C++ Program To Implement Singly Linked List

Next »

This C++ Program Implements Singly Linked List.

Here is source code of the C++ Program to Implement Singly Linked List. The C++ program is successfully compiled and run on a Linux system. The program output is also shown below.

```
1.  /*
2.   * C++ Program to Implement Singly Linked List
3.   */
4.  #include<iostream>
5.  #include<cstdio>
6.  #include<cstdlib>
7.  using namespace std;
8.  /*
9.   * Node Declaration
10.  */
11. struct node
12. {
13.     int info;
14.     struct node *next;
15. }*start;
16.
17. /*
18.  * Class Declaration
19.  */
20. class single_llist
```

```
21. {
22.     public:
23.         node* create_node(int);
24.         void insert_begin();
25.         void insert_pos();
26.         void insert_last();
27.         void delete_pos();
28.         void sort();
29.         void search();
30.         void update();
31.         void reverse();
32.         void display();
33.         single_llist()
34.         {
35.             start = NULL;
36.         }
37. };
38. 
39. /*
40.  * Main :contains menu
41.  */
42. main()
43. {
44.     int choice, nodes, element, position, i;
45.     single_llist sl;
46.     start = NULL;
47.     while (1)
48.     {
49.         cout<<endl<<"---------------------------------"<<endl;
50.         cout<<endl<<"Operations on singly linked list"<<endl;
51.         cout<<endl<<"---------------------------------"<<endl;
52.         cout<<"1.Insert Node at beginning"<<endl;
53.         cout<<"2.Insert node at last"<<endl;
54.         cout<<"3.Insert node at position"<<endl;
55.         cout<<"4.Sort Link List"<<endl;
56.         cout<<"5.Delete a Particular Node"<<endl;
57.         cout<<"6.Update Node Value"<<endl;
58.         cout<<"7.Search Element"<<endl;
59.         cout<<"8.Display Linked List"<<endl;
60.         cout<<"9.Reverse Linked List "<<endl;
61.         cout<<"10.Exit "<<endl;
62.         cout<<"Enter your choice : ";
63.         cin>>choice;
64.         switch(choice)
65.         {
66.         case 1:
67.             cout<<"Inserting Node at Beginning: "<<endl;
68.             sl.insert_begin();
69.             cout<<endl;
70.             break;
71.         case 2:
72.             cout<<"Inserting Node at Last: "<<endl;
```

```
            sl.insert_last();
            cout<<endl;
            break;
        case 3:
            cout<<"Inserting Node at a given position:"<<endl;
            sl.insert_pos();
            cout<<endl;
            break;
        case 4:
            cout<<"Sort Link List: "<<endl;
            sl.sort();
            cout<<endl;
            break;
        case 5:
            cout<<"Delete a particular node: "<<endl;
            sl.delete_pos();
            break;
        case 6:
            cout<<"Update Node Value:"<<endl;
            sl.update();
            cout<<endl;
            break;
        case 7:
            cout<<"Search element in Link List: "<<endl;
            sl.search();
            cout<<endl;
            break;
        case 8:
            cout<<"Display elements of link list"<<endl;
            sl.display();
            cout<<endl;
            break;
        case 9:
            cout<<"Reverse elements of Link List"<<endl;
            sl.reverse();
            cout<<endl;
            break;
        case 10:
            cout<<"Exiting..."<<endl;
            exit(1);
            break;
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
}

/*
 * Creating Node
 */
node *single_llist::create_node(int value)
{
```

```
125.     struct node *temp, *s;
126.     temp = new(struct node);
127.     if (temp == NULL)
128.     {
129.         cout<<"Memory not allocated "<<endl;
130.         return 0;
131.     }
132.     else
133.     {
134.         temp->info = value;
135.         temp->next = NULL;
136.         return temp;
137.     }
138. }
139. 
140. /*
141.  * Inserting element in beginning
142.  */
143. void single_llist::insert_begin()
144. {
145.     int value;
146.     cout<<"Enter the value to be inserted: ";
147.     cin>>value;
148.     struct node *temp, *p;
149.     temp = create_node(value);
150.     if (start == NULL)
151.     {
152.         start = temp;
153.         start->next = NULL;
154.     }
155.     else
156.     {
157.         p = start;
158.         start = temp;
159.         start->next = p;
160.     }
161.     cout<<"Element Inserted at beginning"<<endl;
162. }
163. 
164. /*
165.  * Inserting Node at last
166.  */
167. void single_llist::insert_last()
168. {
169.     int value;
170.     cout<<"Enter the value to be inserted: ";
171.     cin>>value;
172.     struct node *temp, *s;
173.     temp = create_node(value);
174.     s = start;
175.     while (s->next != NULL)
176.     {
```

```
177.         s = s->next;
178.     }
179.     temp->next = NULL;
180.     s->next = temp;
181.     cout<<"Element Inserted at last"<<endl;
182. }
183.
184. /*
185.  * Insertion of node at a given position
186.  */
187. void single_llist::insert_pos()
188. {
189.     int value, pos, counter = 0;
190.     cout<<"Enter the value to be inserted: ";
191.     cin>>value;
192.     struct node *temp, *s, *ptr;
193.     temp = create_node(value);
194.     cout<<"Enter the postion at which node to be inserted: ";
195.     cin>>pos;
196.     int i;
197.     s = start;
198.     while (s != NULL)
199.     {
200.         s = s->next;
201.         counter++;
202.     }
203.     if (pos == 1)
204.     {
205.         if (start == NULL)
206.         {
207.             start = temp;
208.             start->next = NULL;
209.         }
210.         else
211.         {
212.             ptr = start;
213.             start = temp;
214.             start->next = ptr;
215.         }
216.     }
217.     else if (pos > 1  && pos <= counter)
218.     {
219.         s = start;
220.         for (i = 1; i < pos; i++)
221.         {
222.             ptr = s;
223.             s = s->next;
224.         }
225.         ptr->next = temp;
226.         temp->next = s;
227.     }
228.     else
```

```
229.     {
230.         cout<<"Positon out of range"<<endl;
231.     }
232. }
233.
234. /*
235.  * Sorting Link List
236.  */
237. void single_llist::sort()
238. {
239.     struct node *ptr, *s;
240.     int value;
241.     if (start == NULL)
242.     {
243.         cout<<"The List is empty"<<endl;
244.         return;
245.     }
246.     ptr = start;
247.     while (ptr != NULL)
248.     {
249.         for (s = ptr->next;s !=NULL;s = s->next)
250.         {
251.             if (ptr->info > s->info)
252.             {
253.                 value = ptr->info;
254.                 ptr->info = s->info;
255.                 s->info = value;
256.             }
257.         }
258.         ptr = ptr->next;
259.     }
260. }
261.
262. /*
263.  * Delete element at a given position
