INTRODUCTION

In this week we cover the following topics

1.1 SUBJECT GOALS

- The aim of this subject is to teach students about data structures and algorithms using the programming language C++.

- Not every nuance of C++ is taught but only it's main features including

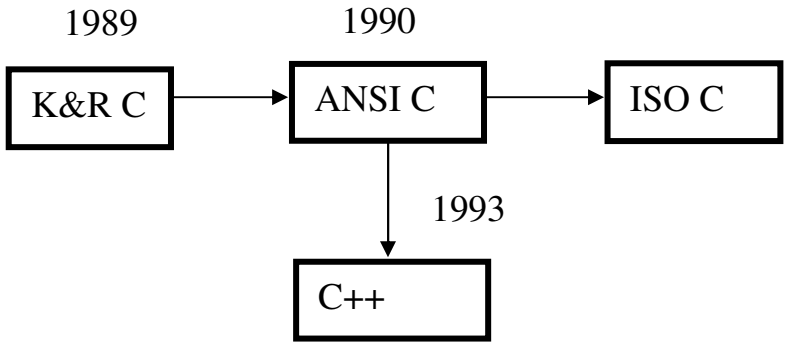
  pointers
  functions
  arrays
  strings
  structs
  classes
  dynamic memory allocation
  file handling
  libraries
  recursion
  templates

- Emphasis is given to good program design and how to achieve it plus consideration of coding style, debugging and testing techniques.

- We also study a number of important data structures and algorithms as they are implemented in C++. These include

  Searching and Sorting
  Vectors
  Linked Lists
  Trees
  Hash Tables
  String Searching
  NP Complete Problems

1.2 C++ BACKGROUND



- C was developed in concert with UNIX

- C++ developed as an OO version of C

- Intended to be a general-purpose language. Equally useful for numeric analysis, text manipulation, graphics, etc.

- C dates from a period when terminals were very slow, so it's terse compared to newer languages.

- For most of the time, the syntax of C++ and Java is the same.

- ANSI: American National Standards Institute
- ISO: International Standards Organisation
- The standards define many aspects of the language but leave some things undefined. Mainly to allow compilers to generate efficient code in different machines.
- If properly used, standard C++ programs
  - are portable between systems and compilers.
  - always produce the same results (or almost!)

1.3 COMPILING PROGRAMS

- Within the faculty the student UNIX servers are `charlie` and `sally`. The C++ compiler which can be used is `g++`.

- There is another package called `lint` which can be used to check code for any errors.

1.4 PROGRAM EXAMPLE - `number.cpp`

```
/*********************************************\
   ask user to enter a number
   print 0 to number-1
\*********************************************/

#include <iostream>

using namespace std;

int main()
{
   int count, number;

   cout << "Please enter a number\n";
   cin >> number;

   if (cin.good()) {
      cout << "Printing numbers from 0 to " <<
         number-1 << "\n";
      for (count=0; count<number; count++) {
         cout << count << "\n";
      }
   } else {
      cout << "Unable to read number\n";
   }

   return 0;
}
```

We compile the program with the following command
`g++ number.cpp`

This compiles the program to `a.out`. We run the compiled program thusly

```
sally% ./a.out
Please enter a number
4
Printing numbers from 0 to 3
0
1
2
3
sally% ./a.out
Please enter a number
3r
Printing numbers from 0 to 2
0
1
2
sally% ./a.out
Please enter a number
r3
Unable to read number
sally%
```

`number.cpp` introduces a number of concepts.

1.4.1 Comments

C++ has two ways to put comments in code.

1. Using the starting sequence of `/*` combined with the ending sequence of `*/` There may be any number of lines of text in between
2. `//` will turn the rest of the text in the line into a comment.

1.4.2 Pre-processor

- As the first step in compiling, the compiler runs through a pre–processor phase. Pre-processor instructions start with `#` and are usually placed at the start of the source code.

- There are many commands in the pre-processor. The most commonly one is `#include`

- `#include` is an instruction to the pre-processor that it will need to get a library of functions that are not part of standard C++. There are many libraries that come standard with C++. Its syntax is as follows

  ```
  #include <library>
  ```

  Where `library` is the name of the library you want to include. There are many libraries in C++ which will be discussed during this course.

1.4.3 `iostream`

- `iostream` is a frequently used library that contains input and output functions and objects. In particular, it contains the `cout` and `cin` objects.

- In the program, the string "`hello world\n`" is fed into `cout` using the `<<` operators which will, in turn, place the string on the screen.

- The object for getting input from the keyboard is `cin` and the operator it uses for extracting input from the input stream is `>>`.

- We can check the status of `cin` to see if it has successfully read data in by calling the `cin.good()` member function.

- We will learn more about `iostream` during the subject.

1.4.4 Namespaces

- In large programs a significant problem is creating objects with the same name as other objects, particularly if the objects are contained in different libraries. Namespaces solve this problem by gathering all the names in a particular library into one named space.

