

Data structure and algorithms lab

LINKED LIST

Lecturers : **Cao Tuan Dung**

**Dept of Software Engineering
Hanoi University of Science and
Technology**

Today's topics

- Introduction to Linked List
- Self referential structure in C
- Data structure "single linked LIST"
 - Implementation of single linked LIST
 - Algorithm for inserting, deleting, traversing, ...
- Data structure "double linked LIST"
 - Implementation of double linked LIST
 - Algorithm for inserting, deleting, traversing, ...

Towards Dynamic Data Structures

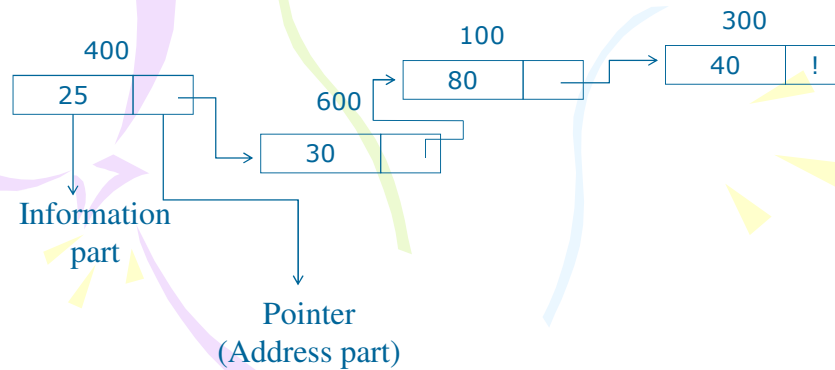
- ❑ Array is a collection of **homogeneous** elements which are stored at **consecutive** locations
- ❑ Main limitations of arrays:
 - It is a static data structure
 - Its size must be known at compilation time, in most programming languages
 - Inefficient insertion and deletion of elements
- ❑ A dynamic data structure can overcome these problems

What is a Dynamic Data Structure?

- ❑ A data structure that can shrink or grow during program execution
- ❑ The size of a dynamic data structure is not necessarily known at compilation time, in most programming languages
- ❑ Efficient insertion and deletion of elements
- ❑ The data in a dynamic data structure can be stored in non-contiguous (arbitrary) locations
- ❑ **Linked list** is an example of a dynamic data structure

What is a Linked List?

- ❑ A linked list is a collection of **nodes**, each node holding some **information** and a **pointer** to another node in the list
- ❑ In the following example, there are four nodes, which are not stored at consecutive locations

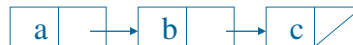


Self-Referential Structures

- One or more of its components is a pointer to itself.

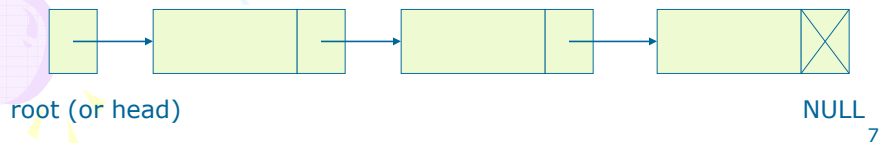
```

struct list {
    char data;
    struct list *link;
};
list item1, item2, item3;
item1.data='a';
item2.data='b';
item3.data='c';
item1.link=item2.link=item3.link=NULL;
  
```



Implemetation of List in C

- "LIST" means data structure that keeps the information of the location of next element generally.
- The elements of "Single linked LIST" have only next location.
- In C, the pointer is used for the location of the next element.
- Array: We can access any data immediately.
- Linked List: We can change the number of data in it.



