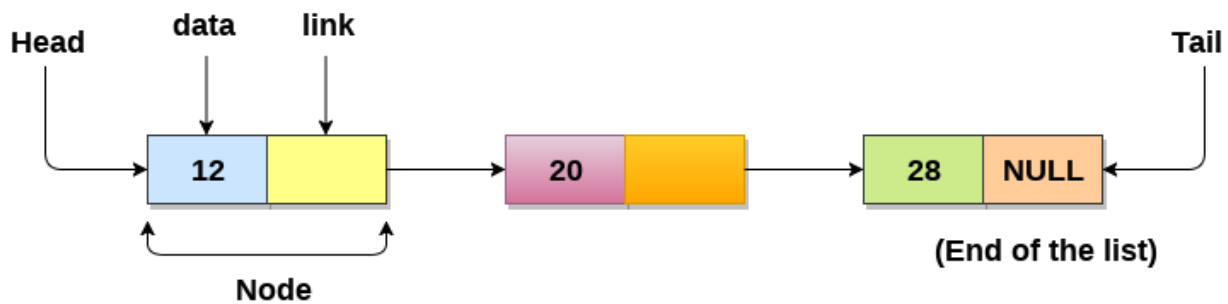


Linked List

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a list. This achieves optimized utilization of space.
- List size is limited to the memory size and doesn't need to be declared in advance.
- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Why use linked list over array?

Till now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known in order to decide the data structure which will be used throughout the program.

Array contains following limitations:

1. The size of array must be known in advance before using it in the program.

2. Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.

3. All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because,

1. It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.

2. Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

Singly linked list or One way chain

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist of two parts: data part and link part. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

One way chain or singly linked list can be traversed only in one direction. In other words, we can say that each node contains only next pointer, therefore we can not traverse the list in the reverse direction.

Consider an example where the marks obtained by the student in three subjects are stored in a linked list as shown in the figure.



In the above figure, the arrow represents the links. The data part of every node contains the marks obtained by the student in the different subject. The last node in the list is identified by the null pointer which is present in the address part of the last node. We can have as many elements we require, in the data part of the list.

Complexity

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Singly Linked List	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$

Operations on Singly Linked List

There are various operations which can be performed on singly linked list. A list of all such operations is given below.

Node Creation

```

1. struct node
2. {
3.     int data;
4.     struct node *next;
5. };
6. struct node *head, *ptr;
7. ptr = (struct node *)malloc(sizeof(struct node *));
  
```

Insertion

66 The insertion into a singly linked list can be performed at different positions. Based on
67 the position of the new node being inserted, the insertion is categorized into the
68 following categories.

69

SN	Operation	Description
1	<u>Insertion at beginning</u>	It involves inserting any element at the front of the list. We just need to a to make the new node as the head of the list.
2	<u>Insertion at end of the list</u>	It involves insertion at the last of the linked list. The new node can be node in the list or it can be inserted as the last one. Different logics are in scenario.
3	<u>Insertion after specified node</u>	It involves insertion after the specified node of the linked list. We need number of nodes in order to reach the node after which the new node will

70

71

72

73

74

75

76

77

78

79

80

81 **Insertion in singly linked list at beginning**

82 Inserting a new element into a singly linked list at beginning is quite simple. We just
 83 need to make a few adjustments in the node links. There are the following steps which
 84 need to be followed in order to insert a new node in the list at beginning.

85 o Allocate the space for the new node and store data into the data part of the
 86 node. This will be done by the following statements.

87 1. `ptr = (struct node *) malloc(sizeof(struct node *));`

88 2. `ptr → data = item`

89 o Make the link part of the new node pointing to the existing first node of the list.
 90 This will be done by using the following statement.

91 1. `ptr->next = head;`

92 o At the last, we need to make the new node as the first node of the list this will be
 93 done by using the following statement.

94 1. `head = ptr;`

95 Algorithm

96 o **Step 1:** IF PTR = NULL

97 Write OVERFLOW

98 Go to Step 7

99 [END OF IF]

