Course Overview and Rough C++ Guide

Lecture 1

Professor Peter Cheung
EEE, Imperial College

- In this lecture,
  - we take an overview of the course, and
  - briefly review the C++ programming language.
- The rough C++ guide is not very complete. You should use a suitable book as a proper reference for the C++ language.

Course Aims

- The aim of this course is to familiarise you with a number of *principles*, *concepts* and *techniques* from computer science.
- These principles, concepts and techniques are general purpose. They apply
  - whatever programming language you use (Java, C++, Pascal), and
  - whatever computer you're using (a PC, a Unix system, a Mac).
- Once you know basic engineering principles, concepts and techniques you'll find it easy to pick up new programming languages, new operating systems, etc.

Recommended Texts

- You could get away with nothing. But a good C++ textbook would help, such as:

  Problem Solving with C++, 5th Edition, Walter Savitch, Addison Wesley, ISBN: 0321269756, 2004 (£40.84).
- You can also get a full C++ Tutorial free on the web at:

  http://www.functionx.com/cppbcb/
- If you are serious about learning really how to programme well, then the following book (language independent) is a must:

  Code Complete, Steve McConnell, Microsoft Press, ISBN: 0735619670 (£25.84).

Principles, Concepts & Techniques

- The ***principles*** you will learn include things like how to construct code that's easy to read, understand and modify.
- The ***concepts*** you will learn about include things like abstract data types, object-oriented programming, and so on.
- The ***techniques*** you will learn about include
  - how to build ***data structures***, like lists and trees, and
  - ***algorithms*** for certain common tasks, like lookup and sorting.

The C++ Language I

- But general-purpose programming principles, concepts and techniques can only be acquired by ***doing lots of programming***.
- Programming is like riding a bike — it can't be learned from a book, it requires lots of practise.
- So we have to learn via a *particular* programming language on a *particular* machine.
- On this course we use C++ using Borland C++ Builder (BCB).

The C++ Language II

- The C++ language — is an object-oriented variant of the C language.
  - "**Object-oriented**" means that code and data can be bundled together into a single entity called an ***object***. Classes of objects are organised into ***hierarchies***. More on this later in the course.
- But Borland C++ Builder (BCB) is not just a programming language. It's a whole development environment (IDE). It includes an ***editor*** and a ***debugger***, as well as a ***compiler***.
- BCB is also a *visual* development environment. It supplies lots of support for building the ***graphical user interface*** (GUI) of a program. It contains a Visual C++ Library (VCL) which helps you to write visual programmes.

Course Outline

- The course will touch on each of the following topics.
  - A review of C++
  - Software engineering principles
  - Data structures
    - Lists
    - Ordered lists
    - Trees
    - Ordered trees
    - Hash tables
  - Parsing
  - Object-orientation

C++

- C++ is a programming language.
- It is a
  - block structured,
  - strongly typed,
  - procedural language.
- It's a lot like C or Pascal or Java, but arguably more flexible and more widely used than others.

Programming Languages

- "**Block structured**" means that the flow of control in a program is strictly mediated by a small number of constructs, namely
  - sequences of statements,
  - conditional statements (e.g. **`if`** *(<condition>)* **`then`** {*<statements>*} **`else`** {*<statement>*}), and
  - loops (e.g. **`while`** *condition* {*<statement>*}).
- "**Strongly typed**" means that the programmer has to declare the *type* of each variable used in the program, ie: whether it is an integer, a string, an array, or whatever.
- "**Procedural**" means that the program specifies a series of instructions and the order in which they are to be carried out. In a *declarative* language, by contrast, the programmer only describes the meanings of things — functions or predicates — and leaves it to the computer to work out the details of how computations are to be performed. Prolog is an example of a declarative language.

A Skeleton C++ Program

- Here is a skeleton program in C++. (This is a console program. It doesn't use the fancy user interface features of BCB.)

