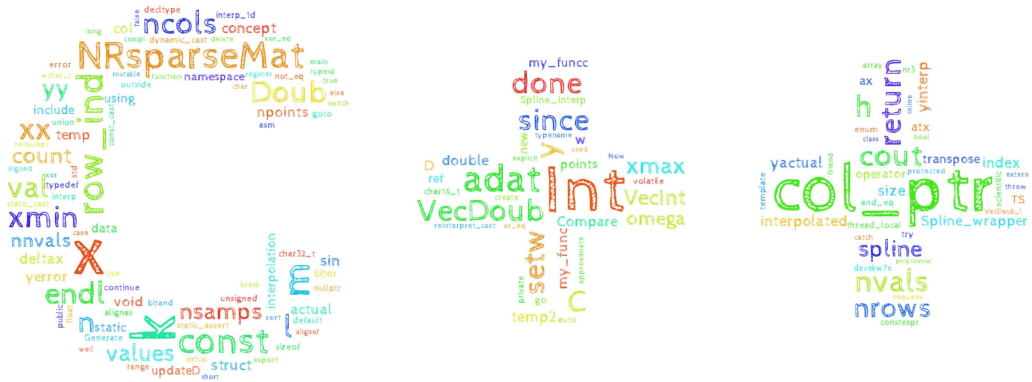
TECHNICAL COMPUTING WITH C++

FEEG6003 - Advanced Computational Methods II



Jan Kamenik & Shriram Sunder

Centre for Doctoral Training in Next Generation Computational Modelling

OVERVIEW

1. Introduction
2. C++ Object Orientied Programming
3. Productivity Tricks
4. Sunder
5. Tutorial
6. Appendix
7. Reference

INTRODUCTION

What is C++?

1. A better C
2. Supports data abstraction
3. Supports object-oriented programming
4. Supports generic programming
5. Bias towards systems programming
6. Maintains most of C’s syntax

HELLO WORLD

An easy C++ syntax example

Print "Hello World" to the console

```
1 #include <stdio.h>
2 #include <iostream>
3 // Another comment
4 int main(void)
5 {
6   printf("Hello World\n");
7   /* main() need not contain an explicit
       return statement; defaults to 0 */
8 }
```

HELLO WORLD

Even easier

Use the `auto` keyword in C++11 (dynamically typed)

- Compile with the `-std=c++11` flag

```
1 #include <iostream>
2 #include <stdio.h>
3 int main(void)
4 {
5   auto x = "Hello world!";
6   printf(x);
7 }
```

C++ FOR SCIENTIFIC COMPUTING

Pros:

- Speed (on a par with C and Fortran)
- Wealth of numerical libraries
- Wide-range of open source and commercial tools
- Flexible memory management
- Object-oriented language

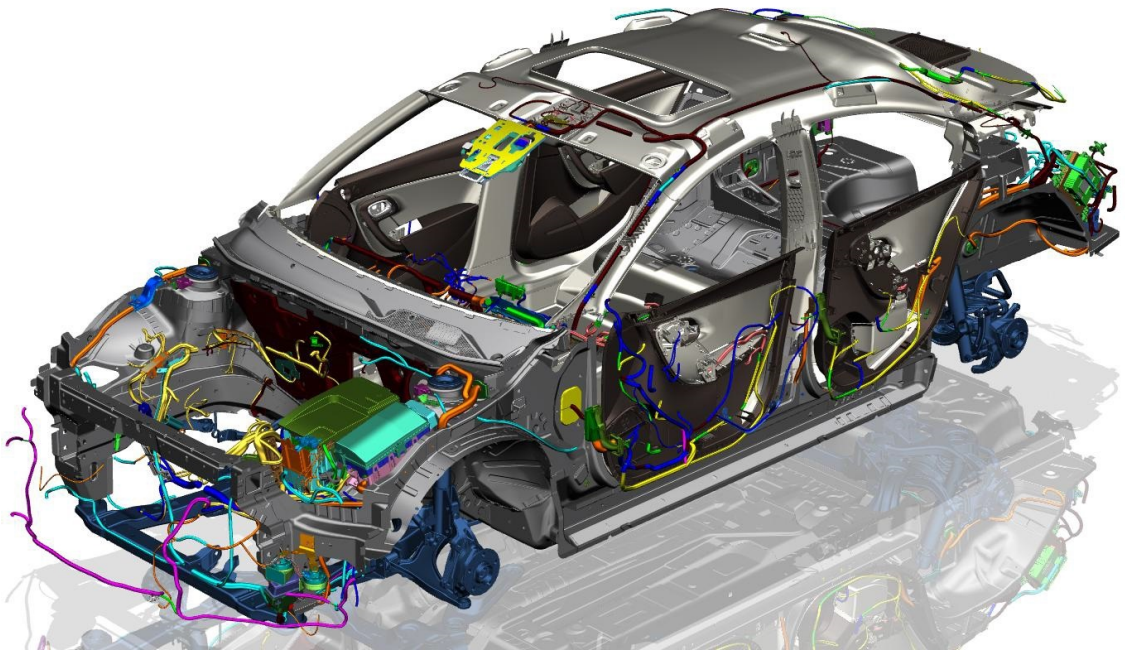
Cons:

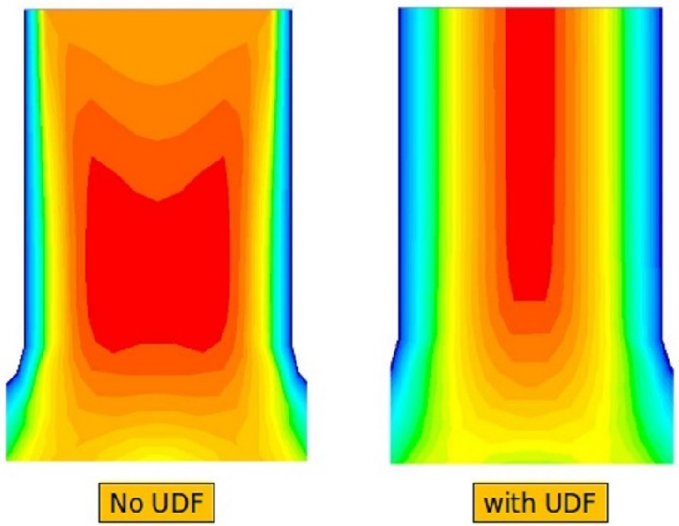
- Other languages might be better for a specialised task
- Clumsy for simple prototype programs and for plotting data

OTHER REASONS TO USE C++

C++ is very widely used

- C++ often pops up in APIs, for example:
- User defined functions (UDFs) in ANSYS are written in C/C++
- CAD: Siemens NX Application Programming Interface





C++ OBJECT ORIENTIED PROGRAMMING

Principles of Object Orientied Programming (OOP):

- **Encapsulation**
  - Combine variables (= members) with functions (= methods)
- **Inheritance**
  - Let an object acquire properties of another object
- **Polymorphism**
  - The same code can be used for a variety of objects
- **Modularity**
  - Split up program not only in functions but in higher-level modules
- **Abstraction**
  - Details “under the hood” of a class are unimportant
- **Extensibility**
  - Functionality can be reused with certain extensions

Encapsulation

- Members and methods are both contained in the **`class`** data structure
  - Access to classes is controlled by 3 “access specifiers”
    - `public`
    - `private`
    - `protected`
- In order to create an **object**, an instance of a class must be declared, similar to a variable declaration: `int num;`
  - `className objectName;`
- Classes (1) allow members and methods to be **encapsulated**, (2) allow **access specifiers** to be set and (3) can form **hierarchical class structures** through inheritance

OBJECT ORIENTIED PROGRAMMING

Class Syntax

```
class ClassName
{
  access specifier:
    member1;
    member2;
  access specifier:
    member3;
    member4;
};
```

Class Example

```
class Dog
{
  private:  // default
    int age, weight;
    string color;
  public:
    void bark();
    // etc...
} Pluto; // object
```

Dot notation

Call methods with the dot notation (just like in Java, C# or Python)

```
1 pluto.bark();
```

The “Dog” Class

Setter methods assign data and getter methods retrieve data

```
1  class Dog {
2    int age, weight ;
3    string color ;
4    public:
5      void bark() {cout << "WOOF!" << endl;}
6      void setAge(int yrs) {age = yrs;}
7      void setWeight(int lbs) {weight = lbs;}
8      void setColor(string hue) {color = hue;}
9      int getAge() {return age;}
10     int getWeight() {return weight;}
11     string getColor() {return color;} };
```

Combining Methods

- Methods longer than 2 lines are often defined outside the class
  - Declare a prototype method inside the class
  - Define the method using the `ClassName::MethodName` syntax

```
1 void setValues(int, int, string) ; //
    Prototype
```

```
1 void Dog::setValues(int age, int weight,
    string color) { // Definition
2 this -> age = age ;
3 this -> weight = weight ;
4 this -> color = color ;}
```

Scope resolution operator

`this ->` class pointer

Another Way to Initialize Class Members

- Use constructor and destructor methods (instead of the `setValues` function) to initialize class members
  - Both methods must have the same name as the class
    - For example `Dog::Dog`
  - The destructor method is prefixed with a ~

In the class definition define both prototypes:

```
1 // constructor prototype
2 Dog( int, int, string ) ;
3 // destructor prototype
4 ~Dog() ;
```

See the next page...

... and define both methods in the main program:

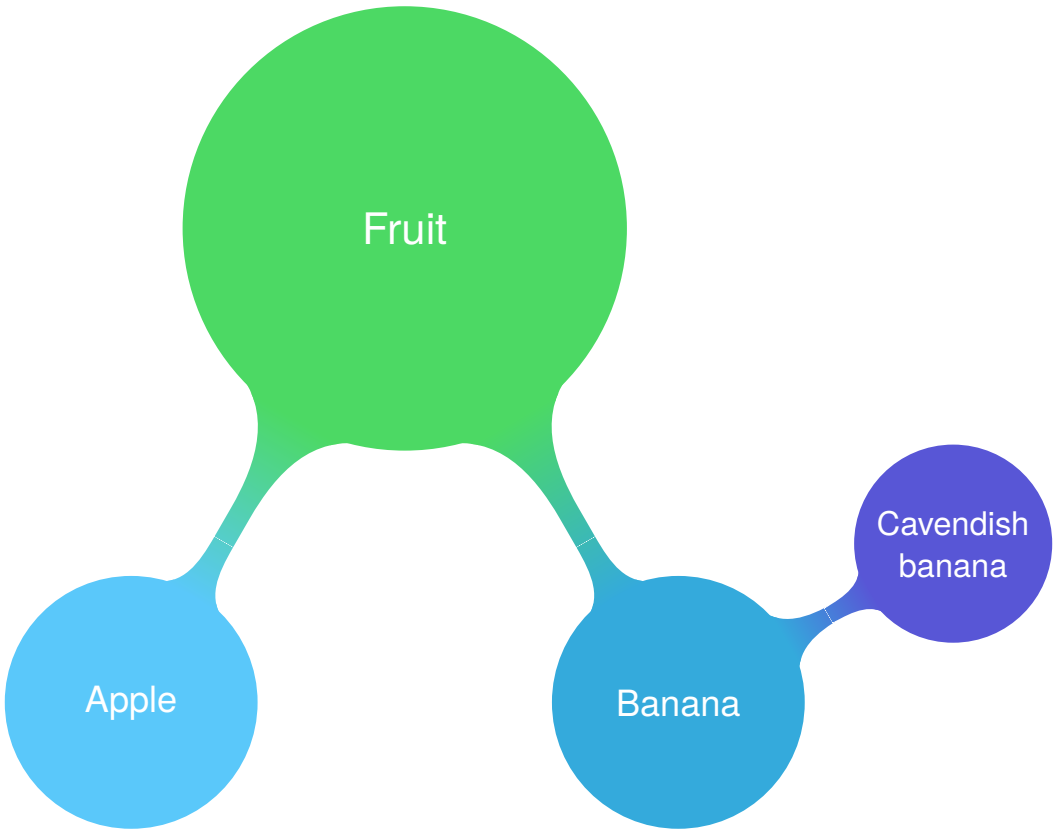
```
1 // constructor method
2 Dog::Dog(int age, int weight, string color)
3 {
4   this -> age = age;
5   this -> weight = weight;
6   this -> color = color;
7 }
```

```
1 // destructor method
2 Dog::~Dog()
3 {
4   cout << "Object destroyed." << endl;
5 }
```

You might have seen the “__init__” constructor in Python already!

Inheritance

- Classes can be derived from existing classes



Inheritance

- Derived classes **inherit** members of the parent class
- Syntax:
  - `class ParentClass : AccessSpecifier ChildClass`
- Elegant way to produce reusable code

Inheriting Class Properties

The rectangle class, which is derived from the polygon class, inherits width and height properties and adds a unique method

```
1  class Polygon {
2    protected:
3      int width, height;
4    public:
5      void setValues(int w, int h) {width = w;
           height = h;} };
6
7  class Rectangle: public Polygon {
8    public:
9      int area() {return(width*height);} };
```

Polymorphism

- C++ class methods can be polymorphic
- Polymorphism is closely related to inheritance
- Derived functions may have the same name but different implementations
- Useful because it allows using instances of different classes without worrying about details under the hood

Polymorphism

Here, the `print` method is polymorphic

```
1  int main()
2  {
3    circle A = circle(0.1, 0.2, 0.3);
4    A.print();
5    square B = square(0.9, 1.2, 5.3);
6    B.print();
7    return 0;
8  }
```

There is more, e.g. the `virtual` keyword, etc.

THE BIG PICTURE

So, what's the point of OOP?

- OOP is useful especially for large projects because they can be broken down into more manageable chunks
- OOP might improve the design, implementation, and maintenance of large scientific codes
- Data encapsulation and templates ensure robust code
- Inheritance and dynamic binding ensure reusable code
- Examples:
  - Frameworks are becoming more popular in software engineering and scientific computing
    - POOMA (Parallel Object Oriented Methods and Applications)
    - POET (Parallel Object oriented Environment and Toolkit)

PRODUCTIVITY TRICKS

The MATLAB Engine

- The MATLAB engine can be used to visualize C++ results
- Boilerplate code to embed this is as follows

```
1 // matsess is the MATLAB session name
2 Engine *matsess;
3 if (!(matsess = engOpen(""))) {
4   fprintf(stderr,"\nCan't start
5     MATLAB engine\n");
6   return EXIT_FAILURE;
7 }
```

MATLAB IN C++ FOR VISUALIZATION

Include this header in your code:

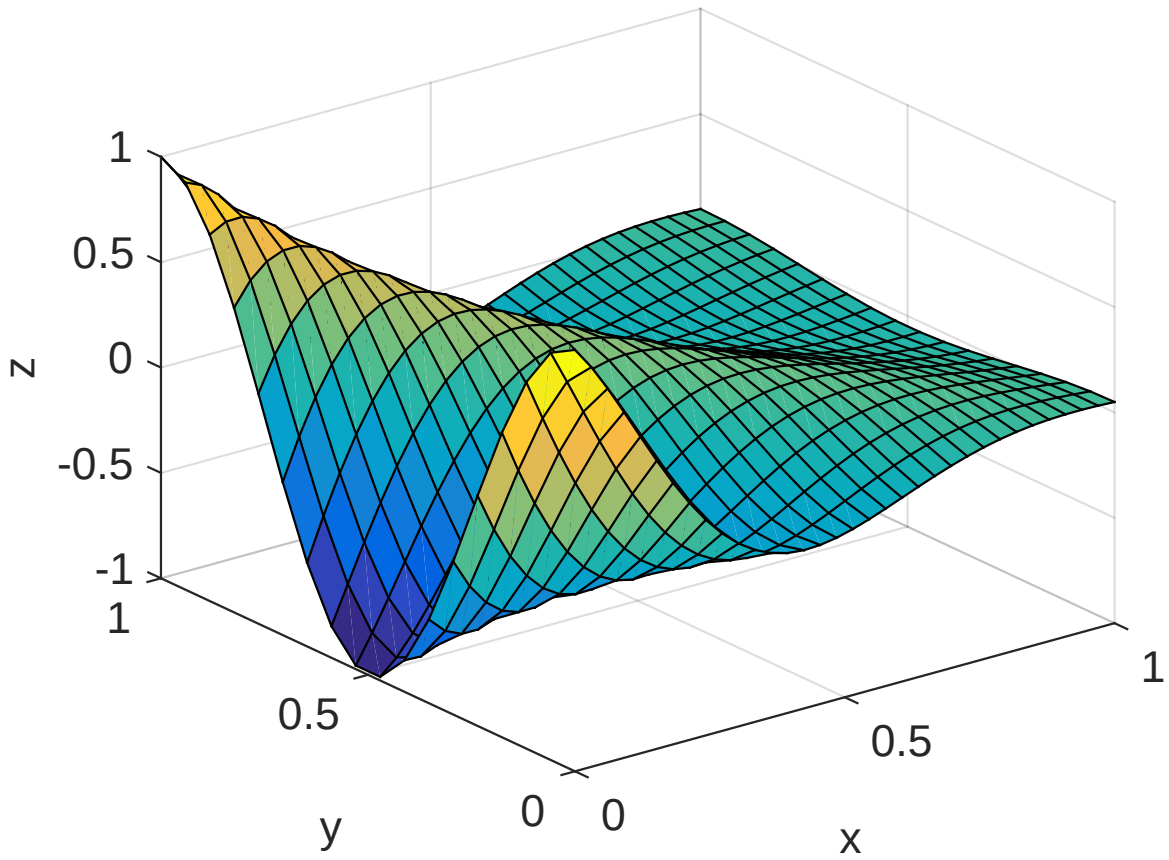
```
1 #include "engine.h"
```

- Windows:
  - Register MATLAB as a COM server via `!matlab -regserver`
  - Compilation via `mex -v -client engine filename.cpp`
- UNIX (64-bit Mac OS X):
  - You need to specify environment variables
    - `export PATH=/Applications/MATLAB_R2015a.app/bin:$PATH`
    - `export DYLD_LIBRARY_PATH=/Applications/MATLAB_R2015a.app/bin/maci64/:$DYLD_LIBRARY_PATH`
  - Compilation via makefile where flags for the linker need to be specified
    - -I to add the directory of `engine.h`
    - -L to add the directory for any link libraries[1]

[1] More information available at http://www.gnu.org/software/make/manual/

MATLAB output example from `mplot3d.cpp`

Basic MATLAB plotting functionality is available



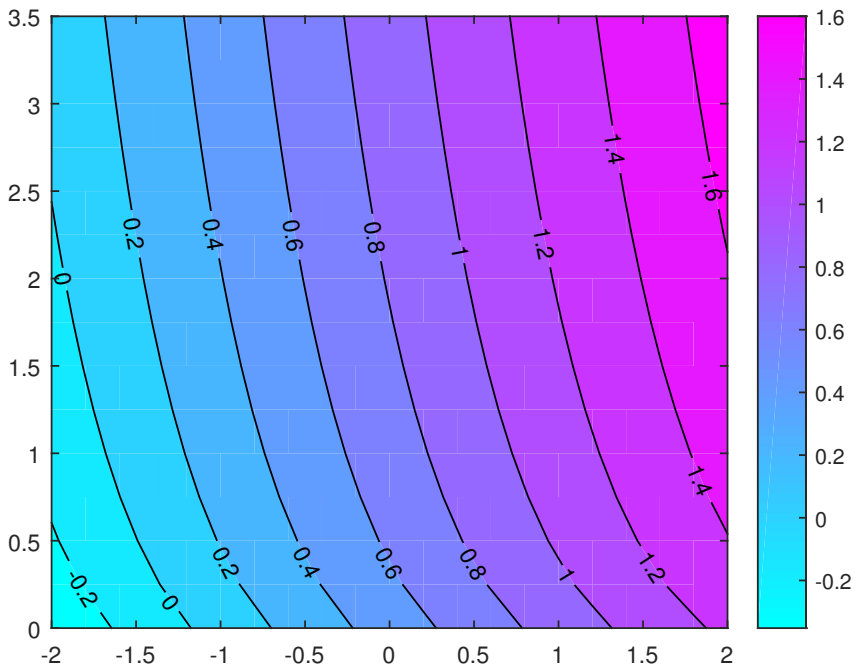
C++ IN MATLAB VIA MEX-FILES

MEX-files

- A MEX-files is the interface between MATLAB and the C/C++/Fortran program
- MEX-files are dynamically-linked subroutines that the MATLAB interpreter loads and executes
- A MEX-files contains only one function

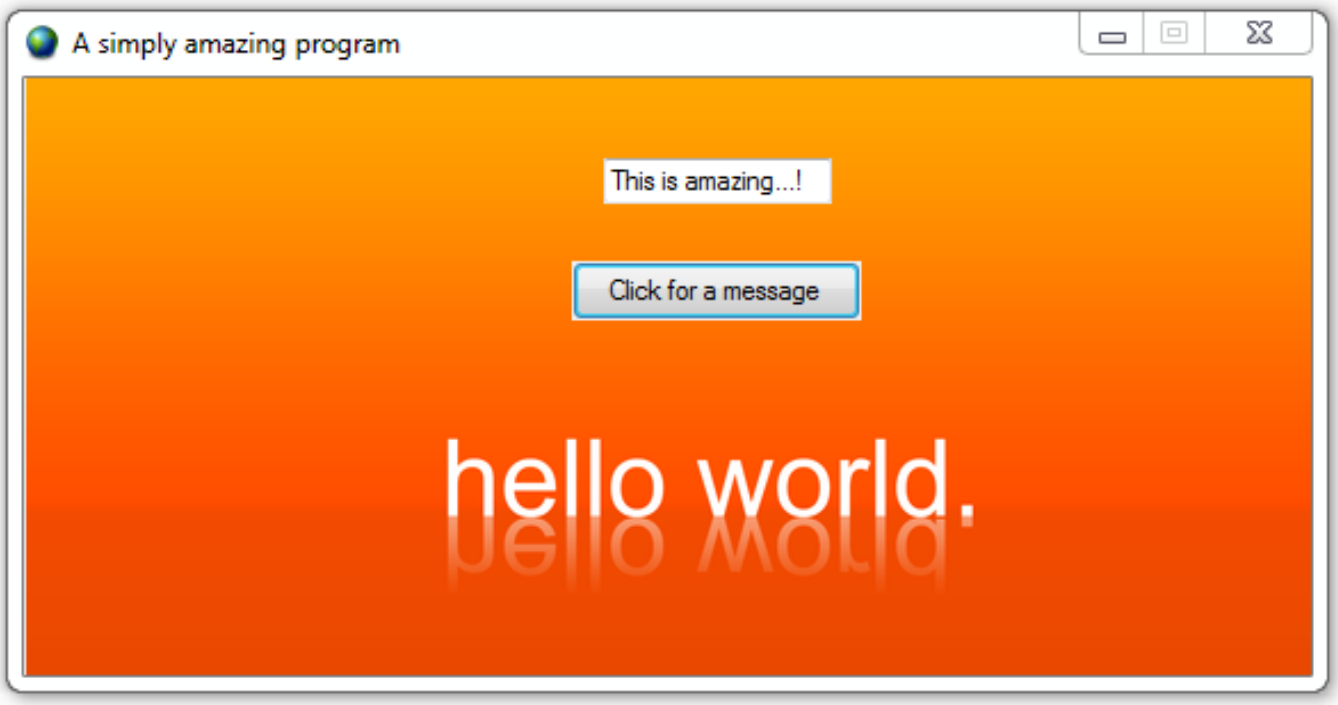
MEX-file example

- Fermi–Dirac integral
- Code from Numerical Recipes C++ code collection



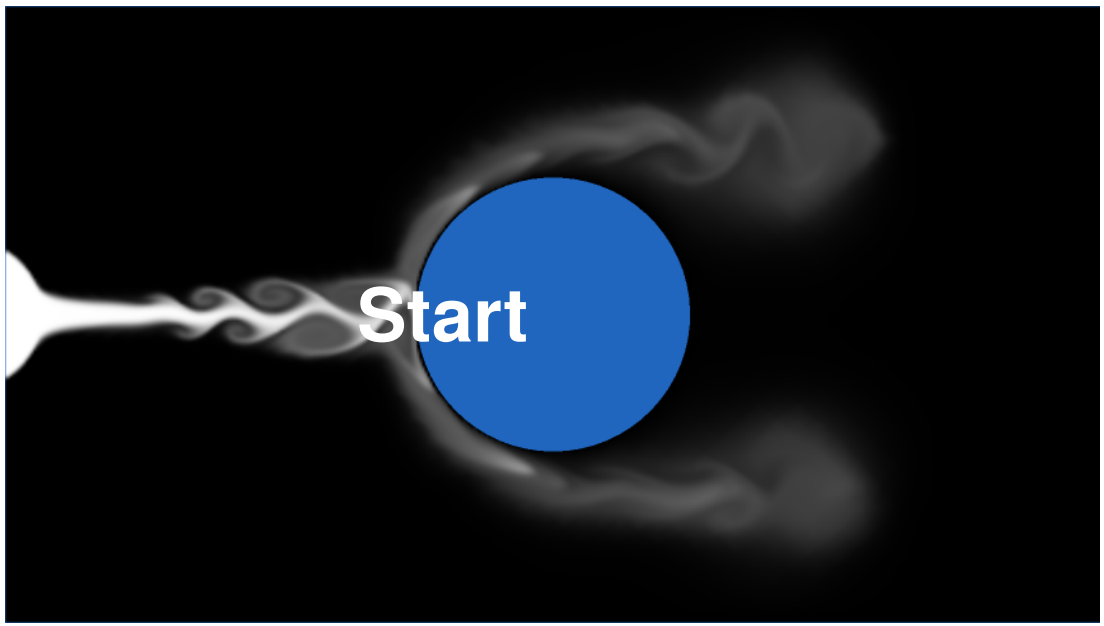
Microsoft Visual C++

Microsoft Visual C++ can be used to develop Windows apps



Microsoft Visual C++

A more advanced example involving fluid flow and OpenGL



Other languages

- C++ can easily be mixed with C and Fortran
- For Python, there are Boost.Python, Cython, Py++ and SWIG

Boost.Python wrapper example:

```
1 #include <boost/python.hpp>
2 BOOST_PYTHON_MODULE(hello_ext)
3 {
4   using namespace boost::python;
5   def("greet", greet);
6 }
```

In Python, you can then `import hello_ext` and
`print hello_ext.greet()`

CLANG++ COMPILER

- Open-source compiler for the C family of programming languages
- Available for Windows, MAC OS X and Linux
- More useful error and warning messages (cf. GCC)
- Clang Static Analyzer automatically finds bugs
- Faster, uses less memory than GCC, etc.[2]

```
hello.cpp:12:3: error: use of undeclared identifier 'coutt'
  coutt << "Hello World!" << endl  ;
  ^
hello.cpp:13:11: error: expected ';' after return statement
  return 0
          ^
          ;
```

[2]More details at http://clang.llvm.org/comparison.html

CALLING C FROM C++

How to call a C function?

Just use "extern "C""

```
1  extern "C" void f(int); // one way
2
3  extern "C" { // another way
4  int g(double);
5  double h();
6  };
7
8  void code(int i, double d)  {
9  f(i);
10 int ii = g(d);
11 double dd = h(); // ....
12 }
```

NEW FEATURES IN C++11

Tuples:

```
1 auto triple = std::make_tuple(5, 6, 7);
```

Range-Based For Loops:

```
1 for (int x : myList)
2   std::cout << x;
```

Variadic templates (variable number of arguments for functions):

```
1 template <typename... T> auto foo(T&&... args
    ) {
2 return std::make_tuple(args...);
3 }
4 ...
5 auto triple = foo(5, 6, 7);
```

NEW FEATURES IN C++14

Generic lambda functions:

```
1 auto lambda = [](auto x, auto y) {return x +
    y;};
```

Return type deduction:

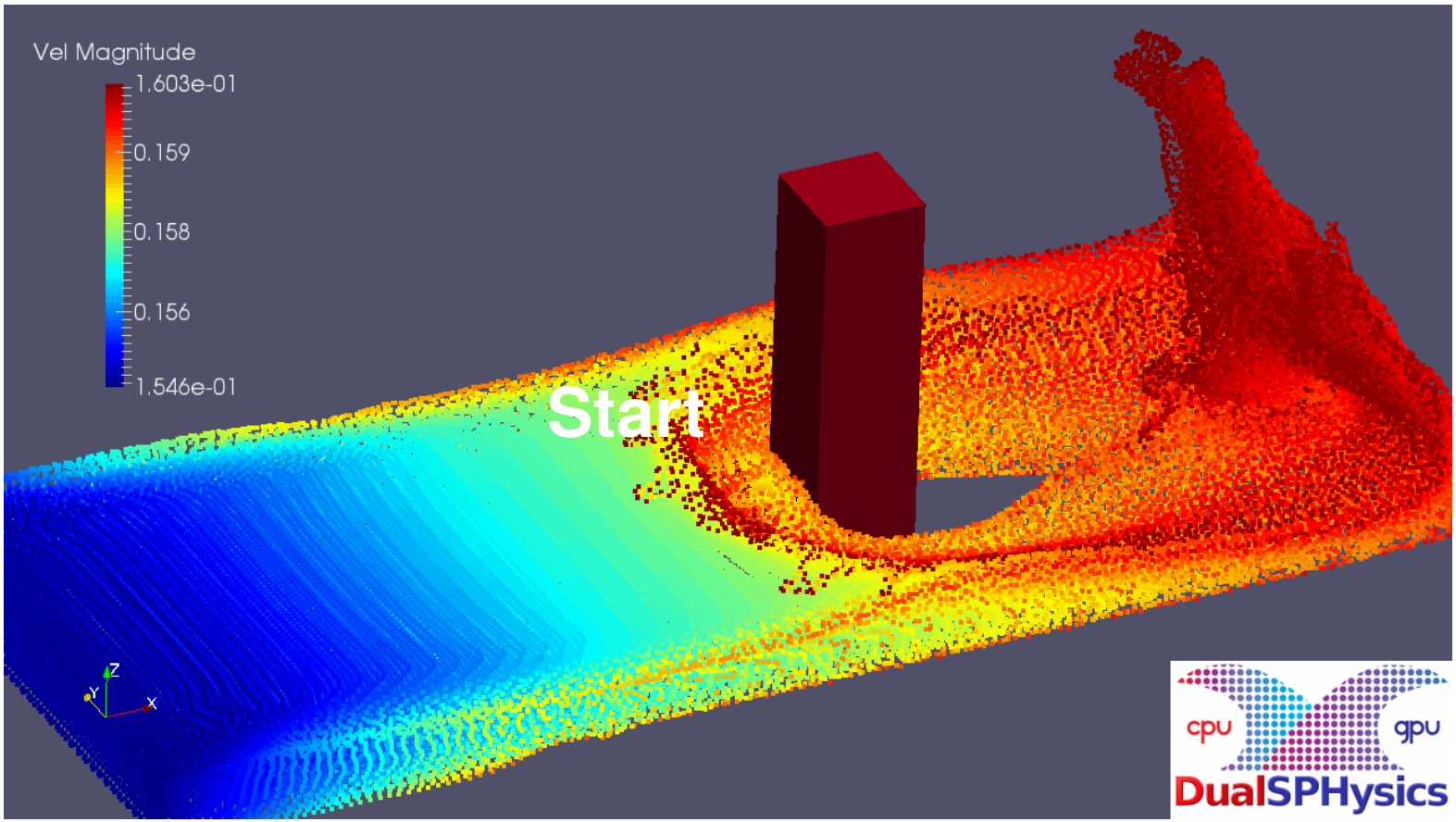
```
1 auto square(int n)
2 {
3 return n * n;
4 }
```

