

Simple List Partial Implementation**Partial List.h:**

```
#ifndef list_h
#define list_h

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

#define MAXLIST 10

struct ListRec
{
    char array[MAXLIST];
    int count;
};

typedef struct ListRec List;

void Initialise(List *lptr);
void Insert(List *lptr, char item);
void Delete(List *lptr, int pos);
bool IsFull(List *lptr);
bool IsEmpty(List *lptr);
int Find(List *lptr, char item);

/* and any more functions you can think of) */

#endif
```

Partial List.c:

```
#include "list.h"

void Initialise(List *lptr)
{
    lptr->count = 0;
}

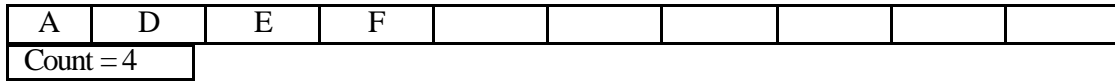
void Insert(List *lptr, char item)
{
    int pos;
    int current;

    if(IsFull(lptr))
    {
        fprintf(stderr, "List is full.");
        exit(1);
    }

    /***** A ****/
    /* Find the position in the list where you need to insert the item */
    for(pos = 0; pos < lptr->count; pos++)
    {
        if(lptr->array[pos] > item)
        {
            break;
        }
    }

    /* Start shuffling the items along in the array, starting at the end */
    current = lptr->count - 1;
    /***** B ****/
    while (current >= pos)
    {
        /* Copy the element into the next position along */
        lptr->array[current+1] = lptr->array[current];
        current--;
    }

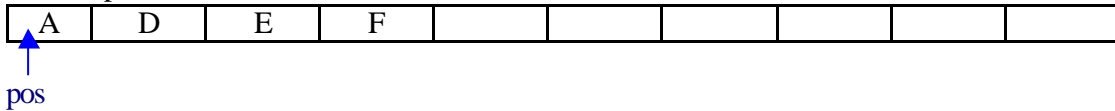
    /* Store the new character into the array */
    lptr->array[pos] = item;
    lptr->count++;
    /***** C ****/
}
}
```



Inserting 'B'

/*A*/ after pos has been initialised to 0

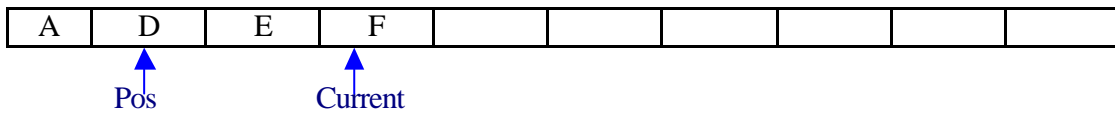
Find the position to insert:



The loop stops when array[pos] > item.

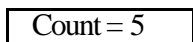
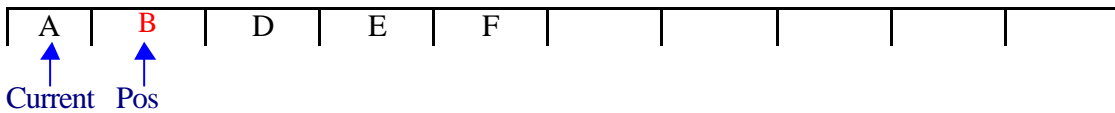
Then we need to shuffle the items after that position along to make space for the new element.

/*B*/



Copying the elements to the next position, this loop stops when it is < position, letting us copy the new item into the list.

/*C*/



Linked List Implementation

Partial LList.h:

```
#ifndef llist_h
#define llist_h

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

#define MAXLIST 10

struct EntryRec
{
    char ch;
    int next;
};

typedef struct EntryRec Entry;

struct LinkedListRec
{
    Entry array[MAXLIST];
    int count;
    int start;
};

typedef struct LinkedListRec LList;

void Initialise(LList *lptr);
void Insert(LList *lptr, char item);
void Delete(LList *lptr, int pos);
bool IsFull(LList *lptr);
bool IsEmpty(LList *lptr);
int Find(LList *lptr, char item);

/* and any more functions you can think of) */

#endif
```

Partial LList.c

```
#include "lList.h"

void Initialise(LList *lptr)
{
    lptr->count = 0;
    lptr->start = -1;
}
```


Note that there are special cases, one where you are inserting into an empty linked list (which can also be taken care of in the other special case if coded correctly), and the other when you are inserting right at the beginning of a linked list.

When you are inserting at the beginning of the linked list, you need to move the `start` value to be the new item, after linking in the new item to the original `start` position.

The best way to get used to linked lists, is to create several, and do the various operations, writing each representation of the linked list by hand as you are going. It's easier using arrows (thus should be easier when you learn to do this with pointers later on).