Systems Programming

Lecture 17: Introduction to C++

Stuart James

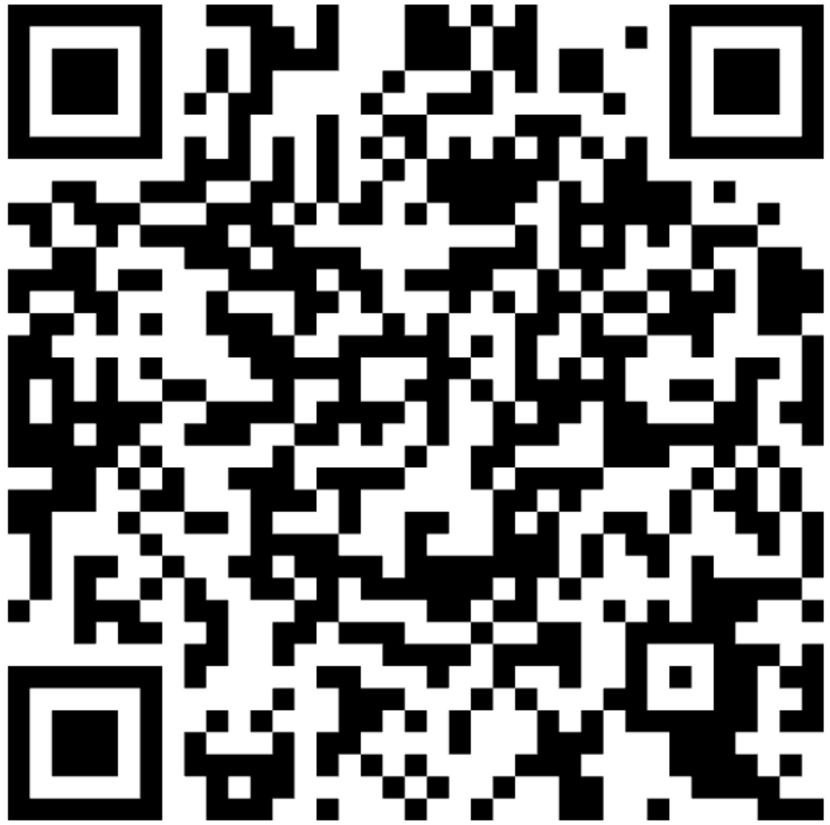
stuart.a.james@durham.ac.uk

Amir Atapour-Abarghouei: amir.atapour-abarghouei@durham.ac.uk

(Slide thanks to Amir Atapour-Abarghouei and Anne Reinarz)

Recap

https://PollEv.com/stuartjames



Recap Parallel Computing

Definitions:

- **Parallelism:** Multiple computations are done simultaneously, e.g. on a multicore processor.
- **Concurrency** is when two or more tasks can start, run, and complete in overlapping time periods.
  - It doesn't necessarily mean they'll ever both be running at the same instant, e.g. multitasking on a single-core machine.
- **Process:** An instance of a program that is being executed in its own address space.
  - Each process maintains its own heap, stack, registers, file descriptors etc.
- **Thread:** A light weight process that shares its address space with others.
  - Each thread maintains the bare essentials: registers, stack, signals.


Recap pthreads

- The `pthread_create()` function starts a new thread. The new thread starts execution by invoking `start_routine()`.

```
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void *),
                   void *restrict arg);
```

- The `pthread_join` function waits for the thread specified by `thread` to terminate. If that thread is terminated, then `pthread_join()` returns.

```
int pthread_join(pthread_t thread, void **retval);
```


Recap pthreads

- **Race condition**, and is typical of the types of problems that come up when multiple threads share access to the same data.
- A common way to implement a critical section of the program is through a **mutual exclusion variable (mutex)**.
- If you forget to unlock the lock, everyone else waits forever (deadlock!).


