Ahmet Alper Dönmez

17050111025

1.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define STACK_EMPTY -654455
#define SIZE 50

typedef int SType;

typedef struct{
     int array[SIZE];
     int top1;
     int top2;
}doublestack;

void initialize_s1(doublestack *s) //initialize our top1 value as the
beginning of doublestack
{
     s->top1=-1;
}

int is_empty_s1(doublestack *s)
{
     if(s->top1==-1)
     return 1;

     else
     return 0;
}

int is_full_s1(doublestack *s)
```

```
{
     if(s->top1==s->top2)
     return 1;

     else
     return 0;
}

void push1(doublestack *s, SType item1)
{
     if(is_full_s1(s))
     printf("Error: Stack is full!\n");

     else
     {
          (s->top1)++;
          s->array[s->top1]=item1;
     }
}

SType pop1(doublestack *s)
{
     SType item1;

     if(is_empty_s1(s))
     {
          printf("Error: Stack is empty!\n");
          item1=STACK_EMPTY;
     }

     else
     {
          item1=s->array[s->top1];
          (s->top1)--;
```

```
    }

    return (item1);
}

void initialize_s2(doublestack *s) //initialize our top2 value
{
    s->top2=SIZE-1;
}

int is_empty_s2(doublestack *s)
{
    if(s->top2==SIZE-1)
    return 1;

    else
    return 0;
}

int is_full_s2(doublestack *s)
{
    if(s->top2==s->top1)
    return 1;

    else
    return 0;
}

void push2(doublestack *s, SType item2)
{
    if(is_full_s2(s))
    printf("Error: Stack is full!\n");

    else
```

```
     {
          (s->top2)--;
          s->array[s->top2]=item2;
     }
}

SType pop2(doublestack *s)
{
     SType item2;

     if(is_empty_s2(s))
     {
          printf("Error: Stack is empty!\n");
          item2=STACK_EMPTY;
     }

     else
     {
          item2=s->array[s->top2];
          (s->top2)++;
     }

     return (item2);
}
```

2.

Q2 Header

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define STACK1_SIZE 100
#define STACK2_SIZE 9999
#define QUEUE_SIZE 100
```

```
typedef struct{
     int day;
     int month;
     int year;
}date_t;                  // application date


typedef struct{
     char name[50];
     char surname[50];
     char adress[100];
     date_t d;
}application_t;             // application form

typedef struct{
     application_t a[QUEUE_SIZE];
     int front;
     int rear;
     int counter;
}queue_t;                 //application forms waiting for process

typedef application_t QType;
QType QUEUE_EMPTY;

void initializeQ(queue_t *q)
{
     q->front=0;
     q->rear=-1;
     q->counter=0;
}

int isemptyQ(queue_t *q)
{
     if(q->front > q->rear)
```

```
     return 1;

     return 0;
}

int isfullQ(queue_t *q)
{
     if(q->rear = QUEUE_SIZE-1)
     return 1;

     return 0;
}


void insertQ(queue_t *q, QType item)
{
     if(isfullQ(q))
     printf("Error: Queue is full!\n");

     else
     {
          q->rear=(q->rear + 1)%QUEUE_SIZE;
          q->a[q->rear]=item;
          q->counter++;
     }
}


QType removeQ(queue_t *q)
{
     QType item;

     if(isemptyQ(q))
     {
```

```
          printf("Error: Queue is empty!\n");
          item = QUEUE_EMPTY;
     }

     else
     {
          item=q->a[q->front];
          q->front=(q->front+1)%QUEUE_SIZE;
          q->counter--;
     }
     return (item);
}

//-------------------------------------------------------------------------
-----------------------------------

//define our stacks and stack operations*/

typedef struct{
   application_t app;
   int phone;
}client_t;              //person's info and his/her dedicated number


typedef struct{
     int top1;
     client_t data[STACK1_SIZE];
}stack1_t;              //first stack which keeps dedicated numbers


typedef struct{
     int top2;
     int phone[STACK2_SIZE];
}stack2_t;        //second stack which keeps all possible numbers
```

```
client_t STACK1_EMPTY;

int STACK2_EMPTY;


void initialize_s1(stack1_t *s1)
{
     s1->top1=-1;
}

int is_empty_s1(stack1_t *s1)
{
     if(s1->top1==-1)
     return 1;

     return 0;
}

int is_full_s1(stack1_t *s1)
{
     if(s1->top1==STACK1_SIZE-1)
     return 1;

     return 0;
}

void push1(stack1_t *s1, client_t item1)
{
     if(is_full_s1(s1))
     printf("Error: Stack is full!\n");

     else
     {
```

```
          (s1->top1)++;
          item1=s1->data[s1->top1];
     }
}

client_t pop1(stack1_t *s1)
{
     client_t item1;

     if(is_empty_s1(s1))
     {
       printf("Error: Stack is empty!\n");
       item1=STACK1_EMPTY;
  }
     else
     {
          item1 = s1->data[s1->top1];
          (s1-> top1)--;
     }
     return item1;
}

void initialize_s2(stack2_t *s2)
{
     s2->top2=-1;
}

int is_empty_s2(stack2_t *s2)
{
     if(s2->top2==-1)
     return 1;

     return 0;
}
```

```
int is_full_s2(stack2_t *s2)
{
     if(s2->top2==STACK2_SIZE-1)
     return 1;

     return 0;
}

void push2(stack2_t *s2, int item2)
{
     if(is_full_s2(s2))
     printf("Error: Stack is full!\n");

     else
     {
          (s2->top2)++;
          item2 = s2->phone[s2->top2];
     }
}

int pop2(stack2_t *s2)
{
     int item2;

     if(is_empty_s2(s2))
     {
       printf("Error: Stack is empty!\n");
       item2=STACK2_EMPTY;
  }
     else
     {
          item2 = (s2->phone[s2->top2]);
          (s2->top2)--;
```

```
    }
  return item2;
}
```

