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

DANH SÁCH LIÊN KẾT (PHẦN II)

IT3230 C Programming Basic

ONE LOVE. ONE FUTURE.

Nội dung chính

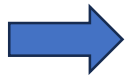
- **Cài đặt (xây dựng) danh sách liên kết tổng quát**

- Cài đặt (xây dựng) danh sách liên kết đôi

Sự cần thiết

- Lập trình viên không nhất thiết phải định nghĩa một kiểu dữ liệu riêng (thường ở dạng struct) cho dữ liệu INFO của mỗi phần tử, mà có thể định nghĩa trực tiếp trong kiểu dữ liệu của node.

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



```
struct list_el {
char name[20];
char tel[11];
char email[25];
struct list_el *next;
};
typedef struct list_el node;
```

- Tuy nhiên, điều gì xảy ra khi sử dụng danh sách liên kết với các bài toán ứng dụng cụ thể tương tự như Quản lý danh bạ, Quản lý sản phẩm, Quản lý sinh viên, ...
  - Khó dùng lại mã nguồn đã làm cho bài toán trước.

Khai báo danh sách liên kết "tổng quát"

- Xây dựng danh sách liên kết tổng quát có thể giúp giải quyết vấn đề

```
typedef ... elementtype;
struct node_t {
   elementtype element;
   struct node_t *next;
};
typedef struct node_t node;
node* root;
node* cur;
node* prev;
```

the part need to be modified through different problems.

```
typedef contact elementtype;
```

generic functions work on the elementtype parameter.

Hàm tổng quát: Tạo một nút mới cho danh sách

```
node* makeNewNode(elementtype e){
 node* new = (node*) malloc(sizeof(node));
 new->element=e;
 new->next =NULL;
 return new;
}
```

Thêm một phần tử vào sau phần tử hiện hành

```
void insertAfterCurrent(elementtype e) {
   node* new = makeNewNode(e);
     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;
  }
}
```

Thêm một phần tử vào trước phần tử hiện hành

```
void insertBeforeCurrent(elementtype e) {
   node* 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;
       else prev->next = new;
       cur = new;
  }
}
```

Thêm nút mới vào cuối danh sách

```
void insertAtTail(elementtype e){
 node* new = makeNewNode(e);
 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;
}
```

Thêm nút mới vào cuối danh sách : phiên bản sử dụng đệ quy

```
node* insertLastRecursive(node* root, elementtype e){
    if(root == NULL){
        return makeNewNode(e);
    }
    root->next = insertLastRecursive(root->next, e);
    return root;
}
```

Xóa phần tử với thông tin liên lạc cụ thể

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

Duyệt danh sách liên kết

```
void traverseList() {
 node* p;
 for (p = root; p != NULL; p = p->next ) {
 printData(p->element);
}
```

Tìm nút chứa một phần tử cụ thể

```
Node* find(Node* root, elementtype e){
  Node* p;
     for(p = root; p != NULL; p = p->next){
          if(p->element == e) return p;
     }
     return NULL;
}
```

Đảo ngược danh sách

```
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;
}
```

Hàm nhập xuất dữ liệu đặc thù theo bài toán

- **Cần xây dựng lại các hàm này tùy thuộc theo bài toán.**

```
elementtype readData(){
  elementtype res;
  printf("name:");gets(res.name);
  printf("tel:");gets(res.tel);
  printf("email:");gets(res.email);
   return res;
}
void printData(elementtype res){

  printf("%15s\t%10s\t%20s\n",res.name, res.tel, res.email);
}
```

