# Introduction to C++

Callum Beddow
calum.beddow12@imperial.ac.uk

Imperial College Software Society

October 13, 2014

# Buy Software Soc.'s Free Membership

at

https://www.imperialcollegeunion.org/shop/club-society-project-products/software-products/

This is to keep the union happy. ^_^

# Contents I

0. Introduction
- What is C++?
- Why C++?
- What You Will Need
- The Compiler
- Compiling and Running

1. A Tutorial Introduction
- Getting Started
- Variables and Arithmetic Expressions
- For Loops
- If Statements
- Constants
- Input and Output
- Arrays
- Functions
- Passing Arguments by Reference

# Contents II

External Variables and Scope

2. Pointers and Arrays
   - Pointers and Addresses
   - Pointers and Function Arguments
   - Address Arithmetic
   - Pointers and Arrays
   - Dynamic Memory Allocation
   - Pointer Arrays: Pointers to Pointers
   - Multidimensional Arrays
   - Command Line Arguments
   - Function Pointers
   - Void Pointers

3. Structures
   - Basics of Structures
   - Operator Overloading
   - Self-referential Structures

# Contents III

4. Header Files & Libraries
   - Header Files
   - Conditional Inclusion
   - Compiling Multiple Files

# 0. Introduction

# 0.1 What is C++?

- C++ is a multi-purpose programming language
- C++ is *imperative*
  You define a sequence of commands that are run one after the next, allowing you to change the state of the computer.
- C++ is based on C, which was based on B, which was based on BCPL
- The latest C++ standard was released in 2011 (C++11)

# 0.2 Why C++?

- C++ is:
  - General purpose
  - Fast
  - Portable
  - Widely used
- It has many libraries and pre-made functions that allow you to perform operations without having to write them yourself. C++ is (nearly entirely) backwards compatible with C allowing you to use C libraries as well.

# 0.3 What You Will Need

- A computer or Laptop.
- A text editor to write the code in (e.g. ▸ Notepad++, ▸ Xcode)
- A compiler, to turn what you write into machine code. ▸ g++

## For Windows:

For the text editor ▸ Notepad++, and the g++ compiler can be installed from ▸ MinGW, which can be downloaded ▸ here. When installing tick the base, g++ and msys boxes. Make a shortcut to the `C:\MinGW\msys\1.0\msys.bat` (MinGW Shell) script. Run the bat script then run the

```
echo 'export PATH=/c/MinGW/bin:$PATH'>> .profile
```

command, then close and open the console again.

## For Mac OS X:

Xcode comes with both g++ and a text editor. Xcode can be downloaded from the App Store, then install 'Command Line Tools' from 'Preferences\Downloads'.

# 0.4 The Compiler

The compiler turns what you write, human readable code, into binary computer code that is run. There are 3 main steps to compiling:

- Preprocessing
  Preprocessor commands all start with a #. They are mainly used to include files from libraries and replace text.
- Translation into object code
  This turns your code into machine code, but also stores the names of functions you need from other files needed.
- Linking
  This gets the functions you need from other files and inserts them into the object code, and makes the file a proper executable.

The translation and linking is the *proper* compiling as it makes the machine code.

# 0.5 Compiling and Running

Using the MinGW Shell (Windows), or terminal (Mac):

- `touch filename.cpp`
  `touch` makes a new file, if it does not exist. If the file does exist it updates the timestamp.
- `start filename.cpp` *or* `open filename.cpp`
  `start` (on Windows) or `open` (on Mac) will open the file for you to write the code in.
- `g++ filename.cpp`
  This command compiles the code and creates the executable file `a.exe` (Windows) or `a.out` (Mac / Linux). `g++` is the name of the compiler.
- `./a.exe` *or* `./a.out`
  This will run the program that was just compiled to `a.exe` (Windows) or `a.out` (Mac / Linux).

# Tips

- Command History

  You can go through your previous commands, by using the up & down arrow keys.
- Autocompletion

  File names and directory names can be completed using the tab key. This saves you from typing out long filenames, e.g. say you wanted to compile the file `verylongfilename.cpp` you could type `g++ ver` then press the tab key. If the file/directory name is ambiguous it will complete up to that point, then list the files.
- Chaining Commands Together

  Commands can be chained together using `&&`. If the first command completes successfully (i.e. returns 0), then the next command will be run. If you do not care whether it was successful, you can use `;` instead. For example, say you want to compile the file `hi.cpp` then run it, you can use `g++ hi.cpp && ./a.out`. That way, if the compilation fails, the program will not be run.

# Other Useful Commands I

- `ls`
  `ls` lists the stuff in the directory you are currently in.
- `cd directoryname`
  `cd` changes directory, moving you into the directoryname directory. To go up a directory use `../` as the directory.
- `g++ -Wall -Wextra codetocompile.cpp`
  The `-Wall` & `-Wextra` turns on all warnings (for things that are bad) and extra warnings (for things the compiler thinks you may make a mistake doing it that way). These are useful for helping you debug your code.
- `g++ codetocompile.cpp -o outputfilename` The `-o` flag for g++ allows you to change the output file name from the default (`a.out`) to outputfilename.

# Other Useful Commands II

- `pwd`
  `pwd` tells you your present working directory, i.e. were you currently are in the filesystem.
- `./a.out > filename`
  `>` can be used to write the output of your program (what would be written to the console using `cout` in code) to a file.

# 1. A Tutorial Introduction

# 1.1 Getting Started

As is the tradition in programming we will start with the "Hello World" program, that prints "Hello World" to the Screen.

```
1  // Hello World Program
2
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8    cout << "Hello World" << endl;
9    return 0;
10 }
```

So what does it all mean and why do we need it?

▸ HelloWorld.cpp

- `//Hello World Program`
  This is a single line comment, and does nothing. Comments are used to remind you what the code does.
- `#include <iostream>`
  This tells the compiler that we will be using things from the input output stream standard library, namely `cout` ans `endl`.
- `using namespace std;`
  In C++ things are grouped together in namespaces, this tells the compiler to look for things in the `std` namespace, otherwise we must tell it to look for them there, so `cout` without this line would be `std::cout` and `endl` would be `std::endl`.

- `int main(){return 0;}`
  This is the main function. All C++ programs have a main function so the computer knows where to start in the program, in case you had many functions. The `int` is the return type, that is an integer that the program gives back to whatever ran this code. The `return 0;`, is when it gives back the integer and leaves the main function. Returning `0` traditionally denotes that the program finished successfully, and other integers are error codes.

## The Most Important Part of the Code

The lines above need to be there (except the comment line), but understanding them isn't necessary at this point. They can just be copy and pasted in, knowing that you only have to change what's between the `{` and the `return 0;`

- `cout << "Hello World"<< endl;`
  `cout` is the standard output stream, i.e. it writes on the screen. You feed in what you want written with <<. In this case `"Hello World"`, then it is written to the screen. After that a new line is fed in with << `endl`. So when the program finishes, the terminal starts on the line after "Hello World".

