HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

LINKED LIST (PART I)

IT3230E Data structure and algorithms Lab

ONE LOVE. ONE FUTURE.

Outline

- **Introduction to linked list**
- Implementation of singly linked list data structure
- Using list in specific problems

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

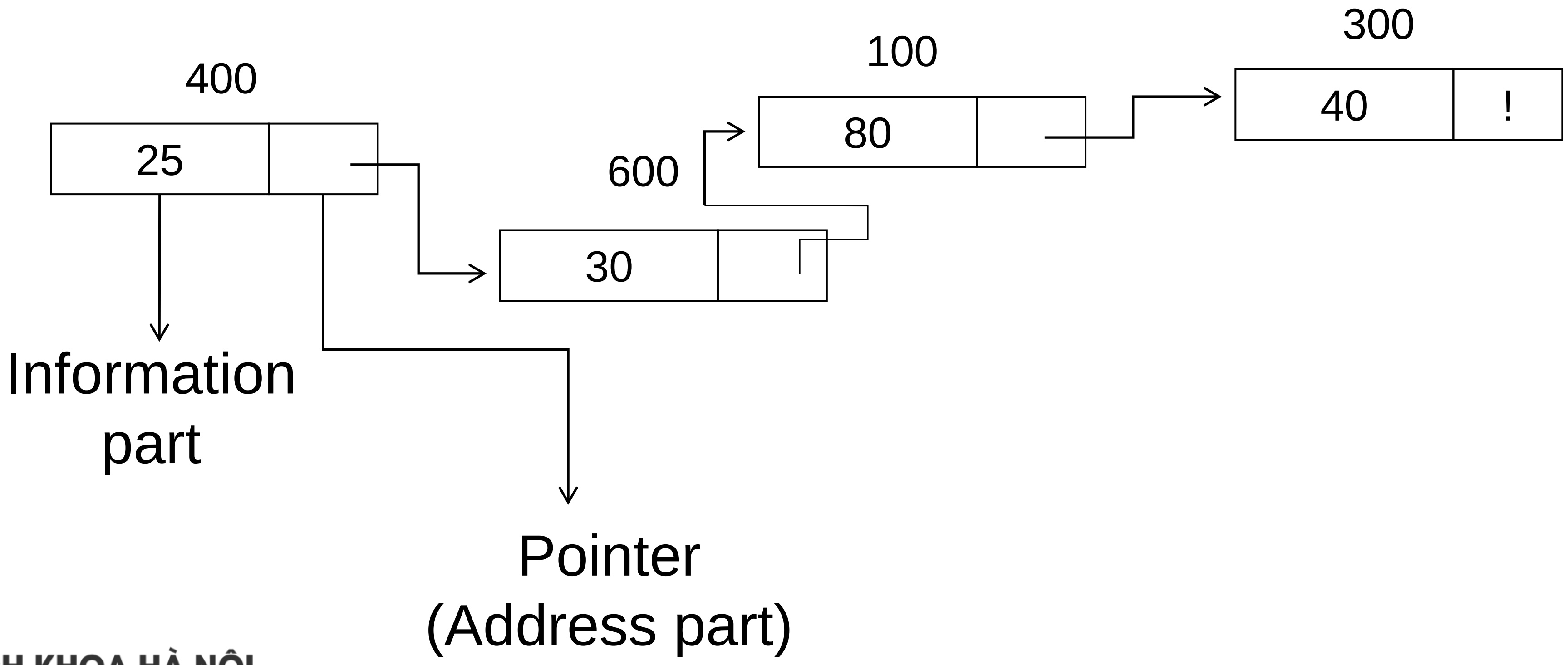
- A dynamic data structure can overcome these problems

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 singly linked List?

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

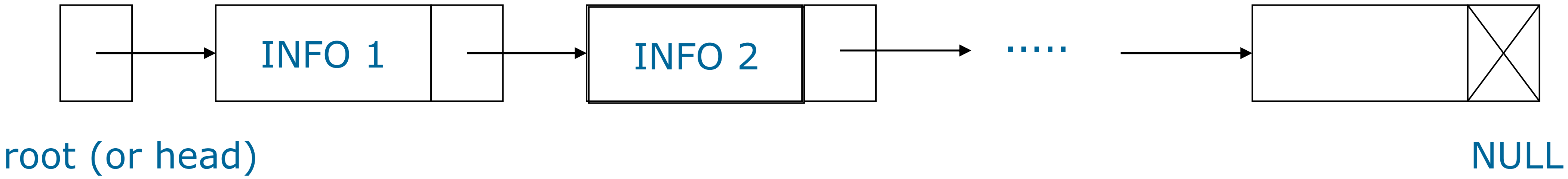


Outline

- Introduction to linked list
- **Implementation of singly linked list data structure**
- Using list in specific problems

Implementation of singly linked list in C

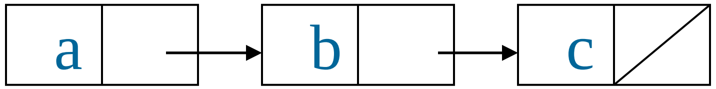
- In C, the pointer is used for the location of the next element.
  - the value of the last node is NULL
- Using Self-Referential Structures
  - You may define the type for INFO data using struct and typedef
- Important factor: root pointer
  - always points to the first element



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;
```



Operations for singly linked list

- Memory allocation for a new element (node)
- Insert new node to the list
  - at head
  - in the middle (based on position of current node or an absolute position):
    - after current node
    - before current node
- Delete (remove) an element
- Navigate (traverse) – Display the content - Searching
- Inverse the list
- Free the list
- …

Main steps in building a program using data structures

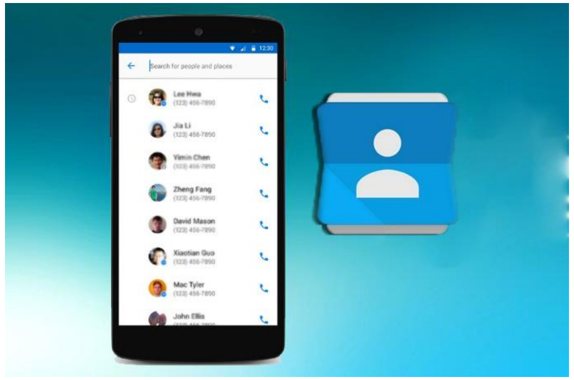
- Data types definition of the data structure
  - for an INFO field, for a data structure item
- Global variable declaration (optional)
  - important pointers for example
- Implementation of the useful operations on data structure as functions
- Usage of data structures and functions in the program

Illustrative Problem: Phone contact management

ONE LOVE. ONE FUTURE.

Description

- Develop a mobile phone contact management program, in which each contact includes information about full name, phone number and email.
- Build a single linked list to store and manage contacts with a set of functions that allow:
  - Add a new element to the beginning of the list.
  - Add a new element after the current contact element
  - Display stored contacts in the list
  - Delete a contact: At the top of the list; In the current position,..
  - Reverse the list - Free up memory allocated to the list



Lee Hwa
(123) 456-7890
Jia Li
(123) 456-7890
Yimin Chen
(123) 456-7890
Zheng Fang
(123) 456-7890
David Mason
(123) 456-7890
Xiaotian Guo
(123) 456-7890
Mac Tyler
(123) 456-7890
John Ellis

Type declaration for INFO field in each node

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

```
typedef struct contact_t {
        char name[20];
        char tel[11];
        char email[25];
} contact; // contact is the type for INFO field
```

Declaration of singly linked list of contacts

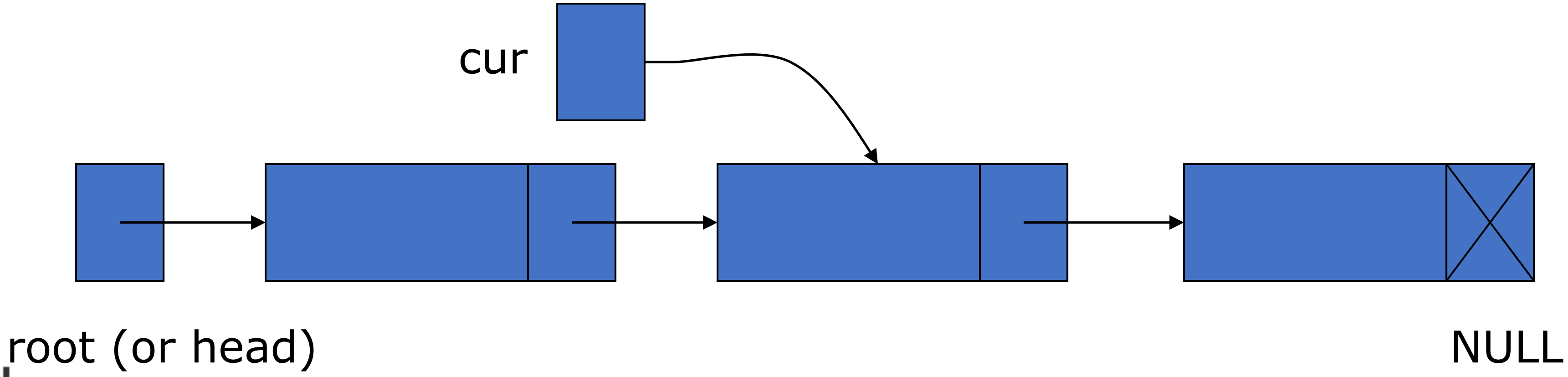
```
struct list_el {
contact el;
struct list_el *next;
};
typedef struct list_el node;
```

- “next” is the pointer variable which can express the next element; an element of the type node.
- “el” is the instance of a contact.

Declaration of important pointers

- root (or head) keeps the head of the list.
  - It is used to manage, get access to the list
- cur: Pointer variable that keeps the element just now.
- prev: Pointer point to the previous node of the one pointed by cur (optional)

```
node *root, *cur;
node *prev; /* in case you used prev */
```



Function implementation: memory allocation for a new node

```
node* makeNewNode(){
  node* new = (node*) malloc(sizeof(node));
  strcpy((new->el).name, "Tran Van Thanh");
  .... // similar statement for other contact fields
  new->next =NULL;
  return new;
}
```

- The function allocates memory and initializes one node but do not add it to the list
- Limitation: low reusability because value of data field is assigned directly in the code

Function implementation: memory allocation for a new node

- Improve the makeNewNode function
  - receive the data field as parameter - give a specific data (for the new node) ➔ allocate new node in the memory and return the pointer
  - higher reusability : For example, load data from a record file and create corresponding nodes

```
node* makeNewNode(contact ct){
  node* new = (node*)malloc(sizeof(node));
  new->el= ct;
  new->next =NULL;
  return new;
}
```

Getting input data for a node

```
contact readNode(){
    contact tmp;
    printf(“Input the full name:”);
    gets(tmp.name);
    ...
    return tmp;
}
```

Display the information of one node

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

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


}
```

- **These functions (read node, display node) do not belong to the data structures but necessary and depend on problem.**

