ACS223 Computer Systems Applications

Semester 1, Weeks 3, 4, 5,6
Lab: C++
There will be a C++ assessment due in week 8

Learning C++

- <u>Assumption: You already know how to program in C.</u>
- Some things about C and C++ are the same.
- In this topic you will review the major constructs in C that are the same as in C++ , and you will learn the constructs that are different in C++.
- The main difference between C and C++ is the availability of **classes in C++.** Since you will create an **object oriented** design for your ACS223 project, you will need classes in C++ to implement your design.
- The teaching method for this section will be "hands-on": you will have programming tasks to complete, examples will be provided, and help will be available.

Whenever you see this icon, lynda.com , it directs you to an online resource, lynda.com, available through MOLE.

MOLE -> lynda.com (you might have to click on 'View all services' to access lynda.com). We will be using two videos:

- Foundations of Programming: Object-Oriented Design
- C++ Essential Training

NOTE: You need to bring your headphones to the lab session if you want to listen to these videos during the session!

Using CodeBlocks Projects

- To set up a project, do the following in CodeBlocks:
  - Choose File > New > Project>Console Application>,
  - If it opens up the wizard: click "skip this next time', then click Next
  - Select C++, then click Next.
  - Give your project a title
  - Create a new folder (if you want) and save the project in the new folder.
  - Under 'Sources' there is a file called 'main.cpp'.

- To add to the project, you can include an existing file to your project, or New file if you want to create a new file in the project.
  - If you include an existing file, right click on the project name and select 'Add file'. **Make sure the file is saved in the project folder first.**
  - To add a new file: File>New>File>C/C++ source>C++>place the new file in the appropriate folder and give it a name.
  - Follow same step above for a header file .h
  - Make sure that the main program, all classes that you have created in .cpp files, along with their header (.h) files are added to the project.
  - Ensure that your main function and the class implementation both have an include

statement for the class' header file.

- Use the Build icon, and CodeBlocks will compile and build the project for you.
- Use the Run icon to run main.

Types in C++

C++ has the built-in types that C also has:

Integral types:

- 1 byte types: char, unsigned char
- 2 byte types: short, unsigned short
- 4 byte types: int, unsigned int
- 8 byte types: long, unsigned long

Floating point types (with an integer part and a fractional part):

- 4 bytes: float
- 8 bytes: double
- 10 bytes: long double

In addition, C++ has a built-in boolean type, e.g.:

```
bool errorCondition;
```

- The bool variable `errorCondition` can take the value 1 (for true) or 0 (for false).

Standard Template Libraries

C++ Essential Training, Chapters 9.1, 9.3 and 9.4

Strings

- An important type that is supplied in the C++ standard library 'string.h' is the string type, e.g.:

```
#include <string>
std::string code;       //variable name
code = "101100111";    //variable assignment
```

For both built-in and user defined types:

- A type has an associated set of permitted values. The set of permitted values for a type is normally architecture dependent.
- Precedence of operators is the same in C++ as in C.
- A type has an associated set of operations that can be performed upon it (e.g. +, -, /, * for integral and floating point types).
- The user defined type, string, has many helpful operations, e.g. concatenation (with +), `length, find, substr, at, sort`, and others.

Example of concatenation:

```
std::string name = "John";
name = name + " Smith";
```

Output in C++

- For output to the screen in C++, use `cout` (pronounced "see-out").
- cout links to the output stream for the standard output device. An output stream links to operating system drivers.
- Use the **insertion operator** << with `cout`. << is usually expressed as "put to".
- When you output a variable with `cout`, you put the variable to the standard output stream.
- `cout` results in default formatting of output dependent on the variable type, and you can also choose to specify the formatting rather than use the default.
- You can put variables, format specifiers and constants to the output stream.
- Put "`endl`" to the output stream to tell the program that you are finished writing a line.

