G52CPP
C++ Programming
Lecture 12

Dr Jason Atkin

Last lecture

- const

- Constants
- const pointers
- const references
- const member functions

This Lecture

- Namespaces and scoping
- Some standard class library classes
  - String
  - Input and output

Namespaces and scoping

Namespaces

- Namespaces are used to avoid name conflicts
  - Only the name is affected
- To put code in a namespace use:

```
namespace <NamespaceName>
{
  <put code for classes or functions here>
}
```

- Can use scoping to specify a namespace to 'look in':

```
  <namespace>::<class>::<function>
```

e.g. `MyNameSpace::MyClass::foo();`

```
  <namespace>::<globalfunction>
```

e.g. `MyNameSpace::bar();`

Namespaces

- Can avoid needing to keep saying **`<namespace>::`** specify '**`using namespace <namespace>`**'
  - **From that point onwards** the namespace will be checked when resolving names

- The standard class library is in the **`std`** namespace
  - The C-type functions are also in the global (unnamed) namespace, so we have been able to ignore namespaces so far
  - A common line near to the start of C++ programs:

```
using namespace std;
```

Example of namespace

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

namespace cfj
{
   void MyPrintFunction1()
   {
       // Do something
   }

// Function in cfj namespace,
// so can use MyPrintFunction1
// without '::' or 'using'
   void MyPrintFunction2()
   {
       MyPrintFunction1();
   }
}
```

```
// Not in cfj namespace!
void MyPrintFunction3()
{
   cfj::MyPrintFunction1();
}

using namespace cfj;
// From this point onwards,
// cfj namespace will be
// checked

int main()
{
   string s1( "Test string" );
   int i = 1;

   MyPrintFunction1();
   MyPrintFunction2();
   MyPrintFunction3();
}
```

The scoping operator

- You can use the scoping operator to call global functions or access global variables
  - use `::` with nothing before it
- Also used to denote that a function is a class member in a definition, e.g.

```
void Sub::modify() { … }
```

- Left of scoping operator is
  - **blank** (to access a global variable/function)
  - **class name** (to access member of that class)
  - **namespace name** (to use that namespace)

Using scoping to access data

```
#include <cstdio>
int i = 1; // Global

struct Base
{
   int i;

   Base()
   : i(3)
   {}
};

struct Sub : public Base
{
   int i;

   Sub()
   : i(2)
   {}
   void modify()
   {
       int i = 7;    // Local
       ::i = 4;      // Global
       Sub::i = 5;   // Sub's i
       Base::i = 6; // Base's i
   }
};

int main()
{
   Sub s;
   printf( "%d %d %d\n",
       i, s.i, s.Base::i );
   s.modify();
   printf( "%d %d %d\n",
       i, s.i, s.Base::i );
   return 0;
}
```

Scoping/using example

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

namespace cpp
{
   void MyPrintFunction1()
   {
      // Do something
   }
}

void MyPrintFunction1()
{
   // Do something
}
```

```
using namespace cpp;

int main()
{
   MyPrintFunction1();
}
```

```
int main()
{
   ::MyPrintFunction1();
}
```

```
int main()
{
   cpp::MyPrintFunction1();
}
```

Which function does each main() function call?

Scoping/using example

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

namespace cpp
{
    void MyPrintFunction1()
    {
        // Do someth
    }
}

void MyPrintFuncti
{
    // Do something
}
```

```
using namespace cpp;
```

```
int main()
{
    MyPrintFunction1();
}
```

```
g++ namespace.cpp

namespace.cpp: In function “int main()”:

namespace.cpp:22:19: error: call of overloaded
“MyPrintFunction1()” is ambiguous

namespace.cpp:22:19: note: candidates are:

namespace.cpp:13:6: void MyPrintFunction1()

namespace.cpp:7:7: void cpp::MyPrintFunction1()
```

Scoping/using clarification

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

namespace cpp
{
   void MyPrintFunctio
   {
      // Do something
   }
}

void MyPrintFunction1()
{
   // Do something
}
```

```
$ g++ namespace.cpp
$

The other two work correctly, compiling
with no errors
They are unambiguous

General rule: If there is ambiguity it
will NOT compile
```

```
int main()
{
   ::MyPrintFunction1();
}
```

```
int main()
{
   cpp::MyPrintFunction1();
}
```

Standard class library classes

An introduction

We will see more later

string and std namespace

- The string class is in the **`std`** namespace
- Can be accessed as **`std::string`**
- Three of the string constructors:

  `string();`

  - Default empty string

  `string(const char* str);`

  - From a **`char*`** type string

  `string(const string& str);`

  - From another string – the copy constructor

- **`#include <string>`** for declarations

string class – for reference

- **string class has many member functions**

`append()`	concatenate more text to the string
`substr()`	return a substring of some size
`insert()`	insert some text into the string
`replace()`	replace part of a string
`erase()`	delete/remove part of a string
`assign()`	replace content of string
`compare()`	lexically compare two strings
`find()`	search for some text in the string
`rfind()`	find, starting at the end
`c_str()`	obtain a const char* for the string

