case 5: DisplayDQ(); break;

default: printf("Invalid choice!\n"); break;

}

printf("Enter 0 to continue: "); int n;

scanf("%d", &n); if(n != 0)

break;

} while(1);

}

## RESULT:

The Program to Implement Double Ended Queue is Successful and Output is Obtained.

## ALGORITHM:

1: To create new node, do typedef struct Node {

int data;

struct Node \*next;

}node;

2: Initialize \*head = NULL

3: Function Insertion\_Front(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->next = NULL
  5. : If head = NULL, Set head = new and go to step 3.7
  6. : Else, do
     1. : Set new->next = head
     2. : Set head = new
  7. : End

4: Function Insertion\_End(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->next = NULL
  5. : If head = NULL, Set head = new and go to step 4.7
  6. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->next!=NULL, Set ptr = ptr->next
     3. : Set ptr->next = new
  7. : End

5: Function Insertion\_Specific(key, item)

* 1. : Begin
  2. : If head = NULL, display “List is Empty” and go to step 5.4
  3. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->next!=NULL and ptr->data!=key, do

DATE: 14/10/2024

PROGRAM NO : 13

# SINGLY LINKED LIST

**AIM:** Create a C program to Implement Singly Linked List

## PROGRAM:

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

typedef struct Node{ int data;

struct Node \*next;

} node;

node \*head = NULL; void In\_Front(int x)

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->next = NULL; if(head == NULL)

{ head = new;} else

{ new->next = head; head = new; }

printf("Element added.\n");

}

void In\_End(int x)

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->next = NULL; if (head == NULL)

{ head = new; } else

{

* + - 1. : Set ptr = ptr->next
    1. : If ptr->data!=key, display “Key not Found” and go to step 5.4
    2. : Else, do
       1. : Create new node, new
       2. : Set new->data = item
       3. : Set new->next = ptr->next
       4. : Set ptr->next=new
  1. : End

6: Function Deletion\_Front()

* 1. : Begin
  2. : If head = NULL, display “List is Empty” and go to step 6.4
  3. : Else, do
     1. : Set \*temp = head

6.2.2: Set head= head->next

6.3.3: Dispose temp

* 1. : End

7: Function Deletion\_End()

* 1. : Begin
  2. : If head=NULL, display “List is Empty” and go to step 7.5
  3. : Else if head->next = NULL, do
     1. : Set \*temp = head
     2. : Set head = NULL
     3. : Dispose temp and go to step 7.5
  4. : Else, do
     1. : Set \*prev = head
     2. : Set \*cur= head->next
     3. : While cur->next!=NULL, do
        1. : Set prev = cur
        2. : Set cur = cur->next
     4. : Set prev->next = NULL
     5. : Dispose cur
  5. : End

8: Function Deletion\_Specific(key)

* 1. : Begin
  2. : If head = NULL, display “List is Empty” and go to step 8.5

void In\_spec(int key, int x)

{ if(head == NULL)

{ printf("List is Empty, Insertion not possible!\n"); else

{ node \*ptr = head;

while(ptr->data != key && ptr->next != NULL)

{ ptr = ptr->next; } if(ptr->data != key)

{ printf("Key not found!\n"); else

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->next = ptr->next; ptr->next = new;

printf("Element added.\n"); }

}

}

void Del\_Front(){

{ if (head == NULL)

{ printf("List is Empty, Deletion not possible.\n"); } else

{ node \*temp = head; head = head->next; free(temp);

printf("Element Deleted.\n"); }

}

void Del\_spec(int key){

{ if (head == NULL)

{ printf(“List is Empty, Deletion not possible.\n”); } else if(head->data == key)

{ node \*temp = head; head = head->next; free(temp);

printf(“Element Deleted.\n”); }

else

{ node \*prev = head;

* 1. : Else if head->data = key, do
     1. : Set \*temp = head
     2. : Set head = head->next
     3. : Dispose temp and go to step 8.5
  2. : Else, do
     1. : Set \*prev = head
     2. : Set \*cur = head->next
     3. : While cur->next!=NULL and cur->data!=key, do
        1. : Set prev = cur
        2. : Set cur = cur->next
     4. : If cur->data!=key, display “Key not Found” and go to step 8.5
     5. : Else, do
        1. : Set prev->next=cur->next
        2. : Dispose cur
  3. : End

9: Function Display()

* 1. : If head = NULL, display “List is Empty” and go to step 9.3
  2. : Else, do
     1. : Set \*ptr = head
     2. : While ptr!=NULL, do
        1. : Display ptr->data
        2. : Set ptr = ptr->next
  3. : End

node \*cur = head->next;

while (cur->data != key && cur->next != NULL)

{ prev = cur;

cur = cur->next; } if (cur->data != key)

{ printf(“Key not found, Deletion not possible.\n”); } else

{ prev->next = cur->next; free(cur);

printf("Element Deleted.\n"); }

}

}

void Del\_End(){

{ if (head == NULL)

{ printf(“List is Empty, Deletion not possible.\n”); } else if(head->next == NULL)

{ node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n"); }

else

{ node \*prev = head, \*cur = head->next; while (cur->next != NULL){

{ prev = cur;

cur = cur->next; } prev->next = NULL; free(cur);

printf(“Element Deleted.\n”);

}

}

void Display()

{ if (head == NULL

{ printf(“List is Empty\n”); } else

{ node \*ptr = head; printf("List: ");

while(ptr != NULL)