# Semicolons Everywhere;

## Remember the Semicolon;

Semicolons denote the end of a statement or instruction in C++, not a newline. You can, if you want, put all your C++ onto one line, so how does the compiler know where one instruction ends and the next starts? Semicolons ;

# More couting

```cpp
#include <iostream>
using namespace std;

int main()
{
  cout << "Hello " // no ; so is continued to the next line
       << "World"
       << endl << endl; // outputs: Hello World

  cout << 42 << endl; // outputs: 42
  cout << 4 << 2 << endl; // outputs: 42 (no spaces)
  cout << 4 << " " << 2 << endl << endl; // outputs: 4 2 (with
      space)

  cout << 4+2 << " " << 4*2 << endl; // outputs: 6 8

  cout << 4/2 << ", " << 2/4 << ", " << 2.0/4.0 << endl;
  // outputs: 2, 0, 0.5
  return 0;
}
```

▸ Morecouting.cpp

# 1.2 Variables and Arithmetic Expressions

## Variables

Variables are used to store values, and can be changed. They must have a *type* and a *name*. The type defines what is being stored, and the name is what you use to access it.

| Type | Description | Examples |
|---|---|---|
| `bool` | A boolean value (true of false) | `true`, `false` |
| `char` | A character | `'a'`, `'@'`, `'7'` |
| `int` | An integer | `0`, `10`, `-100` |
| `double` | A double precision floating point number | `0.1`, `1e-10`, `7.0` |

## Variable Declaration

- `Type Name;`
- `Type Name = Value;`

# Arithmetic Operators

C++ can do arithmetic:

| Syntax | Name | |
|---|---|---|
| `a = b` | Assignment Operator | `int x = 3` |
| `a + b` | Addition | `x + 7` |
| `a - b` | Subtraction | `5 - 4` |
| `a * b` | Multiplication | `6*3` |
| `a / b` | Division | `2/3`— `2.0/3.0` |
| `a % b` | Modulo (integer remainder) | `5 % 3` |
| `++a` | Prefix increment | `++x` |
| `a++` | Postfix increment | `x++` |
| `--a` | Prefix decrement | `--x` |
| `a--` | Postfix decrement | `x--` |

The prefix inc/decrement inc/decrements the variable by 1 before the rest of the operations, postfix does it after the rest of the operations.If `x=3`, `x++*2` is `6` and `x` is `4`, whereas if `x=3` , `++x*2 == 8` and `x` is `4`.

# Things to Watch Out for

## Watch for Integer Division `/`

If two integers are divided, the result will be the solution rounded down, i.e. `3/2` is `1`, `2/3` is `0`.

## `int`'s store Integers

Integers only store integers, other numbers are *rounded down* before being stored, i.e. $2.2 \rightarrow 2$, $2.5 \rightarrow 2$, and even $2.999 \rightarrow 2$.

## `=` is not `==`

`=` (is set to) Sets a variable to something.
`==` (is equal to?) Compares two things to see if they are the same.

# Writing a Fahrenheit to Kelvin Converter

To produce a table of values for Farhrenheit in Kelvin

```
1  #include <iostream>
2  using namespace std;
3
4  /* Kelvin - Fahrenheit table
5    for fahr = 0, 15, ..., 120*/
6  int main () {
7    int min = 0, max = 120, step = 15;
8    int fahr = min;
9
10   cout << "F \t K \n";
11   while (fahr <= max){
12     int kel = 5 * (fahr - 32) / 9.0 +
           273;
13     cout << fahr << "\t" <<kel<< endl;
14     fahr = fahr + step;
15   }
16   return 0;
17 }
```

Conversion Equation:

$$\text{kel} = \frac{5 \times (\text{fahr} - 32)}{9} + 273$$

Program's Output:

| F | K |
|---|---|
| 0 | 255 |
| 15 | 263 |
| 30 | 271 |
| 45 | 280 |
| 60 | 288 |
| 75 | 296 |
| 90 | 305 |
| 105 | 313 |
| 120 | 321 |

▸ FahrtoKel.cpp

## While Loops

`while` can be used to repeat a block of code multiple times, when a specified condition is true. The syntax for a `while` loop is

```
while (condition){
//lines of Code to run while condition is true
}
```

The `{ }` is used for a single or multiple lines of code, and does not need a `;`. Variables declared in `{ }`, such as `int kel;` are only accessible within the `{ }` .

## Rest of the Code

- `/*Some text */`
  A multiline comment can be constructed by putting `/*` and `*/` around what you want to comment out. It should be noted that the comment goes from the `/*` to the first `*/`, so you cannot put multiline comments in multiline comments.
- The characters `'\n'` and `'\t'`
  These are escape characters and represent characters cannot be typed in the source code. In this case they are newline character (`'\n'`), and the tab character (`'\t'`) for producing lined up columns.

# Fahrenheit to Kelvin Converter, with Doubles

```
1  #include <iostream>
2  using namespace std;
3
4  /* Kelvin - Fahrenheit table
5    for fahr = 0, 15, ..., 120*/
6  int main () {
7    double min = 0, max = 120, step = 15;
8    double fahr = min;
9
10   cout << "F \t K \n";
11   while (fahr <= max){
12     double kel = 5 * (fahr - 32) / 9.0
           + 273;
13     cout << fahr << "\t" <<kel<< endl;
14     fahr = fahr + step;
15   }
16   return 0;
17 }
```

Program's Output:

| F | K |
|---|---|
| 0 | 255.222 |
| 15 | 263.556 |
| 30 | 271.889 |
| 45 | 280.222 |
| 60 | 288.556 |
| 75 | 296.889 |
| 90 | 305.222 |
| 105 | 313.556 |
| 120 | 321.889 |

Did you notice the rounding down in the previous version?

▸ FahrtoKeldouble.cpp

# 1.3 For Loops

For loops are like while loops, but are a bit more condensed.

```
1 #include <iostream>
2 using namespace std;
3 int main () {
4   cout << "F \t K \n";
5   for (double fahr=0; fahr <= 120; fahr = fahr +15)
6     cout << fahr << "\t" << 5 * (fahr - 32) / 9 + 274 << endl;
7   return 0;
8 }
```

```
for (Step before loop;
     Condition;
     Step at end of loop cycle) {
... code here repeats while condition is true ...
}
```

▸ forFahrtoKel.cpp

The `for` loop

```
for (Step before loop; Condition; Step at end of loop cycle) {
... code here repeats while condition is true ...
}
```

is equivalent to

```
Step before loop;
while (Condition) {
... code here repeats while condition is true ...
Step at end of loop cycle;
}
```

There is no difference between the two, except that the `for` loop is more compact.

# Comparison Operators

The comparison operators in C++ are:

| Syntax | Operator Name |
|---|---|
| `a == b` | is equal to |
| `a != b` | is not equal to |
| `a > b` | is greater than |
| `a < b` | is less than |
| `a >= b` | is greater than or equal to |
| `a <= b` | is less than or equal to |

a!=b and a=!b

The first is *is a not equal to b?*, the second is *set a to not b*.

