ECE 250 Algorithms and Data Structures

Laboratory 0

Douglas Wilhelm Harder, Hanan Ayad and Tiuley Alguindigue
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

ECE 250 Labs

- Goal
  - To provide support for completing course projects.
- Lab Topics
  - Lab 0 – Introduction to software environment, testing tools and submission procedures.
  - Lab 1 – Lab 5
    - https://ece.uwaterloo.ca/~ece250/Labs/

- Lab Material is in your UW Desire2Learn accounts.
- Six labs and five project submissions.
  - Lab 0 (no submission)
  - Lab 1 – Lab 4 (submissions via your Desire2Learn accounts)

ECE 250 Labs

- Lab attendance is not mandatory but helps you complete your projects successfully.
- Arrive on time.
- Come prepared, so you can spend your time in the lab making progress on your project.
  - Pre-lab readings

ECE 250 Labs

- Use computers in the lab or your laptops.
- If using your laptops, and your OS is Windows, you will need to install:
  - **Windows SSH Client**– used for connecting to the linux server, and transferring files.
  - **Visual Studio 2008** – used as an IDE for C++

Both can be obtained by going to the IST software download page

Outline

- Unix basics
- C++ Basics
- Testing your project
- Project Submission
- Linked List Example

Unix and your choice of OS

- Project implementation in C++
  - Windows, MAC – with IDE's such as MS Visual Studio
  - Linux or Unix – can use department server ecelinux
- Project testing and submission
  - Unix – eceunix server and GNU g++

    You transfer your solution to eceunix, test it, and then build .tar.gz file for submission via Desire2Learn

Unix for Automated Marking

- Project automated marking scripts are run in ecelinux:

  Name and format of files must be as expected

  Files must be placed in base directory of submission (not under a subdirectory)

  Program must compile in eceunix

  Basic tests provided must run in eceunix

Automated marking results in a mark of zero when the requirements above are not met.

Unix for Project testing and submission

- Project testing and submission
  - Transferring files to and from UNIX
  - Package and compress a set of files in a tar.gz file
- Other basic Unix skills
  - Login
  - List files, change directory, edit files

Unix for Project Testing and Submission

Logging in :

- Log onto ECE UNIX

  Start→Programs→Internet Tools→Secure Shell Client

  (View http://ece.uwaterloo.ca/~ece250/Online/SSH/)
- The server name is ecelinux (not eceunix as in example in course webpage)
- Your login name is your UW User ID
- Your password is your Nexus password

Unix for Project Testing and Submission

Basic Unix Commands:

- To make life easier, use the *tc sh*ell:

```
{ecelinux:1} tcsh
```

  - This allows name completion (using Tab) and history:

- In Unix, *m*a*k*e a *dir*ectory `lab0` and *c*hange to that *d*irectory

```
{ecelinux:2} mkdir lab0
{ecelinux:3} cd lab0
```

- If you *l*i*s*t the contents of this new directory, you will see it is empty:

```
{ecelinux:4} ls
{ecelinux:5} history
```

Unix for Project Testing and Submission

- Project testing and submission
  - Transferring files to UNIX
  - Build a submission file
    - Package and compress a set of files in a tar.gz file

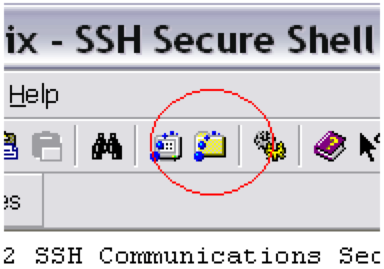
Introduction

Unix for Project Testing and Submission

Transferring files to UNIX:

Suppose that your solution to project 0 is the file Box.h.

- We will start with moving the solution file(s) to Unix
- From desire2learn, download the file Box.h and move it to your Desktop
- Launch the Secure FTP client
  - Click on the folder icon:



- In the right-hand panel, you will see the `lab0` directory
  - Double click on the folder icon to move to this directory
  - Drag the file `Box.h` to this directory

Unix for Project Testing and Submission

Building Submission File:

- List the contents of the directory lab0 again:

```
{ecelinux:6} ls
Box.h
```

- We will build a *g*nu-*zip*ped *ar*chive file with the solution to project 0

```
{ecelinux:8} tar -cvzf uwuserid_p0.tar.gz Box.h
```

uwuserid is your uw user id

- Now the directory will contain the file that you will submit for marking.

```
{ecelinux:6} ls
uwuserid_p0.tar.gz
```