- **And overloads a number of operators**

Assignment:	`=`
Comparison:	`==` `!=` `<` `<=` `>` `>=`
Concatenation:	`+` `+=`
Character at:	`[]`

streams for input/output

- C++ input/output **classes** use streams
- Three standard streams exist already
  - `istream cin;` (matches `stdin`)
  - `ostream cout;` (matches `stdout`)
  - `ostream cerr;` (matches `stderr`)
- Header file includes the declarations:
  - `#include <iostream>`
- They are in `std` namespace
  - Use `std::cin`, `std::cout`, etc
- `>>` and `<<` operators overloaded for input / output
- `endl` sent to a stream will output `\n` and flush

Example

```
#include <string>
#include <iostream>

using namespace std;

int main()
{
   string s1( "Test string" );
   int i = 1;

   cin >> i;

   cout << s1 << " " << i << endl;

   cerr << s1.c_str() << endl;
}
```

Example

```
#include <string>
#include <iostream>

using namespace std;

int main()
{
   string s1( "Test string" );
   int i = 1;

   cin >> i;

   cout << s1 << " " << i << endl;

   cerr << s1.c_str() << endl;
}
```

Header files for string and i/o

Look in **std** namespace for the names which follow e.g. **cin, cout, string**

Overloaded operator - input

Overloaded operator - output

Convert **string** to **const char***

Reminder: string and std namespace

- The string class is in the **`std`** namespace
- Can be accessed as **`std::string`**
- Three of the string constructors:

  `string();`
  - Default empty string

  `string(const char* str);`
  - From a **`char*`** type string

  `string(const string& str);`
  - From another string – the copy constructor

- **`#include <string>`** for declarations

File access using streams

- **`ifstream`** object - open the file for input
- **`ofstream`** object - open the file for output
- **`fstream`** object – specify what to open file for
  - Takes an extra parameter on open (input/output/both)

- Use the **`<<`** and **`>>`** operators to read/write
- In the same way as for **`cin`** and **`cout`**

- Simple examples follow, for reference
- Read the documentation for more information

File output example

```
#include <fstream>
using namespace std;
int main()
{
  ofstream file;
  // Open a file
  file.open("file.txt");
  // Write to file
  file << "Hello file\n“ << 75;
  // Manually close file
  file.close();
  return 0;
}
```

Since the `ofstream` object is destroyed (with the stack frame) the file would close anyway

File input example

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
   ifstream file;
   char output[100];
   string str;
   int x;
   file.open("file.txt");
   file >> output;
   file >> str;
   file >> x;
   file.close();
   cout << output << endl;
   cout << str << endl;
   cout << x << endl;
}
```

Note that the array has enough space to hold the loaded data

Text loaded (and output using **cout**) matches what was written in the previous sample

Those people struggling with **char***s may want to consider **string** for the CW

stringstream

```
#include <iostream>
#include <sstream>

using namespace std;

int main()
{
   stringstream strstream;
   string str;

   short year = 1996;
   short month = 7;
   short day = 28;

   strstream << year << "/";
   strstream << month << "/";
   strstream << day;

   strstream >> str;

   cout << "date: “ << str
                    << endl;

   return 0;
}
```

Send data to the **stringstream** object, a bit at a time

Extract it out again afterwards, as one string

I prefer **sprintf()**, for easier formatting, but this is ‘more C++’

`string/stream` comments

- **You may use the standard C++ classes in the coursework if you wish**
  - **Including the string or stream classes**
- `wstring` is a wide-character version
  - `basic_string` is a template class
  - `string` and `wstring` are instantiations
- `string` is also a container class
  - Can be treated as a container
  - e.g. use `size()`
- Know these exist, and how they are used

Standard Template Library

A large library of template classes
and algorithms
Give guarantees for the speed
Useful for coursework?

STL container classes

`vector`
`string`
`map`
`list`
`set`
`stack`
`queue`
`deque`
`multimap`
`multiset`

- In `std` namespace
- Know that Standard Template Library exists
  - If you go for C++ job interview, learn basics
- These are template classes
  e.g. `vector<int>` for `vector` of `int`s
  C++ vector class will compile-time check types
- Also have iterators
  - Track position/index in a container
  - e.g. to iterate through a container
- And algorithms (over 70 of them)
  - Apply to containers
  - e.g. min(), max(), sort(), search()
- Speed guarantees

Example of using **vector**

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main()
{
   vector<char> v(10);
   // 10 elements

   // Can ask for size
   int size = v.size();

   cout << "Size " << size
        << endl;

   // Set each value, using []
   for( int i=0 ; i < size ; i++ )
        v[i] = i;

   // Iterate through vector
   vector<char>::iterator p
                = v.begin();

   for( ; p != v.end() ; p++ )
        *p += 97; // *p syntax

   // Output the contents
   for( int i=0 ; i < size ; i++ )
        cout << v[i] << endl;

   return 0;
}
```

Exam: do I need to know all of this?

- I will expect you to be able to understand code that you have seen in lectures
  - i.e. if you can understand the lecture slides, samples and lab notes and what the code does then you meet that criterion
  - e.g. if you see the code `cout << x` then know that it sends x to the output (e.g. screen) and that it uses operator overloading to do this

- **I will expect you to know the basics of the standard C++ class library**
  - **i.e. things we cover in lectures or in the lab notes**
  - **Know the basic methods and classes that we have looked at, to use in an exam answer**
  - **E.g. be able to create an array or vector**

This lecture

- Inheritance
- Virtual functions