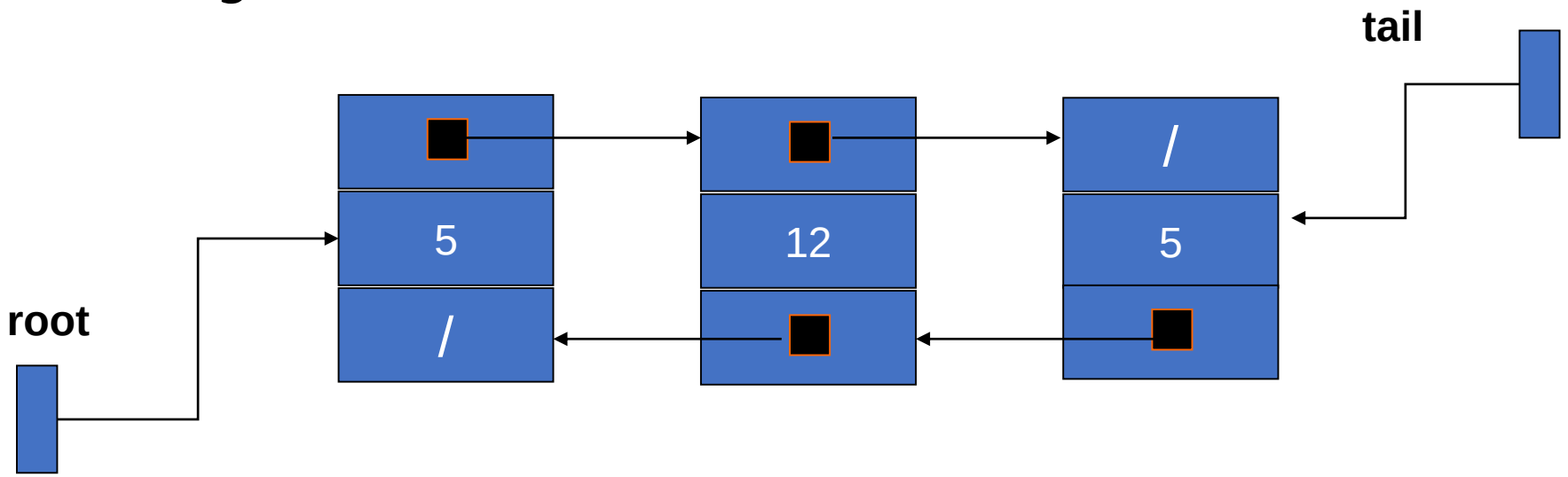
Nội dung chính

- Cài đặt (xây dựng) danh sách liên kết tổng quát

- **Cài đặt (xây dựng) danh sách liên kết đôi**

Danh sách liên kết đôi

- Danh sách liên kết đơn có một số hạn chế:
  - Một số thao tác chưa hiệu quả: thêm phần tử vào cuối danh sách, thêm vào trước, xóa một nút ở giữa.. Độ phức tạp thuật toán O(n) thay vì O(1)
- Danh sách liên kết đôi: Mỗi phần tử có 2 con trỏ, giúp truy cập phần tử trước và sau.
  - Cải thiện hiệu năng các thao tác trên.



Xây dựng danh sách liên kết đôi: Định nghĩa kiểu

```
typedef ... elementtype; // INFO field definition
struct node_t {
   elementtype element;
   struct node_t  *next;
   struct node_t  *prev;
};
typedef struct node_t node;
typedef node* doublelist;
doublelist root, tail, cur;
```

