|  |  |  |  |
| --- | --- | --- | --- |
| **SVKM's-IOT, Dhule**Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| Name : | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**02 | **Date : 13-08-2024** | | Signature |
| **Title :**  Write a program to evaluate a given postfix expression using stacks. | | |
|  | | |
|  | | |

**Code:**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

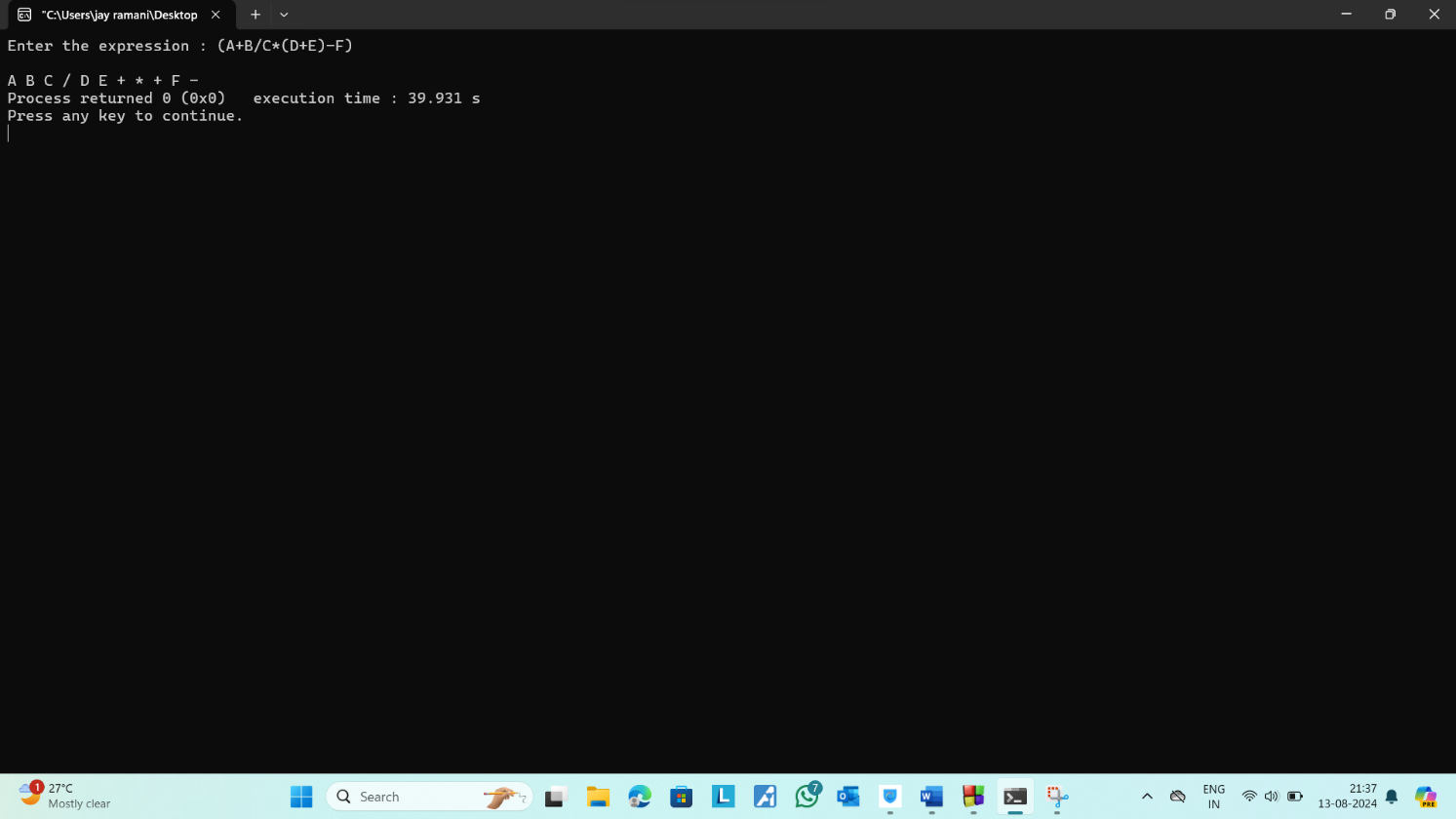
{

printf("%c ",pop());

}return 0;

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| **SVKM's-IOT, Dhule**Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**03 | **Date :** 3-09-2024 | | Signature |
| **Title :**  Write a program to convert a given infix to postfix expression using stacks. | | |
|  | | |
|  | | |

**Code:**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

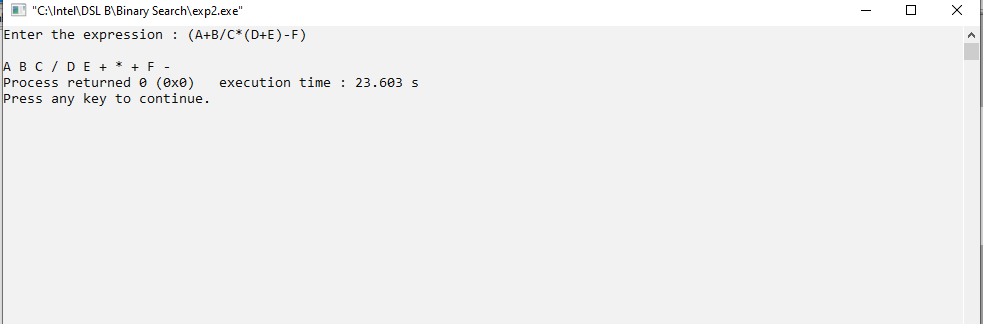
{

printf("%c ",pop());

