Danh sách liên kết đơn – Single linked list

Danh sách liên kết đơn(Single linked list) là ví dụ tốt nhất và đơn giản nhất về cấu trúc dữ liệu động sử dụng con trỏ để cài đặt. Do đó, kiến thức con trỏ là rất quan trọng để hiểu cách danh sách liên kết hoạt động, vì vậy nếu bạn chưa có kiến thức về con trỏ thì bạn nên học về con trỏ trước. Bạn cũng cần hiểu một chút về cấp phát bộ nhớ động. Để đơn giản và dễ hiểu, phần nội dung cài đặt danh sách liên kết của bài viết này sẽ chỉ trình bày về danh sách liên kết đơn.

1. Lý thuyết về danh sách liên kết

Về bản chất, danh sách liên kết có chức năng như một mảng, có thể thêm và xóa các phần tử ở bất kỳ vị trí nào khi cần thiết. Một số sự khác nhau giữa danh sách liên kết và mảng:

Nội dung	Mảng	Danh sách liên kết
Kích thước	• Kích thước cố định • Cần chỉ rõ kích thước trong khi khai báo	• *Kích thước thay đổi trong quá trình thêm/ xóa phần tử* • *Kích thước tối đa phụ thuộc vào bộ nhớ*
Cấp phát bộ nhớ	• Tĩnh: Bộ nhớ được cấp phát trong quá trình biên dịch	• *Động: Bộ nhớ được cấp phát trong quá trình chạy*
Thứ tự & sắp xếp	• Được lưu trữ trên một dãy ô nhớ liên tục	• *Được lưu trữ trên các ô nhớ ngẫu nhiên*
Truy cập	• *Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1)*	• Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n)
Tìm kiếm	• *Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân*	• Chỉ có thể tìm kiếm tuyến tính

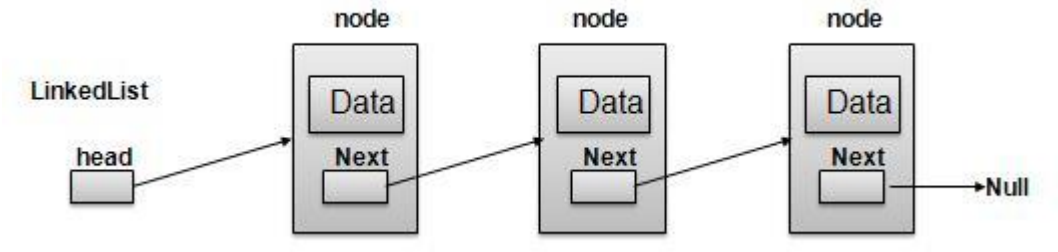
Lưu ý: Ở bảng phía trên, các phần in nghiêng thể hiện đó là ưu điểm so với đối thủ còn lại.

2. Danh sách liên kết là gì?

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa "*một giá trị"(Data)* và "*một con trỏ"(Next).* Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.

Hình ảnh mô tả cho một Node trong danh sách liên kết đơn:

Và đây là hình ảnh mô phỏng một danh sách liên đơn kết đầy đủ:



Mô phỏng của danh sách liên kết đơn

Danh sách các kiểu danh sách liên kết:

- Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau.
- Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía trước với phần tử phía sau
- Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.

3. Cài đặt danh sách liên kết đơn

3.1. Khai báo linked list

Để đơn giản hóa, data của chúng ta sẽ là số nguyên(int). Bạn cũng có thể sử dụng các kiểu nguyên thủy khác(float, char,…) hay kiểu dữ liệu struct(SinhVien, CanBo,…) tự tạo.

```c
struct LinkedList{
    int data;
    struct LinkedList *next;
};
```

Khai báo trên sẽ được sử dụng cho mọi Node trong linked list. Trường `data` sẽ lưu giữa giá trị và `next` sẽ là con trỏ để trỏ đến thằng kế tiếp của nó.

Tại sao next lại là kiểu LinkedList của chính nó? Bởi vì nó là con trỏ trỏ của chính bản thân nó, và nó trỏ tới một thằng Node kế tiếp cũng có kiểu LinkedList.

3.2. Tạo mới 1 Node

Hãy tạo một kiểu dữ liệu của struct LinkedList để code clear hơn:

```c
typedef struct LinkedList *node; //Từ giờ dùng kiểu dữ liệu LinkedList có thể thay bằng node

node CreateNode(int value){
    node temp; // declare a node
    temp = (node)malloc(sizeof(struct LinkedList)); // Cấp phát vùng nhớ dùng malloc()
    temp->next = NULL;// Cho next trỏ tới NULL
    temp->data = value; // Gán giá trị cho Node
    return temp;//Trả về node mới đã có giá trị
}
```

Mỗi một Node khi được khởi tạo, chúng ta cần cấp phát bộ nhớ cho nó, và mặc định cho con trỏ `next` trỏ tới NULL. Giá trị của Node sẽ được cung cấp khi thêm Node vào linked list.

- **typedef** được dùng để định nghĩa một kiểu dữ liệu trong C.
  VD: `typeder long long LL;`
- **malloc** là hàm cấp phát bộ nhớ của C. Với C++ chúng ta dùng `new`
- **sizeof** là hàm trả về kích thước của kiểu dữ liệu, dùng làm tham số cho hàm *malloc*

Lưu ý: Không giống với mảng, cần khai báo *arr[size]*. Trong linked list, vì mỗi Node sẽ có con trỏ liên kết đến Node tiếp theo. Do đó, với danh sách liên kết đơn, bạn chỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi bạn có thể đi tới bất cứ Node nào.

3.3. Thêm Node vào danh sách liên kết

Thêm vào đầu

Việc thêm vào đầu chính là việc cập nhật lại thằng head. Ta gọi Node mới(temp), ta có:

- Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm head luôn
- Ngược lại, ta phải thay thế thằng head cũ bằng head mới. Việc này phải làm theo thứ tự như sau:
    - Cho next của temp trỏ tới head hiện hành
    - Đặt temp làm head mới

```
node AddHead(node head, int value){
    node temp = CreateNode(value); // Khởi tạo node temp với data = value
    if(head == NULL){
        head = temp; // //Nếu linked list đang trống thì Node temp là head luôn
    }else{
        temp->next = head; // Trỏ next của temp = head hiện tại
        head = temp; // Đổi head hiện tại = temp(Vì temp bây giờ là head mới mà)
    }
    return head;
}
```

Thêm vào cuối

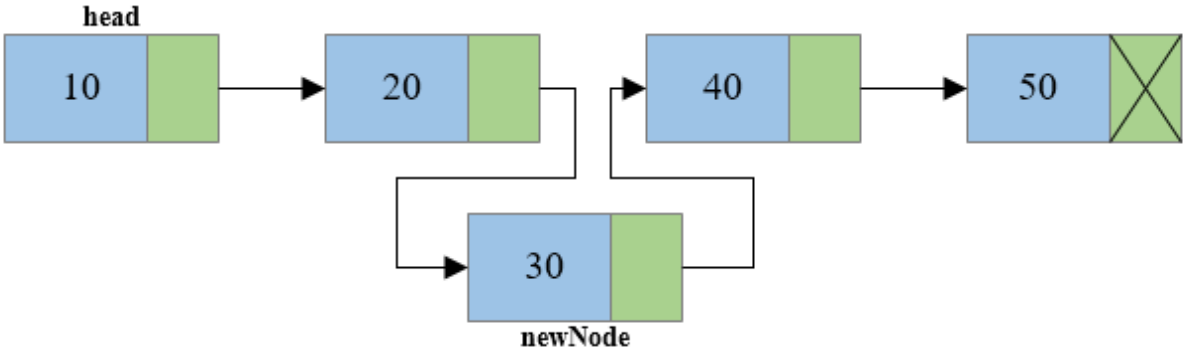
Chúng ta sẽ cần Node đầu tiên, và giá trị muốn thêm. Khi đó, ta sẽ:

1. Tạo một Node mới với giá trị value
2. Nếu head = NULL, tức là danh sách liên kết đang trống. Khi đó Node mới(temp) sẽ là head luôn.
3. Ngược lại, ta sẽ duyệt tới Node cuối cùng(Node có next = NULL), và trỏ next của thằng cuối tới Node mới(temp).

```
node AddTail(node head, int value){
    node temp,p;// Khai báo 2 node tạm temp và p
    temp = CreateNode(value);//Gọi hàm createNode để khởi tạo node temp có next trỏ tới NULL
    if(head == NULL){
        head = temp;     //Nếu linked list đang trống thì Node temp là head luôn
    }
    else{
        p  = head;// Khởi tạo p trỏ tới head
        while(p->next != NULL){
            p = p->next;//Duyệt danh sách liên kết đến cuối. Node cuối là node có next = NULL
        }
        p->next = temp;//Gán next của thằng cuối = temp. Khi đó temp sẽ là thằng cuối(temp->
    }
    return head;
}
```

Tổng quan hơn, chúng ta sẽ sẽ viết hàm thêm một Node vào vị trí bất kỳ nhé.

Thêm vào vị trí bất kỳ



Thêm Node vào giữa danh sách liên kết

Để làm được việc này, ta phải duyệt từ đầu để tìm tới vị trí của Node cần chèn, giả sử là Node Q, khi đó ta cần làm theo thứ tự sau:

- Cho next của Node mới trỏ tới Node mà Q đang trỏ tới
- Cho Node Q trỏ tới Node mới

Lưu ý: Chỉ số chèn bắt đầu từ chỉ số 0 nhé các bạn

```
node AddAt(node head, int value, int position){
    if(position == 0 || head == NULL){
        head = AddHead(head, value); // Nếu vị trí chèn là 0, tức là thêm vào đầu
    }else{
        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí
        int k = 1;
        node p = head;
        while(p != NULL && k != position){
            p = p->next;
            ++k;
        }

        if(k != position){
            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định
            // Nếu bạn không muốn chèn, hãy thông báo vị trí chèn không hợp lệ
            head = AddTail(head, value);
            // printf("Vi tri chen vuot qua vi tri cuoi cung!\n");
        }else{
            node temp = CreateNode(value);
            temp->next = p->next;
            p->next = temp;
        }
    }
    return head;
}
```

Lưu ý: Bạn phải làm theo thứ tự trên, nếu bạn cho p->next = temp trước. Khi đó, bạn sẽ không thể lấy lại phần sau của danh sách liên kết nữa(Vì next chỉ được được lưu trong p->next mà thay đổi p->next rồi thì còn đâu giá trị cũ).

3.4. Xóa Node khỏi danh sách liên kết

Xóa đầu

Xóa đầu đơn giản lắm, bây giờ chỉ cần cho thằng kế tiếp của head làm head là được thôi. Mà thằng kế tiếp của head chính là *head->next.*

```
node DelHead(node head){
    if(head == NULL){
        printf("\nCha co gi de xoa het!");
    }else{
        head = head->next;
    }
    return head;
}
```

Xóa cuối

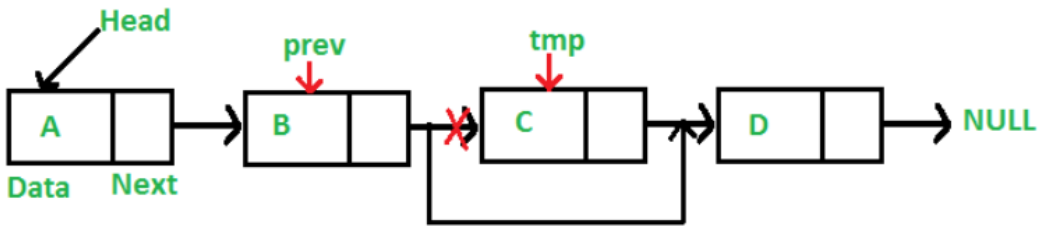
Xóa cuối mới nhọc nè, nhọc ở chỗ phải duyệt đến thằng cuối – 1, cho next của cuối – 1 đó bằng NULL.

```
node DelTail(node head){
    if (head == NULL || head->next == NULL){
        return DelHead(head);
    }
    node p = head;
    while(p->next->next != NULL){
        p = p->next;
    }
    p->next = p->next->next; // Cho next bằng NULL
    // Hoặc viết p->next = NULL cũng được
    return head;
}
```

Thằng Node cuối – 1 là thằng có *p->next->next = NULL*. Bạn cho next của nó bằng NULL là xong.

Xóa ở vị trí bất kỳ

Việc xóa ở vị trí bất kỳ cũng khá giống xóa ở cuối kia. Đơn giản là chúng ta bỏ qua một phần tử, như ảnh sau:



Xóa node trong danh sách liên kết

Lưu ý: Chỉ số xóa bắt đầu từ 0 nhé các bạn. Việc tìm vị trí càn xóa chỉ duyệt tới Node gần cuối thôi(cuối – 1). Sau đây là code xóa Node ở vị trí bất kỳ

```
node DelAt(node head, int position){
    if(position == 0 || head == NULL || head->next == NULL){
        head = DelHead(head); // Nếu vị trí chèn là 0, tức là thêm vào đầu
    }else{
        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí
        int k = 1;
        node p = head;
        while(p->next->next != NULL && k != position){
            p = p->next;
            ++k;
        }

        if(k != position){
            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định
            // Nếu bạn không muốn xóa, hãy thông báo vị trí xóa không hợp lệ
            head = DelTail(head);
            // printf("Vi tri xoa vuot qua vi tri cuoi cung!\n");
        }else{
            p->next = p->next->next;
        }
    }
    return head;
}
```

3.5. Lấy giá trị ở vị trí bất kỳ

Chúng ta sẽ viết một hàm để truy xuất giá trị ở chỉ số bất kỳ nhé. Trong trường hợp chỉ số vượt quá chiều dài của linked list – 1, hàm này trả về vị trí cuối cùng. Do hạn chế là chúng ta không thể raise error khi chỉ số không hợp lệ. Tôi mặc định chỉ số bạn truyền vào phải là số nguyên không âm. Nếu bạn muốn kiểm tra chỉ số hợp lệ thì nên kiểm tra trước khi gọi hàm này.

```
int Get(node head, int index){
    int k = 0;
    node p = head;
    while(p->next != NULL && k != index){
        ++k;
        p = p->next;
    }
    return p->data;
}
```

Lý do dùng `p->next != NULL` là vì chúng ta chỉ muốn đi qua các phần tử có value.

3.6. Tìm kiếm trong danh sách liên kết

Hàm tìm kiếm này sẽ trả về chỉ số của Node đầu tiên có giá trị bằng với giá trị cần tìm. Nếu không tìm thấy, chúng ta trả về -1.

```
int Search(node head, int value){
    int position = 0;
    for(node p = head; p != NULL; p = p->next){
        if(p->data == value){
            return position;
        }
        ++position;
    }
    return -1;
}
```

Chúng ta có thể sử dụng hàm này để xóa tất cả các Node trong danh sách liên kết có giá trị chỉ định như sau:

```
node DelByVal(node head, int value){
    int position = Search(head, value);
    while(position != -1){
        DelAt(head, position);
        position = Search(head, value);
    }
    return head;
}
```

3.7. Duyệt danh sách liên kết

Việc duyệt danh sách liên kết cực đơn giản. Khởi tạo từ Node head, bạn cứ thế đi theo con trỏ next cho tới trước khi Node đó NULL.

```
void Traverser(node head){
    printf("\n");
    for(node p = head; p != NULL; p = p->next){
        printf("%5d", p->data);
    }
}
```

3.8. Một số hàm bổ trợ khác

Hàm khởi tạo Node head

Đơn giản là cho con trỏ head = NULL thôi. Nếu bạn để ý, chúng ta vẫn check head = NULL để biết rằng danh sách liên kết chưa có phần tử nào ở các hàm phía trên.

```
node InitHead(){
    node head;
    head = NULL;
    return head;
}
```

Hàm lấy số phần tử của DSLK

Duyệt và đếm chừng nào các Node chưa NULL. Sau cùng, trả về giá trị đếm được.

```
int Length(node head){
    int length = 0;
    for(node p = head; p != NULL; p = p->next){
        ++length;
    }
    return length;
}
```

Hàm nhập danh sách liên kết

```
node Input(){
    node head = InitHead();
    int n, value;
    do{
        printf("\nNhap so luong phan tu n = ");
        scanf("%d", &n);
    }while(n <= 0);

    for(int i = 0; i < n; ++i){
        printf("\nNhap gia tri can them: ");
        scanf("%d", &value);
        head = AddTail(head, value);
    }
    return head;
}
```

4. Code hoàn chỉnh của danh sách liên kết đơn

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

struct LinkedList{
```

```
    int data;
    struct LinkedList *next;
 };

typedef struct LinkedList *node; //Từ giờ dùng kiểu dữ liệu LinkedList có thể
thay bằng node cho ngắn gọn

node CreateNode(int value){
    node temp; // declare a node
    temp = (node)malloc(sizeof(struct LinkedList)); // Cấp phát vùng nhớ dùng
malloc()
    temp->next = NULL;// Cho next trỏ tới NULL
    temp->data = value; // Gán giá trị cho Node
    return temp;//Trả về node mới đã có giá trị
}

node AddTail(node head, int value){
    node temp,p;// Khai báo 2 node tạm temp và p
    temp = CreateNode(value);//Gọi hàm createNode để khởi tạo node temp có next
trỏ tới NULL và giá trị là value
    if(head == NULL){
        head = temp;     //Nếu linked list đang trống thì Node temp là head luôn
    }
    else{
        p  = head;// Khởi tạo p trỏ tới head
        while(p->next != NULL){
            p = p->next;//Duyệt danh sách liên kết đến cuối. Node cuối là node có
next = NULL
        }
        p->next = temp;//Gán next của thằng cuối = temp. Khi đó temp sẽ là thằng
cuối(temp->next = NULL mà)
    }
    return head;
}


node AddHead(node head, int value){
    node temp = CreateNode(value); // Khởi tạo node temp với data = value
    if(head == NULL){
        head = temp; // //Nếu linked list đang trống thì Node temp là head luôn
    }else{
        temp->next = head; // Trỏ next của temp = head hiện tại
        head = temp; // Đổi head hiện tại = temp(Vì temp bây giờ là head mới mà)
    }
    return head;
```

```
}

node AddAt(node head, int value, int position){
    if(position == 0 || head == NULL){
        head = AddHead(head, value); // Nếu vị trí chèn là 0, tức là thêm vào đầu
    }else{
        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí
        int k = 1;
        node p = head;
        while(p != NULL && k != position){
            p = p->next;
            ++k;
        }

        if(k != position){
            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta
sẽ mặc định chèn cuối
            // Nếu bạn không muốn chèn, hãy thông báo vị trí chèn không hợp lệ
            head = AddTail(head, value);
            // printf("Vi tri chen vuot qua vi tri cuoi cung!\n");
        }else{
            node temp = CreateNode(value);
            temp->next = p->next;
            p->next = temp;
        }
    }
    return head;
}

node DelHead(node head){
    if(head == NULL){
        printf("\nCha co gi de xoa het!");
    }else{
        head = head->next;
    }
    return head;
}

node DelTail(node head){
    if (head == NULL || head->next == NULL){
         return DelHead(head);
    }
    node p = head;
    while(p->next->next != NULL){
        p = p->next;
```

```
    }
    p->next = p->next->next; // Cho next bằng NULL
    // Hoặc viết p->next = NULL cũng được
    return head;
}

node DelAt(node head, int position){
    if(position == 0 || head == NULL || head->next == NULL){
        head = DelHead(head); // Nếu vị trí chèn là 0, tức là thêm vào đầu
    }else{
        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí
        int k = 1;
        node p = head;
        while(p->next->next != NULL && k != position){
            p = p->next;
            ++k;
        }

        if(k != position){
            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta
sẽ mặc định xóa cuối
            // Nếu bạn không muốn xóa, hãy thông báo vị trí xóa không hợp lệ
            head = DelTail(head);
            // printf("Vi tri xoa vuot qua vi tri cuoi cung!\n");
        }else{
            p->next = p->next->next;
        }
    }
    return head;
}

int Get(node head, int index){
    int k = 0;
    node p = head;
    while(p != NULL && k != index){
        p = p->next;
    }
    return p->data;
}

int Search(node head, int value){
    int position = 0;
    for(node p = head; p != NULL; p = p->next){
        if(p->data == value){
            return position;
```

```
        }
        ++position;
    }
    return -1;
}

node DelByVal(node head, int value){
    int position = Search(head, value);
    while(position != -1){
        DelAt(head, position);
        position = Search(head, value);
    }
    return head;
}

void Traverser(node head){
    printf("\n");
    for(node p = head; p != NULL; p = p->next){
        printf("%5d", p->data);
    }
}

node InitHead(){
    node head;
    head = NULL;
    return head;
}

int Length(node head){
    int length = 0;
    for(node p = head; p != NULL; p = p->next){
        ++length;
    }
    return length;
}

node Input(){
    node head = InitHead();
    int n, value;
    do{
        printf("\nNhap so luong phan tu n = ");
        scanf("%d", &n);
    }while(n <= 0);

    for(int i = 0; i < n; ++i){
```

```
        printf("\nNhap gia tri can them: ");
        scanf("%d", &value);
        head = AddTail(head, value);
    }
    return head;
}



int main(){
    printf("\n==Tao 1 danh sach lien ket==");
    node head = Input();
    Traverser(head);

    printf("\n==Thu them 1 phan tu vao linked list==");
    head = AddAt(head, 100, 2);
    Traverser(head);

    printf("\n==Thu xoa 1 phan tu khoi linked list==");
    head = DelAt(head, 1);
    Traverser(head);

    printf("\n==Thu tim kiem 1 phan tu trong linked list==");
    int index = Search(head, 5);
    printf("\nTim thay tai chi so %d", index);

    printf("\nBan co the thu them nhe!");

}
```

Kết quả chạy thử:

```
==Tao 1 danh sach lien ket==
Nhap so luong phan tu n = 5

Nhap gia tri can them: 1

Nhap gia tri can them: 2

Nhap gia tri can them: 3

Nhap gia tri can them: 4

Nhap gia tri can them: 5

    1    2    3    4    5
==Thu them 1 phan tu vao linked list==
    1    2  100    3    4    5
==Thu xoa 1 phan tu khoi linked list==
    1  100    3    4    5
==Thu tim kiem 1 phan tu trong linked list==
Tim thay tai chi so 4
Ban co the thu them nhe!
Process returned 0 (0x0)   execution time : 4.166 s
Press any key to continue.
```