Exercise 1(10 min)

UNIX and Tar command:

- Connect to the Unix server (ecelinux) and practice Unix basic commands.
- Transfer files from Windows to Unix.
  - Lab file is in LEARN
- Use tar utility to compress and decompress tar.gz files.
  - Decompress lab file
  - Make a new tar file that contains class Box.h (similar to what you will do for each of your projects)

C++ Basics – A Hello World Program

- Open sample file hello.cpp. We “ll examine the code line by line:

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

int main() {
   cout << “Hello world!” << endl;

   return 0;
}
```

Hello World! Program

- The first line includes the *i*nput/*o*utput *stream* header file
- This allows input from the keyboard and output to the console
  - C++ refers to such lines of communication as *streams*

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

int main() {
   cout << “Hello world!” << endl;

   return 0;
}
```

Hello World! Program

- This next line allows us to avoid statements like

```
std::cout << "Hello world!" << std::endl;
```

- A namespace is a software means of avoiding conflicting names
  - Critical in industry where a code base could have millions of lines of code developed by 100s of programmers all naming their function `init()`

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

int main() {
  cout << "Hello world!" << endl;

  return 0;
}
```

Hello World! Program

- We can compile the file using

```
{eceunix:10} g++ hello.cpp
{eceunix:11} ls
a.out    hello.cpp    ...other files...
{eceunix:12} ./a.out
Hello world!
{eceunix:13}
```

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

int main() {
   cout << “Hello world!” << endl;

   return 0;
}
```

*a*ssembler *out*put

Hello World! Program

- You may be wondering why we use `./a.out`
- Normally, Unix looks in the path for executables
  - The current directory (`.`) is not explicitly in the path
  - Thus we can explain:

`./a.out`

In the current directory... run the executable named `a.out`

- `a.out` is the filename obtained if we don't explicitly pass an *output* filename to the compiler
  - *E.g.*, `g++ -o hello hello.cpp`

Hello World! Program

- The file is the unit of compilation and when you run the resulting executable, the function `int main()` is where execution starts
  - It must return an integer (usually `0`)
  - Later we will see a file with multiple functions

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

int main() {
   cout << “Hello world!” << endl;

   return 0;
}
```

Hello World! Program

- Finally, we will look at the line which actually *out*puts to the *c*onsole:

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

int main() {
   cout << “Hello world!” << endl;

   return 0;
}
```

Hello World! Program

- The one line is actually a short form for

```
cout << “Hello world!”;
cout << endl;
```

- The second is a platform-independent *end*-of-*l*ine object
- The `operator <<` is said to be *overloaded*

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

int main() {
   cout << “Hello world!” << endl;

   return 0;
}
```

Exercise 2 (5 min)

Compile and execute the program hello.cpp in the ecelinux server, naming the executable “hello”.

Built-In Data Types

- We will consider some of the built-in data types, namely int, double, bool, and char
- We will look at the size (number of bytes) that each type occupies in memory
- `const` allows us define constants of a particular type
- We will look at `data_types.cpp`

Built-In Data Types

```
int  counter = 0;
bool approved = false;     // true or false
char letter = 'A';

// Constant (not assignable)
const double PI = 3.1416;

// Assigning values
counter  = 5;
approved = true;

// Printing values
cout << "counter:  " << counter << endl;
cout << "aproved:  " << approved << endl;
cout << "letter:   " << letter << endl;
cout << "PI:       " << PI << endl;

// Show size(number of bytes) that this type occupies in memory
cout << " size of integer: " << sizeof(counter) << " Bytes" << endl;
cout << " size of bool:    " << sizeof(approved) << " Bytes" << endl;
cout << " size of char:    " << sizeof(letter)  << " Bytes" << endl;
cout << " size of double:  " << sizeof(PI) << " Bytes" << endl;
```

Functions

- Next, let us look at the factorial function and *c*onsole *in*put

```
{eceunix:13} pico factorial.cpp
{eceunix:14} g++ factorial.cpp
{eceunix:15} ./a.out
```

- You can use Ctrl-c to terminate

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

int factorial( int n ) {
    if ( n == 0 ) {
        return 1;
    } else {
        return n*factorial( n - 1 );
    }
}

int main() {
    int value;

    while ( true ) {
        cout << "Enter a value: ";
        cin >> value;

        if ( value < 0 ) {
            break;
        }

         cout << value << "! = "
              << factorial( value ) << endl;
    }

    return 0;
}
```

