ENSF 337: Programming Fundamentals for Software and Computer
Lab 6 - Fall 2022

Department of Electrical & Software Engineering
University of Calgary

Written by: M. Moussavi, PhD, PEng

This lab will not be marked. However, the solutions will be posted on the D2L to allow you comparing your solutions with given answers and learn from your possible mistakes. Please notice that this lab is as important as any other lab and **questions regarding the material in this lab, which may appear in the final exam**.

Objective:

The main objective of this lab is to practice the subjects of File I/O in C, and learning about C++ reference type.

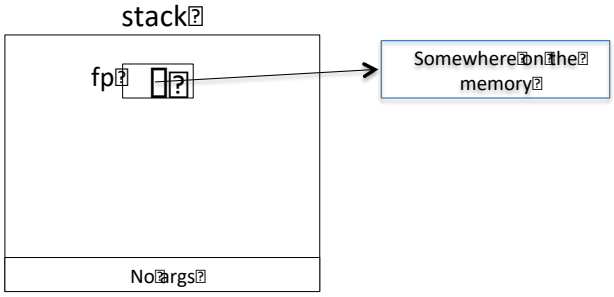
Exercise A: Draw AR Diagrams

What to Do:

Download file `lab6exe_A.c, lab6exe_A.h,` and `lab6exe_A.txt,` from D2L. This exercise receives its input from text file, `lab6exe_A.txt`, and populates an array of structure called Bits_Pattern with the available data in this file. Study the content of the downloaded files carefully to understand what the program does. Then draw a memory when the program reaches point one **for the second time**.

Note: in your AR diagram you don't need to show where exactly a FILE pointer points. Just mention it points to "somewhere on the memory" as following figure shows:

```
FILE * fp = fopen("a_file_name", "r");
```



Exercise B: Writing into a Text File

What to Do:

Download file `lab6exe_B.c, lab6exe_B.h`, and `lab6exe_B.txt`, from D2L. If you read the given files carefully you will notice that this program reads the content of the file `lab6exe_B.txt` into an array of integers that is declared within the body of a structure called `IntVector`. Then, the program displays the stored values of array on the screen in a single column format. Your task in this exercise is to complete the definition of the function called `display_multiple_column`. Please see the function interface comment for more details.

Exercise C: Allocation of Memory on the Heap

What to Do:

Download the file `lab6exC.c` and `lab6exC.h` from D2L. Read these files carefully and draw a memory diagram for point one, assuming that all the calls to the library function `malloc` succeed.

Exercise D: String Manipulation Using Dynamic Allocation

The main objective of this exercise is to give you the opportunity to practice dynamic allocation of memory for the purpose of string operations such as copying a string into another string, appending a string to the end of another string, or truncating the size of a string. The other objective of this exercise is to provide you with more practice on C structure type, and finally to take a very small step towards the concept of data abstraction.

Read This First:

Creating a kind of wrapper data type that encapsulates several related data is a common practice to build data structures such as vectors, string, linked list, trees in object-oriented programming languages such as C++. We will discuss the proper way of applying these concepts later in the C++ part of the course.

In this exercise, we define a structure type called String that simply contains two pieces of related data. First one is a pointer that is supposed to point to a dynamically allocated array of characters, used as a storage for a null-terminated c-string. Second one is an integer number that represents the length of the stored string in the first data member (count of character up to and excluding the '\0').

Here is the definition of this struct:

```
typedef struct String{
    char *dynameic_storage;
    int length;
} String;
```

What To Do:

Download the files `lab6exD.c` and `lab6exD.h` from D2L. If you read these files carefully, you will notice that the existing code contains the following functions plus main and a three other functions for testing operations: copy, append, and truncate:

```
void create_empty_string (String *str);
```
Creates an empty string that its dynamically allocated storage has onle one element holding a '\0' and length of zero.

`void String_cpy(String *dest, const char* src);`
Copies a c-string from src into the dynamically allocated storage of String object that dest points to.

`void String_copy(String *dest, const String* source);`
Copies a c-string from source into the dynamically allocated storage of String object that dest points to.

First you should compile and run the program and pay attention that how functions `String_cpy` and `String_copy` work. Please read the comments in the file `lab6exD.c`, for each call to the function `display_String`. The comments tells you what this function is expected to display if functions `String_cpy` and `String_copy` work properly. Here is a snapshot of the program output:

```
Testing String_cpy and String_copy started:
String is empty      0
String is empty      0
String is empty      0
String is empty      0
William Shakespeare      19
Aaron was Here!!!!      18
But now he is in Italy      22
String is empty      0
String is empty      0
But now he is in Italy      22
String is empty      0

Testing String_cpy and String_copy finished...
```

Your next job in this exercise is to write the implementation of two missing functions `String_truncate` and String_append, according to the given interface comments in the file `Lab6exD.h`. You are advised to write and test function `String_truncate` first, and then once this function works with no errors, start working and testing the other function, `String_append`.

To complete this task please take exactly the following steps:

1. In the main function, change the conditional-compilation-directive for the call to test_copying from `#if 1` to `#if 0`.
2. Read carefully the function interface comment for function `String_truncate` in the header file `Lab5exF.h` and write the implementation of this function.
3. Change the conditional-compilation-directive for the call to `test_truncating` from `#if 0` to `#if 1`.
4. Compile and run the program and make sure the test results matches as comments in the function `test_truncating` states for each call to the function `display_String`.
5. If you function `String_truncate` works fine (no errors), repeat steps 2 to 4 for function `test_appending`.

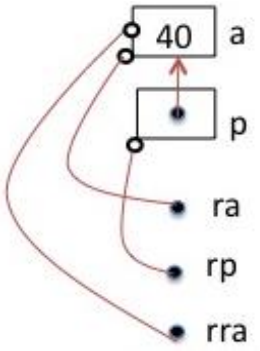
Exercise E – Moving from C to C++:

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 `lab6exE.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, if you are compiling and running at the command line, you should use the g++ command which is the command to compile C++ programs. The executable file created will be still `a.exe`, by default.

```
g++ -Wall lab6exE.cpp
```