6.6 HEADER LINKED LISTS

A header linked list is a special type of linked list which contains a header node at the beginning of the list. So, in a header linked list, START will not point to the first node of the list but START will contain the address of the header node. The following are the two variants of a header linked list:

- *Grounded header linked list* which stores NULL in the next field of the last node.
- *Circular header linked list* which stores the address of the header node in the next field of the last node. Here, the header node will denote the end of the list.

Look at Fig. 6.65 which shows both the types of header linked lists.

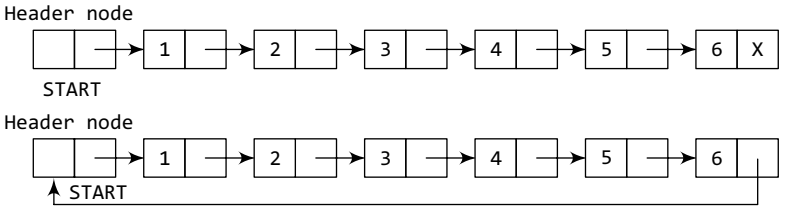
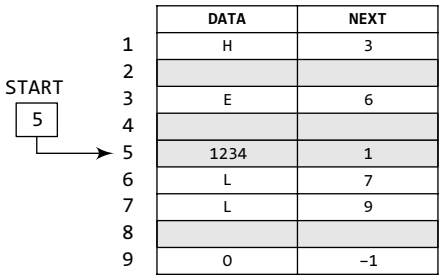
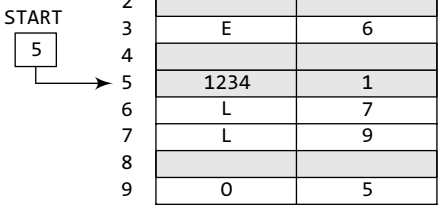


Figure 6.65 Header linked list



	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	O	–1

Figure 6.66 Memory representation of a header linked list



	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	O	5

Figure 6.67 Memory representation of a circular header linked list

As in other linked lists, if START = NULL, then this denotes an empty header linked list. Let us see how a grounded header linked list is stored in the memory. In a grounded header linked list, a node has two fields, DATA and NEXT. The DATA field will store the information part and the NEXT field will store the address of the node in sequence. Consider Fig. 6.66.

Note that START stores the address of the header node. The shaded row denotes a header node. The NEXT field of the header node stores the address of the first node of the list. This node stores H. The corresponding NEXT field stores the address of the next node, which is 3. So, we will look at address 3 to fetch the next data item.

Hence, we see that the first node can be accessed by writing FIRST_NODE = START –> NEXT and not by writing START = FIRST_NODE. This is because START points to the header node and the header node points to the first node of the header linked list.

Let us now see how a circular header linked list is stored in the memory. Look at Fig. 6.67.

Note that the last node in this case stores the address of the header node (instead of –1).

Hence, we see that the first node can be accessed by writing FIRST_NODE = START –> NEXT and not writing START = FIRST_NODE. This is because START points to the header node and the header node points to the first node of the header linked list.

Let us quickly look at Figs 6.68, 6.69, and 6.70 that show the algorithms to traverse a circular header linked list, insert a new node in it, and delete an existing node from it.

```
Step 1: SET PTR = START –> NEXT
Step 2: Repeat Steps 3 and 4 while PTR != START
Step 3:           Apply PROCESS to PTR –> DATA
Step 4:           SET PTR = PTR –> NEXT
        [END OF LOOP]
Step 5: EXIT
```

Figure 6.68 Algorithm to traverse a circular header linked list

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL –> NEXT
Step 4: SET PTR = START –> NEXT
Step 5: SET NEW_NODE –> DATA = VAL
Step 6: Repeat Step 7 while PTR –> DATA != NUM
Step 7:     SET PTR = PTR –> NEXT
        [END OF LOOP]
Step 8: NEW_NODE –> NEXT = PTR –> NEXT
Step 9: SET PTR –> NEXT = NEW_NODE
Step 10: EXIT
```

Figure 6.69 Algorithm to insert a new node in a circular header linked list

```
Step 1: SET PTR = START->NEXT
Step 2: Repeat Steps 3 and 4 while
        PTR –> DATA != VAL
Step 3:     SET PREPTR = PTR
Step 4:     SET PTR = PTR –> NEXT
        [END OF LOOP]
Step 5: SET PREPTR –> NEXT = PTR –> NEXT
Step 6: FREE PTR
Step 7: EXIT
```

Figure 6.70 Algorithm to delete a node from a circular header linked list

After discussing linked lists in such detail, these algorithms are self-explanatory. There is actually just one small difference between these algorithms and the algorithms that we have discussed earlier. Like we have a header list and a circular header list, we also have a two-way (doubly) header list and a circular two-way (doubly) header list. The algorithms to perform all the basic operations will be exactly the same except that the first node will be accessed by writing START –> NEXT instead of START.

Programming Example

5. Write a program to implement a header linked list.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
        int data;
        struct node *next;
};
struct node *start = NULL;
struct node *create_hll(struct node *);
struct node *display(struct node *);
int main()
{
        int option;
        clrscr();
        do
        {
                printf("\n\n *****MAIN MENU *****");
                printf("\n 1: Create a list");
                printf("\n 2: Display the list");
                printf("\n 3: EXIT");
                printf("\n Enter your option : ");
                scanf("%d", &option);
                switch(option)
                {
                        case 1: start = create_hll(start);
                                printf("\n HEADER LINKED LIST CREATED");
                                break;
```

```
                        case 2: start = display(start);
                                break;
                }
        }while(option !=3);
        getch();
        return 0;
}
struct node *create_hll(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter -1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num!=-1)
        {
                new_node = (struct node*)malloc(sizeof(struct node));
                new_node->data=num;
                new_node->next=NULL;
                if(start==NULL)
                {
                        start = (struct node*)malloc(sizeof(struct node));
                        start->next=new_node;
                }
                else
                {
                        ptr=start;
                        while(ptr->next!=NULL)
                                ptr=ptr->next;
                        ptr->next=new_node;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        }
        return start;
}
struct node *display(struct node *start)
{
        struct node *ptr;
        ptr=start;
        while(ptr!=NULL)
        {
                printf("\t %d", ptr->data);
                ptr = ptr->next;
        }
        return start;
}
```

Output

```
*****MAIN MENU *****
1: Create a list
2: Display the list
3: EXIT
Enter your option : 1
Enter -1 to end
Enter the data: 1
Enter the data: 2
Enter the data: 4
Enter the data: -1
HEADER LINKED LIST CREATED
Enter your option : 3
```