Functions

- This is an example of the `main()` function calling another function defined within the same file
- Important: the function must be declared before it is called within the file
  - C++ does not *look ahead* in the file

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

int factorial( int n ) {
    if ( n == 0 ) {
        return 1;
    } else {
        return n*factorial( n - 1 );
    }
}

int main() {
    int value;

    while ( true ) {
        cout << "Enter a value: ";
        cin >> value;

        if ( value < 0 ) {
            break;
        }

         cout << value << "! = "
              << factorial( value ) << endl;
    }

    return 0;
}
```

Functions

- It is possible to declare a function without defining its operation
- See `factorial.declare.cpp`

```
int factorial( int );

int main() {
    int value;

    while ( true ) {
        cout << "Enter a value: ";
        cin >> value;

        if ( value < 0 ) {
            break;
        }

        cout << value << "! = "
             << factorial( value ) << endl;
    }

    return 0;
}

int factorial( int n ) {
    if ( n == 0 ) {
        return 1;
    } else {
        return n*factorial( n - 1 );
    }
}
```

Functions

- We will now look at the difference between pass-by-value and pass-by-reference
- Normally, when a variable is passed to a function, a copy of the value is sent to the function
  - The function can modify the copy of the value
  - The original is unchanged
- See `pass_by_value.cpp`

```
int f( int n );
```

Functions

- We can explicitly have a function call pass a reference to the original variable
  - Now the function can modify the original value
- See `pass_by_reference.cpp`

```
int f( int &n );
```

Exercise 3 (10 min)

Read the code in examples pass_by_value.cpp and pass_by_reference.cpp

Compile and execute the programs in the ecelinux server (use –o option to name both executable files).

Notice the difference in the value of the passed parameters after the call to function f().

Pointers

- We will now look at pointers
- A pointer is nothing more than a variable which stores an address
- Every variable must be stored somewhere in memory
- That memory must have an address so why can't we store that address?
- We will look at `addresses.cpp`

Pointers

```
// An integer
int counter = 1;

cout << "Counter" << endl;
cout << "Value:      " << counter << endl;
cout << "At address: " << &counter << endl;
cout << "Memory:     " << sizeof(int) << " B" << endl << endl;
```

Introduction

Pointers

```
// A pointer to an integer
int *ptr;

ptr = &counter;

cout <<       "The variable ptr = " << ptr << endl;
cout << "The value at that address is *ptr = " << *ptr << endl;

cout << "Updating the value stored at ptr..." << endl;
*ptr = 2;

cout << "The variable is unchanged: ptr = " << ptr << endl;
cout << "The value at that address is *ptr = " << *ptr << endl;
cout << "The original value is also changed: counter = "
     << counter << endl;
```

Dynamic Memory Allocation

- Memory for a single object may be dynamically allocated using `new` and deallocated using `delete`

```
int *ptr_my_int ;

ptr_my_int = new int(42);



*ptr_my_int = 256 ;




delete ptr_my_int ;
```

Dynamic Memory Allocation

- Memory for a single object may be dynamically allocated using `new` and deallocated using `delete`

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

int main() {
	int *ptr = 0;          // pointing to nothing

	cout << "The pointer is initially pointing to 0:  " << ptr << endl;

	ptr = new int( 42 );   // ask for new memory from the OS

	cout << "The pointer storing the address " << ptr << endl;
	cout << "The value stored there is " << *ptr << endl;

	*ptr = 256;

	cout << "The pointer is still storing the address " << ptr << endl;
	cout << "The value stored there is now " << *ptr << endl;

	delete ptr;            // give the memory back to the OS
	ptr = 0;               // set the pointer to 0

	return 0;
}
```

C++ Classes

```
class Box {
        private:
                int element;
        public:
                // Constructor
                Box();
                // Accessors
                int get() const;
                // Mutators
                void set( const int & );
};
```

```
Box::Box(){
  element = 0;
}
/* We could use this syntax as well to
define the constructor
Box::Box():element( 0 ) {}
*/

Box::~Box(){
  //empty destructor
}


Box::get(){
  return element;
}

Box::set(const int &e){
  element = e ;
}
```

C++ Classes

Basic concepts you will need for solving projects in this course:

- Member functions and member variables
- Private vs. Public visibility
- Constructors and Destructors
- Accessors and Mutators
- Operator override (ie. Overriding the = operator)
- Templates

C++ Tutorials are provided in course web site, to help you understand these concepts.