- This `include` statement specifies the header file `iostream.h` to be used. It defines standard i/o `cin` and `cout`.
- All C++ programs begin with a line of this form. It always starts with `main()`.
- All C++ statements are terminated by a semicolon.
- When exit, the main programme returns a value 0 to signify no error has occurred.

```
#include <iostream.h>

int main()

{
  cout << "Hello, World!";
  cout << endl;
  getchar();
  return 0;
}
```


Pascal vs C++ (1 - Comments)

Pascal	C++

Comments are placed between braces:
`{ This is a comment }`
The older notation:
`(* This is a comment *)`
	Comments are placed between `/*` and `*/`
	`/* This is a comment */`
	
	The second method: a comment is placed after `//`. Then it extends to the end of the line:
	`x = x+1; // A comment to the end of the line.`

Extracts from "Pascal and C++ Side by Side", Maria Litvin. (http://www.skylit.com/pascpp/)

Pascal vs C++ (2 - upper/lower case)

Pascal	C++
PASCAL IS CASE-BLIND.	C++ is case-sensitive.

Pascal	C++
Names can use letters and digits but must begin with a letter, e.g.: `amount, x1, str3a`	Names can use letters, digits and the underscore character, but must begin with a letter or the underscore, e.g.: `amount, x1_, _str3a`

Pascal vs C++ (3 - Block & Semicolons)

Pascal	C++

A compound statement is placed between `begin` and `end`:
 	
`begin`
`  <statement1> ;`
`  <statement2>`
`end;`
 	
Semicolon is optional before `end` and is usually required after `end`, unless followed by another `end`.
	A compound statement is placed between braces:
	
	`{`
	`  <statement1> ;`
	`  <statement2> ;`
	`}`
	
	Semicolon is required before the closing brace, and usually omitted after it.

Pascal vs C++ (4 - Built-in & Enumerated Data Types)

Pascal	C++

`char`
`integer`
`real`
`boolean`
	`char`
	`int  long  short`
	`float  double  long double`
	
	`char`, `int`, `short`, and `long` may be preceded by the `unsigned` keyword.
	`double` is a double-precision real number.
	`bool` is in the process of becoming standard.

Pascal	C++
`type` `  Color = (Red, Green, Blue);`	`enum Color {Red, Green, Blue};`

Pascal vs C++ (5 - Constants)

Pascal	C++

All declarations of constants in the main program or in a procedure or a function are grouped together under the keyword `const`:
`const`
`  Pi = 3.14;`
`  Rate = 0.05;  {.05 not allowed }`
`  Hour = 3600;`
`  Dollar = '$';`
`  Greeting = 'Hello, World!';`
There are no "escape" characters. Two single quotes in a row in a literal string represent one single quote character:
`  writeln ('Let''s have fun!');`
	Declarations of constants are the same as declarations of variables with initialization, but they are preceded by the keyword `const`:
	`const double Pi = 3.14,`
	`  Rate = .05;  // or 0.05;`
	`const int Hour = 3600;`
	`const char Dollar = '$';`
	`const char Greeting[] = "Hello, World!";`
	`//  Also allowed:`
	`const double R = 5., Pi = 3.14, Area = Pi * R * R;`
	C++ recognizes so-called "escape characters" for special `char` constants. These are written as a backslash (which serves as the "escape" character) followed by some mnemonic char. For example
	`'\n'  newline  '\''  single quote`
	`'\\'  backslash  '\"'  double quote`
	`'\a'  alarm (bell)  '\f'  form feed`
	`'\t'  tab  '\r'  carriage return`
	Character constants with escape chars are used the same way as regular char constants. For example:
	`const char CR = '\r';  // Carriage Return`
	`cout << "Hello, World\n";`

Pascal vs C++ (6 - Variables)

Pascal	C++

All declarations of variables in the main program or in a procedure or a function are grouped together under the keyword `var`:
 	
`SomeProcedure (...);`
`  ...`
`  var`
`  r : real;`
`  i, j : integer;`
`  star : char;`
`  match : boolean;`
`  ...`
`  begin`
`  ...`
`  end;`
 	