264.  */
265. void single_llist::delete_pos()
266. {
267.     int pos, i, counter = 0;
268.     if (start == NULL)
269.     {
270.         cout<<"List is empty"<<endl;
271.         return;
272.     }
273.     cout<<"Enter the position of value to be deleted: ";
274.     cin>>pos;
275.     struct node *s, *ptr;
276.     s = start;
277.     if (pos == 1)
278.     {
279.         start = s->next;
280.     }
```

```
    else
    {
        while (s != NULL)
        {
            s = s->next;
            counter++;
        }
        if (pos > 0 && pos <= counter)
        {
            s = start;
            for (i = 1;i < pos;i++)
            {
                ptr = s;
                s = s->next;
            }
            ptr->next = s->next;
        }
        else
        {
            cout<<"Position out of range"<<endl;
        }
        free(s);
        cout<<"Element Deleted"<<endl;
    }
}

/*
 * Update a given Node
 */
void single_llist::update()
{
    int value, pos, i;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the node postion to be updated: ";
    cin>>pos;
    cout<<"Enter the new value: ";
    cin>>value;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
        start->info = value;
    }
    else
    {
        for (i = 0;i < pos - 1;i++)
        {
            if (s == NULL)
```

```
            {
                cout<<"There are less than "<<pos<<" elements";
                return;
            }
            s = s->next;
        }
        s->info = value;
    }
    cout<<"Node Updated"<<endl;
}

/*
 * Searching an element
 */
void single_llist::search()
{
    int value, pos = 0;
    bool flag = false;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the value to be searched: ";
    cin>>value;
    struct node *s;
    s = start;
    while (s != NULL)
    {
        pos++;
        if (s->info == value)
        {
            flag = true;
            cout<<"Element "<<value<<" is found at position "<<pos<<endl;
        }
        s = s->next;
    }
    if (!flag)
        cout<<"Element "<<value<<" not found in the list"<<endl;
}

/*
 * Reverse Link List
 */
void single_llist::reverse()
{
    struct node *ptr1, *ptr2, *ptr3;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
```

```
385.     if (start->next == NULL)
386.     {
387.         return;
388.     }
389.     ptr1 = start;
390.     ptr2 = ptr1->next;
391.     ptr3 = ptr2->next;
392.     ptr1->next = NULL;
393.     ptr2->next = ptr1;
394.     while (ptr3 != NULL)
395.     {
396.         ptr1 = ptr2;
397.         ptr2 = ptr3;
398.         ptr3 = ptr3->next;
399.         ptr2->next = ptr1;
400.     }
401.     start = ptr2;
402. }
403. 
404. /*
405.  * Display Elements of a link list
406.  */
407. void single_llist::display()
408. {
409.     struct node *temp;
410.     if (start == NULL)
411.     {
412.         cout<<"The List is Empty"<<endl;
413.         return;
414.     }
415.     temp = start;
416.     cout<<"Elements of list are: "<<endl;
417.     while (temp != NULL)
418.     {
419.         cout<<temp->info<<"->";
420.         temp = temp->next;
421.     }
422.     cout<<"NULL"<<endl;
423. }
```

```
$ g++ singlelinkedlist.cpp
$ a.out

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
The List is Empty.


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 5
Delete a particular node:
List is empty
```

```
---------------------------------

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 6
Update Node Value:
List is empty


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 7
Search element in Link List:
List is empty


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
```

```
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 3
Inserting Node at a given position:
Enter the value to be inserted: 1010
Enter the postion at which node to be inserted: 5
Positon out of range


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 1
Inserting Node at Beginning:
Enter the value to be inserted: 100
Element Inserted at beginning


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 1
Inserting Node at Beginning:
Enter the value to be inserted: 200
Element Inserted at beginning
```

```
---------------------------------

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->NULL

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 2
Inserting node at last:
Enter the value to be inserted: 50
Element Inserted at last

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
```

```
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 2
Inserting node at last:
Enter the value to be inserted: 150
Element Inserted at last


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->50->150->NULL

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 3
Inserting node at a given position:
Enter the value to be inserted: 1111
Enter the position at which node to be inserted: 4


---------------------------------
```

```
Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->50->1111->150->NULL


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 3
Inserting node at a given position:
Enter the value to be inserted: 1010
Enter the position at which node to be inserted: 100
Position out of range

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
```

```
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->50->1111->150->NULL

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 5
Delete a Particular node:
Enter the position of value to be deleted: 1


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
100->50->1111->150->NULL


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

Operations on singly linked list
```

```
---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 6
Update Node Value:
Enter the node position to be updated: 1
Enter the new value: 1010
Node Updated

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
1010->50->1111->150->NULL


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
```

```
9.Reverse Linked List
10.Exit
Enter your choice : 7
Search element in Link List:
Enter the value to be searched: 50
Element 50 is found at position 2


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 9
Reverse elements of Link List


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
150->1111->50->1010->NULL

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

Operations on singly linked list

---------------------------------
```

```
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 4
Sort Link List:

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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
50->150->1010->1111->NULL


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

Operations on singly linked list

---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 10
Exiting...
```

```
$
```