See http://www.ece.uwaterloo.ca/~ece250/intro/

A C++ Class

- Change the directory to the `Box` directory
- Here we see a project which is probably the most simple data structure in the world:
  - This data structure stores exactly one object

A C++ Class

- First we will examine `Box.h`
- This file is similar to the type of file you will edit for the rest of the projects

A C++ Class using templates

- At the top we see a class declaration for Box

```
template <typename Object>
class Box {
        private:
                Object element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                Object get() const;

                // Mutators
                void set( const Object & );
};
```

A C++ Class using templates

- This class is declared to be a template:

```
template <typename Object>
class Box {
        private:
                Object element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                Object get() const;

                // Mutators
                void set( const Object & );
};
```

A C++ Class using templates

- A template allows the user to decide what type an instance of this box will store

```
template <typename Object>
class Box {
        private:
                Object element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                Object get() const;

                // Mutators
                void set( const Object & );
};
```

A C++ Class using templates

- For example, if I declare
  `Box<int> my_box;`
  all instances of the symbol `Object` are replaced with `int`
- Think of `Object` as a placeholder

```
class Box {
        private:
                int element;

        public:
                Box();

                // copy constructor and
    assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
           int get() const;

                // Mutators
                void set( const int & );
};
```

A C++ Class using templates

- Similarly, if I declare

  `Box<double> another_box;`

  all instances of the symbol `Object` are replaced with `double`

```
class Box {
        private:
                double element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                double get() const;

                // Mutators
                void set( const double & );
};
```

A C++ Class using templates

- Anything member variables declared private may only be accessed by member functions (methods) of this class and *friends* of this class

```
template <typename Object>
class Box {
        private:
                Object element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                Object get() const;

                // Mutators
                void set( const Object & );
};
```

A C++ Class using templates

- The public member functions may be called by anyone

```
template <typename Object>
class Box {
	private:
		Object element;

	public:
		Box();

		// copy constructor and assignment
		Box( const Box & );
		Box &operator = ( const Box & );

		// Accessors
		Object get() const;

		// Mutators
		void set( const Object & );
};
```

A C++ Class using templates

- The public member functions are divided into a constructor,

```
template <typename Object>
class Box {
	private:
		Object element;

	public:
		Box();

		// copy constructor and assignment
		Box( const Box & );
		Box &operator = ( const Box & );

		// Accessors
		Object get() const;

		// Mutators
		void set( const Object & );
};
```

A C++ Class using templates

- The public member functions are divided into a constructor, copy constructors and assignment operator,

```
template <typename Object>
class Box {
        private:
                Object element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                Object get() const;

                // Mutators
                void set( const Object & );
};
```

A C++ Class using templates

- The public member functions are divided into a constructor,
  copy constructors and assignment,
  accessors (cannot change any member variables),

```
template <typename Object>
class Box {
        private:
                Object element;

        public:
                Box();

                // copy constructor and assignment
                Box( const Box & );
                Box &operator = ( const Box & );

                // Accessors
                Object get() const;

                // Mutators
                void set( const Object & );
};
```

A C++ Class using templates

- The public member functions are divided into a constructor, copy constructors and assignment, accessors (cannot change any member variables), and mutators

```
template <typename Object>
class Box {
	private:
		Object element;

	public:
		Box();

		// copy constructor and assignment
		Box( const Box & );
		Box &operator = ( const Box & );

		// Accessors
		Object get() const;

		// Mutators
		void set( const Object & );
};
```

A C++ Class using templates

- The constructor is called whenever a new instance of this class is created

```
// Object() creates a default instance of the type
// e.g., the default int and double are both 0
template <typename Object>
Box<Object>::Box():element( Object() ) {
        // empty constructor
}
```

A C++ Class using templates

- Recall we talked about passing-by-value
- This requires a *copy* of what we are passing

```
// This simply calls operator=
template <typename Object>
Box<Object>::Box( const Box<Object> &box ) {
        *this = box;
}

template <typename Object>
Box<Object> &Box<Object>::operator = ( const
    Box<Object> & rhs ) {
        if ( &rhs == this ) {
                return *this;
        }

        element = rhs.element;
        return *this;
}
```

