Unit-3: Singly Linked List

Outline:

- Static versus Dynamic Memory Allocation Technique
- Dynamic Memory Management Functions in C
  - malloc()
  - calloc()
  - realloc()
  - free()
- Definition of Linked List
- Array versus Linked List
- Types of Linked list
  - SLL, DLL, CSLL,CDLL
- Representation of a node in a SLL
- Basic Operations on a SLL
- Ordered SLL
- SLL with header node

Static versus Dynamic Memory Allocation Technique

Static allocation technique	Dynamic allocation technique
• Memory is allocated during compile time. • Once the memory is allocated, it cannot be modified during execution time. • Memory allocated may be over-utilized or under-utilized. • This technique is used when we know in advance how much amount of memory is required. • Faster execution • Example: Arrays	• Memory is allocated during execution time. • Memory allocated can be modified during execution time. • Memory can be used efficiently. • This technique is used when we can't predict the amount of memory required. • Slower execution • Example: Linked lists

Dynamic Memory Management

- For dynamic variables, the memory will be allocated in the "**Heap**" (free memory) area.
- There are *four dynamic memory management functions* which can be used for allocating and freeing the memory during execution time.
  - malloc()
  - calloc()
  - realloc()
  - free()
- The prototype for these functions is defined in <stdlib.h>.
- On successful allocation of memory dynamically, these functions return a **void pointer** to the first byte of the memory allocated.
- When dynamic memory allocation functions fail to allocate the memory required then they return **NULL**.

malloc()

- It is used to allocate a block of memory of specified size dynamically.
- The prototype for the malloc() function is defined in <stdlib.h>
- **Syntax:**

```
void * malloc(int);
        or
ptr =  (casttype *)malloc(int);
```

- It receives one argument which indicates the no. of bytes of memory to be allocated.
- On success, it returns a void pointer to the first byte of the memory allocated.
- On failure, it returns NULL.
- It is usually used to allocate memory for dynamic data structures such as Linked lists, trees etc.
- **Example:**

```
int *p;
p = (int*)malloc(8); // allocates 8 bytes of memory
p = (int *)malloc(5*sizeof(int)); // allocates 20 bytes of memory
```

Program1: *Develop a C program to allocate memory dynamically for an integer. Read an integer and check whether it is odd or even.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *num;
        num = (int*)malloc(sizeof(int));
        if(num==NULL)
        {
                printf("\nInsufficient Memory");
                exit(0);
        }
        printf("\nEnter an integer: ");
        scanf("%d",num);
        if(*num%2 == 0)
                printf("\n%d is an even number",*num);
        else
                printf("\n%d is an odd number",*num);
        free(num);
        return 0;
}
```

Program2: *Develop a C program to allocate memory dynamically for two real numbers. Read two real numbers and compute their sum.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        float *n1,*n2;
        n1 = (float*)malloc(sizeof(float));
        n2 = (float*)malloc(sizeof(float));
        if(n1==NULL  || n2 == NULL)
        {
                printf("\nInsufficient Memory");
                exit(0);
        }
        printf("\nEnter two real numbers:\n ");
        scanf("%f%f",n1,n2);

        printf("\n%f  + %f = %f ",*n1,*n2,*n1+*n2);
        free(n1);
        free(n2);
        return 0;
 }
```

Program3: *Develop a C program to allocate memory dynamically to store n integers. Read n integers and compute their sum.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p;
        int n,i,sum=0;

        printf("\nEnter the size of the array: ");
        scanf("%d",&n);
        p = (int*)malloc(n*sizeof(int));

        if(p == NULL)
        {
                printf("\nInsufficient Memory");
                exit(0);
         }

         printf("\nEnter %d elements for the array:\n ",n);
         for(i=0;i<n;i++)
         {
                scanf("%d",&p[i]);
                sum = sum +p[i];
          }

          printf("\nSum = %d",sum);

          free(p);

          return 0;
}
```

Program4: *Develop a C program to allocate memory dynamically to store the details of a book. Read the details of a book and print it.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>

struct Book
{
        char author[20];
        char title[20];
        int pages;
        float price;
};
```

```
int main()
{
        struct Book *b;
        b = (struct Book*)malloc(sizeof(struct Book));
        if(b == NULL)
        {
              printf("\nInsufficient Memory");
              exit(0);
        }

        printf("\nEnter the details of the book:\n");
        printf("Author: ");  gets(b->author);
        printf("Title: ");  gets(b->title);
        printf("No. of Pages: ");  scanf("%d",&b->pages);
        printf("Price: "); scanf("%f",&b->price);

        printf("\nDetails of the Book.....");
        printf("\nTitle: %s",b->author);
        printf("\nAuthor: %s",b->title);
        printf("\nPages: %d",b->pages);
        printf("\nPrice: %f",b->price);
        free(b);
        return 0;
}
```

Program5: *Develop a C program to allocate memory dynamically to store the details of n students. Read the details of n students and print the details of only those students belonging to "CSE".*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Student
{
        char name[20];
        char USN[20];
        char branch[20];
        float percent;
};

int main()
{
       struct Student *s;
       int i,n;
       printf("\nEnter the number of students: ");
       scanf("%d",&n);
       s = (struct Student*)malloc(n*sizeof(struct Student));
```