}return 0;

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| **SVKM's-IOT, Dhule**Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**04 | **Date :** | | Signature |
| **Title :**  Write a program to implement circular queue using array | | |
|  | | |
|  | | |

**Code:**

**# include<stdio.h>**

**# define MAX 5**

**int cqueue\_arr[MAX];**

**int front = -1; int rear = -1;**

**void insert(int item)**

**{ if**

**((front == 0 && rear == MAX-1) || (front == rear+1))**

**{**

**printf("Queue Overflow \n");**

**return;**

**}**

**if (front == -1)**

**{ front = 0;**

**rear = 0;**

**}**

**else**

**{**

**if(rear == MAX-1)**

**rear = 0;**

**else**

**rear = rear+1;**

**}**

**cqueue\_arr[rear] = item ;**

**}**

**void del()**

**{ if (front == -1)**

**{ printf("Queue Underflow\n");**

**return ;**

**}**

**printf("Element deleted from queue is : %d\n",cqueue\_arr[front]);**

**if(front == rear)**

**{**

**front = -1;**

**rear=-1;**

**}**

**else**

**{**

**if(front == MAX-1)**

**front = 0;**

**else**

**front = front+1;**

**}**

**}**

**void display()**

**{**

**int front\_pos = front,rear\_pos = rear;**

**if(front == -1)**

**{**

**printf("Queue is empty\n");**

**return;**

**}**

**printf("Queue elements :\n");**

**if( front\_pos <= rear\_pos )**

**while(front\_pos <= rear\_pos)**

**{**

**printf("%d ",cqueue\_arr[front\_pos]);**

**front\_pos++;**

**}**

**else**

**{ while(front\_pos <= MAX-1)**

**{ printf("%d ",cqueue\_arr[front\_pos]);**

**front\_pos++;**

**}**

**front\_pos = 0;**

**while(front\_pos <= rear\_pos)**

**{ printf("%d ",cqueue\_arr[front\_pos]);**

**front\_pos++;**

**}**

**} printf("\n");**

**} int main()**

**{ int choice,item;**

**do**

**{**

**printf("1.Insert \n");**

**printf("2.Delete \n");**

**printf("3.Display \n");**

**printf("4.Quit \n");**

**printf("Enter your choice : ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1 :printf("Input the element for insertion in queue : ");**

**scanf("%d", &item);**

**insert(item);**

**break;**

**case 2 : del();**

**break;**

**case 3: display();**

**break;**

**case 4: break;**

**default: printf("Wrong choice\n");**

**}**

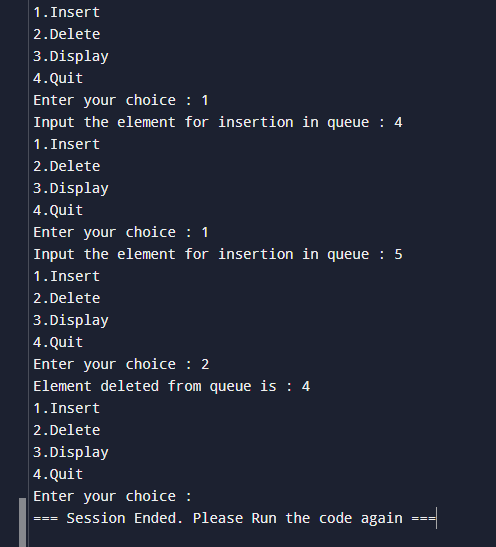
**}**

**while(choice!=4);**

**return 0;**

**}**

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| **SVKM's-IOT, Dhule**Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**05 | **Date : 15-10-24** | | Signature |
| **Title :**  Write a program to implement double ended queue using array. | | |
|  | | |
|  | | |

**Code:**

#include<stdio.h>

#define MAX 5

int deque\_arr[MAX];

int left = -1;

int right = -1;

void insert\_right()

{

int added\_item;

if((left == 0 && right == MAX-1) || (left == right+1))

{

printf("Queue Overflow\n");

return;

}

if (left == -1)

{

left = 0;

right = 0;

}

else

if(right == MAX-1)

right = 0;

else

right = right+1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[right] = added\_item ;

}

void insert\_left()

{

int added\_item;

if((left == 0 && right == MAX-1) || (left == right+1))

{

printf("Queue Overflow \n");

return;

}

if (left == -1)/\*If queue is initially empty\*/

{

left = 0;

right = 0;

}

else

if(left== 0)

left=MAX-1;

else

left=left-1;

printf("Input the element for adding in queue : ");

scanf("%d", &added\_item);

deque\_arr[left] = added\_item ; }

void delete\_left()

{

if (left == -1)

{

printf("Queue Underflow\n");

return ;

}

printf("Element deleted from queue is : %d\n",deque\_arr[left]);

if(left == right)

{

left = -1;

right=-1;

}

else

if(left == MAX-1)

left = 0;

else

left = left+1;

}

void delete\_right()

{

if (left == -1)

{

printf("Queue Underflow\n");

return ;

}

printf("Element deleted from queue is : %d\n",deque\_arr[right]);

if(left == right)

{

left = -1;

right=-1;

}

else

if(right == 0)

right=MAX-1;

else

right=right-1;

}

void display\_queue()

