SENG1120/SENG6120
DATA STRUCTURES

Lecturer:

- Dr. Alexandre Mendes (Callaghan)
  - room ES236 – ES Building
  - phone 4921 6172
  - email alexandre.mendes@newcastle.edu.au

Lectures:

Monday 3 - 5 pm, room **BAG01**
Workshops in **ES409** (five possible times).

Assignments:

3 assignments, deadlines in course outline.

LATE ASSIGNMENTS will lose 10% per day or part thereof.

Assessment:

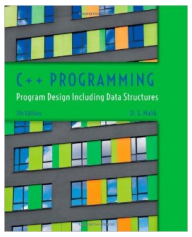
- Assignments are worth 10%, 15% and 15%.
- Mid-semester exam is worth 10%.
- Final exam is worth 50%.

Potential Workshop Times

- Lab ES409 has been booked for the following times:
  - Mon 10am-12pm
  - Tue 12pm-2pm
  - Tue 2pm-4pm
  - Wed 1pm-3pm
  - Wed 6pm-8pm
- Lab sessions are not compulsory, but attendance is highly recommended.
- Students who don't attend to the labs and/or do not complete the tasks will NOT lose marks.

Textbooks

- D.S. Malik. "C++ Programming: Program Design Including Data Structures", 7th Edition, Cengage Learning, 2014.
- This book is available at the bookstore in the Union Building.



Teaching Assistants

- Three people have agreed to be teaching assistants. They are:
  - Ms. Lauren Carter
  - Mr. Daniel Bell
  - Mr. Timothy Pitts
- Laboratories commence in week 2 of semester

Programming Language

- This course is taught using the programming language C++
  - Your previous programming instruction was in Java, so this represents a change of language
- You can install public domain C++ on your home computer. Options include:
  - Cygwin (www.cygwin.com) which implements a Unix emulation environment for Windows PCs.
  - Mac users can download the Mac OS X Developer Tools from the Apple site. These include the GNU compiler for C++
  - In ES409, we will use Cygwin and your assignment must be Cygwin-compatible.

Development Environment

- Examples provided in class will comply with GNU C++ and are compiled using the compiler called g++
- All labs and assignments must be runnable in gnu C++, which means, **NON-STARDARD LIBRARIES SHOULD NOT BE USED!!!!**
- *** Assignments will be marked as per above. If you use an IDE to do your work, ensure that your submission is compatible with the tools that will be used to mark it ***

Development Environment

- Computers in ES409 have cygwin and Visual Studio installed. If you develop your code using Visual Studio, NetBeans, Eclipse, Borland C++, etc, just make sure it compiles and runs under cygwin.
- This is VERY important. When marking, if your code does not compile and runs, you will likely lose 50% of the marks straightaway, because we won't be able to test it.
- Plagiarism:
  - The University has a strict policy on copyright and plagiarism (see the Course Outline which includes a link to the Policy)

What is C++?

- A programming language based on the language "C" which was developed by Dennis Richie in 1972.
- C++ was created by Bjarne Stroustrup in 1979.
- C++ *is not* C. It includes modern constructs that provide support for object-oriented programming techniques
- C++ had a big influence on James Gosling's development of the Java programming language.

Important differences between C++ and Java

- Java does automatic garbage collection. This is not the case for C++, resulting in the possibilities of "memory leaks" and existence of pointers to non-existent (or worse still not intended) objects.
- Java does array bound checking. C++ does not check array indexes by default, and this has been used by hackers to overwrite data values and code, and to obtain data.
- All Java variables are initialised on creation (zero for primitive types and null for object references). This is not the case for C++. Additionally non `void` functions are not checked to ensure they return something.