Question 3-1

- We are now designing "address list" for mobile phones.
- You must declare a record structure that can keep a name, a phone number, and a e-mail address at least.
- And you must make the program which can deals with any number of the data

Hint

- you can organize elements and data structure using following record structure **node_addr**. Define by your self a structure for storing information about an address.

```
typedef struct address_t {
    char name[20];
    char tel[11];
    char email[25];
} address;
```

9

Declaration of address list

```
typedef struct address_t {
    char name[20];
    char tel[11];
    char email[25];
} address;

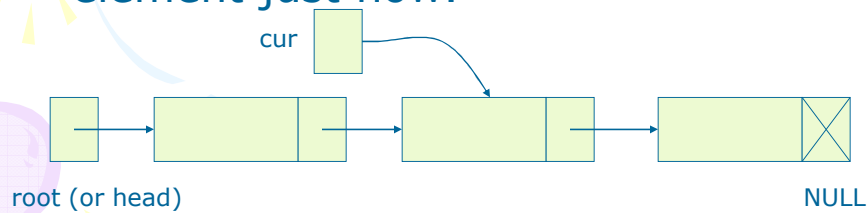
struct list_el {
    address addr;
    struct list_el *next;
};
typedef struct list_el node_addr;
```

- "next" is the pointer variable which can express the next element; an element of node_addr.
- "addr" is instance of an address.

10

Important 3 factors of a LIST

- Root: It keeps the head of the list.
- NULL: The value of pointer. It means the tail of the list.
- Cur: Pointer variable that keeps the element just now.



11

Initialisation

```
node_addr *root, *cur;
```

```
/* in case you used prev */
node_addr* prev;
```

12

Make new node

```
node_addr* makeNewNode() {
    node_addr* new = (node_addr*)
        malloc(sizeof(node_addr));
    strcpy((new->addr).name, « Tran Van Thanh »);
    ...
    new->next = NULL;
    return new;
}
..
root = makeNewNode();
cur = root;
```

13

Attention

- You can modify the makeNewNode function to receive the data field as parameter:

```
node_addr* makeNewNode(address addr) {
    node_addr* new = (node_addr*)
        malloc(sizeof(node_addr));
    new->addr=addr;
    new->next = NULL;
    return new;
}
```

14

Input Data for Node

```
address readNode() {  
    address tmp;  
    printf("Nhap ten:");  
    gets(tmp.name);  
    ...  
    return tmp;  
}
```

15

Display node's information

- Write the function displaying the data inside a give node pointed by p.

```
void displayNode(node_addr* p){  
    /* display name, tel, email in columns */  
  
}
```

16

Solution

```

void displayNode(node_addr* p){
    if (p==NULL){printf("Loi con tro NULL\n");
        return; }
    address tmp = p->addr;
    printf("%-20s\t%-15s\t%-30s %p\n", tmp.name,
        tmp.tel, tmp.email, p->next);
    }
void main(){
    /* root = makeNewNode(); */
    address tmp = readNode();
    root = makeNewNode(tmp);
    displayNode(root);
}

```

17

Exercise

- Create a singly linked list to store a list of phone address.
- Write a function to insert to a list a new element just after the current element and use it to add node to the list
- Write a function for traversing the list to print out all information stored.
- Write a function for the removal of a node in the list.

18

Insert node at head of the list

```
void insertAtHead(address addr){
    node_addr* new = makeNewNode(addr);
    new->next = root;
    root = new;
    cur = root;
}
```

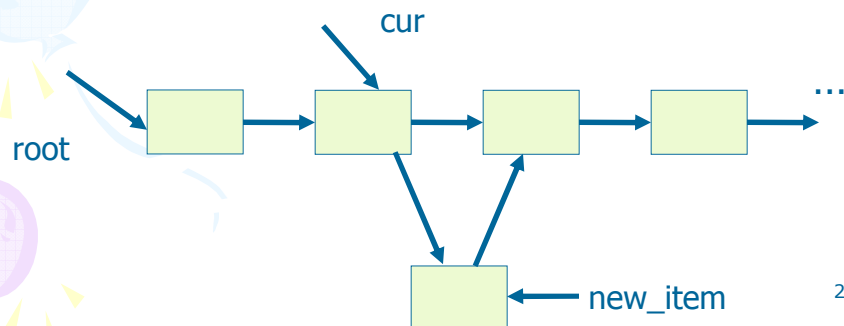
```
void main(){
    address tmp = readNode();
    insertAtHead(tmp);
    displayNode(root);
}
```