{ printf("%d ", ptr->data); ptr = ptr->next; }

printf("\n"); }

}

void main()

{ int c, x, key, n;

printf("1. INSERTION AT FRONT\n"); printf("2. INSERTION AT END\n"); printf("3. DELETION AT FRONT\n"); printf("4. DELETION AT END\n");

printf("5. INSERTION AFTER A SPECIFIC NODE\n"); printf("6. DELETION OF SPECIFIC NODE\n");

printf("7. DISPLAY\n"); do

{ printf("Enter your choice: "); scanf("%d", &c);

switch(c)

{ case 1: printf("Enter data: ");

scanf("%d", &x); In\_Front(x); break;

case 2: printf("Enter data: "); scanf("%d", &x); In\_End(x);

break;

case 3: Del\_Front(); break;

case 4: Del\_End();

break;

case 5: printf("Enter key: "); scanf("%d", &key); printf("Enter data: "); scanf("%d", &x); In\_spec(key, x); break;

case 6: printf("Enter key: ");

## OUTPUT :

1. INSERTION AT FRONT
2. INSERTION AT END
3. DELETION AT FRONT
4. DELETION AT END
5. INSERTION AFTER A SPECIFIC NODE
6. DELETION OF SPECIFIC NODE
7. DISPLAY

Enter your choice: 1 Enter data: 2 Element added.

Enter 0 to continue! 0 Enter your choice: 2 Enter data: 4

Element added.

Enter 0 to continue! 0 Enter your choice: 5 Enter key: 2

Enter data: 3 Element added.

Enter 0 to continue! 0 Enter your choice: 7 List: 2 3 4

Enter 0 to continue! 0 Enter your choice: 3 Element Deleted.

Enter 0 to continue! 0 Enter your choice: 4 Element Deleted.

Enter 0 to continue! 0 Enter your choice: 7 List: 3

Enter 0 to continue! 1

scanf("%d", &key); Del\_spec(key); break;

case 7: Display();

break; default:printf("Invalid choice!\n");

break;

}

printf("Enter 0 to continue! "); scanf("%d", &n);

if(n != 0)

break;

} while(1);

}

## RESULT:

The Program to Implement Singly Linked List is Successful and Output is Obtained.

## ALGORITHM:

1: To create new node, do typedef struct Node {

int data;

struct Node \*next;

}node;

2: Initialize \*top = NULL 3: Function Push(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->next = top
  5. : Set top=new
  6. : End

4: Function Pop()

* 1. : Begin
  2. : If top = NULL, display “Stack is Empty” and go to step 4.4
  3. : Else, do
     1. : Set \*temp = top
     2. : Set top = top->next
     3. : Set temp=NULL
     4. : Dispose temp
  4. : End

5: Function Peek()

* 1. : Begin
  2. : If top= NULL, display “Stack is Empty”, and go to step 5.4
  3. : Else, display top->data
  4. : End

6: Function Display()

* 1. : Begin
  2. : If top = NULL, display “Stack is Empty” and go to step 6.4
  3. : Else, do
     1. : Set \*temp = top

DATE: 14/10/2024

PROGRAM NO : 14

# STACK USING LINKED LIST

**AIM:** Create a C program to Implement Stack Using Linked List

## PROGRAM:

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

typedef struct Node

{ int data;

struct Node \*next;

} node;

node \*top = NULL; void Push(int x)

{ node \*new=(node \*)malloc(sizeof(node)); new->data=x;

new->next=top; top=new;

printf("Element Added.\n");

}

void Pop(){

{ if(top==NULL)

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

else

{ node \*temp=top;

printf("Deleted element is %d \n", temp->data); top=top->next;

temp=NULL; free(temp); }

}

void Peek()

* + 1. : While temp!= NULL, do
       1. : Display temp->data
       2. : Set temp = temp->next
  1. : End

{ if(top==NULL)

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

else

{ printf("Top Element: %d \n", top->data); }

}

void Display()

{ node \*temp=top; if (top == NULL)

printf("Stack is Empty\n");

else

{ printf("Stack: "); while(temp != NULL)

{ printf("%d ", temp->data); temp = temp->next; }

printf("\n");

}

}

void main()

{ int c, x, n;

printf("1. PUSH\n");

printf("2. POP\n");

printf("3. PEEK\n"); printf("4. DISPLAY\n"); do

{ printf("Enter your choice: "); scanf("%d", &c);

switch(c)

{ case 1: printf("Enter data: ");

scanf("%d", &x); Push(x);

break; case 2: Pop(x);

break; case 3: Peek();

break;

## OUTPUT :

1. PUSH
2. POP
3. PEEK
4. DISPLAY

Enter your choice: 1 Enter data: 1 Element Added.

Enter 0 to continue! 0 Enter your choice: 1 Enter data: 2

Element Added. Enter 0 to continue! 0 Enter your choice: 4 Stack: 2 1

Enter 0 to continue! 0 Enter your choice: 2 Deleted element is 2 Enter 0 to continue! 0 Enter your choice: 3 Top Element: 1

Enter 0 to continue! 0 Enter your choice: 4 Stack: 1

Enter 0 to continue! 1

case 4: Display();

break;

default: printf("Invalid choice!\n"); break;

}

} while(1);

}

printf("Enter 0 to continue! "); scanf("%d", &n);

if(n != 0)

break;

## RESULT:

The Program to Implement Stack using Linked List is Successful and Output is Obtained.

