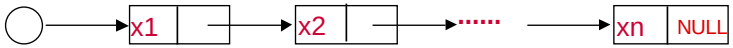
A singly linked list




```
typedef  struct  node*  Nodeptr;

struct  node  {
     int  data;
     Nodeptr  next;
};

Nodeptr  mknode  (int  item)
{
     Nodeptr  np;
     np = (Nodeptr)  malloc  (sizeof  (struct  node));
     if (np  !=  NULL)  {
          np->data  =  item;
          np->next  =  NULL;
     }  else  {
          fprintf  (stderr,  "malloc  failed\n");
          exit  (1);
     }
     return  np;
}
```


Inserting a node into a singly linked list

```
Nodeptr  insertsort  (Nodeptr  *list,  int  item)
{
     Nodeptr  np;
     if  (  (np  =  mknode(item))  !=  NULL)  {
          Nodeptr  curr  =  *list;
          Nodeptr  prev  =  NULL;

          /*  locate  the  position  of  this  node  in  the  list  * /
          while  (curr  !=  NULL  &&  item  <  curr->data)  {
               prev  =  curr;
               curr  =  curr->next;
          }
          /*  let  this  node  point  to  the  next  node  in  the  list  * /
          np->next  =  curr;

          /*  let  the  previous  node  in  the  list  point  to  this  node  * /
          if  (prev  !=  NULL)
               prev->next  =  np;
          else       /*  this  node  is  the  first  in  the  list  * /
               *list  =  np;
     }  else
          exit  (1);
     return  np;
}
```


Prints data values of all elements in the linked list

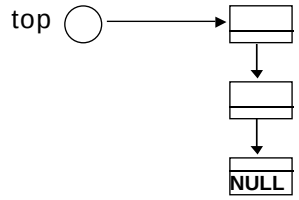
```
void  print  (Nodeptr  list)
{
   while  (  list  !=  NULL  )  {
     printf  ("%d  ",  list->data);
     list  =  list->next
   };
   printf  ("\n");
}
```

Use insert-sort function to place items in descending order and a print function to display them

```
int  main  (void)  {
   Nodeptr  list  =  NULL;
   int  item;

   while  (scanf  ("%d",  &item)  !=  EOF)  {
     if  (insertsort  (&list,  item)  ==  NULL)
          break;
   };
   print  (list);
   return  0;
}
```


A linked representation of a stack




```
#define STACKFULL    -7777
#define STACKEMPTY   -6666

typedef struct node* Nodeptr;

typedef struct node {
    int data;
    Nodeptr next;
} Node;    /* defining Node to be struct node * /

void clearstack (Nodeptr *ptr)
{
     while (*ptr != NULL)
          printf ("%d\n", pop(ptr));
}
```



```
int push (int item, Nodeptr *top)
{                          // top is a pointer to a Nodeptr, so that it can change
    Nodeptr np;
    np = (Nodeptr) malloc (sizeof (Node));
    if (np == NULL)
        return STACKFULL;
    else {
        np->data = item;
        np->next = *top;
        *top = np;
    } ;
    return item;
}
```

```
int pop (Nodeptr *top) {
     Nodeptr np;
     int item;
     if (*top != NULL) {
          item = (*top)->data;
          np = *top;
          *top = (*top)->next;
          free (np);
          return item;
     } ;
     return STACKEMPTY;
}
```



```
int stacksize (Nodeptr top)
{
    Nodeptr np = top;
    int size = 0;
    while (np != NULL) {
         size++;
         np = np->next;
    }
    return size;
}
```

```
void printstack (Nodeptr top)
{
     Nodeptr np = top;
     printf ("stack:");
     while (np != NULL) {
          printf ("%d ", np->data);
          np = np->next;
     }
     printf ("\n");
}
```

```
int main (void) {
     Nodeptr top = NULL;
     int item;
     while (scanf ("%d", &item) != EOF) {
          if (push (item, &top) == STACKFULL)
               break;
     }
     printstack (top);
     printf ("Stack size is %d\n", stacksize (top));
     clearstack (&top);
     if (pop(&top) == STACKEMPTY)
          printf ("Stack Empty\n");
     return 0;
}
```