Example1:

```
std::cout << 4 << std::endl;
```

Example2:

```
int a=4;
std::cout << a << std::endl;
```

Example3:
```
std::cout << "Hello" << std::endl;
```

Example4:
```
std::cout << "Hello" << 4 << std::endl;
```

Input in C++

- For input from the keyboard in C++, use cin (pronounced "see-in").
- cin links to the input stream for the standard input device.
- Use the **extraction operator** >> with cin. >> is usually expressed as "get from".
- When you input a variable using cin, you get the variable from the standard input stream.
- cin has default formatting depending on the variable type.

Example:
```
int a
std::cin >> a;
```

Including libraries

C++ Essential Training, Chapters 3.2

- As with C, in C++ you use the #include pre-processor directive to include the content of header files in your program.
- Header files (*.h) may be standard to the language, or they may be written by users.
- Header files may contain declarations of constants, variables, types and functions.
- For example, the header file `iostream.h` contains `cout, cin, endl` and other declarations.
- Standard header files put identifiers in a namespace block called "`std`".
- If you want to use an identifier from a header file, it isn't enough to include the header file, you also need to tell the compiler that you are using identifiers from the namespace block.
- Do this by using the directive "`using`":

//informs the compiler you are using standard header

Example1:
```
#include <iostream>
 using namespace std;
 int a;
 cin >> a;
 cout << a << endl;
```

Example2:
```
#include <iostream>
 using std::cin;
 using std::cout;
 using std::endl;
 int a;
 cin >> a;
 cout << a << endl;
```

Do Now: Hello World

- Modify a 'Hello World' program by
  - Putting comments at the top of your program.
  - Replacing 'using namespace std' with 'using std::cout' and 'using std::endl';

Modify Hello World

- Modify your hello world program to write: "Hello *<..name..>*, welcome to my world!", using strings:
  - and the concatenate operator to create a new string containing the whole string given above,
  - join the words in the cout expression (as shown in example4 in Output in C++)

Format specifiers in C++

Let's say you want to impose your own formatting on a floating point number. With cout, you can use:

`setw(n)`: Sets minimum output width to n.

`fixed`: Specifies fixed precision.

`setprecision(n)`: Sets maximum number of **meaningful** digits to display both before and after the decimal point, or when used with fixed, setprecision specifies exactly how many digits to display after the decimal point.

Example:

```
// setprecision example
#include <iostream>     // std::cout, std::fixed
#include <iomanip>      // std::setprecision

int main () {
double f =3.14159;
std::cout << std::setprecision(5) << f << '\n';
std::cout << std::setprecision(9) << f << '\n';
std::cout << std::fixed;
std::cout << std::setprecision(5) << f << '\n';
std::cout << std::setprecision(9) << f << '\n';
return 0;
}
```

Output:

```
3.1416
3.14159
3.14159
3.141590000
```

Do Now: Demo2

- In ACS223 MOLE, in the C++ folder, download the file ACS223Demo2.cpp
- Read, run and understand the program.
- Modify the program by setting the precision to 1. What happens?
- Input 0.0001 to the program. What happens?
- Play around with setw(n) y modifying n.
- Modify the program by removing "fixed" from the cout operation. What happens now when you enter 0.0001? Try entering 10.00001, what happens then?

Do Now: Demo2a

Write a program, using setw(n) to display the following (hint: matrix elements are row*column):

```
   1    2    3    4
   2    4    6    8
   3    6    9   12
   4    8   12   16
   5   10   15   20
   6   12   18   24
   7   14   21   28
   8   16   24   32
   9   18   27   36
  10   20   30   40
```

Blocks and Scope in C++

- A block is the set of statements inside a pair of braces { }. For example, this is a block:

```
{
        cin >> x[i];
        cout << "Your input was: " << x[i];
        i++;
}
```

- An identifier has scope in the block in which it is declared.
- An identifier's scope is the area of the program in which the identifier is known.
- Blocks may be nested, e.g.:

```
{  // start of outer block //
        cin >> x[i];
        cout << "You entered " << x[i] << endl;
        if( x[i] < 0.0 )
        {  //start of inner block //
           cout << "x[i] is negative" << endl;
        } // end of inner block //
}  // start of outer block //
```

- If block B is nested within block A, identifiers that are in scope in block A are also in scope in block B, with the following exception:
  - Inner blocks may override declarations of variables by re-using names in declarations.

