KỸ THUẬT LẬP TRÌNH C

Chương 4: Danh sách liên kết

1. Giới thiệu
2. Danh sách liên kết đơn
3. Danh sách liên kết kép

1. Giới thiệu

Danh sách liên kết là một tập hợp các nút mà giữa chúng có một sự nối kết với nhau thông qua vùng liên kết của chúng.

2. Danh sách liên kết đơn

a. Giới thiệu danh sách liên kết đơn

DSLKD là một danh sách liên kết trong đó mỗi nút là một cấu trúc có trường info chứa nội dung của nút và trường next là một con trỏ chỉ nút tiếp theo trong danh sách.

Thứ tự của các nút được thể hiện thông qua trường liên kết next: con trỏ đầu chỉ nút đầu danh sách, nút đầu chỉ nút thứ hai, …, nút cuối của danh sách là nút có trường next có giá trị NULL.

b. Khai báo cấu trúc một nút

```
struct node {
    int info;
    node *next;
};
//Khai báo kiểu con trỏ chỉ đến nút
typedef node *NODEPTR;
```

b. Các thao tác trên DSLKD

- **Cấp phát biến động làm nút cho danh sách**

```
NODEPTR GetNode() {
    NODEPTR p;
    p = (NODEPTR) malloc(sizeof(node));
    return p;
}
```

- **Giải phóng biến động đã cấp phát trước đó**

```
void FreeNode(NODEPTR p) {
    free(p);
}
```

- **Khởi động danh sách liên kết**

```
void Initialize(NODEPTR *plist) {
        *plist = NULL;
}
```

- **Kiểm tra danh sách có bị rỗng không**

```
int Empty(NODEPTR *plist) {
        return (*plist == NULL ? 1 : 0);
}
```

- **Xác định con trỏ của nút thứ i trong danh sách liên kết**

```
NODEPTR NodePointer(NODEPTR *plist, int i) {
    NODEPTR p;
    int pos;
    if(i < 0)  return NULL;
    else {
            p = *plist;
            pos = 0;
            while(p != NULL && pos < i) {
                    p = p->next;
                    pos++;
            }
            return p; }
}
```

- **Xác định nút trước của nút p trong danh sách liên kết.**

```
NODEPTR PreNode(NODEPTR *plist, NODEPTR p) {
    if(p == *plist)
            return NULL;
    else {
            NODEPTR q;
            q = *plist;
            //Di chuyển q đến nút trước nút p
            while(q != NULL && q->next != p)
                    q = q->next;
            return q;}}
```

- **Thêm nút có nội dung x vào đầu danh sách liên kết**

```
void Push(NODEPTR *plist, int x) {
        NODEPTR p;
        p = GetNode();
        p->info = x;
        p->next = *plist;
        *plist = p;
}
```

- **Thêm một nút có nội dung x ngay sau nút p**

```
void InsAfter(NODEPTR p, int x) {
        if(p == NULL)
                printf("Khong them nut duoc\n");
        else {
                NODEPTR q;
                q = GetNode();
                q->info = x;
                q->next = p->next;
                p->next = q;
        }
}
```

- **Xoá nút đầu trong danh sách liên kết**

```
int Pop(NODEPTR *plist) {
        if(Empty(plist))
                printf(“Danh sach bi rong\n”);
        else {
                NODEPTR p;
                int x;
                p = *plist;
                x = p->info;
                *plist = p ->next;
                p->next = NULL;
                FreeNode(p);
                return x;
}}
```

- **Xoá nút ngay sau nút p trong danh sách liên kết**

```
int DelAfter(NODEPTR p) {
        if(p == NULL || p->next == NULL)
                printf("Khong xoa nut duoc\n");
        else {
                NODEPTR q;
                int x;
                q = p->next;
                x = q->info;
                p->next = q->next;
                q->next = NULL;
                FreeNode(q);
                return x;
        }}
```

- **Duyệt danh sách liên kết**

```
void Traverse(NODEPTR *plist) {
if(Empty(plist))
        printf("Danh sách bị rỗng\n");
else {
        NODEPTR p;
        p = *plist;
        while(p != NULL) {
                printf("%d\t", p->info);
                p = p->next ;
        }}}
```

- **Tìm nút có nội dung x trong danh sách liên kết**

```
NODEPTR Search(NODEPTR *plist, int x) {
        NODEPTR p;
        p = *plist;
        while(p != NULL && p->info != x)
                p = p->next;
        return p;
}
```

- **Sắp xếp danh sách liên kết theo thứ tự tăng dần của trường info**

```
void Sort(NODEPTR *plist) {
        NODEPTR p, q;
        int temp;
        for(p = *plist; p != NULL; p = p->next) {
                for(q = p->next; q != NULL; q = q->next) {
                        if(p->info > q->info) {
                                temp = p->info;
                                p->info = q->info;
                                q->info = temp;
                        }}}}
```

- **Thêm nút có nội dung x trên danh sách liên kết đã có thứ tự**

```
void Place(NODEPTR *plist, int x) {
        NODEPTR p, q;
        q = NULL;
        for(p = *plist; p != NULL && x > p->info; p = p->next)
                q = p;
        if(q == NULL)
                Push(plist, x);
        else InsAfter(q, x);
}
```

- **Xoá danh sách liên kết**

```
void ClearList(NODEPTR *plist) {
NODEPTR p, q; p = *list;
q = NULL;
while(p != NULL) {
      q = p;
      p = p->next;
      FreeNode(q);
}
*plist = NULL;}
```

d. Ưu khuyết điểm của DSLK

- **Ưu điểm**

✓ Vì số nút cấp phát cho DSLK thay đổi khi chương trình đang chạy nên việc cấp phát nút cho danh sách liên kết rất linh hoạt.

✓ DSLK rất thích hợp khi thực hiện các thao tác thêm nút và xoá nút.

- **Khuyết điểm**

✓ Mật độ sử dụng bộ nhớ của DSLK thường nhỏ hơn 1 nghĩa là không tối ưu bộ nhớ

✓ Việc truy xuất nút thứ i trên DSLK chậm vì phải truy xuất tuần tự từ đầu danh sách

✓ Thao tác tìm kiếm trên DSLK cũng không tối ưu vì thường dùng phương pháp tìm liếm tuyến tính.

3. Danh sách liên kết kép

a. Giới thiệu DSLKK

DSLKK là một danh sách liên kết trong đó mỗi nút của danh sách có hai trường liên kết: một trường liên kết chỉ nút trước (trường left) và một trường liên kết chỉ nút sau (trường right).

Nút cuối DSLKK có trường right chỉ NULL, nút đầu danh sách liên kết kép có trường left chỉ NULL.

a. Khai báo cấu trúc một nút trong DSLKK

```
struct node {
    int info;
    node *left, *right;
};
//Khai báo kiểu con trỏ chỉ nút
typedef node * NODEPTR;
```

c. Các thao tác trên DSLKK

- **Cấp phát vùng nhớ động làm một nút**

```
NODEPTR GetNode() {
    NODEPTR p;
    p = (NODEPTR) malloc(sizeof(node));
    return p;
}
```

- **Giải phóng vùng nhớ động đã cấp phát trước đó**

```
void FreeNode(NODEPTR p) {
    free(p);
}
```

- **Khởi động DSLKK**

```
void Initialize(NODEPTR *plist) {
        *plist = NULL;
}
```

- **Kiểm tra DSLKK có rỗng hay không**

```
int Empty(NODEPTR *plist) {
        return (*plist == NULL ? 1 : 0);
}
```

- **Xác định con trỏ nút thứ i trong DSLKK**

```
NODEPTR NodePointer(NODEPTR *plist, int i) {
        NODEPTR p;
        p = *plist;
        int pos = 0;
        while(p != NULL && pos < i) {
                p = p->right;
                pos++;
        }
        return p;
}
```

- **Thêm nút có nội dung x vào đầu DSLKK**

```
void Push(NODEPTR *plist, int x) {
NODEPTR p = GetNode();
p->info = x;
if(Empty(plist)) {
        p->left = NULL;
        p->right = NULL;
        *plist = p;
}
else {
        p->right = *plist;
        (*plist)->left = p;
        p->left = NULL;
        *plist = p;
}}
```

- **Thêm nút có nội dung x vào sau nút p**

```
void InsertRight(NODEPTR p, int x) {
        NODEPTR q, r;
        if(p == NULL)
                printf("Nut p khong ton tai\n");
        else {
                q = GetNode();  q->info = x;
                r = p->right;
                //Tạo hai liên kết giữa r và q
                r->left = q;
                q->right = r;
                //Tạo hai liên kết giữa p và q
                q->left = p;
                p->right = q;
        }}
```

- **Thêm nút có nội dung là x vào trước nút p**

```
void InsertLeft(NODEPTR *plist, NODEPTR p, int x) {
        NODEPTR q, r;
        if(p == NULL)
                printf("Nut p khong ton tai\n");
        else if(p == *plist)
                Push(plist, x);
        else {
                q = getnode(); q->info = x; r = p->left;
                //Tạo hai liên kết giữa r và q
                r->right = q;
                q->left = r;
                //Tạo hai liên kết giữa p và q
                q->right = p;
                p->left = q; }}
```

- **Xoá nút ở đầu danh sách liên kết kép**

```
int Pop(NODEPTR *plist) {
        NODEPTR p; int x;
        if(Empty(plist))
                printf("Danh sach rong\n");
        else {
                p = *plist; x = p->info;
                if((*plist)->right == NULL)
                        *plist = NULL;
                else {
                        *plist = p->right;
                        p->right = NULL;
                        (*plist)->left = NULL;
                }
                FreeNode(p); return x; }}
```

- **Xoá nút p của danh sách liên kết kép**

```
int DelNode(NODEPTR *plist, NODEPTR p) {
        int x; NODEPTR q, r;
        if(p == NULL)  printf("Nut p khong ton tai\n");
        else if(Empty(plist))  printf("Danh sach rong\n");
        else if(p == *plist) {   //p là nút đầu
                x = Pop(plist); return x; }
        else {  //p không phải nút đầu
                x = p->info; q = p->left; r = p->right;
                 //Tạo hai liên kết giữa q va r
                if(r != NULL)  r->left = q;
                q->right = r;
                p->left = p->right = NULL;
                FreeNode(p);
                return x;}}
```

Duyệt xuôi danh sách liên kết kép

```
void RightTraverse(NODEPTR *plist) {
        NODEPTR p;
        if(Empty(plist))
                printf("Danh sach rong\n");
        else {
                p = *plist;
                while(p != NULL) {
                        printf("%d\t", p->info);
                        p = p->right ;
                }
        }}
```

- **Duyệt ngược danh sách liên kết kép**

```
void LeftTraverse(NODEPTR *plist) {
        NODEPTR p;
        if(Empty(plist))  printf("Danh sach rong\n");
        else {
                p = *plist;
/*Lần theo liên kết phải, di chuyển p từ nút đầu đến nút
   cuối*/
                while(p->right != NULL)
                        p = p->right;
/*Lần theo liên kết trái, di chuyển p từ nút cuối về nút đầu.*/
                while(p != NULL) {
                        printf("%d\t", p->info); p = p->left ;
        }}}
```

- **Xoá danh sách liên kết kép**

```
void ClearList(NODEPTR *plist) {
        while(*plist != NULL)
                Pop(plist);
}
```

Hết