NAME-MD SABIR HUSSAIN

ROLL-20MCMB13

STACK USING ARRAY

```
#include <stdio.h>


int MAXSIZE = 8;

int stack[8];

int top = -1;


int isempty()

  {


     if(top == -1)

        return 1;

     else

        return 0;

  }


int isfull()

  {


     if(top == MAXSIZE)

        return 1;

     else

        return 0;

  }


int peek()

  {

     return stack[top];

  }
```

```
int pop()
  {
      int data;

      if(!isempty())
      {
         data = stack[top];
         top = top - 1;
         return data;
      }
      else
      {
         printf("Could not retrieve data, Stack is empty.\n");
      }
  }

int push(int data)
  {

      if(!isfull())
      {
         top = top + 1;
         stack[top] = data;
      }
      else
      {
         printf("Could not insert data, Stack is full.\n");
      }
  }
```

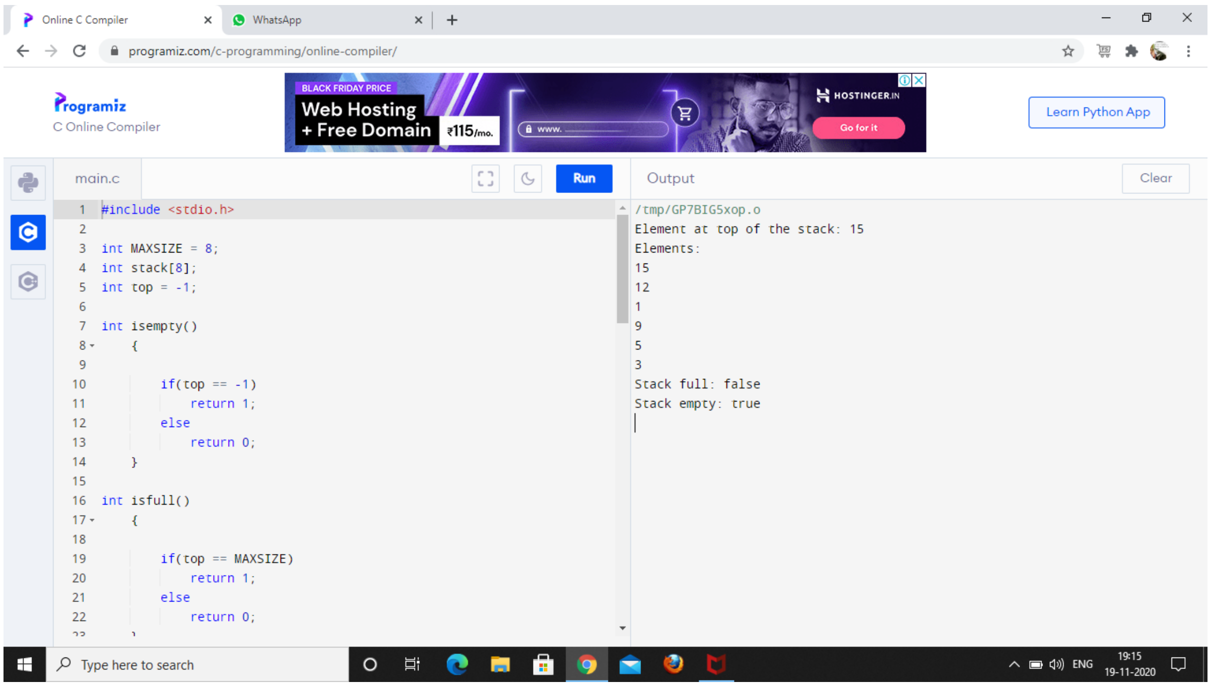
```
int main()
   {
// push items on to the stack
      push(3);
      push(5);
      push(9);
      push(1);
      push(12);
      push(15);

      printf("Element at top of the stack: %d\n" ,peek());
      printf("Elements: \n");

      // print stack data
      while(!isempty())
      {
         int data = pop();
         printf("%d\n",data);
      }

      printf("Stack full: %s\n" , isfull()?"true":"false");
      printf("Stack empty: %s\n" , isempty()?"true":"false");

      return 0;
}
```



```c
#include <stdio.h>

int MAXSIZE = 8;
int stack[8];
int top = -1;

int isempty()
   {

      if(top == -1)
         return 1;
      else
         return 0;
   }

int isfull()
   {

      if(top == MAXSIZE)
         return 1;
      else
         return 0;
```

```
/tmp/GP7BIG5xop.o
Element at top of the stack: 15
Elements:
15
12
1
9
5
3
Stack full: false
Stack empty: true
```

