
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

To enable a namespace, write

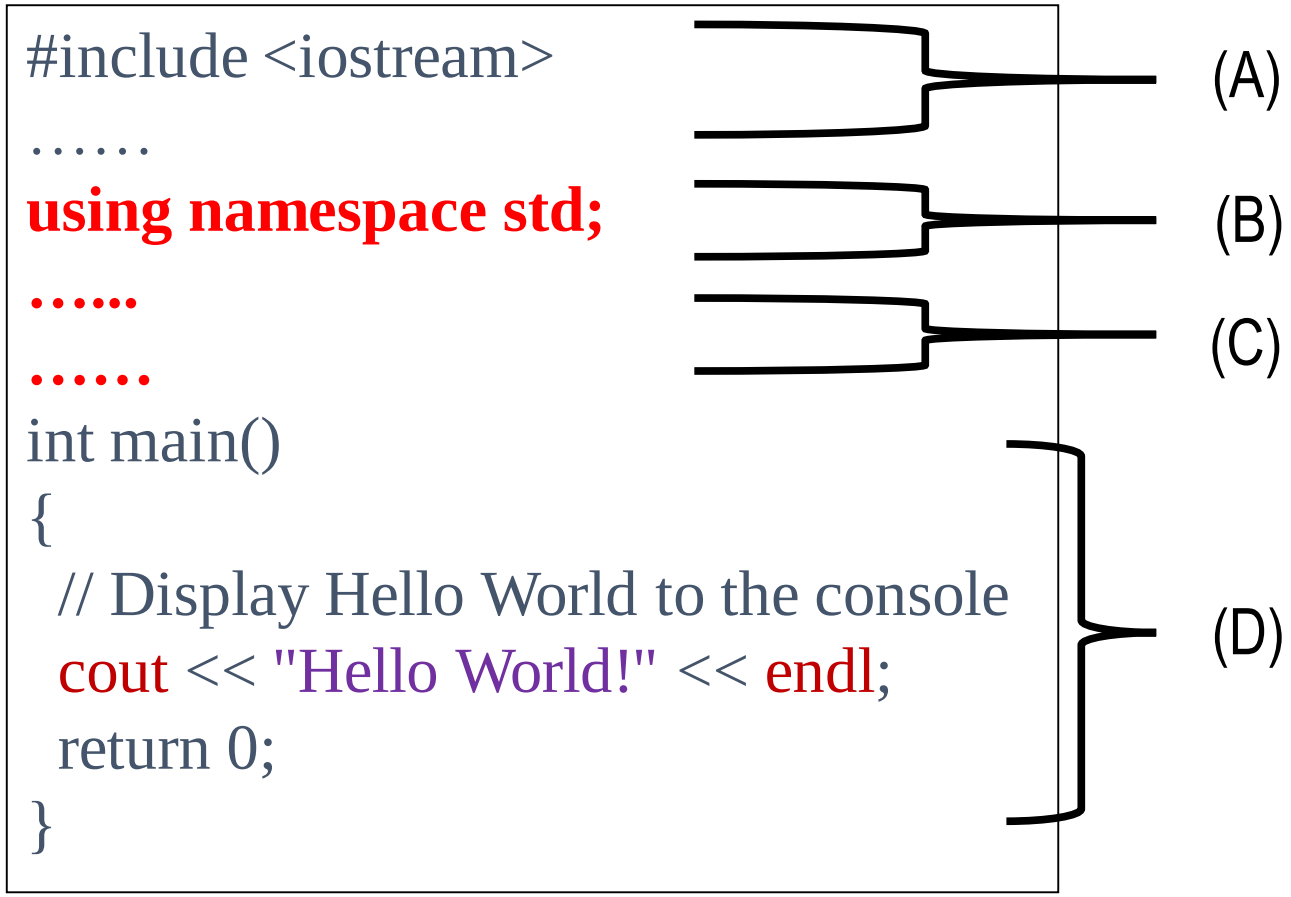
using namespace namespace_name

Note: endl, which is a function, is a manipulator. It terminates a line and flushes the buffer.

cout << “Hello World!\n"; ‘\n’ sends the newline character but does not flush the buffer.

Program structure

A program structure of a program is the overall form of the program.



Program structure

A program structure of a program is the overall form of the program.

```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

(A) Include header files

(B) Enable namespaces

(C) Define global variables, functions

(D) Define the main function.
It is program entry point.

namespace

A **namespace** is a set of signs (*names*) that are used to identify and refer to objects of various kinds.

```
#include <iostream>
#include <my_iostream>

using namespace std;
int main()
{
  // Display Hello World to the console
  cout << “Hello World!" << endl;
  return 0;
}
```

In iostream.h

cout and endl are defined.

In my_iostream.h

cout and endl are also defined.

namespace

A **namespace** is a set of signs (*names*) that are used to identify and refer to objects of various kinds.

```
#include <iostream>
#include <my_iostream>

using namespace std;
int main()
{
  // Display Hello World to the console
  cout << “Hello World!" << endl;
  return 0;
}
```

What should we do if we want to use cout or endl?

In iostream.h

cout and endl are defined.

In my_iostream.h

cout and endl are also defined.

In the file my_iostream.h

```
namespace my_iostream {
   … cout …
   … endl …
};
```

namespace

A **namespace** is a set of signs (*names*) that are used to identify and refer to objects of various kinds.

```
#include <iostream>
#include <my_iostream>

using namespace std;
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

As **std is enabled** but **my_iostream** is **not enbled**, so we use cout and endl in std.

In iostream.h

cout and endl are defined.

In my_iostream.h

cout and endl are also defined.

In the file my_iostream.h

```
namespace my_iostream {
   … cout …
   … endl …
};
```

What should we do if we want to use cout or endl?

namespace

An example

A namespace is **a** A1 **region** that provides a A2 to the identifiers (e.g., functions, variables, data type, and etc) inside it.

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

The namespace std is not enabled.

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

The namespace std is not enabled.

We include the header file.
But the namespace is not enabled.
To access a function,
write **namespace::func_name**

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

The namespace std is not enabled.

We include the header file.
But the namespace is not enabled.
To access a function,
write **namespace::func_name**

In A.h, we have the following:

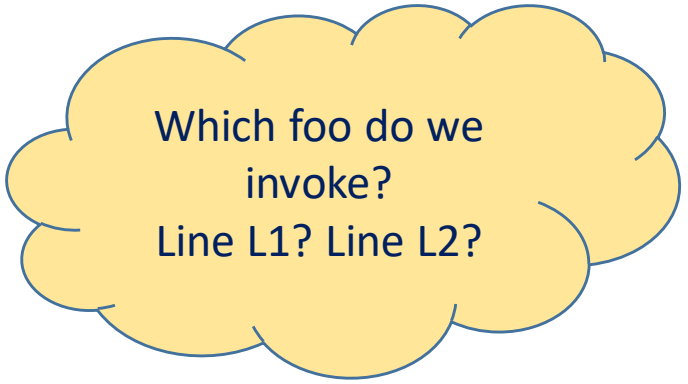
```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```



In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

Which foo do we invoke?
Line L1? Line L2?

L1: foo is in namespace A1
L2: foo is in namespace A2

In A.h, we have the following:

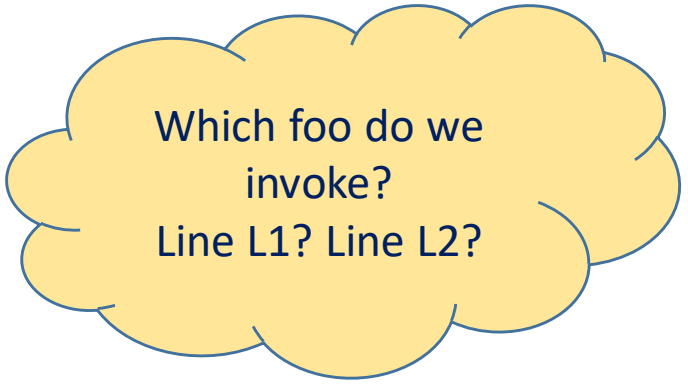
```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```



L1: foo is in namespace Peter.
L2: foo is in namespace Mary.

