BÀI TẬP LINKED LIST

LÝ QUANG THẮNG - 22110202

CÂU 1:

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

struct Node {
    int data;
    struct Node* next;
};
typedef struct Node Node;

struct LinkedList {
    Node* head;
};
typedef struct LinkedList LinkedList;

void init(LinkedList *list) {
        list->head = NULL;
}

Node* makeNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

int size(LinkedList *list) {
    int size = 0;
    Node* tmp = list->head;
    while(tmp != NULL) {
        size += 1;
        tmp = tmp->next;
    }
```

```
    return size;
}

void insertFirst(int data, LinkedList *list) {
    Node* newNode = makeNode(data);
    newNode->next = list->head;
    list->head = newNode;
}

void insertK(int data, LinkedList *list, int pos) {
    int n = size(list);
    if(pos < 1 || pos > n + 1) return;
    if(pos == 1) {
        insertFirst(data, list);
    }else {
        Node* tmp = list->head;
        for(int i = 1;i <= pos - 2; i++) {
            tmp = tmp->next;
        }
        Node *newNode = makeNode(data);
        newNode->next = tmp->next;
        tmp->next = newNode;
    }
}

void deleteFirst(LinkedList* list) {
    if(list->head == NULL)
        return;
    Node* tmp = list->head;
    list->head = list->head->next;
    free(tmp);
}

void deleteK(LinkedList *list, int pos) {
    int n = size(list);
    if(pos < 1 || pos > n) return; // ko hop le
    if(pos == 1) deleteFirst(list);
    else {
        Node* tmp = list->head;
        for(int i = 1; i <= pos - 2; i++) {
            tmp = tmp->next;
```

```
        }
        // tmp = pos - 1
        Node* kth = tmp->next; //node thu pos
        // cho pos - 1 ket noi voi node thu pos + 1
        tmp->next = kth->next;
        free(kth);
    }
}

void printList(LinkedList *list) {
    Node* tmp = list->head;
    while(tmp) {
        printf("%d ", tmp->data);
        tmp = tmp->next;
    }
    printf("\n");
}

int main() {
    LinkedList list;
    init(&list);
    int a[] = {2, 3, 4, 5, 6, 7};
    int n = sizeof(a)/sizeof(int);
    for(int i = 0; i < n; i++) {
        insertK(a[i], &list, size(&list) + 1);
    }
    // cau a:
    insertK(1, &list, 2);
    printList(&list);

    // cau b:
    deleteK(&list, 2);
    printList(&list);
    return 0;
}
```

CÂU 2:

```
#include<stdio.h>
#include<stdlib.h>
```

```c
#include<time.h>

typedef struct NodeType {
    int data;
    struct NodeType* next;
} Node;

typedef struct LinkedListType {
    Node* head;
} LinkedList;

void init(LinkedList* list) {
    list->head = NULL;
}

Node* makeNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertTail(int data, LinkedList *list) {
    Node* newNode = makeNode(data);
    if(list->head == NULL) {
        list->head = newNode;
    }else {
        Node* tmp = list->head;
        while(tmp->next != NULL) {
            tmp = tmp->next;
        }
        tmp->next = newNode;
    }
}

void printList(LinkedList *list) {
    Node* tmp = list->head;
    while(tmp != NULL) {
        printf("%d ", tmp->data);
        tmp = tmp->next;
    }
```

```
    printf("\n");
}

void insert(LinkedList *list, int data) {
    Node* newNode = makeNode(data);
    if(list->head == NULL) {
        list->head = newNode;
        return;
    }
    if(list->head->data <= data) {
        newNode->next = list->head;
        list->head = newNode;
        return;
    }
    Node* tmp = list->head;
    while(tmp->next != NULL && tmp->next->data > data) {
        tmp = tmp->next;
    }
    newNode->next=tmp->next;
    tmp->next = newNode;
}

void delete1(LinkedList *list, int value) {
    if(list->head == NULL) {
        return;
    }
    if(list->head->data == value) {
        Node* tmp = list->head;
        list->head = list->head->next;
        free(tmp);
        return;
    }
    Node* truoc = NULL;
    Node* sau = list->head;

    while(sau->next != NULL && sau->data < value) {
        truoc = sau;
        sau = sau->next;
    }

    if(sau->data == value) {
```

```
        truoc->next = sau->next;
        free(sau);
    }

    return;
}



int main() {
    LinkedList list;
    init(&list);
    int a[] = {7, 5, 4};
    // gia su list ban dau co nhung phan tu nhu mang a
    for(int i = 0; i < 3; i++) {
        insertTail(a[i], &list);
    }
    // 2 a)
    printf("CAU 2a\n");
    printf("List ban dau: "); printList(&list);
    int chen[] = {10, 2, 6};
    for(int i = 0; i < 3; i++) {
        printf("List sau khi chen %d: ", chen[i]);
        insert(&list, chen[i]);
        printList(&list);
    }