```
        if(s == NULL)
        {
                printf("\nInsufficient Memory");
                exit(0);
        }
        printf("\nEnter the details of %d students.....\n");
        for(i=0;i<n;i++)
        {
            printf("Enter name, USN, branch and percentage of student%d:\n",i+1);
            scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
        }

        printf("\nDetails of the students belonging to CSE.....\n");
        printf("NAME\tUSN\tPERCENTAGE\n");
        for(i=0;i<n;i++)
        {
                if(strcmp(s[i].branch,"CSE") == 0)
                        printf("%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].percent);
        }
        free(s);
        return 0;
}
```

free()

- It is used to free the memory allocated dynamically using malloc(), calloc() or realloc() functions.
- The prototype for the free() function is defined in <stdlib.h>
- **Syntax:**

  void free(void *);
- It receives one argument which is a pointer to the first byte of the memory allocated dynamically.
- It doesn't return any value.
- Once the memory is freed using free(), the pointer becomes a *dangling pointer*.
- **Example:**

```
int *p;
p = (int*)malloc(8); // allocates 8 bytes of memory
free(p);
p=NULL;
```

calloc()

- It is used to allocate multiple blocks of memory with each block of same size dynamically and initialize each byte to zero.
- The prototype for the calloc() function is defined in <stdlib.h>
- **Syntax:**

```
void * calloc(int,int);
        or
ptr =  (casttype *)calloc(int,int);
```

- It receives two arguments, first argument indicates the no. of blocks and second argument indicates the size of each block.
- On success, it returns a void pointer to the first byte of the memory allocated.
- On failure, it returns NULL.
- It is usually used to allocate memory for dynamic data structures such as dynamic arrays etc.
- **Example:**

```
int *p;
p = (int *)calloc(5,sizeof(int)); // allocates 20 bytes of memory
```

Program6: *Develop a C program to allocate memory dynamically to store n integers. Read n integers and find the largest integer.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *p;
    int n,i,large;

    printf("\nEnter the size of the array: ");
    scanf("%d",&n);
    p = (int*)calloc(n,sizeof(int));

    if(p == NULL)
    {
        printf("\nInsufficient Memory");
        exit(0);
    }
```

```
        printf("\nEnter %d elements for the array:\n ",n);
        for(i=0;i<n;i++)
              scanf("%d",&p[i]);


        large = p[0];
        for(i=1;i<n;i++)
        {
               if(p[i] > large)
                     large = p[i];
        }
        printf("\nLargest number = %d",large);

        free(p);

        return 0;
}
```

Program7: *Develop a C program to copy the contents of one string to another.*

Solution:

```
 #include<stdio.h>
 #include<stdlib.h>
 int main()
 {
         char str1[30],*str2;
         int i;

         printf("\nEnter a string: ");
         gets(str1);

         for(i=0;str1[i]!='\0';i++); //to find the length of the string

         str2 = (char*)calloc(i+1,sizeof(char));

         if(str2 == NULL)
         {
                 printf("\nInsufficient Memory");
                 exit(0);
         }

         for(i=0;str1[i]!='\0';i++)
                str2[i] =str1[i];
          str2[i] = '\0';
          printf("\nOriginal string:  %s",str1);
          printf("\nResultant string:  %s",str2);
          free(str2);
          return 0;
}
```

malloc() versus calloc()

malloc()	calloc()
• Allocates a single block of memory of specified size.	• Allocates multiple blocks of memory with each block of same size.
• It receives one argument. ptr = (casttype*)malloc(int);	• It receives two arguments. ptr = (casttype*)calloc(int,int);
• Allocated memory space is uninitialized.	• Allocated memory space is initialized to zero.
• Faster execution	• Slower execution.
• Best suitable for data structures such as Linked lists, trees etc.	• Best suitable for data structures such as dynamic arrays etc.

realloc()

- It is used to resize the memory allocated dynamically using malloc() or calloc().
- The prototype for the realloc() function is defined in <stdlib.h>
- **Syntax:**

```
void * realloc(void*,int);
        or
ptr =  (casttype *)realloc(ptr,int);
```

- It receives two arguments, first argument is the pointer to the old memory block and second argument is the new size in bytes.
- On success, it returns a void pointer to the first byte of the memory allocated.
- On failure, it returns NULL.
- **Example1:**

```
int *p;
p = (int *)malloc(10); // allocates 10 bytes of memory
p = (int*)realloc(p,2); //releases 8 bytes of memory
```

- If the new size specified in realloc() is smaller than the size of the old memory block , then realloc() releases extra bytes of memory allocated and returns the same pointer.

- **Example2:**

```
int *p;
p = (int *)malloc(10); // allocates 10 bytes of memory
p = (int*)realloc(p,20); //allocates additional 10 bytes of memory
```

- If the new size specified is larger than the size of the old memory block then realloc() does one of the following:
  - If it is possible to extend the memory block then realloc() extends the memory and returns the same pointer.
  - If it is not possible to extend then realloc() allocates entirely a new block, copies the contents of old block to new block, releases the old block and returns the pointer to the new memory block.
  - If it is not possible to allocate the specified amount of memory then realloc() returns NULL pointer.

Program8: *Develop a C program to allocate memory dynamically to store the string "SIT". Modify the memory allocated to store the string "SIDDAGANGA INSTITUTE OF TECHNOLOGY".*