## ALGORITHM:

1: To create new node, do typedef struct Node {

int data;

struct Node \*next;

}node;

2: Initialize \*Front = NULL and \*Rear = NULL 3: Function Enqueue(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->next = NULL
  5. : If Front = Rear = NULL, Set Front = Rear = new , and go to step 3.7
  6. : Else, Set Rear->next=new
  7. : End

4: Function Dequeue()

* 1. : Begin
  2. : If Front = NULL, display “Queue is Empty” and go to step 4.5
  3. : Else if Front = Rear, do
     1. : Set \*temp = Front
     2. : Set Front=Rear=NULL
     3. : Dispose temp and go to step 4.5
  4. : Else, do
     1. : Set temp = Front
     2. : Set Front=Front->next
     3. : Dispose temp
  5. : End

5: Function Display()

* 1. : Begin
  2. : If Front = NULL, display “Queue is Empty”, and go to step 5.4
  3. : Else, do
     1. : Set \*pointer=Front
     2. : While pointer != NULL, do
        1. : Display pointer->data

DATE: 14/10/2024

PROGRAM NO : 15

# QUEUE USING LINKED LIST

**AIM:** Create a C program to Implement Queue using Linked List

## PROGRAM:

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

typedef struct Node

{ int data;

struct Node \*next;

} node;

node \*R = NULL; node \*F = NULL;

void Enqueue(int x)

{ node \*new=(node \*)malloc(sizeof(node)); new->data=x;

new->next=NULL; if(F==NULL && R==NULL)

{ F=new; R=new; }

else

{ R->next=new; R=new; }

printf("Element Added.\n");

}

void Dequeue()

{ if(F==NULL)

{ printf("Queue is Empty.\n");} else if(F==R)

{ node \*temp=F;

printf("Deleted element is %d \n", temp->data);

5.3.2.2: Set pointer = pointer->next

* 1. : End

F=NULL; R=NULL;

free(temp);}

else

{ node \*temp=F;

printf("Deleted element is %d \n", temp->data); F=F->next;

temp=NULL; free(temp); }

}

void Display()

{ if (F == NULL)

{ printf("Queue is Empty\n"); } else

{ node \*temp=F; printf("Queue: "); while(temp != NULL){

printf("%d ", temp->data); temp = temp->next; }

printf("\n"); }

}

void main(){

{ int c, x, n;

printf("1. ENQUEUE\n"); printf("2. DEQUEUE\n"); printf("3. DISPLAY\n"); do{

{ printf("Enter your choice: "); scanf("%d", &c);

switch(c)

{ case 1: printf("Enter data: ");

scanf("%d", &x); Enqueue(x); break;

case 2: Dequeue();

break;

## OUTPUT :

1. ENQUEUE
2. DEQUEUE
3. DISPLAY

Enter your choice: 1 Enter data: 2 Element Added.

Enter 0 to continue! 0 Enter your choice: 3 Queue: 2

Enter 0 to continue! 0 Enter your choice: 1 Enter data: 7

Element Added. Enter 0 to continue! 0 Enter your choice: 1 Enter data: 3

Element Added. Enter 0 to continue! 0 Enter your choice: 3 Queue: 2 7 3

Enter 0 to continue! 0 Enter your choice: 2 Deleted element is 2 Enter 0 to continue! 0 Enter your choice: 3 Queue: 7 3

Enter 0 to continue! 1

case 3: Display();

break;

default: printf("Invalid choice!\n"); break;

}

printf("Enter 0 to continue! "); scanf("%d", &n);

if(n != 0)

break;

} while(1);

}

## RESULT:

The Program to Implement Queue using Linked List is Successful and Output is Obtained.

## ALGORITHM:

1: To create new node, do typedef struct Node {

int data;

struct Node \*rlink, \*llink;

}node;

2: Initialize \*head = NULL

3: Function Insertion\_Front(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->rlink = NULL and new->llink = NULL
  5. : If head = NULL, Set head = new and go to step 3.7
  6. : Else, do
     1. : Set new->rlink= head and head->llink = new
     2. : Set head = new
  7. : End

4: Function Insertion\_End(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->rlink = NULL and new->llink = NULL
  5. : If head = NULL, Set head = new and go to step 4.7
  6. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->rlink!=NULL, Set ptr = ptr->rlink
     3. : Set ptr->rlink = new and new->llink=ptr
  7. : End

5: Function Insertion\_Specific(key, item)

* 1. : Begin
  2. : If head = NULL, display “List is Empty” and go to step 5.4
  3. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->rlink!=NULL and ptr->data!=key, do

DATE: 19/10/2024

PROGRAM NO : 16

# DOUBLY LINKED LIST

**AIM:** Create a C program to Implement Doubly Linked List

## PROGRAM:

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

typedef struct Node

{ int data;

struct Node \*rlink,\*llink;

} node;

node \*head = NULL; void In\_Front(int x){

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->rlink = NULL; new->llink = NULL; if(head == NULL)

{ head = new; } else

{ new->rlink = head; head->llink = new; head = new; }

printf("Element added.\n");

}

void In\_End(int x)

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->rlink = NULL; new->llink = NULL; if (head == NULL)

* + - 1. : Set ptr = ptr->rlink
    1. : If ptr->data!=key, display “Key not Found” and go to step 5.4
    2. : Else, do
       1. : Create new node, new
       2. : Set new->data = item
       3. : Set new->llink = ptr
       4. : Set new->rlink = ptr->rlink
       5. : Set new->rlink->llink= new
       6. : Set ptr->rlink=new
  1. : End

