ITF22519: Introduction to Operating Systems

Fall Semester, 2021

Lab3: C Programming (cont.)

Submission Deadline: September 16th, 2021 23:59

"The only way to learn a new programming language is by writing programs in it!"
– "Dennis Ritchie".

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 previous lab to your Git repository. Then, try to pull the new lab:

```
$ cd Introduction2OS/labs
$ git pull upstream main
$ cd lab3
```

1 Strings

A string is one-dimensional array of type `char`. In C, a string is terminated by the end-of-string sentinel zero-slash or NULL character which is a byte with all bits off. You can manipulate a string on the same way with an array.
The two following statements are the same

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

and

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

Here, s[0] = a, s[1] = b, s[2] = c, s[3] = d, s[4] = e, and s[5] = NULL.

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 may want to use `strlen()` function in C).

```
#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 locations, called an *address*. A pointer is used to access the memory and to manipulate the address.
If v is a variable, then $\&v$ gives its address.
If p is a pointer, then $*p$ 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 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.

2.2.1 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;
}
```

2.3 Pointer and String

A string constant is treated as a pointer. Its value is the base address of the string. In the following code,

```
char *p = "abc";
printf("%s  %s \n", p, p+1);
```

the output would be:

```
abc bc
```

The variable `p` is assigned the base address of character array `abc`. When a pointer to char is printed in the format of a string, the pointed-at character and each successive characters are printed until the end of string. Thus, in the `printf()` statement, the expression `p` causes `abc` to be printed while the expression `p+1` which points to the letter `b` in the string `abc` causes `bc` to be printed.

2.3.1 Task 4

Write a program that prints out the number of characters in a string. The string is input from a user.(Use pointer and do not use function `strlen()` in C).

Sample output:

```
Enter a string you want to count:
Its length is:
```

Example:

```
Enter a string you want to count: abcdef
Its length is: 6
```

2.4 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 `int`, then the following expressions are the same: `A[i]` and `*(A+i)`.

If `p` is a pointer then

```
p = A       equivalent to p = &A[0];
p = A + 1   equivalent to p = &A[1];
```

The following code will show you 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 address of Array A is %p\n", A);
        printf("That is the address of the first element in the array A\n");

        return 0;
}
```

- Run the code.
- What do you see about the address of each element in the array A?

By using different experession 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 address of Array A is %p\n", A);
        printf("That is the address of the first element in the array A\n");

        return 0;
}
```

2.5 Task 5

Write a program that uses pointer to calculate the sumation of all elements in the input file *testcase0.txt*.

3 Dynamic Memory Allocation

C provides two functions to create space for arrays, structures, and unions. 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 = calloc(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);
```

3.1 Task 6: Dynamic memory allocation with an array

The following program dynamically allocates memory for an array. Complete the code to:

- Get the size of the array from user input.
- Fill-out all elements in the array.
- Print all elements of the array into screen.
- Deallocate memory for the array.

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

       // 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");
              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;
}
```

Sample output:

```
Number of elments in the array:
Enter array elements:
Array:
Memory deallocation done!
```

Example:

```
Number of elments in the array: 5
Enter matrix elements: 6 7 8 90 1
Array: 6 7 8 90 1
Memory deallocation done!
```

3.2 Task 7: Dynamic memory allocation with two-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;
}
```

Sample output:

```
Enter the nmuber of rows and columns for matrix:
Enter matrix elements:
Matrix is:
Memory deallocation done!
```

Example:

```
Enter the nmuber of rows and columns for matrix: 3 2
Enter matrix elements: 1 4 5 6 7 8
Matrix is:
1  4
```

```
5  6
7  8
Memory deallocation done!
```

4 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 proccess from input files. Complete the code to print out each member on the screen:

```
#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);
   }

  // YOUR CODE HERE


   return 0;
}
```

5 Exercises

5.1 Exercise 1: (40 points)

Use pointers to write a C program that gets integers from a user, puts the integers in an array, and prints out the integers in reverse order.

Sample output:

```
Enter the size of array:
Input values for the array:
The input array is:
Array printed in reserve order:
```

Example:

```
Enter the size of array: 6
Input values for the array: 1 23 45 65 78 12
The input array is: 1 23 45 65 78 12
Array printed in reserve order:12 78 65 45 23 1
```

5.2 Exercise 2: (60 points)

In an input file `Input.dat` containing the integer numbers of a matrix, the first and the second elements indicate the number of rows and the number of columns in the matrix, respectively. Write a C programming to:

- Read the first element of the file `Input.dat` and assigns it to a variable `M`.
- Read the second element of the file `Input.dat` and assigns it to a variable `N`.
- Create a matrix A with `M` rows and `N` columns; then dynamically allocate 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.

6 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.