ITF22519 - Introduction to Operating Systems

Lab4: More about C Programming

Fall 2020

In this lab, you will do more practices with C programming in Linux. Contents of this lab are strings, pointers, and dynamic memory allocation.

Before you start, remember to `commit` and `push` your previous lab to your git repository. Then, try to `pull` the new lab:

```
$ cd ITF22519/labs
$ git pull upstream master
$ cd lab4
```

`Tasks` are what you will do in this lab practice. All `Tasks`, `Exercises` and your report have to be submitted to GitHub afterwards.

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

Complete the following code to calculate the number of character `e` in the string `s`.

```
#include <stdio.h>

int main() {
        char s[] = "This Is The Operating System Course";
        char ch = 'e';
```

```
        int count = 0; // store the number of e in string s

        // Your code here

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

2 Pointer

A variable in a program is stored in a certain number of bytes at a particular memory locations, called 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 program that: (1) gets two integers from user input, (2) prints out where the two integers are stored in memory and (3) 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 `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, 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];
```

2.4.1 Task 5

Write a program that uses pointer to solve Task 7.2 in Lab3.

3 Dynamic Memory Allocation

C provides two functions **calloc()** (contiguous-allocation)and **malloc()** (memory-allocation) in the standard library *stdlib.h*. These two functions are used to crate space for arrays, structures, and unions. Example:

```
a = malloc(n*sizeof(int));
```

free() must be called to free the allocated memory with them.

```
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 (1) gets the size of the array from user input (2) fills-out all elements of the array, (3) prints out all elements of the array, and (4) deallocates 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 (1) fill out all elements of the array by getting input from a user, (2) print all elements, (3) 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 Exercises

4.1 Exercise 1: 30 points

Write a program that (1) gets a character from user input (2) gets a string from user input and (3) calculates the number of input character in the input string.

Sample output:

```
Enter a character:
Enter a string:
The number of input character in the string is:
```

Example:

```
Enter a character: a
Enter a string: abcde
The number of input character in the string is: 1
```

4.2 Exercise 2: 30 points

Use pointers to write a 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
```

4.3 Exercise 3: 40 points

(Refer to exercise 8.2 in Lab3).

Write a program that (a) reads the first element of Array.txt and assigns it to a variable `n`, (b) creates an array A with the size `n` and dynamically allocates memory for A, (c) reads the next `n` elements of Array.txt and assigns these elements with the corresponding elements in array A - if there are not enough `n` valid elements in Array.txt, print out error message, (d) prints array A and (e) finds the maximum element of A.

5 What To Submit

Complete all tasks under each section and submit your source files to GitHub. Complete the exercises in this lab, each exercise with its corresponding `.c` file and make your `lab4_report.txt` file. After that, put all of files into the **lab4** 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.