ENSF 337: Programming Fundamentals for Software and Computer

Lab 7, Fall 2018 – Thursday October 25

M. Moussavi

Department of Electrical & Computer Engineering
University of Calgary

Important Notes:

- Due to upcoming midterm exam, we will not be able to mark this lab and return it back to you before November 1st. Therefor, this lab will not be marked. However, I strongly urge you to complete all the exercises in this lab, as the exercises are as important as any other exercises in the previous labs. Please notice that some of the materials in this lab are very essential for understanding important C++ basic concepts that if you don't learn them at this point you may face further difficulties to understand advanced topics that will be discussed in near future. Additionally, there is a possibility that questions related to the material in this lab appear in the upcoming midterm. Therefor you are strongly recommended to complete all exercises in this lab and check your solutions with the solutions that will be posted on the D2L by Monday Oct 29, at 5 PM. We will send you an email when the solutions are available on the D2L.

- Some of the material related to exercises C and D will be discussed on Friday Oct 26 and Monday Oct 29.

Objective:

Here is the summary of the some of the topics that are covered in this lab:

- More on C file I/O and Dynamic Allocation of Memory
- Introduction to C++ reference type
- C++ objects on the computer memory

Note: For C++ exercises in this lab, make sure to use C++ compilation command `g++` instead of `gcc`.

Exercise A:

Read This First – A Brief Note on File I/O in C

C programs consider all kind of files and devices for input and output as logical data streams, regardless of whether the program reads or write character, bytes, text line, or data as a block of bytes. Text stream is a sequence of characters read from or written into a file, while a binary stream is a sequence of a given size of bytes. The C I/O library provides several functions such as fopen, fclose, fscanf, fprintf, fread, and fwrite to allow file manipulations in a C-program. Using function such as fscanf, fprintf is not much different from standard input/output function printf and scanf, as they are all text stream functions. For the same reason fprintf and fscanf can be applied to standard I/O streams, using predefined FILE pointers: stadin, stdout, and stderr. stderr is similar to stdout (means both point to the standard output stream) except that stderr is meant to be used for error messages as we discussed briefly during lectures and tutorials.

Reading from or writing into binary file requires different type functions. First you need to use fopen in binary mode, "rb" instead of "r" for reading from file, and "wb" instead of "w" for writing into a file. Then you need to use library functions fread and fwrite for reading and writing. The way that fread and fwrite works is very different from the way text stream functions work..

In the following function prototype, fread reads up to n objects of size bytes from a binary stearm referenced by fp, and stores them in the array addressed by buffer:

```
size_t fread(void *buffer, size_t size, size_t n, FILE *fp);
```

Similarly fwrite function sends n objects of size bytes from array addressed by buffer to the output stream referenced fp. Here is the prototype of this function:

```
size_t fwrite (void *buffer, size_t size, size_t n, FILE *fp);
```

Notice that void* is data type that can hold the address of any type -- It is not a pointer to nothing.

As an example, lets assume we have an array of integer numbers called buffer, and a FILE pointer called fp pointing to a binary stream as follows:

```
int  buffer[6] = {200, 300, 100, 50, 2, -1};
FILE *fp = fopen("numbers.bin", "wb");
```

Now assuming file is successfully open and the value of fp is not equal to NULL, we can call function fwrite as follows to write the entire bytes of array buffer into the stream:

```
fwrite(buffer, sizeof(buffer), 1, fp);
```

fwrite returns the number of items that successfully write into the stream. Which in this case must be equal to one.

Or you can choose to write 6 items of sizeof (int), in the following form – the final result will be the same.
However the return value of fwrite in this case will 6, if the operation is successful.

```
fwrite(buffer, sizeof(int), 6, fp);
```

There is also a third way to get the same result:

```
for(int j = 0;  j < 6; j++){
   size_t n = fwrite(&buffer[i], sizeof(int), 1, fp);
}
```

In the above statement fwrite is called six times within a loop and each time returns 1 if successfully write one element of buffer into the stream. Please pay attention to address-of-operator (&) in front of each element of buffer array.

What To Do:

Download the file lab7ExA.c from D2L. Read the file carefully, and try to predict the output of the program. Then, run the program to compare the results with your prediction.
Your final task in this exercise is to draw AR diagrams for points **one, two,** and **three** indicated in the source code.

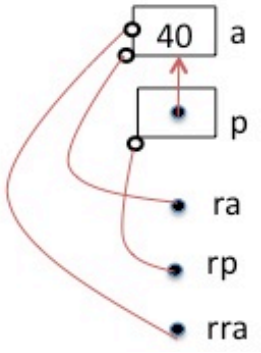
Exercise B:

Read This First:

The AR notations that we use in ENSF 337 to show C++ references are different from ordinary types such as: int, double, and pointer notations. When we declare a reference we just provide an alias name for another memory space. In other words, a reference in C++ doesn't have its own memory space; therefore we show them as a link (a line) between the reference-identifier and the actual allocated memory spaces. There are two little circles on both ends of these links. On one end there is a solid-black circle that represents the reference,

and on the other end there is an open circle that represents the actual allocated memory space. Here are a few examples:

```
int a = 40;
int*p = &a;
int& ra = a;       // ra is referred to integer a
int*& rp = p;      // rp is referred to integer pointer p
int& rra = ra;     // rra is also referred to a
```



Notice that all references `ra`, `rp,` and `rra` **must** be initialized with an expression that represents an actual memory space or another reference.

What To Do:

Download the file `lab7ExB.cpp` from D2L. Then, draw an AR diagram for point **one**.

Note: You don't need to compile and run this program but in case that you decided to run it, you should use the g++ command which is the command to compile C++ programs in our ICT 320 lab under the Cygwin. The executable file created will be still `a.exe,` by default.

```
g++ -Wall lab7ExB.cpp
```

Exercise C:

The objective of this exercise is to help you in understanding how C++ objects are shown on a memory diagram, and how a member function of a class uses the `'this'` pointer to access an object associated with a particular function call. For further details please refer to your lecture notes and slides.

What to Do:

Download files `cplx.cpp`, `cplx.h`, and `lab7ExC.cpp` from the D2L and draw AR diagrams for: **point one** and **point two.**

For this exercise you just need to draw the diagrams. However, if you want to compile and run it from command line in our ICT 320 lab, you should have all of the given files in the same directory and from that directory you should use the following command to compile and create an executable:

```
g++ -Wall cplx.cpp lab7ExC.cpp
```

Please notice that you shouldn't have the header file name(s) in this command.

Exercise D: Construction and Destruction of Objects

What to Do:

Compile the files `lab7ExD.cpp`, and `lab7String.cpp`, run the program and fill out the following table to indicate which object or which pointer (a, p, d, b, c, or g) is associated with the call to a constructor or destructor. As an example the answer for the first row is given – which indicates that the first call to the constructor (abbreviated as `ctor`) is associated with the `Lab7String` object, `a`.
Notes:

- Also many programmer use `dtor` as an abbreviation for destructor.
- When compiling this program, compiler may give you warning for unused variables. Although warnings are always important to be considered and to be fixed, but in this exercise we ignored those warnings to keep the program short and simple.

Program output in order that they appear on the screen	Call is associated with object(s):
`ctor called...`	`a`
`default ctor called...`	
`default ctor called...`	
`ctor called...`	
`ctor called...`	
`ctor called...`	
`The first four calls to dtor`	
`The last two calls to dtor`	