Bài 10_ Linked List

Đối với 1 mảng array thông thường:

```
int arr[10000] = {2,7,4,5,3,...,1,2};
=> arr[10001] = {2,1,7,4,5,3,...,1,2};
```

Address	0x01	0x05	0x09	…	x	x+4
Value	2	1	7	…	1	2

- Khi muốn thêm hoặc xóa một phần tử đầu hoặc giữa mảng phải dịch các phần tử phía sau, khi mang có số lượng phần tử lớn thì sẽ làm việc này trở nên kém hiệu quả.
- Mảng thường có kích thước cố định, muốn thu hẹp hay mở rộng phải tạo mảng mới, và gán phần tử mảng cũ sang.
- Linked list: Có thể dễ dàng mở rộng hoặc thu hẹp kích thước mà không cần cấp phát lại bộ nhớ. Mỗi node được cấp phát động khi cần, giúp linked list linh hoạt trong việc quản lý số lượng phần tử.
- Mảng: Thường có kích thước cố định khi được tạo. Nếu cần thay đổi kích thước, bạn phải tạo một mảng mới và sao chép toàn bộ dữ liệu sang, gây tốn thời gian và bộ nhớ.
-

Để giải quyết nhược điểm này của mảng, người ta đã nghiên cứu ra cấu trúc dữ liệu linked list:

Linked list là một cấu trúc dữ liệu trong lập trình máy tính, được sử dụng để tổ chức và lưu trữ dữ liệu. Một linked list bao gồm một chuỗi các "nút" (nodes), mỗi nút chứa một giá trị dữ liệu và một con trỏ (pointer) đến nút tiếp theo trong chuỗi.

```
int arr[10000] = {2,7,4,5,3,...};
```

Address	0xa1	0xb3	0xef
Value	2	7	4
Next	0xb3	0xef	NULL

10.2. Đặc điểm của 1 danh sách liên kết.

Danh sách liên kết được tạo thành từ nhiều node. Mỗi node nó là 1 struct gồm 2 thành viên.

- Thành viên 1 (value) : chính là data

- Thành viên 2(next): con trỏ lưu địa chỉ của node kế tiếp, do node kế tiếp cũng gồm 2 phần tử trên. => Nên dùng 1 con trỏ kiểu node lưu thông tin node kế tiếp tạo liên kết.
- Tương tự mảng array, danh sách liên kết chỉ cần có địa chỉ của node đầu tiên (node head) là có thể duyệt truy xuất tất cả các node. => node head là tham số đại diện của linklist để truyền vào các hàm xử lý.
- Nếu danh sách rỗng *head = Null
- Node cuối cùng luôn có Next = Null. Node->Next = NULL
- Chèn một node mới vào danh sách, nếu node đó rơi vào node head thì phải đổi lại địa chỉ đứng đầu sang địa chỉ node mới.
- Giả sử muốn NewNode vào giữa Node 2, Node 3 đã có liên kết . thì Node2.Next =Null, Node2.Next = NewNode.address, NewNode.Next = Node3.address.

```
typedef struct Node {
    int data;           // Dữ liệu của nút
    struct Node* next;  // Con trỏ tới nút tiếp theo
} Node;
```

```
node *createNode(int value);
void push_back(node** array, int value); // them 1 node vao phia sau
void push_front(node **array, int value); // them 1 node vao phia truoc
void pop_back(node **array); // xóa phần tử sau cung
void pop_front(node **array); // xoa node dau tien
int front(node *array); // lay gia tri cua node dau tien
int back(node *array); // lay gia tri cua node cuoi cung
void insert(node **array, int value, int pos); // them 1 node vao mot vi
tri bat ky
void erase(node **array, int pos); // xoa 1 node tai mot vi tri bat ky
int size(node *array); // lay kich thuoc cua list
int get(node *array, int pos);
bool empty(node *array); // kiem tra list co rong hay khong
```

Dùng con tro cấp 2 để có thể thay đổi địa chỉ node đầu, khi node đầu bị xóa, hoặc node đầu là null