Laboratorio di Tecnologie dell'Informazione

Ing. Marco Bertini
marco.bertini@unifi.it
http://www.micc.unifi.it/bertini/

“Hello world”: a review

Some differences between C and C++

- Let's review some differences between C and C++ looking at the "Hello world" program

```
#include <iostream>

int main() {
   // let's greet
   std::cout << "Hello, ";
   string name = "world\n";
   std::cout << name;
}
```

The `main` function

- In C++ there are two variants of the function main():
  - int main()
  - int main(int argc, char **argv)
- It is not required to use an explicit `return` statement at the end of main. If omitted main returns 0.

Comments

- There's a new (compact) comment style:
  Two types:
- C Style: `/* block comment */`
  - Usually only used for block comments at the top of programs, classes, functions etc.
  - Can extend over multiple lines
- C++ style: `// comment to end of line`
  - Usually used for most in line comments against variables or algorithm code

C++ library headers

- In C++ all the library headers have names without the .h extension
  - You can still use C header files with the .h extension (but you shouldn't)
- Instead, C++ versions of C header files have the same name with a 'c' in front, e.g.:

```
#include <cmath>
```

Variable declared as needed

- In C we are required to declare all variables either globally or at the top of a function before any executable code
- In C++ we can declare variable anywhere, but always before they are used
  - We tend to declare variables only when we need them and as close as possible to their use
  - A control variable for a for loop is often declared inside the for loop, e.g.

```
for (int i = 0; i < 100; i++) {
```

Default function arguments

- In C++ it is possible to provide "default arguments" when defining a function. These arguments are supplied by the compiler when they are not specified by the programmer.
- Functions may be defined with more than one default argument. They are used only in function calls where trailing arguments are omitted — they must be the last argument(s).
- Default arguments must be known at compile-time since at that moment arguments are supplied to functions. Therefore, the default arguments must be mentioned at the function's declaration, rather than at its implementation.

Default function arguments - cont.

Example:

- ```
  // sample header file
  extern void three_ints(int a, int b=4,
  int c=3);
  ```
- ```
  // code of function in .cpp file
  void three_ints(int a, int b, int c) {
     ...
  }
  ```

Type-safe C++ Stream I/O

- Avoid using the type-unsafe C I/O library (`printf`, `scanf`, etc.), where you need to specify the format of the data (e.g. "%d")
- C++ overloads the right-shift operator (>>) for input and the left-shift operator (<<) for output
  - The target of the I/O is a stream, eg. `cin` (standard input), `cout` (standard output) or a file stream

Operator overloading

- We'll come back on this later but notice:

  in C++ it is possible to define functions and operators having identical names but performing different actions. The functions must differ in their parameter lists (and/or in their const attribute).

  In this example we used overloaded << and >>

C++ Style Strings

- Type safe, space-safe!
- Contrast with C style nul terminated strings
- Can input or manipulate C++ strings without worrying about overrunning the available memory -the string will expand as required!
- But the only way to easily declare a constant string or string literal is as a C-style nul terminated string. , e.g. :

```
string s = "world";
```

Namespace

- A facility for partitioning names (of types, variables, functions etc.)
  - Allows large programs to be built from various components without risk of name clashes
  - A namespace is essentially the ability of the compiler to keep names of functions, variables and type, in separate groupings so that names in different groups can be the same without clashing with each other.
- C++ Standard Library has names in a namespace called `std` as in `std::cout`

Qualified names

- A name can be qualified, using the `::` (scope resolution) operator, with the namespace in which it has been declared. For example, `std::cin` refers to the name *cin* from the namespace *std*.
- the operator :: resolves the scope of the variable/method we are selecting: we are looking for the one of the class whose name is to its left

`using` directive

- If a name is used frequently within a segment of code you can place a *using* declaration into the source file, e.g. `using std::cin;` would say that all the *cins* refer to the one from the namespace *std*.
- To refer to any of the names in a namespace as if they were declared globally, you can place a `using` directive into the source file, e.g.: `using namespace std;` brings all the names from the namespace *std* into use.

Using C functions in C++

- Name mangling (sometimes called name decoration) is a technique used to solve various problems caused by the need to resolve unique names for programming entities in many modern programming languages.
- It provides a way of encoding additional information in the name of a function, structure, class or another datatype in order to pass more semantic information from the compilers to linkers.
- C++ compilers use it, C compilers do not. As C language definitions are unmangled, the C++ compiler needs to avoid mangling references to these identifiers.

Using C functions in C++ - cont.

- To use a C function in a C++ program we need to declare it as... C function, e.g.:

```
extern "C" void *xmalloc(int size);
```

or:

```
extern "C"{
     // C-declarations go in here
     void *xmalloc(int size);
}
```

or:

```
extern "C"{

     #include <xmalloc.h> // contains declaration of xmalloc

}
```

Using C functions in C++ - cont.

- Another common solution is to exploit the __cplusplus symbol defined by the C++ compilers:

```
// This is xmalloc.h
#ifdef __cplusplus
extern "C" {
#endif
        /* declaration of C-data and functions are
inserted here. */
        void *xmalloc(int size);

#ifdef __cplusplus
}
#endif
```

Credits

- These slides are (heavily) based on the material of Dr. Ian Richards, CSC2402, Univ. of Southern Queensland