- The functions in the `iostream` library are in the `std` namespace. Normally you would use `cout` thusly

```
std::cout << "Hello world\n";
```

- The line `using namespace std;` means we don't have to add the `std` to the function call;

1.4.5 Variables and Data Types In C++

- A variable name is a label for a piece of memory set aside to contain a specific type of data. E.g.

```
int average;
double temperature;
```

- You can also initialise variables when declaring them. E.g.

```
int average = 2;
double temperature = 29.6;
```

- C++ has a number of *reserved* words that can't be used as variable names. They are

asm	do	if	return	try
auto	double	inline	short	typedef
bool	dynamic_cast	int	signed	typeid
break	else	long	sizeof	typename
case	enum	mutable	static	union
catch	explicit	namespace	static_cast	unsigned
char	export	new	struct	using
class	extern	operator	switch	virtual
const	false	private	template	void
const_cast	float	protected	this	volatile
continue	for	public	throw	wchar_t
default	friend	register	true	while
delete	goto	reinterpret_cast		

- There are a number of data types but they fall into two broad categories, namely *scalars* (whole numbers) and *reals* (which have fractional values).

Scalar Variables

Data Type	Num Bytes	Signed Range	Unsigned Range	Comments
`bool`	1	-	-	true or false
`char`	1	-128→127	0→255	Store ASCII
`short`	2	-32768→ 32767	0→65535	
`long`	4	-2147483648 →2147483647	0→ 4294967295	
`int`	2→8 Usually 4			Depends on machine and OS.

- ASCII – American Standard Code for Information Interchange. Basically, a numerical code for letters and other printing characters.

- Unless preceded by the `unsigned` modifier all scalar variables are signed - meaning they can have positive and negative values. E.g

```
unsigned int volume;
```

Real variables

Data Type	Num Bytes	Approx Smallest Number	Approx Largest Number	Approx accuracy (decimal digits)
`float`	4	1.4×10^{-45}	$3.4\times10^{+38}$	7
`double`	8	4.9×10^{-324}	$1.8\times10^{+308}$	15

Mixing Data Types In Expressions

- You must always be careful with expressions that mix data types.

- For instance, given the following
  ```
  long L = 88888888;
  char c = L;
  ```
  then the value of `c` will be equal to `L` truncated down from 4 bytes to 1.

- Similar problems can occur if you mix signed and unsigned data types.

- You must also be very careful when your expressions mix scalar and real data types.

1.4.6 Expressions and Statements

- An *expression* is something that the computer processes. An example would be
  ```
  (2.5 * c > d / 1.6)
  ```

- A *statement* is a separable part of a program, usually followed by a semicolon. E.g
  ```
  y = a + b;
  ```

- A block of code in braces `{ }` is called a *compound statement*.

1.4.7 Arithmetic Operators

- There is a standard list of arithmetic operators available. These are the binary operators `* / + - %` and the unary plus and minus.

- The `%` computes remainders (scalars only). E.g `13%5` gives `3`.

- The unary minus refers to the minus in front of values and variables. E.g. `-a` or `-6`.

- Division between scalars produces scalars which are truncated down. E.g. `13 / 5` gives `2` not `2.6`.

Arithmetic Shortcuts

- Instead of `a = a * 6;` you can type `a *= 6;`
  This applies to all the binary operators.

- Instead of `i = i + 1;` you can type `++i;` or `i++;`
  There are subtle differences between these if they are used within larger expressions. So long as they are used as a separate statement they are the same. You can also have `--i;` and `i--;` which decrements `i` by 1.

1.4.8 Relational Operators

- C++ has a number of relational operators. These are

Operator	Meaning
<	Is less than?
<=	Is less than or equal?
==	Is equal to?
!=	Is not equal to?
>=	Is greater than or equal to?
>	Is greater than?

- When C++ evaluates an expression it returns a 0 if the expression is false and a non-zero value if true.

  Example

```
int x=0, y=1;
```

  `if (x < y)` would be the same as `if (1)`
  `if (x > y)` would be the same as `if (0)`

1.4.9 Logical Operators

In C++ the AND operator is `&&` while the OR operator is `||`

- Examples

```
int a = 3, b = 3, c = 4;
```

```
(a > b && b <= c)
(FALSE and TRUE) ⇒ FALSE
(a == b || a == c)
(TRUE or FALSE) ⇒ TRUE
```

Lazy Evaluation

- C++ uses *Lazy Evaluation*. Consider the following
  `if (x != 0 && 1/x < y)`
  If `x` equals zero then the first part of the expression is false. Since AND statements require both parts to be true for the whole expression to be true it follows that the above expression would be false. C++ would not evaluate the second part of the expression if the first part was false. This is lazy evaluation.

- Similarly, if the first part of an OR statement is true it follows that the whole expression is true and C++ won't bother to evaluate the second part of the expression.