{

int front\_pos = left,rear\_pos = right;

if(left == -1)

{

printf("Queue is empty\n");

return;

}

printf("Queue elements :\n");

if( front\_pos <= rear\_pos )

{

while(front\_pos <= rear\_pos)

{

printf("%d ",deque\_arr[front\_pos]);

front\_pos++;

}

}

else

{

while(front\_pos <= MAX-1)

{

printf("%d ",deque\_arr[front\_pos]);

front\_pos++;

}

front\_pos = 0;

while(front\_pos <= rear\_pos)

{

printf("%d ",deque\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

void input\_que()

{

int choice;

do

{

printf("1.Insert at right\n");

printf("2.Delete from left\n");

printf("3.Delete from right\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:insert\_right();

break;

case 2:delete\_left();

break;

case 3:delete\_right();

break;

case 4:display\_queue();

break;

case 5:break;

default:printf("Wrong choice\n");

}

}while(choice!=5);

}

void output\_que()

{ int choice;

do

{ printf("1.Insert at right\n");

printf("2.Insert at left\n");

printf("3.Delete from left\n");

printf("4.Display\n");

printf("5.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:insert\_right();

break;

case 2:insert\_left();

break;

case 3:delete\_left();

break;

case 4:display\_queue();

break;

case 5:break;

default:printf("Wrong choice\n");

}

}while(choice!=5);

}

main()

{

int choice;

printf("1.Input restricted dequeue\n");

printf("2.Output restricted dequeue\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :input\_que();

break;

case 2:output\_que();

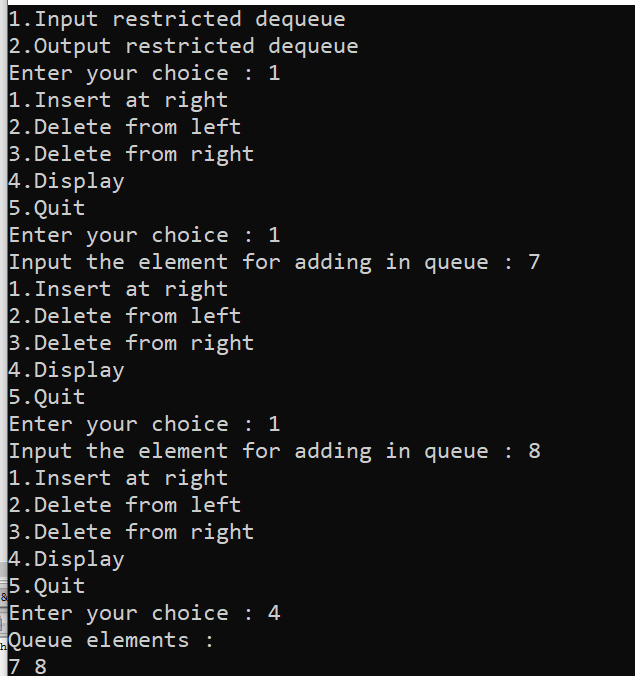
break;

default:printf("Wrong choice\n");

}

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| Shri Vile Parle Kelavani Mandal's SVKM's-IOT, Dhule  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**06 | **Date :** | | Signature |
| **Title :**  Write a program to implement a stack using two queues such that the push  operation runs in constant time and the pop operation runs in linear time. | | |
|  | | |
|  | | |

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define QUEUE\_EMPTY\_MAGIC 0xdeadbeef

typedef struct \_queue\_t {

int \*arr;

int rear, front, count, max;

} queue\_t;

queue\_t \*queue\_allocate(int n);

void queue\_insert(queue\_t \*q, int v);

int queue\_remove(queue\_t \*q);

int queue\_count(queue\_t \*q);

int queue\_is\_empty(queue\_t \*q);

void queue\_display(queue\_t \*q);

void stack\_push(queue\_t \*q, int v);

int stack\_pop(queue\_t \*q);

int stack\_is\_empty(queue\_t \*q);

int stack\_count(queue\_t \*q);

int queue\_count(queue\_t \*q) {

return q->count;

}

queue\_t \*queue\_allocate(int n) {

queue\_t \*queue = malloc(sizeof(queue\_t));

if (queue == NULL) {

return NULL;

}

queue->max = n;

queue->arr = malloc(sizeof(int) \* n);

if (queue->arr == NULL) {

free(queue);

return NULL;

}

queue->rear = n - 1;

queue->front = n - 1;

queue->count = 0;

return queue;

}

void queue\_insert(queue\_t \*q, int v) {

if (q->count == q->max) {

printf("\nQueue is full. Cannot insert %d.\n", v);

return;

}

q->rear = (q->rear + 1) % q->max;

q->arr[q->rear] = v;

q->count++;

}

int queue\_remove(queue\_t \*q) {

if (q->count == 0) {

return QUEUE\_EMPTY\_MAGIC;

}

q->front = (q->front + 1) % q->max;

int retval = q->arr[q->front];

q->count--;

return retval;

}

int queue\_is\_empty(queue\_t \*q) {

return (q->count == 0);

}

void queue\_display(queue\_t \*q) {

if (queue\_is\_empty(q)) {

printf("[Empty Queue]");

return;

}

int i = (q->front + 1) % q->max;

int elements = queue\_count(q);

while (elements--) {

printf("[%d] ", q->arr[i]);

i = (i + 1) % q->max;

}

}

void stack\_push(queue\_t \*q, int v) {

queue\_insert(q, v);