Program Template

- A simple C++ program is shown below:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello" << endl;
    return 0;
}
```

- The `include` directive in the first line tells the compiler to insert code from another file.
  - In this case the `< >` says the filename is a *system file*
  - Use of `" "` would indicate a *user-defined file*
  - The `#` symbol indicates that this is a pre-processor directive
- The `using` directive is like an `import` in Java. This directive makes the `std` classes available (equivalent to the automatic availability of `java.lang.*` in Java)
- The standard output stream is `cout`.The program directs the message followed by an end-of-line character (or "`\n`") to output
- The `return` informs the OS that all went well

Compiling & Running a Program

- To compile the previous program, (assuming it is stored in a file called `test.cpp` you would type:

```
g++ -o test test.cpp
```

  - Alternately a `Makefile` could be used
    - `make test` or simply `make`, depending on the `Makefile`
- This would produce a file called `test.exe` (in Windows) or `test.out` (in Unix)
- The program would be run in the usual way
  - Though it may be necessary to provide the path, e.g. `./test`

Makefiles

- A sample `Makefile` is provided on Blackboard

```
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=test.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=test

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
    $(CC) $(CFLAGS) $< -o $@

clean:
    rm -rf *.o core
```

- This can be used as is with the command `make` after a simple edit that provides the names of the input and output files
- The files to be compiled are listed under "sources" and the name of the resulting executable is next to "executable"
- Note the use of labels. E.g. the command `make clean` will "clean" the project, removing temporary files or files that won't be necessary after the executable is created.

Primitive Types for C++: Integers, Floating Point Numbers & Booleans

- Java provides 8 primitive types with precise sets of possible values spanning all platforms
- C++ has many primitive types with ranges dependent on the host architecture and OS
- Examples of C++ primitive types include:
  - The basic integer type `int` for which machine representation can be 16, 32 or 64 bits, dependent on host architecture. The keywords `short` and `long` can be added to `int`, with `int` never being shorter than a `short int` (or `short`) or longer than a `long int` (or `long`). `long` constants are indicated as, for example `1000L`.
    - The modifiers `signed` and `unsigned` can also be added to `int` and `char`, extending the range of possible values in the case of `unsigned int`, and forming an 8-bit byte in the case of `signed char`

Primitive Types for C++: Integers, Floating Point Numbers & Booleans

- Java provides 8 primitive types with precise sets of possible values spanning all platforms
- C++ has many primitive types with ranges dependent on the host architecture and OS
- Examples of C++ primitive types include:
  - The floating point types `float` and `double`. The standard does not require constants for negative and positive infinity as it does for Java
  - The boolean type is `bool` which has values `true` or `false`.

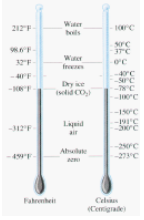
Software Development

- The software development process includes the following phases:
  - Task specification
  - Solution design
  - Implementation of the solution (coding)
  - Analysis of the solution
  - Testing and debugging
  - Maintenance and evolution
  - Obsolescence
  - plus continuous DOCUMENTATION
- The phases may be performed together, and sometimes in a different order. Sometimes you will revisit phases and do them again.
- You are expected to practice some of these phases through this course (and beyond)

Sample Problem

- Let us use a simple problem to demonstrate the stages:
- "Given an interval set by the user (e.g. -10, 40), display a table for converting Celsius temperatures to Fahrenheit as displayed below":

```
CONVERSIONS FROM -10.0 to 40.0C
   Celsius      Fahrenheit
   -10.0C       Display degrees
   -9.0C        F in this column
   .            after they
   .            have been
   .            computed
   35.0C
   36.0C
   37.0C
   38.0C
   39.0C
   40.0C
```



The algorithm

- This is a set of instructions for solving the problem
  - Typically expressed in pseudocode, a mixture of English and program code
- Designing the algorithm involves decomposing the problem into sub-tasks
  - Then treating each sub-task as a problem that can be broken into sub-tasks
- Eventually the sub-tasks become trivial enough to be easily coded
- For example, our problem has the sub-tasks:
  - Input the temperature range from the user
  - Loop through the input range
  - Convert a temperature from degrees C to degrees F
  - Print the line of the conversion table

Pre- and Post-Conditions

- It is the most basic documentation for interface definition.
- When we define a function, we specify *how* it performs its task
- When we use a function, we are only concerned with *what* it does
- Documenting the *what* for a function is done using pre and post conditions
- A *pre-condition* states conditions that must be true when the function is called
  - Otherwise it is not guaranteed to perform correctly
- A *post-condition* states what will be true when the function call has completed.
  - Provided the pre-condition was satisfied and the function is correctly designed and implemented

Example of Pre and Post Conditions

- For our temperature conversion problem we could state the following:

```
double celsius_to_fahrenheit(double c);
// Precondition:  c is a celsius
// temperature that is not less than
// absolute zero (-273.15 degrees C)
// Postcondition: The return value is
// the input temperature c converted
// to Fahrenheit degrees
```

- As a first step in designing any function you should write out its signature (return type, name and parameter list followed by a semi-colon), then the pre and post conditions as comments
  - This may need modification if further development shows it is insufficient or incorrect
- The use of pre and post conditions is extremely important in team situations
  - Because it forms a contract between the user of a function and its writer

The STL and Standard Namespace

- C++ now has compiler requirements defined in the ANSI/ISO C++ Standard (American National Standards Institute/International Organization for Standardization)
  - Providing much better portability of C++ source code than was previously the case
- This standard includes the STL (Standard Template Library)
- Library facilities can be used after an *include directive* has been placed at the top of the file using the facility
- The items in the STL are identified using names in the *standard namespace* or `std`. Thus

```
using namespace std;
```

must follow the `include` directives

Declared Constants

- Declared constants are defined using the keyword `const` before the declaration, e.g.

```
const double LOW_TEMP_LIMIT = -273.15;
```

- Note the convention that capital letters are used for the constant names
- The value of `LOW_TEMP_LIMIT` can never change during program execution

Feedback to the Operating System

- A program's main function should end with a `return` statement that informs the operating system of its exit state
  - The OS can then take further actions based on the returned value, e.g. running a subsequent application or displaying an error dialogue
- Typically the return value `0` (zero) indicates successful execution (see the "hello world" example a few slides back.

Testing and Debugging

- When test data reveals erroneous output, the first step is to *understand why the bug happened*. Then *correct the known error(s)*. Finally *re-run all the test cases*.
- DO NOT:
  - Change suspicious code just hoping things will get better
  - Assume that even controlled rectification will not change things that previously worked properly
    - It is a truism that it is more important when performing maintenance to ensure that what previously worked continues to work than it is to ensure the new feature works properly.

Exception handling

- Similar to Java, but not as flexible due to the "everything goes" nature of C++.

```
try {
   // Try the program flow
}
catch(Argument)
{
   // Catch the exception
}
```

- You can catch any type in C++. It will just depend on what you "throw".

Exception handling

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

int main()
{
   int StudentAge;

   cout << "Enter student age: ";
   cin >> StudentAge;

   try {
         if(StudentAge < 0) throw 1;
         cout << "\nStudent Age: " << StudentAge << endl;
   }
   catch(int param)
   {
         if (param == 1)
         {
               cout << "Student age is negative" << endl;
         }
         return -1;
   }
   cout << "\n";
   return 0;
}
```

Exception handling

- Types of exception:
  - `throw`: generic exception; `catch(…)`
  - `throw <type>`: throws a type
    - `throw 5;  catch(int param)`
    - `throw "error"; catch(string param)`
    - `throw 'a'; catch(char param)`
- Handling `cin >>` exceptions
  - `if (cin.fail()) {throws;}`
  - `cin.clear()` clears `cin` so the program can continue.

Your first code

- Write a pseudo-code for the temperature transformation problem. Include error checking procedures.
- All elements required for the task are included in this lecture slides.
- Think about the documentation
  - Pre- and post-conditions
  - A brief explanation about the variables and constants
  - Any limitations

1. Input the temperature range from the user and a step for displaying intermediate values
2. Perform all error checking procedures
3. Display the labels at the top of the table
4. For each line in the table (start, intermediate and end temperature values), print the temperature in Celsius, then calculate its equivalent in Fahrenheit and print it.
5. Ask the user if he/she wants to repeat the operation.