CS 214 – DATA STRUCTURES FALL 2017

Lecture 6 – Linked Lists

Group **A & C**

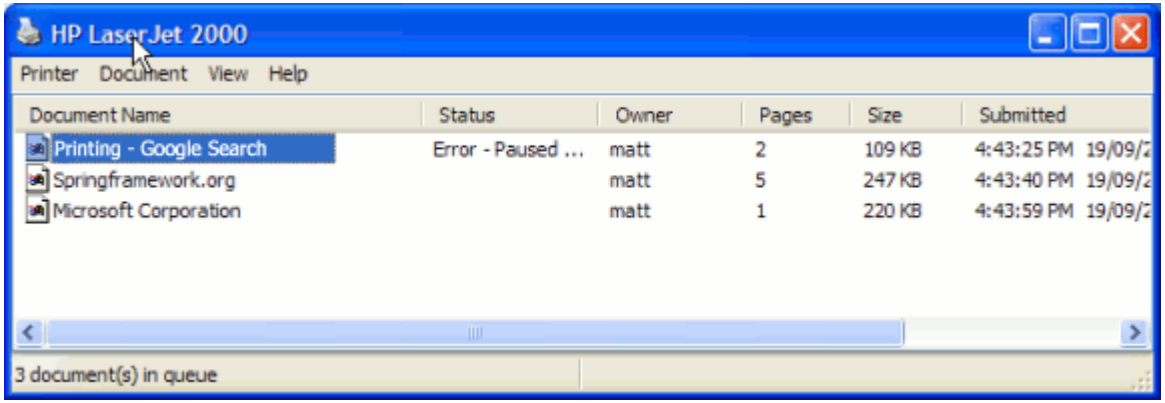
October 23rd 2017

Dr. Mai Hamdalla

mai@fci.helwan.edu.eg

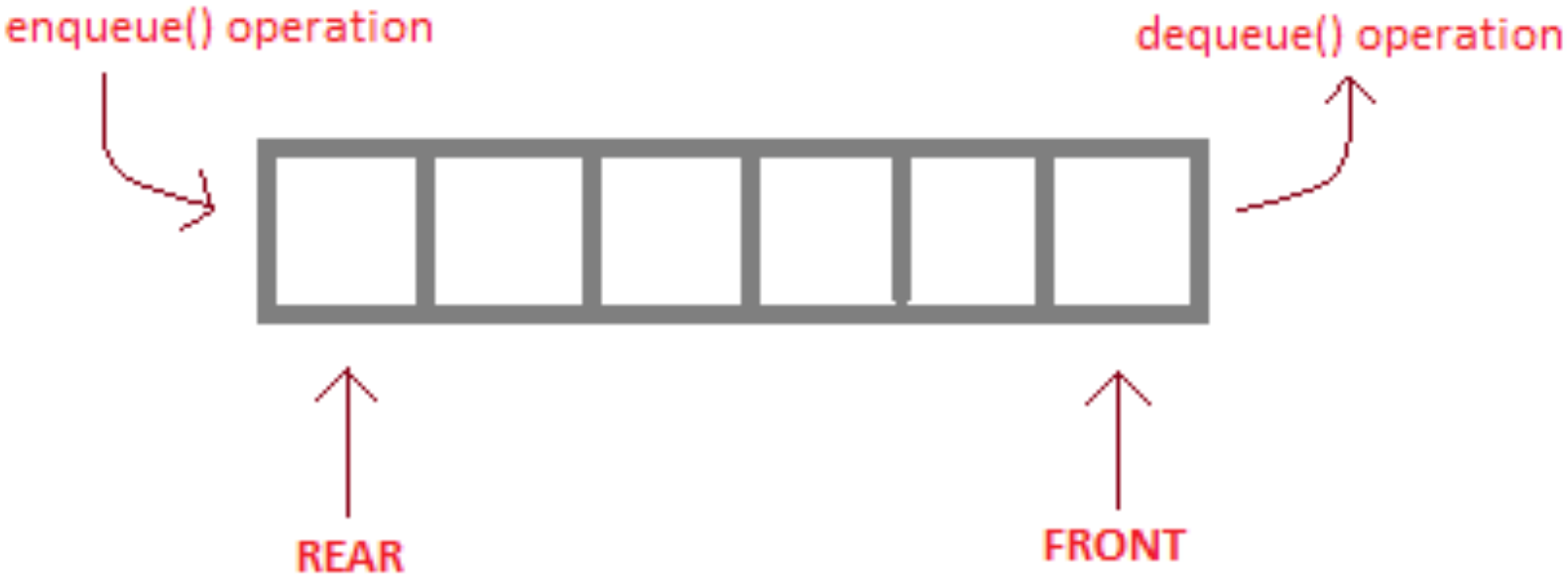
QUEUES

What is a Queue?



What is a Queue?

- Queue is FIFO Structure
- Queue is Ordered List of Elements of **Same Type**.
- New elements are added at one end called **rear** or **tail**.
- Existing elements are deleted from other end called **front** or **head**.