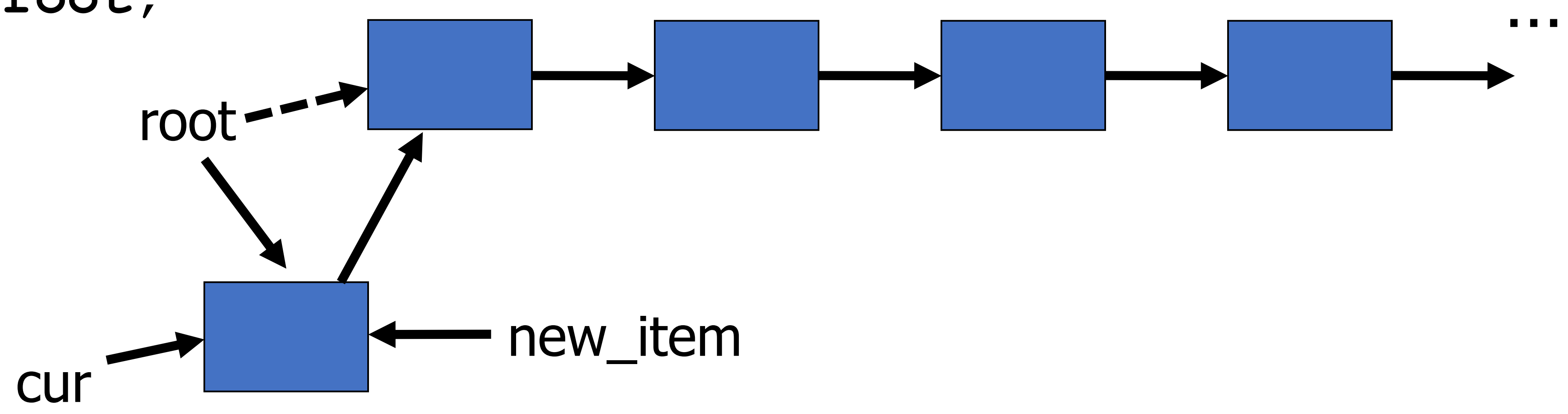
Display the information of one node - Solution

```
   void displayNode(node* p){
   if (p==NULL){printf(“NULL Pointer error.\n”); return; }
   contact tmp = p->el;
   printf(“%-20s\t%-15s\t%-25s%-p\n”, tmp.name, tmp.tel,
    tmp.email, p->next);
   }
//driver main function
void main(){
   contact tmp = readNode();
   root = makeNewNode(tmp);
   displayNode(root);
}
```

Insert a new node to the list : at head

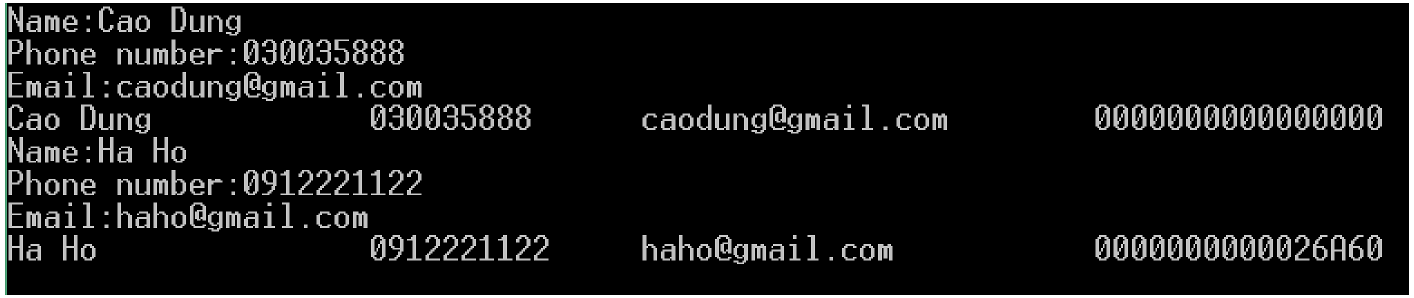
- Hint on logic:

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



Insert node at head of the list : solution

```
void insertAtHead(contact ct){
  node* new = makeNewNode(ct);
  new->next = root;
  root = new;
  cur = root;
}
void main(){
  contact tmp; int i;
  for(i=0;i<2;i++){
      tmp = readNode(); insertAtHead(tmp);
      displayNode(root);
   }
}
```

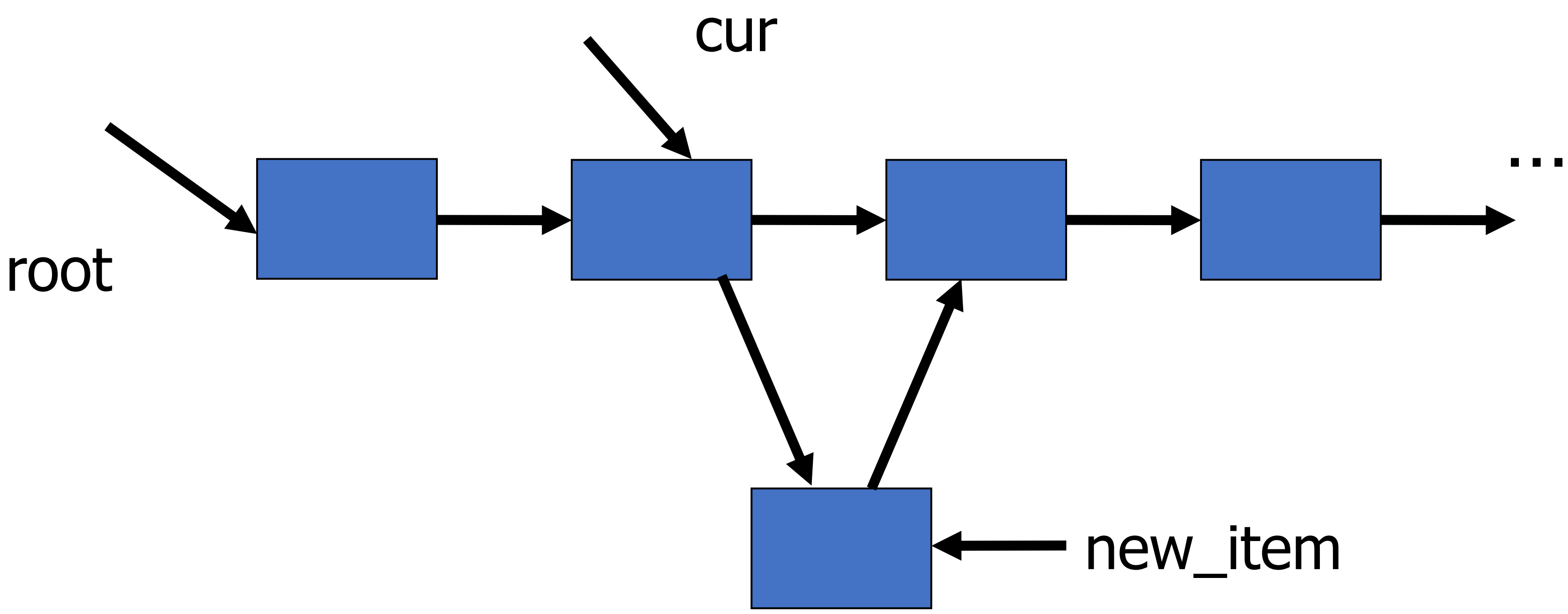


```
Name:Cao Dung
Phone number:030035888
Email:caodung@gmail.com
Cao Dung            030035888      caodung@gmail.com        0000000000000000
Name:Ha Ho
Phone number:0912221122
Email:haho@gmail.com
Ha Ho               0912221122     haho@gmail.com           0000000000026A60
```

Insert new node after the current node

- Pseudo code

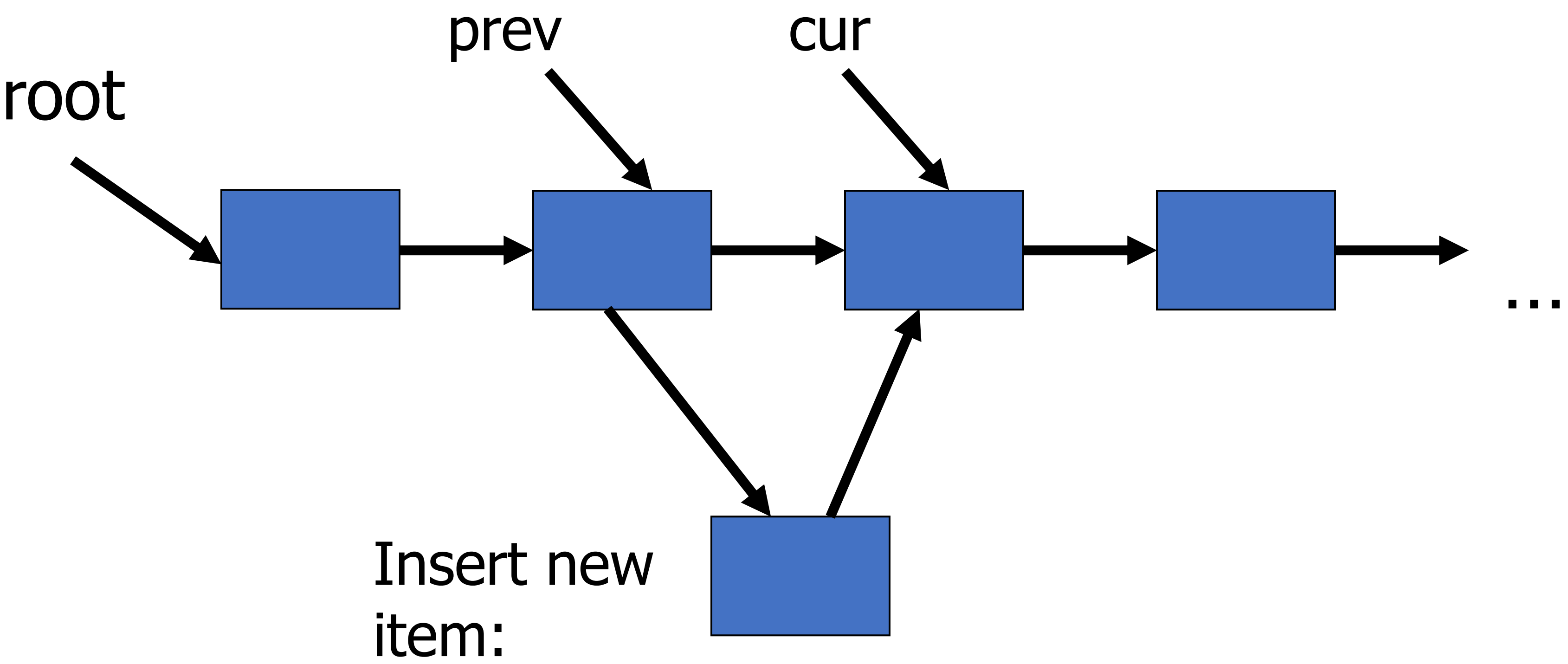
```
create new_item
new->next = cur->next;
cur->next = new;
cur= cur->next;
```



Insert new node after the current node

```
new = makeNewNode(ct); // ct is a contact data
if ( root == NULL ) {/* if there is no element */
    root = new;
    cur = root;
}
else if (cur == NULL) return;
else {
    new->next=cur->next;
    cur->next = new;
    /* prev=cur; */
    cur = cur->next;
}
```