A C++ Class using templates

- Similarly, we may have to deal with assignment

```
// This simply calls operator=
template <typename Object>
Box<Object>::Box( const Box<Object> &box ) {
        *this = box;
}

template <typename Object>
Box<Object> &Box<Object>::operator = ( const
    Box<Object> & rhs ) {
        if ( &rhs == this ) {
                return *this;
        }

        element = rhs.element;
        return *this;
}
```

A C++ Class using templates

- Similarly, we may have to deal with assignment

```
// This simply calls operator=
template <typename Object>
Box<Object>::Box( const Box<Object> &box ) {
        *this = box;
}

template <typename Object>
Box<Object> &Box<Object>::operator = ( const
    Box<Object> & rhs ) {
        if ( &rhs == this ) {
                return *this;
        }

        element = rhs.element;
        return *this;
}
```

ECE250 Software framework

- You write a class that implements data structure:

  I.e. in project 1, you a stack or a queue class.

- We provide you with a driver for I/0, a tester for your class, and some utilities (memory tracking class, and exception classes to handle errors)

Testing Options

You have two options for testing your classes:

1. Write your own test program, with a main() function.

   See the two files box_int.cpp and box_double.cpp

2. Use provided drivers

   One for int type - Box_double_driver.cpp and

   0ne for double type - Box_int_driver.cpp

Testing with Provided Driver

Using option 2 (as in the Automated marking tools):

To test a box able to store integers, you can:

1. Compile the int driver

   {ecelinux:1} g++ Box_int_driver.cpp -o Box_int_driver

2. Execute with command:

   {ecelinux:2} Box_int_driver

3. Enter valid commands at the prompt

Testing Options

Command examples:

- new
- get 0
- set 1
- get 1
- delete
- summary
- details

Exercise 4 (15 min)

1. Compile and execute driver program in directory Box, in both OS environments: UNIX and Windows.

   ecelinux: use command

   ```
   g++ -o Box_int_driver Box_int_driver.cpp
   ```

   Windows – Dev C++

   Create C++, empty project, and all files in Box folder. Use the "Execute" menu to compile.

   (you can also use Visual Studio, create Visual C++ , empty project)

2. Run the driver and try using commands shown in previous slide.

3. Run the driver and execute a series of commands saved in a file.

Repeating a Series of Commands

Create a file with commands (test case):
test.in

```
new
get 0
set 1
get 1
set 2
get 2
assign
  get 2
  set 3
  get 3
  delete
exit
get 2
delete
exit
```

In UNIX Execute with command:

```
./Box_int_driver < test.in
```

ECE250 Framework - Tester Class

- The available commands are listed at the top of the `Box_tester.h` file
  - `get n, set n, assign, summary, details`
  - The commands `new` and `delete` allocate and deallocate the objects we are testing.
  - Nested tests are performed using `assign` and `exit`

Automated marking

Uses provided drivers and compares actual output with expected output:

```
./Box_int_driver < test.in > test.out
```

Your solution passes a test if test.out matches exactly the expected output

(UNIX command `diff` creates no output when comparing the two files)

Your submission

Submit tar.gz file through your UW Desire2Learn account.

File must be named

uwuserid_pM.tar.gz

where uwuserid is your UW User ID, and M is the project number, and p and tar.gz are in lower-case characters.

How do I build a tar.gz file?

If you placed all the required files in directory lab0

You can issue the commands below:

In Unix server:

```
cd lab0
tar –cvzf uwuserid_p0.tar *
```

(the * stands for all files in the folder)

Exercise 5 (15 min)

A. Examine the files provided under directory “Box”

Drivers (int and double)
Tester (Class Tester and ancestor Tester)
ece250.h
Exception.h

What is the pourpose of Tester.h, ece250.h and Exception.h? Why are these files common to all projects?

B. File Box.h is the solution to project 0. It is the only file you will have to submit for marking, since all the other files(driver, tester and utilities) are provided.

1. Place the file Box.h in a tar.gz file named according to guidelines.
2. Submit this file via your UW Desire2Learn account, under the “dropbox” tab for lab 0.

Usage Notes

- These slides are made publicly available on the web for anyone to use
- If you choose to use them, or a part thereof, for a course at another institution, I ask only three things:
  - that you inform me that you are using the slides,
  - that you acknowledge my work, and
  - that you alert me of any mistakes which I made or changes which you make, and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides

Sincerely,
Douglas Wilhelm Harder, MMath
dwharder@alumni.uwaterloo.ca