HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

C Programming Basic

List

Content

- Basic operations on linked lists
- Profile management problem
- Implementation of single linked lists
- Implementation of doubly linked lists

Basic operations on linked lists

- Each node of a linked is defined as follows

```
typedef struct Node{
    int value;
    struct Node* next;
}Node;
```

- Implement following operations
  - Node* insertLast(Node* h, int v);// insert a node at that last position
  - Node* removeFirst(Node* h, int v);// remove a first node having value v
  - Node* removeAll(Node* h, int v);// remove all nodes having value v
  - int count(Node* h);// count number of nodes
  - Node* reverse(Node* h);// reverse the linked list

Basic operations on linked lists

- Define data structures

```
#include <stdio.h>
typedef struct Node{
    int value;
    struct Node* next;// point to the next
      //element of the current element
}Node;

Node*makeNode(int v){// allocate memory for a new node
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v; p->next = NULL;
    return p;
}
```

Basic operations on linked lists

- Insert a node to the end of a linked list (use and not use recursion)

```
Node* insertLast(Node* h, int v){
      Node* p = h;
     if(h == NULL){
            return makeNode(v);
      }
      // general case
      while(p->next != NULL)
            p = p->next;

      Node* q = makeNode(v);
      p->next = q;
      return h;
}
```

```
Node* insertLastRecursive(Node* h, int v){
    if(h == NULL){
        return makeNode(v);
    }
    h->next = insertLastRecursive(h->next, v);
    return h;
}
```

Basic operations on linked lists

- Remove a first node having value v (not use recursion)

```
Node* removeNode(Node* h, int v){
    Node* p = h;
    if(h == NULL) return NULL;
    if(h->value == v
        Node* tmp = h; h = h->next;
        free(tmp);  return h;
    }
    while(p->next != NULL){
        if(p->next->value == v) break;
        p = p->next;
    }
    if(p->next != NULL){
        Node* q = p->next; p->next = q->next; free(q);
    }
    return h;
}
```

Basic operations on linked lists

- Remove a first node having value v (use recursion)

```
Node* removeNodeRecursive(Node* h, int v){
    if(h == NULL) return NULL;
    if(h->value == v){
        Node* tmp = h; h = h->next; free(tmp); return h;
    }
    h->next = removeNodeRecursive(h->next, v);
    return h;
}
```

Basic operations on linked lists

- Remove all nodes having value v (use recursion)

```
Node* removeAll(Node* h, int v){
    // remove all nodes having value v from the linked list headed by h
    if(h == NULL) return NULL;
    if(h->value == v){
        Node* tmp = h; h = h->next; free(tmp);
        h = removeAll(h,v); // continue to remove other elements having value v
        return h;
    }
    h->next = removeAll(h->next,v);
    return h;
}
```

Basic operations on linked lists

- Count number of nodes of a linked list (user and not use recursion)

```
int countRecursive(Node* h){
    if(h == NULL) return 0;
    return 1+countRecursive(h->next);
}
```

```
int count(Node* h){
    int cnt = 0;
    Node* p = h;
    while(p != NULL){
        cnt += 1;
        p = p->next;
    }
    return cnt;
}
```

Basic operations on linked lists

- Reverse a linked list

```
Node* reverse(Node *h){
    Node* p = h;
    Node* pp = NULL;
    Node* np = NULL;
    while(p != NULL){
        np = p->next;
        p->next = pp;
        pp = p;
        p = np;
    }
    return pp;
}
```

Profile management problem

- A student profile consists of
  - name: name of the student
  - email: email of the student
- Write a program running in an interactive mode
  - Load data from a text file into memory establishing a list
  - Print the information of students
  - Add a new profile at the end of the list
  - Remove a profile
  - Find a profile given a name
  - Store the list into an external text file

Profile management problem

- Data structure

```
#include <stdio.h>
#define MAX_L 256

typedef struct Profile{
    char name[MAX_L];
    char email[MAX_L];
    struct Profile* next;
}Profile;

Profile* first, *last;
```