19

Link list: insertion after the current position

- Pseudo code

```
create new_item
new->next = cur->next;
cur->next = new;
cur = cur->next;
```



20

insertion just after the current position

```

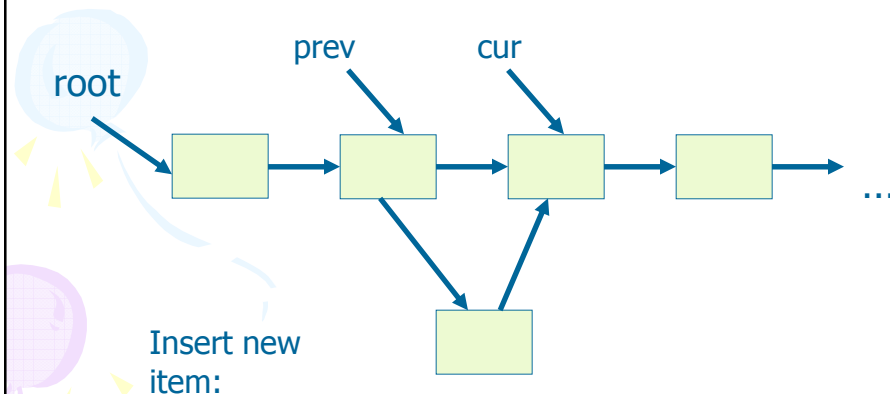
/* input of a address struct variable   addr */
...
new = makeNewNode(addr);
if (cur == NULL) return;
if ( root == NULL ) {
/* if there is no element */
root = new;
cur = root;
} else {
    new->next=cur->next;
    cur->next = new;
    /* prev=cur; */
    cur = cur->next;
}

```

21

Insertion before current position

- Another case: before the current position



22

insertBeforeCurrent

```
void insertBeforeCurrent(address e) {
    node_addr * new = makeNewNode(e);
    if ( root == NULL ) {
        /* if there is no element */
        root = new;
        cur = root;
        prev = NULL;
    } else {
        // Fill In the code here
        ...
    }
}
```

23

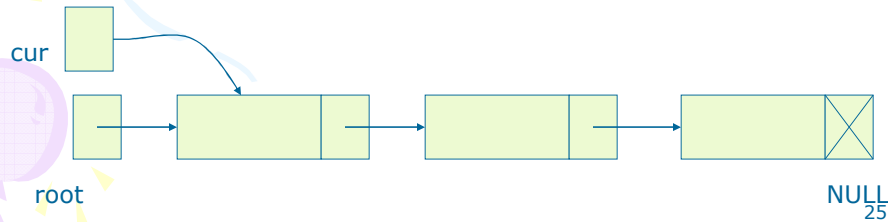
If you do not frequently
update pointer prev

```
/* determine prev if cur does
not point to first element */
tmp = root;
while (tmp!=NULL && tmp-
>next!=cur && cur !=NULL)
    tmp=tmp->next;
prev = tmp;
```

24

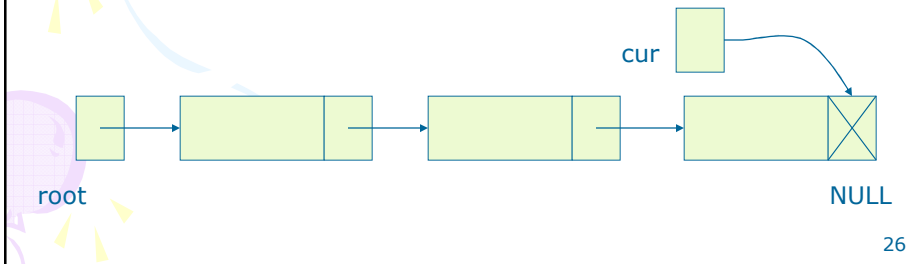
Traversing a list

```
void traversingList() {
    node_addr * p;
    for ( p = root; p != NULL; p = p->next )
        displayNode(p);
}
```