100 o **Step 2:** SET NEW_NODE = PTR

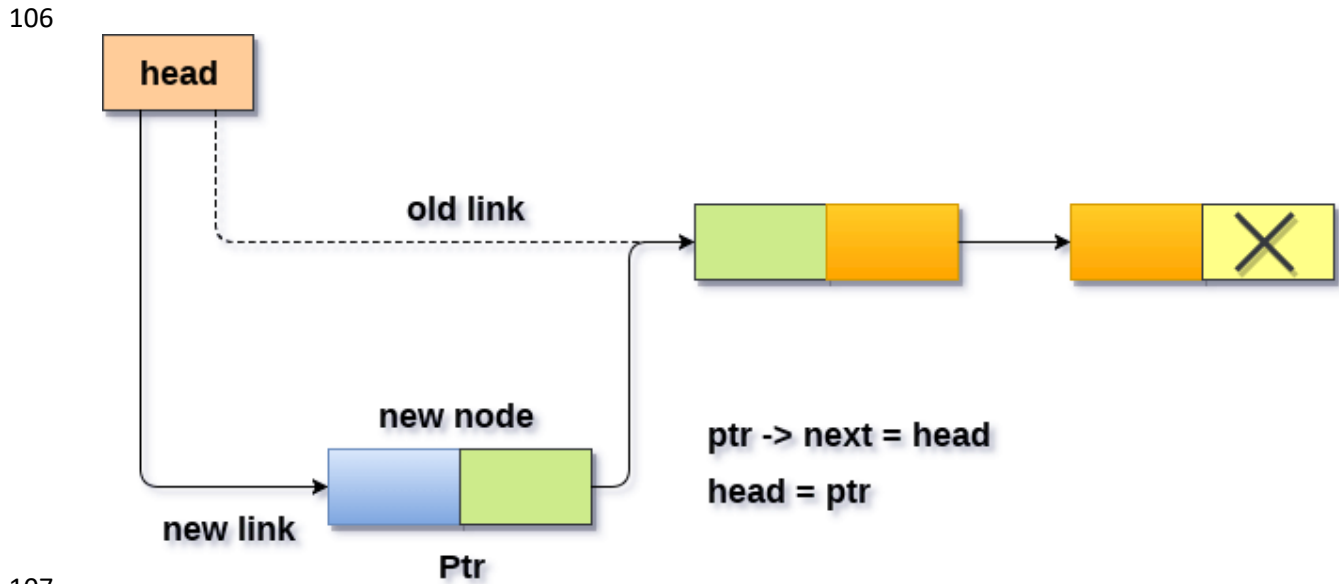
101 o **Step 3:** SET PTR = PTR → NEXT

102 o **Step 4:** SET NEW_NODE → DATA = VAL

103 o **Step 5:** SET NEW_NODE → NEXT = HEAD

104 o **Step 6:** SET HEAD = NEW_NODE

105 o **Step 7:** EXIT



108 C Function

```

109 #include<stdio.h>
110 #include<stdlib.h>
111 void begininsert(int);
112 struct node
113 {
114     int data;
115     struct node *next;
116 };
117 struct node *head;
118 void main ()
119 {
120     int choice,item;
121     do
122     {
123         printf("\nEnter the item which you want to insert?\n");
124         scanf("%d",&item);
    
```

```
125     begininsert(item);
126     printf("\nPress 0 to insert more ?\n");
127     scanf("%d",&choice);
128 }while(choice == 0);
129 }
130 void begininsert(int item)
131 {
132     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
133     if(ptr == NULL)
134     {
135         printf("\nOVERFLOW\n");
136     }
137     else
138     {
139         ptr->data = item;
140         ptr->next = head;
141         head = ptr;
142         printf("\nNode inserted\n");
143     }
144
145 }
146
```

Insertion in singly linked list at the end

In order to insert a node at the last, there are two following scenarios which need to be mentioned.

1. The node is being added to an empty list
2. The node is being added to the end of the linked list

in the first case,

- The condition (`head == NULL`) gets satisfied. Hence, we just need to allocate the space for the node by using `malloc` statement in C. Data and the link part of the node are set up by using the following statements.

1. `ptr->data = item;`
2. `ptr -> next = NULL;`

- Since, **ptr** is the only node that will be inserted in the list hence, we need to make this node pointed by the head pointer of the list. This will be done by using the following Statements.

1. `Head = ptr`

In the second case,

- The condition **Head = NULL** would fail, since Head is not null. Now, we need to declare a temporary pointer `temp` in order to traverse through the list. **temp** is made to point the first node of the list.

1. `Temp = head`

- Then, traverse through the entire linked list using the statements:

1. **while** (`temp→ next != NULL`)
2. `temp = temp → next;`

- At the end of the loop, the `temp` will be pointing to the last node of the list. Now, allocate the space for the new node, and assign the item to its data part. Since,

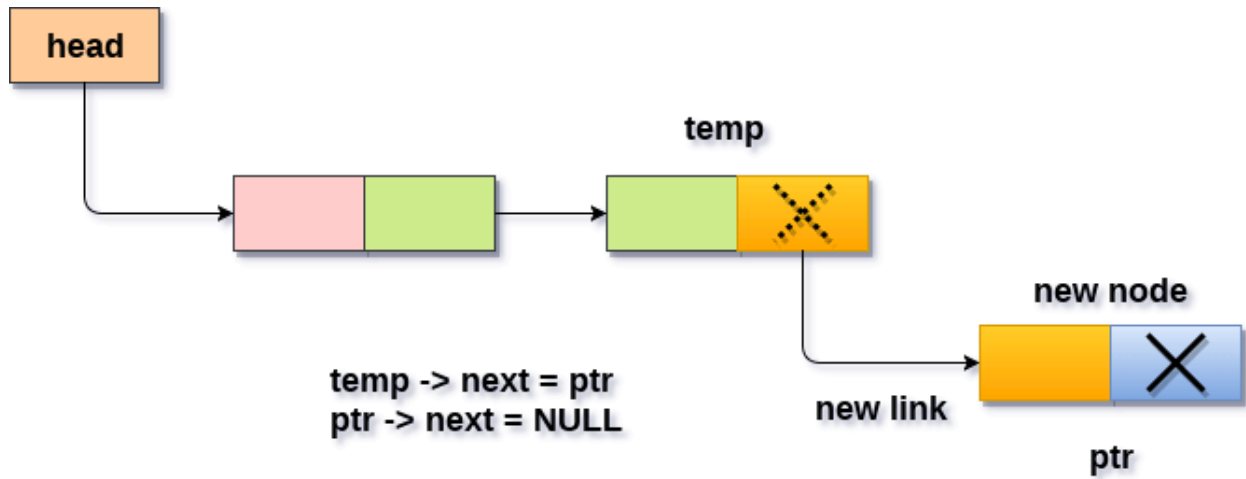
172 the new node is going to be the last node of the list hence, the next part of this
 173 node needs to be pointing to the **null**. We need to make the next part of the
 174 temp node (which is currently the last node of the list) point to the new node
 175 (ptr) .

```
176 1. temp = head;
177 2.   while (temp -> next != NULL)
178 3.   {
179 4.       temp = temp -> next;
180 5.   }
181 6.   temp->next = ptr;
182 7.   ptr->next = NULL;
```

183 Algorithm

```
184 ○ Step 1: IF PTR = NULL Write OVERFLOW
185                Go to Step 1
186 [END OF IF]
187 ○ Step 2: SET NEW_NODE = PTR
188 ○ Step 3: SET PTR = PTR - > NEXT
189 ○ Step 4: SET NEW_NODE - > DATA = VAL
190 ○ Step 5: SET NEW_NODE - > NEXT = NULL
191 ○ Step 6: SET PTR = HEAD
192 ○ Step 7: Repeat Step 8 while PTR - > NEXT != NULL
193 ○ Step 8: SET PTR = PTR - > NEXT
194 [END OF LOOP]
195 ○ Step 9: SET PTR - > NEXT = NEW_NODE
196 ○ Step 10: EXIT
```

197



Inserting node at the last into a non-empty list

198

C Function

199

200 #include<stdio.h>

201 #include<stdlib.h>

202 void lastinsert(int);

203 struct node

204 {

205 int data;

206 struct node *next;

207 };

208 struct node *head;

209 void main ()

210 {

211 int choice,item;