Clang supports all features!

EXISTING C++ CODES

Hundreds of amazing C++ projects

SPH via DualSPHysics written in C++ (and CUDA for GPUs)



SUNDER

C++ templates, meta programming, libraries, e.g. the STL, GSL, blitz++, show some examples, show what the differences are between C and C++ and what to look out for when compiling C with the C++ compiler

Technical libraries: OpenCV Boost Qt OpenCL Boost.Compute CUDA OpenMP Armadillo Eigen ViennaCL LAPACK Dlib Image processing

https://github.com/fffaraz/awesome-cpp


Ceemple

TUTORIAL

ADVANCED: READ .TXT AND DISPLAY NUMBER OF LETTERS A-Z

```
#include <map>
#include <iostream>
using namespace std;
int main()
{
map<char, int> freqs;
char ch;
while (cin .get(ch))
freqs[ch]++;
int i;
map<char,int>::iterator it;
for (i=1, it = freqs.begin(); it != freqs.end(); ++it,++i)
{
switch (it->first)
{
case '\r': cout << "\\r"; break;
case '\t': cout << "\\t"; break;
case '\n': cout << "\\n"; break;
case ' ' : cout << "Space"; break;
default: cout << it->first;
}
cout << "\t" << it->second << ((i%4) ? "\t" : "\n");
}
}
```

Example: http://www.entish.org/realquickcpp/classesetal.html

Because inheritance is the most important OOP feature

Example: http://www.entish.org/realquickcpp/memorymgt.html

THEME OPTIONS

This theme comes with some options to change it's appearance.

Option	Description
`nosectionpages`	Section pages will be supressed.

PRESENTATION STRUCTURE

A section page will be generated and the section name included in the presentation header for each section of the presentation with the current section being emphasized. If you include subsections in your presentation, then a small block will appear under the section name in the header for each frame. Once a frame has been viewed it will turn green.

It's worth noting that a frame can make up multiple slides.

```
\section{Main Section}
\subsection{Main Subsection}
\begin{frame}
\frametitle{Presentation Stucture}
% Frame Contents Here
\end{frame}
```

TABLE OF CONTENTS

Include a listing of the presentation's sections

```
\maketitle
```

For those longer presentations - keep the table of contents compact.

```
\begin{frame}{Overview}
    \tableofcontents[hideallsubsections]
\end{frame}
```

QUOTATIONS

At any time you can highlight text by using the `\alert` definition:

- This is super important!

The sthlm presentation has both `\lstinline` and `\lstinline!` definitions for quoting text.

>This text has been quoted<

»This text has been double quoted«

MULTIPLE COLUMNS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

- Point 1
- Point 2

Questions?

You can also send us an email at:

- Jan: jk4g13@soton.ac.uk
- Sunder: ss6g11@soton.ac.uk

REFERENCES


Bjarne Stroustrup
A Tour of C++
Addison–Wesley, 2013


Mike McGrath
C++ Programming in Easy Steps (4th ed.)
In Easy Steps Limited, 2011


Standard C++ Library reference
`http://www.cplusplus.com/reference/`
2015


GNU Scientific Library (GSL) for C and C++
`http://www.gnu.org/software/gsl/`
2015

APPENDIX

VARIABLES (1)

- Variable types
  - `char`
    - Used for characters (a, b, c, etc.)
  - `int`
    - Used for integers (-5, 100, 15, etc.)
  - `float`
    - Used for low precision floats (3.141593)
  - `double`
    - Used for higher precision floats (3.141592653589793)
  - `bool`
    - Takes values `true` or `false`
- Declaring
  - C syntax requires you to declare variables before they can be used (int adsf, double asdf2, etc.)
  - Declarations go at the top of the main function

VARIABLES (2)

- Using `#define`
  - Constants can also be defined using the syntax `#define CONSTANT VALUE`
  - Ex. `#define PI 3.14159`
  - Ex. `#define GOLD 1.61803`
  - This can also be used to define simple functions (macros).

