ITF22519: Introduction to Operating Systems

Fall Semester, 2022

Lab3: C Programming 2

Submission Deadline: September 13rd, 2022 11:59

In this lab, you will do more practices with C programming in Linux. Contents of this lab are **Strings**, **Pointers**, **Dynamic Memory Allocation**, and **Structure**. Before you start, remember to commit and push your exercises in previous lab to your Git repository. Then, try to pull the new lab:

```
$ cd OS2022/labs
$ git pull main main
$ cd lab3
```

1 Strings

A string is an one-dimensional array of type `char`. In C, a string is terminated by the end-of-string sentinel zero-slash '\∅' or NULL character. It is a byte with all bits off. Because a string is an array, you can manipulate it on the same way with an array.

The followings are two declaration statements of a string. They are the same:

```
char s[] = "abcde";
```

and

```
char s[] =  {'a', 'b', 'c', 'd', 'e', '\0'};
```

In the above statements, `s` is a string, or an array of type `char`. The size of the array is 6; each of the element in the array is: s[0] = a, s[1] = b, s[2] = c, s[3] = d, s[4] = e but and s[5] = NULL-remember this.

1.1 Task 1: Calculate the number of a character in a string

The following code asks for a string input from a user and a character the user wants to find in the string. Complete the code to calculate the number of the character in the string. (Hint: you can use `strlen()` function in C. If you do not know what `strlen()` does, you can use command `man strlen` to check).

```
#include<stdio.h>
#include<string.h>

int main(){
        char s[100] = "";
        char c;
        int count= 0;

        printf("Enter a character to check:");
        scanf("%c",&c);
        printf("You have entered: %c",c);

        printf("\nEnter a string:");
        scanf(" %[^\n]s",s);
        printf("You have entered: %s\n", s);

          // YOUR CODE HERE

        printf("The character %c appears %d time(s)\n",c,count);
        return 0;
}
```

2 Pointer

A variable in a program is stored in a certain number of bytes at a particular memory location, called an *address*. A pointer is used to *point* to that *address*. It is also used to access the memory and to manipulate the address.
If v is a variable, then the operator $\&v$ gives the address of v in the memory.
If p is a pointer, then the operator $*p$ gives the value stored at address p.

Let see the following example:

```
#include <stdio.h>
int main(){
       int i = 7, *p = &i;
       printf("Value %d is stored at the address %p.\n", i, p);
       return 0;
}
```

Make your own *YourFileName.c* file, run the code, and see how it works. We are now going to do some same examples in Lab3 using pointers.

2.1 Task 2: Arithmetic operators using pointer

Write a C program that:

- Gets two integers from user input.
- Prints out where the two integers are stored in memory.
- Calculates their summation, difference, multiplication and division using pointer.

Sample output:

```
Enter one integer:
Enter another integer:
The first integer is stored at the address:
The second integer is stored at the address:
Summation:
```

```
Difference:
Multiplication:
Division:
```

Example:

```
Enter one integer:2
Enter another integer:4
The first integer is stored at the address:0x7fff7dbe359c
The first integer is stored at the address:0x7fff7dbe875d
Summation:6
Difference:2
Multiplication:8
Division:0.5
```

2.2 Pointer and String

A string constant is treated as a pointer whose value is the base address of the string. Look at the following examples:

```
#include<stdio.h>
#include<string.h>

int main(){
        char s[] = "abc1234";
        char *p = s;

        printf("%s\n", p);
        printf("%s\n", p+1);
        printf("%s\n", p+2);

        return 0;
}
```

In the statement `char *p = s`, the pointer `p` is assigned the base address of string `s`. This means that `p` points to the first element of the string which is `s[0]= a`. Therefore, the statement:

```
printf("%s\n", p);
```

causes `abc1234` to be printed.

In the statement

```
printf("%s\n", p+1);
```

`p+1` points to the element which is 1 different from `p` i.e, `s[1] = b`, causing `bc1234` to be printed. Similarly, in the statement

```
printf("%s\n", p+2);
```

`p+2` points to the element which is 2 different from `p` i.e, `s[2] = b`, causing `c1234` to be printed.

- Run the above code and see the output.

Note that, the following block of code:

```
printf("%s\n", p);
printf("%s\n", p+1);
printf("%s\n", p+2);
```

can be replaced by one statement:

```
printf("%s\n%s\n%s \n", p, p+1, p+2);
```

2.3 Pointer and Array

In C programming language, a pointer variable can take different addresses as values. In contrast, an array name is an address which is fixed.