212 do

213 {

214 printf("\nEnter the item which you want to insert?\n");

```
215     scanf("%d",&item);
216     lastinsert(item);
217     printf("\nPress 0 to insert more ?\n");
218     scanf("%d",&choice);
219 }while(choice == 0);
220 }
221 void lastinsert(int item)
222 {
223     struct node *ptr = (struct node*)malloc(sizeof(struct node));
224     struct node *temp;
225     if(ptr == NULL)
226     {
227         printf("\nOVERFLOW");
228     }
229     else
230     {
231         ptr->data = item;
232         if(head == NULL)
233         {
234             ptr -> next = NULL;
235             head = ptr;
236             printf("\nNode inserted");
237         }
238         else
239         {
240             temp = head;
241             while (temp -> next != NULL)
242             {
243                 temp = temp -> next;
```

```
244         }
245         temp->next = ptr;
246         ptr->next = NULL;
247         printf("\nNode inserted");
248
249     }
250 }
251 }
```

252 Output

```
253
254 Enter the item which you want to insert?
255 12
256
257 Node inserted
258 Press 0 to insert more ?
259 0
260
261 Enter the item which you want to insert?
262 23
263
264 Node inserted
265 Press 0 to insert more ?
266 2
267
```

Insertion in singly linked list after specified Node

- In order to insert an element after the specified number of nodes into the linked list, we need to skip the desired number of elements in the list to move the pointer at the position after which the node will be inserted. This will be done by using the following statements.

```
1. emp=head;
2.   for(i=0; i<loc; i++)
3.   {
4.       temp = temp->next;
5.       if(temp == NULL)
6.       {
7.           return;
8.       }
9.
10. }
```

- Allocate the space for the new node and add the item to the data part of it. This will be done by using the following statements.

```
1. ptr = (struct node *) malloc (sizeof(struct node));
2.   ptr->data = item;
```

- Now, we just need to make a few more link adjustments and our node at will be inserted at the specified position. Since, at the end of the loop, the loop pointer temp would be pointing to the node after which the new node will be inserted. Therefore, the next part of the new node ptr must contain the address of the next part of the temp (since, ptr will be in between temp and the next of the temp). This will be done by using the following statements.

```
1. ptr->next = temp->next
```

now, we just need to make the next part of the temp, point to the new node ptr. This will insert the new node ptr, at the specified position.

297 1. temp ->next = ptr;

298 Algorithm

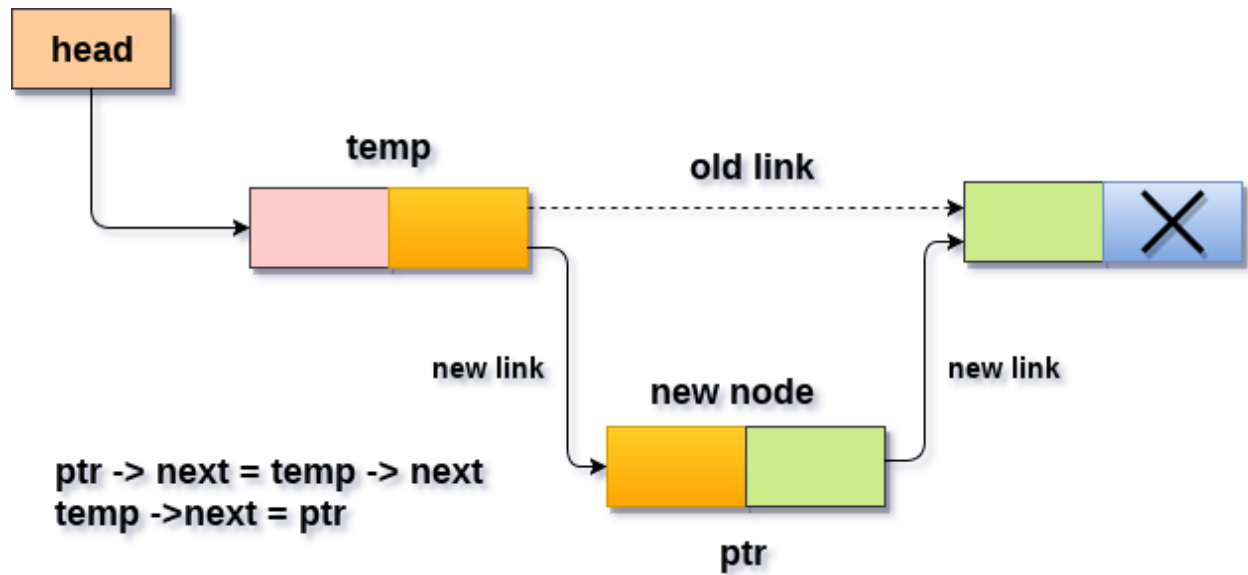
```

299   ○ STEP 1: IF PTR = NULL
300       WRITE OVERFLOW
301       GOTO STEP 12
302   END OF IF

303   ○ STEP 2: SET NEW_NODE = PTR
304   ○ STEP 3: NEW_NODE → DATA = VAL
305   ○ STEP 4: SET TEMP = HEAD
306   ○ STEP 5: SET I = 0
307   ○ STEP 6: REPEAT STEP 5 AND 6 UNTIL I
308   ○ STEP 7: TEMP = TEMP → NEXT
309   ○ STEP 8: IF TEMP = NULL

310       WRITE "DESIRED NODE NOT PRESENT"
311       GOTO STEP 12
312   END OF IF
313   END OF LOOP

314   ○ STEP 9: PTR → NEXT = TEMP → NEXT
315   ○ STEP 10: TEMP → NEXT = PTR
316   ○ STEP 11: SET PTR = NEW_NODE
317   ○ STEP 12: EXIT
    
```