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

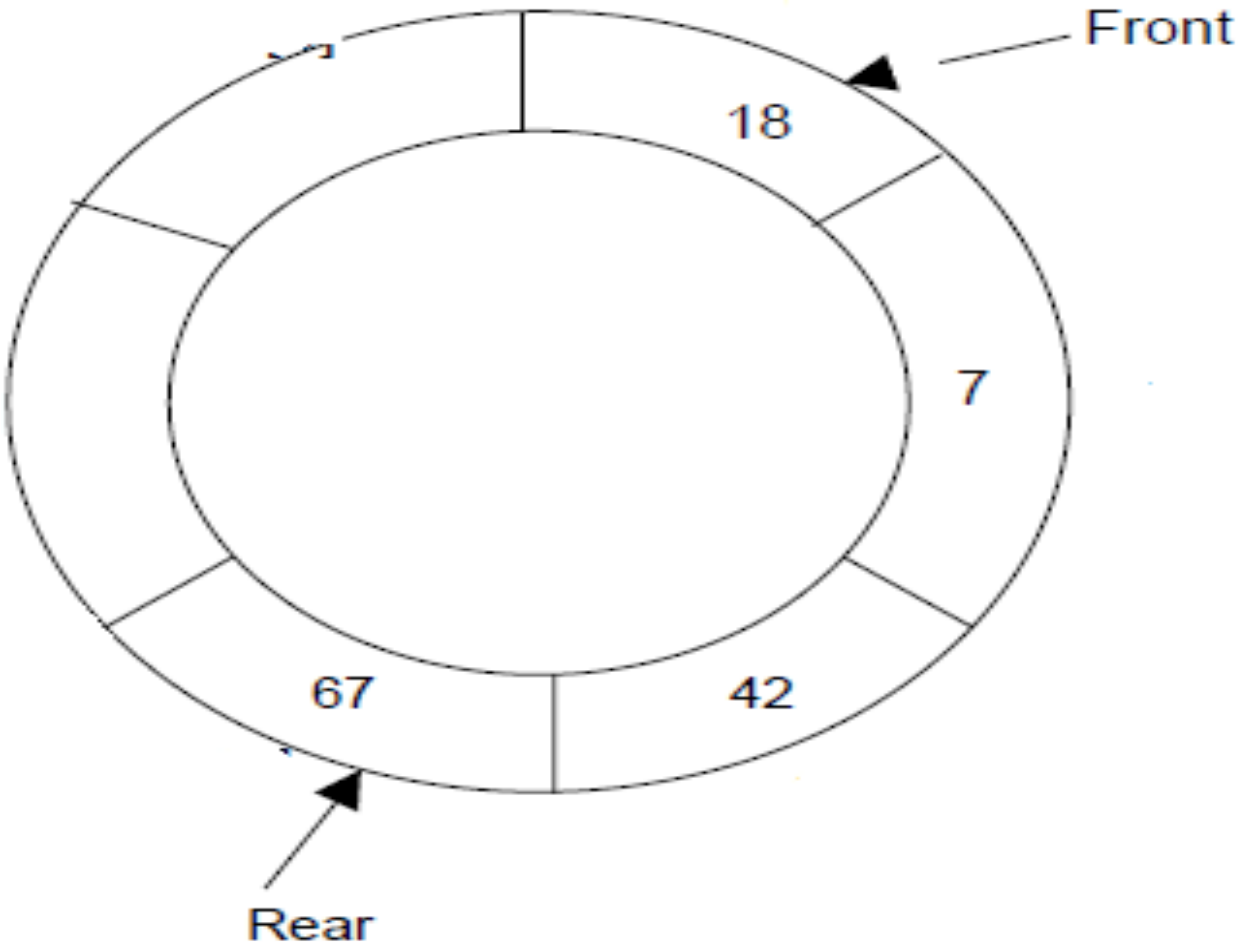
QUEUE DATA STRUCTURE

Operations Performed on Queue

- Create the queue, leaving it empty.
- Determine whether the queue is empty or not.
- Determine whether the queue is full or not.
- Enqueue a new entry onto the end of the queue
- Dequeue the entry at the front of the queue.

Queue Implementation

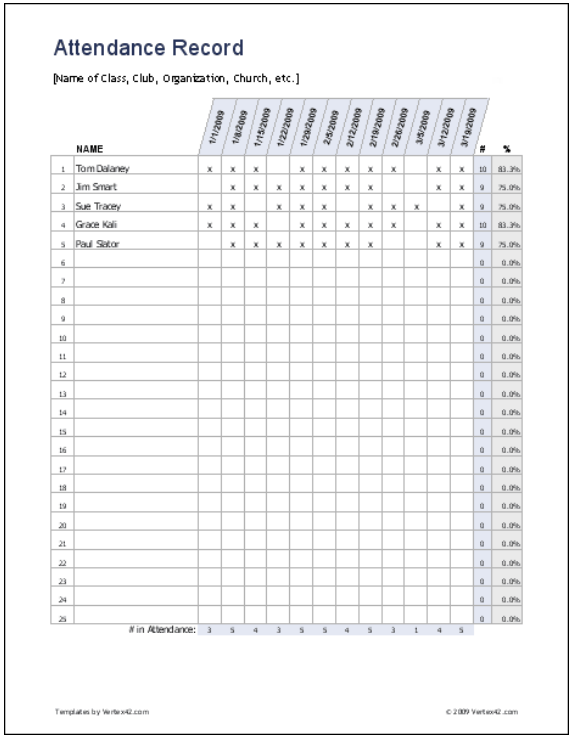
A better solution is a circular Queue



LISTS

What is a list?

- A number of connected items or names written or printed consecutively.



Attendance Record

[Name of Class, Club, Organization, Church, etc.]

	NAME	1/1/2009	1/8/2009	1/15/2009	1/22/2009	1/29/2009	2/5/2009	2/12/2009	2/19/2009	2/26/2009	3/5/2009	3/12/2009	3/19/2009	#	%
1	Tom Delaney	x	x	x		x	x	x	x	x		x	x	10	83.3%
2	Jim Smart		x	x	x	x	x	x				x	x	9	75.0%
3	Sue Tracey	x	x		x	x	x		x	x	x		x	9	75.0%
4	Grace Kali	x	x	x		x	x	x	x	x		x	x	10	83.3%
5	Paul Slator		x	x	x	x	x	x	x			x	x	9	75.0%
6														0	0.0%
7														0	0.0%
8														0	0.0%
9														0	0.0%
10														0	0.0%
11														0	0.0%
12														0	0.0%
13														0	0.0%
14														0	0.0%
15														0	0.0%
16														0	0.0%
17														0	0.0%
18														0	0.0%
19														0	0.0%
20														0	0.0%
21														0	0.0%
22														0	0.0%
23														0	0.0%
24														0	0.0%
25														0	0.0%
	# in Attendance:	3	5	4	3	5	5	4	5	3	1	4	5		

Templates by Vertex42.com

What is a List?

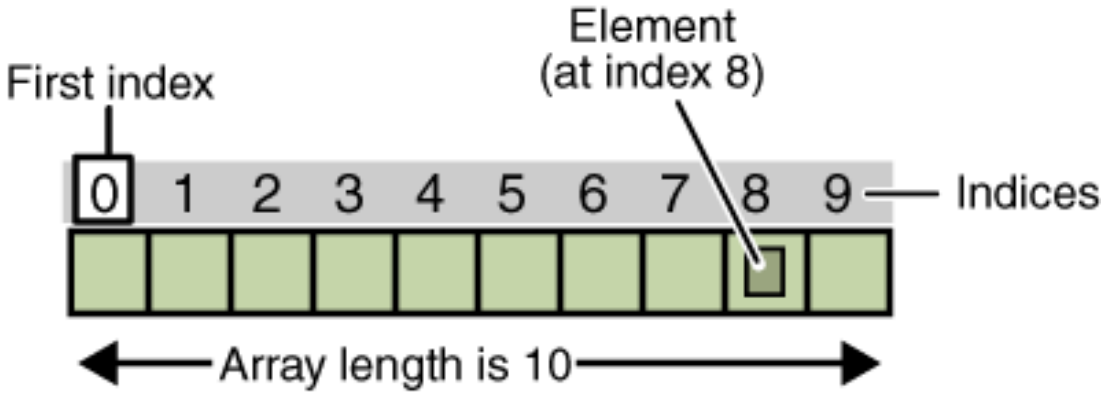
- In Computer Science, A list is:
  - An ordered collection of values (items, entries, elements… etc)
  - Where a value may occur more than once.
  - Example:
    1, 2, 3 and 3, 2, 1 are different lists

Basic List Behavior

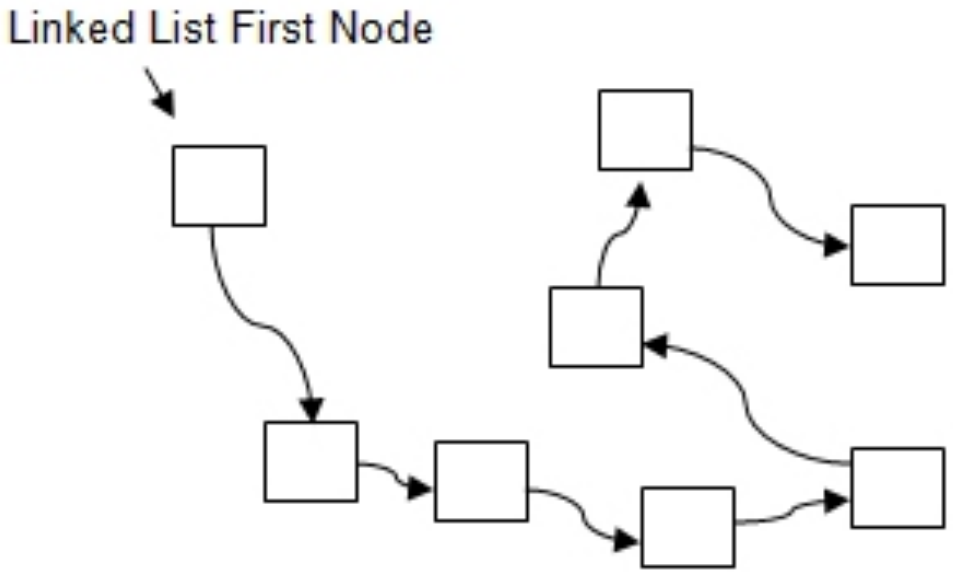
- Add an element
- Remove an element
- Find an element

How are Lists Implemented?

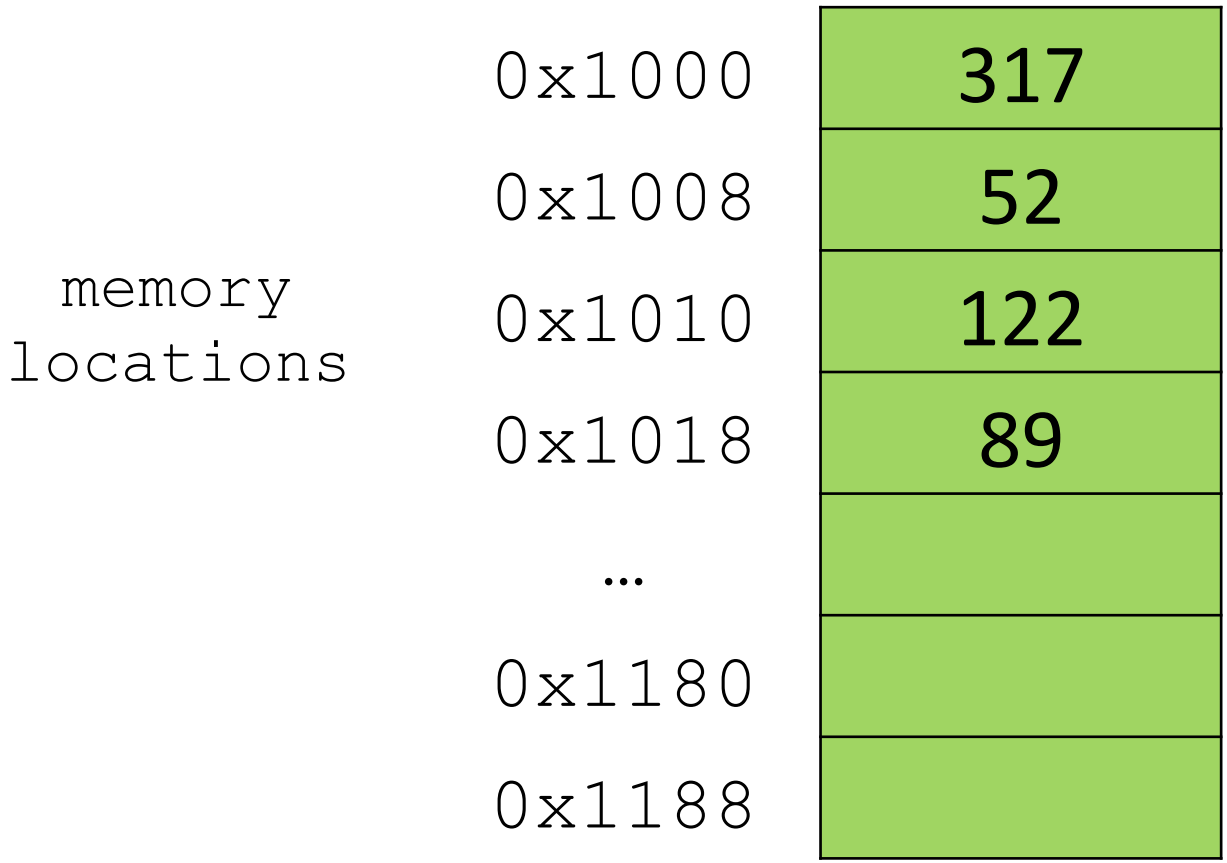
- Arrays



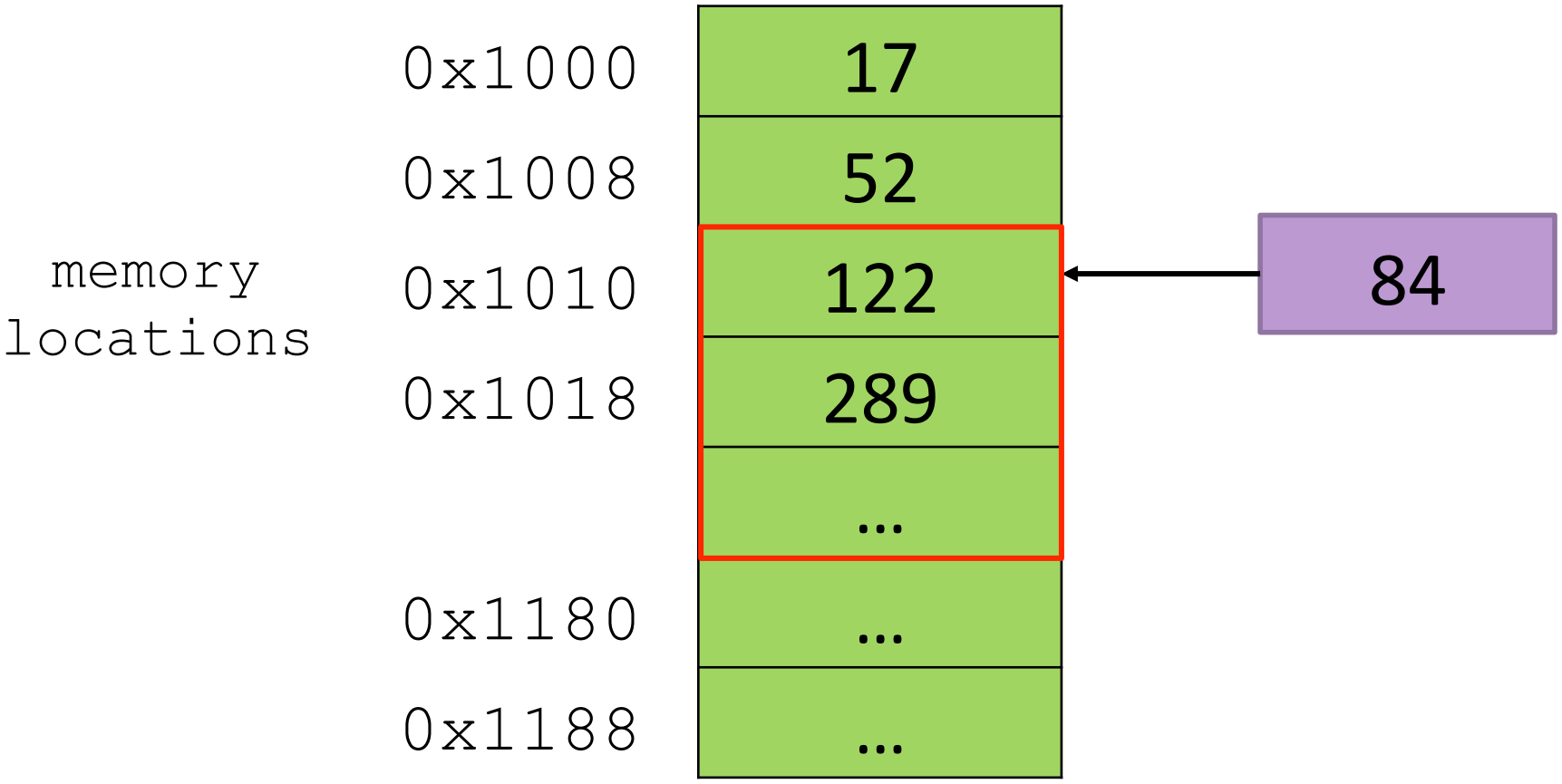
- Linked Lists



List Implementation - Array



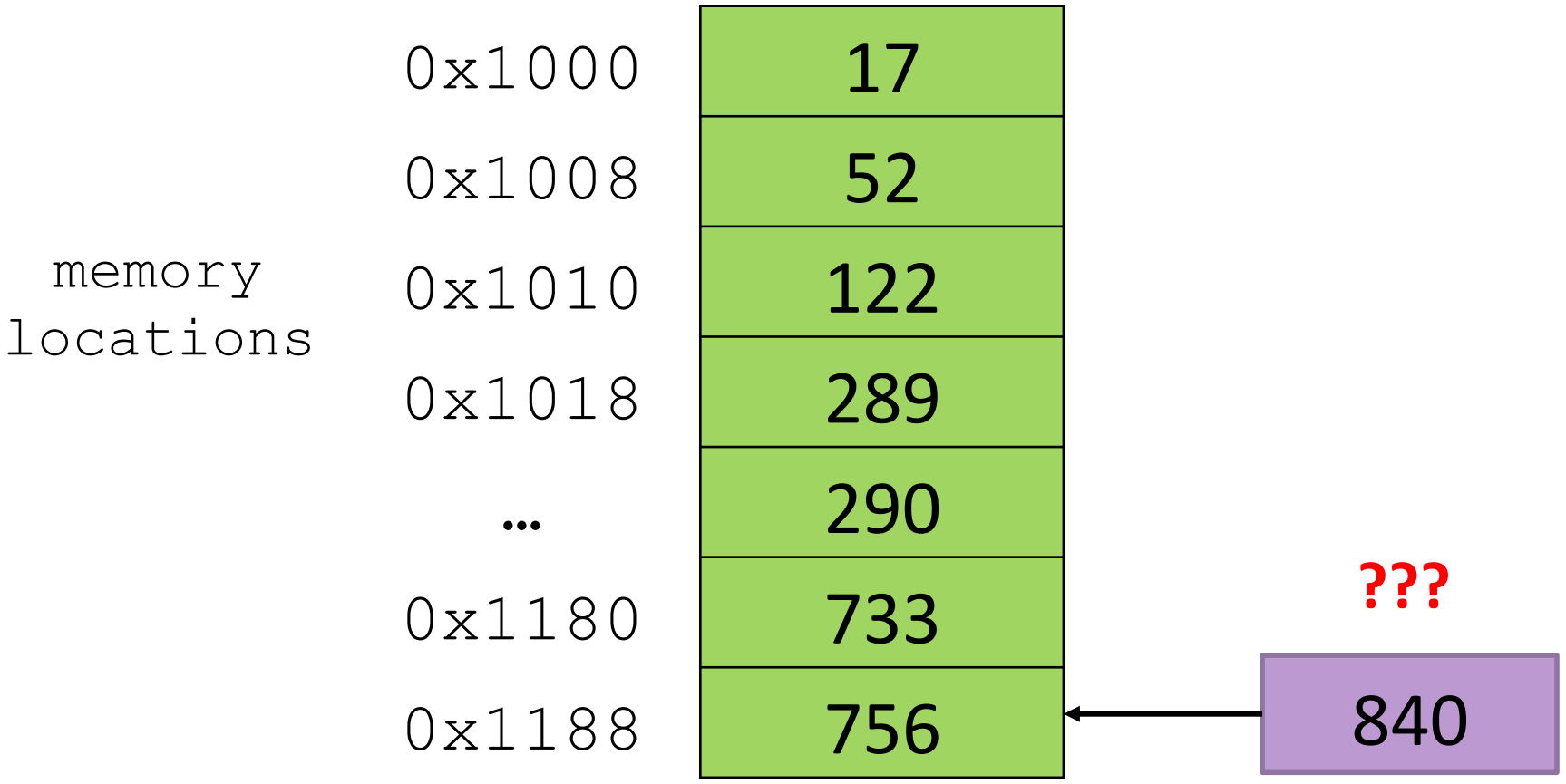
Arrays - Disadvantage I



"Add element" is an expensive operation

Arrays - Disadvantage II

Fixed size!