    printf("CAU 2b\n");
    init(&list);
    int b[] = {4, 5, 9, 10, 20};
    for(int i = 0; i < 5; i++) {
        insertTail(b[i], &list);
    }
    printf("List ban dau: "); printList(&list);
    int xoa[] = {100, -1, 3, 4, 20, 9};
    for(int i = 0; i < 6; i++) {
        printf("List sau khi xoa %d: ", xoa[i]);
        delete1(&list, xoa[i]);
        printList(&list);
    }
```

```
    return 0;
}
```

CÂU 3:

3.1 CÀI ĐẶT STACK = LINKED LIST

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

struct Node {
    int data;
    struct Node* next;
};
typedef struct Node Node;

Node* makeNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Stack {
    Node* top;
};
typedef struct Stack Stack;

void init(Stack *st) {
    st->top = NULL;
}

void push(Stack *st, int data) {
    Node* newNode = makeNode(data);
    if(st->top == NULL) {
        st->top = newNode;
    }else {
        newNode->next = st->top;
        st->top = newNode;
```

```
    }
}

void pop(Stack *st) {
    if(st->top == NULL) return;
    Node* tmp = st->top;
    st->top = st->top->next;
    free(tmp);
}

int top(Stack *st) {
    if(st->top != NULL) {
        return st->top->data;
    }
}

int size(Stack st) {
    int cnt = 0;
    while(st.top != NULL) {
        ++cnt;
        st.top = st.top->next;
    }
    return cnt;
}

void duyet(Stack st) {
    while(st.top != NULL) {
        printf("%d ", st.top->data);
        st.top = st.top->next;
    }
    printf("\n");
}

int main() {
    Stack st;
    init(&st);
    while(1) {
        printf("---------------\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Top\n");
```

```
        printf("4. Size\n");
        printf("5. Duyet\n");
        printf("0. OUT\n");
        printf("---------------\n");
        int lc; scanf("%d", &lc);
        if(lc == 1) {
            int x; printf("nhap data: "); scanf("%d", &x);
            push(&st, x);
        }else if(lc == 2) {
            pop(&st);
        }else if(lc == 3) {
            if(st.top != NULL) {
                printf("Top: %d\n", top(&st));
            }
        }else if(lc == 4) {
            printf("Size: %d\n", size(st));
        }else if(lc == 5) {
            duyet(st);
        }else if(lc == 0) break;
    }


    return 0;
}
```

3.2 CÀI ĐẶT QUEUE = LINKED LIST

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

typedef struct Nodetype{
    int data;
    struct Nodetype* next;
}Node;

Node* makeNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
```

```
    return newNode;
}

typedef struct Queuetype {
    Node* head;
    Node* tail;
}Queue;

void init(Queue *q) {
    q->head = NULL;
}


int isEmpty(Queue *q) {
    if(q->head == NULL) return 1;
    return 0;
}

void push(Queue *q, int data) {
    Node* newNode = makeNode(data);
    if(q->head == NULL) {
        q->head = newNode;
    }else {
        Node * tmp = q->head;
        while(tmp->next != NULL)
            tmp = tmp->next;
        tmp->next = newNode;
    }
}

void pop(Queue *q) {
    if(q->head == NULL) return;
    Node* tmp = q->head;
    q->head = q->head->next;
    free(tmp);
}

int size(Queue q) {
    int cnt = 0;
    while(q.head != NULL) {
        ++ cnt;
```

```
        q.head = q.head->next;
    }
    return cnt;
}

int front(Queue q) {
    return q.head->data;
}

void duyet(Queue q) {
    while(q.head != NULL) {
        printf("%d ", q.head->data);
        q.head = q.head->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    init(&q);

    while(1) {
        printf("---------------\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Front\n");
        printf("4. Size\n");
        printf("5. Duyet\n");
        printf("0. OUT\n");
        printf("---------------\n");
        int lc; scanf("%d", &lc);
        if(lc == 1) {
            int x; printf("nhap data: "); scanf("%d", &x);
            push(&q, x);
        }else if(lc == 2) {
            pop(&q);
        }else if(lc == 3) {
            if(q.head != NULL) {
                printf("Top: %d\n", front(q));
            }
        }else if(lc == 4) {
```

