C++ Program To Implement Circular Singly Linked List

This C++ Program demonstrates circular single linked list.

Here is source code of the C++ Program to demonstrate circular single linked list. The C++ program is successfully compiled and run on a Linux system. The program output is also shown below.

*/\**

*\* C++ Program to Implement Circular Linked List*

*\*/*

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

*/\**

*\* Node Declaration*

*\*/*

struct node

{

int info;

struct node \*next;

}\*last;

*/\**

*\* Class Declaration*

*\*/*

class circular\_llist

{

public:

void create\_node(int value);

void add\_begin(int value);

void add\_after(int value, int position);

void delete\_element(int value);

void search\_element(int value);

void display\_list();

void update();

void sort();

circular\_llist()

{

last = NULL;

}

};

*/\**

*\* Main :contains menu*

*\*/*

int main()

{

int choice, element, position;

circular\_llist cl;

while (1)

{

cout<<endl<<"---------------------------"<<endl;

cout<<endl<<"Circular singly linked list"<<endl;

cout<<endl<<"---------------------------"<<endl;

cout<<"1.Create Node"<<endl;

cout<<"2.Add at beginning"<<endl;

cout<<"3.Add after"<<endl;

cout<<"4.Delete"<<endl;

cout<<"5.Search"<<endl;

cout<<"6.Display"<<endl;

cout<<"7.Update"<<endl;

cout<<"8.Sort"<<endl;

cout<<"9.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter the element: ";

cin>>element;

cl.create\_node(element);

cout<<endl;

break;

case 2:

cout<<"Enter the element: ";

cin>>element;

cl.add\_begin(element);

cout<<endl;

break;

case 3:

cout<<"Enter the element: ";

cin>>element;

cout<<"Insert element after position: ";

cin>>position;

cl.add\_after(element, position);

cout<<endl;

break;

case 4:

if (last == NULL)

{

cout<<"List is empty, nothing to delete"<<endl;

break;

}

cout<<"Enter the element for deletion: ";

cin>>element;

cl.delete\_element(element);

cout<<endl;

break;

case 5:

if (last == NULL)

{

cout<<"List Empty!! Can't search"<<endl;

break;

}

cout<<"Enter the element to be searched: ";

cin>>element;

cl.search\_element(element);

cout<<endl;

break;

case 6:

cl.display\_list();

break;

case 7:

cl.update();

break;

case 8:

cl.sort();

break;

case 9:

exit(1);

break;

default:

cout<<"Wrong choice"<<endl;

}

}

return 0;

}

*/\**

*\* Create Circular Link List*

*\*/*

void circular\_llist::create\_node(int value)

{

struct node \*temp;

temp = new(struct node);

temp->info = value;

if (last == NULL)

{

last = temp;

temp->next = last;

}

else

{

temp->next = last->next;

last->next = temp;

last = temp;

}

}

*/\**

*\* Insertion of element at beginning*

*\*/*

void circular\_llist::add\_begin(int value)

{

if (last == NULL)

{

cout<<"First Create the list."<<endl;

return;

}

struct node \*temp;

temp = new(struct node);

temp->info = value;

temp->next = last->next;

last->next = temp;

}

*/\**

*\* Insertion of element at a particular place*

*\*/*

void circular\_llist::add\_after(int value, int pos)

{

if (last == NULL)

{

cout<<"First Create the list."<<endl;

return;

}

struct node \*temp, \*s;

s = last->next;

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

{

s = s->next;

if (s == last->next)

{

cout<<"There are less than ";

cout<<pos<<" in the list"<<endl;

return;

}

}

temp = new(struct node);

temp->next = s->next;

temp->info = value;

s->next = temp;

*/\*Element inserted at the end\*/*

if (s == last)

{

last=temp;

}

}

*/\**

*\* Deletion of element from the list*

*\*/*

void circular\_llist::delete\_element(int value)

{

struct node \*temp, \*s;

s = last->next;

*/\* If List has only one element\*/*

if (last->next == last && last->info == value)

{

temp = last;

last = NULL;

free(temp);

return;

