Hello, C++

What is C++

- A general purpose programming language
- Originally based on C
- Supports multiple programming paradigms
  - procedural
  - object oriented
  - generic
  - functional
- Developed and implemented by Bjarne Stroustrup

A Simple C++ Program

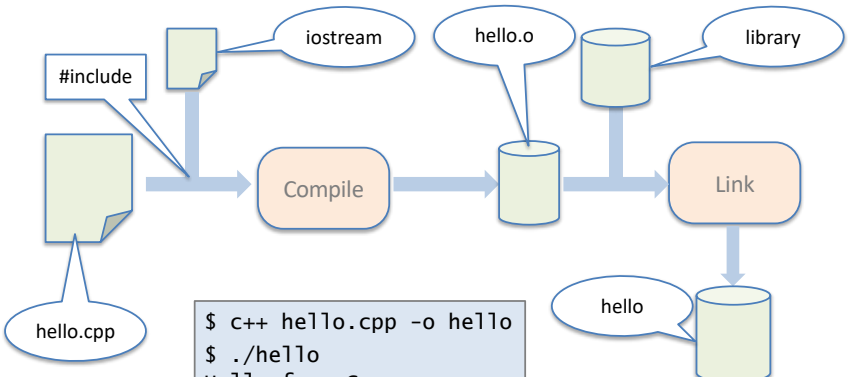
- When a C++ program is run, the operating system uses the function main() as an entry point

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
  cout << "Hello from C++" << endl;
  return 0;
}
```

```
$ c++ hello.cpp -o hello
$ ./hello
Hello from C++
$
```

Writing, Building, Running a C++ Program

- C++ Programs normally consist of a number of "modules", which are individually compiled and then linked to form an executable program




```
$ c++ hello.cpp -o hello
$ ./hello
Hello from C++
$
```

A Simple C++ Program

- When a C++ program is run, the operating system uses the function main() as an entry point

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
  cout << "Hello from C++" << endl;
  return 0;
}
```

```
$ c++ hello.cpp -o hello
$ ./hello
Hello from C++
$
```

A Simple C++ Program

- The main() function is called by the system runtime
  - Should return – to indicate successful completion

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
  cout << "Hello from C++" << endl;
  return 0;
}
```

Function definition: no arguments returns int value

Braces uses to delimit a block

Must pass back value of the correct type

A Simple C++ Program

- Standard output is represented by the C++ object cout
  - The << operator is used to send data to cout

```
#include<iostream>

using std::cout;
using std::endl;

int main()
{
  cout << "Hello from C++" << endl;
  return 0;
}
```

Multiple elements can be sent to cout in one statement

endl represents a system dependent new-line

A Simple C++ Program

- Some additional code is required to allow compilation to succeed

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
  cout << "Hello from C++" << endl;
  return 0;
}
```

Allows the compiler to see the declaration of cout to check its use

Define qualification of symbols in a namespace

Variables

- Named memory locations
  - Associated with a type, must be declared with their type

```
int main()
{
 int number;
 cout << "Please enter a number: ";
 cin >> number;
 cout << number << " * 2 = "
      << number * 2 << endl;
 return 0;
}
```

Declare variable to hold int value

Read from stdin into variable

```
$ c++ -o var1 var1.cpp
$ ./var1
Please enter a number: 2
2 * 2 = 4
$
```

Basic Integer Types

Name	Defined Size	Common Size	Value Range
char	ASCII/ANSI	8 bits	+/- 2^{31}
short	>= 16 bits	16 bits	+/- 32K
int	>= 16 bits	32 bits	+/- 2^{31}
long	>= 32 bits	64 bits	+/- 2^{31}
long long	>= 64 bits	64 bits	+/- 2^{63}

- Use unsigned prefix to cause range of values to start at 0
  - No negative numbers
- long long standardised in C++11
  - Available in other earlier versions though

Declaring and Initialising Variables

- Variables must be declared before use
  - May be declared at any point
  - Always declare variables as close as possible to first use
- Initialisation is allowed at declaration
  - Otherwise initial value is determined by other factors (see later)

```
int a = 0;
double d = 1.7e7;
float f = 2.72828f;
```

```
short c = 027;
long l = 0x14E;
unsigned long e = 4ul;
```

Other Fundamental Data Types

```
int main ()
{
 int i;
 char c;
 double d;
 bool b;      // Really an alias for int
 i = 33;
 c = 'a';
 d = 3.14983; // Literals are double
 b = true;
 int hexnum = 0x522;
 float f = 332.2;
 int a = 'A';
 std::cout << "value in a: "
           << a << std::endl;
 auto ai = i;   // From C++11
 return 0;
}
```

- wchar_t also available
  - 16 bit characters
  - Unicode

About Intrinsic Types

- C++ allows size and capacity of types to be discovered
  - Aids portability
- Sizeof operator yields the size in bytes of a given type or variable
  - E.g. sizeof(int)
  - Returns value of type size_t, typically a synonym for unsigned int
- Min and Max values for types can be discovered in file limits.h
  - Discussed later

const and volatile

- const type qualifier denotes values that do not change
  - Must be initialised at definition

```
const int buffer_size = 128;
const char choice = 'q';
```

- const should be used whenever possible
  - Immutability is considered a good thing
  - Compiler prevents inadvertent updates of variable through assignment

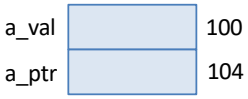
```
bufsize = 32;  // Not allowed
```

  - Type safe, preferable to #define for symbolic values
- volatile indicates variable may change value unexpectedly
  - Important in concurrent code

Pointers

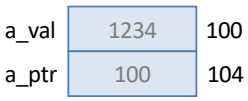
- A pointer holds the *address* of another variable
  - Rather than simply holding a value

```
int a_val;
int *a_ptr
```



- Use the & operator to refer to an address

```
a_val = 1234;
a_ptr = &a_val;
```



- Use the * operator to fetch value through a pointer

```
std::cout << *a_ptr << std::endl;
```

References

- References are a specialised variation on pointers
  - Like a constant pointer to a variable
  - Acts as an alias for the variable
  - No need for special operators

```
int main ()
{
 int i = 100;
 int&i_ref = i;

 ++i_ref;
 std::cout<<"i_ref: "<<i_ref<<std::endl;

 return 0;
}
```

```
$ c++ -o refs refs.cpp
$ ./refs
i_ref: 101
$
```

Strings

- Strings may be represented in different ways
  - Chararacter arrays: "C Style Strings"
  - Standard library string type

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

using std::string;
using std::cout;
using std::cin;
using std::endl;

int main()
{
 string name;
 cout << "Please enter your name: ";
 cin >> name;
 cout << "Hello " << name << endl;
 return 0;
}
```

Definition of the string type

```
$ c++ -o str1 str1.cpp
$ ./str1
Please enter your name: George
Hello George
$ ./str1
Please enter your name: John Doe
Hello John
$
```

>> function uses whitespace to delimit input strings

struct

- struct introduces an *aggregate* type
  - A group of elements that can be be treated as a single entity
  - Basic data structure

```
struct Person {
 std::string name;
 int age;
};
```

- Access fields using . operator

```
Person p;
p.name = "John Doe";
p.age = 21;

std::cout << p.name << ": "
          << p.age << std::endl;
```

union

- Represents a value that may be of different types
  - Depending on usage

```
union Result {
 double res_val;
 int err_code;
};
```

Allocate enough memory to hold the largest type

- Often embedded within another aggregate
  - To determine the value's type

```
struct op_result {
 bool OK;
 Result res;
};
```

- Beware, association is not enforced through the type system