Introduction

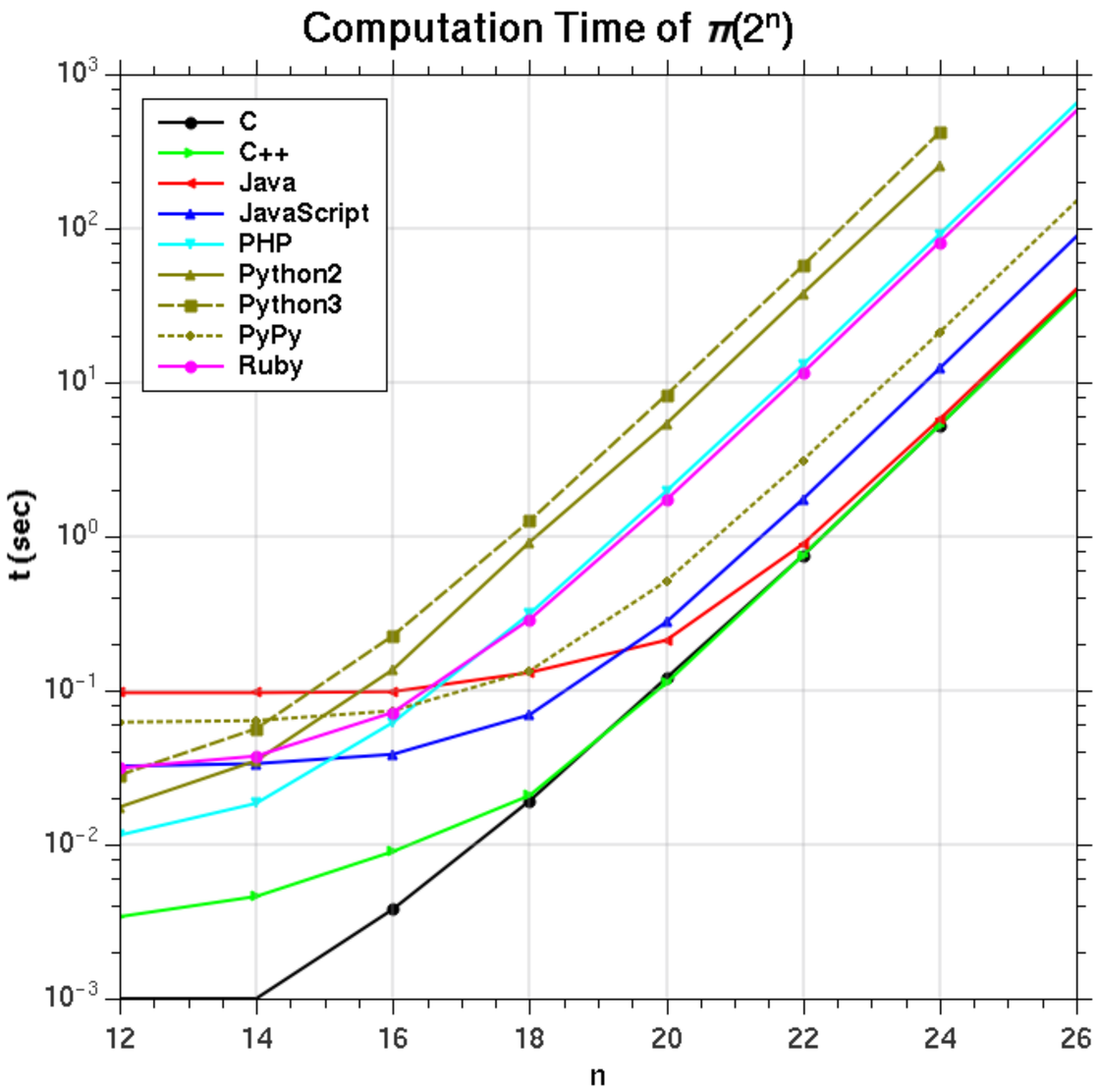
Warwick RSE

28/09/2023

What is this course for?