Do Now: Demo3

- In ACS223 MOLE, in the C++ folder, download the file ACS223Demo3.cpp
- The program inputs a string from the keyboard and echoes it to the screen along with the length of the string. The program loops until the string is '0'.
- What happens when you try building the program? Find out why it doesn't work and fix this problem.

File I/O in C++

- C++ doesn't use the filehandle "FILE" for file I/O as C does.
- Instead, C++ defines two data types for file I/O:
  `ifstream` - to manage the stream of characters input from a file
  `ofstream` – to manage the stream of characters output to a file
- Include `ifstream` and `ofstream` from the header file `fstream.h`, i.e.
  `#include <fstream>`
- Here is an example of declaring filestreams:
```
ifstream myInputFile;
ofstream myOutputFile;
```
- Opening and closing filestreams is done with methods that are part of the `ifstream` and `ofstream` data types:
```
ifstream myInputFile;
ofstream myOutputFile;
myInputFile.open( .... some file name .... );
myOutputFile.open( ... some file name ... );
... some program statements ...
myInputFile.close();  // the close method has no arguments
```
- A filestream remains open in the block in which it is declared, and the program control automatically closes the filestream on exiting the block.
- Filenames that are passed as arguments to the `open` method of a filestream can be either string constants or C string variables.
- Use the `c_str` method for a string variable to convert it to a C string variable.
- Example:
```
ifstream myInputFile;
myInputFile.open( "myImage.bmp" ); // send string constant to open
```
Or
```
string filename;
cout << "enter filename:  ";
cin >> filename;
myInputFile.open( filename.c_str() ); // send C string variable to
    open
```
- Having declared and opened an input or output filestream, this can be used for file I/O in an analogous way to using `cin` and `cout` for keyboard input and screen output. Example:

```
ifstream myInputFile;
string data;
myInputFile.open( "myImage.bmp" );
myInputFile >> data;
```

(Be sure to include `fstream.h, iostream.h` and `string.h`)

Details: if you use a string to hold a filename, you need to convert this to a C String when using the string as an argument to the open function:

```
ofstream aFile;
string aFilename;
char aChar = 'a';
cout << "enter filename: ";
cin >> aFilename;
```

```
aFile.open( aFilename.c_str() );
```

```
// Now put a single char to the file with the "put" method.
// The analagous input method is "get".
aFile.put (aChar);
```

Do Now: Demo4

- In ACS223 MOLE, in the C++ folder, download the file ACS223Demo4.cpp.
- Read, run and understand the program.
- Now expand the program to
  - ask the user to input a new filename
  - open the file
  - read the information in myfile.txt character by character
  - write the information that you just read, character by character to the new file.

Functions in C++

- As with C, C++ programs declare functions using a function prototype, typically at the head of a program or in a header file.
- The function body comprises the statements within the function, and this can be placed after main() or in a separate file.
- The parameter list in a **function prototype** can show simply data types: data names are optional, example: `void myFunction(int );` .
- The parameter list in a **function body** must contain both data types and data names, example: `void myFunction(int myVar){…}` .
- In C, function parameters are passed by value unless they are arrays. If the calling program needs to see changes to function parameters, then the parameters must be sent as pointers (pass by address).
- **In C++, function parameters can be passed by reference or by value.**

Pass by reference

C++ Essential Training, Chapters 1.7 and 1.8

- If a variable in a parameter list is declared as a reference variable, the variable can be used to return values to the calling program.
- In C++, a reference parameter is an address (memory location) so that the function can change the contents at that address, whereas a value parameter copies the value to a new address in scope only in the function.
    - A "`&`" at the end of a data type in the parameter list of a function prototype and definition creates a reference parameter

      example: `void myNewFunction(int& myNewVar){…}`

Function Signature

C++ Essential Training, Chapters 2.7

In C++, the function signature includes the function's return type, and the number and types of input arguments.

Example:
```
double grade(double , double , double);
double grade(double , int );
```
are legitimate function declarations since their signature is different (even though they have the same function name `grade`- this is known as **function overloading**).