Q2 Source

```
#include "mylibrary.h"
#define BEGIN 0000

void application_form(application_t app, queue_t *q)
//application form we'll take datas and put them into our queue
{
     printf("Name: ");
     scanf(" %s", app.name);

     printf("\nSurname: ");
     scanf(" %s", app.surname);

     printf("\nAdress: ");
     scanf(" %s", app.adress);

     printf("\nDate (put a blank between day,month an year): ");
     scanf("%d %d %d", app.d.day, app.d.month, app.d.year);

     insertQ(q, app);
}



void define_all_numbers(stack2_t *s2)
//function defines all possible telephone numbers
{
     int i=0;
    int temp1;

     initialize_s2(s2);
```

```
     while(i!=10000)
     {
          temp1=BEGIN+i;

          push2(s2, temp1);
     }

}

/*all numbers will be a 4 digit number and we have at most 100 citizen*/

void give_number(client_t c, stack1_t *s1, stack2_t *s2)
{
     initialize_s1(s1);

     define_all_numbers(s2);

     int temp2;

     temp2=pop2(s2);

     c.phone=temp2;

     push1(s1, c);

}


void delete_client(stack1_t *s1, stack2_t *s2, queue_t *q)
{
     client_t temp3;

     temp3=pop1(s1);
```

```
    removeQ(q);

}


int main()
{
     queue_t *q;
     application_t app;
     stack1_t *s1;
     stack2_t *s2;
     client_t c;
     int choice;


     printf("        MENU          \n");
     printf("----------------------\n");

     printf("1.Sign Up as a New Client\n");
     printf("2.Deleting Registration\n");
     printf("3.Exit\n");
     printf("Please select the operation above: ");
     scanf("%d", &choice);
     printf("\n");


     while(choice!=3)
     {

     if(choice==1)
     {

     application_form(app, q);
```

```
      printf("Your number is: %d", c.phone);
      give_number(c, s1, s2);

     }

      if(choice==2)
      {
           delete_client(s1, s2, q);

           printf("\nRemoving completed successfully\n");
      }

      printf("\n        MENU          \n");
      printf("----------------------\n");

      printf("1.Sign Up as a New Client\n");
      printf("2.Deleting Registration\n");
      printf("3.Exit\n");
      printf("Please select the operation above: ");
      scanf("%d", &choice);
      printf("\n");
   }
      return 0;
}
```

3.A. In the linked list, we have a pointer to the next node if we want to insert after a specific node we need to first find that node in the linked list and once you find it. Second, update that new node next pointer to the node which node founded in first step was pointing. Then update specific node next pointer to new node. Therefore we don’t need to take care about previous link.

Now it is not hard to add the before a specific node procedure is same search for the specific node in list but this time while doing so you need to maintain a second pointer pointing to previous node while you are traversing as we need to update the links when we do the insertion after that node and second pointer is pointing to the node after which want to insert so we can now follow insert after a specific node procedure.

The difference is just in this we need extra pointer to take care of previous nodes. First find the node in the linked list, the second pointer points to the previous node, once it is found. Secondly update the next pointer of the new node points to the node pointed by the node established in the first step. Then update the pointer of the previous node pointed to by the second pointer to the new node.

And No, it is definitely possible to insert a node before a specific node.

3.B. To implement a stack using linked list first we have to look at the rules of stack. First rule is, last object in first object out. Second rule is, all the operations we should perform so with the help of a top variable.

Push(): Function will insert the element at top node by doing new node will point to address in top pointer then update top points to new node.

Pop(): Function will delete element pointing by top pointer. First delete the node pointing by top pointer then update the top pointer with link value of deleted node.

Stack full: Stack is said to be overflown if the space left in the memory heap is not enough to create a node.

Stack Empty: When top pointer is having null value.

To implement queue using linked list first we have to look at the rules of queue and linked list. In queue we have two pointers front and rear. Front pointers to first value in the queue and Rear points to the latest inserted value. In linked list we take two pointers too. Front pointer pointing to first node, Rear pointer pointing to last node of linked list.

Insert(): In queue will insert the node after the last node. First insert new node pointed by rear and update new node pointer to value pointed by rear. Then update pointer to new node.

Delete(): In queue will delete first node of list. First delete first node of list. Then update front pointing to second node now.

Queue full: Queue is said to be overflown if the space left in the memory heap is not enough to create a node.

Queue Empty: When front is null so that means list is empty.

3.C. First thing is linked list waste lot of memory because pointer requires extra memory for storage. Second thing is we cannot random access like array. Third thing is we cannot traverse linked list in reverse order as we do in array.