Insert node before current position



Function: insertBeforeCurrent

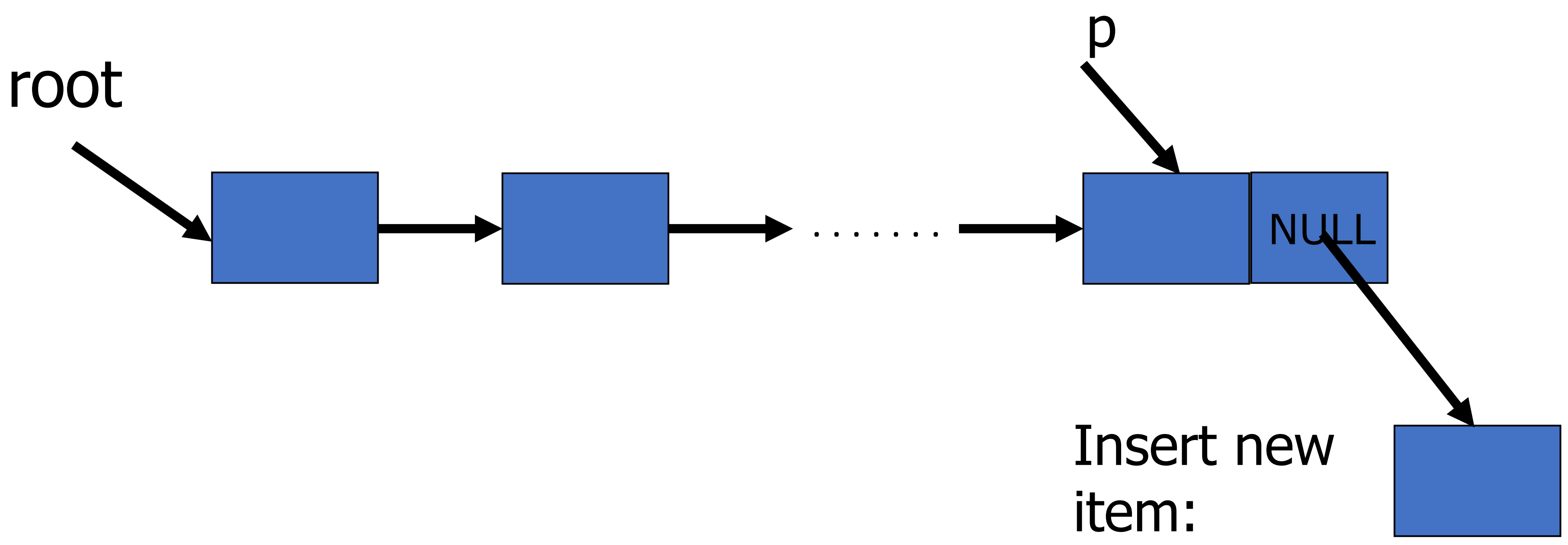
```
void insertBeforeCurrent(contact e) {
   node_addr * new = makeNewNode(e);
    if ( root == NULL ) {  /* if there is no element */
       root = new;
       cur = root;
       prev = NULL;
  } else {
       new->next=cur;
       if (cur==root) {/* if cur pointed to first element */
              root = new; /* nut moi them vao tro thanh dau danh sach */
       }
       else prev->next = new; // assume prev pointer always point to the previous node
       cur = new;
  }
}
```

If you do not frequently update the pointer prev

```
/* determine prev */
tmp = root;
 while (tmp!=NULL && tmp->next!=cur && cur !=NULL)
        tmp=tmp->next;
 prev = tmp;
```

Insert node at the end of the list

- Identify pointer p which points to the last node (next pointer is NULL)



Insert node at the end of the list : solution

```
void insertAtTail(contact ct){
  node* new = makeNewNode(ct);
 if (root == NULL) { root = new; cur = new; prev = NULL; return;
 }
 node* p = root;
 while (p->next !=NULL) p=p->next;
 p->next = new;
 cur = new; prev = p;
}
void main(){
  contact tmp; int i;
  for(i=0;i<2;i++){
     tmp = readNode(); insertAtTail(tmp);
     displayNode(root);
   }
}
```

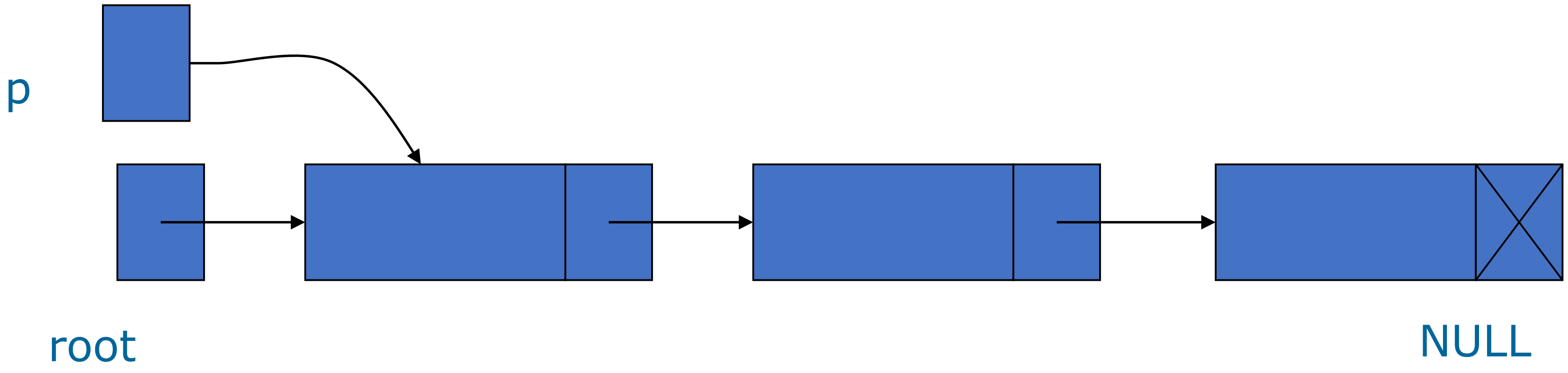
Insert node at the end of the list : version using recursion

```
node* insertLastRecursive(node* root, contact ct){
    if(root == NULL){
        return makeNewNode(ct);
    }
    root->next = insertLastRecursive(root->next, ct);
    return root;
}
void main(){
  contact tmp; int i;
  for(i=0;i<2;i++){
     tmp = readNode(); root = insertLastRecursive(tmp);
     displayNode(root);
   }
}
```

Navigate (traverse) the linked list

- Necessary in tasks such as displaying list's content or copying list
- The traversing is finished if the last node is reached

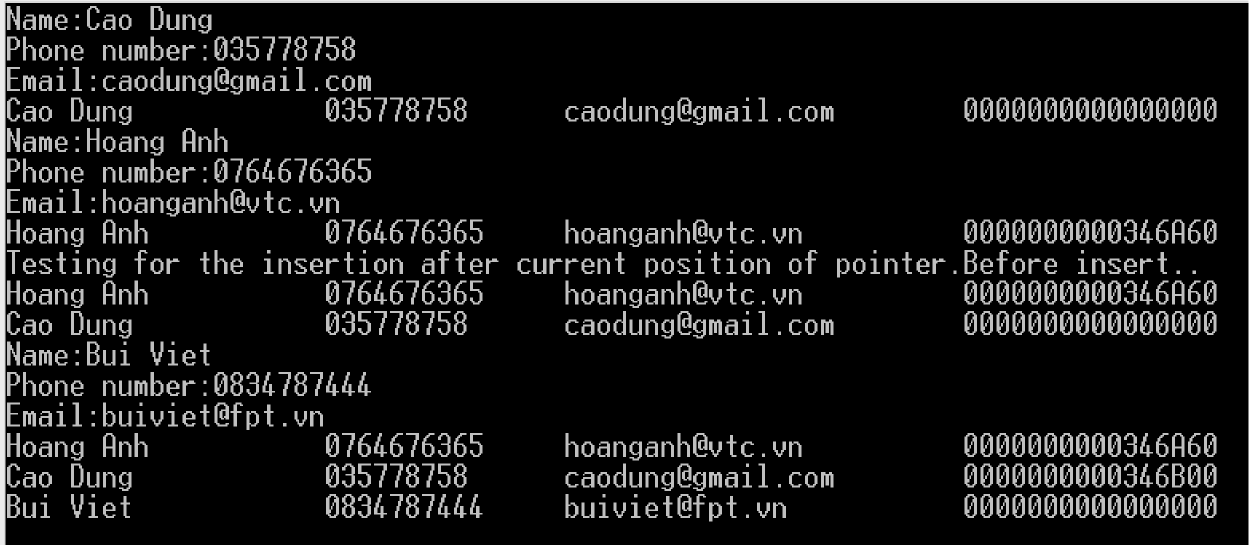
```
void traversingList(node *root){
node * p;
for ( p = root; p!= NULL; p = p->next )
  displayNode(p);
}
```



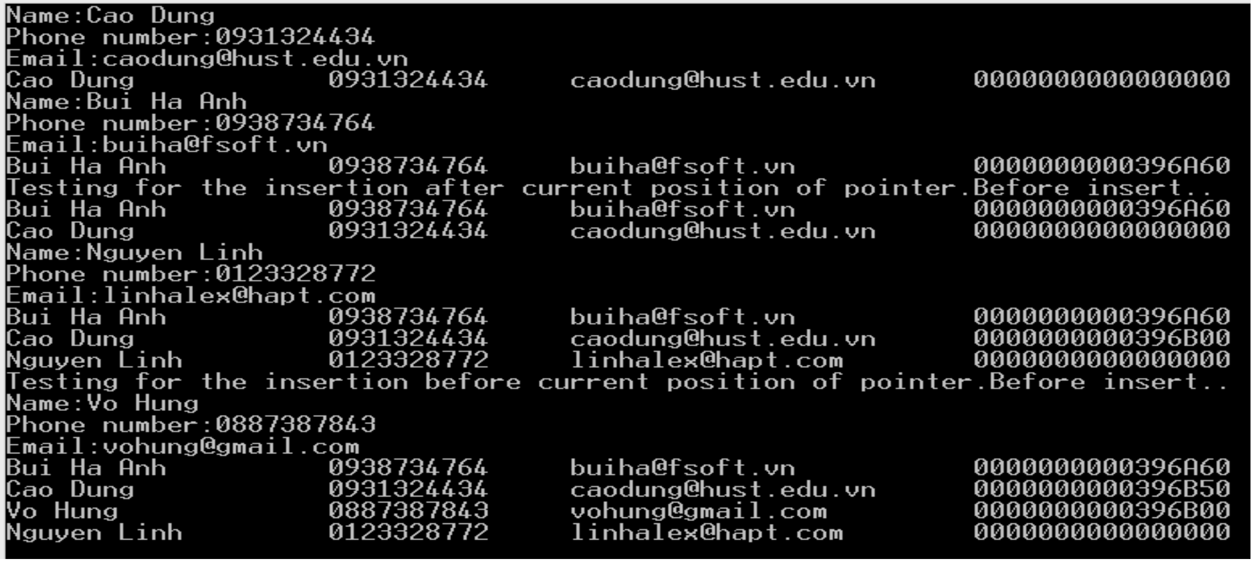
Using the functions to create and display a linked list

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

```
void main(){
   n=5;
    while (n){
      node tmp = readNode();
      insertAtHead(tmp);
     // or insertAfter..
       n--;
    }
    traversingList(root);
}
```



```
Name:Cao Dung
Phone number:035778758
Email:caodung@gmail.com
Cao Dung            035778758      caodung@gmail.com        0000000000000000
Name:Hoang Anh
Phone number:0764676365
Email:hoanganh@vtc.vn
Hoang Anh           0764676365     hoanganh@vtc.vn          0000000000346A60
Testing for the insertion after current position of pointer.Before insert..
Hoang Anh           0764676365     hoanganh@vtc.vn          0000000000346A60
Cao Dung            035778758      caodung@gmail.com        0000000000000000
Name:Bui Viet
Phone number:0834787444
Email:buiviet@fpt.vn
Hoang Anh           0764676365     hoanganh@vtc.vn          0000000000346A60
Cao Dung            035778758      caodung@gmail.com        0000000000346B00
Bui Viet            0834787444     buiviet@fpt.vn           0000000000000000
```