6: Function Deletion\_Front()

* 1. : Begin
  2. : If head = NULL, display “List is Empty” and go to step 6.5
  3. : Else if head->rlink = NULL, do
     1. : Set \*temp = head

6.2.2: Set head= NULL

6.3.3: Dispose temp

* 1. : Else, do
     1. : Set \*temp=head
     2. : Set head=head->rlink
     3. : Set head->llink=NULL
     4. : Dispose temp
  2. : End

7: Function Deletion\_End()

* 1. : Begin
  2. : If head=NULL, display “List is Empty” and go to step 7.5
  3. : Else if head->rlink = NULL, do
     1. : Set \*temp = head
     2. : Set head = NULL
     3. : Dispose temp and go to step 7.5
  4. : Else, do
     1. : Set \*ptr= head
     2. : While ptr->rlink!=NULL, do
        1. : Set ptr = ptr->rlink
     3. : Set ptr->llink->rlink = NULL

7.4.5: Set ptr->llink=NULL

head = new;

else

{ node \*ptr = head; while(ptr->rlink != NULL)

{ ptr = ptr->rlink; } ptr->rlink = new;

new->llink=ptr; } printf("Element added.\n");

}

void In\_spec(int key, int x)

{ if(head == NULL)

{ printf("List is Empty, Insertion not possible!\n"); } else

{ node \*ptr = head;

while(ptr->data != key && ptr->rlink != NULL)

{ ptr = ptr->rlink; } if(ptr->data != key)

printf("Key not found!\n");

else if(ptr->rlink!=NULL && ptr->data==key)

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->llink = ptr;

new->rlink = ptr->rlink; new->rlink->llink = new; ptr->rlink = new;

printf("Element added.\n"); }

else

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->llink = ptr; new->rlink=NULL; ptr->rlink = new;

printf("Element added.\n"); }

}

}

void Del\_Front()

7.4.6: Set ptr->rlink=NULL

* 1. : End

8: Function Deletion\_Specific(key)

* 1. : Begin
  2. : If head = NULL, display “List is Empty” and go to step 8.5
  3. : Else if head->rlink = NULL, do
     1. : If head->data = key, do
        1. : Set \*temp=head
        2. : Set head=NULL
        3. : Dispose temp and go to step 8.5
     2. : Else, display “Search not found” and go to step 8.5
  4. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->rlink !=NULL and ptr->data!=key, do

8.4.3.1: Set ptr = ptr->rlink

* + 1. : If ptr->data!=key, display “Search not Found” and go to step 8.5
    2. : Else, do
       1. : Set ptr->llink->rlink = ptr->rlink
  1. : End

9: Function Display()

* 1. : If head = NULL, display “List is Empty” and go to step 9.3
  2. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->rlink!=NULL, do
        1. : Display ptr->data
        2. : Set ptr = ptr->rlink
  3. : End

{ if (head == NULL)

{ printf("List is Empty, Deletion not possible.\n"); } else if(head->rlink==NULL) {

node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n"); }

else{

}

node \*temp=head; head=head->rlink; head->llink=NULL; free(temp);

printf("Element Deleted.\n"); }

void Del\_spec(int key)

{ if (head == NULL){

printf("List is Empty, Deletion not possible.\n"); } else if(head->rlink == NULL){

if(head->data==key){

node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n"); }

else{

printf("Search not found, deletion not possible!");}}

else {

node \*ptr = head;

while (ptr->data != key && ptr->rlink != NULL){ ptr = ptr->rlink; }

if (ptr->data == key && ptr->rlink!=NULL){ ptr->llink->rlink=ptr->rlink;

ptr->rlink->llink=ptr->llink; free(ptr);

printf("Element Deleted.\n"); }

else if(ptr->data == key && ptr->rlink==NULL){ ptr->llink->rlink=NULL;

free(ptr);

else {

}

printf("Element Deleted.\n");}

printf("Key not found, Deletion not possible.\n"); } }

void Del\_End()

{ if (head == NULL){

printf("List is Empty, Deletion not possible.\n"); } else if(head->rlink == NULL){

node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n"); }

else {

}

node \*ptr = head;

while (ptr->rlink != NULL){ ptr = ptr->rlink; }

ptr->llink->rlink = NULL; ptr->llink=NULL;

ptr->rlink=NULL; printf("Element Deleted.\n"); }

void Display()

{ if (head == NULL){

printf("List is Empty\n"); }

else {

}

node \*ptr = head; printf("List: ");

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

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

printf("\n"); }

void main()

{ int c, item, x, key, n;

## OUTPUT :

1. INSERTION AT FRONT
2. INSERTION AT END
3. DELETION AT FRONT
4. DELETION AT END
5. INSERTION AFTER A SPECIFIC NODE
6. DELETION OF SPECIFIC NODE
7. DISPLAY

Enter your choice: 1 Enter data: 3 Element added.

Enter 0 to continue! 0 Enter your choice: 2 Enter data: 4

Element added.

Enter 0 to continue! 0 Enter your choice: 7 List: 3 4

Enter 0 to continue! 0 Enter your choice: 5 Enter key: 3

Enter data: 5 Element added.

Enter 0 to continue! 0 Enter your choice: 7 List: 3 5 4

Enter 0 to continue! 0 Enter your choice: 6 Enter key: 5

Element Deleted. Enter 0 to continue! 0 Enter your choice: 3 Element Deleted.