C++

Resources and Books

- "C++ Primer" by Stanley Lippman, Josée Lajoie, and Barbara E. Moo
  - More accessible (5th edition covers C++11)
- Online:
  - http://www.cprogramming.com/
  - Free book How to think like a computer scientist C++: http://www.greenteapress.com/thinkcpp/
- https://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list


Resources and Books

More information:

- "The C++ Programming Language" by Bjarne Stroustrup
  - By The designer of C++
  - Assumes you know everything, a good reference book (4th edition covers C++11)
- "Programming: Principles and Practice Using C++" by Bjarne Stroustrup
  - Also by the designer of C++
  - Does not assume previous programming experience (2nd edition covers C++14)


Using lecture resources

	C	C++
Jupyter Kernel	https://github.com/brendan-rius/jupyter-c-kernel	https://github.com/StTu/jupyter-cpp-kernel
Requirements	gcc jupyter python 3 pip	g++ jupyter python 3 pip
Install	pip install jupyter-c-kernel install_c_kernel jupyter notebook	pip install jupyter-cpp-kernel install_c_kernel jupyter notebook

Support: Linux and Mac. For Windows would require adaption contact me and we can work on it. However, probably can be done using Windows Subsystem for Linux (WSL).

A Brief history of C++

- Started in 1979 by Bjarne Stroustrup as "C with classes," trying to add some OOP features to C e.g. classes, derived classes, strong typing, inlining and default arguments
- In 1982, he started developing the sucessor to C with classes and called it C++. New features included:
  - virtual functions,
  - function name and operator overloading,
  - references,
  - constants,
  - type-safe
  - free-store memory allocation (new/delete),
  - improved type checking
- In 1985, he published the book "The C++ Programming Language," which became the definitive reference for the language


A Brief history of C++

- C++ 2.0 was released in 1989. New features included:
  - multiple inheritance,
  - abstract classes,
  - static member functions,
  - const member functions,
  - protected members
- Later feature additions included:
  - templates,
  - exceptions,
  - namespaces,
  - new casts
  - a Boolean type
- In 1998, C++98 became the first C++ standard and had a small update (C++03) in 2003


A Brief history of C++

- In 2011, C++11 added lots of new features and enlarged the standard library. This version and the version that follow it are sometimes known as "Modern C++"
- C++14 and C++17 were relatively minor changes
- C++20 fundamenentally changes how we write C++ code. New features include:
    - coroutines, concepts, modules and ranges
    - https://github.com/AnthonyCalandra/modern-cpp-features
- C++23:
    - library support for coroutines, a modular standard library, executors, and networking.


C++: Compilers

- Clang (clang++)
- gcc (g++) - recommended
- Intel C++ compiler (icc, icpp)

Under windows try Visual Studio:

- https://devblogs.microsoft.com/cppblog/category/new-user/


C++: Integrated Development Environments (IDE)

- Windows, Linux and MacOS:
  - Codeblocks
  - Eclipse
  - Netbeans
  - QT creator
  - Kdevelop
  - Visual Studio (or Visual Studio Code)
- MacOS only:
  - xCode


C++ Overview

- Next 3 sessions:
  - New headers
  - Namespaces
  - New way of doing input and output
  - New memory management
  - Virtual functions, and classes
  - Operator overloading
  - Templates and the standard Template Library (STL)
- Last lecture:
  - Discussion of C++20 and C++23 features


C++

Namespaces


C++ namespaces

Similar to python:

```
import numpy as np
```

call `np.array([1,2,3])`


C++ namespaces


C++ namespaces

In [2]:

```
namespace ns {
    void array(int) {}
}

int main(){
    ns::array(10);
}
```


C++ namespaces

- Included headers will end up in their own namespace
- C headers will all be in the same namespace


C++ namespaces

- Namespaces allow you to group functions and classes into logical groups
  - avoid collisions in naming
- To work in the same namespace within your scope:
  - `using namespace ns`


C++

New headers


New Headers

- C++ allows the use of C libraries
  - compatability with existing code bases
- When writing new code: stick with either C or C++
  - Mixing e.g. methods of memory management can lead to bugs


New Headers

- C++ headers do not have ".h" ending
- C++ standard library headers are included in the compiler
- Example:
  - In C we use: `#import <stdio.h>`
  - In C++ we use: `#import <iostream>`