No initialization is allowed in declarations.
	Declarations of variables (or constants) may be placed more or less anywhere in the code, before they are used. Beginners are advised to place them at the top of `main()` or at the top of a function to avoid complications with the scope rules. Global variables, declared outside any function (and outside `main()`), are allowed, but should be avoided. Values of variables may be initialized to constants or previously defined variables or expressions:
	
	`SomeFunction (...)`
	`{`
	`  double r = 5.;`
	`  int i = 0, j = i+1;`
	`  char star = '*';`
	`  ...`
	`}`

Pascal vs C++ (7 - Arrays)

Pascal	C++

`var`
`  str : packed array [1..80] of char;`
`  grid : array [1..32, 1..25] of integer;`
 	
The `packed` keyword is recommended for an array of characters to save space. The range of subscripts can start from any number, but usually starts from 1. Here `str[1]` refers to the first element of the array `str`. Pascal compilers normally report an error if a subscript value is out of range.
	`char str[80];`
	`int grid[32][25];`
	
	The subscript for the first element of the array is 0. Here `str[0]` refers to the first element of the array `str` and `str[79]` to the last element. C++ compilers do not verify that a subscript value is within the legal range.
	
	Arrays can be initialized in declarations. For example:
	
	`int fibonacciNumbers[6] = {1,1,2,3,5,8};`
	`char phrase[80] = "Hello, World!";`

Pascal vs C++ (8 - Type Definition)

Pascal

The `type` keyword is used to define enumerated and *subrange* types, array types, and records:

```
type
  DigitType = 0..9;                          { Subrange type }
  ColorType = (Red, Green, Blue);            { Enumerated type }
  WordType = packed array [1..30] of char;   { Array type }
```

C++

The `typedef` keyword is used to define aliases for built-in (and, if desired, user-defined) types:

```
                                  // Used later in declarations as:
typedef unsigned char BYTE;       //    BYTE pixel;
typedef double MONEY;             //    MONEY price = 9.95;
typedef int BOARD[8][8];          //    BOARD board;
```


Pascal vs C++ (9 - Procedures vs Functions)

Pascal

Procedures and functions take arguments of specified types. Procedures do not explicitly return a value. Functions return a value of the specified type.

```
procedure DoSomething (arg1 : integer; arg2 : char);
...
  begin
    ...
  end;
{*************************************************}
function ComputeSomething (arg1, arg2 : integer) : real;
...
  begin
    ...
    ComputeSomething := <expression>;
  end;
```

The return value in a function is indicated by using the assignment statement.

Pascal vs C++ (9 - Procedures vs Functions (cont'))

C++

There are no procedures, everything is a function. Functions take arguments of specified types and return a value of the specified type. Functions that do not explicitly return a value are designated as `void` functions.

```
void DoSomething (int arg1, char arg2)
{
  ...
}
```

Functions of the type other than void return a value of the specified type. The return value is indicated by using the return statement.

```
double ComputeSomething (int arg1, int arg2)
{
  ...
  return <expression>;
}
```

A function can have multiple `return` statements. A `void` function can have `return` statements without any value to return.

```
  if ( <condition)> )
    return;
  ...
```

This is used to quit early and return to the calling statement.

Pascal vs C++ (10 - Parameter passing)

Pascal

The `var` keyword is used (passing parameter by **reference**), or without `var` (passing parameter by **value**) :

```
procedure Swap (var x, y : integer);
procedure QuadraticEquation (a, b, c : real; var x1, x2 : real);
```

C++

The `&` symbol is used:

```
void Swap (int &x, int &y);
void QuadraticEquation (double a, double b, double c,
                        double &x1, double &x2);
```


Pascal vs C++ (11 - Arithmetic Operators)

Pascal

`:=` (assignment)
`+`
`-`
`*`
`/` ("real" division)
`div` ("integer" division)
`mod` (modulo division)

Arithmetic operations are allowed only for `integer` and `real` operands. `div` and `mod` are used only with `integer` operands. No arithmetic operation are allowed for variables of the `char` or `boolean` types.

The result of an arithmetic operation has `integer` type when both operands have `integer` type and `real` when at least one of the operands is `real`. The "real" division `/` is an exception: the result is always a `real` value, even if operands are integers.
The result of `div` is the quotient truncated to an integer (in the direction of 0).
Examples:

```
var
  x : real;
  n : integer;
...
  x := 2 / 3;   { x gets the value of 0.666667 }
  n := 2 div 3; { n gets the value of 0 }
```


Pascal vs C++ (11 - Arithmetic Operators)

C++

`=` (assignment)
`+`
`-`
`*`
`/`
`%` (modulo division).

Arithmetic operations are allowed for all built-in types, including `char`, although `%` makes sense only for integral types (`char`, `int`, `long`, `short`, etc.). `char` operands use the actual binary value stored in that byte and have a range from –127 to 127. They are first automatically converted to `int` in arithmetic operations.

The intermediate type of the result is always the same as the type of the operands. If the operands have different types, the "shorter" operand is first *promoted* to the type of the "longer" operand (e.g. `int` may be promoted to `long`; or `long` to `double`).
Examples:

```
double x;
...
  x = 2. / 3;   // x gets the value of 0.666667
  x = 2 / 3;    // x gets the value of 0 (!!!)
```


Pascal vs C++ (12 - Compound Arithmetic Operators)

Pascal	C++
No such thing.	The compound arithmetic operators are very much a part of the C++ style and are widely used.

```
                // Is the same as:
a++;        //    a = a + 1;
b = a++;    //    {b = a; a = a + 1;}
b = ++a;    //    {a = a + 1; b = a;}
a--;        //    a = a - 1;   Also: --a;
a += b;     //    a = a + b;   Also: a -=b; a *= b; a /= b; a %= b;
```

Pascal vs C++ (13 - Built-in Maths Functions)

Pascal	C++

Built-in functions:
 	
`abs(x)`
`sqrt(x)`
`sin(x)`
`cos(x)`
`exp(x)`
`ln(x)`
`sqr(x)`
`arctan(x)`
	Standard library functions (require `#include <math.h>`):
	
	`int abs(int x);  double fabs(double x);`
	`double sqrt(double x);`
	`double sin(double x);`
	`double cos(double x);`
	`double exp(double x);`
	`double log(double x);  // Natural logarithm`
	`double pow(double base, double exponent);`
	`double atan(double x);`

Pascal vs C++ (14 - Boolean variables & values)

Pascal	C++

Has built-in `boolean` type and constants `true` and `false`.	Any integer non-zero value is treated as "true," and zero as "false." `bool` type is in the process of becoming standard. If not supported by their compiler, programmers may use their own definition. For example:
	`typedef int bool;`
	`#define false 0`
	`#define true 1`

Pascal	C++

The result of relational operators has the type `boolean`.
`=`
`<>`
`<`
`<=`
`>`
`>=`
	The result has the type `bool` and has the value `false` or `true`:
	`==`
	`!=`
	`<`
	`<=`
	`>`
	`>=`

Pascal vs C++ (15 - Logical Operators)

Pascal

`and`
`or`
`not`
Example:
`function LeapYear(yr : integer) : boolean;`
`  begin`
`    LeapYear := ((yr mod 4 = 0) and`
`      ((yr mod 100 <> 0) or (yr mod 400 = 0)));`
`  end;`

C++

`&&`
`||`
`!`
Example:
`bool LeapYear (int yr)`
`{`
`  return (yr % 4 == 0 &&`
`        (yr % 100 != 0 || yr % 400 == 0));`
`}`

What to do before the next lecture?

- Use Level 3 computer lab or install a copy of BCB on your own machine.
- Complete Lesson 1 & Lesson 2 of the following C++ Tutorial on the web:

http://www.functionx.com/cppbcb/Lesson01.htm

http://www.functionx.com/cppbcb/Lesson02.htm