

```

1: /*
2:  * C++ Program to Implement Singly Linked List
3:  */
4: #include<iostream>
5: #include<cstdlib>
6: #include<stdlib.h>
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;
73:             sl.insert_last();
74:             cout<<endl;
75:             break;
76:         case 3:
77:             cout<<"Inserting Node at a given position: "<<endl;
78:             sl.insert_pos();
79:             cout<<endl;
80:             break;
81:         case 4:
82:             cout<<"Sort Link List: "<<endl;
83:             sl.sort();
84:             cout<<endl;
85:             break;
86:         case 5:
87:             cout<<"Delete a particular node: "<<endl;
88:             sl.delete_pos();
89:             break;
90:         case 6:
91:             cout<<"Update Node Value: "<<endl;
92:             sl.update();
93:             cout<<endl;
94:             break;
95:         case 7:
96:             cout<<"Search element in Link List: "<<endl;
97:             sl.search();
98:             cout<<endl;
99:             break;
100:        case 8:
101:            cout<<"Display elements of link list"<<endl;
102:            sl.display();
103:            cout<<endl;
104:            break;
105:        case 9:
```

```

106:         cout<<"Reverse elements of Link List"<<endl;
107:         sl.reverse();
108:         cout<<endl;
109:         break;
110:     case 10:
111:         cout<<"Exiting..."<<endl;
112:         exit(1);
113:         break;
114:     default:
115:         cout<<"Wrong choice"<<endl;
116:     }
117: }
118:}
119:
120:/*
121: * Creating Node
122: */
123:node *single_llist::create_node(int value)
124:{
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 position 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<<"Position 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->i n f o > s->i n f o)
252:             {
253:                 value = ptr->i n f o;
254:                 ptr->i n f o = s->i n f o;
255:                 s->i n f o = 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:     }

```



```

281:     else
282:     {
283:         while (s != NULL)
284:         {
285:             s = s->next;
286:             counter++;
287:         }
288:         if (pos > 0 && pos <= counter)
289:         {
290:             s = start;
291:             for (i = 1; i < pos; i++)
292:             {
293:                 ptr = s;
294:                 s = s->next;
295:             }
296:             ptr->next = s->next;
297:         }
298:         else
299:         {
300:             cout<<"Posi ti on out of range"<<endl;
301:         }
302:         free(s);
303:         cout<<"El ement Del et ed"<<endl;
304:     }
305: }
306:
307: /*
308:  * Update a given Node
309:  */
310: void single_llist::update()
311: {
312:     int value, pos, i;
313:     if (start == NULL)
314:     {
315:         cout<<"Li st is empty"<<endl;

```

```

316:         return;
317:     }
318:     cout<<"Enter the node position to be updated: ";
319:     cin>>pos;
320:     cout<<"Enter the new value: ";
321:     cin>>value;
322:     struct node *s, *ptr;
323:     s = start;
324:     if (pos == 1)
325:     {
326:         start->info = value;
327:     }
328:     else
329:     {
330:         for (i = 0; i < pos - 1; i++)
331:         {
332:             if (s == NULL)
333:             {
334:                 cout<<"There are less than "<<pos<<" elements";
335:                 return;
336:             }
337:             s = s->next;
338:         }
339:         s->info = value;
340:     }
341:     cout<<"Node Updated"<<endl;
342: }
343:
344: /*
345:  * Searching an element
346:  */
347: void single_llist::search()
348: {
349:     int value, pos = 0;
350:     bool flag = false;

```

```

351:     if (start == NULL)
352:     {
353:         cout<<"List is empty"<<endl;
354:         return;
355:     }
356:     cout<<"Enter the value to be searched: ";
357:     cin>>value;
358:     struct node *s;
359:     s = start;
360:     while (s != NULL)
361:     {
362:         pos++;
363:         if (s->info == value)
364:         {
365:             flag = true;
366:             cout<<"Element " <<value<<" is found at position " <<pos;
367:         }
368:         s = s->next;
369:     }
370:     if (!flag)
371:         cout<<"Element " <<value<<" not found in the list"<<endl;
372: }
373:
374: /*
375:  * Reverse Link List
376:  */
377: void single_llist::reverse()
378: {
379:     struct node *ptr1, *ptr2, *ptr3;
380:     if (start == NULL)
381:     {
382:         cout<<"List is empty"<<endl;
383:         return;
384:     }
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: }
```