# 1.4 if Statements

If-Then-Else statements allow you to branch your code down two possible routes, depending on whether a condition is `true` or `false`

```
if (condition) {
    // Code to be run if the condition is true
} else {
    // Code to be run if the condition is false
}
```

If-Then statements are also possible, where you do not need to specify what happens if the condition is `false`

```
if (condition) {
    // Code to be run if the condition is true
}
```

Print random numbers and whether they are even or odd until the sum of the numbers will be $> 10000$

```
1  #include <iostream>
2  #include <cstdlib> // for rand();
3  #include <ctime> // for time();
4  using namespace std;
5
6  int main(){
7
8    // Seeding the random number, to prevents the same output each
         time
9    srand(time(NULL));
10
11   int sum = 0;
12
13   while(true){
14     int num = rand() %1000;
15
16     sum = sum + num;
17     if (sum > 10000){
```

rnd.cpp

```
18        break; // Leaves a while/for loop, you are currently in (
              line 13)
19      }
20
21      cout << num << "\t";
22      if (num %2 == 0){
23        cout << "even";
24      } else {
25        cout << "odd";
26      }
27      cout << endl;
28    }
29
30    return 0;
31  }
```

# Logical Operators

The logical operators in C++, where `A` and `B` are logical expressions (i.e. `2 == 1`), which is evaluated to false or true, are :

| Syntax | Operator Name | |
|---|---|---|
| `!A` | not A | (false → true— true → false) |
| `A && B` | A and B | (true iff A and B are true) |
| `A`— B— | A or B | (true if either A or B (or both) are true) |
| `A != B` | A xor B | (true iff only one of A or B are true) |
| `!(A && B )` | A nand B | (true iff A and B not both true) |
| `!(A`— B)— | A nor B | (true iff A and B are false) |
| `!(A != B)` | A xnor B | (true iff A and B are false or true) |

*iff* means if and only if.

# 1.5 Constants - Const

It is bad practice to bury *magic numbers* in your code. It is better to put them at the top of the file, give them clear name, and make them `const` to prevent them from being changed. The syntax for this is:

```
const type NAME
```

This makes the converter:

```
1  #include <iostream>
2  const int MIN = 0, MAX = 120,
3            STEP = 15;
4
5  using namespace std;
6  int main() {
7          cout << "F \t K \n";
8          for (double fahr=MIN;fahr <= MAX; fahr = fahr + STEP)
9                  cout << fahr << "\t" << 5 * (fahr - 32) / 9 + 274
                       << endl;
10         return 0;
11 }
```

▸ constFahrtoKel.cpp

# 1.6 Input && Output

C++ has no built-in routines for handling IO, but there are some standard Libraries. Two libraries for Input and Output are:

- iostream - For IO to the Screen.
- fstream - For IO to Files.

These contain classes (used to 'create objects'):

- iostream automatically defines two streams for input and output: cout and cin.
- ifstream does not automatically define streams for you, so you must define the objects yourself using ofstream and ifstream class (classes are like types, as objects are like variables).

# Reading from Standard Input - cin (in iostream)

How far in the alphabet is a lowercase letter?

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    char letterin;
5    cout << "Please enter a lower case letter: ";
6    cin >> letterin;
7    cout << "\n It is the "
8      << letterin - 'a' + 1
9      << " letter in the alphabet\n";
10   return 0;
11 }
```

cin.cpp

- `#include <iostream>`
  `cin` is in the iostream library
- `char letterin;`
  Declare space to store the character to be read in.
- `cin >> letterin;`
  `cin` reads in one line from standard input. This is split by white spaces (`''`, `'\t'`, etc.) which can then be read into variables with >>. N.B. Beware the direction of the arrows: `cout` uses << and `cin` uses >>.
- `letterin - 'a'+ 1`
  letters are represented and stored as numbers, e.g. `'a'` is stored as 97 as ▸ ASCII. The lower case letters are stored contiguously so `'c'` is 99. When doing subtraction/addition the letters are converted to numbers so `'c'`−`'a'`= 2. The +1 is because we want the number in the alphabet not the distance from `'a'`.

# Writing to Files - ofstream (in fstream)

Writing a square grid of letters to a file becoming pseudo randomised.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ofstream fout("1.5.2.1output.txt");
  char lin; int iin, size = 5;
  cout << "Please enter a lower case letter: ";
  cin >> lin; lin = lin - 'a';
  cout << "Please enter an integer ";
  cin >> iin; iin = iin%26;
  for(int i=0; i < size; i++){
    for (int j=0; j < size; j++)
      fout << char('a'+ (lin +(i*size+j)*iin)%26) << '\t';
    fout << endl;
  } return 0;
}
```

letterrandomise.cpp

- `#include <fstream>`
  The ofstream class (type) is found in the file stream library.
- `ofstream fout("1.5.2.1output.txt");` This makes an ofstream object (variable) `fout` and makes the file `"1.5.2.1output.txt"` in the directory the program is run. It acts just like `cout`, except for writing to screen it writes to a file. You can create as many as you like with

  ```
  ofstream objectname (filename);
  ```

- `iin = iin%26;`
  This stores the remainder of $\mathrm{iin} \div 26$ in `iin`. This guarantees that `iin` is a number between and not excluding 0 and 25.

- `char('a'+ (lin +(i*size+j)*iin)%26)`
  This is *typecasting*. It converts one type to another, in this case from `int` to `char`. It can be done in two ways:

  `typeto(expression of original type)`

  or

  `(typeto) expresion of original type`

  Here, `char(a number)` is used here so that `fout` writes a letter to the file rather than a number, as output streams determine how to write something based on it's type.

# Reading from files - ifstream (in fstream)

Take the grid from the last program and find how far each letter is from 'a'.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ifstream fin("1.5.2.1output.txt");
  char c; int size = 5;
  for(int i = 0; i < size; i++)
  {
    for(int j=0; j < size; j++)
    {
      fin >> c ;
      cout << c-'a'<<'\t';
    }
    cout << endl;
  }
  return 0;
}
```

gridtonum.cpp

- `#include <fstream>`
  `ifstream` is in the file stream library.
- `ifstream fin("1.5.2.1output.txt");`
  This makes an ifstream object(variable) `fin` and opens the file `"1.5.2.1output.txt"` in the directory the program is run for reading. It acts like `cin`, except reads from a file. You can make ifstream objects for opening files by

  ```
  ifstream objectname (filename);
  ```

- `fin >> c ;`
  This reads in a line from the file and puts one character into `c` at a time.

# 1.7 Arrays

Arrays allow you to store a large number of values in a list so that they can be used later in the program, the array has $n$ spaces, but goes from 0 to $(n-1)$, where $n$ is a *const int*:

```
type arrayname[n]
```

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    const int i=5;
5    int evn[i]={2,4};// Declare the array and set the first two
         elements.
6    evn[2] = 6;// Set the 3rd (0,1,2) element.
7    for (int n=3;n<i;evn[n]=2+2*n++); // Fill the rest of the array
          with even numbers
8    for (int n=0; n<i; n++) // Print out all the elements
9      cout << evn[n] << endl;
10   return 0;
11 }
```

▸ evennums.cpp

# Arrays & the Vector Class

- Arrays are fixed length, they cannot change size.
- So how can you store more values as your program needs it?
  - Vectors are similar to array, but they can change in length, so extra values can be added or removed.
  - Vectors has other properties that the arrays do not, for example it knows it's own size.
  - Vectors are part of the standard template Library (STL).
  - `http://www.cplusplus.com/reference/stl/vector/`

## Making Vectors

Vectors are in the `vector` library and are declared by:

```
vector <type> vectorname(size)
```

Note the round brackets (), arrays have square brackets [].