Solution:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
        char *str;

         str = (char*)malloc(4*sizeof(char));

         strcpy(str,"SIT");
         printf("\nOriginal string: %s",str);

         str = (char*)realloc(str,35);

         strcpy(str,"SIDDAGANGA INSTITUTE OF TECHNOLOGY");
         printf("\nModified String: %s",str);

         free(str);

         return 0;
}
```

Program9: *Develop a C program to read n integers into an array and store odd numbers and even numbers into two separate arrays. Allocate memory dynamically for all the three arrays.*

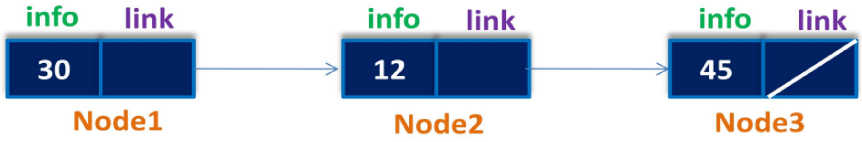
Solution:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *a,*b,*c;
        int i,j=0,k=0,n;
        printf("\nEnter the size of the array: ");
        scanf("%d",&n);

        a = (int*)calloc(n,sizeof(int));

        b = (int*)calloc(n,sizeof(int));

        c = (int*)calloc(n,sizeof(int));

        printf("\nEnter %d integers into the array:\n",n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
                if(a[i]%2 == 0)
                        b[j++] = a[i];
                else
                        c[k++] = a[i];
        }

        b = (int*)realloc(b,j*sizeof(int));
        c = (int*)realloc(c,k*sizeof(int));

        printf("\nArray containing  %d input integers:\n",n);
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);

        printf("\nArray containing  %d even integers:\n",j);
        for(i=0;i<j;i++)
                printf("%d\t",b[i]);

        printf("\nArray containing  %d odd integers:\n",k);
        for(i=0;i<k;i++)
                printf("%d\t",c[i]);

        free(a);
        free(b);
        free(c);
        return 0;
}
```

LINKED LISTS

- Linked list is a non-primitive linear data structure.

Definition: It is an ordered collection of nodes where each node consists of minimum two information: **info field** to hold the data and the **link field** to hold the address of next node in the list.



Differences between Array and Linked List

Array	Linked List
• It is a static data structure.	• It is a dynamic data structure.
• Once the memory is allocated, it cannot be modified during execution time.	• Memory allocated can be modified during execution time.
• Memory allocated may be over-utilized or under-utilized.	• Memory can be used efficiently.
• Memory allocated is contiguous.	• Memory allocated need not be contiguous.
• Accessing arbitrary element is easy.	• Accessing arbitrary element is time consuming since the list need to be traversed.
• Insertion and deletion operations are time consuming since shift overhead is involved.	• Insertion and deletion operations can be done easily just by rearranging the links.

Types of Linked Lists

SINGLY LINKED LIST (SLL):

- Each node contains the address of next node in the list.
- Last node's link field is NULL to indicate last node in the list.
- Traversing is possible in only forward direction.
- Any further node can be reached from the current node.
- Consumes less memory compared to DLL.

DOUBLY LINKED LIST (DLL):

- Each node contains the address of next node as well as address of previous node in the list.
- First node's llink field is NULL to indicate first node in the list.
- Last node's rlink field is NULL to indicate last node in the list.
- Traversing is possible in both forward and backward direction.
- Any successor node or any predecessor node can be reached from the current node.
- Consumes more memory compared to SLL.

CIRCULAR SINGLY LINKED LIST (CSLL):

- Each node contains the address of next node in the list.
- Last node's link field contains the address of first node in the list.
- Traversing is possible in only forward direction.
- Any further node can be reached from the current node.

CIRCULAR DOUBLY LINKED LIST (CDLL):

- Each node contains the address of next node as well as address of previous node in the list.
- First node's llink field contains the address of last node in the list.
- Last node's rlink field contains the address of first node in the list.
- Traversing is possible in both forward and backward direction.
- Any successor node or any predecessor node can be reached from the current node.

Representation of a node in a Singly Linked List

```
typedef struct node
{
        int info;
        struct node *next;
}NODE;
```

A structure containing a member which is a pointer to the same structure type is referred to as a **Self-Referential Structure**.

Allocating memory for a node in a Singly Linked List

getnode() function:

```
NODE *getnode(void)
{
        NODE *temp;
        temp = (NODE*)malloc(sizeof(NODE));
        return(temp);
}
```

Freeing memory for a node in a Singly Linked List

freenode() function:

```
void freenode(NODE *temp)
{
        free(temp);
}
```

Basic Operations on a Singly Linked List

Inserting a newnode at the beginning of SLL:

```
NODE * ins_first(NODE *first,int data)
{
        NODE *newnode;
         newnode = (NODE*)malloc(sizeof(NODE));
         newnode->info = data;
         newnode->next = first;
         printf("\nNode with info %d is inserted as first node in the list",data);
         return(newnode);
}
```

Inserting a newnode at the end of SLL:

```
NODE * ins_last(NODE *first,int data)
{
        NODE *newnode,*temp;
        newnode = (NODE*)malloc(sizeof(NODE));

        newnode->info = data;
        newnode->next = NULL

        if(first == NULL)
                first = newnode;
        else
        {
                temp = first;
                while(temp->next!=NULL)
                        temp = temp->next;
                temp->next = newnode;
        }
        printf("\nNode with info %d is inserted as last node in the list",data);
        return(first);
 }
```

Deleting the first node in a SLL:

```
NODE * del_first(NODE *first)
{
        NODE *temp;
        if(first == NULL)
                printf("\nEmpty List");
         else
         {
                temp = first;
                first = first->next;
```

```
                printf("\nFirst node with info %d is deleted",temp->info);
                free(temp);
          }
          return(first);
}
```

Deleting the last node in a SLL:

```
NODE * del_last(NODE *first)
{
        NODE *temp,*prev=NULL;
        if(first == NULL)
                printf("\nEmpty List");
        else
        {
                temp = first;
                while(temp->next!=NULL)
                {       prev =temp;
                        temp = temp->next;
                }
                if(prev == NULL) //Single node
                        first = NULL;
                else
                        prev->next = NULL; //multiple nodes
                printf("\nLast Node with info %d is deleted",temp->info);
                free(temp);
        }
        return(first);
}
```