```
            printf("Size: %d\n", size(q));
        }else if(lc == 5) {
            duyet(q);
        }else if(lc == 0) break;
    }
    return 0;
}
```

CÂU 4:

HÀM insertK viết bằng đệ quy:

```
void insertK(int data, Node** curNode, int k) {
    if(k == 1) {
        Node* newNode = makeNode(data);
        newNode->next = *(curNode);
        (*curNode) = newNode;
        return;
    }

    if(k < 0 || curNode == NULL) return;

    if(k - 2 == 0) {
        Node* newNode = makeNode(data);
        newNode->next = (*curNode)->next;
        (*curNode)->next = newNode;
        return;
    }
    insertK(data, &(*curNode)->next, k - 1);
}
```

Trong hàm main ta truyền vào địa chỉ con trỏ. Vì hàm cần con trỏ cấp 2.
Ví dụ

```
        insertK(10, &list.head, 3);
```

Ta có thể phân tích hàm đệ quy trên theo quy trình 4 bước

Có 2 trường hợp tùy vào vị trí cần chèn

K là vị trí ở giữa list và hợp lệ

- Base case: Khi vị trí `k - 2 == 0` ta thực hiện chèn phần tử
- Kết quả base case: List đã được chèn vào vị trí k thành công
- Trước base case(nếu có): lúc `k - 2 == 1` Vì nếu list có 4 phần tử và cần chèn vào vị trí `k = 3` thì sẽ có trước base case này. Ngược lại `k = 2` thì không.
- Trước trước base case(nếu có) `k - 2 == 2` Vì nếu list có 4 phần tử và cần chèn vào vị trí `k = 4` thì sẽ có trước base case này. Ngược lại `k = 3` thì không.

K là vị trí đầu list nghĩa là k = 1

- Base case là

```
    if(k == 1) {
        Node* newNode = makeNode(data);
        newNode->next = *(curNode);
        (*curNode) = newNode;
        return;
    }
```

- Kết quả base case: List đã được chèn như ý muốn
- Trước base case: Không có
- Trước trước base case: Không có

LƯU Ý:

```
        if(k < 0 || curNode == NULL) return;
```

dùng để check trường hợp `k` không hợp lệ ví dụ như `k <= 0` hoặc `k` lớn hơn độ dài list + 1.

HÀM deleteK viết bằng đệ quy