318

319 C Function

```
320 #include<stdio.h>
321 #include<stdlib.h>
322 void randominsert(int);
323 void create(int);
324 struct node
325 {
326     int data;
327     struct node *next;
328 };
329 struct node *head;
330 void main ()
331 {
332     int choice,item,loc;
333     do
334     {
335         printf("\nEnter the item which you want to insert?\n");
336         scanf("%d",&item);
```

```
337     if(head == NULL)
338     {
339         create(item);
340     }
341     else
342     {
343         randominsert(item);
344     }
345     printf("\nPress 0 to insert more ?\n");
346     scanf("%d",&choice);
347 }while(choice == 0);
348 }
349 void create(int item)
350 {
351
352     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
353     if(ptr == NULL)
354     {
355         printf("\nOVERFLOW\n");
356     }
357     else
358     {
359         ptr->data = item;
360         ptr->next = head;
361         head = ptr;
362         printf("\nNode inserted\n");
363     }
364 }
365 void randominsert(int item)
```



```
366     {
367         struct node *ptr = (struct node *) malloc (sizeof(struct node));
368         struct node *temp;
369         int i,loc;
370         if(ptr == NULL)
371         {
372             printf("\nOVERFLOW");
373         }
374         else
375         {
376
377             printf("Enter the location");
378             scanf("%d",&loc);
379             ptr->data = item;
380             temp=head;
381             for(i=0;i<loc;i++)
382             {
383                 temp = temp->next;
384                 if(temp == NULL)
385                 {
386                     printf("\ncan't insert\n");
387                     return;
388                 }
389
390             }
391             ptr ->next = temp ->next;
392             temp ->next = ptr;
393             printf("\nNode inserted");
394         }
```

395

396 }

397 Output

398

399 Enter the item which you want to insert?

400 12

401

402 Node inserted

403

404 Press 0 to insert more ?

405 2

406 Deletion and Traversing

407 The Deletion of a node from a singly linked list can be performed at different positions.
 408 Based on the position of the node being deleted, the operation is categorized into the
 409 following categories.

SN	Operation	Description
1	Deletion at beginning	It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers.
2	Deletion at the end of the list	It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios.
3	Deletion after specified node	It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list.
4	Traversing	In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.
5	Searching	In searching, we match each element of the list with the given element. If the element is found on any of the location then location of that element is returned otherwise null is returned. .

410

411

412

413

414

415

Deletion in singly linked list at beginning

Deleting a node from the beginning of the list is the simplest operation of all. It just need a few adjustments in the node pointers. Since the first node of the list is to be deleted, therefore, we just need to make the head, point to the next of the head. This will be done by using the following statements.

1. ptr = head;
2. head = ptr->next;

Now, free the pointer ptr which was pointing to the head node of the list. This will be done by using the following statement.

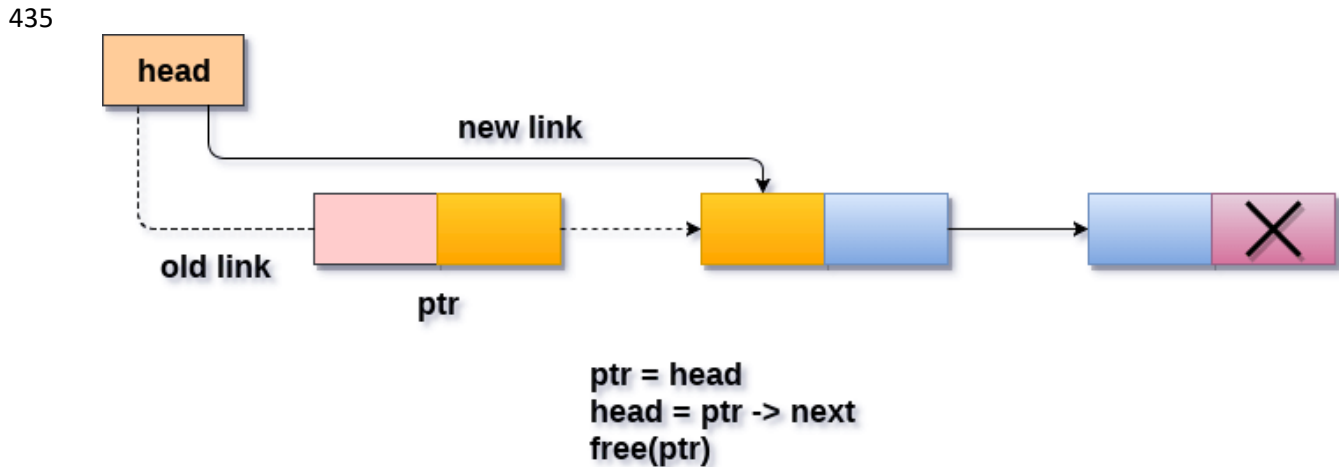
1. free(ptr)

Algorithm

- **Step 1:** IF HEAD = NULL

Write				UNDERFLOW
Go	to	Step	5	
[END OF IF]				

- **Step 2:** SET PTR = HEAD
- **Step 3:** SET HEAD = HEAD -> NEXT
- **Step 4:** FREE PTR
- **Step 5:** EXIT



Deleting a node from the beginning

C function

```
438 #include<stdio.h>
439 #include<stdlib.h>
440 void create(int);
441 void begdelete();
442 struct node
443 {
444     int data;
445     struct node *next;
446 };
447 struct node *head;
448 void main ()
449 {
450     int choice,item;
451     do
452     {
453         printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
```

```
454     scanf("%d",&choice);
455     switch(choice)
456     {
457         case 1:
458             printf("\nEnter the item\n");
459             scanf("%d",&item);
460             create(item);
461             break;
462         case 2:
463             begdelete();
464             break;
465         case 3:
466             exit(0);
467             break;
468         default:
469             printf("\nPlease enter valid choice\n");
470     }
471
472     }while(choice != 3);
473 }
474 void create(int item)
475 {
476     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
477     if(ptr == NULL)
478     {
479         printf("\nOVERFLOW\n");
480     }
481     else
482     {
```

```
483     ptr->data = item;
484     ptr->next = head;
485     head = ptr;
486     printf("\nNode inserted\n");
487 }
488
489 }
490 void begdelete()
491 {
492     struct node *ptr;
493     if(head == NULL)
494     {
495         printf("\nList is empty");
496     }
497     else
498     {
499         ptr = head;
500         head = ptr->next;
501         free(ptr);
502         printf("\n Node deleted from the begining ...");
503     }
504 }
505 Output
506
507 1.Append List
508 2.Delete node
509 3.Exit
510 4.Enter your choice?1
511
```

512 Enter the item

513 23

514

515 Node inserted

516

517 1.Append List

518 2.Delete node

519 3.Exit

520 4.Enter your choice?2

521

522 Node deleted from the begining ...

523

Deletion in singly linked list at the end

There are two scenarios in which, a node is deleted from the end of the linked list.