}

if (s->info == value) */\*First Element Deletion\*/*

{

temp = s;

last->next = s->next;

free(temp);

return;

}

while (s->next != last)

{

*/\*Deletion of Element in between\*/*

if (s->next->info == value)

{

temp = s->next;

s->next = temp->next;

free(temp);

cout<<"Element "<<value;

cout<<" deleted from the list"<<endl;

return;

}

s = s->next;

}

*/\*Deletion of last element\*/*

if (s->next->info == value)

{

temp = s->next;

s->next = last->next;

free(temp);

last = s;

return;

}

cout<<"Element "<<value<<" not found in the list"<<endl;

}

*/\**

*\* Search element in the list*

*\*/*

void circular\_llist::search\_element(int value)

{

struct node \*s;

int counter = 0;

s = last->next;

while (s != last)

{

counter++;

if (s->info == value)

{

cout<<"Element "<<value;

cout<<" found at position "<<counter<<endl;

return;

}

s = s->next;

}

if (s->info == value)

{

counter++;

cout<<"Element "<<value;

cout<<" found at position "<<counter<<endl;

return;

}

cout<<"Element "<<value<<" not found in the list"<<endl;

}

*/\**

*\* Display Circular Link List*

*\*/*

void circular\_llist::display\_list()

{

struct node \*s;

if (last == NULL)

{

cout<<"List is empty, nothing to display"<<endl;

return;

}

s = last->next;

cout<<"Circular Link List: "<<endl;

while (s != last)

{

cout<<s->info<<"->";

s = s->next;

}

cout<<s->info<<endl;

}

*/\**

*\* Update Circular Link List*

*\*/*

void circular\_llist::update()

{

int value, pos, i;

if (last == NULL)

{

cout<<"List is empty, nothing to update"<<endl;

return;

}

cout<<"Enter the node position to be updated: ";

cin>>pos;

cout<<"Enter the new value: ";

cin>>value;

struct node \*s;

s = last->next;

for (i = 0;i < pos - 1;i++)

{

if (s == last)

{

cout<<"There are less than "<<pos<<" elements.";

cout<<endl;

return;

}

s = s->next;

}

s->info = value;

cout<<"Node Updated"<<endl;

}

*/\**

*\* Sort Circular Link List*

*\*/*

void circular\_llist::sort()

{

struct node \*s, \*ptr;

int temp;

if (last == NULL)

{

cout<<"List is empty, nothing to sort"<<endl;

return;

}

s = last->next;

while (s != last)

{

ptr = s->next;

while (ptr != last->next)

{

if (ptr != last->next)

{

if (s->info > ptr->info)

{

temp = s->info;

s->info = ptr->info;

ptr->info = temp;

}

}

else

{

break;

}

ptr = ptr->next;

}

s = s->next;

}

}

$ **g++** circular\_llist.cpp

$ a.out

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 4

List is empty, nothing to delete

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 5

List is empty, nothing to search

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

List is empty, nothing to display

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 7

List is empty, nothing to update

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 8

List is empty, nothing to **sort**

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 1

Enter the element: 100

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 2

Enter the element: 200

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

Circular Link List:

200-**>**100

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 3

Enter the element: 50

Insert element after position: 2

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

Circular Link List:

200-**>**100-**>**50

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 3

Enter the element: 150

Insert element after position: 3

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

Circular Link List:

200-**>**100-**>**50-**>**150

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 3

Enter the element: 1000

Insert element after position: 50

There are **less** than 50 **in** the list

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 4

Enter the element **for** deletion: 150

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

Circular Link List:

200-**>**100-**>**50

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 5

Enter the element to be searched: 100

Element 100 found at position 2

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 7

Enter the node position to be updated: 1

Enter the new value: 1010

Node Updated

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

Circular Link List:

1010-**>**100-**>**50

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 8

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 6

Circular Link List:

50-**>**100-**>**1010

---------------------------------

Operations on Circular singly linked list

---------------------------------

1.Create Node

2.Add at beginning

3.Add after

4.Delete

5.Search

6.Display

7.Update

8.Sort

9.Quit

Enter your choice : 9

------------------

**(**program exited with code: 1**)**

Press **return** to **continue**