Lecture 7:
Introduction to the Lists
(Array-based linked lists)
Lists
Lists

- We used successive data structures up to now:

  - If $a_{ij}$ in the memory location $L_{ij}$, then $a_{ij+1}$ is in $L_{ij+c}$ (c: constant)

  - In a queue, if the $i^{th}$ item is in $L_i$, $i+1$. item is in $(L_i+c)\%n$. (i.e. circular queue)

  - In a stack, if the top item is in $L_{T}$, the below item is in $L_{T-c}$.

  Insertion and deletion: $O(1)$
Sequential Access
(ascending or descending)

Example 1:
- Alphabetically ordered lists:
  - Ape
  - Butterfly
  - Cat
  - Dog
  - Mouse

- Delete ‘Ape’, what happens?
- Delete ‘Cat’, what happens?
- Add ‘Bear’, what happens?
- Add ‘Chicken’, what happens?
**Sequential Access**  
*(ascending or descending)*

**Example 2:**
- The result of the multiplication of two polynomials
  - \((x^7+5x^4-3x^2+4)(3x^5-2x^3+x^2+1)\)

<table>
<thead>
<tr>
<th>3</th>
<th>-2</th>
<th>1</th>
<th>1</th>
<th>15</th>
<th>-10</th>
<th>5</th>
<th>5</th>
<th>-9</th>
<th>6</th>
<th>-3</th>
<th>12</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Powers are not ordered. So either we need to sort or shift in order to solve this problem.
Sorted items

• We want to keep the items sorted, and we want to avoid the sorting cost.
  • We may need to sort after each insertion of a new item.
  • Or we need to do shifting.

What is the solution?
Towards the Linked List

- Each item has to have a second data field – link.
  - Each item has two fields: **data** and **link**.
Linked List

```c
#define MAX_LIST 10
#define TRUE 1
#define FALSE 0
#define NULL -1

typedef struct{
    char name[5];
    //other fields
    int link;
}item;

item linkedlist[MAX_LIST];
int free_;
void make_empty_list()
{
    int i;
    for(i=0;i<MAX_LIST-1;i++)
        list[i].link=i+1; // every item points the next

    linkedlist [MAX_LIST-1].link=NULL; // last item
    free_=0;
}
Linked List
--get item

Returns a free item from the list:

```c
int get_item(int* r)
{
    if(free_ == NULL) //there is no item to get
        return FALSE;
    else{
        *r = free_; //get the item which is pointed by free_
        free_ = linkedlist[free_].link; //points next free item
        return TRUE;
    }
}
```
Linked List
--return item

*Free the item:*

```c
void return_item(int r)
{
    linkedlist[r].link=free_;  //return item that is pointed by r
    free_=r;  //free the item
}
```
<table>
<thead>
<tr>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>1</td>
</tr>
<tr>
<td>[1]</td>
<td>2</td>
</tr>
<tr>
<td>[2]</td>
<td>3</td>
</tr>
<tr>
<td>[3]</td>
<td>4</td>
</tr>
<tr>
<td>[4]</td>
<td>5</td>
</tr>
<tr>
<td>[5]</td>
<td>6</td>
</tr>
<tr>
<td>[6]</td>
<td>7</td>
</tr>
<tr>
<td>[7]</td>
<td>8</td>
</tr>
</tbody>
</table>

...    ....    ...

...    ....    ...

-1

free_ = 0

List starts at 0 (*list=0)

<table>
<thead>
<tr>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Arzu</td>
</tr>
<tr>
<td>[1]</td>
<td>Ayse</td>
</tr>
<tr>
<td>[2]</td>
<td>Aziz</td>
</tr>
<tr>
<td>[3]</td>
<td>Bora</td>
</tr>
<tr>
<td>[6]</td>
<td>Ugur</td>
</tr>
<tr>
<td>[7]</td>
<td></td>
</tr>
</tbody>
</table>

free_ = 7
<table>
<thead>
<tr>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arzu</td>
<td>1</td>
</tr>
<tr>
<td>Ayse</td>
<td>2</td>
</tr>
<tr>
<td>Aziz</td>
<td>3</td>
</tr>
<tr>
<td>Bora</td>
<td>4</td>
</tr>
<tr>
<td>Kaan</td>
<td>7</td>
</tr>
<tr>
<td>Muge</td>
<td>6</td>
</tr>
<tr>
<td>Ugur</td>
<td>-1</td>
</tr>
<tr>
<td>Leyla</td>
<td>5</td>
</tr>
<tr>
<td>....</td>
<td>9</td>
</tr>
<tr>
<td>....</td>
<td>...</td>
</tr>
</tbody>
</table>

*free_ = 8 (“Leyla” added) *

*list = 0*

<table>
<thead>
<tr>
<th>name</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eyup</td>
<td>4</td>
</tr>
<tr>
<td>Ayse</td>
<td>2</td>
</tr>
<tr>
<td>Aziz</td>
<td>3</td>
</tr>
<tr>
<td>Bora</td>
<td>0</td>
</tr>
<tr>
<td>Kaan</td>
<td>7</td>
</tr>
<tr>
<td>Muge</td>
<td>6</td>
</tr>
<tr>
<td>Ugur</td>
<td>-1</td>
</tr>
<tr>
<td>Leyla</td>
<td>5</td>
</tr>
<tr>
<td>....</td>
<td>9</td>
</tr>
<tr>
<td>....</td>
<td>...</td>
</tr>
</tbody>
</table>

*free_ = 0 (“Arzu” deleted) *

*free_ = 8 (“Eyup” added) *

*list = 1*
void insert_item(char name[], int* list)
{
    int r, q, p;
    if(get_item(&r)){
        strcpy(linkedlist[r].name, name);
        q = NULL;
        p = *list;
        while( p != NULL && strcmp(linkedlist[p].name, name) < 0) { //search right position
            q = p;
            p = linkedlist[p].link;
        }
        if(q == NULL){ //new item is inserted to the front of the list.
            *list = r;
            linkedlist[r].link = p;
        }
        else{ //new item is inserted in the middle
            linkedlist[q].link = r;
            linkedlist[r].link = p;
        }
    }
    else printf("\n not enough free space!!");
}
Linked List
--delete item

```c
void delete_item(char name[], int* list)
{
    int q,p;
    q = NULL;
    p = *list;
    int l;
    while( p != NULL && (l = strcmp(linkedlist[p].name, name)) < 0 ) { //search for the item
        q = p;
        p = linkedlist [p].link;
    }
    if(p == NULL || l>0) { //end of the list
        printf("\n %s cannot be found!! ", name);
    } else if( q == NULL) { //the first item of the list will be deleted.
        *list = linkedlist [p].link;
        return_item(p);
    } else{ //get the item pointed by ‘p’
        linkedlist [q].link = linkedlist [p].link;
        return_item(p);
    }
}
```