1. There is only one node in the list and that needs to be deleted.
2. There are more than one node in the list and the last node of the list will be deleted.

In the first scenario,

the condition `head → next = NULL` will survive and therefore, the only node head of the list will be assigned to null. This will be done by using the following statements.

1. `ptr = head`
2. `head = NULL`
3. `free(ptr)`

In the second scenario,

The condition `head → next = NULL` would fail and therefore, we have to traverse the node in order to reach the last node of the list.

For this purpose, just declare a temporary pointer `temp` and assign it to head of the list. We also need to keep track of the second last node of the list. For this purpose, two pointers `ptr` and `ptr1` will be used where `ptr` will point to the last node and `ptr1` will point to the second last node of the list.

this all will be done by using the following statements.

1. `ptr = head;`
2. `while(ptr->next != NULL)`
3. `{`
4. `ptr1 = ptr;`
5. `ptr = ptr ->next;`
6. `}`

549 Now, we just need to make the pointer ptr1 point to the NULL and the last node of the
550 list that is pointed by ptr will become free. It will be done by using the following
551 statements.

- 552 1. ptr1->next = NULL;
- 553 2. free(ptr);

554 Algorithm

555 ○ **Step 1:** IF HEAD = NULL

556	Write		UNDERFLOW
557	Go	to	Step 8
558	[END OF IF]		

559 ○ **Step 2:** SET PTR = HEAD

560 ○ **Step 3:** Repeat Steps 4 and 5 while PTR -> NEXT!= NULL

561 ○ **Step 4:** SET PREPTR = PTR

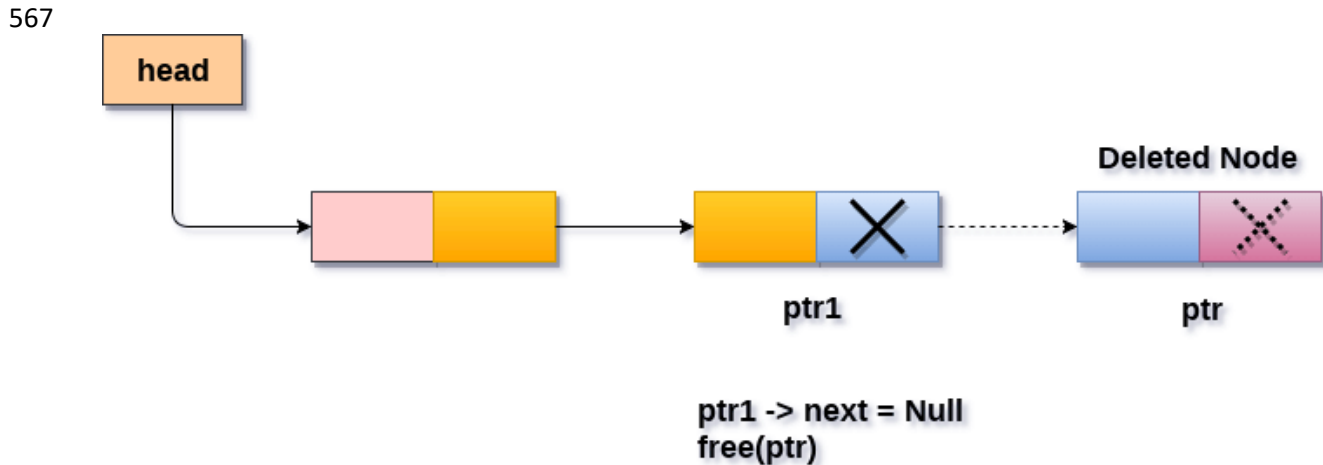
562 ○ **Step 5:** SET PTR = PTR -> NEXT

563 [END OF LOOP]

564 ○ **Step 6:** SET PREPTR -> NEXT = NULL

565 ○ **Step 7:** FREE PTR

566 ○ **Step 8:** EXIT



Deleting a node from the last

568

569 C Function :

```

570 #include<stdio.h>
571 #include<stdlib.h>
572 void create(int);
573 void end_delete();
574 struct node
575 {
576     int data;
577     struct node *next;
578 };
579 struct node *head;
580 void main ()
581 {
582     int choice,item;
583     do
584     {
585         printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
    
```

```
586     scanf("%d",&choice);
587     switch(choice)
588     {
589         case 1:
590             printf("\nEnter the item\n");
591             scanf("%d",&item);
592             create(item);
593             break;
594         case 2:
595             end_delete();
596             break;
597         case 3:
598             exit(0);
599             break;
600         default:
601             printf("\nPlease enter valid choice\n");
602     }
603
604 }while(choice != 3);
605 }
606 void create(int item)
607 {
608     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
609     if(ptr == NULL)
610     {
611         printf("\nOVERFLOW\n");
612     }
613     else
614     {
```

```
615         ptr->data = item;
616         ptr->next = head;
617         head = ptr;
618         printf("\nNode inserted\n");
619     }
620
621 }
622 void end_delete()
623 {
624     struct node *ptr,*ptr1;
625     if(head == NULL)
626     {
627         printf("\nlist is empty");
628     }
629     else if(head -> next == NULL)
630     {
631         head = NULL;
632         free(head);
633         printf("\nOnly node of the list deleted ...");
634     }
635
636     else
637     {
638         ptr = head;
639         while(ptr->next != NULL)
640         {
641             ptr1 = ptr;
642             ptr = ptr ->next;
643         }
```

```
644         ptr1->next = NULL;
645         free(ptr);
646         printf("\n Deleted Node from the last ...");
647     }
648 }
```

649 Output

650

651 1.Append List

652 2.Delete node

653 3.Exit

654 4.Enter your choice?1

655

656 Enter the item

657 12

658

659 Node inserted

660

661 1.Append List

662 2.Delete node

663 3.Exit

664 4.Enter your choice?2

665

666 Only node of the list deleted ...

667

Deletion in singly linked list after the specified node :

In order to delete the node, which is present after the specified node, we need to skip the desired number of nodes to reach the node after which the node will be deleted. We need to keep track of the two nodes. The one which is to be deleted the other one if the node which is present before that node. For this purpose, two pointers are used: ptr and ptr1.