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it. **Remark: Avoid name conflict.**

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

Which foo do we invoke?
Line L1? Line L2?

L1: foo is in namespace Peter.
L2: foo is in namespace Mary.

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is **a declarative region** that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl; // L1
   std::cout << foo(5) << std::endl; // L2

   return 0;
}
```

Which foo do we invoke?
Line L1? Line L2?

L1: foo is in namespace Peter.
L2: foo is in namespace Mary.

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;    // L1: error?
   std::cout << b << endl;               // L2: error?
   return 0;
}
```

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;    // L1:
   std::cout << b << endl;               // L2:
   return 0;
}
```

A1

A2

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

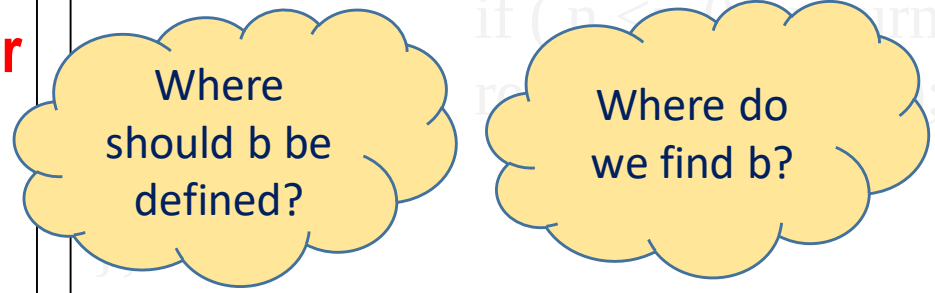
```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;    // L1: No error
   std::cout << b << endl;               // L2: error
   return 0;
}
```

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n*foo(n-1);
        }
};
```



A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include "A.h"
#include "B.h"
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;    // L1: No error
   std::cout << b << endl;               // L2: error
   return 0;
}
```

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n*foo(n-1);
        }
};
```

Try to find it in the global space, and the enabled namespaces.

Where should b be defined?

Where do we find b?

A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;    // L1: No error
   std::cout << b << endl;               // L2: error
   return 0;
}
```

In A.h, we have the following:

```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        j
                                    0;
};
```

How to fix the error?

A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;//no error
   std::cout << b << endl;
   return 0;
}
```

In A.h, we have the following:

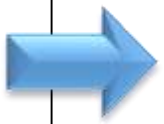
```
namespace Peter {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int b = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

A namespace is a declarative region that provides a scope to the identifiers (e.g., functions, variables, data type, and etc) inside it.

```
#include “A.h”
#include “B.h”
#include <iostream>

using namespace Mary;
int main() {
   std::cout << Peter::foo(5) << std::endl;
   std::cout << foo(5) << std::endl;
   std::cout << Mary::a << std::endl;//no error
   std::cout << b << endl;
   std::cout << Peter::b << std::endl;
   return 0;
}
```



In A.h, we have the following:

```
namespace Peter {
        int b = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

In B.h, we have the following:

```
namespace Mary {
        int a = 0;
        int foo( int n ) {
                if ( n <= 0 ) return 0;
                return n+foo(n-1);
        }
};
```

Multiple declarations

How to avoid multiple declarations?

Avoid multiple declarations

```
#include “A.h”
#include “A.h”
#include <iostream>

using namespace ns_A, std;
int main() {
        cout << foo(5) << std::endl;

        return 0;
}
```

When we build the program, what occurs?

In A.h, we have the following:

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

Avoid multiple declarations

```
#include “A.h”




#include “A.h”
#include <iostream>

using namespace ns_A, std;
int main() {
        cout << foo(5) << std::endl;

        return 0;
}
```

In A.h, we have the following:

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

Avoid multiple declarations

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

#include “A.h”

```
#include “A.h”
#include <iostream>

using namespace ns_A, std;
int main() {
        cout << foo(5) << std::endl;

        return 0;
}
```

In A.h, we have the following:

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

Avoid multiple declarations

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

#include “A.h”

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

#include “A.h”

a and foo are defined in multiple times.

```
#include <iostream>
using namespace ns_A, std;
int main() {
  cout << foo(5) << std::endl;
  return 0;
}
```

In A.h, we have the following:

```
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};
```

Avoid multiple declarations

```
#include “A.h”
#include “A.h”

#include <iostream>

using namespace ns_A, std;

int main() {
   cout << foo(5) << std::endl;
   return 0;
}
```

When we build the program, what occurs?

In A.h, we have the following:

```
#ifndef  __A_file__          // directive
#define __A_file__
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};

#endif
```

Avoid multiple declarations

```
#include “A.h”



#include “A.h”



#include <iostream>

using namespace ns_A, std;
```

In A.h, we have the following:

```
#ifndef  __A_file__          // directive
#define __A_file__
namespace ns_A {
        int a = 1;
        int foo( int n ) {
                if ( n <= 0 ) return 1;
                return n*foo(n-1);
        }
};

#endif
```

Avoid multiple declarations

```
#ifndef     __A_file__   // directive
#define     __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};
#endif
```

```
#include “A.h”
```

```
#include <iostream>
```

In A.h, we have the following:

```
#ifndef  __A_file__          // directive
#define __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};

#endif
```

Avoid multiple declarations

```
#ifndef     __A_file__   // directive
#define     __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};
#endif
```

```
#ifndef     __A_file__   // directive
#define     __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};
#endif
```

