

Linked Lists

Data and File Structures Laboratory

<http://www.isical.ac.in/~dfs/lab/2017/index.html>

Traditional implementation



```
typedef struct {
```

```
    ...
```

```
} DATA;
```

```
typedef struct node {
```

```
    DATA data;
```

```
    struct node *next;
```

```
} NODE;
```

```
NODE *create_node(DATA d) {
```

```
    NODE *nptr;
```

```
    if (NULL == (nptr = Malloc(1, NODE)))
```

```
        ERR_MSG("out of memory");
```

```
    nptr->data = d;
```

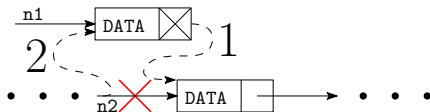
```
    nptr->next = NULL;
```

```
    return nptr;
```

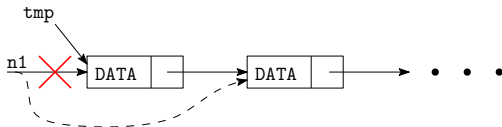
```
}
```

Traditional implementation

```
void insert(NODE *n1, NODE **n2) {  
    /* insert n1 in front of n2 */  
    if (n1 != NULL) {  
        n1->next = *n2;  
        *n2 = n1;  
    }  
}
```



```
void delete(NODE **n1) {  
    NODE *tmp;  
    if (n1 != NULL && *n1 != NULL) {  
        tmp = *n1;  
        *n1 = tmp->next;  
        free(tmp);  
    }  
}
```



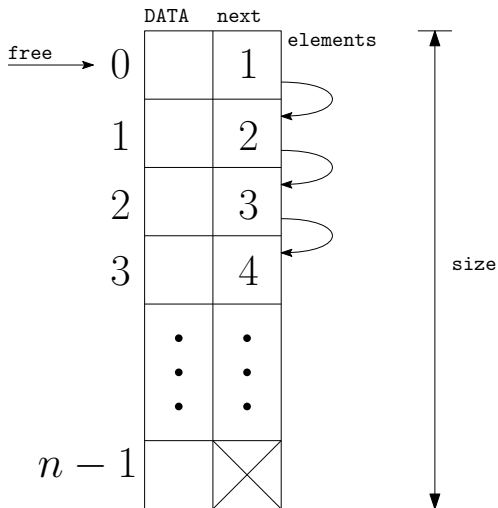
Check if boundary cases are correctly handled!

Alternative implementation

```
typedef struct {  
    ...  
} DATA;  
  
typedef struct {  
    DATA data;  
    int next;  
} NODE;  
  
typedef struct {  
    int head, free;  
    int length, size;  
    NODE *elements;  
} LIST;
```

Alternative implementation

```
typedef struct {  
    ...  
} DATA;  
  
typedef struct {  
    DATA data;  
    int next;  
} NODE;  
  
typedef struct {  
    int head, free;  
    int length, size;  
    NODE *elements;  
} LIST;
```



Alternative implementation

```
LIST create_list(int n) {
    int i;
    LIST l;
    if (NULL ==
        (l.elements = Malloc(n, NODE)))
        ERR_MESG("out of memory");
    for (i = 0; i < n-1; i++)
        l.elements[i].next = i+1;
    l.elements[n-1].next = -1;
    l.size = n;
    l.free = 0;
    l.head = -1;
    l.length = 0;
    return l;
}
```

Alternative implementation

```
LIST create_list(int n) {
    int i;
    LIST l;
    if (NULL ==
        (l.elements = Malloc(n, NODE)))
        ERR_MESG("out of memory");
    for (i = 0; i < n-1; i++)
        l.elements[i].next = i+1;
    l.elements[n-1].next = -1;
    l.size = n;
    l.free = 0;
    l.head = -1;
    l.length = 0;
    return l;
}

void insert(LIST *l, DATA *d, int *node)
{ /* insert d in front of node */
    int position = l->free;
    if (-1 == position)
        // no space left; what to do??
    l->free = l->elements[l->free].next;
    l->elements[position].data = *d;
    l->elements[position].next = *node;
    *node = position;
    l->length++;
}
}
```

What to do when no space left

```
if (-1 == position) {
    l->size *= 2;
    if (NULL == Realloc(l->elements, l->size, NODE))
        ERR_MESG("out of memory");
    l->free = l->size/2;
    for (i = l->size/2; i < l->size - 1; i++)
        l->elements[i].next = i+1;
    l->elements[l->size - 1] = -1;
    position = l->free;
}
```

Alternative implementation

```
void delete(LIST *l, int node;) {
    int tmp;
    if (-1 != node) {
        tmp = l->elements[node].next;
        if (-1 != tmp) {
            l->elements[node].next = l->elements[tmp].next;
            l->elements[tmp].next = l->free;
            l->free = tmp;
            l->length--;
        }
    }
}
```

1. Write a program that takes a single positive integer (say N) as a command line argument, generates N random integers between 0 and 10,000 one by one, and inserts them (one by one) into an initially empty list in sorted order.

Example:

Generated elements: 10, 3, 7, 1, ...

List:

10

 →

3	10
---	----

 →

3	7	10
---	---	----

 →

1	3	7	10
---	---	---	----

Use the following in turn to store the list:

- (a) an array;
- (b) a “traditional” linked list;
- (c) an array implementation of a linked list.

Run your program 5 times each for $N = 100, 500, 1000, 2000, 3000, \dots, 10000$. Print the sorted list to standard output, and the time taken (followed by a single tab, but no newline) to standard error. Find the average time taken for each value of N and for each implementation method given above. You may use the shell script given below.

2. Modify your program above so that it generates *two* sorted lists instead of one. Write a function to merge these two lists into a single sorted list. For this problem, use traditional linked lists only.

Problems III

3. Write a program that takes a linked list of linked lists, and creates a single flattened linked list, as shown in the example below.

Input

5	→	10	→	19	→	28
↓		↓		↓		↓
7		20		22		35
↓				↓		↓
8				50		40
↓						↓
30						45

Output

5 → 7 → 8 → 30 → 10 → 20 → 19 → 22 → 50
→ 28 → 35 → 40 → 45

Input file format:

```
4 # Number of lists
5 7 8 30 # List 1
10 20 # List 2
19 22 50 # List 3
28 35 40 45 # List 4
```

4. Given a list of numbers (provided as command line arguments), write a program to compute the nearest larger value for the number at position i (nearness is measured in terms of the difference in array indices). For example, in the array $[1, 4, 3, 2, 5, 7]$, the nearest larger value for 4 is 5.

Implement a naive, $O(n^2)$ time algorithm, as well as an $O(n)$ time algorithm for this problem. Compare the run times of your algorithms.

Shell script for Problem 1

```
1  # Assumes that your program prints the time followed by a tab /
    space
2  # e.g. using
3  # printf("%d\t", (int) DURATION(start_time, end_time));
4  cp /dev/null prob1-output.txt
5  for i in 100 500 {1000..10000..1000}; do
6      echo -n "$i "
7      for j in {1..5}; do
8          ./prog1 $i >> prob1-output.txt
9      done
10     echo ""
11 done
```