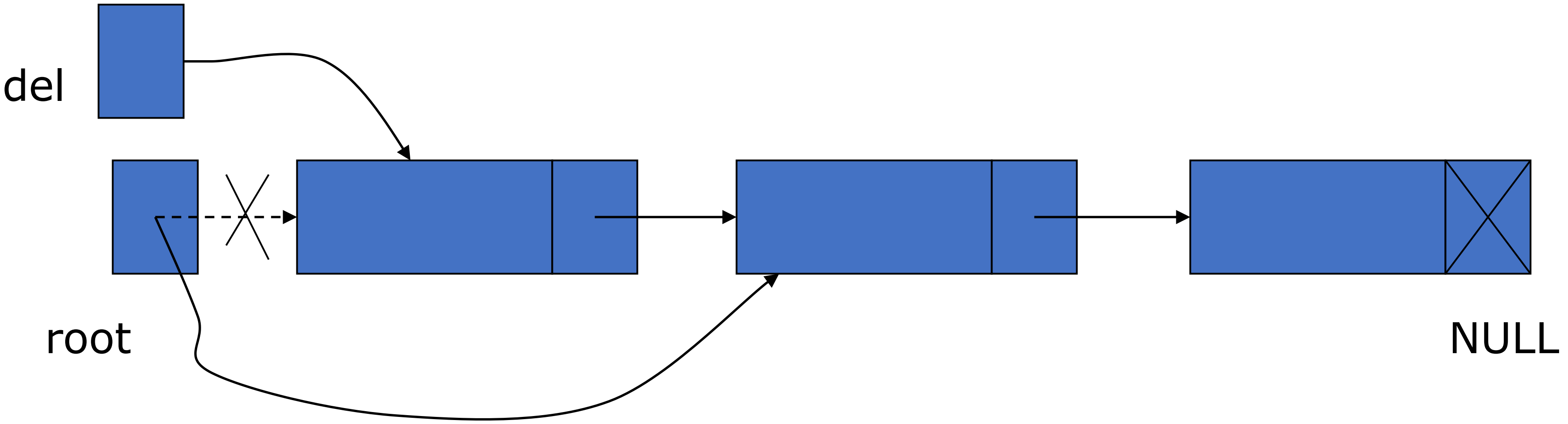
```
Name:Cao Dung
Phone number:0931324434
Email:caodung@hust.edu.vn
Cao Dung            0931324434     caodung@hust.edu.vn      0000000000000000
Name:Bui Ha Anh
Phone number:0938734764
Email:buiha@fsoft.vn
Bui Ha Anh          0938734764     buiha@fsoft.vn           0000000000396A60
Testing for the insertion after current position of pointer.Before insert..
Bui Ha Anh          0938734764     buiha@fsoft.vn           0000000000396A60
Cao Dung            0931324434     caodung@hust.edu.vn      0000000000000000
Name:Nguyen Linh
Phone number:0123328772
Email:linhalex@hapt.com
Bui Ha Anh          0938734764     buiha@fsoft.vn           0000000000396A60
Cao Dung            0931324434     caodung@hust.edu.vn      0000000000396B00
Nguyen Linh         0123328772     linhalex@hapt.com        0000000000000000
Testing for the insertion before current position of pointer.Before insert..
Name:Vo Hung
Phone number:0887387843
Email:vohung@gmail.com
Bui Ha Anh          0938734764     buiha@fsoft.vn           0000000000396A60
Cao Dung            0931324434     caodung@hust.edu.vn      0000000000396B50
Vo Hung             0887387843     vohung@gmail.com         0000000000396B00
Nguyen Linh         0123328772     linhalex@hapt.com        0000000000000000
```

Delete first element of the List

- Write a function that delete the first element of the list
- Logic:

```
del=root; root = del->next; free(del);
```

- Change the value of “root” into the value of “next” which is pointed by “del.”



Delete first element of the list

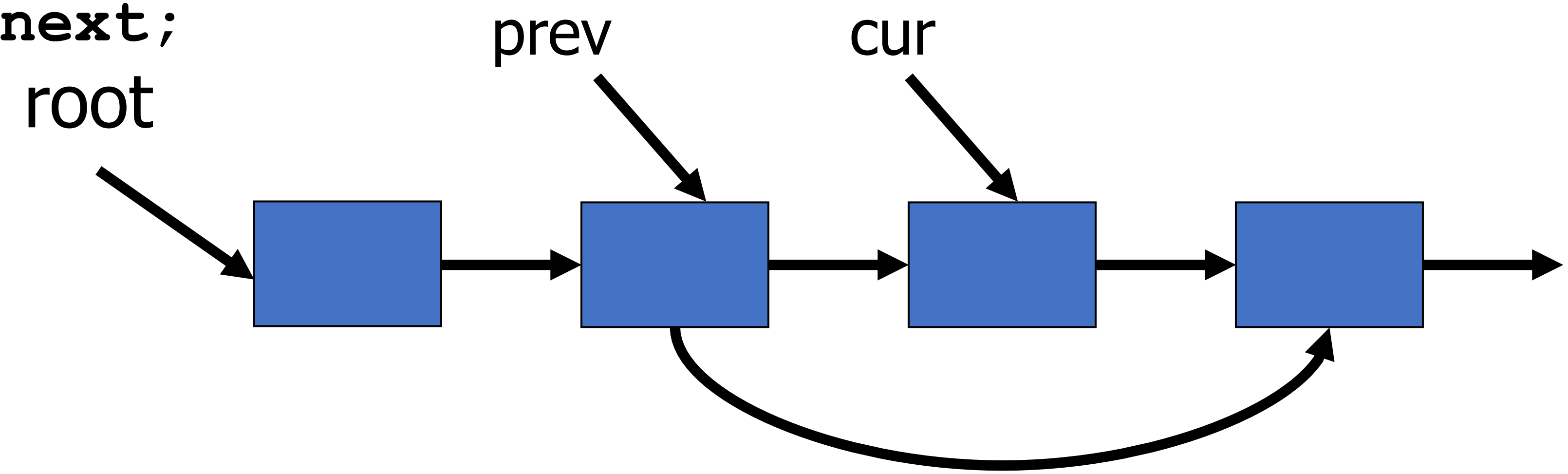
```
void deleteFirstElement(){
 node* del = root;
 if (del == NULL) return;
 root = del->next;
 free(del);
 cur = root;
 prev = NULL;
}
```

Delete an element from the middle

- Remove the node pointed by the pointer cur (current node).
  - Design and implement of ***deleteCurrentElement*** function

- Logic: Use pointer prev which point to the node just before the node to delete

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



Solution: Delete element pointed by cur

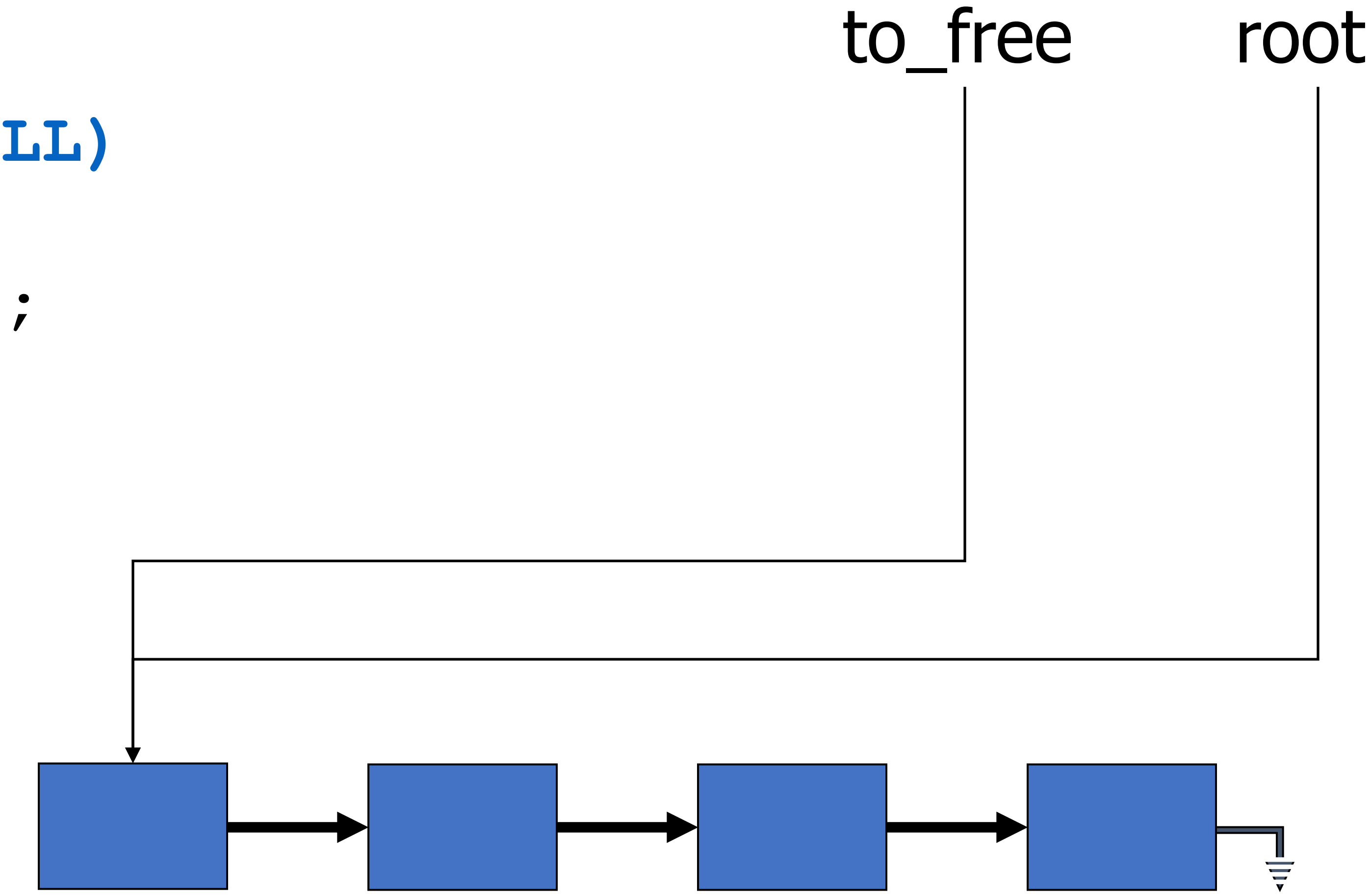
```
void deleteCurrentElement(){
 if (cur==NULL) return;
 if (cur==root) deleteFirstElement();
 else {
  prev->next = cur->next;
 free(cur);
 cur = prev->next; // or cur = root;
}
```

Delete a node with a specific contact (using recursion)

```
Node* removeNodeRecursive(Node* root, contact e){
    if(root == NULL) return NULL;
    if(root->el == e){
        Node* tmp = root; root = root->next; free(tmp);
        return root;
    }
    root->next = removeNodeRecursive(root->next, e);
    return root;
}
```