Sanfoundry Global Education & Learning Series – 1000 C++ Programs.

If you wish to look at all C++ Programming examples, go to C++ Programs.

Participate in the Sanfoundry Certification contest to get free Certificate of Merit. Join our social networks below and stay updated with latest contests, videos, internships and jobs!

Telegram | Youtube | LinkedIn | Instagram | Facebook | Twitter | Pinterest

» Next - C++ Program To Implement Circular Singly Linked List

Recommended Posts:

1. C# Programming Examples on Delegates
2. C# Programming Examples on Networking
3. C# Programming Examples
4. C Programming Examples on Combinatorial Problems & Algorithms
5. Java Algorithms, Problems & Programming Examples
6. C Programming Examples
7. Python Programming Examples on Graphs
8. Python Programming Examples on Trees
9. Java Programming Examples on Collection API
10. C Programming Examples using Recursion
11. C++ Programming Examples on STL
12. C Programming Examples on Stacks & Queues
13. C Programming Examples on Trees
14. C Programming Examples without using Recursion
15. Java Programming Examples on Data-Structures
16. C Programming Examples on Data-Structures
17. C++ Programming Examples on Data-Structures
18. C# Programming Examples on Data Structures
19. Python Programming Examples on Linked Lists
20. C Programming Examples on Linked List

Manish Bhojasia, a technology veteran with 20+ years @ Cisco & Wipro, is Founder and CTO at Sanfoundry. He is Linux Kernel Developer & SAN Architect and is passionate about competency developments in these areas. He lives in Bangalore and delivers focused training sessions to IT professionals in Linux Kernel, Linux Debugging, Linux Device Drivers, Linux Networking, Linux Storage, Advanced C Programming, SAN Storage Technologies, SCSI Internals & Storage Protocols such as iSCSI & Fiber Channel. Stay connected with him @ LinkedIn