Data Structures

- theory that tells us how to best organise data for easy storage and access

Linked Lists

- node : an element of a linked list
- node = data + pointer to next node

start --> [data | ptr] --> [data | ptr] --> ..... --> [data | ptr] --> NULL

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

struct node
{
   int data;
   struct node *ptr;
};

struct node *create_node();
int is_empty(struct node *);
void add_beg(struct node **, int);
void add_end(struct node *, int);
int delete_beg(struct node **);
int delete_end(struct node *);
void print_list(struct node *);
void free_all(struct node **);

int main()
{
   char ch;
   int x;
   struct node *start;
   start = NULL;

   printf("A - Add a node at the beginning.\n");
   printf("B - Add a node at the end.\n");
   printf("C - Delete a node from the beginning.\n");
   printf("D - Delete a node from the end.\n");
   printf("P - Print the linked list.\n");
   printf("Q - Quit.\n\n");

   while(1)
   {
      printf("Enter your Choice: ");
      scanf(" %c",&ch);
      if(ch == 'A')
      {
         scanf(" %d",&x);
         add_beg(&start,x);
```

```
}
else if(ch == 'B')
{
   scanf(" %d",&x);
   if(start == NULL)
   {
      add_beg(&start,x);
   }
   else
   {
      add_end(start,x);
   }
}
else if(ch == 'C')
{
   if(is_empty(start)==1)
   {
      printf("\nLinked List is Empty.\n\n");
   }
   else
   {
      x = delete_beg(&start);
      printf("Deleted Node Data = %d\n",x);
   }
}
else if(ch == 'D')
{
   if(is_empty(start)==1)
   {
      printf("\nLinked List is Empty.\n\n");
   }
   else if(start->ptr == NULL)
   {
      x = delete_beg(&start);
      printf("Deleted Node Data = %d\n",x);
   }
   else
   {
      x = delete_end(start);
      printf("Deleted Node Data = %d\n",x);
   }
}
else if(ch == 'P')
{
   if(is_empty(start)==1)
   {
      printf("\nLinked List is Empty.\n\n");
   }
   else
   {
      print_list(start);
   }
}
else if(ch == 'Q')
```

```
        {
            free_all(&start);
            break;
        }
        else
        {
            printf("INVALID  INPUT\n");
        }
    }
    return 0;
}

struct node *create_node()
{
    return (struct node *) malloc(sizeof(struct node));
}


int is_empty(struct node *p)
{
    if(p == NULL) return 1;
    else return 0;
}

void add_beg(struct node **p, int x)
{
    struct node *temp;
    temp = create_node();
    temp->data = x;
    temp->ptr = *p;
    *p = temp;
}

void add_end(struct node *p, int x)
{
    while(p->ptr != NULL)
    {
        p = p->ptr;
    }
    p->ptr = create_node();
    p = p->ptr;
    p->data = x;
    p->ptr = NULL;
}

int delete_beg(struct node **p)
{
    struct node *temp;
    temp = *p;
    int x = temp->data;
    temp = temp->ptr;
    free(*p);
    *p = temp;
    return x;
```

```
}

int delete_end(struct node *p)
{
  struct node **temp;
  temp = &(p->ptr);
  while(((*temp)->ptr) != NULL)
  {
      temp = &((*temp)->ptr);
  }
   int x = (*temp)->data;
   free(*temp);
   *temp = NULL;
   return x;
}

void print_list(struct node *p)
{
   struct node *temp;
   printf("\n");
   for(temp=p; temp!=NULL; temp=temp->ptr)
   {
      printf("%d ",temp->data);
   }
   printf("\n\n");
}

void free_all(struct node **p)
{
   struct node *temp;
   temp = *p;
   while(temp!=NULL)
   {
      temp = temp->ptr;
      free(*p);
      *p = temp;
   }
}
```