Enter 0 to continue! 0 Enter your choice: 2 Enter data: 7

Element added.

Enter 0 to continue! 0

printf("1. INSERTION AT FRONT\n"); printf("2. INSERTION AT END\n"); printf("3. DELETION AT FRONT\n"); printf("4. DELETION AT END\n");

printf("5. INSERTION AFTER A SPECIFIC NODE\n"); printf("6. DELETION OF SPECIFIC NODE\n");

printf("7. DISPLAY\n"); do

{ printf("Enter your choice: "); scanf("%d", &c);

switch(c)

{ case 1: printf("Enter data: ");

scanf("%d", &x); In\_Front(x); break;

case 2: printf("Enter data: "); scanf("%d", &x); In\_End(x);

break;

case 3: Del\_Front(); break;

case 4: Del\_End();

break;

case 5: printf("Enter key: "); scanf("%d", &key); printf("Enter data: "); scanf("%d", &x); In\_spec(key, x); break;

case 6: printf("Enter key: "); scanf("%d", &key); Del\_spec(key); break;

case 7: Display();

break;

default: printf("Invalid choice!\n"); break;

}

Enter your choice: 7 List: 4 7

Enter 0 to continue! 1

printf("Enter 0 to continue! "); scanf("%d", &n);

if(n != 0)

break;

} while(1);

}

## RESULT:

The Program to Implement Doubly Linked List is Successful and Output is Obtained

## ALGORITHM:

1: To create new node, do typedef struct Node {

int data;

struct Node \*next;

}node;

2: Initialize \*head = NULL

3: Function Insertion\_Front(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->next = NULL
  5. : If head = NULL, do
     1. : Set head = new
     2. : Set head->next = head and go to step 3.7
  6. : Else
     1. : Set \*ptr = head
     2. : While ptr->next != head, Set ptr = ptr->next
     3. : Set new->next = head
     4. : Set head = new
     5. : Set ptr->next = head
  7. : End

4: Function Insertion\_End(item)

* 1. : Begin
  2. : Create a new node, new
  3. : Set new->data = item
  4. : Set new->next = NULL
  5. : If head = NULL
     1. : Set head = new
     2. : Set head->next = head and go to step 4.7
  6. : Else, do
     1. : Set \*ptr = head
     2. : While ptr->next != head, Set ptr = ptr->next
     3. : Set ptr->next = new
     4. : Set new->next = head

DATE: 19/10/2024

PROGRAM NO : 17

# CIRCULAR LINKED LIST

**AIM:** Create a C program to Implement Circular Linked List

## PROGRAM:

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

typedef struct Node

{ int data;

struct Node \*next;

} node;

node \*head = NULL; void In\_Front(int x)

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->next = NULL; if(head == NULL){

head = new;

head->next=new; }

else {

}

node \*ptr=head; while(ptr->next!=head){

ptr=ptr->next;} new->next = head; head = new;

ptr->next=head; } printf("Element added.\n");

void In\_End(int x)

{ node \*new = (node \*)malloc(sizeof(node)); new->data = x;

* 1. : End

5: Function Insertion\_Specific(key, item)

* 1. : Begin
  2. : If head = NULL
     1. : Display "List is Empty, Insertion not possible!"
  3. : Else
     1. : Set \*ptr = head
     2. : While ptr->data != key and ptr->next != head, Set ptr = ptr->next
     3. : If ptr->data != key, Display "Key not found!"
     4. : Else
        1. : Create a new node, new
        2. : Set new->data = item
        3. : Set new->next = ptr->next
        4. : Set ptr->next = new
  4. : End

6: Function Deletion\_Front()

* 1. : Begin
  2. : If head = NULL

Display "List is Empty, Deletion not possible."

* 1. : Else if head->next == head
     1. : Set \*temp=head
     2. : Set head = NULL
     3. : Dispose temp
  2. : Else
     1. : Set \*temp = head
     2. : Set \*ptr = head
     3. : While ptr->next != head, Set ptr = ptr->next
     4. : Set head = head->next
     5. : Set ptr->next = head
     6. : Dispose temp
  3. : End

7: Function Deletion\_End()

* 1. : Begin
  2. : If head = NULL, Display "List is Empty, Deletion not possible."
  3. : Else if head->next == head, do

new->next = NULL; if (head == NULL){ head = new;

head->next=head; } else {

node \*ptr = head; while(ptr->next != head){

ptr = ptr->next; } ptr->next = new;

new->next=head; } printf("Element added.\n");

}

void In\_spec(int key, int x)

{ if(head == NULL){

printf("List is Empty, Insertion not possible!\n"); }

else {

node \*ptr = head;

while(ptr->data != key && ptr->next != head){ ptr = ptr->next; }

if(ptr->data != key){

printf("Key not found!\n"); }

else {

}

node \*new = (node \*)malloc(sizeof(node)); new->data = x;

new->next = ptr->next; ptr->next = new;

printf("Element added.\n"); } }

void Del\_Front()

{ if (head == NULL){

printf("List is Empty, Deletion not possible.\n"); } else if(head->next==head) {

node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n"); }

* + 1. : Set \*temp= head
    2. : Set head = NULL
    3. : Dispose temp
  1. : Else
     1. : Set \*prev = head
     2. : Set \*cur = head->next
     3. : While cur->next != head, do
        1. : Set prev = cur

7.3.3.2: Set cur = cur->next

* + 1. : Set prev->next = head
    2. : Dispose cur
  1. : End

8: Function Deletion\_Specific(key)