# Vector Class Example: Even Numbers

```
 1 #include <iostream>
 2 #include <vector>
 3 using namespace std;
 4 int main() {
 5   int i=5; // Does not need to be const
 6   vector<int> evn(i); // Note the ( ) brackets
 7
 8   // Elements can be accessed just like arrays with [ ]
 9   for(int n=0; n<i; evn[n]=2+2*n++);
10
11   evn.push_back(12); // Adding extra values
12   evn.push_back(14);
13
14   for(int n=0; n < evn.size(); n++) // They know their own size
15     cout << evn[n] << endl; // the last element is evn.size()-1
16
17   return 0;
18 }
```

▸ evennumsvector.cpp

# 1.8 Functions

```
returnType functionName (type1 argName1, type2 argName2 /*, etc
    ...*/ )
{
    ... code here ...
    return (valueOfReturnType);
}
```

- Functions are like subroutines or procedures, they encapsulate bits of computation.
- They split a program up into different reusable parts.
- Functions are called from other parts of the program, with their name and parameter list.

```
functionName(arg1, arg2 /*, etc...*/)
```

- They *must* have a function prototype, or be declared before they are used.

```
returnType functionName (type1, type2 /*, etc... )
```

# Function Example - $x^n$

```
1  #include <iostream>
2  using namespace std;
3
4  int power(int, int); // Function Prototype
5
6  int main() {
7    const int b = 2;
8
9    for (int i = 0; i < 10; i++)
10     cout << power(b,i) << '\n'; // Calling the Function
11   return 0;
12 }
13
14 int power(int x, int n){ // Function Declaration.
15   int ret = 1;
16   for(int i=1; i <= n; i++)
17     ret = ret * x;
18   return ret;
19 }
```

power.cpp

## Things to Note:

- *Copies* of the arguments are passed into the function.
  In the previous example, changing `x` and `n` in the power function in no way effect `b` or `i` in the main function.
- This is called passing by value, as the value of the variable is copied into the function.
- The function ends when it gets to a `return`. Anything afterwards is ignored.
- There is a much better power function in the cmath library, called pow (see ▸ here).

## Functions can be Recursive

Functions can call themselves, this is known as recursion.

# Recursive Function Example - $x^n$

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int power(int, int); // Function Prototype
 5
 6 int main() {
 7   for (int i = 0; i < 10; i++)
 8     cout << power(2,i) << '\n'; // Calling the Function
 9   return 0;
10 }
11
12 int power(int x, int n){ // Function Declaration.
13   if(n>0){
14     return x * power (x, n-1); // Function Calls itself
15   } else {
16     return 1;
17   }
18 }
```

▸ powerrecursive.cpp

# 1.9 Passing Arguments by Reference

- Variables can be given directly to functions, such that they are not copies of the variables. This is done by putting the reference operator & after the type.

```
returnType functionName (type1& argName1, type2& argName2 /*, etc
    ...*/ )
{
    ... code here ..
    return (valueOfReturnType);
}
```

- This gives the address (location) of the variable in memory is given to the function.
- This is useful if you do not want copies made of that which is passed to it, for example if it will take a long time to copy.
- `const` can be used to prevent changes to the variable.

# Interchanging two Variable

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int&, int&);
5
6  int main(){
7    int a = 5, b = 7;
8    cout << "a = " << a << "\tb ="<< b << "\nswap\n";
9    swap(a,b);
10   cout << "a = " << a << "\tb ="<< b << endl;
11 }
12
13 void swap (int& x, int& y){
14   int z;
15   z = x; x = y; y = z;
16 }
```

There is a better version of swap in the algorithm library, which works for all types, not just `int`'s, see ▸ here.

▸ swap.cpp

# 1.10 External Variables and Scope

```
1  #include <iostream>
2  using namespace std;
3
4  int a=2, b;
5
6  int sum () {
7    return a+b;
8  }
9
10 int main(){
11   b=5;
12   cout << sum() <<endl;
13 }
```

- Variables declared outside of functions can be used anywhere lower in the code. They are global/external variables.
- Although global variables can exist, *it is bad practice to make or use them*.
- Variables declared between { } only exist between the { }. This is their scope.
- A variable declared between { } can have the same name as an existing variable, but will access a different, new portion of memory.
- This is known as variable shadowing, and is also bad practice.

▸ external.cpp

# 2. Pointers and Arrays

# 2.1 Pointers and Addresses

- A Pointer is a variable that stores the address of a variable.

A simplified view of memory is that it is consecutively numbered (addressed) memory cells (of 1 byte = 8 bits (1's and 0's)). Different types of variables take up different numbers of bytes (e.g. a char is 1 byte). The number of bytes taken up by the type can be found using the `sizeof` operator:

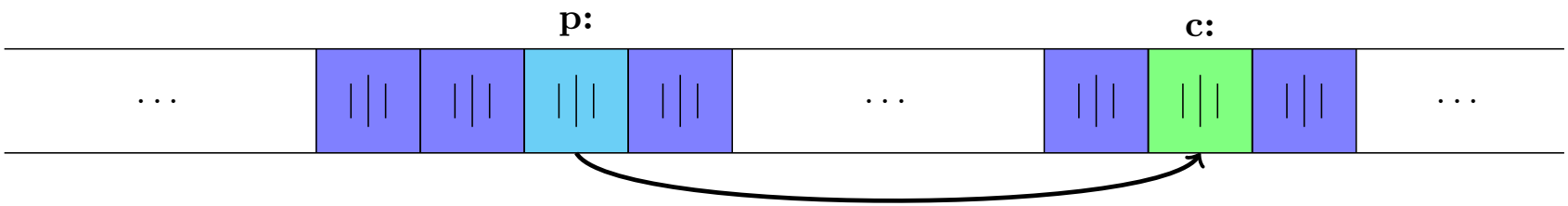
```
1 #include <iostream>
2 using namespace std;
3 int main(){
4   cout << sizeof(int)<< endl;
5 }
```

My laptop and compiler outputs 4, however the program may give different results on different computers or for different types.

▸ sizeof.cpp

# Pointers and Addresses cont.

A pointer is a group of cells (usually 2 or 4) that can hold the address of a variable. This could be represented by:




where 'p' is a pointer to 'c'. The operator '&' gives the address of an object, so to set the value of p to the address of c:

```
p = &c
```

. p is then said to *point to* c.

The & only applies to objects in memory, it cannot be applied to expressions, constants of register variables.

# Pointers and Addresses cont.

- The '*' operator is the indirection operator.
  It is used to give you access to the memory pointed to by a pointer.

## Declaring Pointers

```
int c = 5; // Declaring an integer
int *p1; //Declaring a pointer to an integer (p1)
p1 = &c; // pointing p1 to c
int *p2 = &c; // Declaring a pointer to an integer (p2) and
    pointing it to c
```

# Pointers and Addresses Example

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4    int a = 5, b = 7, c;
5    int *pa, *pb = &a;
6    pa = &b; // pa points to b
7    cout << pa << " contains: " << *pa << endl;
8    c = *pb; // c is set to the value pointed to by pb
9    std::cout << "c at " << &c << " contains: " << c << '\n';
10   *pb = 0;
11   std::cout << "pb at: " << &pb << " points to: " << pb
12     << " containing: " << *pb
13     << endl;
14   return 0;
15 }
```

```
0x7fff11232620 contains: 7
c at 0x7fff1123261c contains: 5
pb at: 0x7fff11232610 points to: 0x7fff11232624 containing: 0
```

▸ pointing.cpp

# 2.2 Pointers and Function Arguments

We saw how you could pass variables by reference to a function to change them. We can also do the same with pointers.

```
1  #include <iostream>
2  using namespace std;
3  void swap(int*, int*);
4  int main(){
5    int a = 5, b = 7;
6    cout << "a = " << a << "\tb = "<< b << "\nswap\n";
7    swap(&a,&b);
8    cout << "a = " << a << "\tb = "<< b << endl;
9  }
10 void swap (int *ipx, int *ipy){
11   int z;
12   z = *ipx; *ipx = *ipy; *ipy = z;
13 }
```

```
a = 5 b = 7
swap
a = 7 b = 5
```

pointswap.cpp

# 2.3 Address Arithmetic

- Pointers can be $+$ or $-$ with an int, but not multiplied or divided.

Adding an integer $n$ moves the pointer by the sizeof the pointed object $n$ times.

## Segmentation Fault

A *Seg Fault* occurs when a program tries to access memory that it shouldn't. This is usually caused by bad address arithmetic, or reading elements much further than the end of an array.

# A Bad Address Arithmetic Example

To show address arithmetic it, a pointer is made to point at a variable and

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5    char b[] = "Hello World", c = 'c';
6    char *ipc = &c;
7    while (true){
8      cout << *ipc++;
9    }
10 }
```

This program can produce different output when it is run.

badaddress.cpp

# Bad Address Arithmetic Example - Output

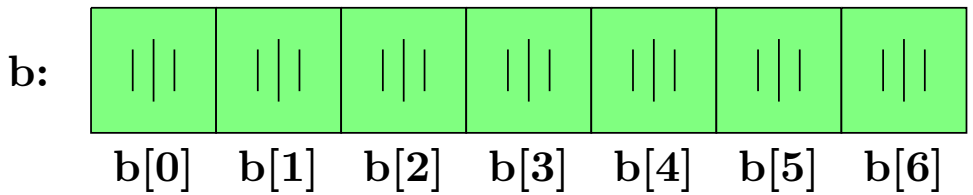
```
[twig@twignovo-arch code]$ g++ -Wall -Wextra badaddress.cpp && ./a.out
badaddress.cpp: In function 'int main()':
badaddress.cpp:5:7: warning: unused variable 'b' [-Wunused-variable]
cHello World°téÙÿ ÈséÙÿ %Dú ,téÙÿ @íûYA`@°téÙÿ íûüm j%íûã668º%@,téÙÿ `@°téÙÿ @"téÙÿ
                                                                                yéÙÿ yéÙÿ yéÙÿ ayéÙÿ yéÙÿ yé
Ùÿ »yéÙÿ ÙyéÙÿ õyéÙÿ zéÙÿ  zéÙÿ qzéÙÿ }zéÙÿ zéÙÿ zéÙÿ ¤zéÙÿ ¤zéÙÿ ÇzéÙÿ õzéÙÿ   {éÙÿ #{éÙÿ éÙÿ -}éÙÿ ^}éÙÿ r}éÙÿ
 }éÙÿ £}éÙÿ ³}éÙÿ È}éÙÿ Ð}éÙÿ ß}éÙÿ Ï}éÙÿ
                                          ~éÙÿ /~éÙÿ M~éÙÿ n~éÙÿ ~éÙÿ ¯~éÙÿ µ~éÙÿ á~éÙÿ  éÙÿ - éÙÿ : éÙÿ E éÙÿ d
 éÙÿ w éÙÿ  éÙÿ ² éÙÿ ô éÙÿ P éÙÿ !ÀñÙÿ ÿûë¿d@@þú       `@
                                                          è
ddwéÙÿ õ éÙÿ wéÙÿ .RPqùg¬_æd¤ýx86_64./a.outXDG_VTNR=1MANPATH=:/opt/devkitpro/devkitPPC/man:/opt/devkitpro/devkit
PPC/manDEVKITPPC=/opt/devkitpro/devkitPPCXDG_SESSION_ID=1XDG_DATA_HOME=/home/twig/.local/shareANDROID_HOME=/opt/
android-sdkANDROID_SWT=/usr/share/javaTERM=rxvt-unicode-256colorSHELL=/bin/bashXDG_SESSION_COOKIE=9dde461c8fa3a0
ccd5cacafc00000900-1352993673.432193-1966766905QT_XFT=trueDE=kdeWINDOWID=54525964QTDIR=/opt/qtUSER=twigDEVKITPRO
=/opt/devkitproMOZ_PLUGIN_PATH=/usr/lib/mozilla/pluginsXDG_CONFIG_DIRS=/etc/xdgMAIL=/var/spool/mail/twigPATH=/us
r/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/opt/android-sdk/platform-tools:/opt/android-sdk/tools
:/opt/cuda/bin:/opt/devkitpro/devkitPPC/bin:/usr/bin/vendor_perl:/usr/bin/core_perl:/opt/qt/bin:/usr/local/MATLA
B/R2011b/bin:/opt/android-sdk/tools:/opt/android-sdk/platform-tools:/opt/devkitpro/devkitPPC/bin:/home/twig/.gem
/ruby/1.9.1/bin:/usr/local/MATLAB/R2011b/bin:/opt/android-sdk/tools:/opt/android-sdk/platform-tools:/opt/devkitp
ro/devkitPPC/bin:/home/twig/.gem/ruby/1.9.1/binQT_IM_MODULE=ibusPWD=/home/twig/Documents/Dropbox/C++/Slides/code
XMODIFIERS=@im=ibusJAVA_HOME=/usr/lib/jvm/java-7-openjdkEDITOR=vimLANG=en_GB.utf8COLORFGBG=15;defaultSHLVL=5XDG_
SEAT=seat0HOME=/home/twigTERMINFO=/usr/share/terminfoXDG_CONFIG_HOME=/home/twig/.config_JAVA_AWT_WM_NONREPARENTI
NG=1_humbleintbundlekey=xxwVuDWqErMFXDG_CACHE_HOME=/home/twig/.cachePYTHONPATH=/usr/lib/rootLOGNAME=twigXDG_DATA
_DIRS=/usr/local/share/:/usr/share/J2SDKDIR=/usr/lib/jvm/java-7-openjdkPKG_CONFIG_PATH=:/opt/qt/lib/pkgconfigWIN
DOWPATH=1DISPLAY=:0XDG_RUNTIME_DIR=/run/user/1000GTK_IM_MODULE=ibusJ2REDIR=/usr/lib/jvm/java-7-openjdk/jreCOLORT
ERM=rxvt-xSegmentation fault
^[[?1;2c[twig@twignovo-arch code]$ 1;2c
```

Note: The Segmentation fault, and also that it prints "Hello World", and the program name "a.out".

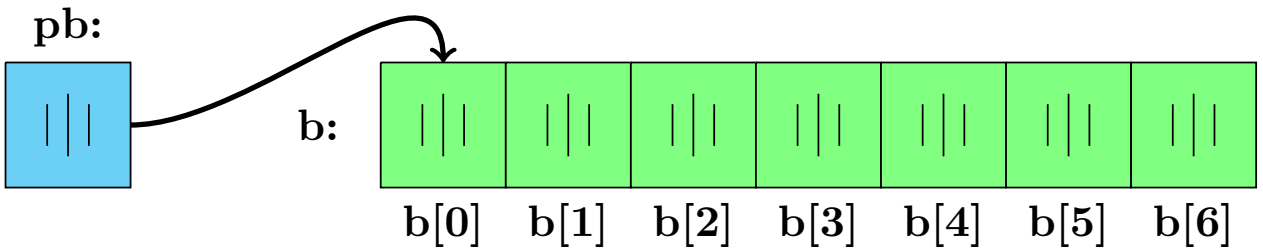
# 2.4 Pointers and Arrays

- There is a relationship between pointers and arrays (as can be seen in the last example).

An array declared `int b[7];` defines an integer array b of size 7:



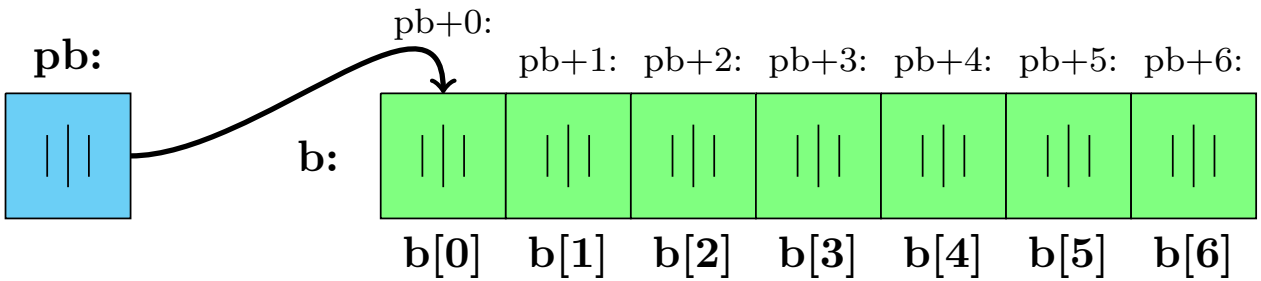

A pointer can be declared `int*pb = &b[0];` creating a pointer that points to the first element:




In fact thats what `b` is! Elements can be accessed by `b[i]` or `*(b+i)`, this is why arrays count form zero. `&b[0]` also works with vectors to give the address of the 1st element, whereas just the name does not.

# Moving Through Arrays - Example

Pointers can be incremented to move along arrays `i` blocks, Then the value can be read by `*(pb+i)`:




```
1 #include <iostream>
2 using namespace std;
3 int main(){
4   char b[] = "abcdefghijklmnopqrstuvwxyz";
5   char *ipc = &b[0];
6   for(int i = sizeof(b)-1; i>=0; i--)
7     cout << *(ipc+i);
8   cout << endl ;
9 }
```

This example prints the alphabet in reverse.

- The `sizeof()` operator knows the size of an arrays.
- The last character of a string is '\0', so b is 27 long.

pointarray.cpp

# 2.5 Dynamic Memory Allocation

- Some programs cannot know how much memory they need before they are run.
- Memory can be allocated with the `new` operator.
- `new` returns a *pointer* to the allocated memory.
- When it is finished using the memory it *must* be released with `delete`.

The `new` and `delete` operators come in two forms:

| Form | Allocate | Free |
|---|---|---|
| Single Variable: | new *type*; | delete *pointer* |
| Array: | new *type*[*n*]; | delete[] *pointer* |

## Do Not Mix the Forms

C++ does not keep track of whether it is a variable or array, you must do so.

# Calculating the Mean of some Random Numbers

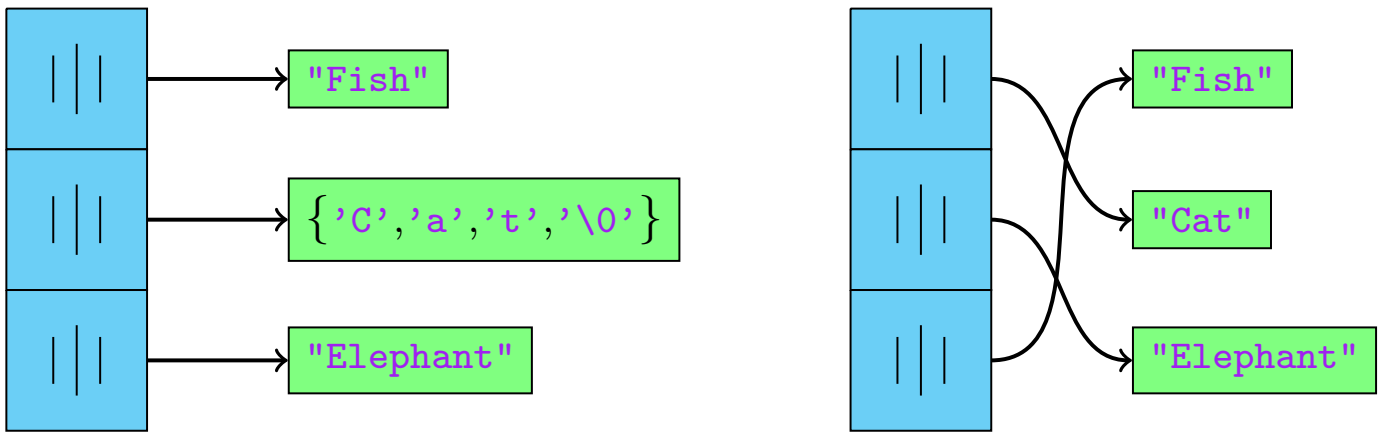
```
 1 #include <iostream>
 2 #include <cstdlib>
 3 using namespace std;
 4 int main(){
 5   int num;
 6   cout << "How many points? "; cin >> num;
 7   double *points = new double[num];
 8
 9   for (int i = 0; i != num; points[i++] = (double)rand() /
         RAND_MAX);
10   cout << "Points filled with random numbers from 0 - 1" << endl;
11
12   double sum = 0;
13   for (int j=0; j < num; j++) sum += points[j];
14
15   cout << "Mean of Points: " << sum/num << endl;
16
17   delete [] points;
18   return 0;
19 }
```

▸ Mean.cpp

- `#include <cstdlib>`
  cstdlib is included as the `rand()` function and largest possible value from that function `RAND_MAX` are in there.
- `double *points = new double[num];`
  This creates the pointer to a double `points` and points it at a new dynamically allocated array of `num` `int`'s.
- `(double)rand()/ RAND_MAX`
  `rand()` returns a random number between 0 and `RAND_MAX`. Typecasting it to a `double` prevents integer division, then dividing it by `RAND_MAX` gives a random number between 0 and 1
- `delete [] points;`
  This frees the array of dynamically allocated memory pointed to by points.

# 2.6 Pointer Arrays: Pointers to Pointers

- Since pointers are variables they can also be stored in arrays.
- This can then be used to sort that which is pointed to, by only changing the pointers in memory.




- This will therefore be much more efficient than actually sorting them if the memory pointed to is large.

C Strings are character arrays that are terminated by the `'\0'`. This is so functions know were the end of the string is and do not need to know its length.

# Sorting C Strings

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  int main(){
7    char *wordlist[6] = {"cat","dog","caterpillar","fish","catfish"
         };
8    wordlist[5] = "bat";
9
10   for(bool swapped=false; !swapped; swapped = !swapped)
11     for(int i = 1; i < 6; i++)
12       if (strcmp(wordlist[i-1], wordlist[i]) > 0){
13         swap(wordlist[i-1],wordlist[i]);
14         swapped = true;
15       }
16
17   for(int i = 0 ; i<6; i++) cout << wordlist[i] << endl;
18   return 0;
19 }
```

▸ stringswap.cpp

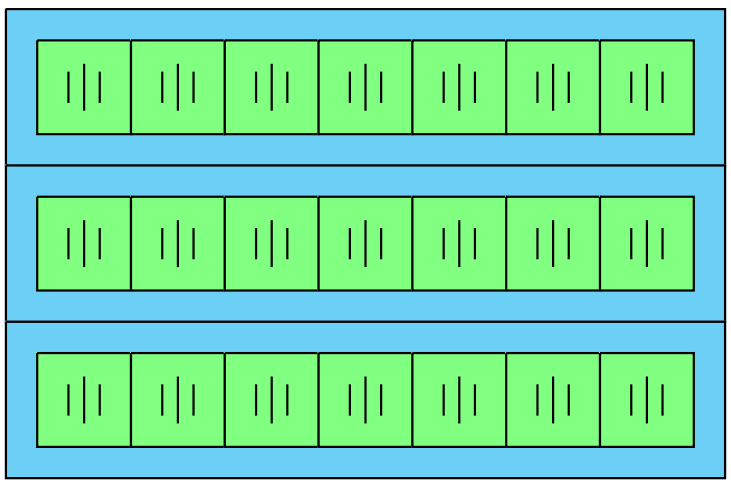
- `#include <algorithm>`
  The algorithm library contains a generalised swap function.
- `char *wordlist[6] = {"cat","dog","caterpillar" ...`
  Declaring an array of character pointers and setting the to some C strings.
- `strcmp(wordlist[i-1], wordlist[i])`
  This compares the previous word in the list with the current word. `strcmp` returns 0 if the strings are the same, a number $< 0$ if the first is less, and $> 0$ if the first is greater.
- `swap(wordlist[i-1],wordlist[i]);`
  The algorithm library swap function interchanges two pointers to pointers of characters.

# 2.7 Multidimensional Arrays

- C++ allows for rectangular multi-dimensional arrays.
- They are declared with the number of pairs of `[ ]` equal to the number of dimensions, e.g. 2D array:

```
Type name[n][m]
```

- C++ 2D arrays are really a 1D array whose elements are an array.



# 2.8 Command Line Arguments

Command line arguments can be used to supply programs with input on the command line. The main function gives you an `int` for the number of arguments and a `char**`, an array of C strings of the arguments.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char *argv[])
       {
5
6    for(int i = 0 ; i<argc; i++)
7      cout << argv[i]<< ' ';
8    cout << endl;
9
10   return 0;
11 }
```

```
$ ./a.out Hello World
./a.out Hello World
```

- argv[0] is the name of the program.

commandlineargs.cpp

# 2.9 Function Pointers

- Although functions are not variables, you can define pointers to functions.
- These pointers can then be passed between functions, put in arrays etc.
- To declare a pointer to a function:

```
returntype (*ptrname)(type1 arg1,type2 arg2 /*,...*/);
```

- All functions you want to use with the same function pointer, must have the same arguments and return type.

# Folding Vectors I

Left folding takes a value $x$ and a list of values and repeatedly applies $x = \text{function}(x, \text{valueFromList})$, for all the values starting at the top of list.

Here is a left folding function for integers:

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int add(int a, int b){return a+b;}
6  int multiply(int a, int b){return a*b;}
7  int second(int, int b){return b;}
8
9  int foldl (const vector<int> &vecin, int lvalue, int(*function)(
       int,int)){
10   for(unsigned int i = 0; i != vecin.size(); ++i)
11     lvalue = function(lvalue,vecin[i]);
12   return lvalue;
13 }
```

foldl.cpp

# Folding Vectors II

```
14
15 int main(){
16   vector <int> nums(5);
17   for (unsigned int i = 0; i != nums.size(); ++i)
18     nums[i]= 2*i+1; // Fill vector with odd nums
19   cout << foldl(nums,0,second) << endl;
20   cout << foldl(nums,0,add) << endl;
21   cout << foldl(nums,1,multiply) << endl;
22   cout << foldl(nums,0,multiply) << endl;
23
24   return 0;
25 }
```

```
twig$ g++ -Wall -Wextra foldl.cpp && ./a.out
9
25
945
0
```

# 2.10 Void Pointers

- Void in C++ means that there is no type. (e.g. int)
- They cannot be deferenced(∗), as they have no type.
- They cannot be incremented or decremented, as they have no length.
- To use a void pointer, you must type cast it.
- Void pointers in C may be used to pass parameters to functions.

As with all pointers, they are best avoided in C++ wherever possible. Templates or derived classes should be used instead for type safety.

# void* Example

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main(){
 5   char a[] = "hello";
 6   void *ptr; //Declare the pointer
 7
 8   ptr = &a; // Point it to a
 9   cout << (char*)ptr << endl; // Typecast it as a (char*)
10
11   ptr = new int; // Allocate some memory for an int
12   *(int*)ptr = 43110; // Set the allocated memory to 43110
13   cout << *(int*)ptr << endl; // Typecast and dereference to an
         int to print
14
15   return 0;
16 }
```

▸ voidstar.cpp

# 3. Structures

# 3.1 Basics of Structures

- Structures allow you to define new types, by grouping together other named variables or objects into a single object.
- To declare a struct type that contains an `int` field `i` and a `double` field `d` :

```
struct structTypeName{
    int i;
    int d;
};
```

- A struct can be made by `structTypeName structname;`.
- Fields can be accessed by `structname.fieldname`, acting just like variables.
- In C structs are defined as

```
typedef struct {/*...*/} structname;.
```

# Cartesian to Polar Coordinates I

Take an $(x, y)$ point (struct) in Cartesian coordinates and convert it to $(r, \theta)$ in polar coordinates.

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  struct point{ // Declare the point type
6    double x;
7    double y;
8  };
9
10 point carttopolar (point in){
11   double r = sqrt(in.x *in.x + pow(in.y,2));
12   double theta = atan2(in.x, in.y); // atan2 is correct for x and
         y <0
13   point out = {r, theta}; // Initialise the output structure
14   return out; // out.x = r; out .y = theta;
15   }
```

carttopolar.cpp

# Cartesian to Polar Coordinates II

```
16
17 int main (){
18   point a = {3.0,4.0};
19   a = carttopolar(a); // Set a to the polar version of a
20   cout << a.x << " " << a.y << endl;
21   return 0;
22 }
```

# 3.2 Operator Overloading

- Operators such as +, = and <<, are functions that can be overload, that is given new meanings with a unique type signature.
- To overload an operator, e.g. + as a new function :