Do Now: Demo5

- In ACS223 MOLE, in the C++ folder, download the file ACS223Demo5.cpp.
- Read, run and understand the program.
- Now expand the program by creating a new function that halves "value" and uses a **reference parameter** to return the halved value.

Object-oriented programming

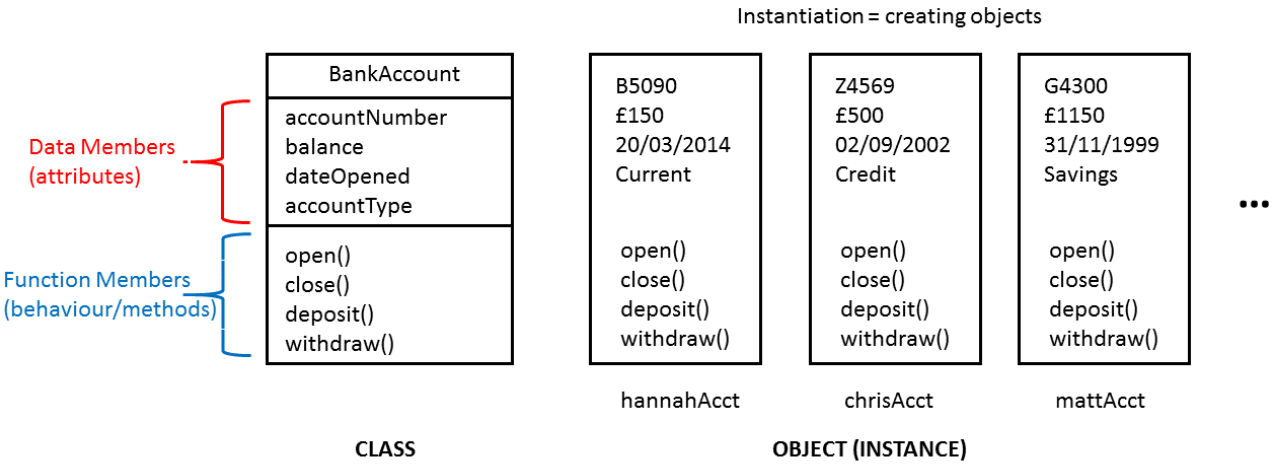
lynda.com

Foundations of Programming: Object-Oriented Design, Chapters 1.1 - 1.4
C++ Essential Training, Chapters 4.1, 4.2 (4.3, 4.4 if needed)

Structured data types

- In C and in C++, a structured data type has a collection of component items.
- A simple data type is a single data item.
- In C, the structured data types are:
  `array`
  `struct`
  `union`
- C++ has the structured data types `array`, `struct` and `union`, and also has the structured data type: `class`. This `class` data type allows object-oriented programming.

Classes and Objects



Creating a class

- **Name (Type)**: what is it? Example: `BankAccount`
- **Attributes (Properties)**: what describes it? Example: `accountNumber, balance`
- **Behaviour (Operations or Methods)**: what can it do? Example: `open()`, `withdraw()`.

ADT

- A class is a structured, abstract data type (ADT). A class combines abstract data with abstract control (abstraction).
- Encapsulation: data and actions are bound together in a class.
- In a class, a set of data has associated operations that are permitted on it.

Class members

- Classes have members.
- Class members can be either data or functions.
    - Data members are **attributes.**
    - Function members are **methods/behaviour**.
- Members can be accessed from outside of a class if and only if they are public members.
- The default is that members will be private.
- The members of a class are in scope in the class and not outside of the class.

Properties

The class ADT has

- Specification
- Implementation
- State
- Responsibilities

Responsibilities

- The responsibilities of a class are implemented in its methods (member functions).
- Responsibilities can be
    - Action – in which the state of an object is changed.
    - Knowledge – in which information related to the state (i.e. values of the attributes) is reported.