Khởi tạo danh sách và kiểm tra danh sách rỗng

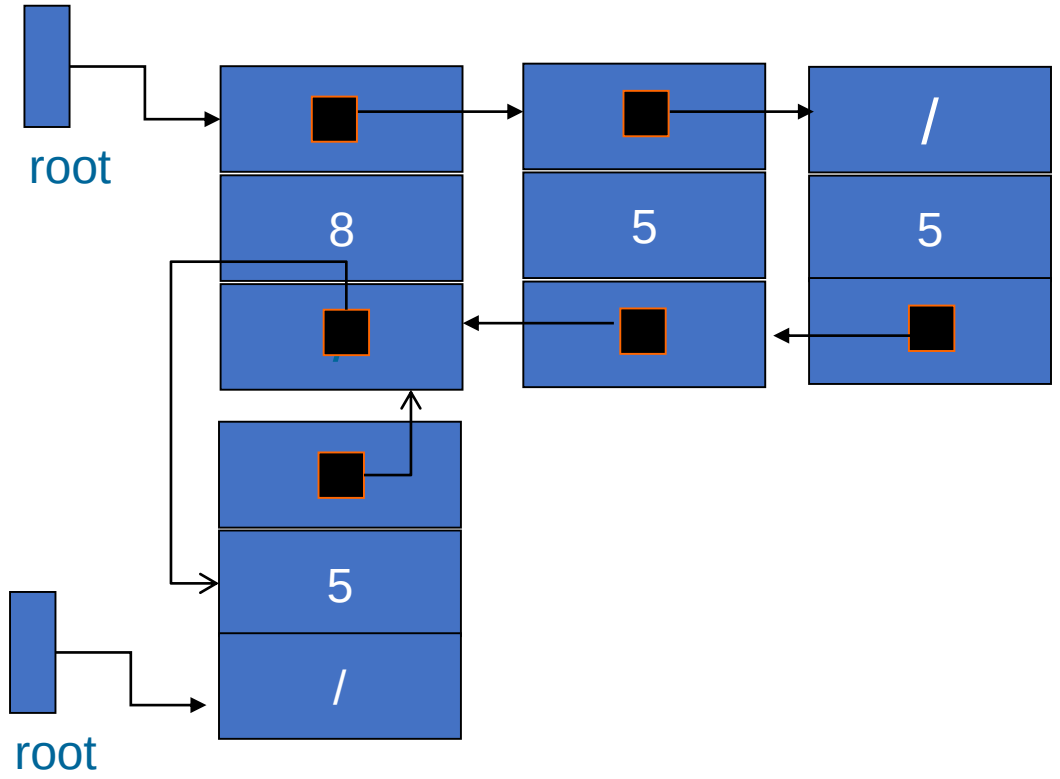
```
void MakeNull_List (doublelist *root, doublist *tail, doublist
  *cur){
  (*root)= NULL;
   (*tail)= NULL; (*cur)= NULL;
}
int isEmpty (doublelist root){
  return (root==NULL);
}
```

Tạo một nút mới

```
node* makeNewNode(elementtype ele){
 node* new = (node*) malloc(sizeof(node));
 new->element=ele;
 new->next =NULL;
 new->prev =NULL;
 return new;
}
```

Thêm phần tử vào đầu danh sách (phiên bản 1: con trỏ toàn cục)

```
void insertAtHead(elementtype ele){
  node* new = makeNewNode(ele);
  if (root==NULL)      {
       root= new;
       tail= new;
       cur = new;
       return;
  }
  new->next = root;
  root->prev = new;
  root = new;
  cur = root;
}
```



