C++ Introduction

Basics

Contents

Features

Features:

- Object oriented like Java
- fast like C
- low level programming still possible
- functional programming also possible[1]
- programmer is in control of everything (especially memory management)

[1] Why would you do this?

Hello World!

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

Differences from C

Differences from C

The include file is different!

```
1 #include <iostream>
```

This includes general C++ stuff.

Differences from C

The include file is different!

```
1 #include <iostream>
```

This includes general C++ stuff.

A new line appeared!

```
1 using namespace std;
```

This means that we want to use std functions (like std::cout) without writing std.

Differences from C

The include file is different!

```
1 #include <iostream>
```

This includes general C++ stuff.

A new line appeared!

```
1 using namespace std;
```

This means that we want to use std functions (like std::cout) without writing std.

The "Hello World!" line looks different!

```
1 cout << "Hello World!" << endl;
```

That's because we use streams (more on that later).

New data types

There are new data types!

Most importantly:

- string - replaces char*
- vector - replaces arrays
- stream - reads bytes in order

Note: all of these are in the 'std' namespace!

String

What is it?

- a class for character arrays (and is not a pointer)
- can be converted to char* this c_str()
- length is known
- can be extended
- has overloaded operators for assignment, assertion, ...

Important:
Include the string header file with:

```
1 #include <string>
```

String – Usage

Declaration:

```
1 string str;
```

Assignment:

```
1 str = string("This is a string!"); //or
2 str = "This is also a string";
```

Altogether:

```
1 String str1 = string("This is a string!"); //or
2 String str2 = "This is also a string";
```

Concatenation:

```
1 string s1 = "one string " + "other string";
```

Vector

What is it?

- a array wrapper
- size changeable
- memory is reallocated on size change
- size is known
- as fast as arrays
- usable like an array or like an object

Important:
Include the string header file with:

```
1 #include <vector>
```

Vector – Usage

Declaration:

```
1 vector<int> vec;
```

Assignment:

```
1 vec = vector<int>{1, 2, 3};
```

Altogether:

```
1 vector<int> new_vec = vector<int>{1, 2, 3};
```

Adding a new value:

```
1 vec.push_back(3);
```

Accessing a value:

```
1 cout << vec.at(1) << endl; //or
2 vec[1] = 3;
```

Streams

What is it?

- a destination for byte characters
- converts all data types to human readable byte array
- work in two directions
- easy to use

Stream – Usage

Declaration:

```
1 streamType stream;
```

Feed characters:

```
1 stream << "This is a string! " << 3 << 'A' << endl;
```

Example:

```
1  //write to standard output
2  cout << "This is a string! " << 3 << 'A' << endl;
3  //write to file
4  fstream stream("test.txt");
5  if ( stream.is_open() ) {
6      stream << "This is a text" << endl;
7      stream.close();
8  }
9  //read form standard input
10 string str;
11 cin >> str;
```

Stream – Standard Out

Example:

```
1 #include <iostream>
2 //...
3 std::cout << "Text" << std::endl;
4 /*Output:
5  * 'Text'
6  */
```

Stream – String Stream

Example:

```
#include <string>
#include <sstream>
//...
std::stringstream ss;
ss << "text " << 50;
std::string str;
int i;
ss >> str >> i;
std::cout << str << i;
/* Output:
 * 'text 50'
 */
```

Stream – File Stream

Example:

```
#include <fstream>
//...
std::fstream fs;
string text;
fs.open("test.txt", std::fstream::in | std::fstream::out);
if(fs.is_open()) {
    fs >> text;
    fs << " more text";
    fs.close();
    std::cout << text << std::endl;
}
```

Stream – File Stream – Result

Before (test.txt):

```
1 Here is text.
```

Stream – File Stream – Result

Before (test.txt):

```
1 Here is text.
```

Output:

```
1 Here
```

Stream – File Stream – Result

Before (test.txt):

```
1 Here is text.
```

Output:

```
1 Here
```

After (test.txt):

```
1 Here more text
```