Traversing a list

- Changing the value of pointer variable cur in sequence.
- These variables are called "iterator."
- The traversing is finished if the value is NULL



new test scenario

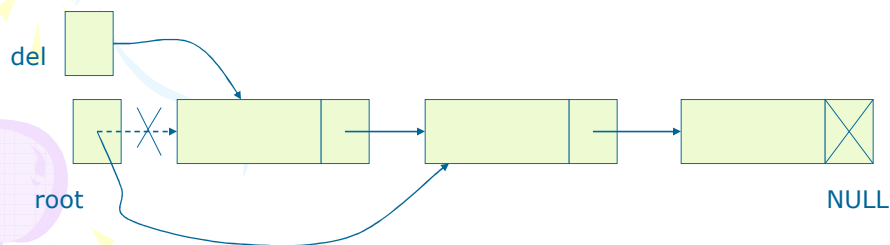
- Using a loop to input data to Linked List then display the whole list.

```
void main() {
    n=5;
    while (n) {
        address tmp = readNode();
        insertAtHead(tmp);
        n--;
    }
    traversingList();
}
```

27

Deletion

- When we remove the first element
root = del->next; free(del);
- When we remove the first element, change the value of "root" into the value of "next" which is pointed by "del."



28

A decorative graphic on the left side of the slide featuring three balloons in green, blue, and purple, each with yellow streamers and small yellow triangles representing sunbeams or confetti.

Delete first element

```
void deleteFirstElement() {  
    /* do it your self */  
}
```

29

A decorative graphic on the left side of the slide featuring three balloons in green, blue, and purple, each with yellow streamers and small yellow triangles representing sunbeams or confetti.

Delete first element of the list

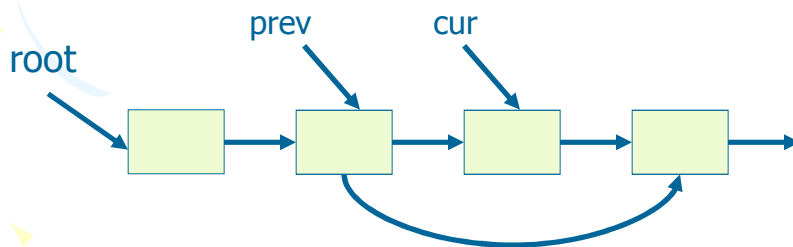
```
void deleteFirstElement(){  
    node_addr* del = root;  
    if (del == NULL) return;  
    root = del->next;  
    free(del);  
    cur = root; /* prev = NULL; */  
}
```

30

Deletion from the middle

- We want to remove the node pointed by cur
- Determine prev which point to the node just before the node to delete

```
prev->next = cur->next;
free(cur);
cur = prev->next;
```



31

Deletion from the middle

- Design and implement of deleteCurrentElement function

```
/* Do it your self
```

```
*/
```

32

Other useful function for deleting node

- Delete the first node corresponding to an address.
- `void deleteElement(address adr);`

33

Freeing a list

```
to_free = root ;  
while (to_free != NULL)  
{  
    root = root->next;  
    free(to_free);  
    to_free = root;  
}
```