}

int stack\_pop(queue\_t \*q) {

int n = queue\_count(q);

if (n == 0) {

return QUEUE\_EMPTY\_MAGIC;

}

for (int i = 0; i < n - 1; i++) {

queue\_insert(q, queue\_remove(q));

}

return queue\_remove(q);

}

int stack\_is\_empty(queue\_t \*q) {

return queue\_is\_empty(q);

}

int stack\_count(queue\_t \*q) {

return queue\_count(q);

}

#define MAX 128

int main(void) {

queue\_t \*q = queue\_allocate(MAX);

if (q == NULL) {

printf("Failed to allocate memory for the queue.\n");

return 1;

}

int x, select;

do {

printf("\n[1] Push\n[2] Pop\n[0] Exit");

printf("\nChoice: ");

scanf(" %d", &select);

switch (select) {

case 1:

printf("\nEnter value to Push: ");

scanf(" %d", &x);

stack\_push(q, x);

printf("\n\n\_\_\nCurrent Queue:\n");

queue\_display(q);

printf("\n\nPushed Value: %d", x);

printf("\n\_\_\n");

break;

case 2:

x = stack\_pop(q);

printf("\n\n\_\_\nCurrent Queue:\n");

queue\_display(q);

if (x == QUEUE\_EMPTY\_MAGIC) {

printf("\n\nNo values removed");

} else {

printf("\n\nPopped Value: %d", x);

}

printf("\n\_\_\n");

break;

case 0:

printf("\nQuitting.\n");

free(q->arr);

free(q);

return 0;

default:

printf("\nInvalid choice. Try again.\n");

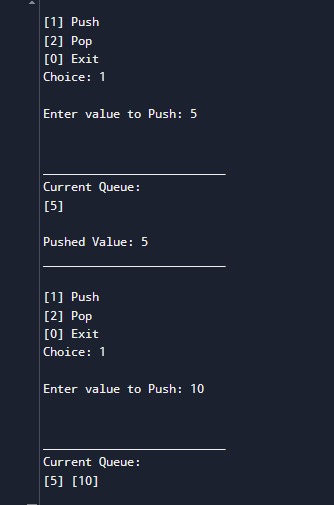
}

} while (1);

    return 0;

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| Shri Vile Parle Kelavani Mandal's SVKM's-IOT, Dhule  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :**  ] | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**07 | **Date :** | | Signature |
| **Title :** Write a program to implement a stack using two queues such that the push operation  runs in linear time and the pop operation runs in constant time. | | |
|  | | |
|  | | |

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define QUEUE\_EMPTY\_MAGIC 0xdeadbeef

typedef struct \_queue\_t {

int \*arr;

int front, rear, count, max;

} queue\_t;

typedef struct \_stack\_t {

queue\_t \*queue1;

queue\_t \*queue2;

} stack\_t;

queue\_t \*queue\_allocate(int n);

void queue\_insert(queue\_t \*q, int v);

int queue\_remove(queue\_t \*q);

int queue\_is\_empty(queue\_t \*q);

stack\_t \*stack\_allocate(int n);

void stack\_push(stack\_t \*s, int v);

int stack\_pop(stack\_t \*s);

int stack\_is\_empty(stack\_t \*s);

queue\_t \*queue\_allocate(int n) {

queue\_t \*q = (queue\_t \*)malloc(sizeof(queue\_t));

if (!q) return NULL;

q->arr = (int \*)malloc(sizeof(int) \* n);

if (!q->arr) {

free(q);

return NULL;

}

q->front = -1;

q->rear = -1;

q->count = 0;

q->max = n;

return q;

}

void queue\_insert(queue\_t \*q, int v) {

if (q->count == q->max) {

printf("Queue is full. Cannot insert %d.\n", v);

return;

}

if (q->count == 0) {

q->front = q->rear = 0;

} else {

q->rear = (q->rear + 1) % q->max;

}

q->arr[q->rear] = v;

q->count++;

}

int queue\_remove(queue\_t \*q) {

if (q->count == 0) {

return QUEUE\_EMPTY\_MAGIC;

}

int val = q->arr[q->front];

if (q->count == 1) {

q->front = q->rear = -1;

} else {

q->front = (q->front + 1) % q->max;

}

q->count--;

return val;

}

int queue\_is\_empty(queue\_t \*q) {

return (q->count == 0);

}

stack\_t \*stack\_allocate(int n) {

stack\_t \*s = (stack\_t \*)malloc(sizeof(stack\_t));

if (!s) return NULL;

s->queue1 = queue\_allocate(n);

s->queue2 = queue\_allocate(n);

if (!s->queue1 || !s->queue2) {

if (s->queue1) free(s->queue1);

if (s->queue2) free(s->queue2);

free(s);

return NULL;

}

return s;

}

void stack\_push(stack\_t \*s, int v) {

queue\_t \*q1 = s->queue1;

queue\_t \*q2 = s->queue2;

queue\_insert(q1, v);

while (!queue\_is\_empty(q2)) {

queue\_insert(q1, queue\_remove(q2));

}

queue\_t \*temp = s->queue1;

s->queue1 = s->queue2;

s->queue2 = temp;

}

int stack\_pop(stack\_t \*s) {

if (stack\_is\_empty(s)) {

return QUEUE\_EMPTY\_MAGIC;

}

return queue\_remove(s->queue2);

}

int stack\_is\_empty(stack\_t \*s) {

return queue\_is\_empty(s->queue2);