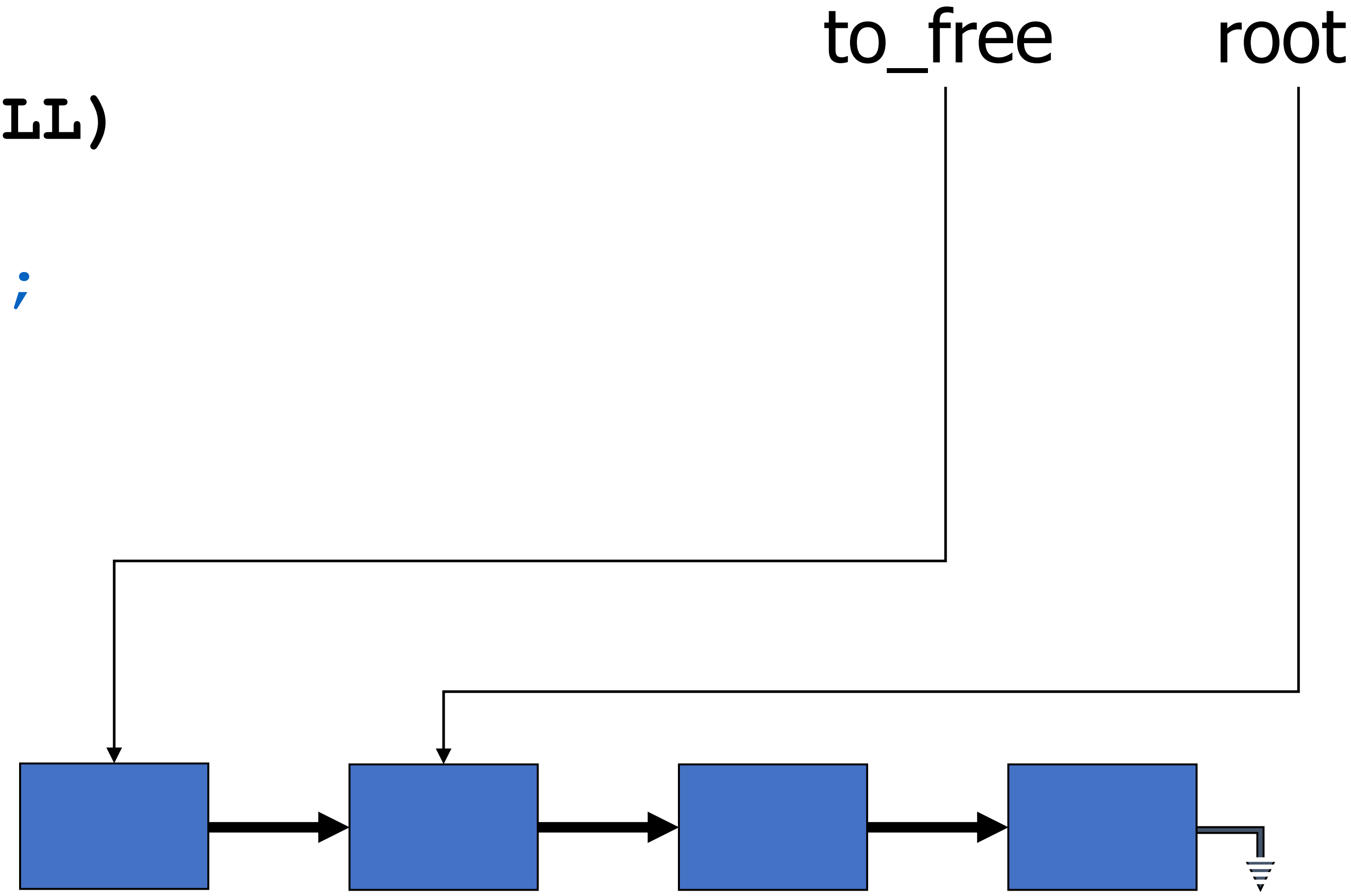
Freeing all nodes of a list

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



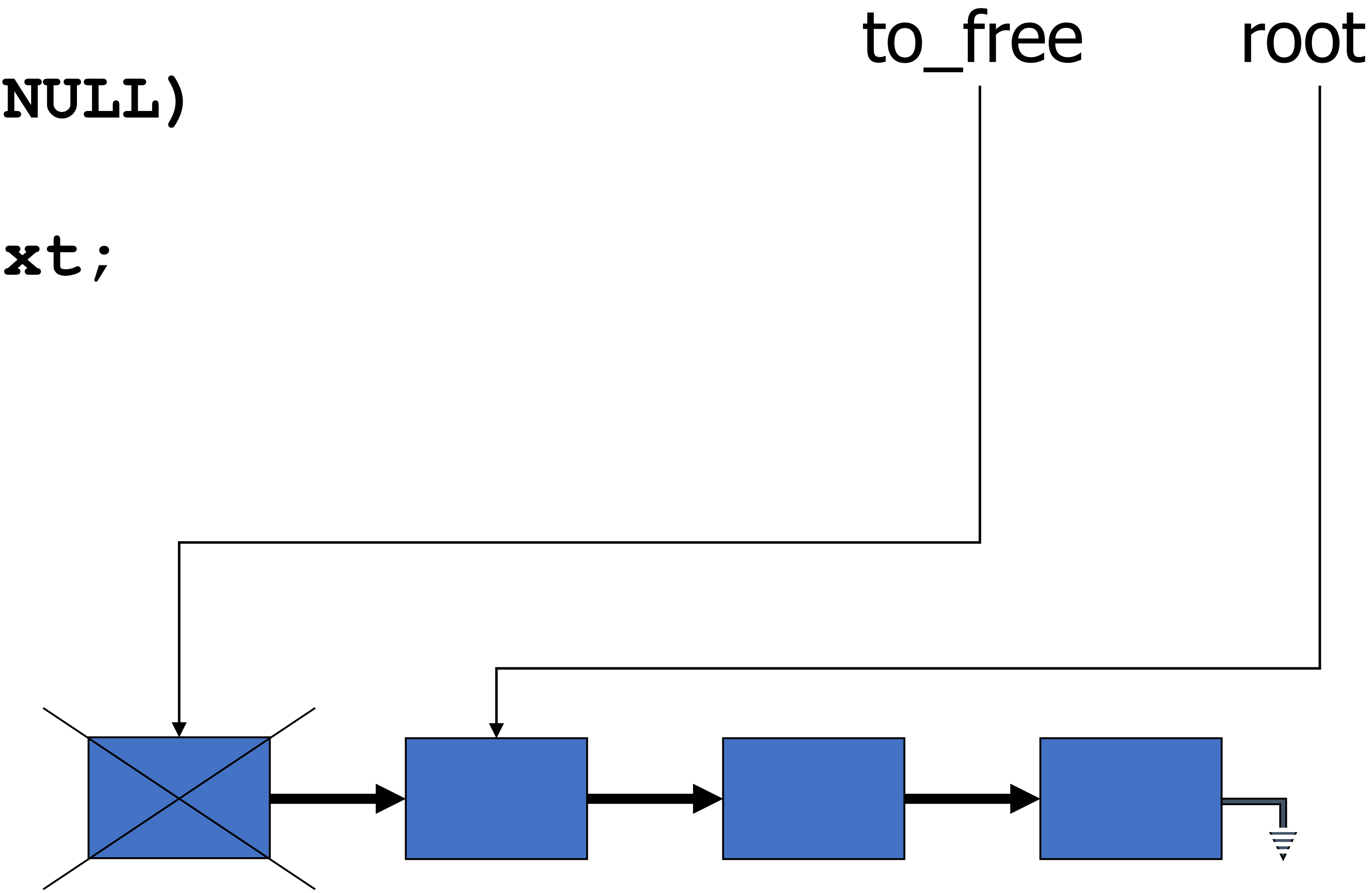
Freeing all nodes of a list

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



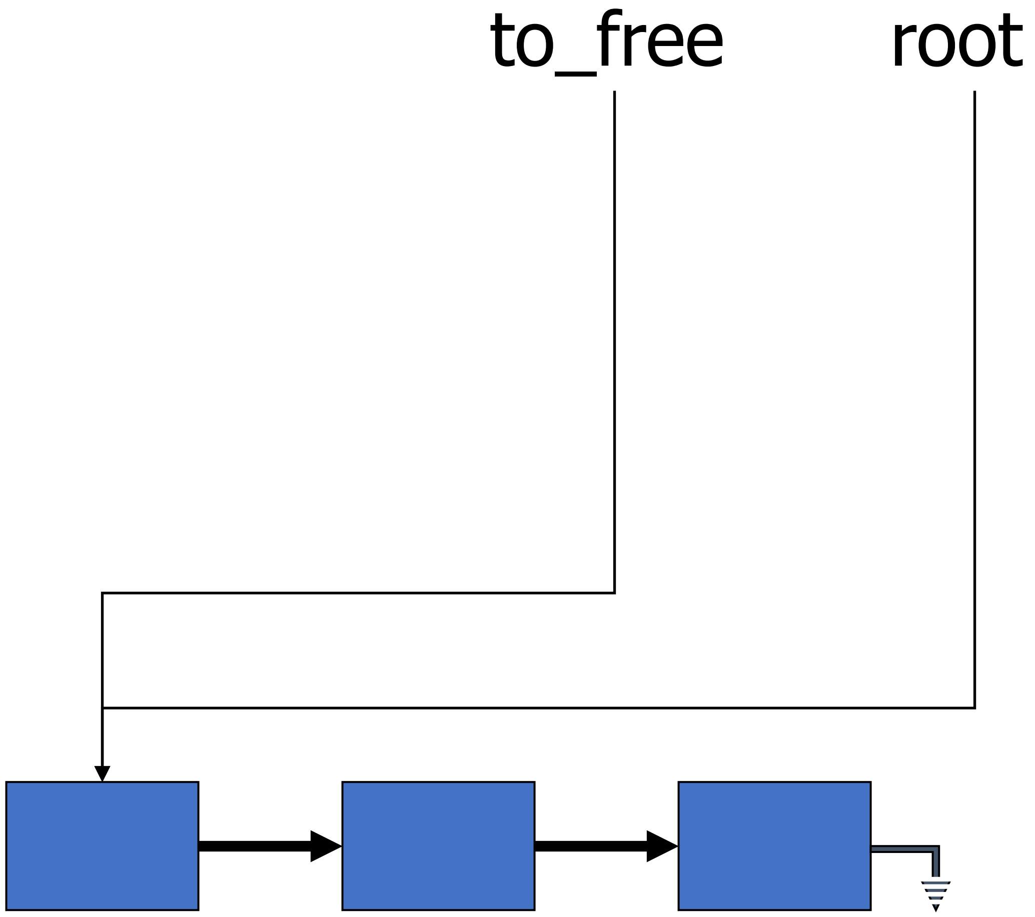
Freeing all nodes of a list

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



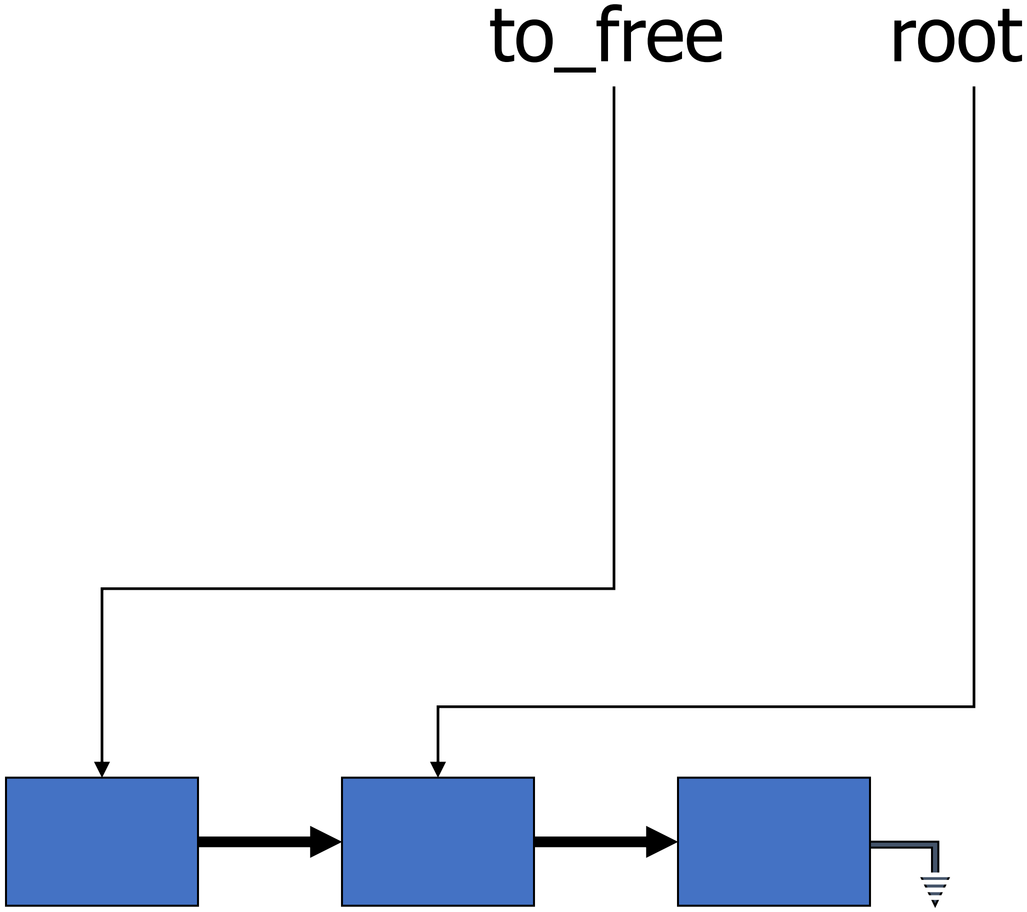
Freeing all nodes of a list