Use the following statements to do so.

```
1. ptr=head;
2.   for(i=0;i<loc;i++)
3.   {
4.       ptr1 = ptr;
5.       ptr = ptr->next;
6.
7.       if(ptr == NULL)
8.       {
9.           printf("\nThere are less than %d elements in the list..",loc);
10.          return;
11.      }
12.  }
```

Now, our task is almost done, we just need to make a few pointer adjustments. Make the next of ptr1 (points to the specified node) point to the next of ptr (the node which is to be deleted).

This will be done by using the following statements.

```
1. ptr1 ->next = ptr ->next;
2.   free(ptr);
```

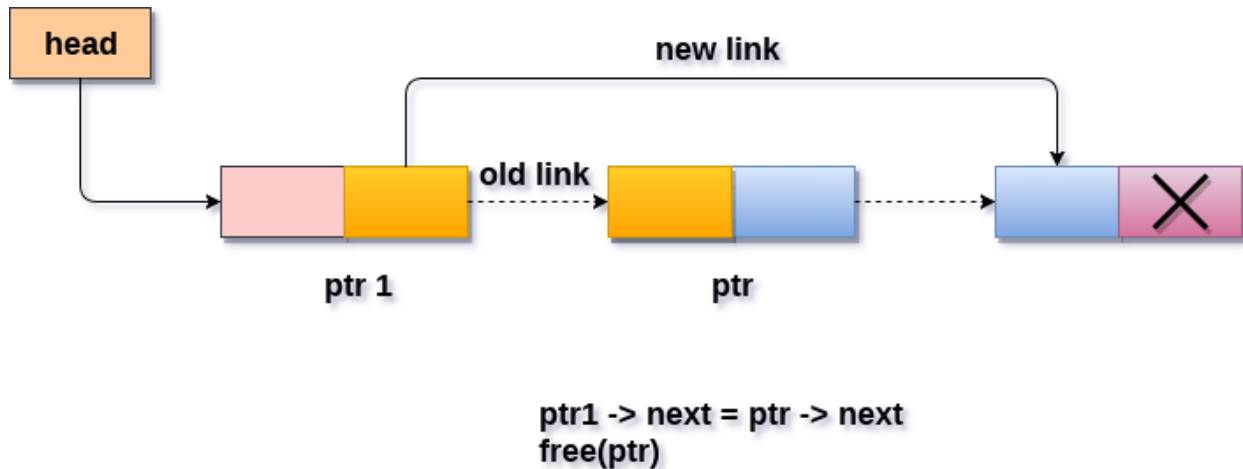
Algorithm

- **STEP 1:** IF HEAD = NULL

```

697      WRITE                                UNDERFLOW
698                                GOTO                                STEP                                10
699      END OF IF
700      ○ STEP 2: SET TEMP = HEAD
701      ○ STEP 3: SET I = 0
702      ○ STEP 4: REPEAT STEP 5 TO 8 UNTIL I
703      ○ STEP 5: TEMP1 = TEMP
704      ○ STEP 6: TEMP = TEMP → NEXT
705      ○ STEP 7: IF TEMP = NULL
706      WRITE                                "DESIRED                                NODE                                NOT                                PRESENT"
707                                GOTO                                STEP                                12
708      END OF IF
709      ○ STEP 8: I = I+1
710      END OF LOOP
711      ○ STEP 9: TEMP1 → NEXT = TEMP → NEXT
712      ○ STEP 10: FREE TEMP
713      ○ STEP 11: EXIT
714

```



Deletion a node from specified position

715

716 C function

```

717 #include<stdio.h>
718 #include<stdlib.h>
719 void create(int);
720 void delete_specified();

```



```
721  struct node
722  {
723      int data;
724      struct node *next;
725  };
726  struct node *head;
727  void main ()
728  {
729      int choice,item;
730      do
731      {
732          printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
733          scanf("%d",&choice);
734          switch(choice)
735          {
736              case 1:
737                  printf("\nEnter the item\n");
738                  scanf("%d",&item);
739                  create(item);
740                  break;
741              case 2:
742                  delete_specified();
743                  break;
744              case 3:
745                  exit(0);
746                  break;
747              default:
748                  printf("\nPlease enter valid choice\n");
749          }
```

```
750
751     }while(choice != 3);
752 }
753 void create(int item)
754 {
755     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
756     if(ptr == NULL)
757     {
758         printf("\nOVERFLOW\n");
759     }
760     else
761     {
762         ptr->data = item;
763         ptr->next = head;
764         head = ptr;
765         printf("\nNode inserted\n");
766     }
767
768 }
769 void delete_specified()
770 {
771     struct node *ptr, *ptr1;
772     int loc,i;
773     scanf("%d",&loc);
774     ptr=head;
775     for(i=0;i<loc;i++)
776     {
777         ptr1 = ptr;
778         ptr = ptr->next;
```

```
779
780     if(ptr == NULL)
781     {
782         printf("\nThere are less than %d elements in the list..\n",loc);
783         return;
784     }
785 }
786 ptr1 ->next = ptr ->next;
787 free(ptr);
788 printf("\nDeleted %d node ",loc);
789 }
```

790 Output

791

792 1.Append List

793 2.Delete node

794 3.Exit

795 4.Enter your choice?1

796

797 Enter the item

798 12

799

800 Node inserted

801

802 1.Append List

803 2.Delete node

804 3.Exit

805 4.Enter your choice?1

806

807 Enter the item

808 23

809

810 Node inserted

811

812 1.Append List

813 2.Delete node

814 3.Exit

815 4.Enter your choice?2

816 12

817

818 There are less than 12 elements in the list..

819

820 1.Append List

821 2.Delete node

822 3.Exit

823 4.Enter your choice?2

824 1

825

826 Deleted 1 node

827

828 Traversing in singly linked list

829 Traversing is the most common operation that is performed in almost every scenario of
830 singly linked list. Traversing means visiting each node of the list once in order to
831 perform some operation on that. This will be done by using the following statements.

```
832     1. ptr = head;
833     2.     while (ptr!=NULL)
834     3.         {
835     4.             ptr = ptr -> next;
836     5.         }
```