Displaying the contents of SLL:

```
void display(NODE *first)
{
        if(first == NULL)
        {
                printf("\nEmpty List");
                return;
        }
        printf("\nListContents:\nBegin->");
        while(first!=NULL)
        {
                printf("%d->",first->info);
                first = first ->next;
        }
        printf("End");
}
```

C Program to implement Singly Linked List to perform insertion, deletion and display operations.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int info;
        struct node *next;
}NODE;

NODE * ins_first(NODE *first,int data)
{
         NODE *newnode;
          newnode = (NODE*)malloc(sizeof(NODE));
          newnode->info = data;
          newnode->next = first;
          printf("\nNode with info %d is inserted as first node in the list",data);
          return(newnode);
}

NODE * ins_last(NODE *first,int data)
{
        NODE *newnode,*temp;
        newnode = (NODE*)malloc(sizeof(NODE));

        newnode->info = data;
        newnode->next = NULL;

        if(first == NULL)
                first = newnode;
        else
        {
                temp = first;
                while(temp->next!=NULL)
                        temp = temp->next;
                temp->next = newnode;
        }
        printf("\nNode with info %d is inserted as last node in the list",data);
        return(first);
 }

NODE * del_first(NODE *first)
{
         NODE *temp;
         if(first == NULL)
                 printf("\nEmpty List");
          else
          {
```

```
                temp = first;
                first = first->next;
                printf("\nFirst node with info %d is deleted",temp->info);
                free(temp);
          }
        return(first);
}

NODE * del_last(NODE *first)
{
         NODE *temp,*prev=NULL;
         if(first == NULL)
                printf("\nEmpty List");
        else
        {
                temp = first;
                while(temp->next!=NULL)
               {       prev =temp;
                       temp = temp->next;
               }
               if(prev == NULL) //Single node
                       first = NULL;
               else
                       prev->next = NULL; //multiple nodes
               printf("\nLast Node with info %d is deleted",temp->info);
               free(temp);
         }
         return(first);
 }

void display(NODE *first)
{
          if(first == NULL)
         {
                printf("\nEmpty List");
                return;
        }

        printf("\nListContents:\nBegin->");

        while(first!=NULL)
        {
                printf("%d->",first->info);
                first = first ->next;
         }
          printf("End");
}
```

```
int main()
{
      NODE *first=NULL;
      int choice,data;
      while(1)
     {
        printf("\n\n1:Ins@first\n2:Ins@last\n3:Del@first\n4:Del@last\n5:Display\n6:Exit");
            printf("\nEnter your choice: ");
            scanf("%d",&choice);
            switch(choice)
            {
                 case 1: printf("\nEnter the data to be inserted: ");
                         scanf("%d",&data);

                         first = ins_first(first,data);
                         break;

                case 2:  printf("\nEnter the data to be inserted: ");
                         scanf("%d",&data);

                         first = ins_last(first,data);
                         break;

               case 3:   first = del_first(first);
                         break;

               case 4:  first = del_last(first);
                         break;

               case 5: display(first);
                        break;

               case 6: exit(0);
               default: printf("\nInvalid choice");
           }
      }
      return 0;
}
```

C Program to implement Stack to perform Push, Pop and display operations using Singly Linked List.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int info;
        struct node *next;
}NODE;
```

```
//ins_first() function
NODE * ins_first(NODE *first,int data)
{
        NODE *newnode;
        newnode = (NODE*)malloc(sizeof(NODE));
        newnode->info = data;
        newnode->next = first;
        printf("\n%d is pushed on to the stack",data);
        return(newnode);
}

//del_first() function
NODE * del_first(NODE *first)
{
        NODE *temp;
        if(first == NULL)
                printf("\nStack underflow");
        else
        {
                temp = first;
                first = first->next;
                printf("\n%d is popped from the stack",temp->info);
                free(temp);
        }
        return(first);
}

//display() function
void display(NODE *first)
{
        if(first == NULL)
        {
                printf("\nEmpty Stack");
                return;
        }
        printf("\nStack Contents:\nTop->");
        while(first!=NULL)
        {
                printf("%d->",first->info);
                first = first ->next;
        }
        printf("Bottom");
}

int main()
{
        NODE *first=NULL;
        int choice,data;
```

```
        while(1)
        {
                printf("\n\n1:PUSH\n2:POP\n3:Display\n4:Exit");
                printf("\nEnter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:   printf("\nEnter the data to be pushed: ");
                                  scanf("%d",&data);

                                  first = ins_first(first,data);
                                  break;

                        case 2:   first = del_first(first);
                                  break;

                        case 3:  display(first);
                                 break;
                        case 4: exit(0);

                        default: printf("\nInvalid choice");
                }
        }
        return 0;
}
```

Implementation of Queue to perform Insertion, Deletion and Display operations using Singly Linked List:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int info;
        struct node *next;
}NODE;

//ins_last() function
NODE * ins_last(NODE *first,int data)
 {
       NODE *newnode,*temp;
       newnode = (NODE*)malloc(sizeof(NODE));

       newnode->info = data;
       newnode->next = NULL;

       if(first == NULL)
             first=newnode;
```

```
        else
        {
                temp = first;
                while(temp->next!=NULL)
                        temp = temp->next;
                temp->next = newnode;
        }
        printf("\n%d is inserted into Queue",data);
        return(first);
   }
```

//del_first() function

```
NODE * del_first(NODE *first)
{
      NODE *temp;
      if(first == NULL)
            printf("\nQueue underflow");
      else
      {
            temp = first;
            first = first->next;
            printf("\n%d is deleted from the queue",temp->info);
            free(temp);
      }
     return(first);
}
```

//display() function