Stack using link list

```c
#include <stdio.h>


#include < stdlib.h>


struct node
{
int info;
struct node *ptr;
}*top,*top1,*temp;


int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();
```

```
int count = 0;

void main()
{
int no, ch, e;

printf("\n 1 - Push");
printf("\n 2 - Pop");
printf("\n 3 - Top");
printf("\n 4 - Empty");
printf("\n 5 - Exit");
printf("\n 6 - Dipslay");
printf("\n 7 - Stack Count");
printf("\n 8 - Destroy stack");

create();

while (1)
{
printf("\n Enter choice : ");
scanf("%d", &ch);

switch (ch)
{
case 1:
printf("Enter data : ");
scanf("%d", &no);
push(no);
break;
case 2:
```

```
pop();
break;
case 3:
if (top == NULL)
printf("No elements in stack");
else
{
e = topelement();
printf("\n Top element : %d", e);
}
break;
case 4:
empty();
break;
case 5:
exit(0);
case 6:
display();
break;
case 7:
stack_count();
break;
case 8:
destroy();
break;
default :
printf(" Wrong choice, Please enter correct choice  ");
break;
}
}
}
```

```
/* Create empty stack */
void create()
{
top = NULL;
}

/* Count stack elements */
void stack_count()
{
printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
if (top == NULL)
{
top =(struct node *)malloc(1*sizeof(struct node));
top->ptr = NULL;
top->info = data;
}
else
{
temp =(struct node *)malloc(1*sizeof(struct node));
temp->ptr = top;
temp->info = data;
top = temp;
}
count++;
}
```

```
/* Display stack elements */
void display()
{
top1 = top;

if (top1 == NULL)
{
printf("Stack is empty");
return;
}

while (top1 != NULL)
{
printf("%d ", top1->info);
top1 = top1->ptr;
}
}

/* Pop Operation on stack */
void pop()
{
top1 = top;

if (top1 == NULL)
{
printf("\n Error : Trying to pop from empty stack");
return;
}
else
top1 = top1->ptr;
```

```
printf("\n Popped value : %d", top->info);

free(top);

top = top1;

count--;

}


/* Return top element */

int topelement()

{

return(top->info);

}


/* Check if stack is empty or not */

void empty()

{

if (top == NULL)

printf("\n Stack is empty");

else

printf("\n Stack is not empty with %d elements", count);

}


/* Destroy entire stack */

void destroy()

{

top1 = top;


while (top1 != NULL)

{

top1 = top->ptr;

free(top);

top = top1;
```

```
  top1 = top1->ptr;
  }
  free(top1);
  top = NULL;

  printf("\n All stack elements destroyed");
  count = 0;
  }
```

Queue Program USING ARRAY

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

#define MAX_SIZE 100

int main()
{
  int item, choice, i;
  int arr_queue[MAX_SIZE];
  int rear = 0;
  int front = 0;
  int exit = 1;

  printf("\nSimple Queue Example - Array");
  do {
    printf("\n\n Queue Main Menu");

    printf("\n1.Insert \n2.Remove \n3.Display \nOthers to exit");
    printf("\nEnter Your Choice : ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
          if (rear == MAX_SIZE)
            printf("\n## Queue Reached Max!!");
          else {
            printf("\nEnter The Value to be Insert : ");
            scanf("%d", &item);
            printf("\n## Position : %d , Insert Value  : %d ", rear + 1, item);
            arr_queue[rear++] = item;
          }
          break;
        case 2:
          if (front == rear)
            printf("\n## Queue is Empty!");
          else {
            printf("\n## Position : %d , Remove Value  : %d ", front, arr_queue[front]);
            front++;
          }
          break;
        case 3:
          printf("\n## Queue Size : %d ", rear);
          for (i = front; i < rear; i++)
            printf("\n## Position : %d , Value  : %d ", i, arr_queue[i]);
          break;
        default:
          exit = 0;
          break;
      }
    } while (exit);

    return 0;
  }
```

QUEUE USING LINKLIST

```
#include <stdio.h>
#include <conio.h>
struct Node
{
int data;
struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();

void main()
{
  int choice, value;
  clrscr();
  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1)
  {
     printf("\n*** MENU ***\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1: printf("Enter the value to be insert: ");
              scanf("%d", &value);
              insert(value);
              break;
```

```
            case 2: delete(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nWrong selection!!! Please try again!!!\n");
        }
    }
}
void insert(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
     newNode -> next = NULL;
    if(front == NULL)
        front = rear = newNode;
    else
    {
        rear -> next = newNode;
        rear = newNode;
    }
    printf("\nInsertion is Success!!!\n");
}
void delete()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else
    {
        struct Node *temp = front;
        front = front -> next;
        printf("\nDeleted element: %d\n", temp->data);
        free(temp);
    }
}
void display()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else
    {
        struct Node *temp = front;
        while(temp->next != NULL)
        {
            printf("%d--->",temp->data);
            temp = temp -> next;
        }
        printf("%d--->NULL\n",temp->data);
    }
}
```