837 Algorithm

838 ○ **STEP 1:** SET PTR = HEAD

839 ○ **STEP 2:** IF PTR = NULL

840	WRITE	"EMPTY	LIST"
841	GOTO	STEP	7
842	END OF IF		

843 ○ **STEP 4:** REPEAT STEP 5 AND 6 UNTIL PTR != NULL

844 ○ **STEP 5:** PRINT PTR → DATA

845 ○ **STEP 6:** PTR = PTR → NEXT

846 [END OF LOOP]

847 ○ **STEP 7:** EXIT

848 C function

849 #include<stdio.h>

850 #include<stdlib.h>

851 void create(int);

852 void traverse();

853 struct node

```
854  {
855      int data;
856      struct node *next;
857  };
858  struct node *head;
859  void main ()
860  {
861      int choice,item;
862      do
863      {
864          printf("\n1.Append List\n2.Traverse\n3.Exit\n4.Enter your choice?");
865          scanf("%d",&choice);
866          switch(choice)
867          {
868              case 1:
869                  printf("\nEnter the item\n");
870                  scanf("%d",&item);
871                  create(item);
872                  break;
873              case 2:
874                  traverse();
875                  break;
876              case 3:
877                  exit(0);
878                  break;
879              default:
880                  printf("\nPlease enter valid choice\n");
881          }
882
```

```
883     }while(choice != 3);
884 }
885 void create(int item)
886 {
887     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
888     if(ptr == NULL)
889     {
890         printf("\nOVERFLOW\n");
891     }
892     else
893     {
894         ptr->data = item;
895         ptr->next = head;
896         head = ptr;
897         printf("\nNode inserted\n");
898     }
899
900 }
901 void traverse()
902 {
903     struct node *ptr;
904     ptr = head;
905     if(ptr == NULL)
906     {
907         printf("Empty list..");
908     }
909     else
910     {
911         printf("printing values . . . . \n");
```

```
912         while (ptr!=NULL)
913         {
914             printf("\n%d",ptr->data);
915             ptr = ptr -> next;
916         }
917     }
918 }
```

919 Output

920

921 1.Append List

922 2.Traverse

923 3.Exit

924 4.Enter your choice?1

925

926 Enter the item

927 23

928

929 Node inserted

930

931 1.Append List

932 2.Traverse

933 3.Exit

934 4.Enter your choice?1

935

936 Enter the item

937 233

938

939 Node inserted

940

941 1.Append List
942 2.Traverse
943 3.Exit
944 4.Enter your choice?2
945 printing values
946
947 233
948 23
949

Searching in singly linked list

Searching is performed in order to find the location of a particular element in the list. Searching any element in the list needs traversing through the list and make the comparison of every element of the list with the specified element. If the element is matched with any of the list element then the location of the element is returned from the function.

Algorithm

- **Step 1:** SET PTR = HEAD
- **Step 2:** Set I = 0
- **STEP 3:** IF PTR = NULL
WRITE "EMPTY LIST"
GOTO STEP 8
END OF IF
- **STEP 4:** REPEAT STEP 5 TO 7 UNTIL PTR != NULL
- **STEP 5:** if ptr → data = item
write i+1
End of IF
- **STEP 6:** I = I + 1
- **STEP 7:** PTR = PTR → NEXT
[END OF LOOP]
- **STEP 8:** EXIT

C function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
```

```
975 void search();
976 struct node
977 {
978     int data;
979     struct node *next;
980 };
981 struct node *head;
982 void main ()
983 {
984     int choice,item,loc;
985     do
986     {
987         printf("\n1.Create\n2.Search\n3.Exit\n4.Enter your choice?");
988         scanf("%d",&choice);
989         switch(choice)
990         {
991             case 1:
992                 printf("\nEnter the item\n");
993                 scanf("%d",&item);
994                 create(item);
995                 break;
996             case 2:
997                 search();
998             case 3:
999                 exit(0);
1000             break;
1001             default:
1002                 printf("\nPlease enter valid choice\n");
1003         }
```

```
1004
1005     }while(choice != 3);
1006 }
1007 void create(int item)
1008 {
1009     struct node *ptr = (struct node *)malloc(sizeof(struct node *));
1010     if(ptr == NULL)
1011     {
1012         printf("\nOVERFLOW\n");
1013     }
1014     else
1015     {
1016         ptr->data = item;
1017         ptr->next = head;
1018         head = ptr;
1019         printf("\nNode inserted\n");
1020     }
1021
1022 }
1023 void search()
1024 {
1025     struct node *ptr;
1026     int item,i=0,flag;
1027     ptr = head;
1028     if(ptr == NULL)
1029     {
1030         printf("\nEmpty List\n");
1031     }
1032     else
```

```
1033     {
1034         printf("\nEnter item which you want to search?\n");
1035         scanf("%d",&item);
1036         while (ptr!=NULL)
1037         {
1038             if(ptr->data == item)
1039             {
1040                 printf("item found at location %d ",i+1);
1041                 flag=0;
1042             }
1043             else
1044             {
1045                 flag=1;
1046             }
1047             i++;
1048             ptr = ptr -> next;
1049         }
1050         if(flag==1)
1051         {
1052             printf("Item not found\n");
1053         }
1054     }
1055
1056 }
1057 Output
1058
1059 1.Create
1060 2.Search
1061 3.Exit
```

1062 4.Enter your choice?1

1063

1064 Enter the item

1065 23

1066

1067 Node inserted

1068

1069 1.Create

1070 2.Search

1071 3.Exit

1072 4.Enter your choice?1

1073

1074 Enter the item

1075 34

1076

1077 Node inserted

1078

1079 1.Create

1080 2.Search

1081 3.Exit

1082 4.Enter your choice?2

1083

1084 Enter item which you want to search?

1085 34

1086 item found at location 1

1087

1089 **Linked List in C: Menu Driven Program all in one**

```

1090 #include<stdio.h>
1091 #include<stdlib.h>
1092 struct node
1093 {
1094     int data;
1095     struct node *next;
1096 };
1097 struct node *head;
1098
1099 void beginsert ();
1100 void lastinsert ();
1101 void randominsert();
1102 void begin_delete();
1103 void last_delete();
1104 void random_delete();
1105 void display();
1106 void search();
1107 void main ()
1108 {
1109     int choice =0;
1110     while(choice != 9)
1111     {
1112         printf("\n\n*****Main Menu*****\n");
1113         printf("\nChoose one option from the following list ...\n");
1114         printf("\n===== \n");
1115         printf("\n1.Insert in beginning\n2.Insert at last\n3.Insert at any random location\n4.Delete from
1116 Beginning\n

```