```
 void display(NODE *first)
 {
          if(first == NULL)
         {
                  printf("\nEmpty Queue");
                  return;
          }
          printf("\nQueue Contents:\nFront->");
          while(first!=NULL)
          {
                    printf("%d->",first->info);
                     first = first ->next;
           }
           printf("Rear");
}

int main()
{
        NODE *first=NULL;
        int choice,data;
```

```
        while(1)
        {
                printf("\n\n1:Insert\n2:Delete\n3:Display\n4:Exit");
                printf("\nEnter your choice: ");
                scanf("%d",&choice);
                switch(choice)
               {
                      case 1:   printf("\nEnter the data to be inserted: ");
                                scanf("%d",&data);

                                first = ins_last(first,data);
                                break;

                      case 2:   first = del_first(first);
                                break;

                      case 3:   display(first);
                                break;

                     case 4: exit(0);

                     default: printf("\nInvalid choice");
               }
         }
         return 0;
}
```

C Function to search for a node with specified key element in a SLL.

```
 void search_key(NODE *first)
 {
        int key;
        if(first==NULL)
              printf("\nEmpty list");
        else
        {
              printf("\nEnter the key to search: ");
              scanf("%d",&key);

              while(first!=NULL && first->info!=key)
                    first = first->next;

              if(first == NULL)
                    printf("\nFailure, Node with key %d is not found",key);
               else
                    printf("\nSuccess, Node with key %d is found",key);
         }
}
```

C Function to display the info of the nodes whose value is in the range 100-200 in a SLL.

```
void display(NODE *first)
{
        if(first==NULL)
        {
                printf("\nEmpty list");
                return;
        }
        printf("\nData in the range 100-200:");
        while(first!=NULL)
        {
                if(first->info >= 100 && first->info <= 200)
                        printf("\n%d",first->info);
                first = first->next;
         }
}
```

Lab Program7: *Define a structure to represent a node in a Singly Linked List. Each node must contain following information: player name, team name and batting average. Develop a C program using functions to perform the following operations on a list of cricket players:*

- *Add a player at the end of the list.*
- *Search for a specific player and update his/her batting average if the player exists.*
- *Display the details of all the players.*

Solution:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
        char player[20];
        char team[20];
        float bavg;
        struct node *next;
}NODE;

NODE * addPlayer(NODE *first)
{
       NODE *newnode,*temp;
       newnode=(NODE*)malloc(sizeof(NODE));
       newnode->next=NULL;
       printf("\nEnter the player details.....\n");
       printf("Name: ");scanf("%s",newnode->player);
       printf("Team: ");scanf("%s",newnode->team);
```

```
        printf("Batting Average: ");scanf("%f",&newnode->bavg);

        if(first==NULL)
                first=newnode;
        else
        {
                temp=first;
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=newnode;
        }
        printf("\nPlayer %s is added at the end of the list",newnode->player);
        return first;
}

void display(NODE *first)
{
        if(first==NULL)
        {
                printf("\nEmpty list");
                return;
        }
        printf("\nPlayer Details.....\n");
        printf("\nNAME\tTEAM\tBATTING AVERAGE\n");
        while(first!=NULL)
        {
                printf("%s\t%s\t%f\n",first->player,first->team,first->bavg);
                first=first->next;
        }
}

NODE *searchPlayer(NODE *first)
{
        NODE *temp;
        char player[20];

        if(first==NULL)
                printf("\nEmpty list");
        else
        {
                printf("\nEnter the player name to search: ");
                scanf("%s",player);

                temp=first;
                while(temp!=NULL && strcmp(temp->player,player)!=0)
                        temp=temp->next;

                if(temp==NULL)
                        printf("\nPlayer %s not existing in the list",player);
```

```
                else
                {
                    printf("\nPlayer %s is existing in the list",player);
                    printf("\nCurrent batting average: %f",temp->bavg);

                    printf("\nEnter new value for batting average: ");
                    scanf("%f",&temp->bavg);

                    printf("\nBatting average of player %s is updated successfully ", player);
                }
        }

        return first;
}

int main()
{
        NODE *first=NULL;
        int choice;

        while(1)
        {
            printf("\n1:ADD PLAYER\n2:SEARCH PLAYER\n3:DISPLAY PLAYER\n4:EXIT");
            printf("\nEnter your choice: ");
            scanf("%d",&choice);

            switch(choice)
            {
                    case 1: first=addPlayer(first);
                            break;

                    case 2: first=searchPlayer(first);
                            break;

                    case 3: display(first);
                            break;

                    case 4: exit(0);

                    default: printf("\nInvalid choice");
            }
        }

        return 0;
}
```

Lab Program8: *Develop a C program to add two two-variable polynomials using Singly Linked list.*

```
#include <stdio.h>
#include<stdlib.h>
typedef struct node
{
        float coeff;
        float powx;
        float powy;
        int flag;
        struct node *next;
}NODE;

NODE * ins_last(NODE *first,float cf,float px,float py)
{
         NODE *newnode,*temp;
         newnode=(NODE*)malloc(sizeof(NODE));
         newnode->coeff=cf;
         newnode->powx=px;
         newnode->powy=py;
         newnode->flag=0;
         newnode->next=NULL;
         if(first==NULL)
                 first=newnode;
         else
        {
                temp=first;
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=newnode;
         }
         return first;
}

NODE * read_p(NODE *first)
{
          float cf,px,py;

          printf("\nEnter the coefficient: ");
          scanf("%f",&cf);

          while(cf!=999)
          {
                  printf("\nEnter power of x: ");
                  scanf("%f",&px);
                  printf("\nEnter power of y: ");
                  scanf("%f",&py);
                  first=ins_last(first,cf,px,py);
```

```
                printf("\nEnter the coefficient: ");
                scanf("%f",&cf);
        }
        return first;
}

void display(NODE *first)
{
         if(first==NULL)
         {
                 printf("\nEmpty list");
                 return;
         }

         while(first->next!=NULL)
         {
               printf("%.0f x^%.0f y^%.0f + ",first->coeff,first->powx,first->powy);
               first = first->next;
         }
         printf("%.0f x^%.0f y^%.0f ",first->coeff,first->powx,first->powy);
}

NODE *add_p(NODE *p1,NODE *p2,NODE *p3)
{
         NODE *temp;
         float cf;
         temp = p2;
         while(p1!=NULL)
        {
                while(p2!=NULL)
                {
                      if((p1->powx == p2->powx) &&(p1-> powy == p2->powy))
                            break;
                      p2 = p2->next;
                }
                if(p2==NULL)
                      p3=ins_last(p3,p1->coeff,p1->powx,p1->powy);
                else
                {
                      cf = p1->coeff + p2->coeff;
                      p2->flag = 1;

                      if(cf!=0)
                            p3=ins_last(p3,cf,p1->powx,p1->powy);
                }
                p2=temp;
                p1=p1->next;
         }
```

```
        while(p2!=NULL)
        {
            if(p2->flag==0)
            p3 = ins_last(p3,p2->coeff,p2->powx,p2->powy);
            p2=p2->next;
        }
        return p3;
}

int main()
{
    NODE *p1=NULL,*p2=NULL,*p3=NULL;

    printf("\nEnter the first polynomial:\n");
    p1= read_p(p1);
    printf("\nEnter the second polynomial:\n");
    p2 = read_p(p2);

    p3 = add_p(p1,p2,p3);

    printf("\n\nFirst polynomial:\n");
    display(p1);
    printf("\n\nSecond polynomial:\n");
    display(p2);
    printf("\n\nResultant polynomial:\n");
    display(p3);

    return 0;
}
```

Ordered List

A list in which the elements are arranged in some order is referred to as an ordered list.

C Function to construct an ordered SLL

```
NODE * Insert(NODE *first, int data)
{
    NODE *newnode,*temp,*prev;
    newnode = (NODE*)malloc(sizeof(NODE));
    newnode->info = data;

    //Empty list or data less than first node info
    if(first == NULL || data < first->info)
    {
        newnode->next = first;
        first = newnode;
    }
```

```
        else
        {
                temp = first;
                while(temp!=NULL && data>temp->info)
                {
                        prev = temp;
                        temp = temp->next;
                }
                if(temp == NULL || data != temp->info)
                {
                        prev->next = newnode;
                        newnode->next = temp;
                }
         }
         return first;
}
```

Ascending Priority Queue

C Program to implement Ascending Priority Queue.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int info;
        struct node *next;
}NODE;
NODE * Insert(NODE *first, int data)
{
        NODE *newnode,*temp,*prev;

        newnode = (NODE*)malloc(sizeof(NODE));
        newnode->info = data;

       //Empty list or data less than first node info
       if(first == NULL || data < first->info)
       {
                newnode->next = first;
                first = newnode;
       }
       else
       {
                temp = first;
                while(temp!=NULL && data>temp->info)
                {
                        prev = temp;
                        temp = temp->next;
                }
```

```
                prev->next = newnode;
                newnode->next = temp;
            }
            printf("\n%d is inserted into the queue", data);
            return first;
}

NODE * del_first(NODE *first)
{
            NODE *temp;
            if(first == NULL)
                    printf("\nQueue Underflow");
             else
             {
                    temp = first;
                    first = first->next;
                    printf("\n%d is deleted from the queue",temp->info);
                    free(temp);
             }
              return(first);
}

void display(NODE *first)
 {
             if(first == NULL)
            {
                    printf("\nEmpty Queue");
                    return;
            }
            printf("\nQueue Contents:\nBegin->");
            while(first!=NULL)
            {
                    printf("%d->",first->info);
                    first = first ->next;
             }
             printf("End");
}

int main()
{
            NODE *first=NULL;
            int choice,data;

           while(1)
           {
                    printf("\n\n1:Insert\n2:Delete\n3:Display\n4:Exit");
                    printf("\nEnter your choice: ");
                    scanf("%d",&choice);
```

```
            switch(choice)
            {
                case 1: printf("\nEnter the data to be inserted: ");
                        scanf("%d",&data);

                        first = Insert(first,data);
                        break;

                case 2:  first = del_first(first);
                         break;

                case 3:  display(first);
                         break;

                case 4: exit(0);
                default: printf("\nInvalid choice");
            }
        }
        return 0;
}
```

C Function to compute union of two ordered SLLs/to concatenate two ordered SLLs/to merge two ordered SLLs.

```
NODE *list_union(NODE *L1,NODE *L2)
{
        NODE *L3 = NULL;
        if(L1 == NULL && L2 == NULL)
        {
                printf("\nBoth lists are empty");
                return NULL;
        }
        while(L1!=NULL && L2!=NULL)
        {
                if(L1->info < L2->info)
                {
                        L3 = ins_last(L3,L1->info);
                        L1 = L1->next;
                }
                else if(L2->info < L1->info)
                {
                        L3 = ins_last(L3,L2->info);
                        L2 = L2->next;
                }
                else
                {
                        L3 = ins_last(L3,L1->info);
                        L1 = L1->next;
```

```
                    L2 = L2->next;
                }
            }

            while(L1!=NULL)
            {
                L3 = ins_last(L3,L1->info);
                L1 = L1->next;
            }

            while(L2!=NULL)
            {
                L3 = ins_last(L3,L2->info);
                L2 = L2->next;
            }
            return(L3);
}
```

C Function to compute intersection of two ordered SLLs.

```
NODE *list_intersection(NODE *L1,NODE *L2)
{
        NODE *L3 = NULL;

        if(L1 == NULL || L2 == NULL)
        {
                printf("\nIntersection not possible");
                return NULL;
        }

        while(L1!=NULL && L2!=NULL)
        {
                if(L1->info < L2->info)
                        L1 = L1->next;
                else if(L2->info < L1->info)
                        L2 = L2->next;
                else
                {
                        L3 = ins_last(L3,L1->info);
                        L1  = L1->next;
                        L2 = L2->next;
                }
        }

        return(L3);
}
```

LAB PROGRAM9: *Develop a C program to construct two ordered singly linked lists using functions to perform following operations:*

- *Insert an element into a list.*
- *Merge the two lists.*
- *Display the contents of the list.*

Display the two input lists and the resultant list with suitable messages.

Solution:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct node
{
        int  info;
        struct node *next;
}NODE;

NODE *insert(NODE *first,int data)
{
        NODE *newnode,*temp,*prev;
        newnode=(NODE*)malloc(sizeof(NODE));
        newnode->info=data;

        if(first==NULL || data<first->info)
        {
                newnode->next=first;
                first=newnode;
        }
        else
        {
                temp=first;

                while(temp!=NULL && data>temp->info)
                {
                        prev=temp;
                        temp=temp->next;
                }

                if(temp==NULL || data!=temp->info)
                {
                        prev->next=newnode;
                        newnode->next=temp;
                 }
         }

         return first;
  }
```

```
NODE * ins_last(NODE *first,int data)
{
        NODE *newnode,*temp;
        newnode = (NODE*)malloc(sizeof(NODE));

        newnode->info = data;
        newnode->next = NULL;

        if(first == NULL)
                first = newnode;
        else
        {
                temp = first;
                while(temp->next!=NULL)
                        temp = temp->next;
                temp->next = newnode;
        }
        printf("\nNode with info %d is inserted as last node in the list",data);
        return(first);
}


void display(NODE *first)
{
       if(first==NULL)
       {
               printf("Empty");
               return;
       }

       printf("Contents:\nBegin->");

       while(first!=NULL)
      {
               printf("%d->",first->info);
               first=first->next;
        }
        printf("End");
}


NODE *merge(NODE *L1,NODE *L2)
{
         NODE *L3=NULL;
         if(L1==NULL && L2==NULL)
         {
                printf("\nList1 and List2 are Empty");
                return NULL;
         }
```

```
        while(L1!=NULL && L2!=NULL)
        {
              if(L1->info<L2->info)
              {
                    L3 = ins_last(L3,L1->info);
                    L1 = L1->next;
              }
              else if(L2->info<L1->info)
              {
                    L3 = ins_last(L3,L2->info);
                    L2 = L2->next;
              }
              else
             {
                    L3 = ins_last(L3,L1->info);
                    L1 = L1->next;
                    L2 = L2->next;
             }
        }

        while(L1!=NULL)
       {
              L3 = ins_last(L3,L1->info);
              L1 = L1->next;
       }

      while(L2!=NULL)
      {
            L3 = ins_last(L3,L2->info);
            L2 = L2->next;
       }

       printf("\nLists are merged successfully");

       return L3;
}


int main()
{
       NODE *L1=NULL,*L2=NULL,*L3=NULL;
       int data,choice;

      while(1)
      {
            printf("\n\n1:INS_LIST1\n2:INS_LIST2\n3:MERGE\n4:DISPLAY\n5:EXIT");
           printf("\nEnter your choice: ");
           scanf("%d",&choice);
```

```
            switch(choice)
            {
                  case 1:   printf("\nEnter the data: ");
                            scanf("%d",&data);

                            L1 = insert(L1,data);

                            break;

                  case 2:   printf("\nEnter the data: ");
                             scanf("%d",&data);

                             L2 = insert(L2,data);

                             break;

                   case 3:  L3=merge(L1,L2);

                             printf("\nList3 ");
                             display(L3);

                             break;

                   case 4:  printf("\nList1 ");
                             display(L1);

                             printf("\nList2 ");
                             display(L2);

                             break;

                    case 5: exit(0);

                    default:printf("\nInvalid choice");
            }
      }

      return 0;
}
```

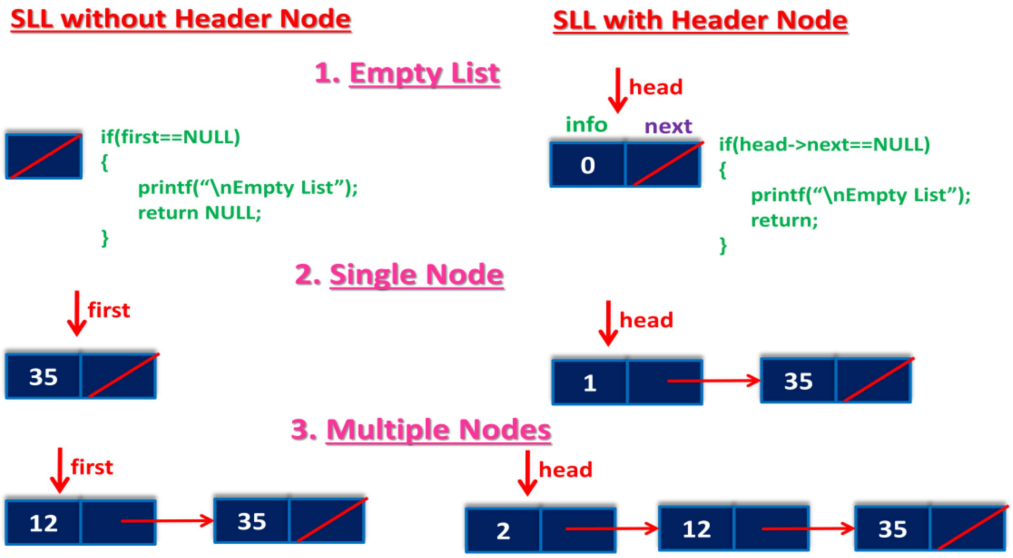
Header Node

- It is a special extra node present at the front of the list which is used to store some global information about the list. It doesn't represent the data item of the list.

