Lab N°03 : Linked Lists, Stacks and Queues

Exercise III.1 (Pointers & Functions)

Q1) In this program, add the necessary instructions to display the variable addresses and their values. Then execute the code and fill the memory map at the bottom.

```
Program
#include <stdio.h>
int main (){
long int x=100;
long int *y;
long int **z;
y=&x;
z=&y;
```
Bloc 1
```
x++;
*y=*y++;
*z=*z++;
```
Bloc 2
```
z++;
return 0;
}
```
Bloc 3

Memory map after execution Bloc 1

vriables	Addresses	values
x	……	……
y	……	……
z	……	……

Memory map after execution Bloc 2

vriables	Addresses	values
x	……	……
y	……	……
z	……	……

Memory map after execution Bloc 3

vriables	Addresses	values
x	……	……
y	……	……
z	……	……

Q2) Rerun a program, using a different type each time (short, int, double, long double...).

What is your remark? How can the different operation of increase be explained?

Q3) The following code has a bug. What's the problem, and how would you fix it?

```
void bar(char *str) {
      str = "ok bye!";
}
int main() {
      char *str = "hello world!";
      bar(str);
      printf("%s\n", str);          // should print "ok bye!"
      return 0;
}
```

Basic Linked List Functions

void `display (LIST List)` : Display the linked list elements.

void **insertAtBegin**`(LIST &List, int val)`: Insertion at the beginning.

void `insertAtEnd(LIST &List, int val)` : Insertion at end.

void `insertAfter(Node* prevNode, int val)` : Insertion after a given node.

void `deleteLL(LIST &List, Node* del)`: Delete a node from a linked list.

bool `searchLL(LIST List, int val)` : Search any value in the linked list.

void `updateLL(LIST &List, int val, int newVal)`: Update the value of the node.

Also, all functions implemented in exercise series N° 3 on linked list can be use.

init_List(), Create_Node(), Create(), Length(), Count(), GetNth(), InsertNth(), Reverse(), DeleteList(), SortedInsert(), InsertSort(), SwapNodes(), BubbleSort(), SelectionSort(), InsertionSort()

A node in a linked list is defined as follows:

```
struct Node {
    int value;
    struct Node* next;
};
typedef Node* LIST;
```

xercise III.2 (Linked Lists)

Q1) Place all the basic functions in a "**LIST.hpp**" file and save it.

Q2) Write all the basic functions seen in the course to a "**Test_Linked_List.cc**" file and Test all these functions.

Q3) Add all the functions implemented in exercise series N° 3 to a "**LIST.hpp**" file and save it.

Q4) Write all the functions implemented in exercise series N° 3 to a "**Test_Linked_List.cc**" file and Test all these functions.

Exercise III.3 (Sorting & Linked Lists)

Q1) Write an **isSort_List()** function to check whether a linked list is sorted by ascending **data** field values.

Q2) Write an **isExists()** function to check whether the value is in a sorted linked list or not.

Q3) Given two sorted lists, **L1** and **L2**, write an **Intersection_List()** function to compute $L1 \cap L2$ using only the basic list function.

Q4) Given two sorted lists, **L1** and **L2**, write a **Union_List()** function to compute $L1 \cup L2$ using only the basic list function.

Q5) Write a **CreateRandom()** function to create a linked list of n random elements.

Q6) Write a **List_Sum()** function that takes as input a linked list of integers and returns their sum (or **0** if the linked list is empty).

Q7) You are given a list, **L**, and another list, **P**, containing integers sorted in ascending order. Write the **printLots(L,P)** function will print the elements in **L** that are in positions specified by **P**. For instance, if **P = 1, 3, 4, 6**, the elements in positions **1, 3, 4**, and **6** in **L** are printed.

Q8) Write a program to generate **20** random numbers, store them in a linked list, and then print all numbers that are below their average.

Q9) Write a program to generate **50** random numbers and store them in a linked list, then ask the user to enter a number. Store the number in variable (**x**) and print all numbers in the linked list that is below this number.

Basic Stack Functions

```
STACK init_Stack() : Stack initialization
bool isEmpty(STACK Stack) : Check if the stack is empty
bool isFull() : Check if the stack is full
int Size(STACK Stack): Returns the number of elements in the stack
void push(STACK &Stack, int val): Adding an element to the top of a stack
int peek(STACK Stack): Examine the top element in the stack
int pop(STACK &Stack): Remove and return the element on top of the stack
```

Also, all functions implemented in exercise series N° 3 on stack can be used.

Init_List(), Create_Node(), Create(), Length(), Count(), GetNth(), InsertNth(), Reverse(), DeleteList(), SortedInsert(), InsertSort(), SwapNodes(), BubbleSort(), SelectionSort(), InsertionSort()

Basic Queue Functions

```
QUEUE init_Queue() : Queue initialization
bool isEmpty(QUEUE Queue) : Check if the queue is empty
bool isFull() : Check if the queue is full
int Size(QUEUE Queue): Returns the number of elements in the queue
void EnQueue(QUEUE &Queue, int val): Adding a new element to the queue
int DeQueue (QUEUE &Queue): Remove and return the front item from the queue
```

Also, all functions implemented in exercise series N° 3 on queue can be used.

Init_Stack(), Stack_Reverse(), Stack_Copy(), Stack_Sorted_Ascending(), Stack_Sorted_Descending(), Init_Queue(), Queue_Reverse(), Queue_Copy(), Queue_Sum()

A node in a **stack** is defined as follows:

```
struct Node {
    int data;
    struct Node* next;
};
typedef struct  {
    struct Node *top;
    int size;
} STACK;
```

A node in a **queue** is defined as follows:

```
struct Node {
    int data;
    struct Node* next;
};
typedef struct  {
    struct Node *first, *last;
    int size;
} QUEUE;
```

Exercise III.4 (Stacks & Queues)

Q1) Place all the basic functions in a "**Stack_Queue.hpp**" file and save it.

Q2) Write all the basic functions seen in the course to a "**Test_Stack_Queue.cc**" file and Test all these functions.

Q3) Add all the functions implemented in exercise series N° 3 to a "**Stack_Queue.hpp**" file and save it.

Q4) Write all the functions implemented in exercise series N° 3 to a "**Test_Stack_Queue.cc**" file and Test all these functions.

Exercise III.5 (Sorting & Stacks & Queues)

Q1) Write a **Stack_Sort()** function to sort the elements of a stack. The resulting stack should be sorted in ascending order, with the largest item at the top and the smallest at the bottom.

Q2) Write a **Queue_Sort()** function to sort the elements of a queue.

Q3) Write the **Queue_Duplicate()** function that duplicates all the elements in an ordered queue, each element duplicated once and only once.

Q4) Write a **Queue_Remove_Duplicate()** function to remove all duplicate elements from a sorted queue and leaves only one occurrence of each element.