Specification and implementation

- A class **specification** is like a function prototype: you define the name of the class and the members (data and function) within it, and typically goes in a header (e.g. bankAccount.h) file.
- In addition to a specification, you also need to show the **implementation** of a class, showing the function bodies for all member functions, and typically goes in a separate c++ file (e.g. bankAccount.cpp)

Include guard: lynda.com **C++ Essential Training, Chapter 9.4**

Example class specification

```
#ifndef GUARD_bankAccount_h       // include guard, which includes bankAccount.h
#define GUARD_bankAccount_h      // only once

// header file bankAccount.h
// this is the specification of the class bankAccount

#include <string>
```

```
class bankAccount
{
private:
    double balance;

public:
    // Constructors- have same name as class
    bankAccount();
    bankAccount(double init_balance);


    // Public function
    void withdraw(double amount);
    void deposit(double amount);
    double getBalance();
};
#endif      // end include guard
```

Example class implementation

```
// implementation file bankAccount.cpp
// this is the implementation of the class bankAccount


#include "bankAccount.h"


// constructor 1
bankAccount::bankAccount()
{
    balance = 0.0;
}
// constructor 2
bankAccount::bankAccount(double init_balance)
{
    balance = init_balance;
}
// public functions
void bankAccount::withdraw(double amount)
{
    balance = balance - amount;
}
void bankAccount::deposit(double amount)
{
    balance = balance + amount;
}

double bankAccount::getBalance()
{
    return balance;
}
```

Objects

- An object is a specific instance of a class.
- The internal **state** of an object is determined by the value of its attributes.

Example of using a class

```
#include "bankAccount.h"
#include <iostream>      //std::cout, std::fixed, std::endl
#include <iomanip>       //std::setprecision

using std::cout;
using std::endl;
using std::fixed;
using std::setprecision;

// main program to show use of bankAccount class
int main()
{
    bankAccount myAcct(500);   //  instantiation of the object
    myAcct, and initialisation to £500


    cout << fixed << setprecision(2) << myAcct.getBalance() << endl;
    //balance is £500

    myAcct.deposit(250);
    cout << fixed << setprecision(2) << myAcct.getBalance() << endl;
    //balance is now £750

}
```

Do Now: Demo6

- From the C++ folder in ACS223 MOLE, download the file ACS223Demo6.cpp to a new folder. Also copy student.h and student.cpp to this folder.
- Read and understand the program.
- Create a project in CodeBlocks; the project should contain the three files you downloaded. Build and run the program.
- Expand the "student" class to include the attribute studentLevel, which should take the values "UG1", "UG2", "UG3" or "UG4". Create a new method (function) to set the value of this attribute, and change the writeRecord method (function) to include this new attribute in the record that it writes. To do these things, you will need to modify student.h and student.cpp. Then, modify ACS223Demo6.cpp to invoke the new method. Compile, build and test.

Array of objects

In C++ you can have an array of objects, e.g.:

```
class student
{
private:
            ....
public:
            ...
};
```

And then in main(), you'll need a declaration like this:

```
student classList[CLASS_SIZE];
```

Public methods in the object can be accessed by including an index into the array, e.g.:

```
classList[i].setName( name );
```

where `i` is a loop variable.

Example of using an array of classes/objects (using bankAccount class of previous example)

To instantiate more than one object you can use `bankAccount myAcct[3]` which instantiates 3 objects myAcct of class bankAccount. To access each object separately use `myAcct[i].someMethod()` where i is a loop variable having values 0, 1, and 2.

```
#include "bankAccount.h"
#include <iostream>      //std::cout,std::fixed,std::endl
#include <iomanip>       //std::setprecision

using std::cout;
using std::endl;
using std::fixed;
using std::setprecision;

// main program to show use of bankAccount class
int main()
{
    bankAccount myAcct[3]={bankAccount(100), bankAccount(200),
    bankAccount(-100)};   //  instantiation of 3 objects myAcct, and
    initialisation of each object
    int i;


    cout << "The initial balance in the accounts is:" << endl;
    for (i=0;i<3;i++){
        cout << fixed << setprecision(2) << myAcct[i].getBalance()
    << endl;
    }

    cout << "Depositing '£'100 pounds into all 3 bank accounts" <<
```

```
    endl;
    for (i=0;i<3;i++){
        myAcct[i].deposit(100);
    }

    cout << "The final balance in the accounts is:" << endl;
    for (i=0;i<3;i++){
        cout << fixed << setprecision(2) << myAcct[i].getBalance()
    << endl;
    }
}
```