Suppose that `A` is an array and that `i` is an integer number less than the size of array A, then the following expressions are the same: `A[i]` and `*(A+i)`. As you may remember from the previous lab, `A[i]` is the value of the element with index i in the array. `A(i+1)` can be considered as a pointer pointing the element `A[i]`. Therefore, the operator `*(A+i)` has the value of `A[i]`.

If `p` is a pointer then

```
p = A       equivalent to p = &A[0];
```

p points to the element `A[0]`.

```
p = A + 1   equivalent to p = &A[1];
```

p points to the element `A[1]`.
The following code shows how each element of the array A is stored in the memory.

```
#include <stdio.h>
int main(){
        int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int i = 0;
        for (i = 0; i < 10; i++){
              printf("The value of A[%d] is %d\n", i, A[i]);
              printf("It is stored at the address %p\n\n", &A[i]);
        }
        printf("The base address of Array A is %p\n", A);
        printf("That is the address of the first element in the array A or A[0]\n");

        return 0;
}
```

- Run the code.

By using different expression for Array with pointer, the following code will give the same output as the code above.

```
#include <stdio.h>
int main(){
        int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int i = 0;
        for (i = 0; i < 10; i++){
              printf("The value of A[%d] is %d\n", i, *(A+i));
```

```
            printf("It is stored at the address %p\n\n", (A+i));
        }
        printf("The base address of Array A is %p\n", A);
        printf("That is the address of the first element in the array A or A[0]\n");

        return 0;
}
```

3 Call-by-reference

Whenever variables are passed as arguments to a function, their values are copied into the corresponding parameters in the function and the variable themselves are not changed in the calling environment. This is called *call-by-value* mechanism.
In C, *call-by-reference* mechanism is a way of passing address (or reference) of variable to a function thanks to the pointers. For a function to be affected by *call-by-reference*, pointers must be used in the parameter list in function definition. Then, when the function is called, the address of variables must be passed as arguments. As a result, the variables are changed in the calling environment.

3.1 Task 3: Fixing bugs

The following program aims to change the value of two variables which are input from a user. However, the program does not run as expected due to *call-by-value* mechanism. Run the program to see how it works and then fix its bugs.

```
#include<stdio.h>
void change(int num1, int num2) {
     int temp;
     temp = num1;
     num1 = num2;
     num2 = temp;
}
int main() {
     int num1, num2;
     printf("\nEnter the first number: ");
     scanf("%d", &num1);
     printf("\nEnter the second number: ");
     scanf("%d", &num2);

     change(num1, num2);

     printf("\n\nAfter changing two numbers:");
     printf("\nThe first number is: %d", num1);
     printf("\nThe second number is: %d\n", num2);

     return 0;
}
```

4 Dynamic Memory Allocation

When we use Array, it is stored at a space in the memory location. The size of space is corresponding to the size of Array and is fixed. This way is not efficient when the number of actual elements is dynamic. C provides two functions to dynamically create space for arrays (also for structures, and unions) which are more efficient than Array. These two functions are **calloc()** (contiguous-allocation) and **malloc()** (memory-allocation). They are both in the standard library *`stdlib.h`*.

- calloc()

Example:

```
# stdlib.h
int *a;
int n;
scanf(\%d",&n);
a = calloc(N,sizeof(int));
```

This function allocates contiguous space in memory for an array of N elements and returns to a pointer of the allocated memory. The space is initialized with all bits set to zero.

- malloc()

Example:

```
# stdlib.h
int *a;
int n;
scanf(\%d",&n);
a = malloc(n*sizeof(int));
```

malloc() does all what **calloc()** does except that zero out all bytes in the allocated memory. Therefore, **malloc()** is faster than **calloc()**.

- free()

The allocated memory must be free after using. Use **free()** for this purpose.
Example:

```
free(a);
```

4.1 Task 4: Dynamic memory allocation with 1-dimensional array

The following code use `malloc()` to create a space for an Array. If `malloc()` does not return `NULL`, then `A` is the pointer pointing to the first address of memory block allocated to the Array.

```
#include<stdio.h>
#include <stdlib.h>
int main(){
       int n; // number of elements in the array
       int *A;

       // YOUR code to get array size, put it in variable n
       // End of array size
```

```
        // MY code for memory allocation
        A = (int*)malloc(n*sizeof(int));
        if (A== NULL){
                printf("Error in Memory Allocation\n");
                exit (0);
        }
        // END of my code

        // YOUR code to fill out elements of the array

        // YOUR code to print the array

        // Your code for memory deallocation

        // DONE!
        return 0;
}
```

Complete the code to:

- Get the size of the array from user input.
- Get all elements in the array from user input
- Use pointer to print all elements of the array into screen.
- Deallocate memory for the array.