```
1117      5.Delete from last\n6.Delete node after specified location\n7.Search for an
1118 element\n8.Show\n9.Exit\n");
1119      printf("\nEnter your choice?\n");
1120      scanf("\n%d",&choice);
1121      switch(choice)
1122      {
1123          case 1:
1124              begininsert();
1125              break;
1126          case 2:
1127              lastinsert();
1128              break;
1129          case 3:
1130              randominsert();
1131              break;
1132          case 4:
1133              begin_delete();
1134              break;
1135          case 5:
1136              last_delete();
1137              break;
1138          case 6:
1139              random_delete();
1140              break;
1141          case 7:
1142              search();
1143              break;
1144          case 8:
1145              display();
```

```
1146         break;
1147         case 9:
1148             exit(0);
1149             break;
1150             default:
1151                 printf("Please enter valid choice..");
1152             }
1153     }
1154 }
1155 void beginsert()
1156 {
1157     struct node *ptr;
1158     int item;
1159     ptr = (struct node *) malloc(sizeof(struct node *));
1160     if(ptr == NULL)
1161     {
1162         printf("\nOVERFLOW");
1163     }
1164     else
1165     {
1166         printf("\nEnter value\n");
1167         scanf("%d",&item);
1168         ptr->data = item;
1169         ptr->next = head;
1170         head = ptr;
1171         printf("\nNode inserted");
1172     }
1173
1174 }
```

```
1175 void lastinsert()
1176 {
1177     struct node *ptr,*temp;
1178     int item;
1179     ptr = (struct node*)malloc(sizeof(struct node));
1180     if(ptr == NULL)
1181     {
1182         printf("\nOVERFLOW");
1183     }
1184     else
1185     {
1186         printf("\nEnter value?\n");
1187         scanf("%d",&item);
1188         ptr->data = item;
1189         if(head == NULL)
1190         {
1191             ptr -> next = NULL;
1192             head = ptr;
1193             printf("\nNode inserted");
1194         }
1195         else
1196         {
1197             temp = head;
1198             while (temp -> next != NULL)
1199             {
1200                 temp = temp -> next;
1201             }
1202             temp->next = ptr;
1203             ptr->next = NULL;
```

```
1204         printf("\nNode inserted");
1205
1206     }
1207 }
1208 }
1209 void randominsert()
1210 {
1211     int i,loc,item;
1212     struct node *ptr, *temp;
1213     ptr = (struct node *) malloc (sizeof(struct node));
1214     if(ptr == NULL)
1215     {
1216         printf("\nOVERFLOW");
1217     }
1218     else
1219     {
1220         printf("\nEnter element value");
1221         scanf("%d",&item);
1222         ptr->data = item;
1223         printf("\nEnter the location after which you want to insert ");
1224         scanf("\n%d",&loc);
1225         temp=head;
1226         for(i=0;i<loc;i++)
1227         {
1228             temp = temp->next;
1229             if(temp == NULL)
1230             {
1231                 printf("\ncan't insert\n");
1232                 return;
```

```
1233     }
1234
1235     }
1236     ptr ->next = temp ->next;
1237     temp ->next = ptr;
1238     printf("\nNode inserted");
1239 }
1240 }
1241 void begin_delete()
1242 {
1243     struct node *ptr;
1244     if(head == NULL)
1245     {
1246         printf("\nList is empty\n");
1247     }
1248     else
1249     {
1250         ptr = head;
1251         head = ptr->next;
1252         free(ptr);
1253         printf("\nNode deleted from the begining ...\n");
1254     }
1255 }
1256 void last_delete()
1257 {
1258     struct node *ptr,*ptr1;
1259     if(head == NULL)
1260     {
1261         printf("\nlist is empty");
```

```
1262     }
1263     else if(head -> next == NULL)
1264     {
1265         head = NULL;
1266         free(head);
1267         printf("\nOnly node of the list deleted ...\n");
1268     }
1269
1270     else
1271     {
1272         ptr = head;
1273         while(ptr->next != NULL)
1274         {
1275             ptr1 = ptr;
1276             ptr = ptr ->next;
1277         }
1278         ptr1->next = NULL;
1279         free(ptr);
1280         printf("\nDeleted Node from the last ...\n");
1281     }
1282 }
1283 void random_delete()
1284 {
1285     struct node *ptr,*ptr1;
1286     int loc,i;
1287     printf("\n Enter the location of the node after which you want to perform deletion \n");
1288     scanf("%d",&loc);
1289     ptr=head;
1290     for(i=0;i<loc;i++)
```

```
1291     {
1292         ptr1 = ptr;
1293         ptr = ptr->next;
1294
1295         if(ptr == NULL)
1296         {
1297             printf("\nCan't delete");
1298             return;
1299         }
1300     }
1301     ptr1 ->next = ptr ->next;
1302     free(ptr);
1303     printf("\nDeleted node %d ",loc+1);
1304 }
1305 void search()
1306 {
1307     struct node *ptr;
1308     int item,i=0,flag;
1309     ptr = head;
1310     if(ptr == NULL)
1311     {
1312         printf("\nEmpty List\n");
1313     }
1314     else
1315     {
1316         printf("\nEnter item which you want to search?\n");
1317         scanf("%d",&item);
1318         while (ptr!=NULL)
1319         {
```

```
1320         if(ptr->data == item)
1321         {
1322             printf("item found at location %d ",i+1);
1323             flag=0;
1324         }
1325         else
1326         {
1327             flag=1;
1328         }
1329         i++;
1330         ptr = ptr -> next;
1331     }
1332     if(flag==1)
1333     {
1334         printf("Item not found\n");
1335     }
1336 }
1337
1338 }
1339
1340 void display()
1341 {
1342     struct node *ptr;
1343     ptr = head;
1344     if(ptr == NULL)
1345     {
1346         printf("Nothing to print");
1347     }
1348     else
```



```
1349    {
1350        printf("\nprinting values . . . .\n");
1351        while (ptr!=NULL)
1352        {
1353            printf("\n%d",ptr->data);
1354            ptr = ptr -> next;
1355        }
1356    }
1357 }
1358
1359
```

1360