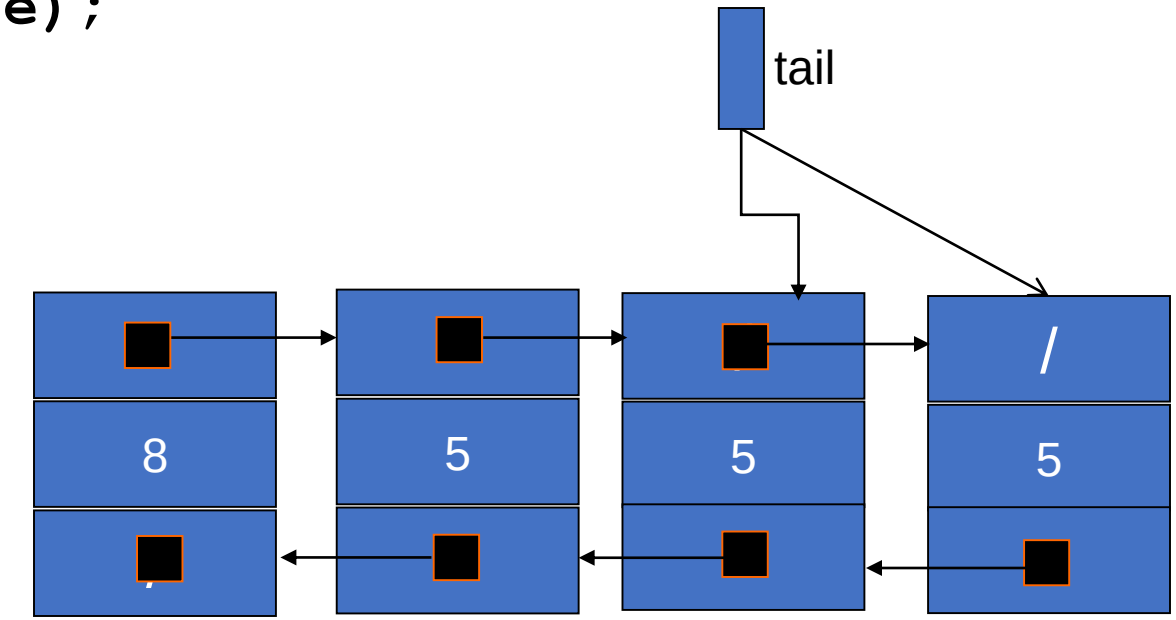
Thêm phần tử vào đầu danh sách (phiên bản 2: sử dụng tham số con trỏ)

```
void insertAtHead(elementtype e, doublelist *root, doublelist
  *tail, doublelist *cur){
  node* new = makeNewNode(e);
  if (*root==NULL) {
      *root=new; *tail= new;
      *cur=new; return;
   }
  new->next = *root;
  (*root)->prev = new;
  *root = new;
  *cur = *root;
}
```

call in main(): insertAtHead(e, &root, &tail, &cur);

Thêm phần tử vào cuối (append-pushBack)

```
void append(elementtype ele){
  node* new = makeNewNode(ele);
  if (tail==NULL){
       root= new;
       tail= new;
       cur = new;
       return;
  }
  tail->next = new;
  new->prev = tail;
  tail = new;
  cur = tail;
}
```



Thêm phần tử vào cuối (phiên bản 2)

```
void append(elementtype e, doublelist *root, doublelist *tail, doublelist
  *cur){
  node* new = makeNewNode(e);
  if (*tail==NULL){
        *root= new;
        *tail= new;
        *cur = new;
        return;
  }
  (*tail)->next = new;
  new->prev = *tail;
  *tail = new;
  *cur = tail;
}
```

call in main(): append(e, &root, &tail, &cur);

Thêm phần tử vào sau phần tử hiện hành (Phiên bản 1)

```
void insertAfterCurrent(elementtype ele){
  node *new = makeNewNode(ele);
  if(root == NULL){
      root = new; tail = new; cur = new;
  } else if (cur == NULL) {
       printf("Current pointer is NULL.\n"); return;
  } else {
    new->next = cur->next;
    if (cur->next!=NULL) cur->next->prev = new;
    else tail = new;
    cur->next = new;
    new->prev = cur;
    cur = new;
  }
}
```

Thêm phần tử vào sau phần tử hiện hành (Phiên bản 2)

```
void insertAfterCurrent(elementtype e, doublelist *root, doublelist
  *tail, doublelist *cur) {
   node* new = makeNewNode(e);
     if ( *root == NULL ) {
       *root = new; *tail=new; *cur = *root;
  } else {
       new->next=(*cur)->next;
      if ((*cur)->next!=NULL)(*cur)->next->prev=new;
       else *tail=new;
       (*cur)->next = new;
       new->prev=*cur;
       *cur = new;
  }
}
```

Thêm phần tử vào trước phần tử hiện hành (Phiên bản 1)

```
void insertBeforeCurrent(elementtype e) {
  node* new = makeNewNode(e);
     if ( root == NULL ) { /* if there is no element */
    root = new;
    tail = new;
      cur = new;
  } else {
       new->next=cur;
       if (cur->prev!=NULL) cur->prev->next=new;
       else root = new;
       new->prev=cur->prev;
       cur->prev = new;
       cur = new;
  }
}
```

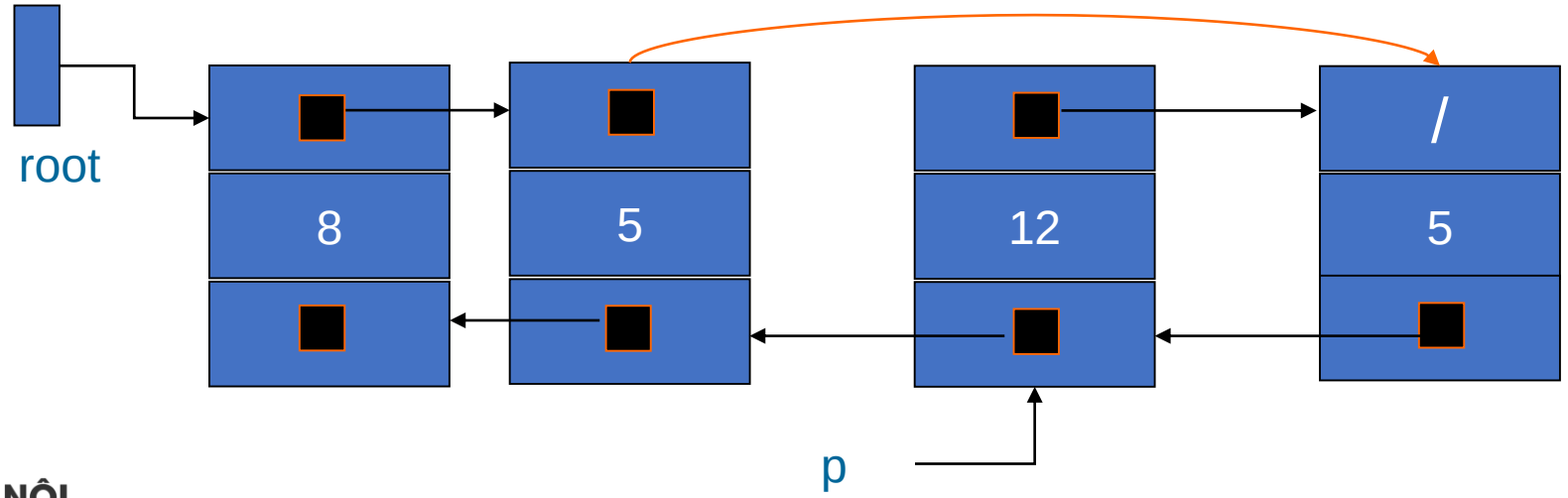
Thêm phần tử vào một vị trí tuyệt đối

```
void insertAtPosition(elementtype e, int position){
  cur = root; int i;
  if(position <= 0) { //pos <= 0 : insert at Head
    insertBeforeCurrent(e);
    return;
  }
  //move pointer to node at position
  for (i = 1; i < position; i++)
    if(cur->next != NULL) cur= cur->next;

  insertAfterCurrent(e);
}
```