Advantage:

- Using header node simplifies operations such as inserting at the end of the list, deleting the last node etc.

Diagrammatic Representation



Basic Operations on a Singly Linked List with header node

Inserting a newnode at the beginning of SLL with header node:

```
void  ins_first(NODE *head,int data)
{
          NODE *newnode;
           newnode = (NODE*)malloc(sizeof(NODE));
           newnode->info = data;
           newnode->next = head->next;
           head->next = newnode;
           head->info++;
           printf("\nNode with info %d is inserted as first node in the list",data);
}
```

Inserting a newnode at the end of SLL with header node:

```
void ins_last(NODE *head,int data)
{
        NODE *newnode,*temp;
        newnode = (NODE*)malloc(sizeof(NODE));

        newnode->info = data;
        newnode->next = NULL;

        temp = head;
        while(temp->next!=NULL)
              temp = temp->next;
       temp->next = newnode;
       head->info++;
       printf("\nNode with info %d is inserted as last node in the list",data);
}
```

Deleting the first node in a SLL with header node:

```
void del_first(NODE *head)
{
         NODE *temp;
         if(head->next == NULL)
               printf("\nEmpty List");
          else
          {
               temp = head->next;
               head->next = temp->next;
               printf("\nFirst node with info %d is deleted",temp->info);
               head->info--;
               free(temp);
          }
}
```

Deleting the last node in a SLL with header node:

```
void del_last(NODE *head)
{
         NODE *temp,*prev;
         if(head->next == NULL)
               printf("\nEmpty List");
        else
        {
               prev = head;
               temp = head->next;
               while(temp->next!=NULL)
               {
```

```
                prev = temp;
                temp = temp->next;
            }
            prev->next = NULL;
            printf("\nLast Node with info %d is deleted",temp->info);
            head->info--;
            free(temp);
        }

}
```

Displaying the contents of SLL with header node:

```
void display(NODE *head)
{
        NODE *temp;
        if(head->next == NULL)
        {
            printf("\nEmpty List");
            return;
        }
        printf("\nListContents:\nBegin->");
        temp=head->next;
        while(temp!=NULL)
        {
            printf("%d->",temp->info);
            temp = temp->next;
        }
        printf("End");
        printf("\nTotal number of nodes = %d",head->info);
}
```

C Program to implement Singly Linked List with header node to perform insertion, deletion and display operations.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
        int info;
        struct node *next;
}NODE;

void  ins_first(NODE *head,int data)
{
        NODE *newnode;
        newnode = (NODE*)malloc(sizeof(NODE));
        newnode->info = data;
```

```
            newnode->next = head->next;
            head->next = newnode;
            head->info++;
            printf("\nNode with info %d is inserted as first node in the list",data);
}

void ins_last(NODE *head,int data)
{
          NODE *newnode,*temp;
          newnode = (NODE*)malloc(sizeof(NODE));
          newnode->info = data;
          newnode->next = NULL;
          temp = head;
          while(temp->next!=NULL)
                temp = temp->next;
         temp->next = newnode;
         head->info++;
         printf("\nNode with info %d is inserted as last node in the list",data);
}

void del_first(NODE *head)
{
          NODE *temp;
          if(head->next == NULL)
                printf("\nEmpty List");
           else
           {
                temp = head->next;
                head->next = temp->next;
                printf("\nFirst node with info %d is deleted",temp->info);
                head->info--;
                free(temp);
            }
}

void del_last(NODE *head)
{
          NODE *temp,*prev;
          if(head->next == NULL)
                printf("\nEmpty List");
         else
         {
                prev = head;
                temp = head->next;
                while(temp->next!=NULL)
                {
                     prev = temp;
                     temp = temp->next;
                }
```

```
                prev->next = NULL;
                printf("\nLast Node with info %d is deleted",temp->info);
                head->info--;
                free(temp);
          }

 }

void display(NODE *head)
{
           NODE *temp;
           if(head->next == NULL)
           {
                  printf("\nEmpty List");
                  return;
          }
          printf("\nListContents:\nBegin->");
          temp=head->next;
          while(temp!=NULL)
          {
                    printf("%d->",temp->info);
                    temp = temp->next;
           }
           printf("End");
           printf("\nTotal number of nodes = %d",head->info);
}

int main()
{
       NODE *head;
       int choice,data;

       head = (NODE*)malloc(sizeof(NODE));
       head->info = 0;
       head->next = NULL;
      while(1)
      {
         printf("\n\n1:Ins@first\n2:Ins@last\n3:Del@first\n4:Del@last\n5:Display\n6:Exit");
              printf("\nEnter your choice: ");
              scanf("%d",&choice);

              switch(choice)
              {
                   case 1: printf("\nEnter the data to be inserted: ");
                              scanf("%d",&data);

                              ins_first(head,data);
                              break;
```

```
                case 2:  printf("\nEnter the data to be inserted: ");
                         scanf("%d",&data);

                         ins_last(head,data);
                         break;

                case 3:   del_first(head);
                         break;

                case 4:  del_last(head);
                         break;

                case 5: display(head);
                        break;

                case 6: exit(0);

               default: printf("\nInvalid choice");
            }
        }

        return 0;
}
```

********** END OF UNIT - 3 **********