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



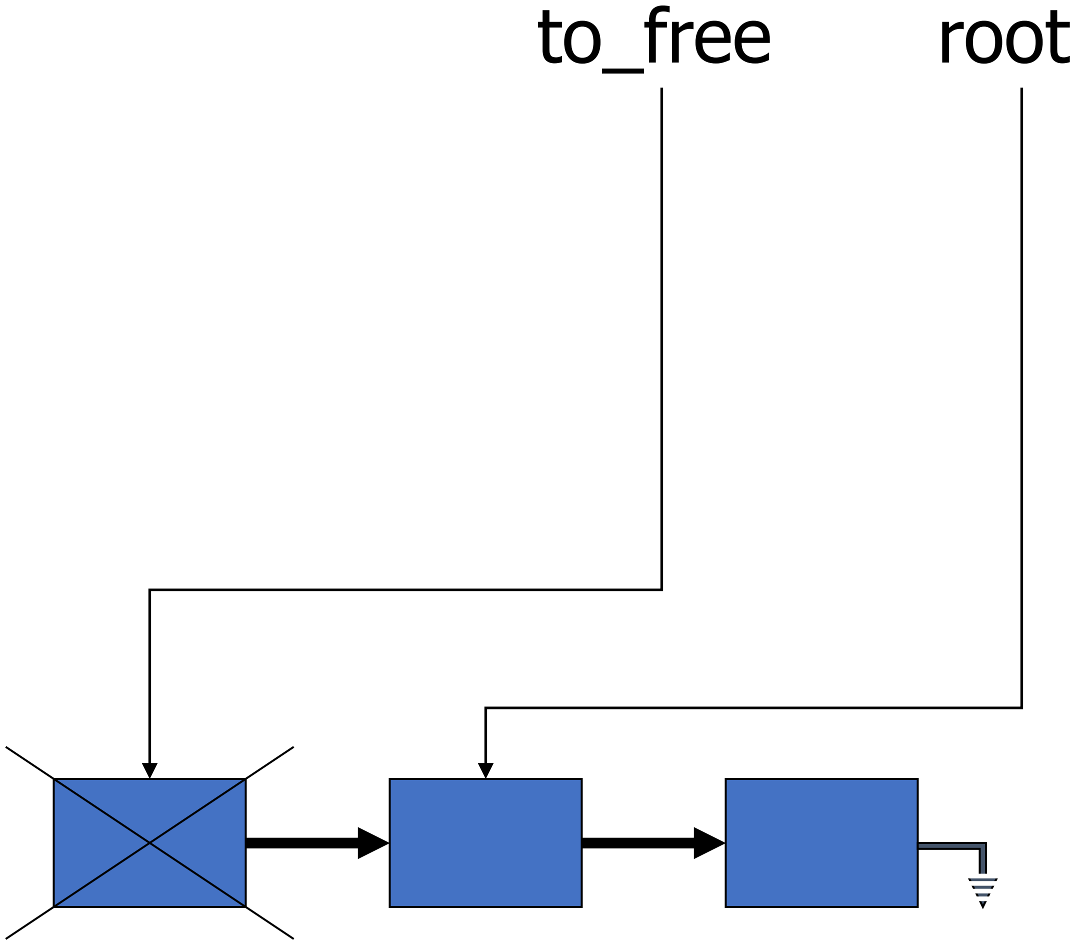
Freeing all nodes of a list

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



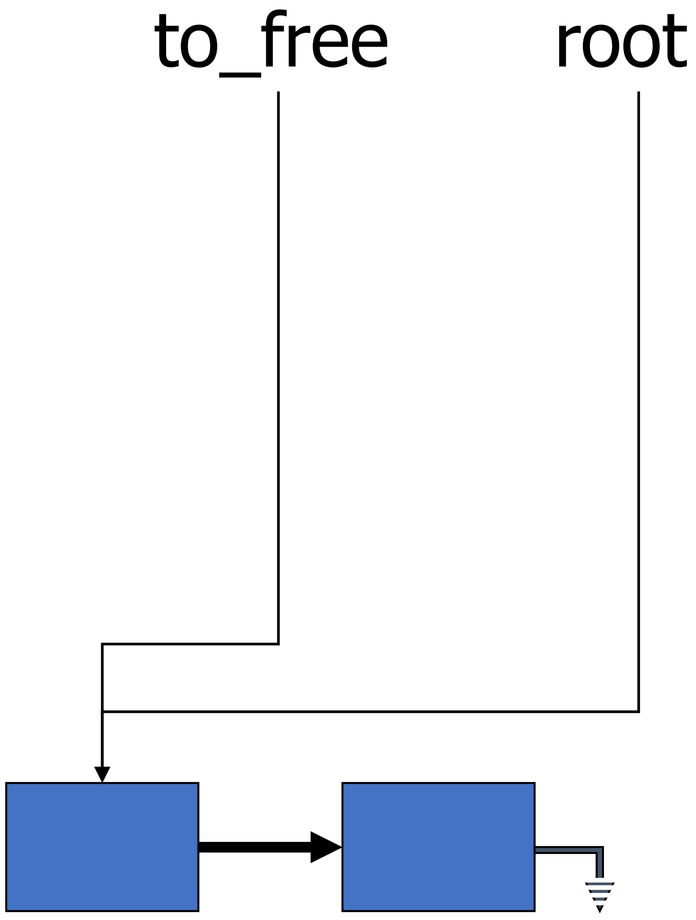
Freeing all nodes of a list

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



Freeing all nodes of a list

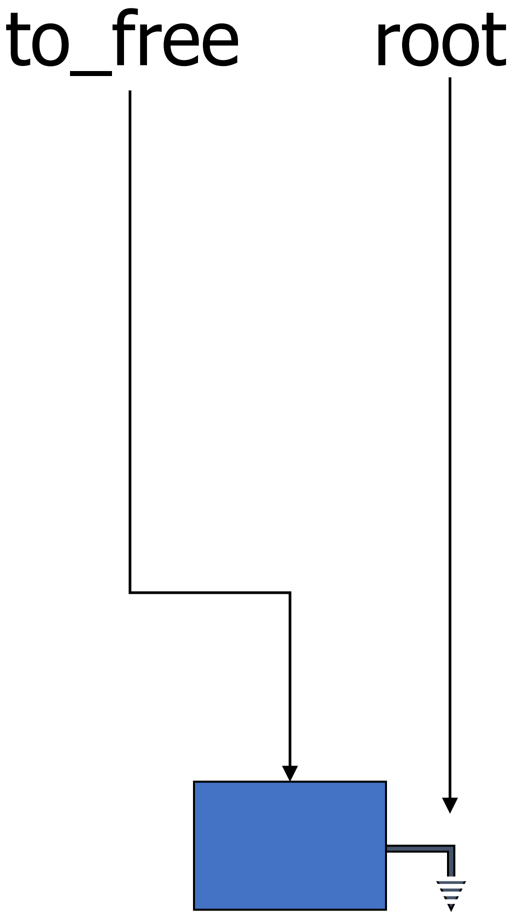
```
while (to_free != NULL)
{
    root = root->next;
    free(to_free);
    to_free = root;
}
```