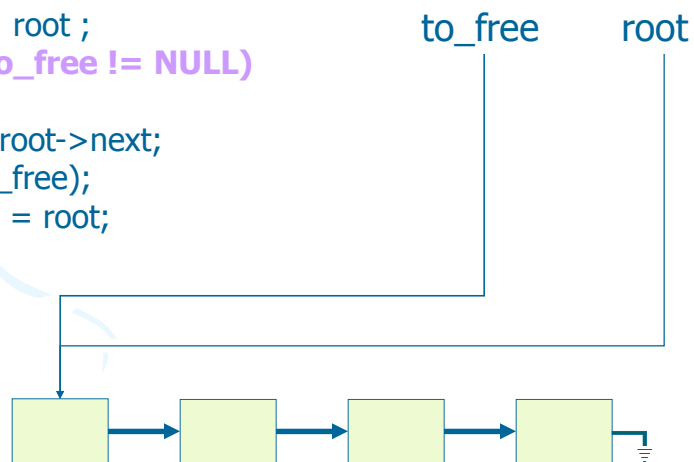
to_free root



34

Freeing all nodes of a list

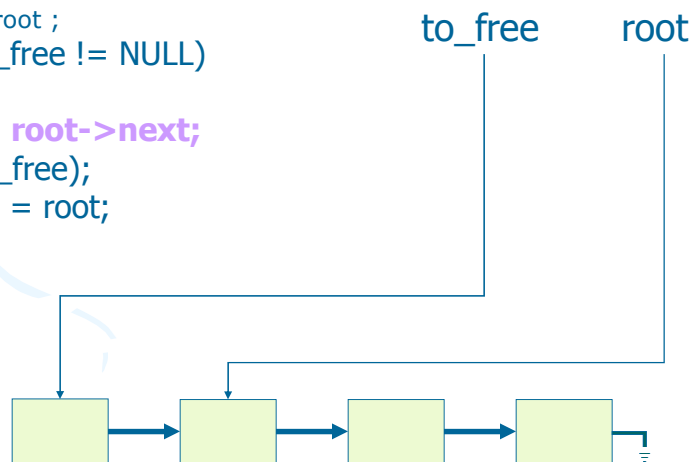
```
to_free = root ;
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



35

Freeing all nodes of a list

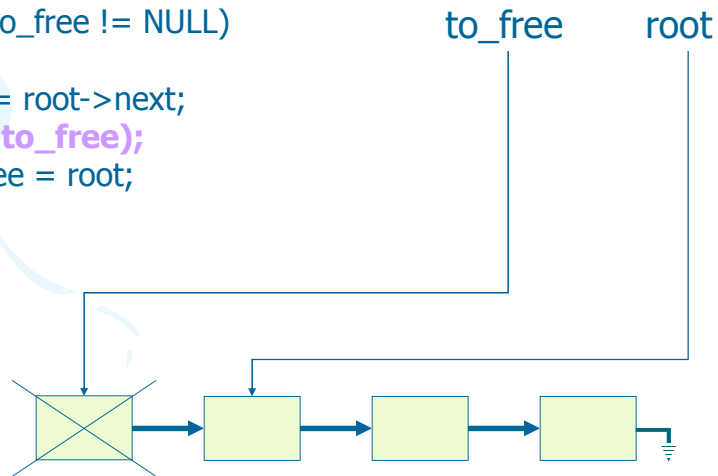
```
to_free = root ;
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



36

Freeing all nodes of a list

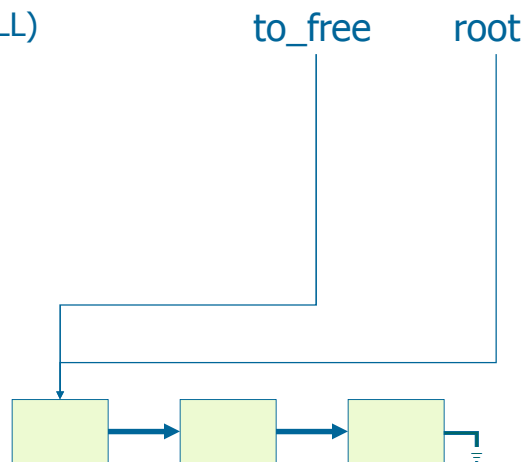
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