* 1. : Begin
  2. : If head = NULL, Display "List is Empty, Deletion not possible."
  3. : Else if head->next = head, do
     1. : If head->data = key, do
        1. : Set \*temp = head
        2. : Set head = NULL
        3. : Dispose temp
     2. : Else, Display "Key not found, Deletion not possible."
  4. : Else
     1. : Set \*prev = head
     2. : Set \*cur = head
     3. : While cur->data != key and cur->next != head, do
        1. : Set prev = cur
        2. : Set cur = cur->next
     4. : If cur->data != key, Display "Key not found, Deletion not possible."
     5. : Else, do
        1. : Set prev->next = cur->next
        2. : Dispose cur
  5. : End

9: Function Display()

* 1. : Begin
  2. : If head = NULL, Display "List is Empty"
  3. : Else, do

else{

}

node \*temp=head; node \*ptr=head;

while(ptr->next!=head){

ptr=ptr->next;} head=head->next;

ptr->next=head; printf("Element Deleted.\n"); }

void Del\_spec(int key)

{ if (head == NULL){

printf("List is Empty, Deletion not possible.\n"); } else if(head->next == head){

if(head->data==key){

node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n");}

else{

printf("Key not found, Deletion not possible.\n"); }

else {

node \*prev = head; node \*cur = head;

while (cur->data != key && cur->next != head){ prev = cur;

cur = cur->next; } if (cur->data != key){

printf("Key not found, Deletion not possible.\n"); }

else {

}

prev->next = cur->next; free(cur);

printf("Element Deleted.\n"); } }

void Del\_End()

{ if (head == NULL){

printf("List is Empty, Deletion not possible.\n"); }

* + 1. : Set \*ptr = head
    2. : While ptr->next != head
       1. : Display ptr->data
       2. : Set ptr = ptr->next
  1. : End

## OUTPUT :

* + 1. INSERTION AT FRONT
    2. INSERTION AT END
    3. DELETION AT FRONT
    4. DELETION AT END
    5. INSERTION AFTER A SPECIFIC NODE
    6. DELETION OF SPECIFIC NODE
    7. DISPLAY

Enter your choice: 1 Enter data: 5 Element added.

Enter 0 to continue! 0 Enter your choice: 2 Enter data: 7

Element added.

Enter 0 to continue! 0 Enter your choice: 5 Enter key: 5

Enter data: 6 Element added.

Enter 0 to continue! 0 Enter your choice: 7 List: 5 6 7

Enter 0 to continue! 0 Enter your choice: 6 Enter key: 6

Element Deleted. Enter 0 to continue! 0 Enter your choice: 4 Element Deleted.

else if(head->next == head)

{ node \*temp = head; head = NULL; free(temp);

printf("Element Deleted.\n"); }

else {

}

node \*prev = head, \*cur = head->next; while (cur->next != head){

prev = cur;

cur = cur->next; } prev->next = head; free(cur);

printf("Element Deleted.\n"); }

void Display()

{ if (head == NULL){

printf("List is Empty\n"); }

else {

}

node \*ptr = head; printf("List: ");

while(ptr->next != head){ printf("%d ", ptr->data); ptr = ptr->next; }

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

printf("\n");}

void main()

{ int c, item, x, key, n;

printf("1. INSERTION AT FRONT\n"); printf("2. INSERTION AT END\n"); printf("3. DELETION AT FRONT\n"); printf("4. DELETION AT END\n");

printf("5. INSERTION AFTER A SPECIFIC NODE\n"); printf("6. DELETION OF SPECIFIC NODE\n");

printf("7. DISPLAY\n"); do{

Enter 0 to continue! 0 Enter your choice: 7 List: 5

Enter 0 to continue! 0 Enter your choice: 3 Element Deleted.

Enter 0 to continue! 0 Enter your choice: 7 List is Empty

Enter 0 to continue! 1

if(n != 0) break;

} while(1);

}

printf("Enter your choice: "); scanf("%d", &c);

switch(c)

{ case 1: printf("Enter data: ");

scanf("%d", &x); In\_Front(x); break;

case 2: printf("Enter data: "); scanf("%d", &x); In\_End(x);

break;

case 3: Del\_Front(); break;

case 4: Del\_End();

break;

case 5: printf("Enter key: "); scanf("%d", &key); printf("Enter data: "); scanf("%d", &x); In\_spec(key, x); break;

case 6: printf("Enter key: "); scanf("%d", &key); Del\_spec(key); break;

case 7: Display();

break;

default: printf("Invalid choice!\n"); break;

printf("Enter 0 to continue! "); scanf("%d", &n);

## RESULT:

The Program to Implement Circular Linked List is Successful and Output is Obtained.