C++

New way of doing input and output


Introduction to `iostream`

- Input and output
- Text input and output supported by the `iostream` library
- This predefines three streams for I/O:
  - `cin` - standard input (terminal) (c.f. `stdin` in C)
  - `cout` - standard output (screen) (c.f. `stdout` in C)
  - `cerr` - standard error (c.f. `stderr` in C)


Introduction to `iostream`

- The `<<` (put to) operator directs output to a stream
- The `>>` (get from) operator retrieves data from a stream
- A stream is a sequence of characters


`cout`

- The following displays Hello World on the screen:

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

- Flow of information from right to left
- `endl = "\n" =` new line
- Multiple `<<` operators can be concatenated


`cout`

- The following displays Hello World on the screen:

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

- Flow of information from right to left
- `endl = "\n" =` new line
- Multiple `<<` operators can be concatenated

In [3]:

```
#include <iostream>

int main(){
    std::cout << "Hello, world\n" << " helloworld" << 3 << std::endl;
}
```

```
Hello, world
 helloworld3
```


A simple C++ program

- Note: in the `#include`, although file is `iostream.h` we drop the `.h` here
- New Scoping rule: `using namespace std;`
    - Global variables/functions can be in spaces. Normally you'd do: `std::function()`


A simple C++ program

- Note: in the `#include`, although file is `iostream.h` we drop the `.h` here
- New Scoping rule: `using namespace std;`
    - Global variables/functions can be in spaces. Normally you'd do: `std::function()`

In [12]:

```
#include <iostream> // Note no '.h'

using namespace std; // <-- includes functions in namespace

int main(){
    std::cout << "Hello, world explicit namespace" << std::endl;
    // or (exploiting line 2)
    cout << "Hello, world using namespace" << endl;
}
```

```
Hello, world explicit namespace
Hello, world using namespace
```


`cout`

- The operator `<<` knows how to display values
- no need for a string to specify the data types
- For C we would have had to say:

```
printf("%s\n", "Hello, world");
```

- i.e. `<<` is an overloaded operator (like `+`)
  - We can define this for each object type we create - later
- Output can be formatted, if required


`cin`

```
cin >> name;
```

- Multiple `>>` operators can be concatenated

  ```
  cin >> num1 >> num2;
  ```

- Flow of information is from left to right
- `>>` operator can be overloaded for new object types


All together

All together

In [6]:

```cpp
#include <iostream>

using namespace std;

int main(){
    int num1, num2;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2; // This line doesn't run inside Jupyter(i.e. non-sense comes in)
    cout << "The sum is " << num1 + num2 << endl;
}
```

```
Enter two numbers: The sum is 141004419
```


File handling - Writing

File handling - Writing

In [45]:

```cpp
#include<iostream>
#include<fstream>

using namespace std;

int main() {
    ofstream outputFileStream;
    outputFileStream.open("example.txt");
    if (!outputFileStream) {
        cout<<" Error while creating the file ";
    }
    else {
        cout<<"File created and data got written to file";
        outputFileStream<< "Hello World" << endl;
        outputFileStream.close();
    }
    return 0;
}

```

```
File created and data got written to file
```

File handling - Reading


File handling - Reading

In [46]:

```cpp
#include<iostream>
#include <fstream>

using namespace std;

int main() {
    ifstream inputFileStream;
    inputFileStream.open("example.txt");
    if (!inputFileStream) {
        cout<<"File doesn't exist.";
    }
    else {
        char x;
        while (1) {
            inputFileStream>>x;
            if(inputFileStream.eof())
                break;
            cout<<x;
        }
    }
    inputFileStream.close();
    return 0;
}
```

```
HelloWorld
```

File handling - Reading


File handling - Reading

In [47]:

```cpp
#include<iostream>
#include <fstream>

using namespace std;

int main() {
    ifstream inputFileStream;
    inputFileStream.open("example.txt");

    std::string line;

    while (std::getline(inputFileStream, line))
    {
        cout << line << endl;
    }
}
```

```
Hello World
```