}

int main() {

int n = 10;

stack\_t \*s = stack\_allocate(n);

if (!s) {

printf("Failed to allocate memory for the stack.\n");

return 1;

}

int choice, val;

do {

printf("\n[1] Push\n[2] Pop\n[3] Check Empty\n[0] Exit\nChoice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &val);

stack\_push(s, val);

printf("%d pushed onto the stack.\n", val);

break;

case 2:

val = stack\_pop(s);

if (val == QUEUE\_EMPTY\_MAGIC) {

printf("Stack is empty. Cannot pop.\n");

} else {

printf("Popped: %d\n", val);

}

break;

case 3:

if (stack\_is\_empty(s)) {

printf("Stack is empty.\n");

} else {

printf("Stack is not empty.\n");

}

break;

case 0:

printf("Exiting.\n");

break;

default:

printf("Invalid choice. Try again.\n");

}

} while (choice != 0);

free(s->queue1->arr);

free(s->queue1);

free(s->queue2->arr);

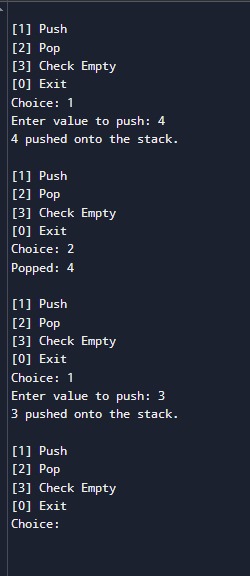
free(s->queue2);

free(s);

return 0;

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| SVKM's-IOT, DhuleShri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :08** | **Date :** | | Signature |
| **Title :** Write a program to implement a queue using two stacks such that dequeue  operation runs in constant time and dequeue operation runs in linear time. | | |
|  | | |
|  | | |

**CODE**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 100**

**typedef struct {**

**int data[MAX];**

**int top;**

**} Stack;**

**void initStack(Stack \*s) {**

**s->top = -1;**

**}**

**int isEmpty(Stack \*s) {**

**return s->top == -1;**

**}**

**void push(Stack \*s, int value) {**

**if (s->top < MAX - 1) {**

**s->data[++s->top] = value;**

**} else {**

**printf("Stack overflow\n");**

**}**

**}**

**int pop(Stack \*s) {**

**if (!isEmpty(s)) {**

**return s->data[s->top--];**

**} else {**

**printf("Stack underflow\n");**

**return -1;**

**}**

**}**

**typedef struct {**

**Stack stack1;**

**Stack stack2;**

**} Queue;**

**void initQueue(Queue \*q) {**

**initStack(&q->stack1);**

**initStack(&q->stack2);**

**}**

**void enqueue(Queue \*q, int value) {**

**push(&q->stack1, value);**

**}**

**int dequeue(Queue \*q) {**

**if (isEmpty(&q->stack2)) {**

**while (!isEmpty(&q->stack1)) {**

**push(&q->stack2, pop(&q->stack1));**

**}**

**}**

**return pop(&q->stack2);**

**}**

**int main() {**

**Queue q;**

**initQueue(&q);**

**enqueue(&q, 1);**

**enqueue(&q, 2);**

**enqueue(&q, 3);**

**printf("Dequeued: %d\n", dequeue(&q)); // Output: 1**

**printf("Dequeued: %d\n", dequeue(&q)); // Output: 2**

**enqueue(&q, 4);**

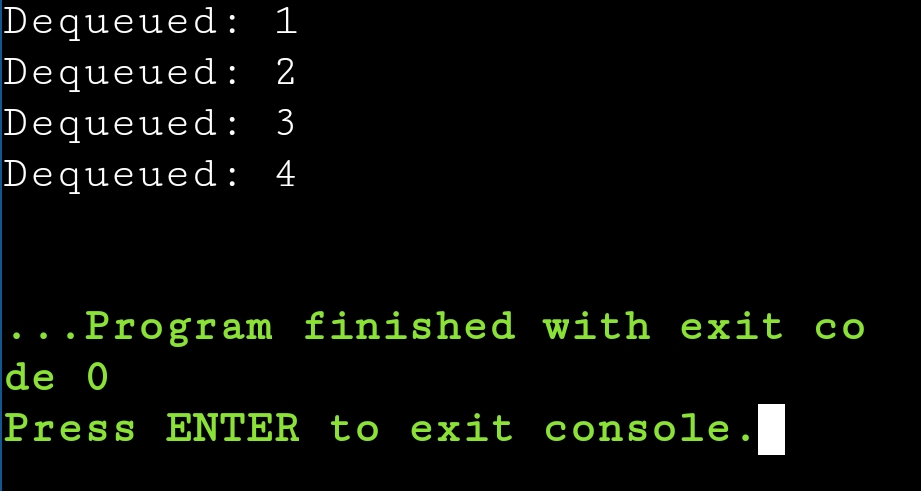
**printf("Dequeued: %d\n", dequeue(&q)); // Output: 3**

**printf("Dequeued: %d\n", dequeue(&q)); // Output: 4**

**return 0;**

**}**

**OUTPUT**

****

|  |  |  |  |
| --- | --- | --- | --- |
| SVKM's-IOT, DhuleShri Vile Parle Kelavani Mandal's  INSTITUTE OF TECHNOLOGY  DHULE (M.S.)  DEPARMENT OF COMPUTER ENGINEERING | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No.** :09 | **Date :** | | Signature |
| **Title :** Write a program to implement a stack using a linked list such that the push and pop operations of stack still take O(1) time. | | |
|  | | |
|  | | |

