Lebanese University
Faculty of Sciences 1

I2204E – INFO216 E

Imperative Programming

Final 2018-2019
Tue. January. 22nd pm
Duration: 2 hours

Exercise I: List and Files (15 points)

Consider the following declarations :

`typedef struct {`
`char name[20];`
`int age;`
`}element;`
	`typedef struct node {`
	`element data;`
	`struct node *next;`
	`} node;`

1. Write a function that creates a simply linked list of `elements` where names and ages are read from the keyboard.
   The user will provide an empty name to indicate that there are no more elements to register.
2. Write a function that creates a file (text or binary) and fils the `elements` of the linked list created above <u>in reverse order</u> in the file.

Exercise II: Sorting (10 points)

Write a function that sorts the distinct elements of a simply linked list of integers in such a way that:

- Odd elements are in the same order at the beginning of the list,
- Even elements are in reverse order at the end of the list.

<u>PS</u>: you may iterate only once on the list. The use of arrays is prohibited. No allocation or deallocation of memory is allowed.

<u>Example</u>: After sorting, the following list

0→ 5 →4 →9 →40 →1 →26 →7 →6 →8 →61→ 100→ 101

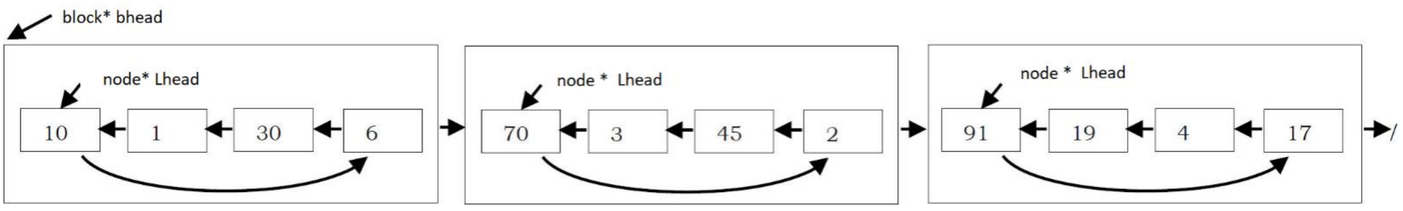
becomes

5 →9→ 1→ 7→61→ 101 →100 →8 →6 →26 →40 →4 →0

Exercise III: Unrolled linked list (28 points)

We consider in this exercise a simple variation of the singly linked list called *<u>unrolled linked lists</u>*.

An unrolled linked list stores multiple elements in each node (let us call it a **<u>`block`</u>** for our convenience). In each block, a circular linked list is used to connect all nodes.



We consider that there will be no more than *n* elements in the unrolled linked list at any time. To simplify this problem, all blocks, except the last one, should contain exactly $\lceil\sqrt{n}\rceil$ elements. Thus, there will be no more than $\lceil\sqrt{n}\rceil$ blocks at any time.

Consider the following declarations :

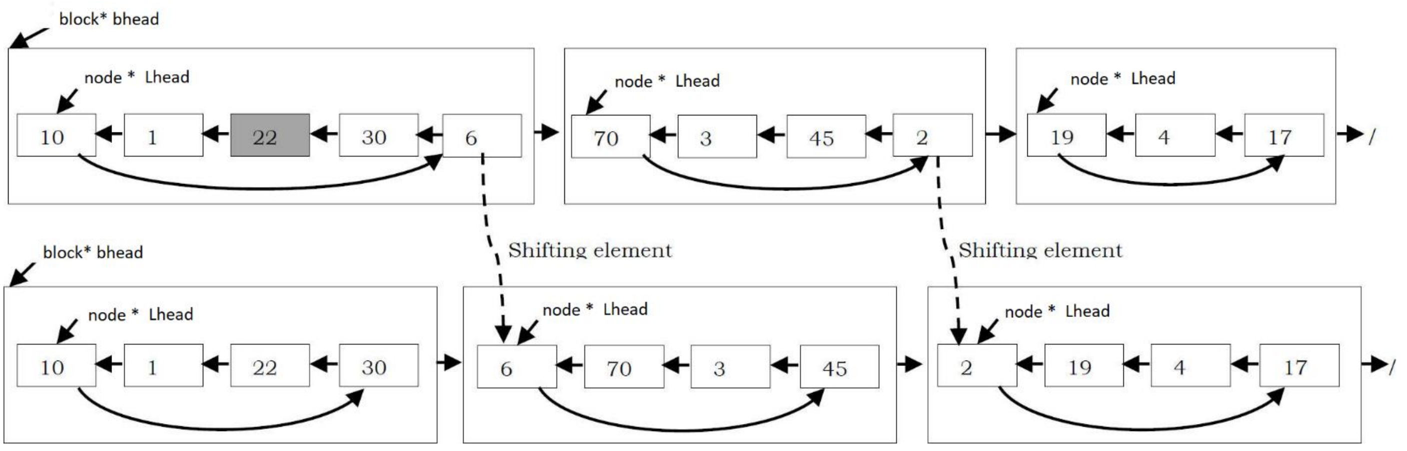
`typedef int element;`
`typedef struct node {`
`       element data;`
`       struct node* next;`
`} node;`
	`typedef struct block {`
	`      int nodeCount;`
	`      struct block* next;`
	`      node* Lhead;`
	`} block;`

To search for the *k*th element in an unrolled linked list:

a. traverse the list of blocks to the one that contains the *k*th node, i.e., the $\left\lceil\frac{k}{\sqrt{n}}\right\rceil$th block;
b. find the (*k* mod$\lceil\sqrt{n}\rceil$)th node in the circular linked list of this block.

1. Write a function that searches for the *k*th element in an unrolled linked list and returns a pointer to the block and a pointer to the node containing the *k*th element.
   `void searchKElement(block* bhead, int k, block** pblock, node** pnode);`

When inserting a node, we have to re-arrange the nodes in the unrolled linked list to maintain the properties previously mentioned, that each block contains $\lceil\sqrt{n}\rceil$ nodes. Suppose that we insert a node *x* after the *i*th node, and *x* should be placed in the *j*th block. Nodes in the *j*th block and in the blocks after the *j*th block have to be shifted toward the tail of the list so that each of them still have $\lceil\sqrt{n}\rceil$ nodes. In addition, a new block needs to be added to the tail if the last block of the list is out of space, i.e., it has more than $\lceil\sqrt{n}\rceil$ nodes.



Steps showing, adding the element 22 at position 3 and the shifting operation of nodes

2. Write the **shift** function, which given a block pointer, removes a node from the tail of the circular linked list in the block and inserts the node to the head of the circular linked list in the block after, and so on.
   Prototype: `void shift(block* bhead);`
   Attention: Besides the general case, consider when a new block needs to be added

3. Using above written functions, write a function that adds an element at the *k*th position in an unrolled linked list.
   Prototype: `void addElement(block** bhead, int k, element x);`
   Attention: Besides the general case, consider when the unrolled linked list is empty and when *k* is equal to 0.

Good luck!