Profile management problem

- Memory allocation

```
Profile* makeProfile(char* name, char* email){
    Profile* node = (Profile*)malloc(sizeof(Profile));
    strcpy(node->name,name);
    strcpy(node->email,email);
    node->next = NULL;
    return node;
}
void initList(){
    first = NULL; last = NULL;
}
int listEmpty(){
    return first == NULL && last == NULL;
}
```

Profile management problem

- Insert a profile to the end of the list

```
void insertLast(char* name, char* email){
    Profile* profile = makeProfile(name,email);
    if(listEmpty()){
            first = profile; last = profile;
    }else{
        last->next = profile; last = profile;
    }
}
void printList(){
    for(Profile* p = first; p != NULL; p = p->next)
        printf("%s, %s\n",p->name, p->email);
}
```

Profile management problem

- Remove a profile given a name

```
Profile* removeProfile(Profile* f, char* name){
    if(listEmpty()) return NULL;
    if(strcmp(f->name,name) == 0){
        Profile* tmp = f->next;
        free(f);
        if(tmp == NULL) last = NULL;
        return tmp;
    }else{
        f->next = removeProfile(f->next,name);
        return f;
    }
}
```

Profile management problem

- Load data from file to the memory, establish the linked list

```
void load(char* filename){
    FILE* f = fopen(filename,"r");
    if(f == NULL) printf("Load data -> file not found\n");

    while(!feof(f)){
        char name[256], email[256];
        fscanf(f,"%s%s",name, email);
        insertLast(name,email);
    }
    fclose(f);
}
```

Profile management problem

```
void processFind(){
    char name[256];
    scanf("%s",name);
    Profile* profile = NULL;
    for(Profile* p = first; p != NULL; p = p->next){
        if(strcmp(p->name,name)==0){
            profile = p; break;
        }
    }
    if(profile == NULL){
        printf("NOT FOUND profile %s\n",name);
    }else{
        printf("FOUND profile %s, %s\n",profile->name,profile->email);
    }
}
```

Profile management problem

```
void processLoad(){
  char filename[256];
  scanf("%s",filename);
  load(filename);
}
void processStore(){
  char filename[256];
  scanf("%s",filename);
  FILE* f = fopen(filename,"w");
  for(Profile* p = first; p != NULL; p = p->next){
        fprintf(f,"%s %s",p->name,p->email);
        if(p->next != NULL) fprintf(f,"\n");
  }
  fclose(f);
}
```

Profile management problem

```
void processInsert(){
    char name[256], email[256];
    scanf("%s%s",name,email);
    insertLast(name,email);
}
void processRemove(){
    char name[256];
    scanf("%s",name);
    first = removeProfile(first,name);
}
```

Profile management problem

```
int main(){
    initList();
    while(1){
        printf("Enter command: ");
        char cmd[256];
        scanf("%s",cmd);
        if(strcmp(cmd,"Quit")==0) break;
        else if(strcmp(cmd,"Load")==0) processLoad();
        else if(strcmp(cmd,"Print")==0)  printList();
        else if(strcmp(cmd,"Find")==0) processFind();
        else if(strcmp(cmd,"Insert")==0) processInsert();
        else if(strcmp(cmd,"Remove")==0) processRemove();
        else if(strcmp(cmd,"Store")==0) processStore();
    }
}
```

Profile management problem

- Data structure

```
#include <stdio.h>
#define MAX_L 256

typedef struct Profile{
    char name[MAX_L];
    char email[MAX_L];
    struct Profile* next;// pointer to the next element
    struct Profile* prev;// pointer to the predecessor
}Profile;

Profile* first, *last;
```

Profile management: doubly linked list

```
Profile* makeProfile(char* name, char* email){
    Profile* node = (Profile*)malloc(sizeof(Profile));
    strcpy(node->name,name);
    strcpy(node->email,email);
    node->next = NULL;
    node->prev = NULL;
    return node;
}
```

Profile management: doubly linked list

```
void initList(){
    first = NULL; last = NULL;
}
int listEmpty(){
    return first == NULL && last == NULL;
}
void printListLeft2Right(){
    for(Profile* p = first; p != NULL; p = p->next)
        printf("%s, %s\n",p->name, p->email);
}
void printListRight2Left(){
    for(Profile* p = last; p != NULL; p = p->prev)
        printf("%s, %s\n",p->name, p->email);
}
```

Profile management: doubly linked list

```
void insertLast(char* name, char* email){
    Profile* profile = makeProfile(name,email);
    if(listEmpty()){
            first = profile; last = profile;
    }else{
        last->next = profile;  profile->prev = last;
        last = profile;
    }
}
```

Profile management: doubly linked list

```
Profile* find(char*name){
    for(Profile* p = first; p != NULL; p = p->next){
        if(strcmp(p->name,name)==0){
                return p;
        }
    }
    return NULL;
}
```

Profile management: doubly linked list

```
void removeProfile(char* name){
    if(listEmpty()) return NULL;
    Profile* profile = find(name);
    if(profile == NULL){
        printf("NOT FOUND %s\n",name);
    }else{
        Profile* left = profile->prev;
        Profile* right = profile->next;
        if(left != NULL)  left->next = right;
        if(right != NULL) right->prev = left;
        if(left == NULL) first = right;
        if(right == NULL) last = left;
        free(profile);
    }
}
```

Profile management: doubly linked list

```
void load(char* filename){
    FILE* f = fopen(filename,"r");
    if(f == NULL) printf("Load data -> file not found\n");
    initList();
    while(!feof(f)){
        char name[256], email[256];
        fscanf(f,"%s%s",name, email);
        insertLast(name,email);
        printf("insert %s, %s\n",name,email);
    }
    fclose(f);
}
```

Profile management: doubly linked list

```
void processFind(){
    char name[256];
    scanf("%s",name);
    Profile* profile = find(name);
    if(profile == NULL){
        printf("NOT FOUND profile %s\n",name);
    }else{
        printf("FOUND profile %s, %s\n",profile->name,profile->email);
    }
}
```

Profile management: doubly linked list

```
void processLoad(){
  char filename[256];
  scanf("%s",filename);
  load(filename);
}
void processStore(){
  char filename[256];
  scanf("%s",filename);
  FILE* f = fopen(filename,"w");
  for(Profile* p = first; p != NULL; p = p->next){
        fprintf(f,"%s %s",p->name,p->email);
        if(p->next != NULL) fprintf(f,"\n");
  }
  fclose(f);
}
```

Profile management: doubly linked list

```
void processInsert(){
    char name[256], email[256];
    scanf("%s%s",name,email);
    insertLast(name,email);
}
void processRemove(){
    char name[256];
    scanf("%s",name);
    removeProfile(name);
}
```

Profile management: doubly linked list

```
void processPrintList(){
    printf("Danh sach tu trai qua phai\n");
    printListLeft2Right();
    printf("Danh sach tu phai qua trai\n");
    printListRight2Left();
}
void finalize(){
    Profile* p = first;
    while(p != NULL){
        Profile* np = p->next;
        free(p);
        p = np;
    }
}
```

Profile management: doubly linked list

```
int main(){
    initList();
    while(1){
        printf("Enter command: ");
        char cmd[256];
        scanf("%s",cmd);
        if(strcmp(cmd,"Quit")==0) break;
        else if(strcmp(cmd,"Load")==0) processLoad();
        else if(strcmp(cmd,"Print")==0)  processPrintList();
        else if(strcmp(cmd,"Find")==0) processFind();
        else if(strcmp(cmd,"Insert")==0) processInsert();
        else if(strcmp(cmd,"Remove")==0) processRemove();
        else if(strcmp(cmd,"Store")==0) processStore();
    }
    finalize();
}
```

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

soict.hust.edu.vn/ fb.com/groups/soict