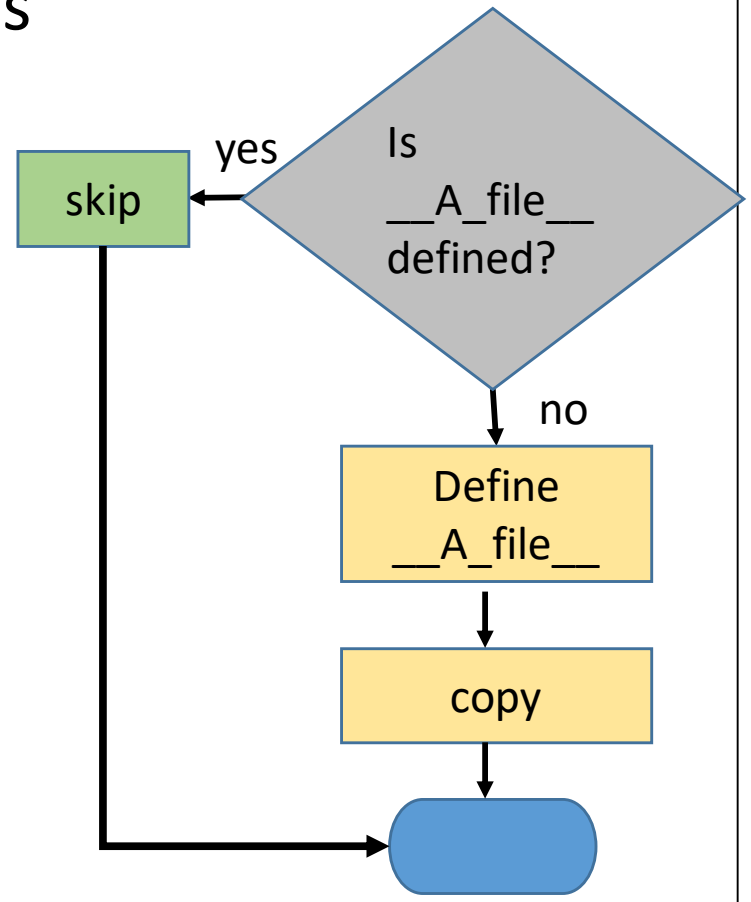
In A.h, we have the following:

```
#ifndef  __A_file__          // directive
#define __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};

#endif
```

Avoid multiple declarations

```
#ifndef     __A_file__   // directive
#define     __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};
#endif
```

```
#ifndef     __A_file__   // directive
#define     __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};
#endif
```



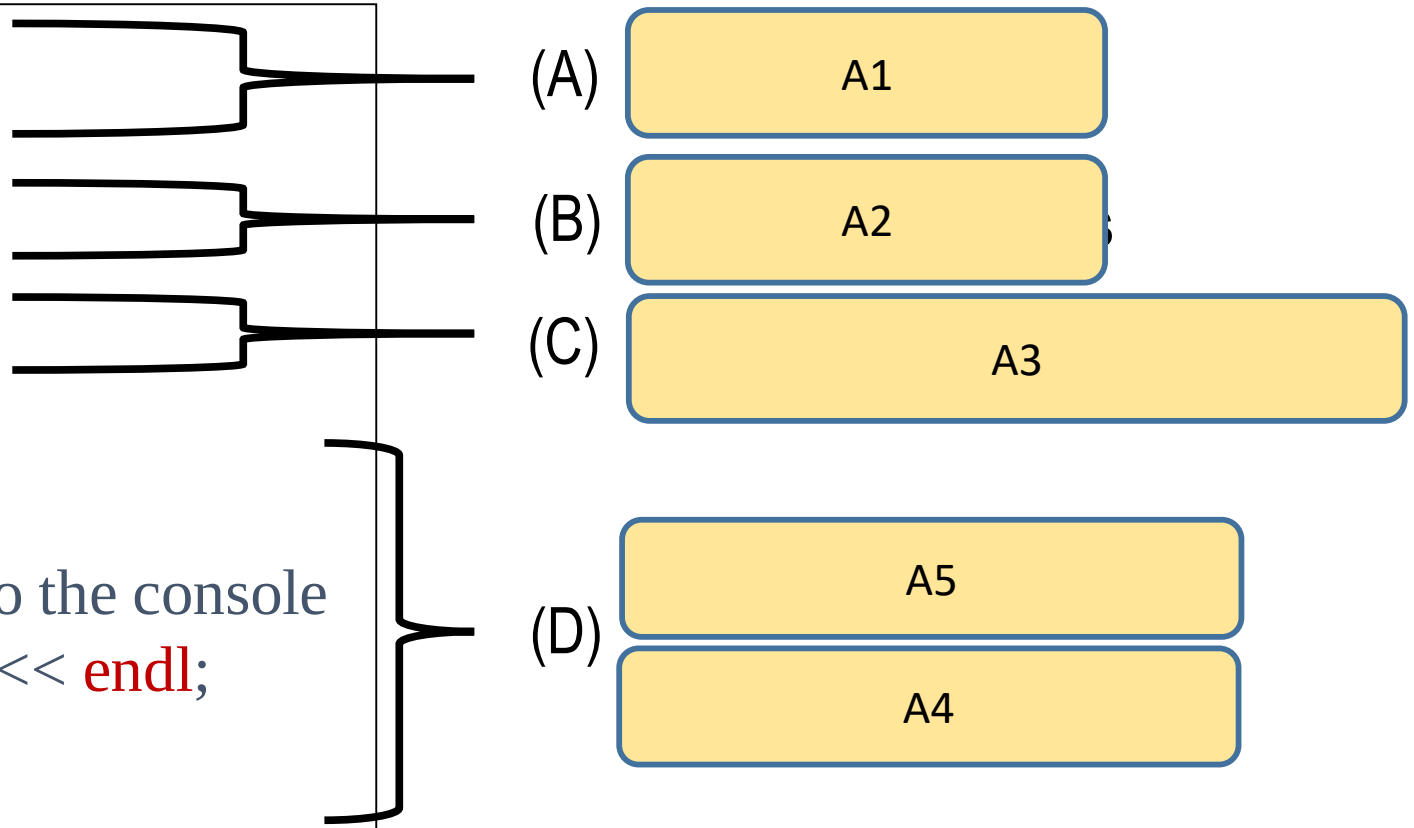
In A.h, we have the following:

```
#ifndef  __A_file__          // directive
#define __A_file__
namespace ns_A {
            int a = 1;
            int foo( int n ) {
                        if ( n <= 0 ) return 1;
                        return n*foo(n-1);
            }
};

#endif
```

Program structure

A program structure of a program is the overall form of the program.

```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```

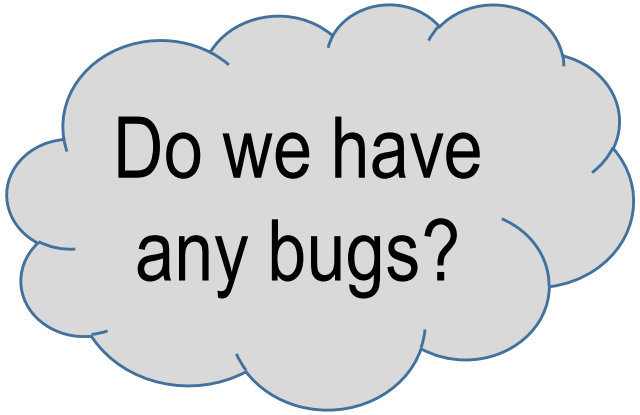


Exercise: Extending the C++ Program

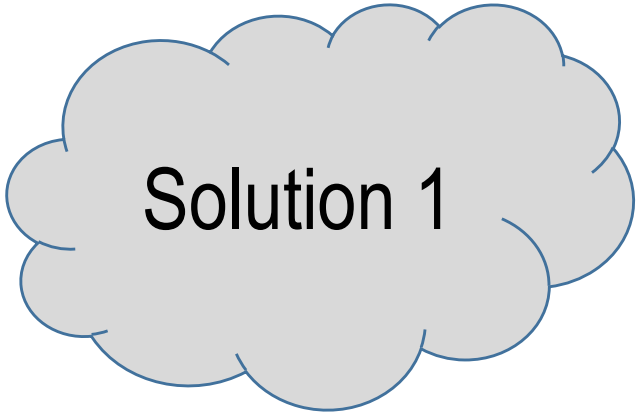
```
#include <iostream>

int main()
{
  cout << "Programming is fun!" << endl;
  cout << "I like programming!" << endl;
  cout << "I want to learn more…" << endl;
  return 0; // normal exit
}
```

Exercise: Extending the C++ Program

```
#include <iostream>

int main()
{
   cout << "Programming is fun!" << endl;
   cout << "I like programming!" << endl;
   cout << "I want to learn more…" << endl;
   return 0; // normal exit
}
```

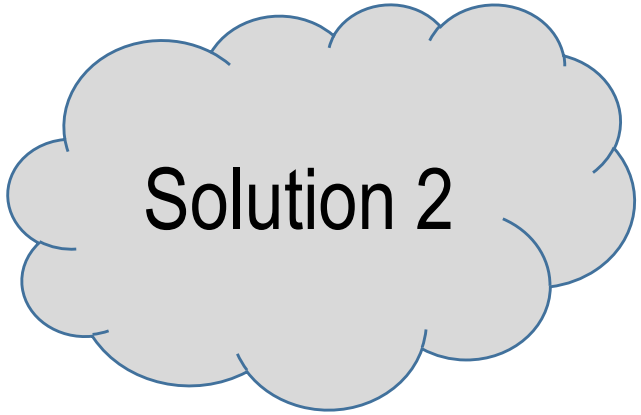


Exercise: Extending the C++ Program



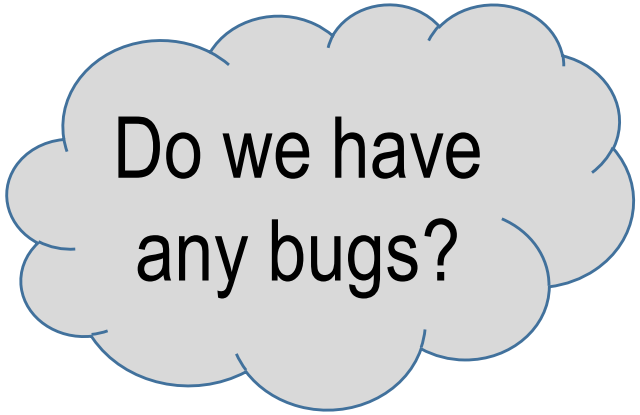

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Programming is fun!" << endl;
   cout << "I like programming!" << endl;
   cout << "I want to learn more…" << endl;
   return 0; // normal exit
}
```

Exercise: Extending the C++ Program



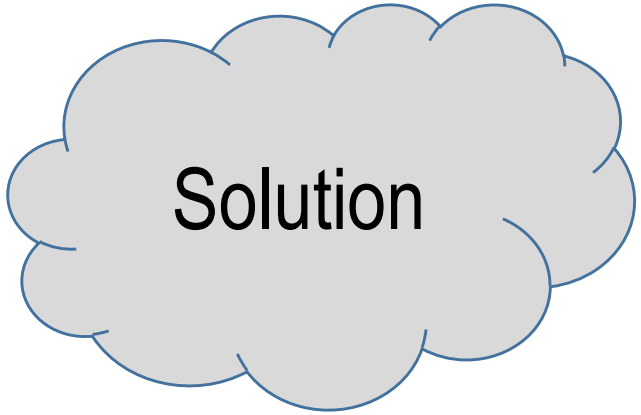

```
#include <iostream>

int main()
{
   std::cout << "Programming is fun!" << std:: endl;
   std::cout << "I like programming!" << std:: endl;
   std::cout << "I want to learn more…" << std:: endl;
   return 0; // normal exit
}
```

Exercise: Extending the C++ Program



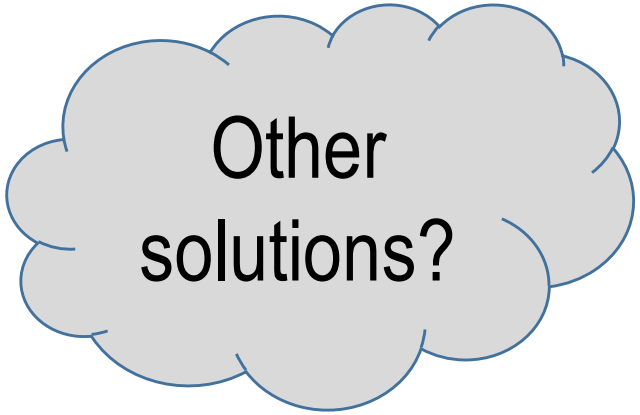

```
using namespace std;
int main()
{
   cout << "Programming is fun!" << endl;
   cout << "I like programming!" << endl;
   cout << "I want to learn more…" << endl;
   return 0; // normal exit
}
```

Exercise: Extending the C++ Program




```
#include <iostream>
using namespace std;
int main()
{
   cout << "Programming is fun!" << endl;
   cout << "I like programming!" << endl;
   cout << "I want to learn more…" << endl;
   return 0; // normal exit
}
```

Exercise: Extending the C++ Program




```
#include <iostream>
using namespace std;
int main()
{
   cout << "Programming is fun!" << endl;
   cout << "I like programming!" << endl;
   cout << "I want to learn more…" << endl;
   return 0; // normal exit
}
```

Example

We can perform mathematical computations and displays the result to the console.

```
#include <iostream>
using namespace std;
int main()
{
  cout << (1 + 2 + 3) / 3 << endl;
  return 0;
}
```

Example

We can perform mathematical computations

and displays the result to the console.

```
#include <iostream>
using namespace std;
int main()
{
  cout << (1 + 2 + 3) / 3 << endl;
  return 0;
}
```

```
#include <iostream>

int main()
{
 A1 cout << (1 + 2 + 3) / 3 << A2 endl;
  return 0;
}
```

Example: Display records of students

- Display the records of students

Example: Display records of students

- Display the records of students

```
#include <iostream>
using namespace std;
class STUDENT {
   protected:                          // access modifier
      int score;                       // data field
   public:                             // access modifier
     STUDENT( ) { score = 0; }         // no-arg constructor
     void printf( ) {                  // member function
         cout << score << endl;
     }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Example: Display records of students

- Display the records of students

```
#include <iostream>
using namespace std;
class STUDENT {
  protected:                          // access modifier


  int score;                          // data field

  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) {                  // member function


       cout << score << endl;
    }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Example: Display records of students

- Display the records of students

```
#include <iostream>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field

  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) {                  // member function
       cout << “name:” << name << endl;
       cout << “ID:” << ID << endl;
       cout << score << endl;
    }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Example: Display records of students

- Display the records of students

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field

  public:                             // access modifier
   STUDENT( ) { score = 0; }          // no-arg constructor
   void printf( ) {                   // member function
      cout << “name:” << name << endl;
      cout << “ID:” << ID << endl;
      cout << score << endl;
   }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Example: Display records of students

- Display the records of students

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                        // access modifier
  string name;                      // C++ string
  string ID;                        // student ID
  int score;                        // data field

  public:                           // access modifier
   STUDENT( ) { score = 0; }        // no-arg constructor
   void printf( ) {                 // member function
      cout << “name:” << name << endl;
      cout << “ID:” << ID << endl;
      cout << score << endl;
   }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Example: Display records of students

- Display the records of students

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                        // access modifier
  string name;                      // C++ string
  string ID;                        // student ID
  int score;                        // data field

  protected:                        // access modifier
   STUDENT( ) { score = 0; }        // no-arg constructor
   void printf( ) {                 // member function
      cout << "name:" << name << endl;
      cout << "ID:" << ID << endl;
      cout << score << endl;
   }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Modification:
public -> protected or private

Do we have an error?

Example: Display records of students

- Display the records of students

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                    // access modifier
  string name;                  // C++ string
  string ID;                    // student ID
  int score;                    // data field

  protected:                    // access modifier
   STUDENT( ) { score = 0; }    // no-arg constructor
   void printf( ) {             // member function
      cout << "name:" << name << endl;
      cout << "ID:" << ID << endl;
      cout << score << endl;
   }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main( ) {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

error

A client cannot use protected data members, methods of a class object, and constructors.

Example: Display records of students

- Display the records of students

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                    // access modifier
  string name;                  // C++ string
  string ID;                    // student ID
  int score;                    // data field

  protected:                    // access modifier
   STUDENT( ) { score = 0; }    // no-arg constructor
   void printf( ) {             // member function
      cout << "name:" << name << endl;
      cout << "ID:" << ID << endl;
      cout << score << endl;
   }
};
```

error

error

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects
int main( ) {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

A client cannot use protected data members, methods of a class object, and constructors.

In this case, the constructor is

A1

The member function printf( ) is

A2

Intended Learning Outcomes

- Display the message “Hello World!” on the console.
- Describe the purpose of namespace.
- Implement a program with multiple header files.
- List the essential good programming styles
- List different types of errors.

Organization of a program

Files

Organization of a program – Approach One

- Store each class one file(s)

In student.h

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const {            // member function
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
    }
};
```

```
const int NUM = 80;
STUDENT s[NUM]; // 80 objects

int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};
```

In student.cpp

```
#include "student.h"
STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}
```

```
#include "student.h"
const int NUM = 80;
STUDENT s[NUM]; // NUM objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};
```

In student.cpp

```
#include "student.h"
STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}
```

```
#include "student.h"
const int NUM = 80;
STUDENT s[NUM]; // NUM objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#ifndef __STUDENT_H__
#define __STUDENT_H__
#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};
#endif
```

In student.cpp

```
#include "student.h"
STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}
```

```
#include "student.h"
const int NUM = 80;
STUDENT s[NUM]; // NUM objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#ifndef __STUDENT_H__
#define __STUDENT_H__

#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};

#endif
```

In student.cpp

```
#include "student.h"
STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}
```

```
#include "student.h"
const int NUM = 80;
STUDENT s[NUM]; // NUM objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#ifndef __STUDENT_H__
#define __STUDENT_H__

#include <iostream>
#include <string>
using namespace std;
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};

#endif
```

In student.cpp

```
#include "student.h"
STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}
```

Any potential problems?

Name conflict problem.

```
#include "student.h"
const int NUM = 80;
STUDENT s[NUM]; // NUM objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#ifndef __STUDENT_H__
#define __STUDENT_H__

#include <iostream>
#include <string>
using namespace std;
```

A1

```
class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};
```

A2

```
#endif
```

Use namespace to resolve name conflict problem.

In student.cpp

```
#include "student.h"
STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}
```

Any potential problems?

Name conflict problem.

```
#include "student.h"
const int NUM = 80;
STUDENT s[NUM]; // NUM objects
int main() {
  ……
  for ( int i = 0; i < NUM; ++i ) s[ i ].printf( );
  return 0;
}
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

In student.h

```
#ifndef __STUDENT_H__
#define __STUDENT_H__

#include <iostream>
#include <string>
using namespace std;
namespace my_ns {

class STUDENT {
  protected:                          // access modifier
  string name;                        // C++ string
  string ID;                          // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};

};
#endif
```

Use namespace to resolve name conflict problem.

In student.cpp

```
#include "student.h"

A1

STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        cout << "name:" << name << endl;
        cout << "ID:" << ID << endl;
        cout << score << endl;
}

A2
```

Organization of a program – Approach Two

- Store each class in two (or more) files: .h file and .cpp file

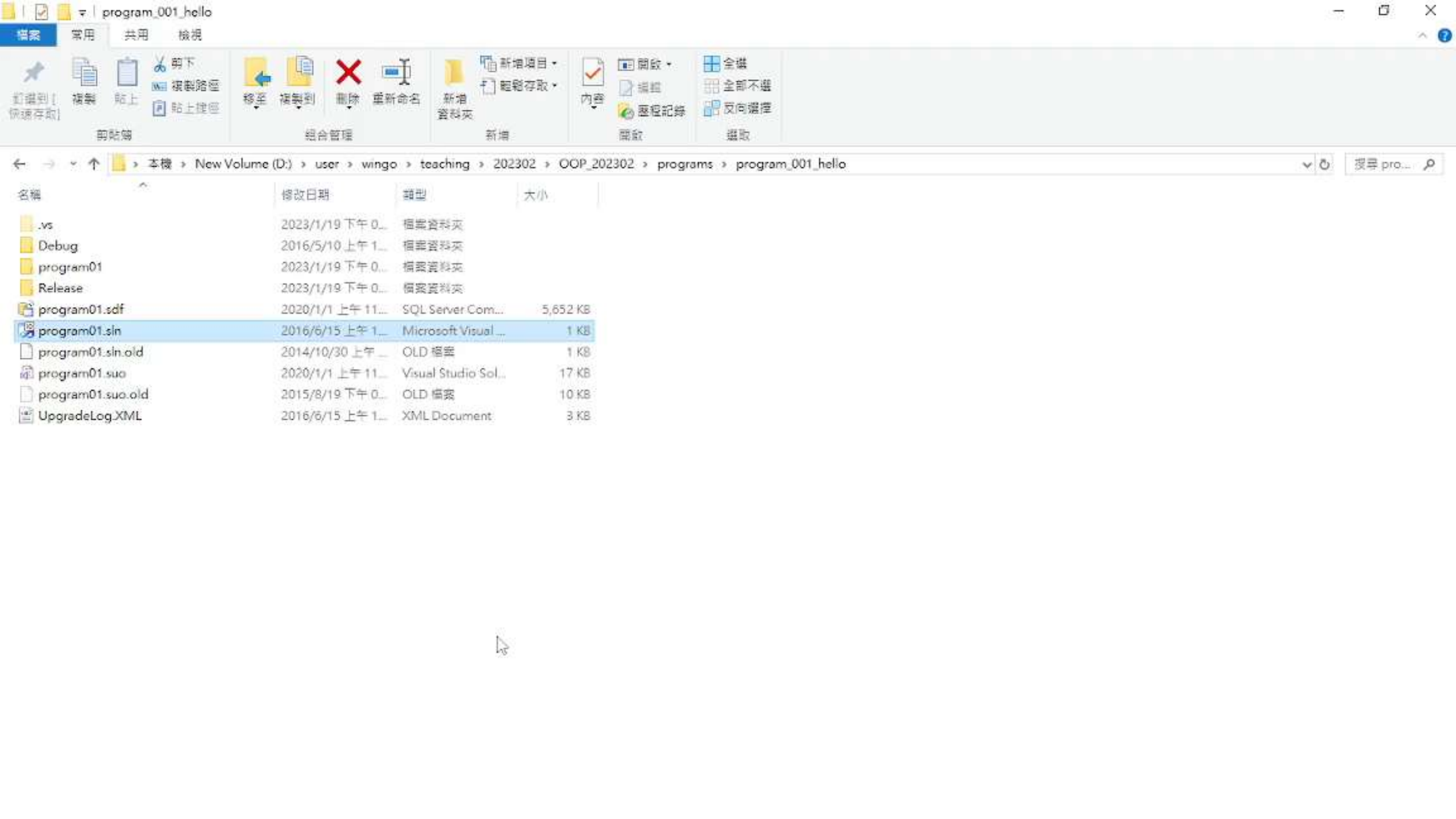
In student.h

```
#ifndef __STUDENT_H__
#define __STUDENT_H__

#include <iostream>
#include <string>
using namespace std;
namespace my_ns {

class STUDENT {
  protected:                          // access modifier
  std::string name;                   // C++ string
  std::string ID;                                  // student ID
  int score;                          // data field
  public:                             // access modifier
    STUDENT( ) { score = 0; }         // no-arg constructor
    void printf( ) const;             // member function/method
};

};
#endif
```

In student.cpp

```
#include "student.h"

namespace my_ns {

STUDENT::STUDENT( ) {
        score = 0;
}
void STUDENT:: printf( ) {
        std::cout << "name:" << name << std::endl;
        std::cout << "ID:" << ID << std::endl;
        std::cout << score << std::endl;
}

};
```

If the namespace is not enabled,
to access a function/data field,
write A1 A2


program_001_hello
檔案
常用
共用
檢視
剪下
複製路徑
貼上捷徑
複製
貼上
移至
複製到
刪除
重新命名
新增資料夾
新增項目
輕鬆存取
內容
開啟
編輯
歷程記錄
全選
全部不選
反向選擇
剪貼簿
組合管理
新增
開啟
選取
本機 › New Volume (D:) › user › wingo › teaching › 202302 › OOP_202302 › programs › program_001_hello
名稱
修改日期
類型
大小
.vs 2023/1/19 下午 0... 檔案資料夾
Debug 2016/5/10 上午 1... 檔案資料夾
program01 2023/1/19 下午 0... 檔案資料夾
Release 2023/1/19 下午 0... 檔案資料夾
program01.sdf 2020/1/1 上午 11... SQL Server Com... 5,652 KB
program01.sln 2016/6/15 上午 1... Microsoft Visual ... 1 KB
program01.sln.old 2014/10/30 上午 ... OLD 檔案 1 KB
program01.suo 2020/1/1 上午 11... Visual Studio Sol... 17 KB
program01.suo.old 2015/8/19 下午 0... OLD 檔案 10 KB
UpgradeLog.XML 2016/6/15 上午 1... XML Document 3 KB

Input argument to a program

Describe the process

Input argument to a program

argc: number of arguments.

argv: an array of C-string pointers.

```
#include <iostream>
using namespace std;
//
// argc: number of arguments
// argv[]: store the arguments
int main(int argc, char **argv)
{
  // Display Hello World to the console
  cout << “Hello World!" << endl;
  return 0;
}
```

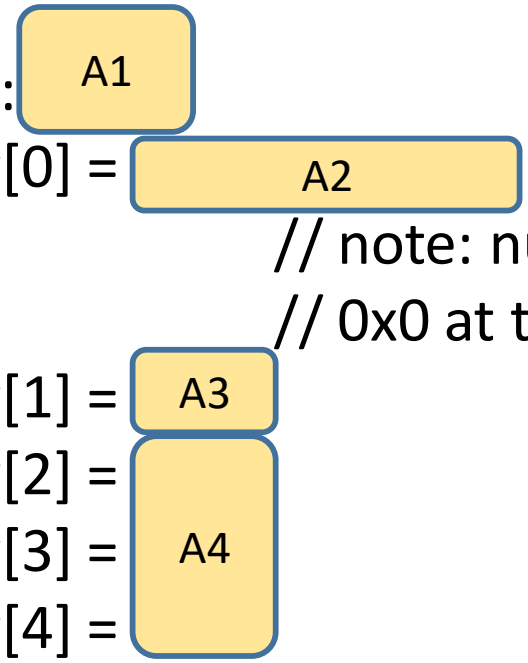
Input argument to a program

argc: number of arguments.

argv: an array of C-string pointers.

```
#include <iostream>
using namespace std;
//
// argc: number of arguments
// argv[]: store the arguments
int main(int argc, char **argv)
{
  // Display Hello World to the console
  cout << “Hello World!" << endl;
  return 0;
}
```

```
my_program c1 c2 ar1 ar2

argc:
argv[0] =                    // string
                  // note: null-terminated string
                  // 0x0 at the end of the string
argv[1] =                    // string
argv[2] =                    // string
argv[3] =                    // string
argv[4] =                    // string
```



Input argument to a program

argc: number of arguments.

argv: an array of C-string pointers.

```
#include <iostream>
using namespace std;
//
// argc: number of arguments
// argv[]: store the arguments
int main(int argc, char **argv)
{
  // Display Hello World to the console
  cout << “Hello World!" << endl;
  return 0;
}
```

```
my_program c1 c2 ar1 ar2

argc: 5
argv[0] = “my_program”        // string
                  // note: null-terminated string
                  // 0x0 at the end of the string
argv[1] = “c1”                // string
argv[2] = “c2”                // string
argv[3] = “ar1”               // string
argv[4] = “ar2”               // string
```

Programming Style and Documentation

A good habit

Programming Style and Documentation

- Appropriate Comments; Proper Indentation and Spacing Lines; Block Styles

```
.
#include <iostream>
using namespace std;
int main( )
{

cout <<                                  "Programming is
fun!"          <<                              endl;
  cout << “I like programming!"          << endl;
  cout << “I want to learn more…" <<          endl;
  return 0; // return the function
}
```

Programming Style and Documentation

- Appropriate Comments; Proper Indentation and Spacing Lines; Block Styles

```
.
#include <iostream>
using namespace std;
int main( )
{

cout <<                                  "Programming is
fun!"          <<                              endl;
  cout << “I like programming!"          << endl;
  cout << “I want to learn more…" <<          endl;
  return 0; // return the function
}
```

poor

```
// A simple example to show messages.
#include <iostream>
using namespace std;
int main( )
{
 // The followings are the messages.
  cout << "Programming is fun!"            <<  endl;
  cout << "I like programming!"            << endl;
  cout << "I want to learn more…“          << endl;
  return 0; // return the function
}
```

Not very good

Programming Style and Documentation

- Appropriate Comments; Proper Indentation and Spacing Lines; Block Styles

```
// A simple example to show messages.
#include <iostream>

using namespace std;

int main( )
{
 // The followings are the messages.
  cout << "Programming is fun!"          <<  endl;
  cout << "I like programming!"          << endl;
  cout << "I want to learn more…“        << endl;

  return 0; // return the function
}
```

Good style

```
// A simple example to show messages.
#include <iostream>
using namespace std;
int main( )
{
 // The followings are the messages.
  cout << "Programming is fun!"          <<  endl;
  cout << "I like programming!"          << endl;
  cout << "I want to learn more…“        << endl;
  return 0; // return the function
}
```

Not very good

Programming Style and Documentation

- Appropriate Comments; Proper Indentation and Spacing Lines; Block Styles

```
// A simple example to show messages.
#include <iostream>

using namespace std;

int main( )
{
 // The followings are the messages.
  cout << "Programming is fun!"          <<  endl;
  cout << "I like programming!"          << endl;
  cout << "I want to learn more…“        << endl;

  return 0; // return the function
}
```

Good style

```
// A simple example to show messages.
#include <iostream>
using namespace std;
int main( )
{
 // The followings are the messages.
  cout << "Programming is fun!"          <<  endl;
  cout << "I like programming!"          << endl;
  cout << "I want to learn more…“        << endl;
  return 0; // return the function
}
```

Not very good

Programming Style and Documentation

- Long enough names for data members and methods.

```
#include <iostream>
using namespace std;

COURSE c;
int main( )
{
 c.Iff( );
 c.id( );
 c.c( )
 c.d( );
 c.e( );
 c.sf( );

  return 0; // return the function
}
```

Does not make any sense

```
#include <iostream>
using namespace std;

COURSE course;
int main( )
{
   course.loadFromFile( );
   course.initialize_data( );
   course.askForInput( );
   course.process( );
   course.showResult( );
   course.saveToFile( );

   return 0; // return the function
}
```

Make sense. Can tell a complete story. Very good.

Programming Errors

Avoid them …

Programming Errors

Programming Errors

- **Syntax Error** A1 occurs when there is an A2 line of code.

Programming Errors

- **Syntax Error**: An error occurs when there is an incorrect line of code.
- **Runtime Error**: An error occurs while the program is A1 after being successfully A2.

Programming Errors

- **Syntax Error**: An error occurs when there is an incorrect line of code.
- **Runtime Error**: An error occurs while the program is running after being successfully compiled.
- **Logic Error**: It is a bug that causes the program to operate A1, but not to terminate A2

Programming Errors

- **Syntax Error**: An error occurs when there is an incorrect line of code.
- **Runtime Error**: An error occurs while the program is running after being successfully compiled.
- **Logic Error**: It is a bug that causes the program to operate incorrectly, but not to terminate abnormally (or crash)

Syntax Errors

Syntax Error: An error occurs when there is an incorrect line of code. This kind of errors are detected during compilation.

```
#include <iostream>
using namespace std;
int main(
{
        int i = 4;
        int j = 1
        cout i / j << endl;
        return 0;
}
```

Syntax Errors

Syntax Error: An error occurs when there is an incorrect line of code. This kind of errors are detected during compilation.

```
#include <iostream>
using namespace std;
int main( )
{
        int i = 4;
        int j = 1;
        cout << i / j << endl;
        return 0;
}
```

Syntax Errors

```
#include <iostream>
using namespace std;
int main(
{
	int i = 4;
	int j =
1;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
	cout << i / j << endl;
	return 0;
}
```

Syntax Errors

```
#include <iostream>
using namespace std;
int main( )
{
        int i = 4;
        int j =
1;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        cout << i / j << endl;
        return 0;
}
```

Syntax Errors

```
#include <iostream>
using namespace std;
int main( )
{
	int i = 4;
	int j = 1;;;  ;;  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;  ;;;;;;;;;;;;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
	cout << i / j << endl;
	return 0;
}
```

Runtime Errors

Runtime Error: An error occurs while the program is running after being successfully compiled.

```
#include <iostream>
using namespace std;
int main()
{
	int i = 4;
	int j = 0;
	cout << i / j << endl;
	return 0;
}
```

Runtime Errors

Runtime Error: An error occurs while the program is running after being successfully compiled.

```
#include <iostream>
using namespace std;
int main()
{
        int i = 4;
        int j = 0;
        cout << i / j << endl;      // divided by zero
        return 0;
}
```

Logic Errors

Logic Error: It is a bug that causes the program to operate incorrectly, but not to terminate abnormally (or crash)

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Celsius 35 is Fahrenheit degree " << endl;
        cout << (9 / 5) * 35 + 32 << endl;
        return 0;
}
```

Logic Errors

Logic Error: It is a bug that causes the program to operate incorrectly, but not to terminate abnormally (or crash)

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Celsius 35 is Fahrenheit degree " << endl;
        cout << (9 / 5) * 35 + 32 << endl;  // A1 division
        return 0;
}
```

Logic Errors

Logic Error: It is a bug that causes the program to operate incorrectly, but not to terminate abnormally (or crash)

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Celsius 35 is Fahrenheit degree " << endl;
        cout << (9 / (float) 5) * 35 + 32 << endl;     //
        return 0;
}
```

A1 division

Logic Errors

Logic Error: It is a bug that causes the program to operate incorrectly, but not to terminate abnormally (or crash)

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Celsius 35 is Fahrenheit degree " << endl;
        cout << (9 / 5.0) * 35 + 32 << endl;          // floating point division
        return 0;
}
```

Common Errors

- Using reserved words as variable names

  ```
  false = 1;
  ```

- Missing Braces

  ```
  { int good = true; }
  ```

- Missing Semicolons

  ```
  int a = 0;
  ```

- Missing Quotation Marks

  ```
  “Programming is good … “
  ```

- Misspelling Names

  ```
  int l = 2111;
  1 = 3222;
  ```

Common Errors

- Missing Braces

  { int hi = true; }

- Missing Semicolons

  int b = 0;

- Missing Quotation Marks

  “Programming is very good…”

- Misspelling Names

  int l = 20;

  1 = 32;

Common Errors

- Missing Braces

  { int hi = true; }
- Missing Semicolons

  int b = 0;
- Missing Quotation Marks

  “Programming is very good…”
- Misspelling Names

  int l = 20;

  1 = 32;

```
int i, l, l, L =5;
l = 3; l = 4;
l = l + l * 2 + 1;
cout << “l:” << l << endl;
```

Common Errors

- Missing Braces

  { int hi = true; }

- Missing Semicolons

  int b = 0;

- Missing Quotation Marks

  “Programming is very good…”

- Misspelling Names

  int l = 20;

  1 = 32;

```
int i, l, l, L =5;
l = 3; l = 4;
l = l + l * 2 + 1;
cout << “l:” << l << endl;
```

```
int iL, ll =5, ii, ll = 6;
cout << “r:” << ll-ll << endl;
```

Missing header files

```
using namespace std;
void main ( )
{
        STUDENT records[10];
        cout << “Hello!” << std::endl;
}
```

Missing header files

```
using namespace std;
void main ( )
{
        STUDENT records[10];            // need A1.h
        cout << “Hello!” << std::endl;  // need A2
}
```