CODE

#include <stdio.h>

#include <stdlib.h>

Struct Node {

Int data;

Struct Node\* next;

};

Void push(struct Node\*\* top, int value) {

Struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

If (!newNode) {

Printf(“Heap overflow\n”);

Return;

}

newNode->data = value;

newNode->next = \*top;

\*top = newNode;

Printf(“%d pushed to stack\n”, value);

}

Int pop(struct Node\*\* top) {

If (\*top == NULL) {

Printf(“Stack Underflow\n”);

Return -1;

}

Struct Node\* temp = \*top;

\*top = (\*top)->next;

Int popped = temp->data;

Free(temp);

Return popped;

}

Void display(struct Node\* top) {

If (top == NULL) {

Printf(“Stack is empty\n”);

Return;

}

Struct Node\* temp = top;

While (temp != NULL) {

Printf(“%d -> “, temp->data);

Temp = temp->next;

}

Printf(“NULL\n”);

}

Int main() {

Struct Node\* top = NULL;

Push(&top, 10);

Push(&top, 20);

Push(&top, 30);

Display(top);

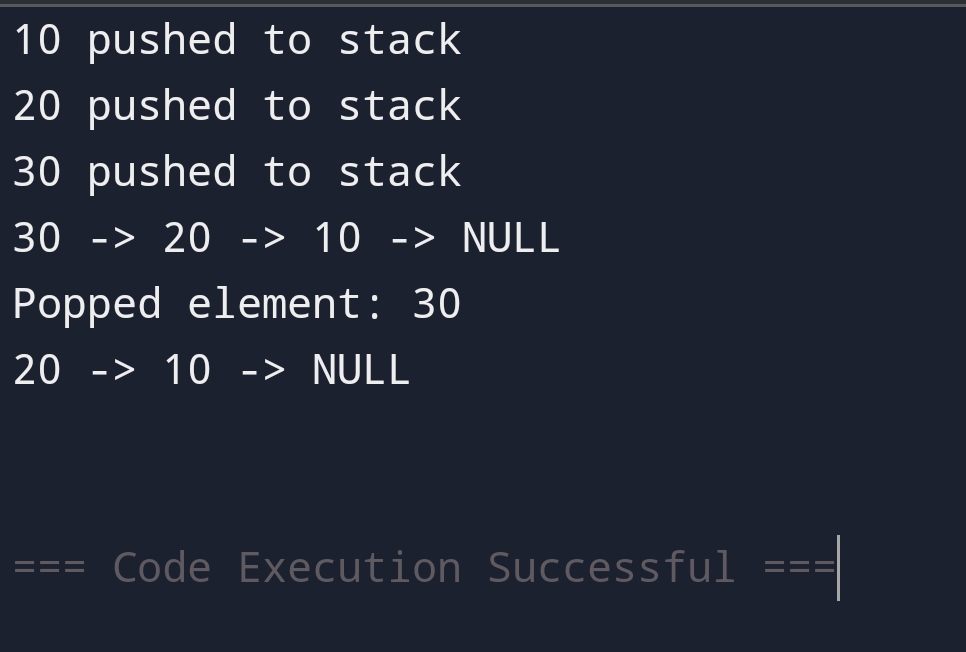
Printf(“Popped element: %d\n”, pop(&top));

Display(top);

Return 0;

}

**OUTPUT**

****

|  |  |  |  |
| --- | --- | --- | --- |
| Shri Vile Parle Kelavani Mandal's SVKM's-IOT, Dhule  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**10 | **Date :** | | Signature |
| **Title :** Write a program to implement a stack using a linked list such that the push and pop  operations of stack still take O(1)time | | |
|  | | |
|  | | |

**Code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

Node \*create\_node(int data) {

Node \*new\_node = (Node \*)malloc(sizeof(Node));

if (!new\_node) {

printf("Memory allocation failed.\n");

exit(1);

}

new\_node->data = data;

new\_node->next = NULL;

return new\_node;

}

void push(Node \*\*head, int data) {

Node \*new\_node = create\_node(data);

new\_node->next = \*head;

\*head = new\_node;

printf("%d pushed onto the stack.\n", data);

}

int pop(Node \*\*head) {

if (\*head == NULL) {

printf("Stack underflow. Cannot pop.\n");

return -1;

}

Node \*temp = \*head;

int popped\_data = temp->data;

\*head = temp->next;

free(temp);

return popped\_data;

}

int is\_empty(Node \*head) {

return head == NULL;

}

void display(Node \*head) {

if (is\_empty(head)) {

printf("Stack is empty.\n");

return;

}

printf("Stack: ");

Node \*current = head;

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

Node \*stack = NULL;

int choice, value;

do {

printf("\n[1] Push\n[2] Pop\n[3] Display\n[0] Exit\nChoice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

break;

case 2:

value = pop(&stack);

if (value != -1) {

printf("Popped: %d\n", value);

}

break;

case 3:

display(stack);

break;

case 0:

printf("Exiting.\n");

break;

default:

printf("Invalid choice. Try again.\n");