Do Now: modify Demo6 to produce Demo7

- Copy the ACS223Demo6.cpp program that you produced and save it in a new folder as ACS223Demo7.cpp.
- As shown above, declare an array of objects:

  `student classList[CLASS_SIZE];`

  where CLASS_SIZE is a constant int that you set to some value, e.g.:

  `const int CLASS_SIZE=3;`
- Modify the main program you wrote for demo 6 (now in ACS223Demo7.cpp) to work with the elements of the array "classList" instead of a single item, that is, you are now going to print to screen the record of CLASS_SIZE number of students instead of just one student.

Composition relationship

Foundations of Programming: Object-Oriented Design, Chapter 6.5

Composition describes an **"has a"** relationship between objects, for example:

A PC **has a** motherboard.

A class list **has many** students.

- The composition relationship between classes can be realised in C++ by creating a data member (attribute) that is an instantiation of another class.
- Composition **implies ownership**, and when the owning object is destroyed (eg, classList) then so are the contained objects (eg, student).
- Make sure to include the specification file for the contained class (eg, student.h) in the specification (.h header file) and implementation (.cpp file) for the containing/owning class (classList).

Example (specification file for owning/containing class):

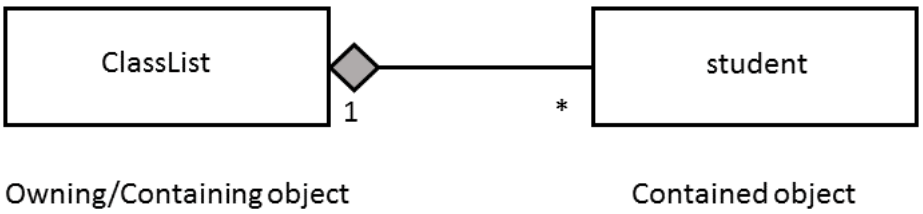
```
// header file classList.h
// this is the specification of the class classList
#include "student.h"
#include <string>
#include <iostream>


using namespace std;


class classList
{
    private:
      student studentList[10]; // this is an instantiation of the student class, inside
    classList
      int listPointer; // to keep track of the last item in the list


  public: // Public functions
      classList(); //Constructor

     void appendStudent(string, int, char ); // to append a record to the list
      void writeRecord(int );  // to write out a single record from the list
};
```



Do Now: Demo 8

- In ACS223 MOLE, in the C++ folder, download the file ACS223Demo8.cpp. Also copy student.h, student.cpp, classList.h and classList.cpp from this folder.
- Read, run and understand the program.
- Expand the "classList" class to include a new method (function) to insert a new record at the top of the list, having first moved all existing records down by one. To do this, you will need to modify classList.h and classList.cpp. Then, expand ACS223Demo8.cpp to invoke the new method. Compile, build and test.

Inheritance relationship

lynda.com **Foundations of Programming: Object-Oriented Design, Chapter 6.1**
C++ Essential Training, Chapters 7.1 - 7.3

Inheritance describes an **"is a"** relationship, for example:

A savings account is a type of bank account.

A current account is a type of bank account.

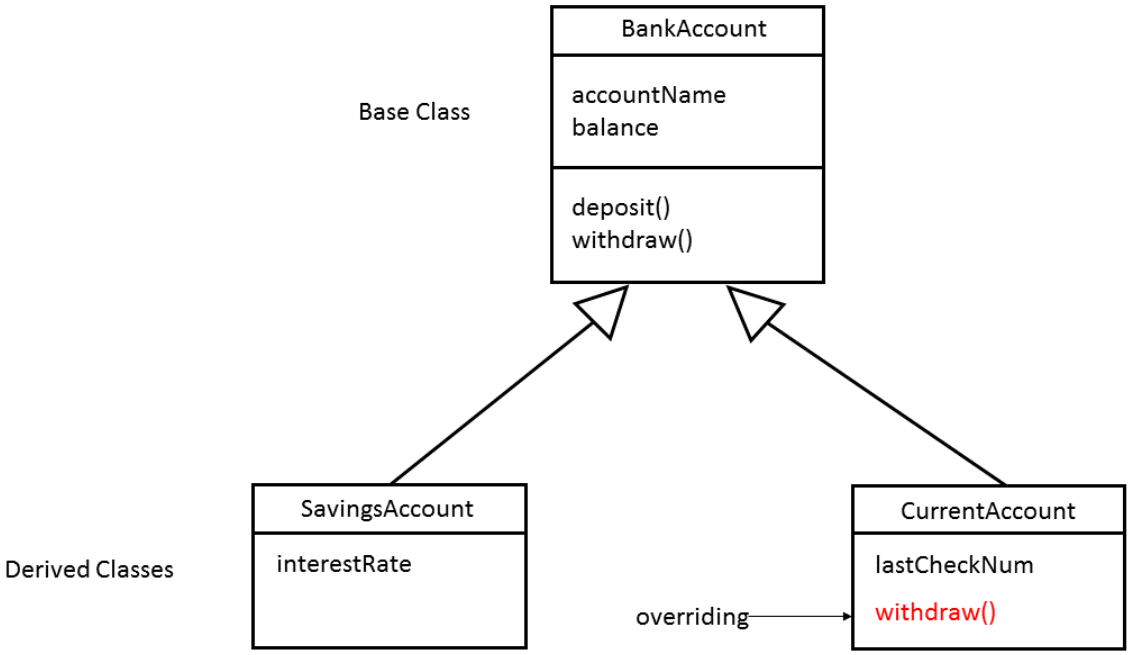
So the savings and current accounts are derived from bank account.

- The inheritance relationship between classes can be realised in C++ by creating a derived class from a base class.
- The derived class inherits all attributes and methods from the base class.
- If you change the base class, your changes will also have effect in the derived class.
- The derived class does not have access to private members in the base class: add public methods to the base class to get private attributes and use these in the derived class.
- In the specification file for the derived class, be sure to include the specification (header) file for the base class.
- Syntax for declaring the derived class:

```
class <derived class name> : public <base class name>
{
        public:
        ...
        private:
        ...

};

e.g.
class SavingsAccount : public BankAccount
{
...
};
```



Do Now: Demo 9

- In ACS223 MOLE, in the C++ folder, download the file ACS223Demo9.cpp. Also copy student.h, student.cpp, firstYearStudent.h and firstYearStudent.cpp from this folder.
- Read, run and understand the program.
- Create a new class "secondYearStudent" which is a class derived from the base class "student". The derived class should have these new private attributes:

  float firstYearWgtMean; // will be weighted mean of first year module marks
  char studyAbroadInYear3; // will be Y, N, or U, for Yes, No or unknown
- Provide public methods to set and get the new private attributes.
- Modify ACS223Demo9.cpp to demonstrate simple use of the new class "secondYearStudent".

Pointers

- As in C, a pointer in C++ contains the address of another variable.
- Pointers are declared with a * next to the type name, e.g.:

```
int array[100];
int* ptr;
ptr = array; // ptr now points to the beginning of the array
// note this is equivalent to ptr = &array[0]
```

- Pointers can be dereferenced with the unary * operator, e.g.:

```
int value;
value = *ptr; // dereferences: gets the value at the address
```

Example:

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

void main()
{
    int array[10] = {0,10,20,30,40,50,60,70,80,90};
    int* ptr;
    int i;

    cout << "Using ordinary indexing: array contains " << endl;
    for( i = 0; i<10; i++ ) cout << array[i] << " ";
    cout << endl;

    ptr = array; // now ptr points to the start of the array
    cout << "Using pointer addressing: array contains " << endl;
    for( i = 0; i<10; i++ ) cout << *(ptr + i) << " "; // ptr + i accesses
    the ith element, and use * to dereference
    cout << endl;
}
```

Summary

This has been a presentation of the main differences between C and C++, with opportunities to practice the different features. Example programs have been provided in ACS233 MOLE in the C++ folder, and you also should now have a collection of C++ programs that you have written. These resources should help you understand object oriented design principles, and also help you implement the software for your project in ACS223.