```
void deleteK(Node** curNode, int k) {
    // xoa dau`
    if(k == 1) {
        if(*curNode == NULL) return;
        Node* tmp = *curNode;
        (*curNode) = (*curNode)->next;
        free(tmp);
        return;
    }
    if(*curNode == NULL || k < 0) return; // truong hop vi tri
k hop le
    if(k - 2 == 0) {
        Node* kth = (*curNode)->next;
        (*curNode)->next = kth->next;
        free(kth);
        return;
    }
    deleteK(&(*curNode)->next, k - 1);
}
```

Phân tích tương tự hàm insert

CÂU 5:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

struct Node{
    int soPhong, soTang;
    char ten;
    struct Node* next;
};
typedef struct Node Node;

struct LinkedList {
    Node* head;
};
typedef struct LinkedList LinkedList;
```

```
void init(LinkedList *list) {
    list->head = NULL;
}
// khai bao thong tin //
int khachSan[1000][1000];// 17 tang va moi tang 12 phong
int soPhongKhachSan, soTangKhachSan;
int isFull[1000]; // check xem tang thu i co full chua
LinkedList doan[11]; // 10 doan
int sodoan = 10;
// --------------------//

void initKhachSan() {
    scanf("%d %d", &soTangKhachSan, &soPhongKhachSan);
    for(int i = soTangKhachSan; i >= 1; i--) {
        for(int j = 1; j <= soPhongKhachSan; j++) {
            scanf("%d", &khachSan[i][j]);
        }
    }
    printf("DA NHAP XONG THONG TIN PHONG KHACH SAN!\n");
}

Node* makeNode(int phong, int tang, char Ten) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->soPhong = phong;
    newNode->soTang = tang;
    newNode->ten = Ten;
    newNode->next = NULL;
    return newNode;
}

int size(LinkedList *list) {
    int cnt = 0;
    Node* tmp = list->head;
    while(tmp) {
        cnt++;
        tmp = tmp->next;
    }
    return cnt;
}
```

```
void insCus(LinkedList *list, int phong, int tang, char Ten) {
    Node* newNode = makeNode(phong, tang, Ten);
    if(list->head == NULL) {
        list->head = newNode;
    }else {
        Node* tmp = list->head;
        while(tmp->next) {
            tmp = tmp->next;
        }
        tmp->next = newNode;
    }
}

void deleteFirst(LinkedList* list) {
    if(list->head == NULL)
        return;
    Node* tmp = list->head;
    list->head = list->head->next;
    free(tmp);
}

void deleteCus(LinkedList *list, int pos) {
    int n = size(list);
    if(pos < 1 || pos > n) {
        printf("So thu tu nay khong hop le\n");
        return;
    }
    if(pos == 1) deleteFirst(list);
    else {
        Node* tmp = list->head;
        for(int i = 1; i <= pos - 2; i++) {
            tmp = tmp->next;
        }
        // tmp = pos - 1
        Node* kth = tmp->next; //node thu pos
        printf("Thong tin:\nTen: %c, tang %d, phong %d\nDa
xoa!\n", kth->ten, kth->soTang, kth->soPhong);
        int phong = kth->soPhong, tang = kth->soTang;
        khachSan[tang][phong] = 0; isFull[tang] = 0;
        // cho pos - 1 ket noi voi node thu pos + 1
        tmp->next = kth->next;
```

```
        free(kth);
    }
}

void timPhongTrong(int *soTangTraVe, int*soPhongTraVe) {
    // bat dau tu tang 2
    // tim tang` chua full
    int i = 2;
    while(isFull[i]) {
        i++;
    }

    for(i; i <= soTangKhachSan; i++) {
        for(int j = 1; j <= soPhongKhachSan; j++) {
            if(khachSan[i][j] == 0) {
                khachSan[i][j] = 1;
                *soTangTraVe = i; *soPhongTraVe = j;
                return;
            }
        }
        isFull[i] = 1;
    }
}

void nhapThongTin(LinkedList *list) {
    printf("Nhap so luong thanh vien: ");
    int soluong; scanf("%d", &soluong);
    char danhSachTen[100] = {};
    // nhap danh sach ten doan
    char tmp; // dau cach
    scanf("%c", &tmp);
    for(int i = 1; i <= soluong; i++) {
        scanf("%c", &danhSachTen[i]);
        scanf("%c", &tmp);
    }
    // tim phong cho doan truong
    int phongCuaDoanTruong = -1;
    for(int i = 1; i <= soPhongKhachSan; i++) {
        if(!khachSan[1][i]) {
            khachSan[1][i] = 1;
            phongCuaDoanTruong = i;
```

```
            break;
        }
    }
    insCus(list, phongCuaDoanTruong, 1, danhSachTen[1]);
    // khoi tao danh sach thanh vien
    for(int i = 2; i <= soluong; i++) {
        int SOPHONG, SOTANG;
        timPhongTrong(&SOTANG, &SOPHONG);
        insCus(list, SOPHONG, SOTANG, danhSachTen[i]);
    }

}

void xuatThongTinDoan(LinkedList *list) {
    if(list->head == NULL) {
        // printf("Khong ton tai doan nay`\n");
        return;
    }
    Node* tmp = list->head;
    printf("DOAN TRUONG: %c  - tang: %d  - phong: %d\n", tmp-
>ten, tmp->soTang, tmp->soPhong);
    tmp = tmp->next;
    int stt = 1;
    while(tmp) {
        printf("THANH VIEN %d:  %c  - tang: %d  - phong: %d\n",
stt, tmp->ten, tmp->soTang, tmp->soPhong);
        tmp = tmp->next;
        stt++;
    }
    printf("----------------------\n");
}



void VANDE2() {
    int phong;
    printf("Nhap so phong cua doan truong ban muon tim: ");
scanf("%d", &phong);
    xuatThongTinDoan(&doan[phong]);
}
```

```
void VANDE3() {
    int soLuongKhachCheckOut;
    printf("Nhap so luong khach check out: "); scanf("%d",
&soLuongKhachCheckOut);
    int phongTrong[1000] = {}, tangTrong[1000] = {}; // muc
dich luu lai nhung phong da check out
    int idx = 1;
    while(soLuongKhachCheckOut--) {
        int tang, phong;
        printf("Nhap so tang va so phong khach le check-out:
"); scanf("%d %d", &tang, &phong);
        phongTrong[idx] = phong; tangTrong[idx] = tang; idx ++;
        khachSan[tang][phong] = 0; // danh dau phong luc nay da
trong
        isFull[tang] = 0; // danh dau tang luc nay k con full
    }
    int soLuongCheckIn; printf("Nhap so luong khach checkin:
"); scanf("%d", &soLuongCheckIn);
    idx--;
    while(soLuongCheckIn--) {
        if(idx == 0) {
            printf("DA HET PHONG!\n"); return;
        }
        printf("Khach hang moi thuoc doan so may: "); int
so_doan; scanf("%d", &so_doan);
        char tmp; // dau cach
        scanf("%c", &tmp);
        printf("Ten khach hang: "); char ten_khach; scanf("%c",
&ten_khach);
        insCus(&doan[so_doan], phongTrong[idx], tangTrong[idx],
ten_khach);
        idx--;
    }
}

void VANDE4() {
    printf("Nhap so doan co khach check-out: "); int so_doan;
scanf("%d", &so_doan);
    printf("Nhap so thu tu cua nguoi trong doan: "); int
so_thu_tu; scanf("%d", &so_thu_tu);
    deleteCus(&doan[so_doan], so_thu_tu + 1); // k tinh doan
```

```
truong
}

void VANDE5() {
    printf("Nhap so thu tu cua doan muon check out: "); int
so_doan; scanf("%d", &so_doan);
    if(doan[so_doan].head == NULL) {
        printf("Doan khong ton tai!\n"); return;
    }else {
        Node* tmp = doan[so_doan].head;
        int phong = tmp->soPhong, tang = tmp->soTang;
        khachSan[tang][phong] = 0; isFull[tang] = 0;
        tmp = tmp->next;
        while(tmp) {
            Node* xoa = tmp;
            int phong = tmp->soPhong, tang = tmp->soTang;
            khachSan[tang][phong] = 0; isFull[tang] = 0;
            tmp = tmp->next;
            free(xoa);
        }
        doan[so_doan].head = NULL;
        printf("DA XOA!\n");
    }
}

int main() {

    while(1) {
        printf("--------------QUAN LY KHACH SAN--------------
\n");
        printf("1. Khoi tao khach san va nhap thong tin
doan\n");
        printf("2. Tim phong cua doan truong\n");
        printf("3. Khach le check out, bo sung khach moi check
in\n");
        printf("4. Khach trong doan check out\n");
        printf("5. Ca doan check out va bao cao trang thai
phong\n");
        printf("6. Xuat thong tin tat ca\n");
        printf("0. Thoat\n");
        int lc; scanf("%d", &lc);
```

```
        if(lc == 1) {
            initKhachSan();
            // printf("Nhap so doan`: "); scanf("%d", &sodoan);
            for(int i = 1; i <= sodoan; i++) {
                init(&doan[i]);
                printf("Nhap thong tin doan %d:\n", i);
                nhapThongTin(&doan[i]);
            }
        }else if(lc == 2){
            VANDE2();
        }else if(lc == 3) {
            VANDE3();
        }else if(lc == 4) {
            VANDE4();
        }else if(lc == 5) {
            VANDE5();
            for(int i = 1; i <= soTangKhachSan; i++) {
                for(int j = 1; j <= soPhongKhachSan; j++) {
                    printf("%d ", khachSan[i][j]);
                }
                printf("\n");
            }
            printf("\n");
        }else if(lc == 6) {
            for(int i = 1; i <= sodoan; i++) {
                printf("DOAN THU %d\n", i);
                xuatThongTinDoan(&doan[i]);
            }
        }else continue;
    }

    return 0;
}
```