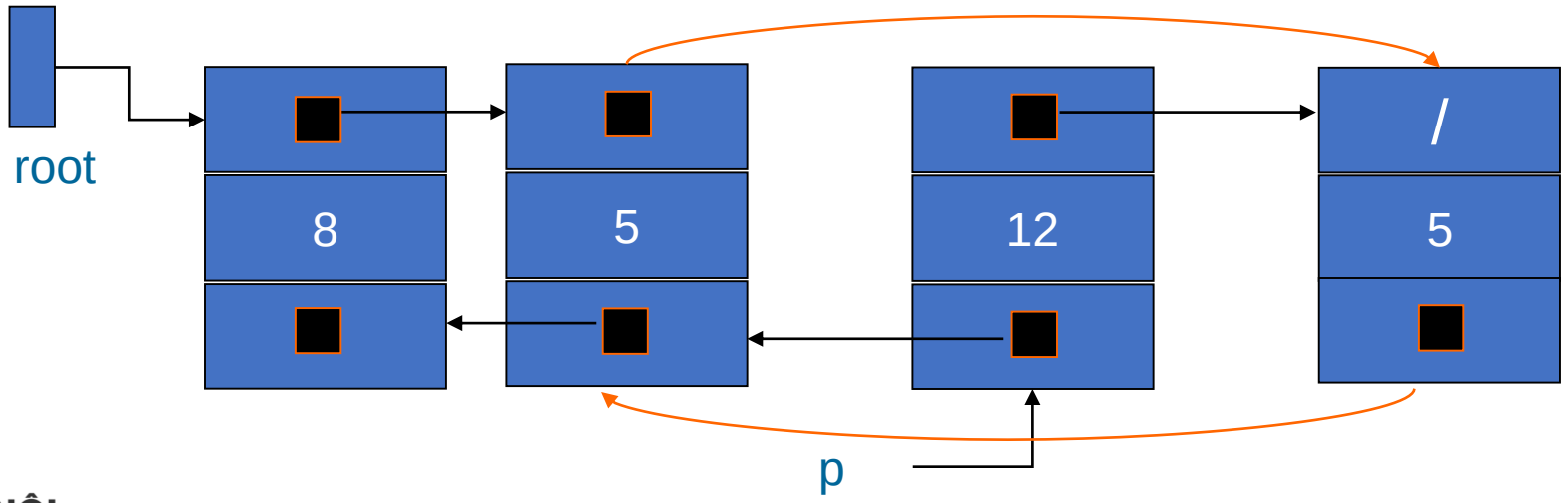
Xóa một phần tử được trỏ bởi p

```
void Delete_List (doublelist p, doublelist *root){
 if (*root == NULL) printf("Empty list");
 else {
   if (p==*root) (*root)=(*root)->Next; //Delete first element
    else p->prev->next=p->next;
    if (p->next!=NULL) p->next->prev=p->prev;
    free(p);
 }
}
```



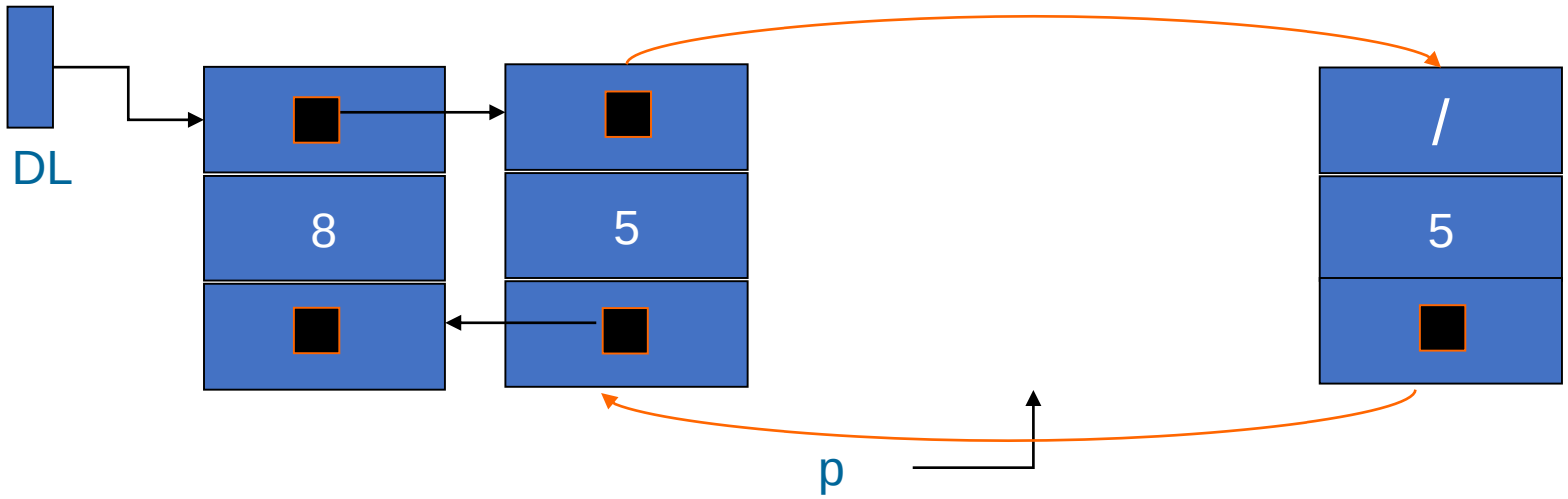
Xóa một phần tử được trỏ bởi p

```
void Delete_List (doublelist p, doublelist *root){
 if (*root == NULL) printf("Empty list");
 else {
   if (p==*root) (*root)=(*root)->Next; //Delete first element
    else p->prev->next=p->next;
    if (p->next!=NULL) p->next->prev=p->prev;
    free(p);
 }
}
```