Freeing all nodes of a list

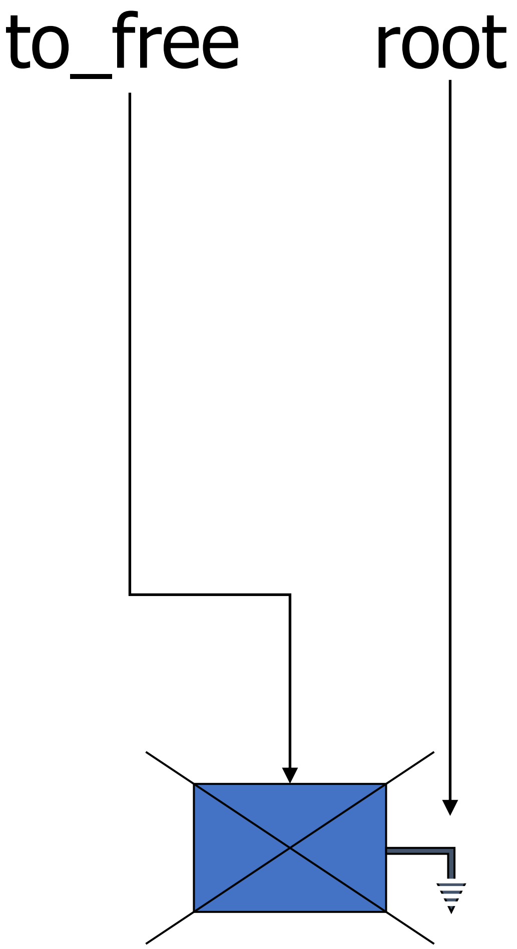
- After some iteration …

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



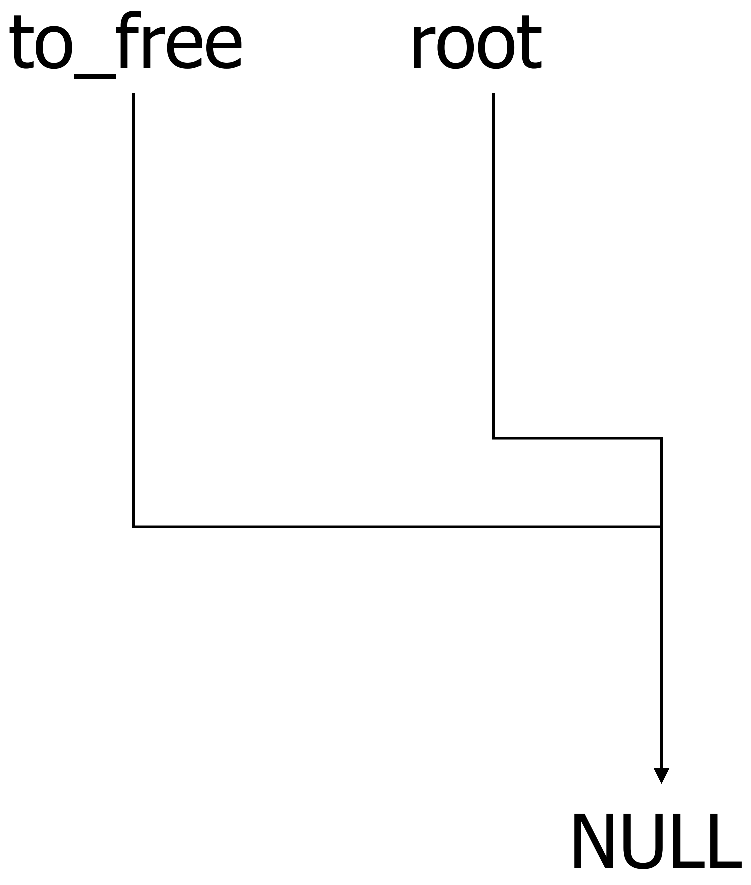
Freeing all nodes of a list

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



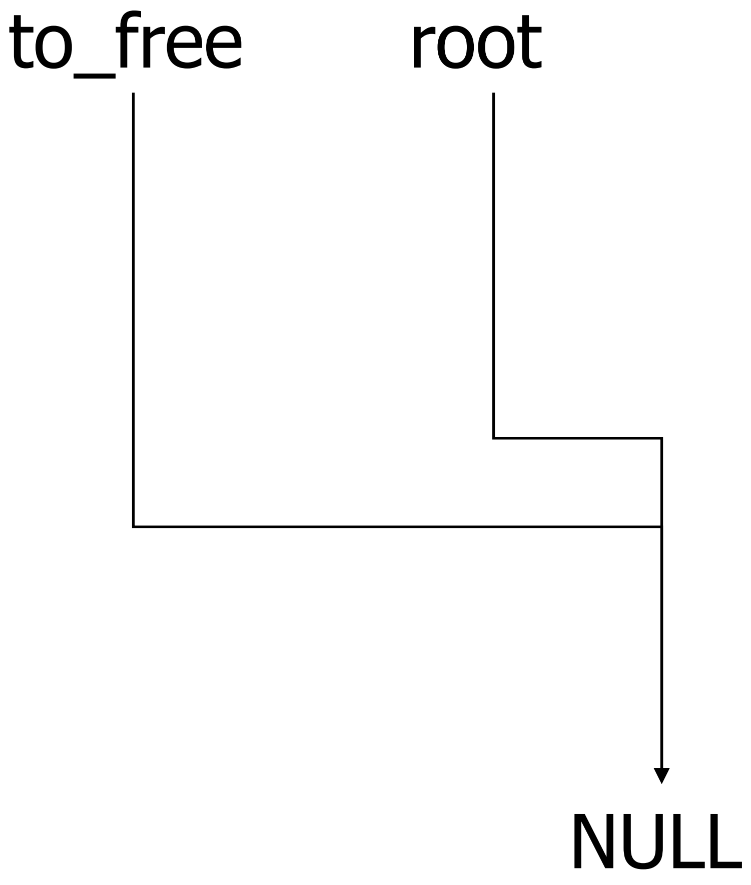
Freeing all nodes of a list

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



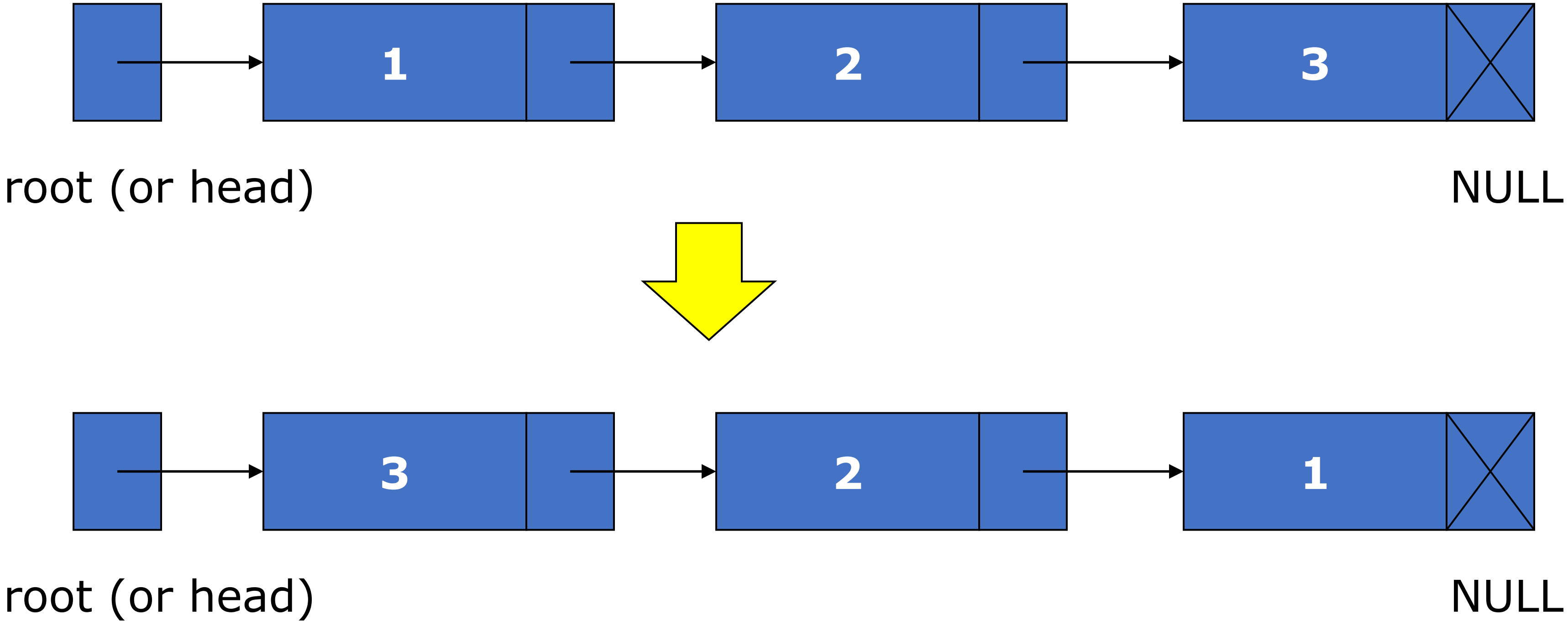
Freeing all nodes of a list

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



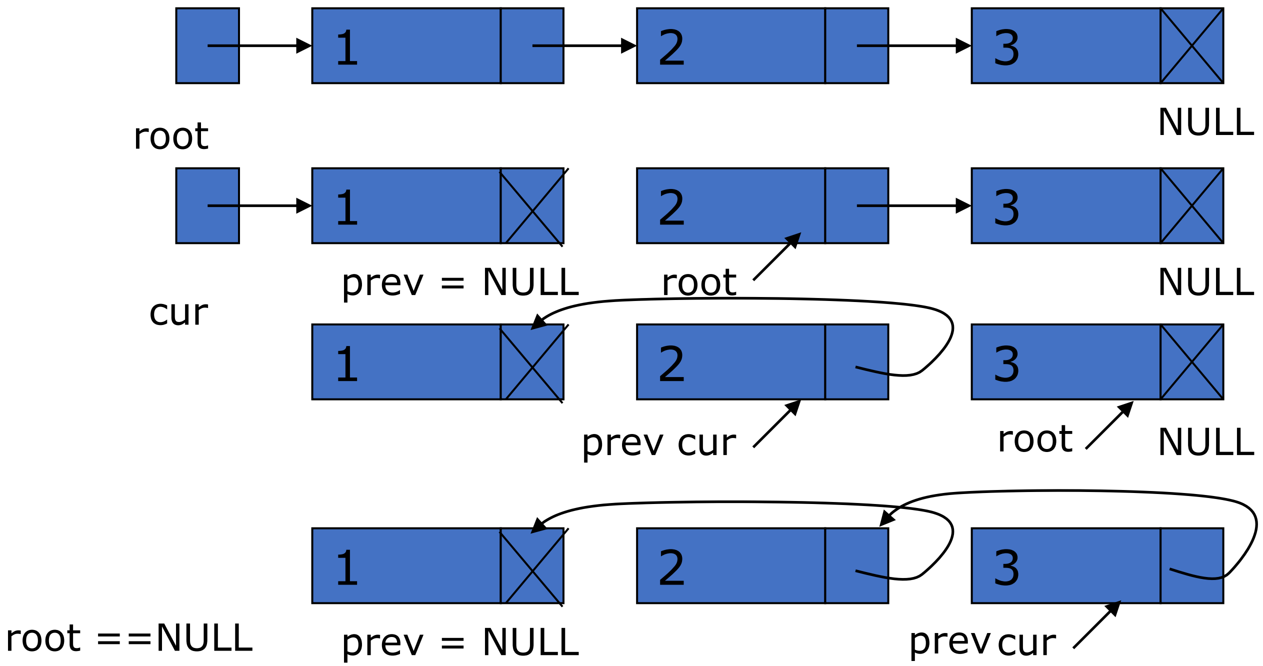
Reverse a list

- Write a function that reverse the content a list.

