القوائم الوصلية:

```
1    #include  <iostream>
2
3    using namespace std;
4
```

تعريف العقدة في القائمة الوصلية على شكل عقدة:

```
5    struct Node {
6    int data;
7    Node* next;
8    };
9
```

تابع تهيئة العقدة:

```
10   // only for the 1st Node
11   void  initNode(struct  Node  *head,int  n){
12   head->data  =  n;
13   head->next =NULL;
14   }
15
```

اضافة العقدة الى القائمة الوصلية:

```
16   // apending
17   void  addNode(struct  Node  *head,  int  n)  {
18   Node  *newNode  =  new  Node;
19   newNode->data = n;
20   newNode->next  =  NULL; 21
22   Node *cur  =  head;
23   while(cur)  {
24   if(cur->next ==  NULL)  {
25   cur->next  =  newNode;
26   return;
```

```
27    }
28    cur  =  cur->next; 29 }
30    }
31
```

حشر عقدة في بداية القائمة الوصلية:

```
32    void insertFront(struct Node **head, int n) {
33    Node  *newNode  =  new  Node;
34    newNode->data = n;
35    newNode->next  =  *head;
36    *head  =  newNode; 37 }
38
```

تابع البحث عن عقدة ويقوم بإعادة مؤشر إلى العقدة:

```
39    struct Node *searchNode(struct Node *head, int n) {
40    Node *cur  =  head;
41    while(cur)  {
42    if(cur->data  ==  n)  return  cur;
43    cur  =  cur->next;
44    }
45    cout << "No Node "  << n  << " in list.\n";
46    }
47
```

تابع حذف عقدة:

```
48    bool  deleteNode(struct  Node  **head,  Node  *ptrDel)  {
49    Node *cur  =  *head;
50    if(ptrDel ==  *head)  {
51    *head  =  cur->next;
52    delete ptrDel;
53    return true;
54    }
55
56    while(cur)  {
```

```
57    if(cur->next  ==  ptrDel)  {
58    cur->next   =   ptrDel->next;
59    delete ptrDel;
60    return true;
61    }
62    cur  =  cur->next; 63 }
64    return false;
65    }
66
```

تابع يقوم بعكس عقدة:

```
67    /* reverse the list */
68    struct Node* reverse(struct Node** head) 69 {
70    Node  *parent  =  *head;
71    Node  *me  =  parent->next;
72    Node  *child  =  me->next; 73
74    /* make parent as tail */
75    parent->next  =   NULL;
76    while(child)  {
77    me->next  =  parent;
78    parent  =  me;
79    me  =  child;
80    child   =   child->next; 81 }
82    me->next  =  parent;
83    *head =  me;
84    return *head;

85    }
86
```

تابع نسخ السلسلة الوصلية:

```
87    /* Creating a copy of a linked list */
88    void copyLinkedList(struct Node *node,  struct Node **pNew)
```

```
89    {
90    if(node != NULL)  {
91    *pNew  =  new  Node;
92    (*pNew)->data  =   node->data;
93    (*pNew)->next   =   NULL;
94    copyLinkedList(node->next,   &((*pNew)->next));
95    }
96    }
97
```

تابع مقارنة سلسلتين وصليتين:

```
98    /* Compare two linked list */
99    /* return value: same(1), different(0) */
100   int compareLinkedList(struct Node *node1, struct Node *node2)
101   {
102   static int  flag; 103
104   /* both lists are NULL  */
105   if(node1  ==  NULL  &&  node2  ==  NULL)   {
106   flag =  1;
107   }
108   else  {
109   if(node1 == NULL || node2 == NULL)
110   flag =  0;
111   else if(node1->data != node2->data)
112   flag =  0;
113   else
114   compareLinkedList(node1->next, node2->next);
115   }
116
117   return flag;
118   }
119
```

تابع حذف كامل السلسلة الوصلية:

```
120  void  deleteLinkedList(struct  Node  **node)
121  {
122  struct Node *tmpNode;
123  while(*node)  {
124  tmpNode  =  *node;
125  *node  =  tmpNode->next;
126  delete tmpNode;
127  }
128  }
129
```

تابع طباعة السلسلة الوصلية بدون تحريك المؤشر إلى رأس السلسلة:

```
130  void  display(struct  Node  *head)  {
131  Node  *list =  head;
132  while(list)   {
133  cout << list->data << " ";
134  list =  list->next;
135  }
136  cout <<  endl;
137  cout <<  endl;
138  }
139
```

التابع الرئيسي:

```
140  int main()
141  {
142  struct Node *newHead;
143  struct Node  *head  =  new  Node; 144
145  initNode(head,10);
146  display(head);
147
148  addNode(head,20);
```

```
149   display(head);
150
151   addNode(head,30);
152   display(head);
153
154   addNode(head,35);
155   display(head);
156
157   addNode(head,40);
158   display(head);
159
160   insertFront(&head,5);
161   display(head);
162
163   int numDel =  5;
164   Node  *ptrDelete  =  searchNode(head,numDel);
165   if(deleteNode(&head,ptrDelete))
166   cout << "Node "<< numDel << " deleted!\n";
167   display(head);
168
 169        cout << "The list is reversed\n";
170        reverse(&head);
171        display(head);
172
173        cout << "The list is copied\n";
174        copyLinkedList(head,&newHead);
175        display(newHead);
176
177        cout << "Comparing the two lists...\n";
178        cout  <<  "Are  the  two  lists same?\n";
179        if(compareLinkedList(head,newHead))
```

```
180         cout << "Yes, they are same!\n";
181         else
182         cout <<  "No, they are different!\n";
183         cout <<  endl;
184
185         numDel = 35;
186         ptrDelete  =  searchNode(newHead,numDel);
187         if(deleteNode(&newHead,ptrDelete))   {
188         cout << "Node "<< numDel << " deleted!\n";
189         cout << "The new list after the delete is\n";
190         display(newHead);
191         }
192         cout << "Comparing the two lists again...\n";
193         cout  <<  "Are  the  two  lists same?\n";
194         if(compareLinkedList(head,newHead))
195         cout << "Yes, they are same!\n";
196         else
197         cout <<  "No, they are different!\n";
198
199         cout <<  endl;
200         cout << "Deleting the copied list\n";
201         deleteLinkedList(&newHead);
202         display(newHead);
203         return 0;
204   }
205
```

جَـامعة
المَـنارة
MANARA UNIVERSITY