## ALGORITHM:

1: To create new node, do typedef struct Node {

int exp, coef; struct Node \*next;

}node;

2: Initialize \*phead = NULL, \*qhead=NULL, \*rhead=NULL 3: Function add(phead, qhead)

* 1. : Begin
  2. : Set \*p = phead
  3. : Set \*q = qhead
  4. : Set \*r = NULL
  5. : While p != NULL and q != NULL
     1. : Create a new node, new\_node
     2. : If p->exp = q->exp
        1. : Set new\_node->coef = p->coef + q->coef
        2. : Set new\_node->exp = p->exp
        3. : Set p = p->next
        4. : Set q = q->next
     3. : Else if p->exp > q->exp, do
        1. : Set new\_node->coef = p->coef
        2. : Set new\_node->exp = p->exp
        3. : Set p = p->next
     4. : Else, do
        1. : Set new\_node->coef = q->coef
        2. : Set new\_node->exp = q->exp
        3. : Set q = q->next
     5. : Set new\_node->next = NULL
     6. : If rhead == NULL
        1. : Set rhead = new\_node
        2. : Set r = new\_node
     7. : Else
        1. : Set r->next = new\_node
        2. : Set r = new\_node
  6. : While p != NULL
     1. : Create a new node, new\_node

DATE: 04/11/2024

PROGRAM NO : 18

# POLYNOMIAL ADDITION USING LINKED LIST

**AIM:** Create a C program to Implement Polynomial addition using Linked List

## PROGRAM:

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

typedef struct Node {

{ int coef, exp; struct Node \*next;

} node;

node \*phead = NULL; node \*qhead = NULL; node \*rhead = NULL;

void add(node \*phead, node \*qhead)

{ node \*p = phead; node \*q = qhead; node \*r = NULL;

while (p != NULL && q != NULL) {

node \*new\_node = (node \*)malloc(sizeof(node)); if (p->exp == q->exp) {

new\_node->coef = p->coef + q->coef; new\_node->exp = p->exp;

p = p->next;

q = q->next; }

else if (p->exp > q->exp) { new\_node->coef = p->coef; new\_node->exp = p->exp; p = p->next; }

else {

new\_node->coef = q->coef; new\_node->exp = q->exp;

* + 1. : Set new\_node->coef = p->coef
    2. : Set new\_node->exp = p->exp
    3. : Set new\_node->next = NULL
    4. : Set p = p->next
    5. : If rhead = NULL
       1. : Set rhead = new\_node
       2. : Set r = new\_node
    6. : Else
       1. : Set r->next = new\_node
       2. : Set r = new\_node
  1. : While q != NULL
     1. : Create a new node, new\_node
     2. : Set new\_node->coef = q->coef
     3. : Set new\_node->exp = q->exp
     4. : Set new\_node->next = NULL
     5. : Move q = q->next
     6. : If rhead = NULL
        1. : Set rhead = new\_node
        2. : Set r = new\_node
     7. : Else
        1. : Set r->next = new\_node
        2. : Set r = new\_node
  2. : End

4: Function display(head)

* 1. : Begin
  2. : If head == NULL, Display "Polynomial is Empty" and go to step 4.4
  3. : Else
     1. : Set \*ptr = head
     2. : While ptr != NULL
        1. : Display ptr->coef and ptr->exp
        2. : Set ptr = ptr->next
        3. : If ptr != NULL, output " + "
  4. : End

5: Function Read\_Polynomial()

* 1. : Begin
  2. : Input number of terms, n

q = q->next; } new\_node->next = NULL; if (rhead == NULL) {

rhead = new\_node; r = new\_node; }

else {

r->next = new\_node; r = new\_node; } }

while (p != NULL) {

node \*new\_node = (node \*)malloc(sizeof(node)); new\_node->coef = p->coef;

new\_node->exp = p->exp; new\_node->next = NULL; p = p->next;

if (rhead == NULL) {

rhead = new\_node; r = new\_node; }

else {

}

r->next = new\_node; r = new\_node; }

while (q != NULL) {

node \*new\_node = (node \*)malloc(sizeof(node)); new\_node->coef = q->coef;

new\_node->exp = q->exp; new\_node->next = NULL; q = q->next;

if (rhead == NULL) {

rhead = new\_node; r = new\_node; }

else {

}

r->next = new\_node; r = new\_node; }

} }

void display(node \*head)

* 1. : Set \*head = NULL
  2. : For i from 1 to n
     1. : Create a new node, new\_node
     2. : Input new\_node->coef
     3. : Input new\_node->exp
     4. : Set new\_node->next = head
     5. : Set head = new\_node
  3. : Return head
  4. : End

{ if (rhead == NULL) { printf("Polynomial is Empty\n"); }

else {

node \*ptr = rhead; while (ptr != NULL) {

printf("%dx^%d", ptr->coef, ptr->exp); ptr = ptr->next;

if (ptr != NULL) { printf(" + "); } }

printf("\n"); }

}

void main()

{ int n, m;

printf("Polynomial addition\n");

printf("Enter number of terms of first polynomial: "); scanf("%d", &n);

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

node \*new\_node = (node \*)malloc(sizeof(node)); printf("Enter coefficient of term %d: ", (i + 1)); scanf("%d", &new\_node->coef);

printf("Enter exponent of term %d: ", (i + 1)); scanf("%d", &new\_node->exp);

new\_node->next = phead; phead = new\_node; }

printf("Enter number of terms of second polynomial: "); scanf("%d", &m);

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

node \*new\_node = (node \*)malloc(sizeof(node)); printf("Enter coefficient of term %d: ", (i + 1)); scanf("%d", &new\_node->coef);

printf("Enter exponent of term %d: ", (i + 1)); scanf("%d", &new\_node->exp);

new\_node->next = qhead; qhead = new\_node; }

add(phead, qhead); display(rhead);

}

## OUTPUT :

Polynomial addition

Enter number of terms of first polynomial: 2 Enter coefficient of term 1: 2

Enter exponent of term 1: 1

Enter coefficient of term 2: 3

Enter exponent of term 2: 2

Enter number of terms of second polynomial: 3 Enter coefficient of term 1: 4

Enter exponent of term 1: 1

Enter coefficient of term 2: 5

Enter exponent of term 2: 3

Enter coefficient of term 3: 7

Enter exponent of term 3: 7 7x^7 + 5x^3 + 3x^2 + 6x^1

## RESULT:

The Program to Implement Polynomial addition using Linked List is Successful and Output is Obtained.