Xóa một phần tử được trỏ bởi p

```
void Delete_List (DoubleList p, DoubleList *DL){
 if (*DL == NULL) printf("Empty list");
 else {
   if (p==*DL) (*DL)=(*DL)->next; //Delete first element
    else p->prev->next=p->next;
    if (p->next!=NULL) p->next->prev=p->prev;
    free(p);
 }
}
```



Phiên bản khác

```
void Delete_List (doublelist p, doublelist *root, doublelist *cur, doublelist
  *tail){
 if (*root == NULL) printf("Empty list");
 else {
   if (p==*root) (*root)=(*root)->next;
    //Delete first element
    else p->prev->next=p->next;
    if (p->next!=NULL) p->next->prev=p->prev;
    else //p is tail
       *tail = p->prev;
    }
    *cur = p->prev;
    free(p);
 }
}
```

Đảo ngược danh sách liên kết đôi

```
node* list_reverse(node *root) {
  node *cur, *temp;
  cur = root;
  if (cur==NULL) return NULL;
  while (cur != NULL) {
       temp = cur->prev; //temp point to previous node
       cur->prev = cur->next; // point inversely to the next node
       cur->next = temp; // point next to previous node stored in temp
       cur = cur->prev;
  }
  if(temp!=NULL) temp=temp->prev;
  return temp;
}
```

Một số hàm hữu ích nên xây dựng

- Implement the necessary functions
  - deleteFirstElement
  - deleteCurrentElement
  - deleteAtPosition..

  - and new one: deleteLastElement
  - traverseListFromTail

Xóa phần tử cuối (đuôi) danh sách

```
void deleteLastElement(){
  node* del = tail;
  if (del == NULL) return;
  tail = tail->prev;
  if (tail==NULL){ /* one node in the list ==> empty list */
      root=NULL;
  } else tail->next = NULL;
  free(del);
  cur = tail;
}
```

Duyệt theo chiều ngư

```
void traverseListFromTail(node *tail){
 node *p;
 for (p = tail; p!= NULL; p = p->prev )
   displayNode(p);
}
```

Phân tách mã nguồn chương trình/thư viện cấu trúc dữ liệu

- Create lib.h
  - Type declaration
  - Function prototype
- Create lib.c
  - #include “lib.h”
  - Function Implementation
- Main Program: pro.c
  - #include “lib.h”

- Compile
- gcc – o ex pro.c lib.c

Another way:

gcc –c lib.c

gcc –c pro.c

gcc –o lib.o pro.o

ex

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !