Cursor Implementation of Linked Lists

Dr. Abdallah Karakra

Computer Science Department

COMP242

Friday, March 22, 2024

Linked Lists: Cursor Implementation

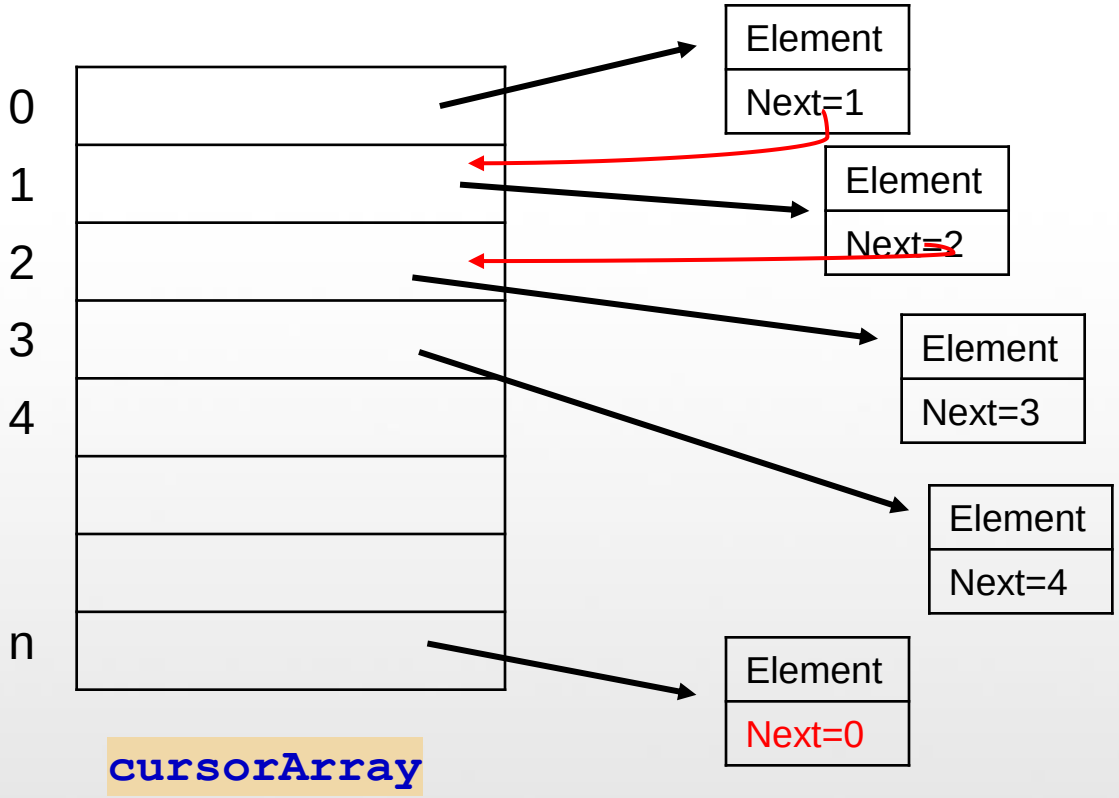
Many languages, such as Old BASIC and FORTRAN, do not support pointers, but we can simulate using cursors.

Linked Lists: Cursor Implementation

Implementation:

- Have a global array of structures(objects)
- Array index can be used in place of an address

Imagine (Not exactly)



Linked Lists: Cursor Implementation

Global array of structures (objects)

Address: Array index (slot)

```
private final static int SPACE_SIZE = 1000 ;
```

```
public class Node {

      public  Object element;
      public  int next;
      .
      .
}
```

slot	*Element*	*Next*
0	Null	1
1	Null	2
2	Null	3
3	Null	4
4	Null	5
5	Null	6
6	Null	7
7	Null	8
8	Null	9
9	Null	10
10	Null	0

Linked Lists: Cursor Implementation

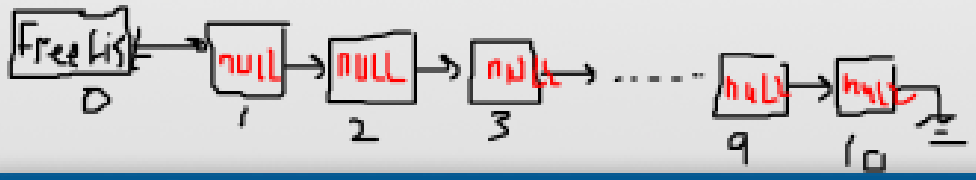
free list head

header

slot	Element	Next
0	Null	1
1	Null	2
2	Null	3
3	Null	4
4	Null	5
5	Null	6
6	Null	7
7	Null	8
8	Null	9
9	Null	10
10	Null	0

cursor space array:

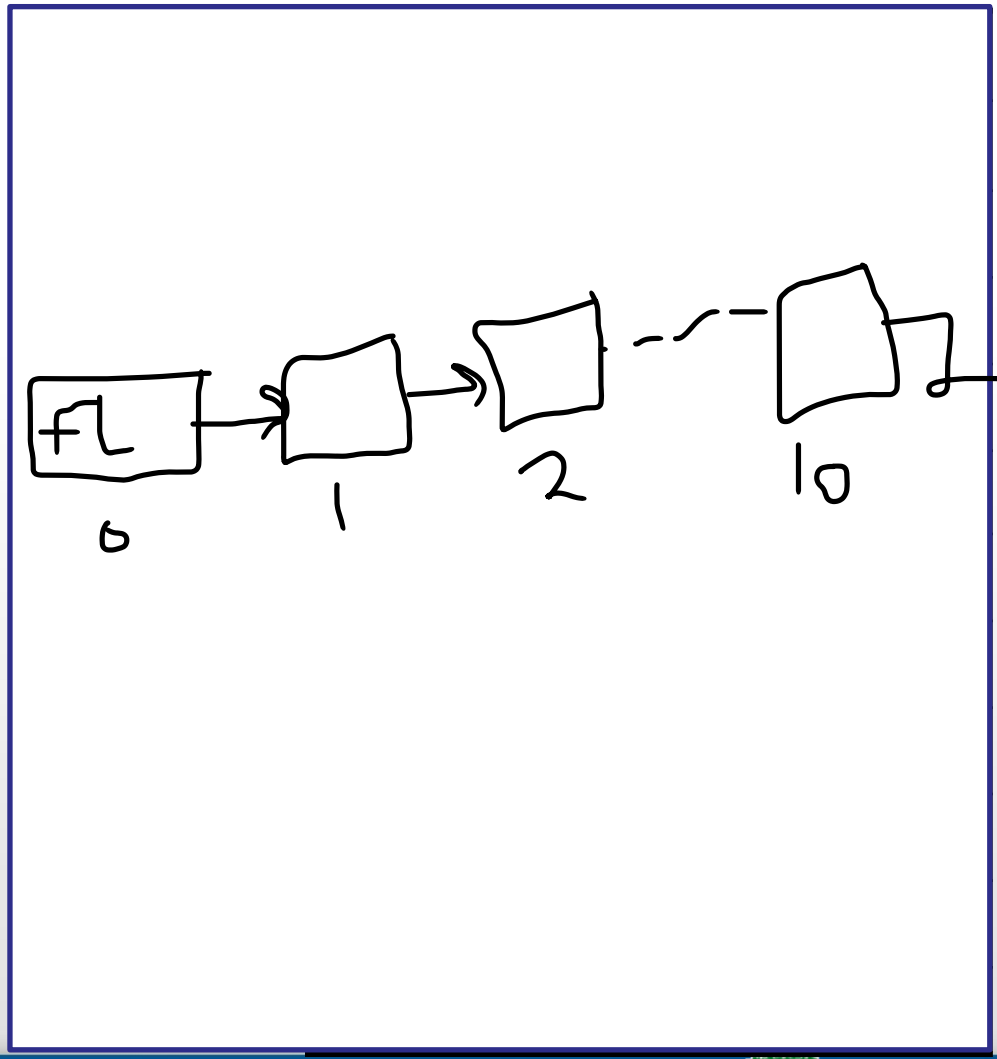
- Freelist: cells that **are not in any lists**
- Use **cell 0** as a header of **freelist**



Linked Lists: Cursor Implementation

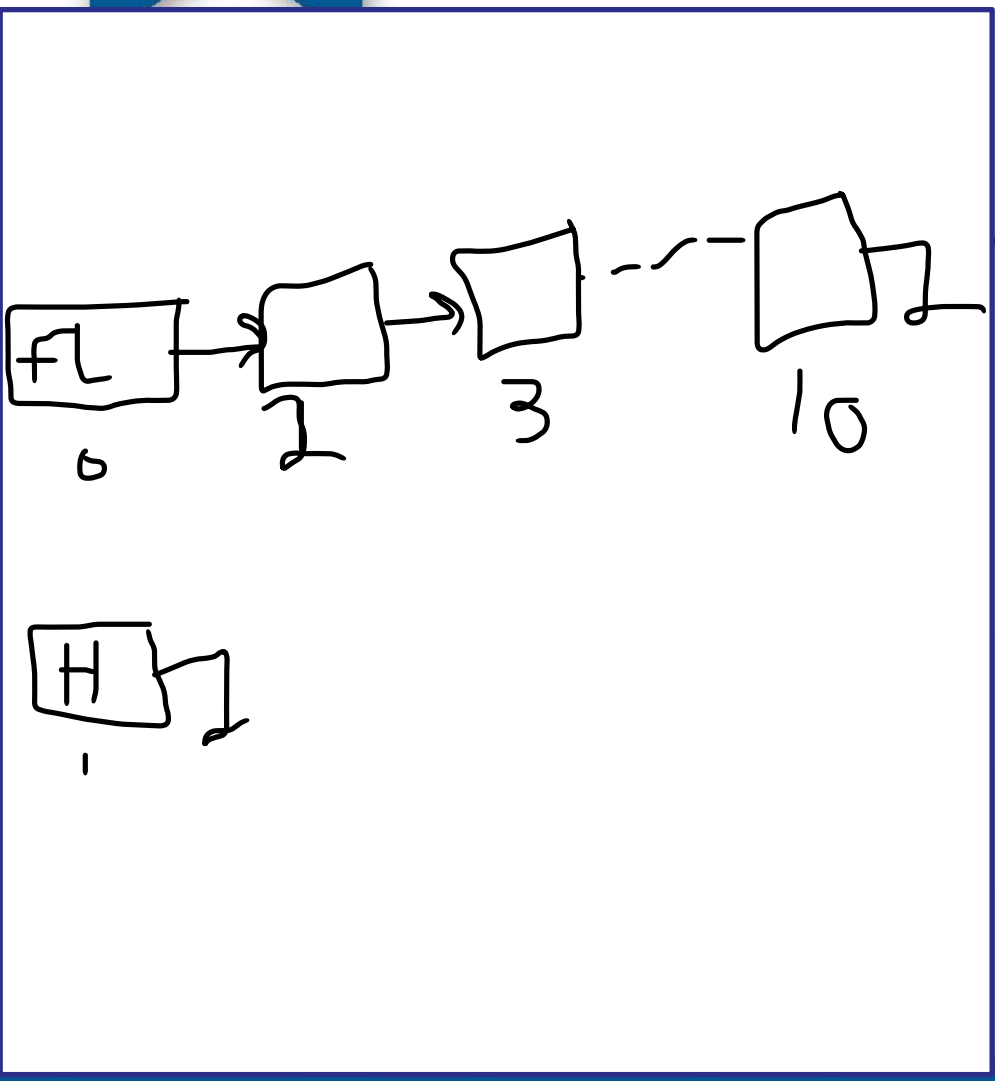
CursorAlloc to simulate malloc()

slot	Element	Next
0	freelist	1
1	Null	2
2	Null	3
3	Null	4
4	Null	5
5	Null	6
6	Null	7
7	Null	8
8	Null	9
9	Null	10
10	Null	0



Linked Lists: Cursor Implementation

CursorAlloc to simulate malloc()



slot	Element	Next
0	freelist	2
1	header	0
2	Null	3
3	Null	4
4	Null	5
5	Null	6
6	Null	7
7	Null	8
8	Null	9
9	Null	10
10	Null	0

Linked Lists: Cursor Implementation

CursorFree to simulate Free()

slot	Element	Next
0	freelist	4
1	header	2
2	c	3
3	b	5
4	Null	8
5	g	6
6	f	7
7	d	0
8	Null	9
9	Null	10
10	Null	0

Linked Lists: Cursor Implementation

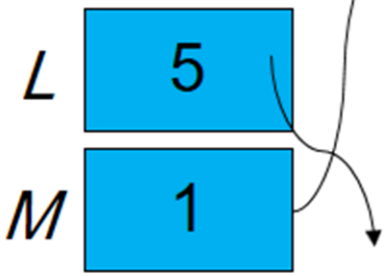
CursorFree to simulate Free()

slot	Element	Next
0	freelist	8
1	header	2
2	c	3
3	b	4
4	e	5
5	g	6
6	f	7
7	d	0
8	Null	9
9	Null	10
10	Null	0

Linked Lists: Cursor Implementation Further Example

L--> [8] B --> [3] C --> Null

M--> [7] E --> Null



	Element	next
1		7
2		4
3	C	0
4		6
5		8
6		0
7	E	0
8	B	3
9		10
10		3

Node implementation

```
public class Node {

     public  Object element;
     public  int next;


     public Node(Object element) {
        this(element, 0);
    }

      public Node(Object element, int next) {
        this.element = element;
        this.next = next;
    }

}
```

Cursor implementation

```
public class Cursor {

private Node[] cursorArray;
private final static int MAX_SIZE = 10; /* max array size of
                                           cursor */




/* Methods go here */

}
```

Initialize Array Cursor

```
/* Initialize the cursor array to null and next index */
public void initialization() {

    cursorArray = new Node[MAX_SIZE];

    for (int i = 0; i < MAX_SIZE; i++)
        cursorArray[i] = new Node(null, i + 1);

    cursorArray[MAX_SIZE - 1].next = 0;

}
```

index	element	next
0	null	1
1	null	2
2	null	3
3	null	4
4	null	5
5	null	6
6	null	7
7	null	8
8	null	9
9	null	0

WHY CURSOR ARRAY?

index	element	next
0	null	1
1	null	2
2	null	3
3	null	4
4	null	5
5	null	6
6	null	7
7	null	8
8	null	9
9	null	0

cursorAlloc

```
/* returns the first node after the header (next of the head) */
public int cursorAlloc() {

   int p = cursorArray[0].next;
   cursorArray[0].next = cursorArray[p].next;



   return p;// return the index of the available node (most likely empty node)
}
```

index	element	next
0	null	1
1	null	2
2	null	3
3	null	4
4	null	5
5	null	6
6	null	7
7	null	8

---

alloc: p=1

---

index	element	next
0	null	2
1	null	2
2	null	3
3	null	4
4	null	5
5	null	6
6	null	7
7	null	8

---

alloc: p=1
alloc: p=2

---

index	element	next
0	null	3
1	null	2
2	null	3
3	null	4
4	null	5
5	null	6
6	null	7
7	null	8

You can check if the returns P is 0 or not, if p is 0 mean “Out Of Memory”

WHY CURSOR ARRAY?

index	element	next
0	null	1
1	null	2
2	null	3
3	null	4
4	null	5
5	null	6
6	null	7
7	null	8
8	null	9
9	null	0

createList

```
public int createList(){
  /*create new  empty list*/
 int l= cursorAlloc();
 if (l==0)
     System.out.println ("Error:out of space");
 else

      cursorArray[l]=new Node("-",0);//Empty Linked List
return l; /*Head of the list*/

}
```

cursorFree

```
private void cursorFree(int p) {
```

```
-------------------
alloc: p=1
alloc: p=2
-------------------
index    element        next
 0        null           3
 1        null           2
 2        null           3
 3        null           4
 4        null           5
 5        null           6
 6        null           7
 7        null           8
 8        null           9
 9        null           0
```

cursorFree

```
private void cursorFree(int p) {

    cursorArray[p].element = null; // free the content
    cursorArray[p].next = cursorArray[0].next;
    cursorArray[0].next = p;

}
```


```
-------------------
alloc: p=1
alloc: p=2
-------------------
index    element        next
 0        null           3
 1        null           2
 2        null           3
 3        null           4
 4        null           5
 5        null           6
 6        null           7
 7        null           8
 8        null           9
 9        null           0
```

step2
step1

```
--------------------
free(1):
-------------------
index    element        next
 0        null           1
 1        null           3
 2        null           3
 3        null           4
 4        null           5
 5        null           6
 6        null           7
 7        null           8
 8        null           9
 9        null           0
```

Useful Functions

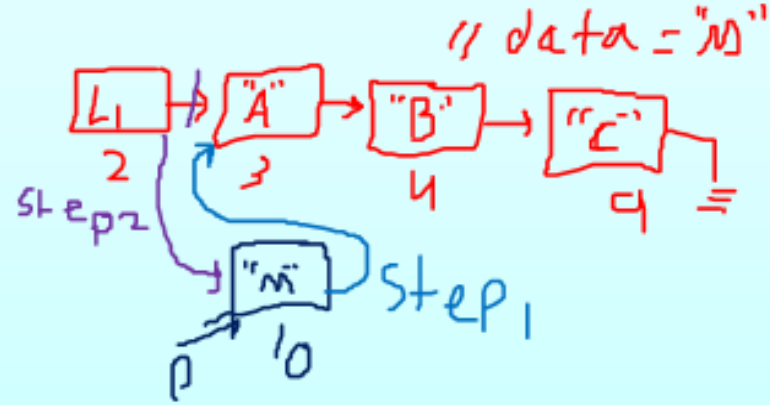
```
public boolean isNull (int l){

/*return true if the list not created*/
  return cursorArray[l]==null;
}

public boolean isEmpty (int l){
 //return true if the list is empty
  return cursorArray[l].next== 0;
}

public boolean isLast(int p){
 //check if the node p is last or not
   return cursorArray[p].next==0;

}
```

insertAtHead

```
public void insertAtHead (Object data,int l)
```

insertAtHead

```
public void insertAtHead (Object data,int l)
{
    if (isNull(l))  //list not created
      return ;

    int p=cursorAlloc();
    if (p!=0){
      cursorArray[p]=new Node (data,cursorArray[l].next);
      cursorArray[l].next=p;
    }
    else

    System.out.println("Out Of Space");

}
```



find

```
public int find (Object data,int l)  {

    int p = cursorArray[l].next;

    while((p != 0 )&& !cursorArray[p].element.equals(data))
        p = cursorArray[p].next;

    return p;

}
```

insert

```
public void insert( Object data,int l,int p ){// l: list, p:pos
```

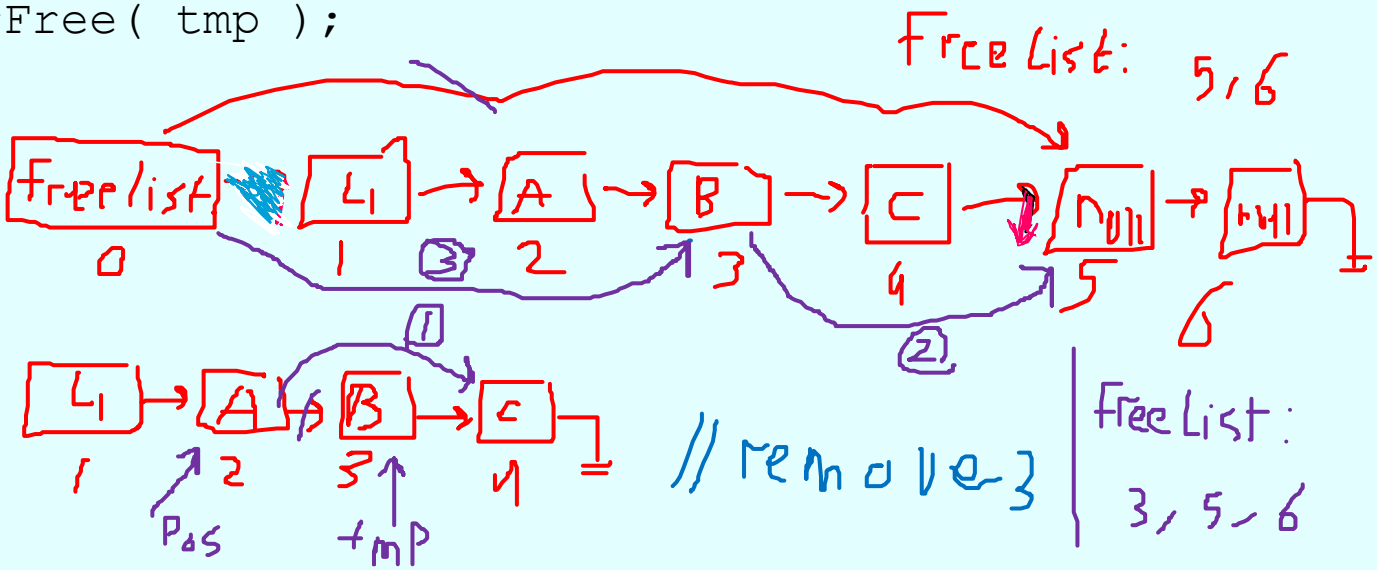
traversList (print list)

```
public void  traversList (int l){
```

remove

remove

```
public void remove(Object data,int l ) {

    int pos = findPrevious( data,l );//Implementation left as
                                     //an exercise

    if( cursorArray[pos].next != 0 ){//!isLast (pos)

        int tmp = cursorArray[pos].next;
        cursorArray[pos].next = cursorArray[tmp].next;
        cursorFree( tmp );

    }

}
```



Cursor Implementation of Linked Lists: H.W

You have to do the following:

```
/*Returns the previous location for a specific element*/

public int findPrevious (Object element, int l)
```

Question?

“Success is the sum of small efforts, repeated day in and day out.”
Robert Collier

Reference:

1. Dr. Iftikhar Azim Niaz Lecture Notes
2. Data Structures and Algorithm Analysis in C++, 2nd ed. by Mark Weiss.