```
outType operator+ (Type1 leftofoperator, Type2
    rightofoperator){
    /* function code*/
    return variableofTypeoutType;
};
```

- Like functions they need a function prototype.
- A full list of operators and whether they are overloadable is avaliable at `http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B`

# Multiply a Vector by a Scalar

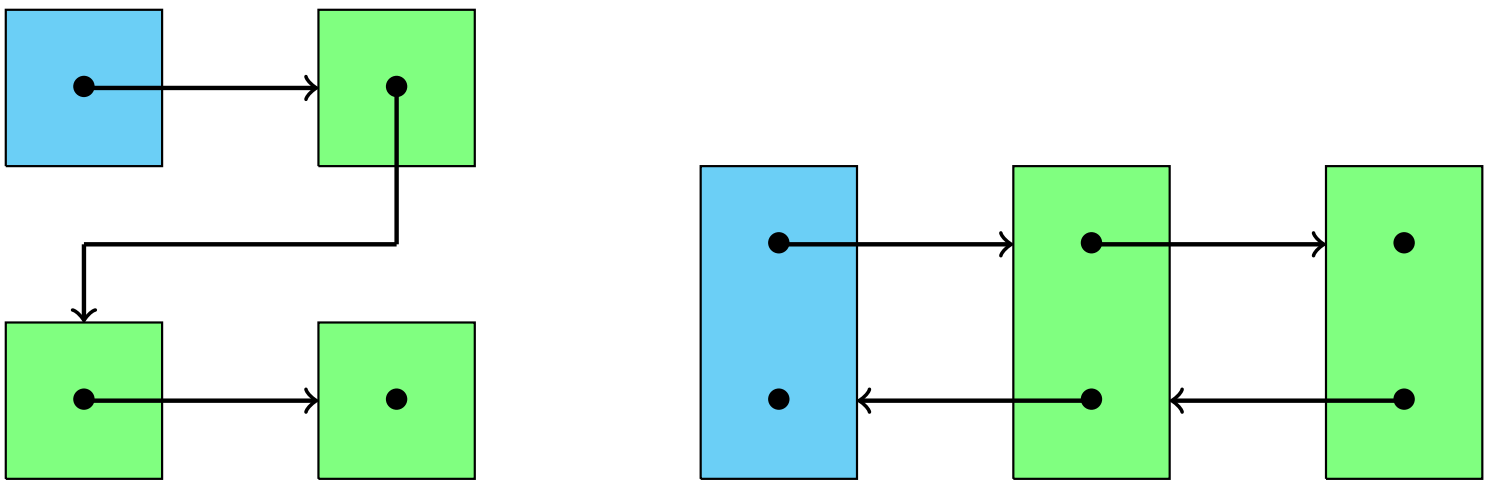
Take a vector and multiply it by a scalar.

```
1  #include <iostream>
2  using namespace std;
3
4  struct vector {
5    double x;
6    double y;
7  };
8
9  vector operator*(double phi, vector p){ // overloading the *
       operator
10   vector out = {p.x * phi, p.y* phi};
11   return out;
12 }
13
14 int main(){
15   vector a = {3,5};
16   a = 3*a; // Note a*3 will give a compile error
17   cout << a.x << " " << a.y << endl;
18 }
```

scalarmult.cpp

# 3.3 Self-referential Structures

- Structures cannot contain themselves, as they would need an infinite amount of memory. However, they can contain pointers to the same type.
- `a->b` can be used as a short hand for `(*a).b`, when you have a pointer (`a`) to a structure.



(a) Singly Linked

(b) Doubly Linked

Figure: Linked Lists

# Singly Linked List

```cpp
#include <iostream>
using namespace std;

struct node{
  int i; //An integer i
  node *next; // pointer to the next node in the list
};

int main(){
  node *head = new node; // 1st node
  (*head).i = 1;
  (*head).next = new node; // second node
  (*(*head).next).i = 2;
  head->next->next = new node; //a->b is the same as (*a).b
  head->next->next->i = 3;

  for (; head != NULL; head = head->next) // move head along the
      nodes
    cout << head -> i << " "; // Print out i in each node
  cout << endl;
}
```

linkedlist.cpp

# 4. Header Files & Libraries

# 4.1 Header Files

- Header files are used to divide programs into parts.
- They are included by:

```
#include "nameofheaderfile.h"
```

  note: `""` rather than <>.
- Header files end in '`.h`' (or occasionally '`.hpp`').

## using namespace

Do not use `using namespace` in header files, as when they are included the other code will inherit it. Instead refer to the function directly using the scope resolution operator `::`, e.g.

```
std::cout << "hi"<< std::endl;.
```

# Header File Example: Power Function

Here we will print out the powers of 2, but separate the pow function into a separate header file.

`pow.h`:

```
1 int pow (int, int); // Function Prototype
2
3 int pow(int b, int e){ // Function Definition
4   int out = 1;
5   for (;e >= 1; e--) out*=b;
6   return out;
7 }
```

- If `pow.h` is included multiple times, there will be a redefinition error. This is because including a file is just like having the contents there instead.

pow.h

power2.cpp:

```
1 #include <iostream>
2 #include "pow.h" // includeing the header file
3 using namespace std;
4
5 int main(){
6   for (int i=0; i <10; i++)
7     cout << pow(2,i) << endl;
8   return 0;
9 }
```

# 4.2 Conditional Inclusion

- The C++ preprocessor can be controlled with conditional statements (if-then-else)
- It has `#if`, `#else`, `#elif` (else if). The end of the *if* statement it needs a `#endif`.
- There are also specialised forms, e.g. `#ifdef` (if . . . is defined then) and `#ifndef` (if . . . is not defined then).
- These can be used to ensure the contents of a header file are only included once:

```
#ifndef headername
#define headername
... Contents of header file ...
#endif
```

, by checking if `headername` has been defined, then if not define it, and include the content of the header file.

# 4.3 Compiling Multiple Files

- The more code you have the longer it will take to compile. If you have a lot of code then it saves time to re-compile only the parts that are changed and *link* together with the compiled parts.
- For this two things need to be done:
  - Split the code into multiple `.cpp` files, and put the function prototypes into header files.
  - Compile the `.cpp` files to object code separately, and link them together to form the executable.

# Multiple Files of Power Example

Here the power (`int pow(int, int)`) is in the `mpow.cpp` file with its function prototype in the `mpow.h` file, both in the `thomasmath` namespace, so as not to conflict with the definition in `cmath` (in the `std` namespace).

`mpower2.cpp`:

```
1  #include <iostream>
2  #include "mpow.h" // including the header file
3  #include "mpow.h" // including the header file
4  using namespace std;
5
6  int main(){
7    for (int i=0; i <10; i++)
8      cout << thomasmath::pow(2,i) << endl;
9    return 0;
10 }
```

▸ mpower2.cpp

mpow.h:

```
1 #ifndef __mpow__ // if __mpow__ is not Defined
2 #define __mpow__ // define __mpow__
3
4 namespace thomasmath{ // the function is put in the thomasmath
      namespace
5   int pow (int, int);
6 }
7
8 #endif
```

mpow.cpp:

```
1 #include "mpow.h"
2
3 // Function definition for pow in thomasmath namespace
4 int thomasmath::pow(int b, int e){
5   int out = 1;
6   for (;e >= 1; e--) out*=b;
7   return out;
8 }
```

▸ mpow.h

▸ mpow.cpp

# Multiple Files of Power Example:Compiling

- To compile them altogether:

  ```
  g++ -Wall -Wextra mpower2.cpp mpow.cpp
  ```

  `-Wall -Wextra` turns on warnings, to help you find bugs in your code.

- To compile them separately, then link:

  ```
  g++ mpow.cpp -c
  g++ mpower2.cpp mpow.o
  ```

  this creates `mpow.o` which is a compiled version of `mpow.cpp`, then this is compiled in with `mpower2.cpp` to produce the executable.

# References

# References


- B. Kernighan, D. Ritche, *The C Programming Language*, 1988
- J. Nash, P. Dauncey, *Introduction to C++*, Blackett Lab, (2003)
- D. Lee, *First Year Laboratory Computing Laboratory*, Blackett Lab, (2006-7)
- *C++*, `http://en.wikipedia.org/wiki/C%2B%2B`, (6/11/2007)
- L. Haendel, *The Function Pointer Tutorials*, `http://www.newty.de/fpt/zip/efpt.pdf`, (22/1/2008)
- J. Soulie, *Pointers*, `http://www.cplusplus.com/doc/tutorial/pointers.html`, (25/6/2007)