}

} while (choice != 0);

while (!is\_empty(stack)) {

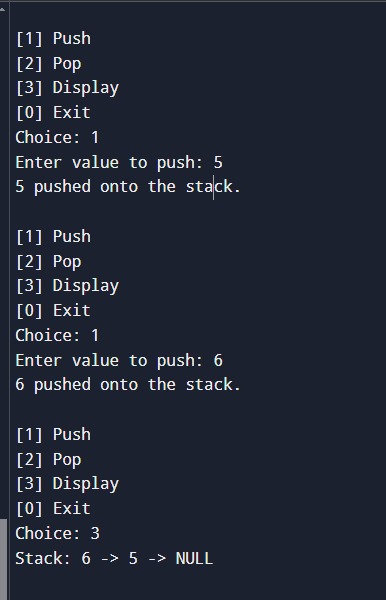
pop(&stack);

}

return 0;

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  SVKM's-IOT, Dhule  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**11 | **Date :** | | Signature |
| **Title:**  Write a program to create a binary search tree (BST) by considering the keys in given order and perform the following operations on it.  (a) Minimum key  (b) Maximum key  (c) Search for a given key | | |
|  | | |
|  | | |
|  | | |  |

Code:

#include <stdio.h>

#include <stdlib.h>

// Node structure for BST

struct Node {

int key;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int key) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->key = key;

newNode->left = newNode->right = NULL;

return newNode;

}

// Function to insert a key into the BST

struct Node\* insert(struct Node\* root, int key) {

if (root == NULL) return createNode(key);

if (key < root->key) root->left = insert(root->left, key);

else if (key > root->key) root->right = insert(root->right, key);

return root;

}

// Function to find the minimum key in the BST

int findMin(struct Node\* root) {

while (root->left) root = root->left;

return root->key;

}

// Function to find the maximum key in the BST

int findMax(struct Node\* root) {

while (root->right) root = root->right;

return root->key;

}

// Function to search for a key in the BST

int search(struct Node\* root, int key) {

if (root == NULL) return 0;

if (root->key == key) return 1;

if (key < root->key) return search(root->left, key);

return search(root->right, key);

}

int main() {

struct Node\* root = NULL;

int keys[] = {50, 30, 70, 20, 40, 60, 80};

int n = sizeof(keys) / sizeof(keys[0]);

// Insert keys into the BST

for (int i = 0; i < n; i++) root = insert(root, keys[i]);

// Perform operations

printf("Minimum key: %d\n", findMin(root));

printf("Maximum key: %d\n", findMax(root));

int searchKey = 40;

if (search(root, searchKey))

printf("Key %d is found in the BST.\n", searchKey);

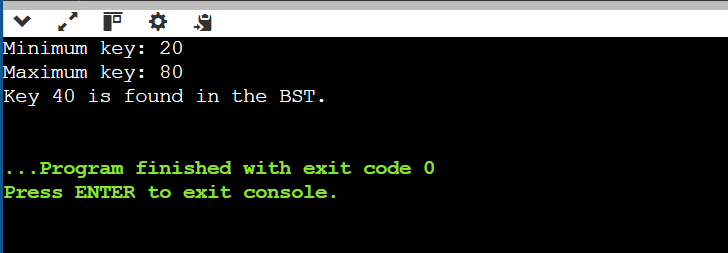
else

printf("Key %d is not found in the BST.\n", searchKey);

return 0;

}

Output:



|  |  |  |  |
| --- | --- | --- | --- |
| Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  SVKM's-IOT, Dhule  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**13 | **Date :** | | Signature |
| **Title:**  Write a program to implement hashing with              (a) Separate Chaining and              (b) Open addressing methods. | | |
|  | | |
|  | | |
|  | | |  |

Separate Chaining:

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

// Node structure for chaining

struct Node {

int key;

struct Node\* next;

};

struct Node\* hashTable[SIZE] = {NULL};

// Hash function

int hash(int key) {

return key % SIZE;

}

// Insert a key

void insert(int key) {

int index = hash(key);

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->key = key;

newNode->next = hashTable[index];

hashTable[index] = newNode;

}

// Search for a key

int search(int key) {

int index = hash(key);

struct Node\* temp = hashTable[index];

while (temp) {

if (temp->key == key) return 1;

temp = temp->next;

}

return 0;

}

// Display the hash table

void display() {

for (int i = 0; i < SIZE; i++) {

printf("%d: ", i);

struct Node\* temp = hashTable[i];

while (temp) {

printf("%d -> ", temp->key);

temp = temp->next;

}

printf("NULL\n");

}

}

int main() {

insert(10);

insert(20);

insert(15);

insert(25);

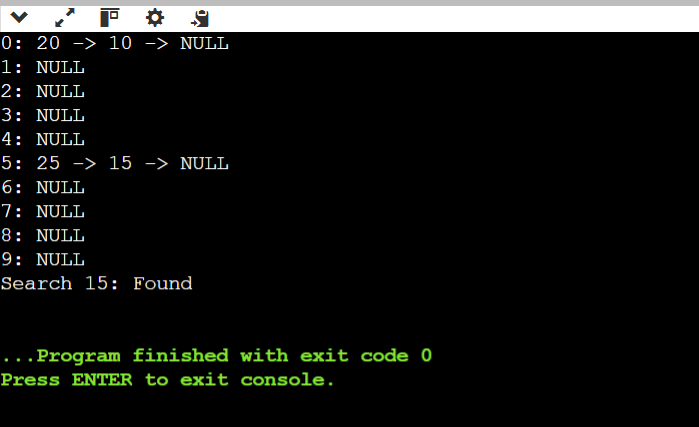
display();

printf("Search 15: %s\n", search(15) ? "Found" : "Not Found");

return 0;

