EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Introduction to Scientific Programming

Introduction to C++ - ”Hello World!”

Christian Strandhagen

SS 2021

Kepler Center for Astro and Particle Physics

Contents

The Anatomy of a C++ Program

The structure of a C++ program

We use the “Hello World“ example to explain the structure of a C++ program.

hello_world.cpp

```cpp
/* This is a block comment.
   It can span multiple lines. */

#include <iostream>

// This is a single line comment.
int main()
{
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

include

```
#include <system-wide-header>

#include "/full/path/to/ownheader.h"
#include "../relative/path/to/ownheader.h"
```

The include statements are not part of the actual code, but are so-called preprocessor directives. These start with the # sign and don't need a semicolon at the end of the line.

They are used to make code defined in other files available in your program.

You can include system wide headers[1] (e.g. things from the standard libary) or your own headers.

[1] We will discuss later what headers are.

The main function

```
int main()
{
    //Put your code here

    return 0;
}
```

In C++ you need to define a function called `main`, that serves as the entry point of the program.

The main function is executed when you run the compiled program. It returns a number:

- 0 means the code has run without problems
- any other number can be used as an error code

Function Definition

```
<type> function_name(<type> arg1, <type> arg2)
{
    //Put your code here

    return <return_value>;
}
```

For every function (also for `main`), you have to define the return type (e.g. `int, float`) in front of the function name.

A list of arguments is specified in round brackets. Each function argument needs to be defined with a data type. The arguments are used to bring data into the function.

C++ uses curly brackets (`{...}`) to define code blocks. In this case, everything in the code block is executed when the function is called.

To pass data back to the calling code you need the `return` statement.

Input/Output - iostream

To enable input/output from and to the terminal, you need to include the `iostream` header. This makes available the input stream object `cin` and the output stream object `cout`.

They live in the namespace `std`, so they have to be prefixed with `std::`[2].

You can add something to the output stream using the operator `<<`, conversely you can read from the input stream using the operator `>>`.

See the example on the next slide.

[2] More on namespaces maybe later

Input/Output - iostream

Example

```cpp
#include <iostream>

int main()
{
    int number;

    std::cout << "Please enter a number: ";
    std::cin >> integer;
    std::cout << "You have entered " << integer << "!" << std::endl;

    return 0;
}
```

You can chain the output as long as you want. To force a new line, you can use the newline character `\n` in the text or use `std::endl`[3].

Numbers are automatically converted in both directions.

[3] There are some subtle difference for the advanced users.

Compiling and Running Hello World

Compilation

C++ code needs to be translated (compiled) into machine code in a separate step. This is done by a so-called compiler.

Popular free C++ compilers are e.g. `g++`[4] or `clang`[5]

The compiler checks the code for syntax errors and translates the source code into machine code.

[4] `g++` is part of GCC (the GNU compiler collection) https://gcc.gnu.org/
[5] https://clang.llvm.org/

Compiling Hello World

- Open your favorite text editor
- Create a new file (with the ending .cpp)
- Write your code
- Open command prompt and navigate to folder containing your source file
- Compile the code and run it

```
user@pc > g++ helloworld.cpp -Wall -o helloworld
user@pc > ./helloworld
```

If you can't execute the code because of a permission error, you might need to make the output file executable

```
user@pc > chmod u+x helloworld
```

Some g++ compiler options

Here are some useful compiler options for g++

-o name defines the output filename (default is `a.out`)

-Wall enables all compiler warnings (it's good practice to have no warnings pop up)

-O enables compiler optimizations (use -O2 or -O3 for more agressive optimizations)

-g adds information for debuggers[6] (only use when needed)

[6] More on debuggers later