Assignment 2

Name: Bharat Upadhyay
2017641
PCS 302(Data Structures with C)
Section: AI & DS

Q1 – Assuming that we have a stack with address top. Write a recursive function to print data from top to bottom

```
/Bharat Upadhyay
//2017641
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
  int info;
  struct node *next;
}node;
void insert(node **);
void display_stack(node*);
void display_queue(node*);
int main()
{
  node *top=NULL;
  int ch;
  do
  {
    printf("\n1 - Insert");
    printf("\n2 - Display stack in front order");
    printf("\n3 - Display stack in reverse order");
    printf("\nEnter your choice : ");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:
      insert(&top);
      break;
    case 2:
      display_stack(top);
      break;
    case 3:
      display_queue(top);
      break;
    default:
            printf("\nInvalid Choice");
            break;
        }
    }while(ch<=3);
}

void insert(node **top)
{
    node * p=NULL;
    int num;
    printf("Enter the value to be inserted : ");
    scanf("%d",&num);
    p=(node*)malloc(sizeof(node));
    if(p!=NULL)
    {
        p->info=num;
        if((*top)==NULL)
        {
            (*top)=p;
            (*top)->next=NULL;
        }
        else
        {
            p->next=(*top);
            (*top)=p;
        }
        printf("\nNode has been successfully added\n");
    }
    else
        printf("Memory not allocated");
}

void display_stack(node *top)
{
    if(top==NULL)
        printf("\nList is empty");
    else
    {
        printf("\n");
        while(top!=NULL)
        {
            printf("%d\n",top->info);
            top=top->next;
        }
    }
}
```

```
void display_queue(node *top)
{
   if(top==NULL)
   {
      return;
   }
   display_queue(top->next);
   printf("%d  ",top->info);
}
```

Q2 – Write a program(using circular linked list) to implement processor scheduling in multiprogramming environment, where there is only one processor and multiple processes which share the processor on time sharing environment. It means that every process will get processor for fixed amount of time. After that the process gets suspended and the processor switches over to the next process and so on until all the processes are over.

The user will supply the number of processes and time taken by each process. User will also supply the value of time which will be allocated to each process in a single go.

```
//Bharat Upadhyay
//2017641
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   int processNumber;
   int data

   struct node *next;
} nodetype;

void insert(nodetype **last, int processTime, int n)
{
   nodetype *p = NULL;
   p = (nodetype *)malloc(sizeof(nodetype));
   if (p != NULL)
   {
      p->processNumber = n;
      p->data = processTime;
      if (*last == NULL)
      {
         *last = p;
         (*last)->next = p;
         return;
```

```
        }
        p->next = (*last)->next;
        (*last)->next = p;
        (*last) = p;
    }
}

void deletenode(nodetype **prev)
{
    nodetype *p = *prev, *curr = NULL;
    if (p->next == p)
    {
        free(p);
        *prev = NULL;
        return;
    }
    curr = p->next;
    p->next = curr->next;
    free(curr);
    *prev = p;
}

void taskprocess(nodetype **last, int processTime)
{
    nodetype *prev = *last;
    int currTime = 0;
    nodetype *curr = NULL;
    while (prev != NULL)
    {
        curr = prev->next;
        curr->data -= processTime;
        currTime += processTime;
        if ((curr->data) <= 0)
        {

            printf("\nProcess %d is Completed in %dns", curr->processNumber, currTime);
            deletenode(&prev); // previous node of the node to be deleted
        }
        else
            prev = prev->next;
    }
}
int main()
{
    nodetype *last = NULL;
    int time, n, x;
    printf("Enter the processing unit in nanosecond : ");
    scanf("%d", &time);
    printf("Enter the Number of processes : ");
    scanf("%d", &n);
    printf("*Enter all the processes*\n");
    int i = 0;
    while (i < n)
    {
        printf("Enter the time of Process %d : ", i + 1);
        scanf("%d", &x);
        insert(&last, x, i + 1);
        i++;
    }
    taskprocess(&last, time);
    printf("\n*All processes completed successfully !!*");
    return 0;
}
```

Q3 – What is the difference between binary search and binary search tree.

BINARY SEARCH :
Binary search is an algorithm for finding an element in a binary search tree. It's a way of searching an ordered or sorted collection of elements. It is an algorithm that works on a sorted data structure. This is an efficient method of search because at each step, you can remove half the search space.

BINARY SEARCH TREE :
A binary search tree is a binary tree (i.e., a node, typically called the root) with the property that the left and right subtrees are also binary search trees, and that all the elements of all the nodes in the left subtree are less than the root's element, and all the elements of all the nodes in the right subtree are greater than the root's element. It is a type of tree data structure. Each node has at most 2 children.

ADVANTAGE OF BINARY SEARCH TREE OVER BINARY SEARCH:
Yes, there is an advantage of using Binary Search Tree over Binary Search. If we need to search for an element in a sorted array then binary search is applied to get the job done, but if the values of the array under consideration need to be updated at runtime without breaking its sorted property and maintaining order of log (size of array) complexity on average then you need a binary search tree.

This is because insertion and deletion in an array while maintaining its sorted property is order of size of array i.e O(n) but for a BST it is O(lg n) on average and O(n) in worst case.