Linked Lists

Data and File Structures Laboratory

`http://www.isical.ac.in/~dfslab/2019/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)))  /* see common.h for definitions */
       ERR_MESG("out of memory");          /* of macros                    */
    nptr->data = d;
    nptr->next = NULL;
    return nptr;
}
```

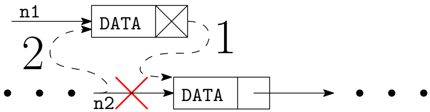
Traditional implementation

Functions:

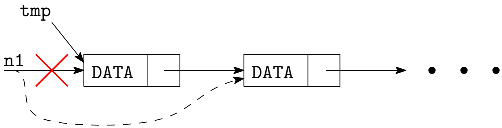
- `create_node()`
- `insert()`
  - insert at beginning / end
  - insert in front of given node ✓
  - insert after given node
- `delete()`
  - delete from beginning / end
  - delete given node ✓

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);
  }
}
```



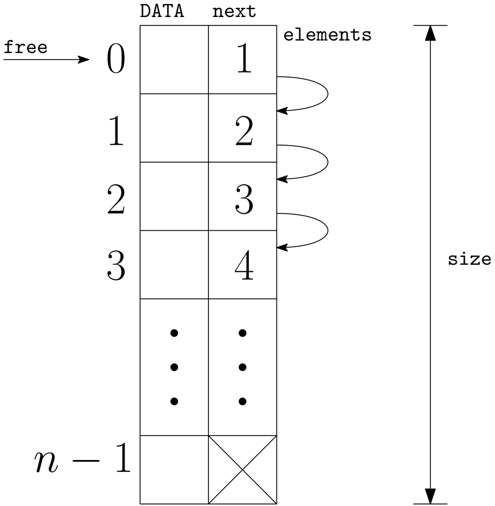
Check if boundary cases are correctly handled!

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--;
     }
  }
}
```

Problems I

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, $\ldots$

   List: $\boxed{10} \rightarrow \boxed{3 \mid 10} \rightarrow \boxed{3 \mid 7 \mid 10} \rightarrow \boxed{1 \mid 3 \mid 7 \mid 10}$

   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, $\ldots$, 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.

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 $\rightarrow$	10 $\rightarrow$	19 $\rightarrow$	28
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
7	20	22	35
$\downarrow$		$\downarrow$	$\downarrow$
8		50	40
$\downarrow$			$\downarrow$
30			45

Output

$5 \rightarrow 7 \rightarrow 8 \rightarrow 30 \rightarrow 10 \rightarrow 20 \rightarrow 19 \rightarrow 22 \rightarrow 50 \rightarrow 28 \rightarrow 35 \rightarrow 40 \rightarrow 45$

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
```

Problems IV

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.
5. Given a set of sorted numbers as user input, write a program to remove the duplicate elements.
6. Write a program that takes a linked list and an index (starting with 0) as user inputs, and prints the alternating elements in the linked list starting from the given index.

Problems V

7. Write a program to take the coefficients of two polynomials as user inputs and print the indices (powers of the terms) that get added and cancelled out in their multiplication result.

   Sample Input:

   1 1 (representing $(x + 1)$)

   1 -1 (representing $(x - 1)$)

   Sample Output:

   Added: 2 (x^2 gets added)

   Cancelled: 1 (x gets cancelled out)