- C++ is a very powerful but quite complex language
- C++ has substantial advantages as an academic development language
  - Because of the complexity it can be easy to fall into “doing things because they are what you do” without understanding them
- This course attempts to give you a more grounded understanding of C++ to help you understand what you are doing and why you are doing it
  - From this base you can learn more advanced features of the language and develop more complex code while retaining performance and safety

Why Program in C++



- Faster code than Matlab, Python, R etc.
- As fast as C (at least on any problem that isn't already very fast)
- Easier to write than C and fewer opportunities for serious errors
  - Writing C like code in C++ is not better!
- Lots of libraries and good to write your own libraries in
- Lots of experienced developers in academia

Language Standards

- C++ is a language that is defined by ISO standards
  - Mostly a kind of “tick-tock” development of major change followed by update based on experience
- C++98/03 - Original C++ standard. Introduced much of the “shape” of the language. 03 firmed up the structure and approach
- C++11/14 - Added new features that were aimed at improving developer productivity. 14 expanded the capabilities of the new features
- C++17 - Mostly aimed at adding features to make C++ code more readable and remove boilerplate code
- C++20 - Variety of changes, mostly concentrating on features for people writing libraries in C++
- C++23 - Upcoming standard, bit of a mixture of things again

Language Standards

- C++ is a language that is define by ISO standards
  - Mostly a kind of “tick-tock” development of major change followed by update based on experience
- C++98/03 - Original C++ standard. Introduced much of the “shape” of the language. 03 firmed up the structure and approach
- C++11/14 - Added new features that were aimed at improving developer productivity. 14 expanded the capabilities of the new features
- **C++17 - Mostly aimed at adding features to make C++ code more readable and remove boilerplate code**
- *C++20 - Variety of changes, mostly concentrating on features for people writing libraries in C++*
- C++23 - Upcoming standard, bit of a mixture of things again

Intro to C++

Warwick RSE

28/09/2023

Introduction

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " << mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
#include <iostream>

int mymax(int i1, int i2){
  if (i1>i2) return i1;
  return i2;
}

int main(){
  std::cout << "Maximum of 1 and 10 is " << mymax(1,10) << "\n";
  std::cout << "Maximum of 1 and -1 is " << mymax(1,-1) << "\n\n";

  int mytestval = 5;
  for (int value1 = 1; value1 < 10; value1++){
    std::cout << "Maximum of " << value1 << " and " << mytestval << " is " <<
mymax(value1, mytestval) << "\n";
  }
  return 0;
}
```

C++

```
g++ 01-Simple.cpp -o 01-Simple
```

```
./01-Simple
```

```
Maximum of 1 and 10 is 10
Maximum of 1 and -1 is 1

Maximum of 1 and 5 is 5
Maximum of 2 and 5 is 5
Maximum of 3 and 5 is 5
Maximum of 4 and 5 is 5
Maximum of 5 and 5 is 5
Maximum of 6 and 5 is 6
Maximum of 7 and 5 is 7
Maximum of 8 and 5 is 8
Maximum of 9 and 5 is 9
```

Multiple Files

02a-main.cpp

```
#include <iostream>
#include "02b-functions.h"

int main(){
  std::cout << "Result of my_function(double) is " << my_function(1.234) << "\n";
  std::cout << "Result of my_function(string) is " << my_function("Hello world”)<<"\n";
  function_with_no_return("I do not return anything");
}
```

02a-main.cpp

```
#include <iostream>
#include "02b-functions.h"

int main(){
  std::cout << "Result of my_function(double) is " << my_function(1.234) << "\n";
  std::cout << "Result of my_function(string) is " << my_function("Hello world”)<<"\n";
  function_with_no_return("I do not return anything");
}
```

02a-main.cpp

```
#include <iostream>
#include "02b-functions.h"

int main(){
  std::cout << "Result of my_function(double) is " << my_function(1.234) << "\n";
  std::cout << "Result of my_function(string) is " << my_function("Hello world”)<<"\n";
  function_with_no_return("I do not return anything");
}
```

02a-main.cpp

```
#include <iostream>
#include "02b-functions.h"

int main(){
  std::cout << "Result of my_function(double) is " << my_function(1.234) << "\n";
  std::cout << "Result of my_function(string) is " << my_function("Hello world”)<<"\n";
  function_with_no_return("I do not return anything");
}
```

02a-main.cpp