C++

Dynamic memory management


The `new` and `delete` operators

- Recall: Dynamic memory management allows the programmer to control when memory is allocated and deallocated.
    - The memory is allocated on the heap, while local variables are allocated on the stack
- C uses `malloc()` and `calloc()` for dynamic memory allocation and `free()` to deallocate it
- In C++ this has been simplified:
    - `new` creates space - no need to specify size
    - `delete` frees up the space


In C:

In C:

In [8]:

```c
#include <cstdlib>
#import <stdio.h>

int main(){
    int *p = (int*)malloc(sizeof(int));
    *p = 100 ;
    printf("Value of *p is %d\n", *p);
    free(p);
    return 0;
}
```

```
Value of *p is 100
```


In C++:

In C++:

In [13]:

```cpp
#include <iostream> // Note no '.h'
using namespace std;

int main(){
    int *p = new int;
    *p = 100;
    cout << "Value of *p is " << *p << endl;
    delete p;
    return 0;
}
```

```
Value of *p is 100
```


The `new` and `delete` operators

- `new` requests memory. If sufficient memory is available, `new` returns the address of the newly allocated and initialized memory to a pointer
  - `data-type *pointer = new data-type;`
  - `data-type` can be a built-in data type (such as `int`) or a custom data type you have created, such as a `class`
- If there is insufficient memory available on the heap an exception `std::bad_alloc` will be thrown.
  - you can suppress this with `new(nothrow)` if you want to continue even if the memory is not available
  - in this case `new` returns a `NULL` pointer


The `new` and `delete` operators

- `new` can also be used to allocate a set amount of memory
  - `pointer = new data-type[size];`
- Example:
  - `int *p = new int[10]`
  - allocates memory for 10 `int`s and returns a pointer to the location of the first int
  - to get to the second `*p++` or `p[1]` can be used (recall from C component)


The `new` and `delete` operators

- `new` can also be used to allocate a set amount of memory
  - `pointer = new data-type[size];`
- Example:
  - `int *p = new int[10]`
  - allocates memory for 10 `int`s and returns a pointer to the location of the first int
  - to get to the second `*p++` or `p[1]` can be used (recall from C component)

In [32]:

```
#include <iostream> // Note no '.h'
using namespace std;

int main(){
    int *p = new int[10]{};
    p[0] = 100;
    cout << "Value of p[0] is " << p[0]  << endl;
    cout << "Value of p[1] is " << p[1]  << endl;
    cout << "Value of p[2] is " << p[2]  << endl;
    delete [] p;
    return 0;
}
```

```
Value of p[0] is 100
```

The `new` and `delete` operators

- `delete` deallocates the memory
- `delete pointer` will deallocate memory that has been allocated with `new`
- do not mix this with C syntax! E.g. do NOT use `malloc` and `delete` together (or `new` and `free`)
- `delete[] pointer` may be used to free an entire array


Arrays in C++

In C++ there are different ways to initalise arrays:

```
int* x = new int[3];      // [garbage, garbage, garbage]
int* x = new int[3]();    // [0, 0, 0]
int* x = new int[3]{};    // [0, 0, 0] (Modern C++)
int* x = new int[3]{1,2,3};   // [1, 2, 3] (Modern C++)
```


Arrays in C++

In C++ there are different ways to initalise arrays:

```
int* x = new int[3];      // [garbage, garbage, garbage]
int* x = new int[3]();    // [0, 0, 0]
int* x = new int[3]{};    // [0, 0, 0] (Modern C++)
int* x = new int[3]{1,2,3};   // [1, 2, 3] (Modern C++)
```

In [34]:

```
#include <iostream> // Note no '.h'
using namespace std;

int main(){
    int *p = new int[3]{1,2,3};
    cout << "Value of p[0] is " << p[0]  << endl;
    cout << "Value of p[1] is " << p[1]  << endl;
    cout << "Value of p[2] is " << p[2]  << endl;
    delete [] p;
    return 0;
}
```

```
Value of p[0] is 1
Value of p[1] is 2
Value of p[2] is 3
```


Summary

Today we covered

- Brief history of C++
- C++ input/output
- Memory management

Next time

- Classes!