## ALGORITHM:

1: To create new node, do typedef struct Node {

int data;

struct Node \*lchild, \*rchild;

}node;

2: Function Insertion (item)

* 1. : Begin
  2. : If root=NULL, do
     1. : Create new node
     2. : Set new->data = item
     3. : Set new->Lchild =NULL
     4. : Set new->Rchild =NULL
     5. : Set root = new
  3. : Else, do
     1. : Set \*ptr=root
     2. : Set flag=0
     3. : While ptr!=NULL and flag!=0, do
        1. : If item=ptr->data, Display “Insertion not possible , key already exists”
        2. : If ptr->data <item , set ptr = ptr->rchild
        3. : Else , set ptr = ptr->lchild
  4. : If ptr = NULL
     1. : Create new node
     2. : Set new->data = item
     3. : new->lchild=NULL
     4. : new->rchild=NULL
     5. : If ptr->data<item, Set ptr->rchild = new
     6. : Else, Set ptr->lchild= new
  5. : End

DATE: 04/11/2024

PROGRAM NO : 19

# BINARY SEARCH TREE

**AIM:** Create a C program to Implement Binary Search Tree

## PROGRAM:

#include<stdio.h> #include<stdlib.h> typedef struct Node

{ int info;

struct Node \*lchild, \*rchild;

}node;

node \*root=NULL; node\* newnode(int val)

{ node\*p=(node\*)malloc(sizeof(node)); p->info=val;

p->lchild=p->rchild=NULL; return p;

}

void insert(int val)

{ if(root==NULL) root=newnode(val);

else

{ node\* par=NULL; node\* curr=root; while(curr!=NULL)

{ par=curr; if(val<curr->info)

curr=curr->lchild;

else

curr=curr->rchild;

}

if(val<par->info)

## OUTPUT :

1. Insert
2. Display
3. Quit

Enter your choice:1

Enter the element to be inserted 5

Enter your choice:1

Enter the element to be inserted 4

Enter your choice:1

Enter the element to be inserted 7

Enter your choice:1

Enter the element to be inserted 8

Enter your choice:2

4 5 7 8

Enter your choice:3 Exiting...

else

}

}

par->lchild=newnode(val); par->rchild=newnode(val);

void inorder(node\* p)

{ if(p!=NULL)

{ inorder(p->lchild); printf("%d\t", p->info); inorder(p->rchild); }

}

void main()

{ int choice,val;

printf("\n1. Insert\n2. Display\n3. Quit\n"); do

{ printf("Enter your choice:"); scanf("%d", &choice); switch(choice)

{ case 1: printf("Enter the element to be inserted\n"); scanf("%d", &val);

insert(val); break;

case 2: inorder(root);

printf("\n"); break;

case 3: printf("Exiting...\n"); break;

default: printf("Invalid choice.\n Enter a valid choice from 1 to 3\n");

}

}while(choice!=3);

}

## RESULT:

The Program to Implement Binary Search Tree is Successful and Output is Obtained.

## ALGORITHM:

1: Build maxheap(A[], n) 2: For i = 0 to n-1, do

* 1. : Swap A[0], with A[i]
  2. : Heapify(A,i,n)

3: Build Max\_Heap(A,n)

* 1. : for i = n/2-1 to 0, Heapify(A,n,i)

4: Function Heapify(A,i,n)

1. 1: Begin
   1. : Set L=left child index(i)
   2. : Set R=right child index(i)
   3. : Set largest = i
   4. : If n<L and A[L]> A[largest], set largest = L
   5. : If n>R and A[R]> A[largest], set largest = R
   6. : If largest!= i , Swap A[i] with largest
   7. : Heapify

DATE: 04/11/2024

PROGRAM NO : 20

# HEAP SORT

**AIM:** Create a C program to Implement Heap Sort

## PROGRAM:

#include<stdio.h>

void swap(int \*a,int \*b)

{ int temp=\*a;

\*a=\*b;

\*b=temp;

}

void heapify(int a[], int size, int i)

{ int lc=2\*i;

int rc=(2\*i)+1; int large=i;

if(lc<=size && a[lc]>a[large]) large=lc;

if(rc<=size && a[rc]>a[large]) large=rc;

if(large!=i)

{ swap(&a[i], &a[large]); heapify(a,size,large); }

}

void buildheap(int a[], int n)

{ for(int i=n/2;i>=1;i--)

heapify(a, n, i);

}

void heapsort(int a[], int n)

{ buildheap(a,n); for(int i=n;i>=1;i--)

{ swap(&a[1], &a[i]);

heapify(a,i-1,1); } }

## OUTPUT :

Enter the number of elements 5

Enter 5 elements

6

1

8

7

2

1 2 6 7 8

void main()

{ int size;

printf("Enter the number of elements\n"); scanf("%d",&size);

int a[size+1];

printf("Enter %d elements\n", size); for(int i=1;i<=size;i++)

scanf("%d", &a[i]); heapsort(a, size);

for(int i=1;i<=size;i++)

printf("%d\t", a[i]);

}

## RESULT:

The Program to Implement Heap Sort is Successful and Output is Obtained.