```cpp
#include <iostream>
#include "02b-functions.h"

int main(){
  std::cout << "Result of my_function(double) is " << my_function(1.234) << "\n";
  std::cout << "Result of my_function(string) is " << my_function("Hello world”)<<"\n";
  function_with_no_return("I do not return anything");
}
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.h

```
#ifndef FUNCTIONS_HEADER_H
#define FUNCTIONS_HEADER_H
#include <string>

  double my_function(double d);
  std::string my_function(std::string s);
  void function_with_no_return(std::string s);

#endif
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

02b-functions.cpp

```cpp
#include "02b-functions.h"
#include <iostream>

double my_function(double d){return d * 2.0;}
std::string my_function(std::string s){return s;}
void function_with_no_return(std::string s){std::cout << s << "\n";}
```

Compiling

```
g++ 02b-functions.cpp 02a-main.cpp
```

```
g++ -c 02b-functions.cpp
g++ -c 02a-main.cpp
g++ 02b-functions.o 02a-main.o
```

```
g++ 02a-main.cpp 02b-functions.cpp
```

```
g++ -c 02a-main.cpp
g++ -c 02b-functions.cpp
g++ 02b-functions.o 02a-main.o
```

```
g++ -c 02b-functions.cpp
g++ 02b-functions.o 02a-main.cpp
```

```
./a.out
```

```
Result of my_function(double) is 2.468
Result of my_function(string) is Hello world
I do not return anything
```

~~`g++ 02b-functions.h`~~

Other variables

Const variables

- Inherited from C, but more commonly used and extended in C++
- You can flag variables as **constant** by putting **const** before the type of the variable when you declare it
- **const** variables **must** be assigned a value on the line where they are declared
- After that line they **cannot** be changed
  - If you try to change a **const** variable's value the code will not compile
- **const** parameters to functions cannot be changed inside that function
- You cannot pass a **const** value to a function as a non-**const** parameter

auto variables

- Quite often in C++ you will see code like
  - **int value = get_integer();**
- Fine for integer, but for more complex functions you have to find out and match the variable type to the return type
  - The compiler checks that you've used the right type, so it must know what the right type is - can't it work out the type of my variable for me?
- **YES** - use **auto** in place of the type when declaring a variable and assign it a value on the same line and the compiler will "paste-in" the correct return type
- You are still giving the variable a fixed type, just telling the compiler to work out which

References

“The Angry Penguin“, used under creative commons licence from Swantje Hess and Jannis Pohlmann.

Warwick RSE

28/09/2023

Concept

What is a referencce

- A reference variable (usually just called a reference) is a variable that doesn't have a value of its own - it **refers** to another variable
- One of the distinguishing features of a **reference** is that it can be used anywhere the variable that it refers to can be used - no extra code is needed to use a reference
- Reference variables have to be **initalized** when they are **defined**
  - i.e. reference variables are created and must be immediately told which variable they refer to - they will refer to the same variable for their entire lifetime
- So if they only refer to a single variable for their whole lifetime, why are they useful?

Pass by reference

Pass by reference languages

- You've probably never thought about what the computer is actually doing when it calls a function
- There are a lot of complexities, but in general it has to prepare a special structure called a **stack frame** which contains the parameters to the function and jump to a special point in the program code that uses the data in the stack frame to execute the function
- How does it put the variables into the stack frame?
  - Actually quite complex, but two basic approaches

Pass by value

- C and C++ are pass by value languages
- Variables are put into the stack frame by **copying** them
- The variables in the function are **not** the variables that were passed as parameters to the function when it is called
  - They have the same values, but they refer to completely different parts of memory
- Changing the parameters in a function **doesn't** change the variables that were used as parameters to the function calls

Pass by reference

- There aren't really any modern **strict** pass by reference languages, but Fortran is **almost** a pass by reference language. We're going to describe a strict pass by reference language for clarity
- Variables are put into the stack frame as **references to the parameters**
- The variables in the function **are** the variables that were passed as parameters to the function when it is called
  - They refer to the same underlying memory
- Changing the parameters in a function **does** change the variables that were used as parameters to the function calls

Pass by reference

- A question then is **why** would you want to pass a variable by reference
- Basically two reasons
  - I want to modify the value of a parameter that I pass to a function
  - I want to pass such a large object that copying it would take too much time or memory

Reference Parameters

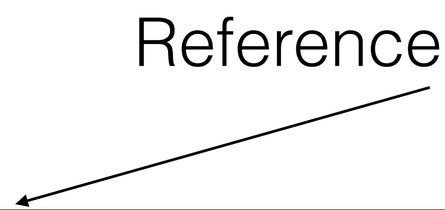
- Effectively if you have a reference parameter to a function in C++ then you make **that parameter** passed by reference
- You can control for each parameter whether you pass it by value or reference
- Mostly this makes no other changes to the code - call the function just like you always used to
- There are some restrictions about what you can pass to a function parameter expecting a reference
  - Mostly you can't pass **literals**, that is things like **"Hello world!"** or **14**
  - You can pass more things to a **const** reference since that is a guarantee that you won't change the variable

Reference Parameters