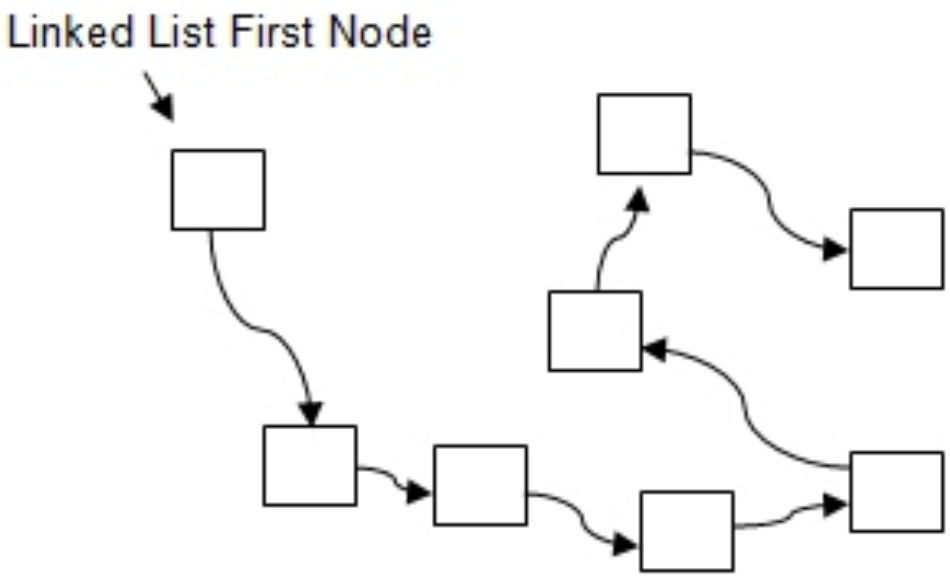


If you allocate too much, it's a waste of memory

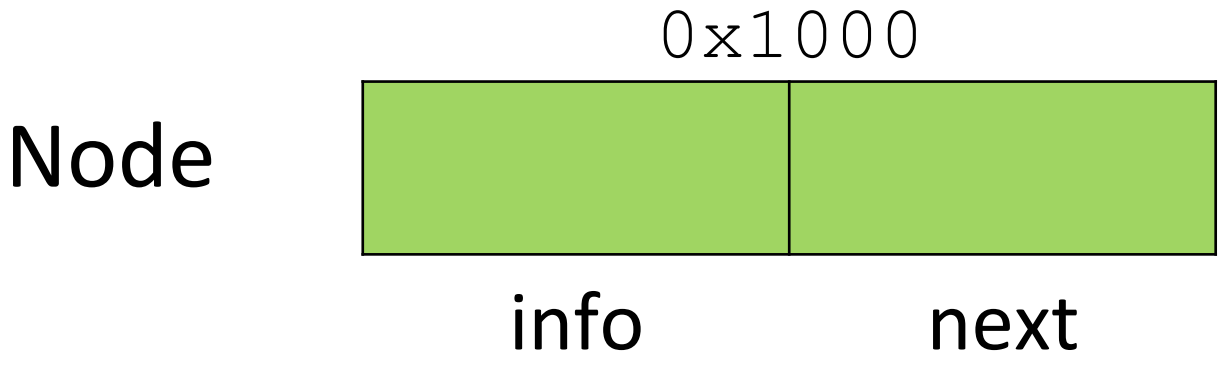
LINKED LISTS

What is a linked list?

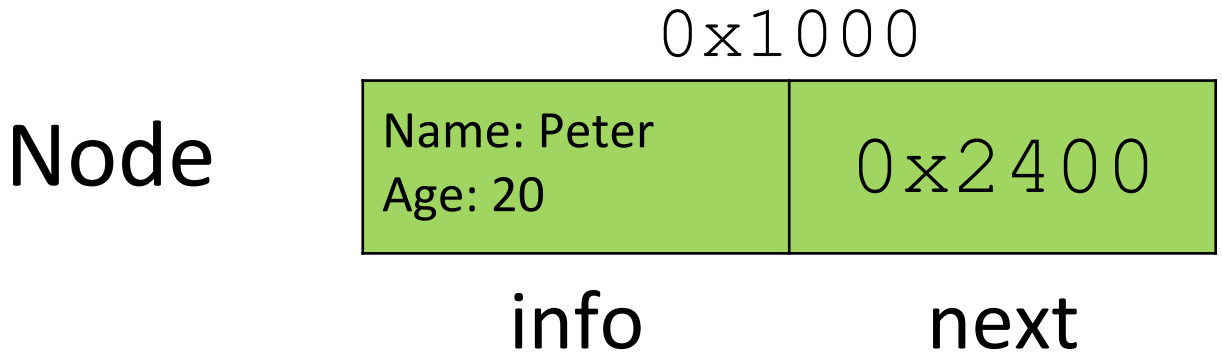
- A sequence of nodes such that each node contains a reference or a link to the next node.



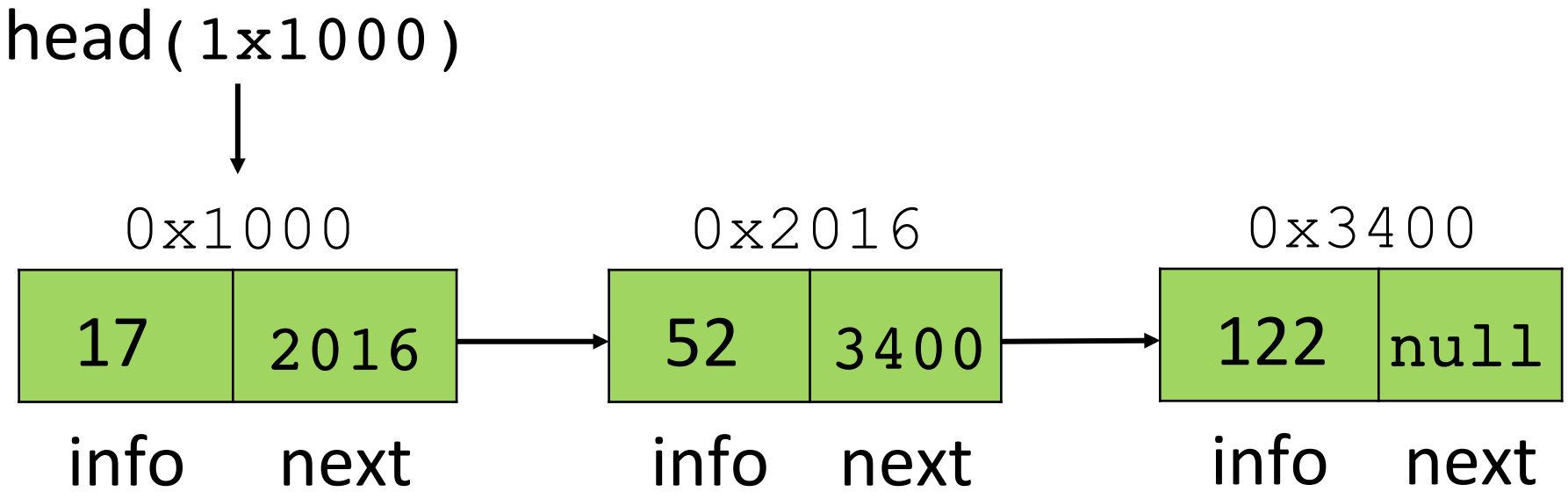
List Implementation – Linked List



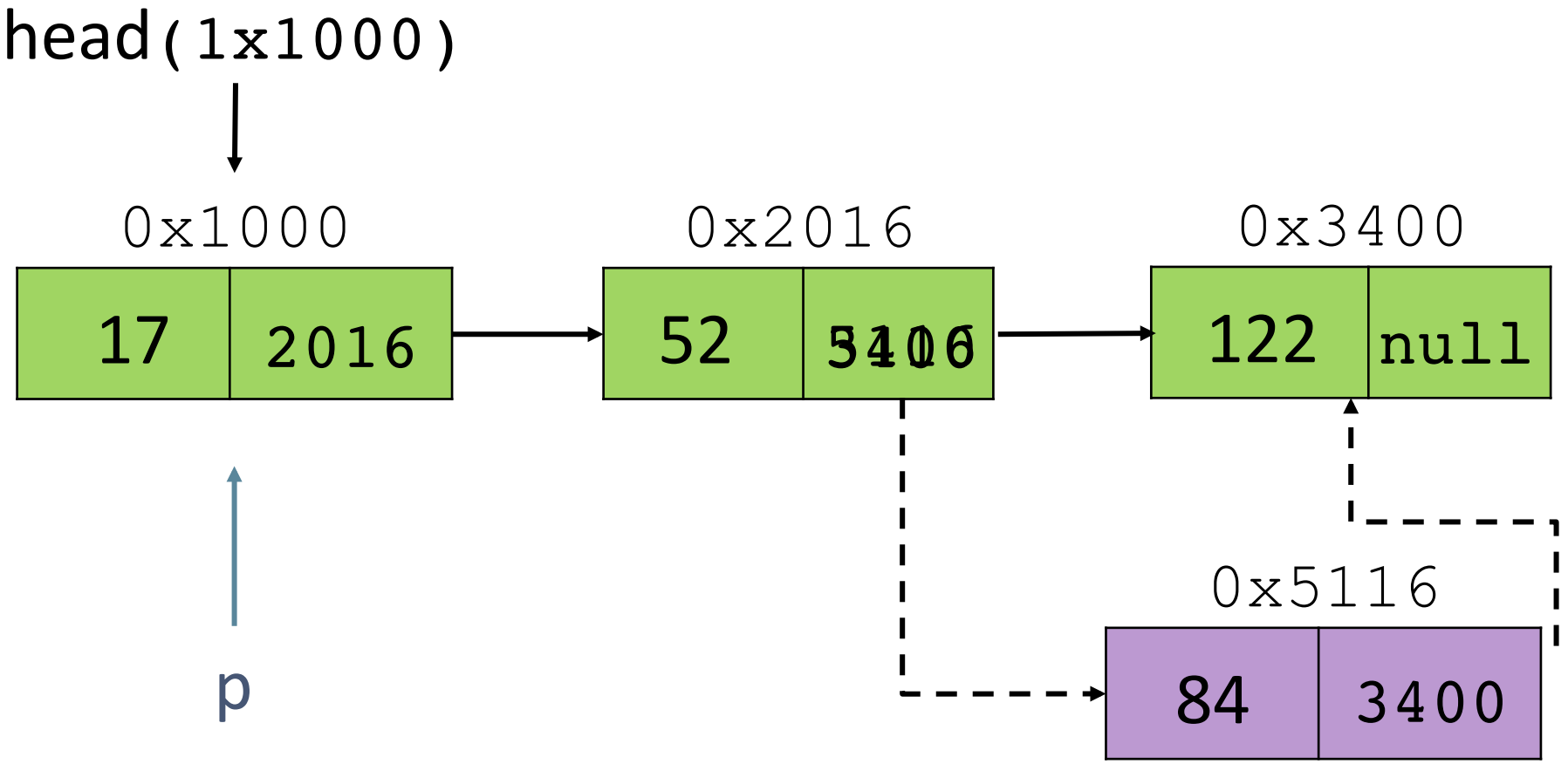
List Implementation – Linked List



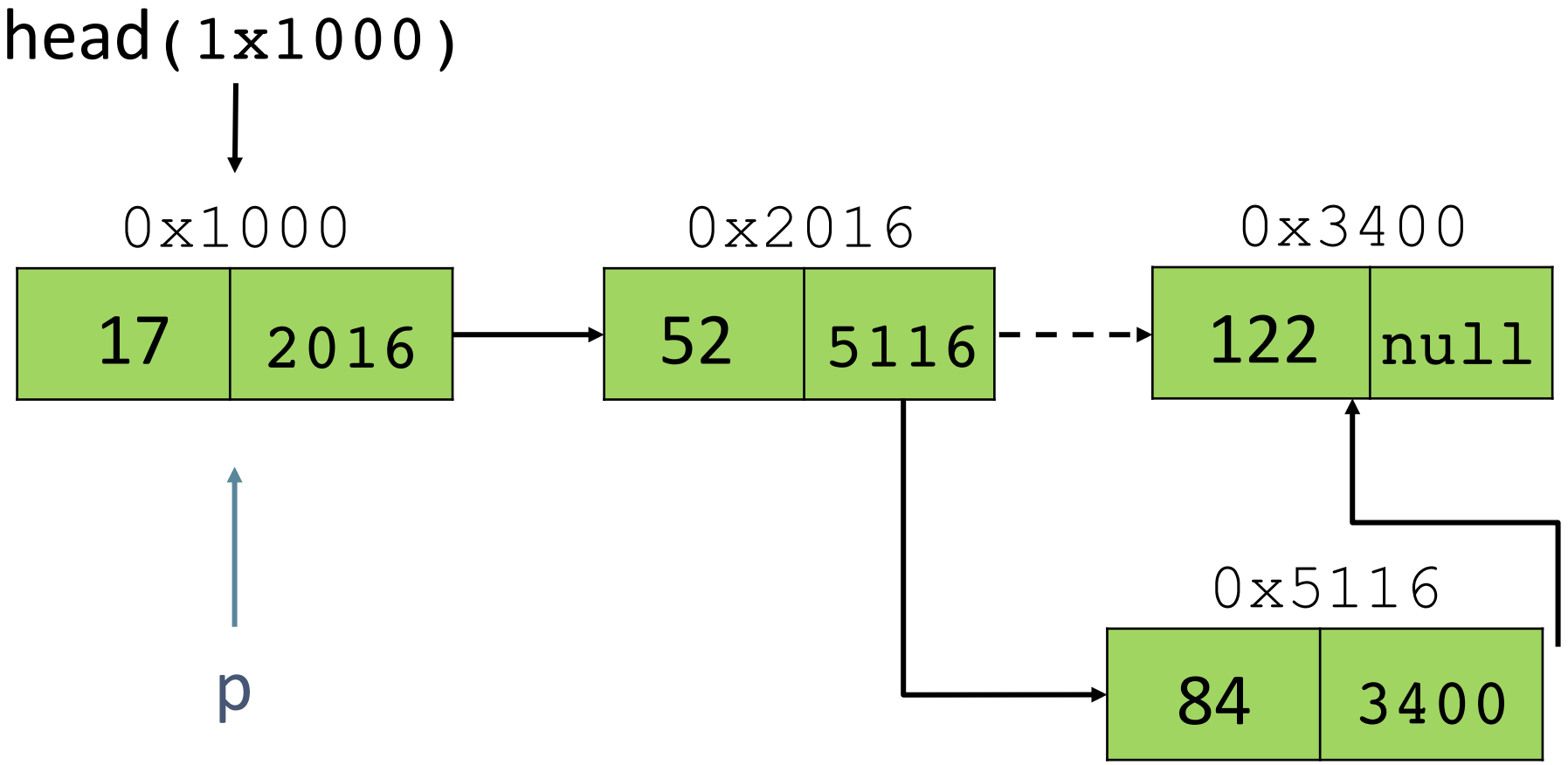
List Implementation – Linked List



Linked List – Add Element



Linked List – Delete Element

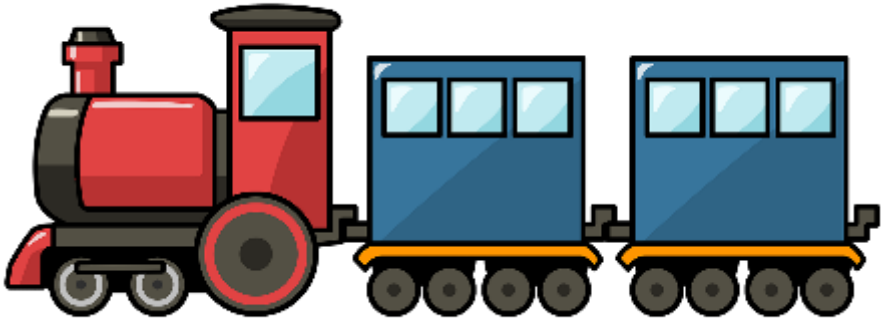


Linked List

- A dynamic data structure that grows or shrinks during the execution of a program.

- Efficient memory utilization:
  - Memory is allocated whenever it is required. And it is deallocated when it is not needed.

What is a linked list?

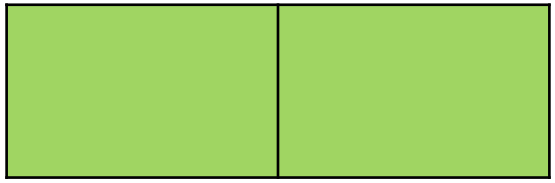


Operations Performed on Linked Lists

- Create the list, leaving it empty.
- Determine whether the list is empty or not.
- Determine whether the list is full or not.
- Insert a new entry into a specific location in the list
- Remove an entry from a specific location in the list
- Clear the list to make it empty

LINKED LISTS IMPLEMENTATION

Node Struct




```
typedef char entry_type;

// A linked list node
typedef struct n{
   entry_type info;
   struct n *next;
} node;

typedef node *list_type;
```

Linked List Implementation

- **Insert Operation**:

Pre: The list is initialized, not full and 0<=pos<=size of the list.

Post: Item is added to specific position of the list.

```
void  Insert(list_type *L, entry_type
                                item, int  pos){
   Node *p = (Node *)malloc(sizeof(Node));
   p->info = item;
   Node  *q;
   for(q=*L, int i=0; i<pos-1; i++)
        q=q->next;
   p->next=q->next;
   q->next=p;
}
```

Linked List Implementation

- **Insert operation:**

```
void  Insert(list_type *L, entry_type  item,
 int  pos){
    Node *p = (Node *)malloc(sizeof(Node));
    p->info = item;
    if (pos==0){   //will work also for empty list
         p->next=*L;
        *L = p;
    } else{   Node  *q;
            for(q=*L, int i=0; i<pos-1; i++)
                q=q->next;
            p->next=q->next;
            q->next=p;
    } }
```

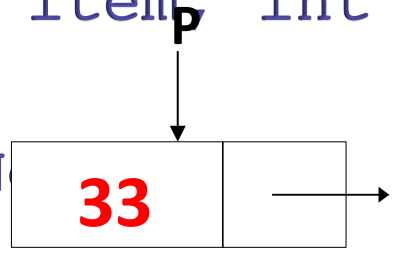
Linked List Implementation

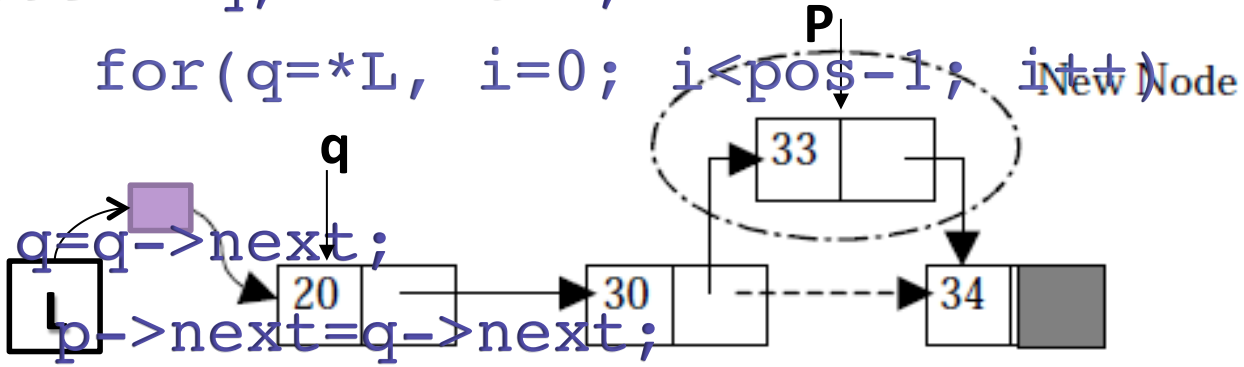
- **Insert operation:**

Pre: The list is initialized, not full and 0<=pos<=size of the list.

Post: Item is added to specific position of the list.

```
void  Insert(ListType *L, EntryType  item, int
 pos){
    Node *p = (Node *)malloc(sizeof(N
    p->info = item;
    if (pos==0){   //will work also for empty
 list
         p->next=*L          *L = p;   }
    else{   Node  *q;   int i;
              for(q=*L, i=0; i<pos-1; i++)
q=q->next;
p->next=q->next;
```





Linked List Implementation

- **Retrieve Operation:**

Pre: The list is initialized, not empty and 0<=pos<=size of the list.

Post: An element has been retrieved from position pos.

```
void  Retrieve(list_type *L, entry_type
                          *item, int pos){
     Node *q, *tmp;
     for(q=*L,  int i=0; i<pos-1; i++)
              q=q->next;
     *item=q->next->info;
     tmp = q->next;
     q->next = Tmp->next;
     free(tmp);
}
```

Linked List Implementation

- **Retrieve Operation:**

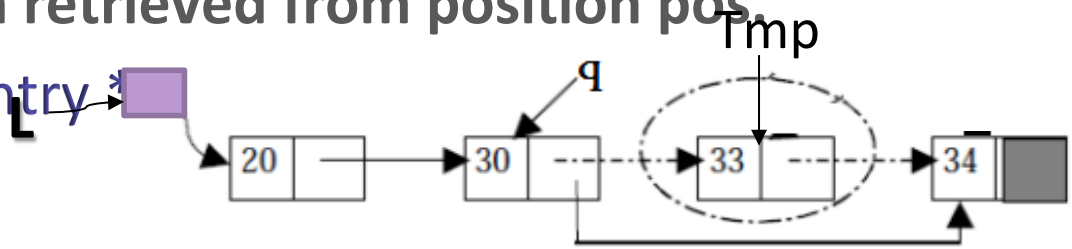
```
void  Retrieve(list_type *L, entry_type *item,
                                        int pos){
     Node *q, *tmp;
    if (pos==0){
          *item = *L->info;
          tmp=*L;
          *L=*L->next;
          free(tmp);
    } else{
        for(q=*L,  int i=0; i<pos-1; i++)
            q = q->next;
        *item = q->next->info;
         tmp = q->next;     q->next = tmp->next;
         free(tmp); }    }
```

Linked List Implementation

- **Retrieve operation:**

Pre: The list is initialized, not empty and 0<=pos<=size of the list.

Post: An element has been retrieved from position pos.




```
void  Retrieve(ListType *L, Entry *
        int i;    Node *q, *tmp;
       if (pos==0){
                    *item=*L->info;    tmp=*L;          *L=*L->next;
                    free(tmp);}          // it works also for one node
       else{
                  for(q=*L,  i=0; i<pos-1; i++)    q=q->next;
                  *item=q->next->info;
                  tmp=q->next;        q->next=Tmp->next;
                  free(tmp); }// check for retrieving last node
}
```

Linked List - Create

Pre: None.

Post: The list is initialized to be empty.

```
void create_list(list_type *l){
    *l= NULL;
}
```

Linked List - Empty

Pre: The list is initialized.

Post: If the list is empty (1) is returned.
Otherwise (0) is returned.

```
int empty_list(list_type   l){
   return (l==Null);
}
```

Linked List - full

Pre: The list is initialized.

Post: If the list is full (1) is returned.
Otherwise (0) is returned.

```
int full_list(lis_type  l){
   return 0;
}
```

Linked List - Clear

Pre: The list is initialized.

Post: the list is cleared to be empty.

```
void  clear_list(list_type  *l){
   node *q;
   while(*l){
        q = *l;
        *l=*l->next;
        free(q);
   }
```

Linked List - Drawbacks

- Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.

- Extra memory space for a pointer is required with each element of the list.

Think

- How to use the Linked List as a Linked Stack?!!
- How to use the Linked List as a Linked Queue?!!
- Could you keep track with the list size in the List ADT?!! How?!! is that useful?!!

CS 214 – DATA STRUCTURES FALL 2017

Lecture 6 – Linked Lists

Group **A & C**

October 23rd 2017

Dr. Mai Hamdalla

mai@fci.helwan.edu.eg