1.4.10 Operator Precedence

- Expressions are evaluated from left to right. However, the following general precedence follows.
  1. Evaluate expressions in braces first.
  2. Evaluate Arithmetic operators second.
  3. Evaluate Relational operators third.
  4. Evaluate Logical Operators last.

- The following table gives operator precedence, from highest to lowest.

`()  []  ->  .`	highest
`!  -  +  ++  --  *  &`	
`*  /  %`	
`+  -`	
`<  <=  >  >=`	
`==  !=`	
`&&`	
`||`	
`=  +=  -=  *=  /=  %=`	
`,`	lowest

- NOTE. These are only the operators we will see in this course. There are others but we will not be looking at them.

1.4.11 Program Control

`if` Statement

The structure of the `if` statement looks like

```
if (logical expression) statement;
```

or

```
if (logical expression) statement;
else statement;
```

The statements can be replaced by compound statements so we can have

```
if (logical expression) {
   many statements;
} else {
   many statements;
}
```

`switch` Statement

It is possible to write code like this

```
char command;

cout << "\nEnter menu letter ";
cin >> command;

if (command == 'f' || command == 'F') {
   cout << "\n Menu F selected\n";
} else if (command == 'e' || command == 'E') {
   cout << "\n Menu E selected\n";
} else if (command == 'v' || command == 'V') {
   etc, etc, etc
```

Repeated `else if` statements gets clumsy after a while. C++ has the `switch` statement to help tidy this up.

```
switch (command) {
    case 'f' : case 'F' :
       cout << "\n Menu F selected\n";
       break;
   case 'e' : case 'E' :
       cout << "\n Menu E selected\n";
       break;
   case 'v': case 'V':
       etc, etc, etc
       . . . . .
   default :
       cout << "\n Invalid menu item\n";
```

```
}
```

Notes

- The `break` statement in each `case` tells the program to exit the `switch`. For example, in the earlier `switch` example the program first tests the `case 'f' :` and `case 'F' :` Assuming the command variable is one of these then the `cout << "\n Menu F selected\n";` is run then the `break`. The `break` says not to test any more cases but finish the `switch`.

- The `default` case is part of the `switch` structure. It is there for any variable for which there is no specific `case`. It is always wise to use the `default`, especially as a form of error checking.

1.4.12 Loops

There are often times when you want your code to repeat some action many times over. To accomplish this C++ has a number of looping constructs. These are the

- `while` loop
- `do-while` loop
- `for` loop

`while` Loop

The structure of the `while` statement looks like

```
while (logical expression) statement;
```

or

```
while (logical expression) {
   many statements;
}
```

The while loop will run continuously until the logical expression becomes false. In other words, the logical expression is the loop's terminating condition.

`do-while` Loop

The structure of the `do-while` statement looks like

```
do statement;
while (logical expression);
```

or

```
do {
   many statements;
} while (logical expression);
```

The loop continues to run while the logical expression is true.

The `do-while` loop runs its code first then checks to see if it should loop again.

`for` Loop

The `for` loop is more complex than the previous two loops but it is one that is used very often.
It's structure is as follows

```
for (init; test; control) statement;
```

or

```
for (init; test; control) {
   many statements;
}
```

- The `init` section of the loop is where we initialise any variables used within the loop.

- The `test` section of the loop is where we test if the loop continues or finishes.

- The `control` section is where we can update any variables used within the loop.

- Any of the sections may be omitted except the semi-colons.

- `for` loops are equivalent to

```
init;
while (test) {
   many statements;
   control;
}
```

1.4.13 `continue`, `break`, `return` & `exit`

C++ has a number of commands that further control how loops work. These are the `continue`, `break`, `return` and `exit` commands.

- When the `continue` command is executed the program will ignore the rest of the statements in the loop and start back at the top. E.g.

```
int count;
double value, big;

for (count=1, big=0.0; count<=10; count++)
{
   cout << "\nEnter a number ";
   cin >> value;
   if (value < 0) continue;
   if (value > big) big = value;
}
cout << "\nBiggest value entered << big
     << "\n";
```

  In this example, when the value is less than zero the following lines in the loop are ignored and the loop starts back at the top.

- When the `break` command is executed the loop will be terminated and the program will continue with the first line after the loop. E.g.

```
int number = 1;

while (1)
{
   cout << "Please enter a number ";
   cin >> number;
   if (number == 0) break;
}
cout << "while loop ended\n";
```

- In this example `while (1)` will run forever since 1 is always true (see section on relational operators). When the number equals zero it executes the `break` statement which causes the program to exit the loop and continue with the next statement after the loop.

- When the `return` command is executed it causes the program to exit the function it is in.

- When the `exit` command is executed it causes the program to terminate. It requires the inclusion of the `stdlib.h` header library and is usually written in one of two ways

```
exit(0)  // Terminate successfully
exit(1)  // Terminate with error
```