Exercise Series N°03 : Linked Lists, Stacks and Queues

Exercise III.1 (Pointer)

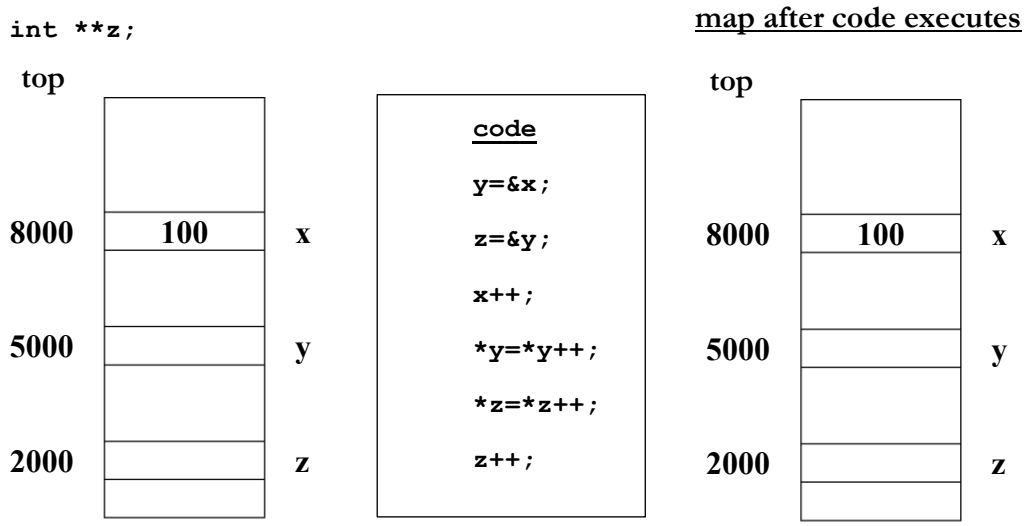
Q1) Assume p is a pointer to a **float**. Further, assume, the value of p is **1 000** (i.e., the address of the **float** it points to is **1 000**).

a) The value of the **float** is **17.6**. What value is $p++$?

b) Define in words what $*p$ and $\&p$ mean. Is there a way to determine the values of $*p$ and $\&p$ given the info above?

Q2) Given the initializations and memory map at the top, fill out the memory map on the bottom after the code has executed. Assume pointers are **32 *bits*** wide.

```
long int x=100;
long int *y;
long int **z;
```



Exercise III.2 (Pointer & Function)

Q1) What is the output of the following code fragment in C++ ? (assumption: all #include and the rest of the code are correct)

```
void Value (int p){
    p++;
    cout << p;
}
void Reference(int &p){
    p++;
    Value(p);
    cout << p;
}
```

```
void Pointer(int *p){
    (*p)++;
    Reference(*p);
    cout << *p;
}
```

```
int main(){
   int value =3;
   Pointer(&value);
   cout << value << endl;
   return 0;
}
```

Q2) Draw a memory diagram for the following code and determine what the output will be.

```
void foo(int *x, int *y, int *z) {
    x = y;
    *x = *z;
    *z = 37;
}
```

```
int main() {
int x = 5, y = 22, z = 42;
   foo(&x, &y, &z);
   printf("%d, %d, %d\n", x, y, z);
   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.

Exercise III.3 (Linked Lists)

A node in a linked list is defined as follows:

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

Q1) Write an **Init_List()** function to initialize a linked list.

Q2) Write a **Create_Node()** function to create a node.

Q3) Write a **Create()** function to create a linked list of n items read from the keyboard.

Q4) Write a **Length()** function to count the number of nodes in a linked list and returned.

Q5) Write a **Count()** function that counts the number of times a given **int** occurs in a linked list.

Q6) Write a **GetNth()** function that takes a linked list and an integer index and returns the data value stored in the node at that index position.

Q7) Write an **InsertNth()** function that can insert a new node at any index within a list.

Q8) Complete the function **Reverse()** to reverse the linked list. Do not create new list nodes and do not modify the contents of a list node.

Q9) Write a **DeleteList()** function that takes a list, deallocates all of its memory and sets its head pointer to **NULL** (the empty list).

Exercise III.4 (Sorting & Linked Lists)

Q1) Write a **SortedInsert()** function which given a list that is sorted in increasing order, and a single node, inserts the node into the correct sorted position in the list.

Q2) Write an **InsertSort()** function which given a list, rearranges its nodes so they are sorted in increasing order. It should use **SortedInsert()**.

Q3) Write a **SwapNodes()** function to swap two nodes in a linked list.

Q4) Write a **BubbleSort()** function to sort a linked list using the bubble sort algorithm.

Q5) Write a **SelectionSort()** function to sort a linked list using the selection sort algorithm.

Q6) Write an **InsertionSort()** function to sort a linked list using the insertion sort algorithm.

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
```

Exercise III.5 (Stacks)

A node in a stack is defined as follows:

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

Q1) Write an **Init_Stack()** function to initialize a stack.

Q2) Write a **Stack_Reverse()** that destructively returns a stack with the same elements as its input stack but in reverse order. The input stack shall be empty when the call returns.

Q3) Write a **Stack_Copy()** function to copy one stack to another stack.

Q4) Write two functions **Stack_Sorted_Ascending()** and **Stack_Sorted_Descending()** the first returns **true** if the items in stack are sorted in ascending order, with the largest item at the top and the smallest at the bottom, and **false** otherwise.

Similarly, the second function checks that its argument is sorted in descending order.

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
```

Exercise III.6 (Queues)

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;
```

Q1) Write an **Init_Queue()** function to initialize a queue.

Q2) Write a **Queue_Reverse()** function to reverse the elements of a queue using a stack.

Q3) Write a **Queue_Copy()** function to copy one queue to another.

Q4) Write a **Queue_Sum()** function that takes as input a queue of integers and returns their sum (or **0** if the queue is empty). Upon returning, the input queue should contain the same elements in the same order as when it was called.