4.2 Task 5: Dynamic memory allocation with 2-dimensional array

The following program dynamically allocates memory for a two-dimensional array. Complete the code to:

- Fill out all elements of the array by getting input from a user.
- Print all elements of the array into screen.
- Deallocate memory for the array.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
          int m, n;
          printf("Enter the nmuber of rows and columns for matrix: ");
          scanf("%d%d", &m, &n);
          int **a;
          //Allocate memory to matrix
          a = (int **) malloc(m * sizeof(int *));
          for(int i=0; i<m; i++){
                a[i] = (int *) malloc(n * sizeof(int));
          }
         //YOUR CODE HERE
         // Fill-out matrix from a user input
```

```
        // Matrix printing.

        // Memory deallocation

        // END OF YOUR CODE

        return 0;
}
```

5 Structure in C

C programming language allows you to define a new data types which are constructed from fundamental types. A **structure()** type is a type defined by a user and used to present heterogeneous data. It has `members` which are individually named. Structure provides a mean to aggreate variables of different types. Here is one example of a structure type in C:

```
typedef struct{
       double start_time;
       double execution_time;
} process_t;
```

The following code assigns the value for each member of a process from input files.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct Process{
   int id;
   int priority;
   int starting_Time;
   int execution_Time;
};

int main( ) {

   FILE* myFile1;
   FILE* myFile2;
   FILE* myFile3;
   FILE* myFile4;

   struct Process P[10];
   int i = 0;

   myFile1 = fopen("ID.txt", "r");
   if (myFile1 == NULL) {
      printf("Error Reading File\n");
      exit(0);
   }

  myFile2 = fopen("Priority.txt", "r");
```

```
   if (myFile2 == NULL) {
      printf("Error Reading File\n");
      exit(0);
   }

   myFile3 = fopen("Starttime.txt", "r");
   if (myFile4 == NULL) {
      printf("Error Reading File\n");
      exit(0);
   }
   myFile4 = fopen("Execution.txt", "r");
   if (myFile3 == NULL) {
      printf("Error Reading File\n");
      exit(0);
   }

   for (i = 0; i< 10; i++){
        fscanf(myFile1, "%d", &P[i].id);
        fscanf(myFile2, "%d", &P[i].priority);
        fscanf(myFile3, "%d", &P[i].starting_Time);
        fscanf(myFile4, "%d", &P[i].execution_Time);
   }

  for (i = 0; i < 10; i++){
       printf("Process %d has: ID: %d, Priority: %d, start at time %d, executes during %d seconds\n", i
   }
   return 0;
}
```

- Run the code
- Try to understand each statement in the code

6 Exercises

6.1 Exercise 1: (50 points)

Write a C program to:

- Get an integer number from user input as the size of an array.
- Get a number of integer numbers from user input and put the integers in the array. The number of integers is the size of the array.
- Use pointer to access array elements to print all the elements and their corresponding address in the memory into screen.
- From the output in the previous step, how many bytes are used to store an integer number?
- Use pointer to access array elements to find the minimum value in the array.

6.2 Exercise 2: (50 points)

In an input file `Input.dat` containing the integer numbers of a square matrix, the first element indicates the size of the matrix. Write a C programming to:

- Read the first element of the file `Input.dat` and assigns it to a variable `N`.
- Create a matrix A with the size `N` and dynamically allocates memory for A.
- Read the remaining elements of the input file and assign them to the corresponding elements of matrix A.
- Print matrix A into screen.
- Find the maximum number in matrix A.
- Deallocate memory for A.

7 What To Submit

Complete the **Exercises** in this lab, each exercise with its corresponding `.c` file. After that, put all of files into the **lab3** directory of your repository. Run `git add .` and `git status` to ensure the file has been added and commit the changes by running `git commit -m "Commit Message"`. Finally, submit your files to GitHub by running `git push`. Check the GitHub website to make sure all files have been submitted.