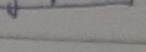
singly linked list

① → Declaring header files

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h> → for the memory.
#include <process.h>
```

② → creating structure for node which has info and link




```
struct node
{
  int info;
  struct node* link;
};
```

③ → giving [struct node] one name so that writing will be easy using typedef.

```
typedef struct node * NODE;
```

④ Declaring function prototype for getting a node

```
NODE getnode()
{
  NODE x;
```

⑤ taking malloc for memory allocation for allocating memory during run time

```
x = (NODE) malloc (sizeof(NODE));
```

⑥ If memory is full printing that memory full and exiting from that.

if memory is not full
returning memory

(8) Declaring function prototype for free node, to free up the node (during deleting node) memory
using free (memory allocated).

(9) Declaring function prototype for inserting the item at front end.
→ In this creating a node called temp
→ then inserting an item to that node
→ then checking if the link of that temp node is null or not,
→ if it is null returning address of temp node
→ if it is not null returning temp of link.

(10) Declaring function prototype for inserting at rear ends.
→ In this creating a node called temp
→ then inserting item,
→ same as inserting item at front end.
→ here difference is that taking another node same as temp.

(11) Declaring prototype for inserting at any end
→ same as inserting at front end
→ but here previous current node are used to insert the an integer at any position.

→ In the main taking in which position should we insert an item,
→ then inserting it.

(12) Declaring prototype for deleting at ~~rear~~ front end.

* If (first node is ~~empty~~ Null)
  Print list is empty.
* or else assign that first ~~node~~ assigned node to temp.
  → temp = temp of link
  ⇒ then print the deleted item or info

(13) Declaring prototype for deleting node at rear end.

same as that of front end.
here prev and cur is used to move to last end.

(14) Declaring prototype for deleting node at specified position.
  → using count to see if they are equal to specified ~~count~~ position
  → then displaying deleted position

⇒ In the main,

taking switch for cases

In

case 1 :- ~~calling~~ inserting at front
calling the function for front

case 2 : inserting at rear
calling the function for insert end.

case 3 : inserting at any position
calling the function for positioned.

case 4 : ~~list~~ deleting at front

case 5 : deleting at rear

case 6 : deleting at specified location

case 7 : ~~exit~~ display

case 8 : exit