```
void demo_function(int &i){
  i+=5;//Increment i by 5
}

int main(){
  int i=7;
  demo_function(i);
  std::cout << "Modified value is " << i << "\n";
  //demo_function(17) won't work because you can't take a reference to a literal
}
```

Reference Parameters




```
void demo_function(int &i){
  i+=5;//Increment i by 5
}

int main(){
  int i=7;
  demo_function(i);
  std::cout << "Modified value is " << i << "\n";
  //demo_function(17) won't work because you can't take a reference to a literal
}
```

Reference Parameters

- Specifying that a function parameter is a reference in C++ is very easy
  - Prepend the name of the variable with an **&** when it is declared
    - **&** is used in other places in C++, but when it is placed before the name of a variable in a declaration it **only** makes that variable a reference

Reference Variables

- You can create a reference variables just as easily as a reference function parameter, and in the same way
- You always have to then initialise the reference variable with the variable that it is to be the reference to
- The easiest way of doing that is just by assignment
  - **int i;**
    int &i_ref = i;
- **N.B! The & only goes with a specific variable name**
- **int i, &i_ref=i; is exactly the same as the above example!**

Reference Variables

- Why would you want a reference variable?
  - Especially if you can't change what the variable refers to?
- Mostly to store a reference to a variable inside some kind of "container" (we'll come to C++ built in containers properly in the next session)
- For example, you can select an element from an array, store it in a reference variable and then modify its value later
- This is particularly useful if you have quite complex logic to refer to an item
  - i.e. First select an array from a set of possible arrays, then select an item from that array
- Can also avoid branches in code which can improve performance if you have to use "if" to select an item

Reference Variables

```
int main(){

  int values[10]={1,2,3,4,5,6,7,8,9,10};
  int &value = values[random_index(10)];//Select a random element from 0 to 9
  value=95; //Set a random element of the array to 95

  for(int i=0;i<10;++i) std::cout << values[i] << " ";
  std::cout << "\n";

}
```

Reference Variables

- More commonly reference variables are used as part of a **class**
- We'll cover this later in detail, but you can write a **constructor** that allows you to initialise the reference variable when the class is constructed
- Useful way of having a class that keeps a reference to another class or a shared resource
- Also commonly, you might have a member function of a class that returns a reference to a member variable of a class
  - Lets you change internal state of a class while still controlling access

Pointers

- References are similar to the concept of **pointers**
- Pointers are also variables that refer to other variables
- The difference between pointers and references are
  - Pointers can be repointed to new variables
  - Pointers can be **nullified** - assigned to point to a special location that means “not pointing to anything”
  - In most languages pointers are *generally* not usable interchangeably with variables - they have to be **referenced** and **dereferenced**. C/C++ pointers are like this
    - Fortran pointers are an exception - they behave more like references in many ways

Pointers

- One big thing that you can do with pointers that you can't do with references is **manual memory management**
- That is request memory directly at a point in your code and then release it for reuse when you are done with it
- Common in C code (although one of the major sources of error in C codes!)
- **NOT CONSIDERED GOOD FORM IN NORMAL C++ CODE**
- Mostly you can write good C++ code without doing this which reduces the possibilities for error