}

Output:



Open Addresing:

#include <stdio.h>

#define SIZE 10

#define EMPTY -1

int hashTable[SIZE];

// Hash function

int hash(int key) {

return key % SIZE;

}

// Insert a key using linear probing

void insert(int key) {

int index = hash(key);

while (hashTable[index] != EMPTY) {

index = (index + 1) % SIZE;

}

hashTable[index] = key;

}

// Search for a key

int search(int key) {

int index = hash(key);

int start = index;

while (hashTable[index] != EMPTY) {

if (hashTable[index] == key) return 1;

index = (index + 1) % SIZE;

if (index == start) break;

}

return 0;

}

// Display the hash table

void display() {

for (int i = 0; i < SIZE; i++) {

printf("%d: %d\n", i, hashTable[i]);

}

}

int main() {

for (int i = 0; i < SIZE; i++) hashTable[i] = EMPTY;

insert(10);

insert(20);

insert(15);

insert(25);

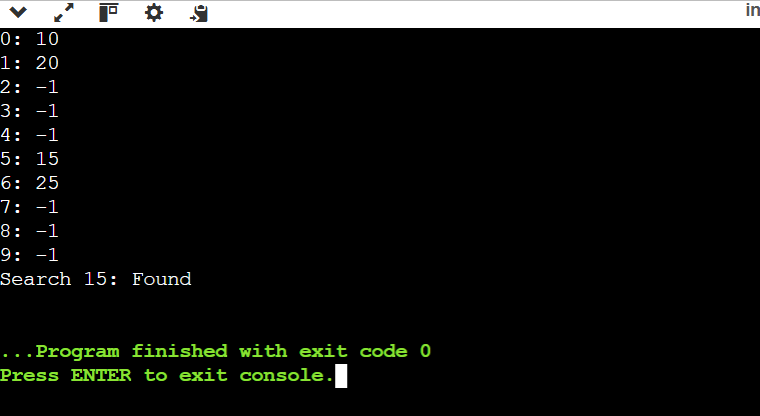
display();

printf("Search 15: %s\n", search(15) ? "Found" : "Not Found");

return 0;

}

Output:



|  |  |  |  |
| --- | --- | --- | --- |
| Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  SVKM's-IOT, Dhule  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | | | |
| **Subject :** Data Structures Lab | | | Remark |
| **Name :** | | **Roll No. :** |
| **Class :** SY. Comp. Engg. | **Batch :** | **Division:** |
| **Expt. No. :**14 | **Date :** | | Signature |
| **Title:**  Implement the following sorting algorithms:  (a) Insertion sort  (b) Merge sort  (c) Quick sort  (d) Heap sort. | | |
|  | | |
|  | | |
|  | | |  |

Insertion Sort

#include <stdio.h>

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

void display(int arr[], int n) {

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {5, 2, 9, 1, 5, 6};

int n = sizeof(arr) / sizeof(arr[0]);

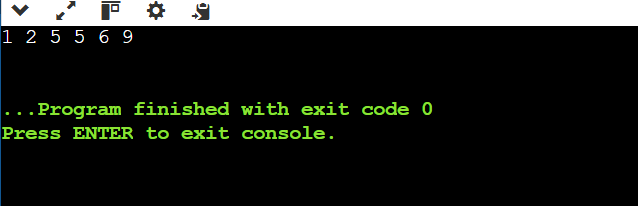
insertionSort(arr, n);

display(arr, n);

return 0;

}

Output:



Merge Sort:

#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1, n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void display(int arr[], int n) {

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

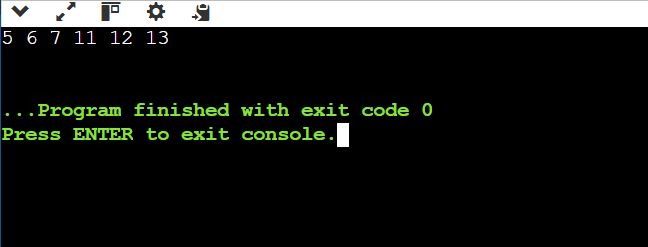
int n = sizeof(arr) / sizeof(arr[0]);

mergeSort(arr, 0, n - 1);

display(arr, n);

return 0;

}

Output:  


Quick Sort:

#include <stdio.h>

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void display(int arr[], int n) {

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr) / sizeof(arr[0]);

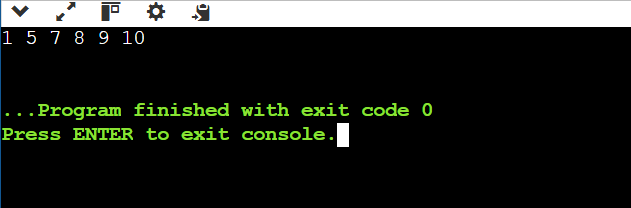
quickSort(arr, 0, n - 1);

display(arr, n);

return 0;

}

Output:



Heap Sort:

#include <stdio.h>

void heapify(int arr[], int n, int i) {

int largest = i, left = 2 \* i + 1, right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest]) largest = left;

if (right < n && arr[right] > arr[largest]) largest = right;

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

void display(int arr[], int n) {

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {4, 10, 3, 5, 1};

int n = sizeof(arr) / sizeof(arr[0]);

heapSort(arr, n);

display(arr, n);

return 0;

}

Output:  