VARIABLES (3)

- You can also define arrays, which are analogous to vectors and matrices.
- Remember: array indexing starts at 0
- Examples: Declaring arrays
  1. `int A[3] = {51, 5, 3};`
  2. `float B[] = {51.675, 5.0, 3.0};`
  3. `double C[5];`
  4. `int D[2][2] = { {1, 2}, {3, 4} };`
- Examples: Referencing arrays
  1. `thisVar = A[2];` would set `thisVar` to 3
  2. `otherVar = D[0][1];` would set `otherVar` to 2

FUNCTIONS

- Functions allow you to easily reuse chunks of code
- Consists of two parts:
  - A **function prototype**, that comes before `main`
  - A **function definition**, that comes after `main`
- Example:
  - Prototype:

```
int polynomial(int i);
```

  - Definition:

```
int polynomial(int i) {
  int f, i2;
  i2=i*i;
  f = -1*i2 + 4*i - 3;
  return f;
}
```

SIMPLE MATH

- You can use basic math without special effort:
  - Addition: +
  - Subtraction: -
  - Multiplication: *
  - Division: /
  - Modulo: %
- For more complicated math, `#include <math.h>`
- This will give you access to functions like `atan`, `atan2`, `pow`, `log10`, `erf`.
- Full list here: `http://www.cplusplus.com/reference/cmath/`

LIBRARIES

- `iostream`
  - Enables basic input and output functions.
- `fstream`
  - Class for reading *and* writing to files.
- `iomanip`
  - Provides additional i/o functionality (specifically `endl`)
- `math.h`
  - Provides access to a large number of mathematical functions.
- `time.h`
  - Generally required for use of `iostream`

Pointers

* is the dereference operator meaning “value pointed to by”

```
1 int num[] = {1, 2, 3};
2 int * ptr;    // pointer definition
3 ptr = num;
4 cout << "Memory address: " << ptr << endl;
5 cout << "Value pointed to: " << *ptr << endl;
```

CONTROL STRUCTURES (1): IF

if Syntax

```
if ( CONDITION1 ) {
  doThis;
}
else if ( CONDITION2 )
{
  doSomethingElse;
}
else if ( CONDITION3 )
{
  doAnotherThing;
}
else ( CONDITION4 ) {
  ifNothingElseDoThis;
}
```

if Example

```
int D = 1;
if(D==2) {
  D = D+5;
}
else
  D = D+2;
}
```

CONTROL STRUCTURES (2): WHILE AND DOWHILE

while Syntax

```
while ( CONDITION1 ) {
  doThis;
}
```

do while Syntax

```
do {
  allThisStuff;
  andThisToo;
} while ( CONDITION1
);
```

while Example

```
float s = 2.5;
while (s < 100) {
  s = 2*s;
}
```

CONTROL STRUCTURES (3): FOR

for Syntax

```
for ( INIT; CONDITION; INCREMENT ) {
  pleaseDoThisSeveralTimes;
}
```

for Example

```
int i;
for(i=1; i<10; i++) {
  printf("Hello!\n");
}
```

CONTROL STRUCTURES (4): SWITCH

switch Syntax

```
switch ( VARIABLE ) {
  case VALUE1:
    doThis;
    break;
  case VALUE2:
    sorryDoThisInstead ;
    break;
  default:
    butReallyDoThisJK;
    break;
}
```

switch Example

```
int a = 10;
int b = 10;
int c = 20;
switch ( a ) {
  case b:
    a = a + 1;
    break;
  case c:
    a = a + 2;
    break;
  default:
    a = a/2;
    break;
}
```

REFERENCE

This section can be deleted when we are completely finished with

PRIMARY PRESENTATION COLORS

The following ios7 inspired colors structure the sthlm theme.

sthlmLightRed
sthlmGreen
sthlmDarkBlue
sthlmDarkGrey
sthlmLightGrey

SECONDARY PRESENTATION COLORS

sthlmRed

sthlmYellow

sthlmLightYellow

sthlmLightBlue

sthlmBlue

sthlmPurple

sthlmGrey

BLOCKS

The default Beamer Box

Block Title Here

- point 1
- point 2

```
\begin{block}{Block Title Here}
\begin{itemize}
\item point 1
\item point 2
\end{itemize}
\end{block}
```

ADDITIONAL BLOCKS

Alert Block

Highlight important information.

```
\begin{alertblock}{Alert Block}
Highlight important information.
\end{alertblock}
```

ADDITIONAL BLOCKS

Example Block

Examples can be good.

```
1       \begin{exampleblock}{Example Block}
2       Examples can be good
3       \end{exampleblock}
```

CUSTOM BLOCKS

Purple customization

Using the theme colors to generate colored blocks.

```
\begingroup
\setbeamercolor{block title}{bg=
   sthlmPurple}
\setbeamercolor{block body}{bg=
   sthlmLightGrey}
\begin{block}{Custom Blocks}
Using the theme colors to generate
   colored blocks.
\end{block}
\endgroup
```