Missing header files

```
using namespace std;
void main ( )
{
        STUDENT records[10];              // need student.h
        cout << “Hello!” << std::endl;    // need iostream
}
```

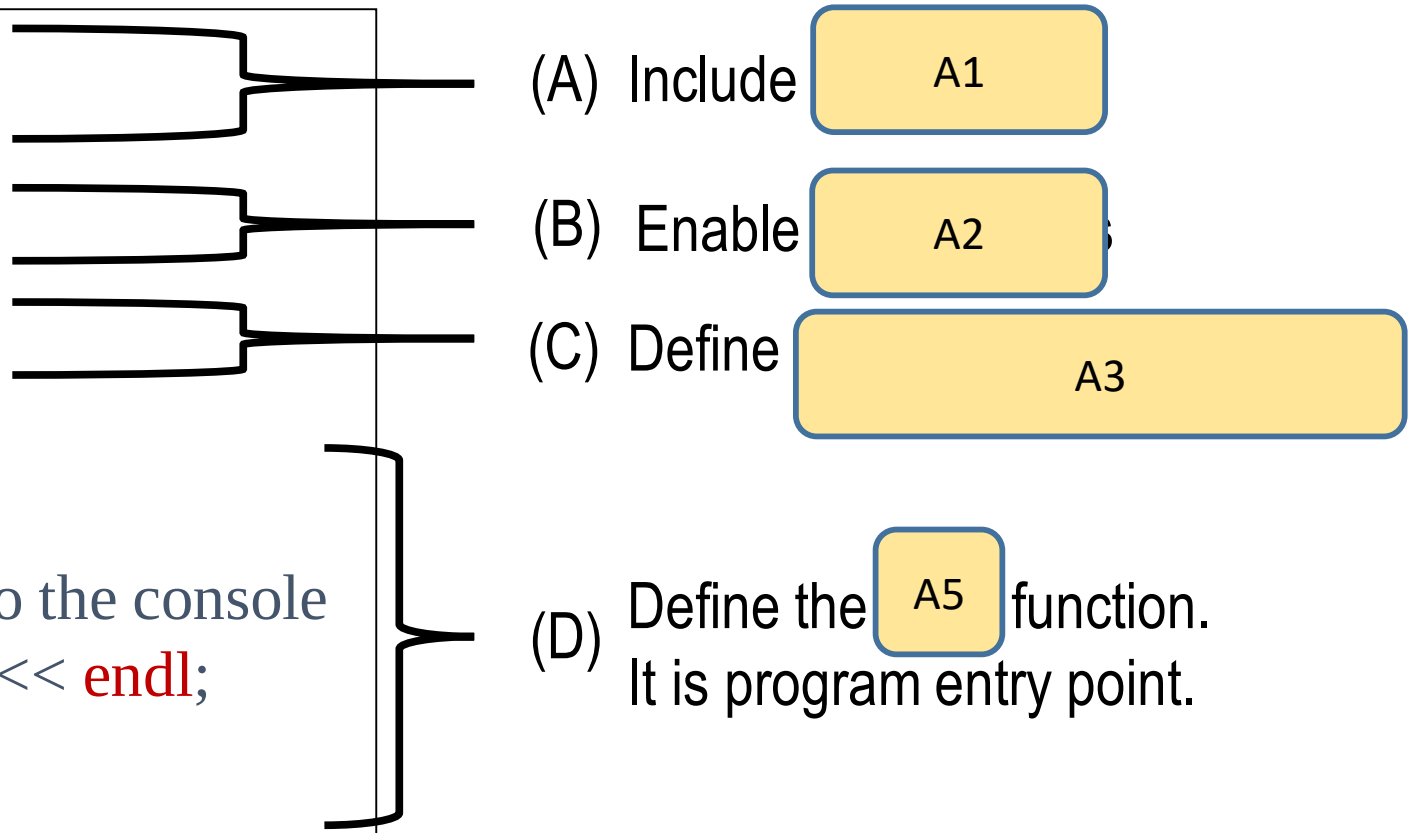
Intended Learning Outcomes

- Display the message “Hello World!” on the console.
- Describe the purpose of namespace.
- Implement a program with multiple header files.
- List the essential good programming styles
- List different types of errors.

Exercise: Program structure

A program structure of a program is the overall form of the program.

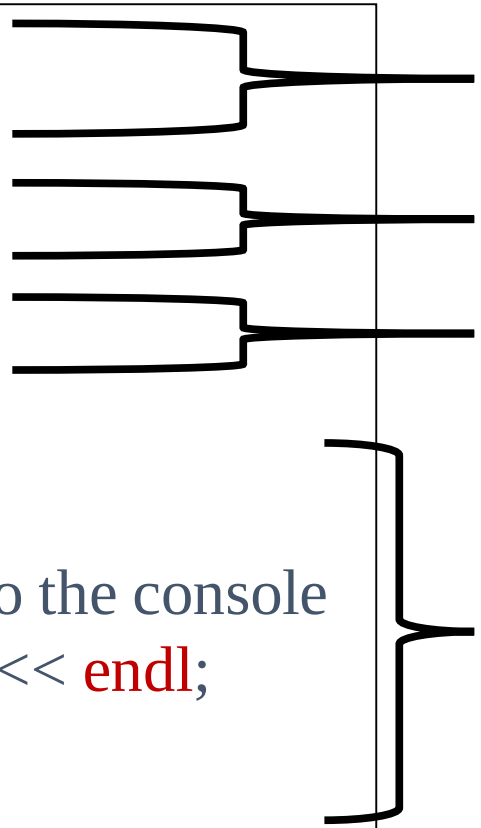
```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```



Exercise: Program structure

A program structure of a program is the overall form of the program.

```
#include <iostream>
……
using namespace std;
……..
……
int main()
{
  // Display Hello World to the console
  cout << "Hello World!" << endl;
  return 0;
}
```



(A) Include header files

(B) Enable namespaces

(C) Define global variables, functions

(D) Define the main function.
It is program entry point.

Supplemental Material

Creating, Compiling, and Running Programs

1. Create/modify source code
2. Source code
3. Compiler
4. Machine code
5. Linker
6. Executable code
7. Run executable code
8. Result

```
#include <iostream>
using namespace std;
int main()
{
  // Display Hello World to the console
  cout << “Hello World!" << endl;
  return 0;
}
```

Linker:

Take one or more object files and combine them into a single executable file, library, or another object file.

Exercise:
Assume that cout is defined in both spaces:
std
my_std

- How do you use cout defined in std?

Exercise:
Assume that cout is defined in both spaces:
std
my_std

- How do you use cout defined in std?

- Include the header file for `cout`
  - call `std::cout`
  - or `using namespace std`, and then simply call `cout`

About header files

#include <string.h>

#include <string>

What are they?

About header files

#include <string.h>
#include <string>

The two files **string** and **string.h** are not the same file.

string.h: store declarations of functions that are defined in C
string: store declarations of functions that are defined in C++

Different functions are defined in these two files **string.h** and **string**.

Exercises

- Why do we need to include header files?
- When should we use “using namespace ns_s”?
- How do we create a namespace?
- How do we avoid multiple declarations?
- What is the purpose of *cout*?
- What kinds of errors a program may have?