37

Freeing all nodes of a list

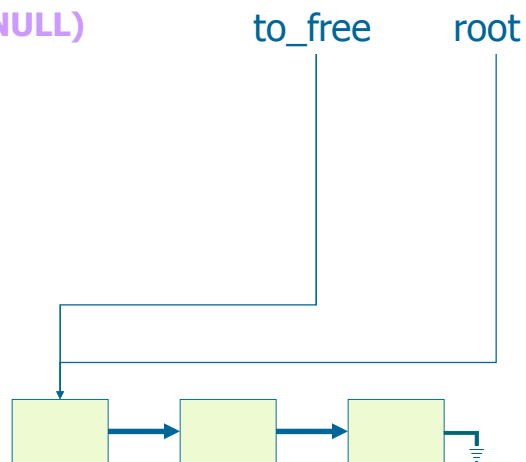
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



38

Freeing all nodes of a list

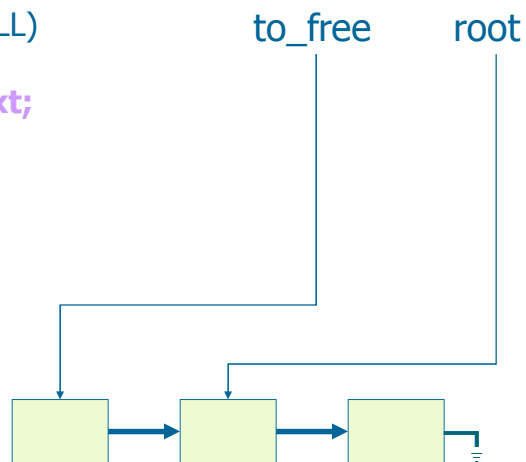
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



39

Freeing all nodes of a list

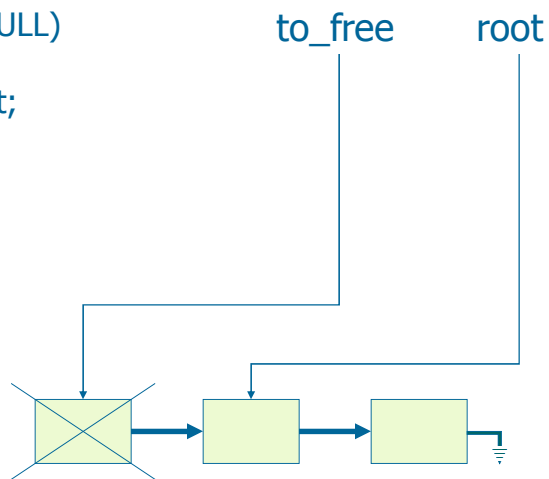
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



40

Freeing all nodes of a list

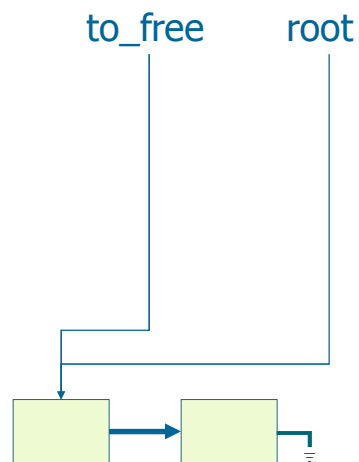
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



41

Freeing all nodes of a list

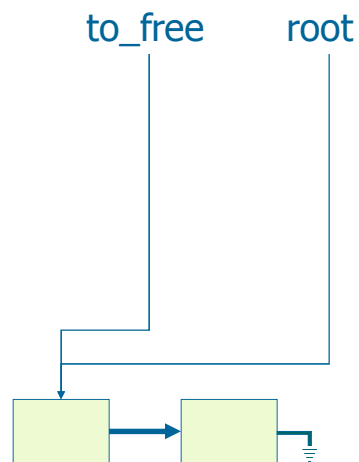
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



42

Freeing all nodes of a list

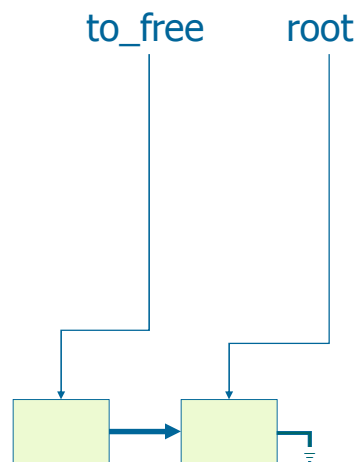
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



43

Freeing all nodes of a list

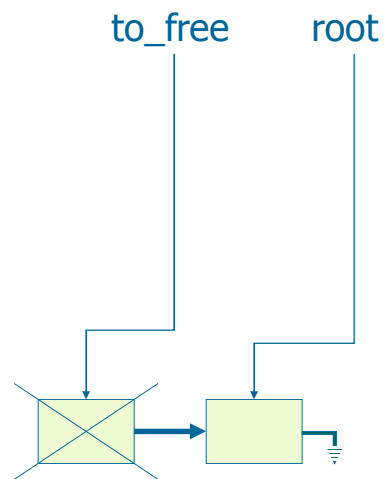
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



44

Freeing all nodes of a list

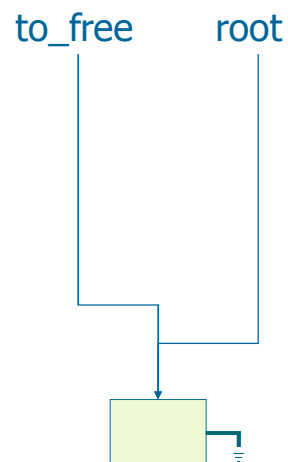
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



45

Freeing all nodes of a list

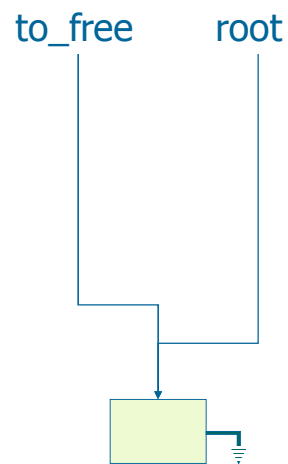
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



46

Freeing all nodes of a list

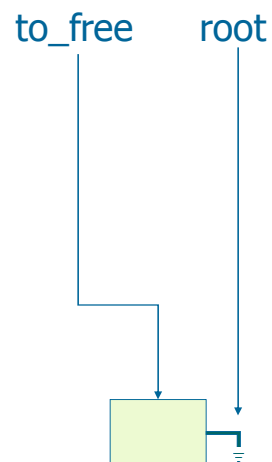
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



47

Freeing all nodes of a list

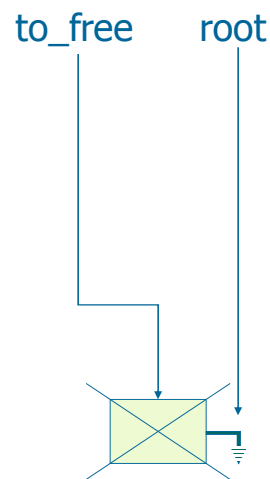
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



48

Freeing all nodes of a list

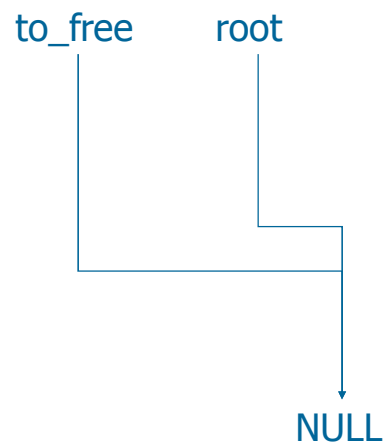
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



49

Freeing all nodes of a list

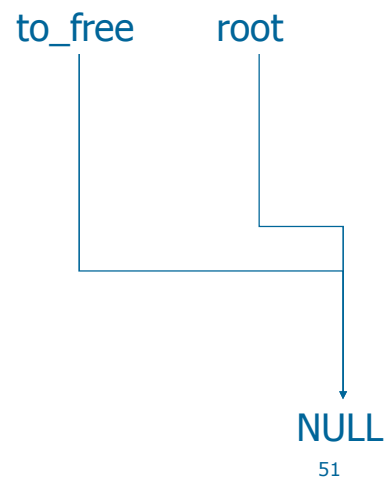
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



50

Freeing all nodes of a list

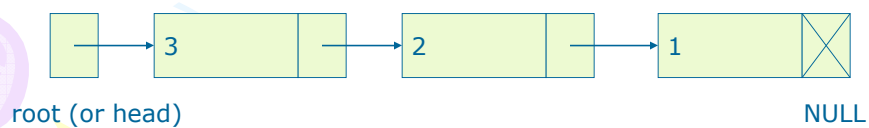
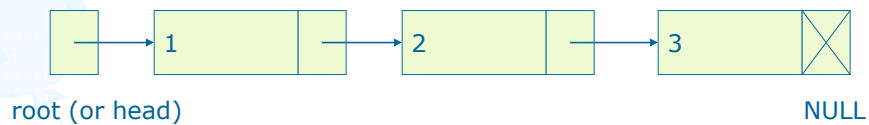
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



51

Reverse a list

- Write a function that reverse a list.



52

Solution

```
node_addr* list_reverse (node_addr* root)
{
    node_addr *cur, *prev;
    cur = prev = NULL;
    while (root != NULL) {
        cur = root;
        root = root->next;
        cur->next = prev;
        prev = cur;
    }
    return prev;
}
```

53

Exercise

- **Write a program that reads data from file phone.dat (created in previous assignment) and load them to a single linked list.**
- **You must use functions in your linked list library**
- **Display the contacts stored in list.**
- **Ask user to input more contacts and insert them to list in two ways:**
 - Insert at Head
 - or after current position of cur pointer.
- **Display the list again to verify.**

54

Exercise II

- Add to program in exercise I two functionalities: insert and delete element at given position.
 - You must use two functions in your library
- Implement a searching function by
 - Phone number
 - Name
- Test inverseList function.

55

Exercise III

- a) Write and test splitList function
 - Divide list in to 2 sub-lists.
 - Syntax split n1 n2: n1: start position (indexed from 0) – n2 number of element of sublist 1. The rest is the sublist 2
- b) Write a function that print the content of a list to a text file. Parameters are root pointer and file path. Use this function to view the sublists.
- c) Test data: Phone.dat

56

Homework

- **Make a improved version of PhoneDB Phone management program using linked list. Here is the functionalities in the menu:**
 - 1. **Import from Text:** read data from text file and build the list (using InsertAtHead)
 - 2. **Import from Dat:** read data from .dat file and build the list (using InsertAfterCurrentPos)
 - 3. **Display List:** Display all elements, each element in a line.
 - 4. **Search phone by Model**
 - 5. **Search phone of which the price is under the value inputted.**
 - 6. **Export to Dat:** store information in linked list to PhoneDB.dat
 - 7. **Manual Insertion (Add data for a phone model).** Program should ask the insertion mode: before or after current position.
 - 8. **Quit**

57

Homework

- **Continuing with the phone book management exercise, add the following functions that:**
 1. delete the entire list
 2. insert an element before the Cur(rent) element
 3. Find an element in the list by phone number.
 4. Save the entire list elements in a text file (phonebook.txt) or binary file (phonebook.dat) based on the filetype parameter.

58



Summary of the functionalities of the PhoneDB exercises (Single LL & double LL)

- **1. Import from PhoneDB.dat (insertafter)**
- **2. Display (traverse)**
- **3. Add new phone (insertbefore/after)**
- **4. Insert at Position**
- **5. Delete at Position**
- **6. Delete current**
- **7. Delete first**
- **8. Search and Update: (Search by model – update all field of information)**
- **9. Divide and Extract (split): Output is the content of two sublists.**
- **10. Reverse List**
- **11. Save to File**
- **12. Quit(Free)**

59