Test case Mẫu khách sạn

```
17 12
0 0 0 1 0 0 0 1 0 0 1 0
0 0 1 1 0 0 1 0 0 1 1 1
0 1 0 0 1 1 0 1 0 0 1 0
```

```
1 0 1 0 0 0 1 0 1 0 1 1
0 0 1 1 0 1 0 0 0 1 0 0
1 1 1 0 1 0 1 1 0 0 1 1
1 0 0 1 0 0 1 0 1 1 0 0
0 0 1 0 0 1 0 1 0 0 1 1
0 1 0 0 0 0 1 0 1 1 0 1
0 0 1 1 1 0 0 1 0 0 1 0
1 0 1 0 0 1 0 0 0 0 0 1
0 1 0 0 1 0 1 1 0 0 1 1
1 0 0 1 0 0 0 0 1 1 0 1
0 0 1 0 1 0 1 0 0 0 1 0
1 0 0 1 0 1 0 1 1 0 0 1
1 0 1 0 0 0 1 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 1 1
12 A B S E F J K L S W Q A
10 T M K L J S A Q C S
9 H J S S A S M Q D
13 S A Q D A S S A Q D S Q A
15 Q S A Q E D F S S A S X V B F
14 H A V A N T H A O T T H U S
12 P H L A M D S A T T H U
15 P P N H U N G S V E L E V E N
10 L P T R U O N G D S
10 C H G I A O K D L H
```

Sau đó hãy chọn lựa theo MENU. Cảm ơn