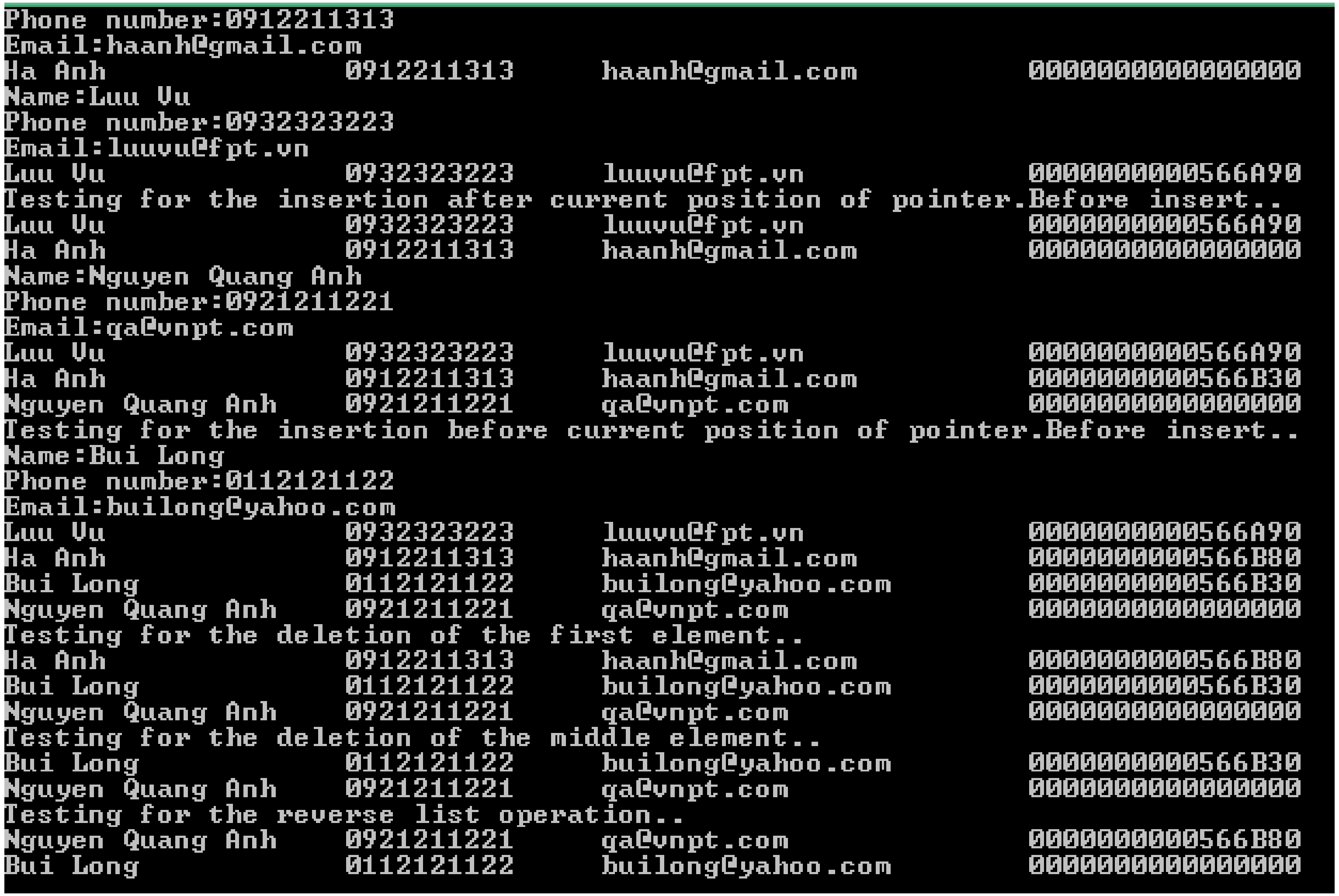


Solution

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



Program Output



```
Phone number:0912211313
Email:haanh@gmail.com
Ha Anh              0912211313     haanh@gmail.com          0000000000000000
Name:Luu Vu
Phone number:0932323223
Email:luuvu@fpt.vn
Luu Vu              0932323223     luuvu@fpt.vn             0000000000566A90
Testing for the insertion after current position of pointer.Before insert..
Luu Vu              0932323223     luuvu@fpt.vn             0000000000566A90
Ha Anh              0912211313     haanh@gmail.com          0000000000000000
Name:Nguyen Quang Anh
Phone number:0921211221
Email:qa@vnpt.com
Luu Vu              0932323223     luuvu@fpt.vn             0000000000566A90
Ha Anh              0912211313     haanh@gmail.com          0000000000566B30
Nguyen Quang Anh    0921211221     qa@vnpt.com              0000000000000000
Testing for the insertion before current position of pointer.Before insert..
Name:Bui Long
Phone number:0112121122
Email:builong@yahoo.com
Luu Vu              0932323223     luuvu@fpt.vn             0000000000566A90
Ha Anh              0912211313     haanh@gmail.com          0000000000566B80
Bui Long            0112121122     builong@yahoo.com        0000000000566B30
Nguyen Quang Anh    0921211221     qa@vnpt.com              0000000000000000
Testing for the deletion of the first element..
Ha Anh              0912211313     haanh@gmail.com          0000000000566B80
Bui Long            0112121122     builong@yahoo.com        0000000000566B30
Nguyen Quang Anh    0921211221     qa@vnpt.com              0000000000000000
Testing for the deletion of the middle element..
Bui Long            0112121122     builong@yahoo.com        0000000000566B30
Nguyen Quang Anh    0921211221     qa@vnpt.com              0000000000000000
Testing for the reverse list operation..
Nguyen Quang Anh    0921211221     qa@vnpt.com              0000000000566B80
Bui Long            0112121122     builong@yahoo.com        0000000000000000
```

Outline

- Introduction to linked list
- Implementation of singly linked list data structure
- **Using list in specific problems**

Lab 1: Linked list manipulation

- Write a program to perform the following tasks:
  - Build a linked list with the initially provided keys as the sequence a_1, a_2, …, a_n.
  - Perform the following operations on the list:
    - adding 1 element to the beginning, to the end of the list,
    - go before or after an element in the list,
    - or remove an element from the list
- Submit the program on the automatic evaluation system

Input and output format

- **Input**
- Line 1: input a positive integer n (1 <= n <= 1000)
- Line 2: series of n positive integer numbers a_1, a_2, ..., a_n.
- The next lines are commands (ending with the # symbol) :
- **addlast k**: add element with key k to the end of the list (if k does not already exist)
- **addfirst k**: add element with key k to the beginning of the list (if k does not already exist)
- **addafter u v:** add element with key equal to u after element with key equal to v on the list (if v already exists on the list and u does not exist yet)
- **addbefore u v**: add the element with key equal to u before the element with key equal to v on the list (if v already exists on the list and u does not exist)
- **remove k**: remove the element with key k from the list
- **reverse**: reverses the order of list elements (no new elements can be allocated, only links can be changed)
- **Output:** Displays the key sequence of the list obtained after a given sequence of operation commands

Example for input and output

Input

```
5
5 4 3 2 1
addlast 3
addlast 10
addfirst 1
addafter 10 4
remove 1
#
```

Output

```
5 4 3 2 10
```

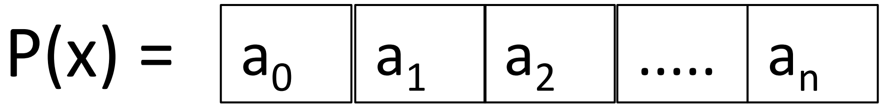
Polynomial manipulation

- A polynomial p(x) is the expression in variable x which is in the form $a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_{0,}$
  - where a_i fall in the category of real numbers
  - n is non negative integer, which is called the degree of polynomial.

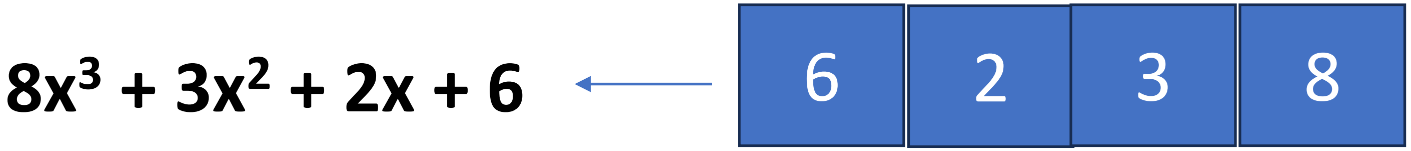
- Some basic operations in Polynomial manipulation
  - Polynomial creation (representation)
  - Addition (subtraction) of polynomials
  - Multiplication of polynomials

Polynomial representation using arrays

- Coefficients are stored in the elements in an array at corresponding subscript (index)
- Indexes represent exponents

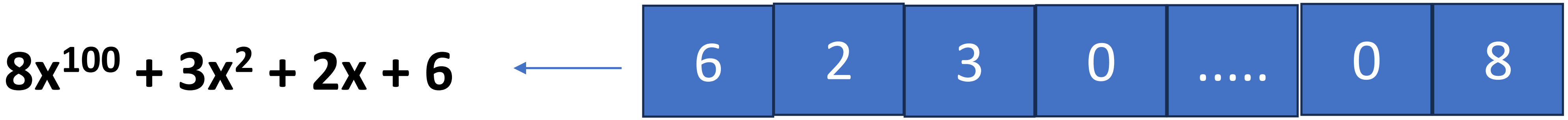


P(x) = | a_0 | a_1 | a_2 | ..... | a_n |



$8x^3 + 3x^2 + 2x + 6$ ← | 6 | 2 | 3 | 8 |

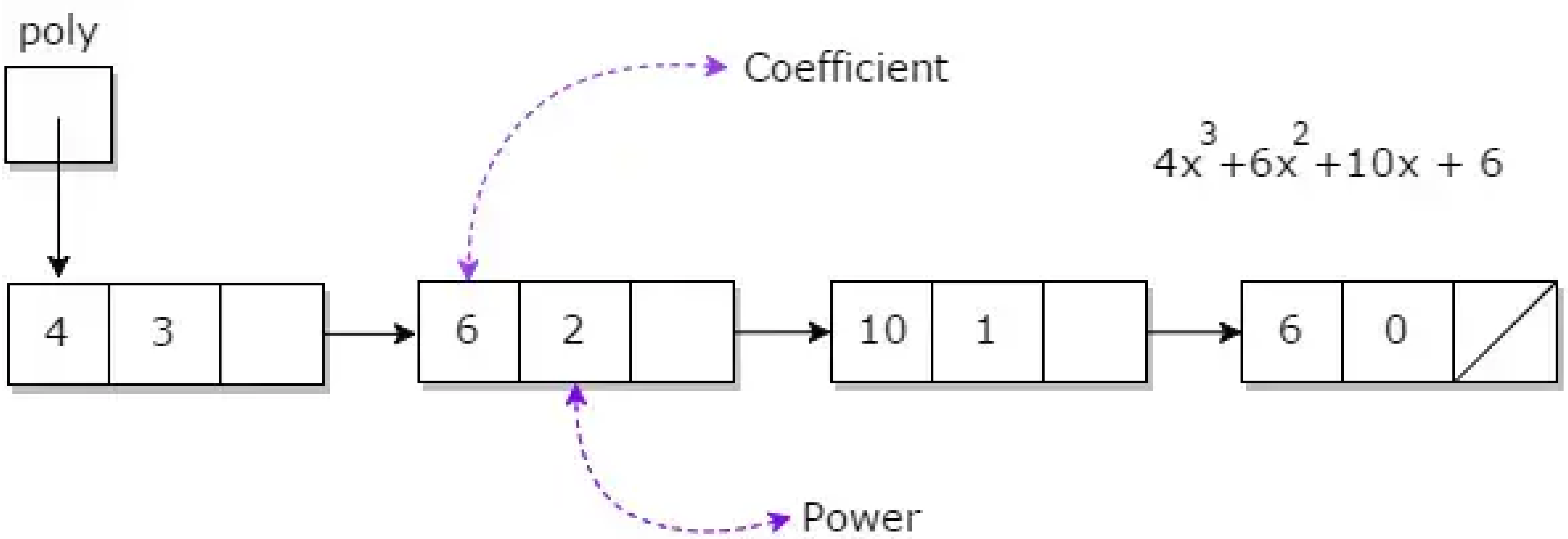
- Limitation: waste lot of memory spaces for sparse polynomials.



$8x^{100} + 3x^2 + 2x + 6$ ← | 6 | 2 | 3 | 0 | ..... | 0 | 8 |

Polynomial representation using linked list

- Each term is stored in a node of the list with 2 fields: coefficient and exponent
- Nodes are always sorted in a decreasing order of exponents
- No two nodes have the same value of exponents
- Save spaces for every polynomials



Lab2: Polynomial manipulation

- Write a program providing a list of commands over polynomials below, knowing that Each polynomial has an identifier which is a positive integer from 1 to 10000:
- **Create <poly_id>:** create a polynomial with identifier <pol_id> if this polynomial does not exists, otherwise, do nothing
- **AddTerm <poly_id> <coef> <exp>:** Add a term with coefficient <coef> and exponent <exp> to the polynomial having identifier **<poly_id>** (create a new polynomial if it does not exist)
- **EvaluatePoly <poly_id> <variable_value>**: Evaluate and print the value of the polynomial having identifier <poly_id> and <variable_value> is the value of the variable (print 0 if the polynomial does not exist)
- **AddPoly <poly_id1> <poly_id2> <result_poly_id>**: Perform the addition operation over two polynomials <pol_id1> and <poly_id2>. The resulting polynomial will have identifier <result_poly_id> (if the polynomial <result_poly_id> exists, then overrides the existing polynomial)
- **PrintPoly <poly_id>:** print the polynomial <poly_id> (if it exists) to stdout under the form <c_1> <e_1> <c_2> <e_2> ... (sequence of pairs of (coefficient, exponent) of terms of the polynomial in a decreasing order of exponents)
- **Destroy <poly_id>:** destroy the polynomial having identifier <poly_id>

Input and output format

- **Input:** Each line contains a command described above (terminated by a line containing *)
- Example:

```
AddTerm 1 3 2
AddTerm 1 4 0
AddTerm 1 6 2
AddTerm 2 3 2
AddTerm 2 7 5
PrintPoly 1
PrintPoly 2
AddPoly 2 1 3
PrintPoly 3
EvaluatePoly 2 1
```

Input and output format

- **Output:** Each line contains the information printed out by the PrintPoly and EvaluatePoly above
- Example:

```
9 2 4 0
7 5 3 2
7 5 12 2 4 0
10
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !