Single Linked List

Linked List

Sinh viên: Nguyễn Văn Tín

Email: tinnguyensp.ict@gmail.com

Github: https://github.com/tinict

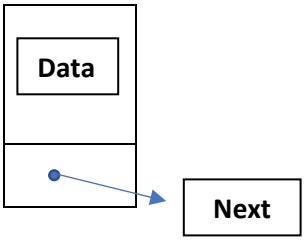
MỤC LỤC

Single Linked List

I Giới thiệu danh sách liên kết đơn

Danh sách liên kết đơn (Singhle Linked List) 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 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 đó là phần tử cuối cùng của danh sách liên kết (**Linked List**).

Danh sách liên kết đơn được cấu tạo gồm hai thành phần chính **data** và một con trỏ **next**.



Ví dụ:

Input:

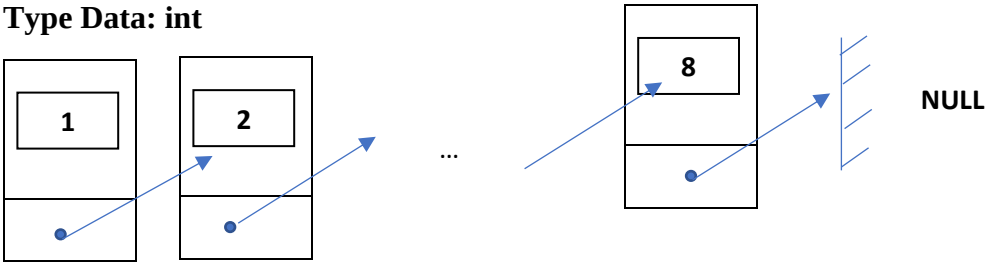
8

1 2 3 4 5 6 7 8

Output:

1 2 3 4 5 6 7 8

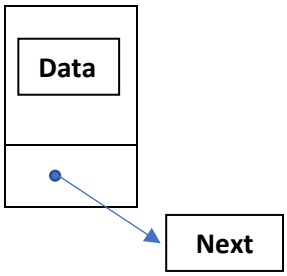
Type Data: int



II. Cài đặt danh sách liên kết – Single Linked List

1. **Khai báo Node**

$$\textbf{Node} = \begin{cases} \boldsymbol{Data}\ (Lưu\ trữ\ dữ\ liệu) \\ \boldsymbol{Next}\ (Con\ trỏ\ next) \end{cases}$$



Xây dựng lớp Node:

Lớp **Node** có nhiệm vụ khai báo cấu trúc (hình hài) của Node

```
class Node
{
	public:
		int data;
		Node *next;
};
```

Xây dựng lớp danh sách

Khởi tạo node

Một **Node** được xây dựng khởi tạo 1 không gian nhớ đặt tên là biến **temp** có nhiệm vụ cấp không gian cho node lưu trữ nhận dữ liệu (**data**). Sau khi cấp không gian nhớ cho **temp** xong **temp** sẽ trỏ đến **data** nhận dữ liệu lưu trữ vào không gian vừa mới khởi tạo. Vào sau đó **temp** sẽ được trỏ tới **next** vào **next = NULL** (kết thúc 1 Node).

```
class List
{
	public:
		Node *first;
	public:
		Node *CreateNode(int); // Khởi tạo Node
};

Node *List::CreateNode(int x)
{
	Node *temp = new Node(); // Khởi tạo không gian Node
	temp -> data = x; // temp trỏ tới data nhận giá trị x
	temp -> next = NULL; // temp trỏ tới next = NULL
	return temp;
}
```

Hàm tạo

Khởi tạo một con trỏ **first** ở đầu danh sách và con trỏ **first**. Con trỏ **first** có nhiệm vụ duyệt danh sách từ đầu danh sách tới cuối danh sách, khi mình muốn truy xuất đến đâu thì con trỏ **first** theo đến đó. Khi khởi tạo danh sách thì con trỏ **first** này chưa chứa dữ liệu nên **first** được gán = **NULL**.

Code:

```
Class List
{
	private:
		node *first;
	public:
		node *CreateNode(int); // Khởi tạo Node
		List()
		{
			first = NULL;
		}
};
```

2. Chèn đầu và chèn sau vào một danh sách

a. Chèn đầu danh sách (addHead)

Xây dựng hàm chèn đầu danh sách

Bước 1: Kiểm tra danh sách có phải rỗng hay không **(== NULL**)

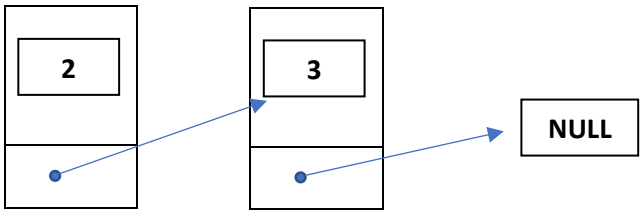
- Nếu **NULL** thì khởi tạo **Node** thông qua hàm **node *CreateNode(int x);**
- Nếu khác **NULL** thì qua **bước 2**

Bước 2:

- Khởi tạo một **node** tạm để lưu dữ liệu mới khởi tạo (nhiệm vụ tránh làm mất dữ liệu có sẵn trong **data**).
- Sau đó **node** tạm vừa được khởi tạo trỏ tới **next =** danh sách cũ đều này giúp cho 2 danh sách mới vào cũ nối lại với nhau.
- **Bước cuối** cùng đưa con trỏ **first** quay lại vị trí đầu danh sách.

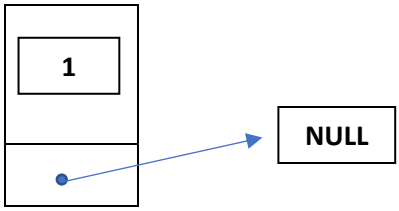
Minh họa hình vẽ:

Danh sách cũ:

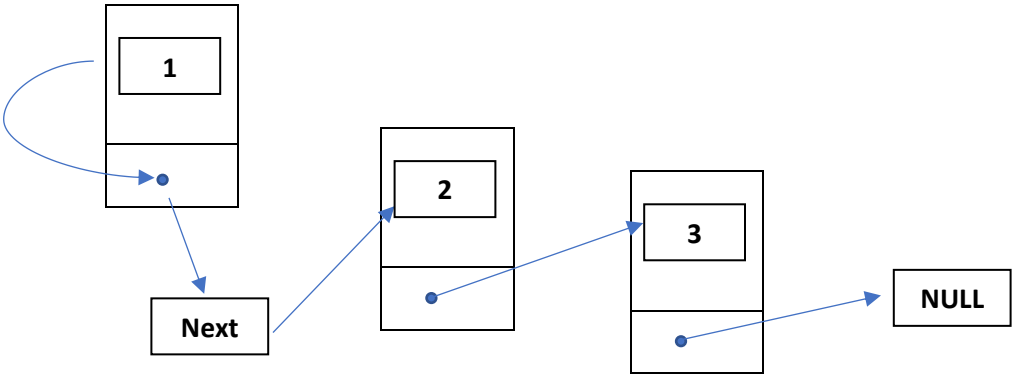


Chèn số 1 vào đầu danh sách:

Khởi tạo node ***temp = new node(int x);** khởi tạo node tạm để lưu trữ dữ liệu mới.



Tiếp theo ta thực hiện việc nối 2 danh sách lại với nhau



Code:

```
void List::addHead(int x)
{
	if(first == NULL) // Trường hợp danh rỗng
		first = CreateNode(x); // khởi tạo node đầu tiên
	else if(first != NULL) // Trường hợp danh sách không rỗng
	{
		Node *temp = CreateNode(x); // Khởi tạo node tạm thời
		temp->next = first; // Nối 2 danh sách cũ và mới lại với nhau
		first = temp; // Đưa con trỏ first quay lại vị trí đầu danh sách
	}
}
```

b. Chèn vào cuối danh sách (addTail)

Xây dựng hàm chèn sau danh sách

Bước 1: Kiểm tra danh sách có phải rỗng hay không **(== NULL**)

- Nếu **NULL** thì khởi tạo **Node** thông qua hàm **node *CreateNode(int x);**
- Nếu khác **NULL** thì qua **bước 2**

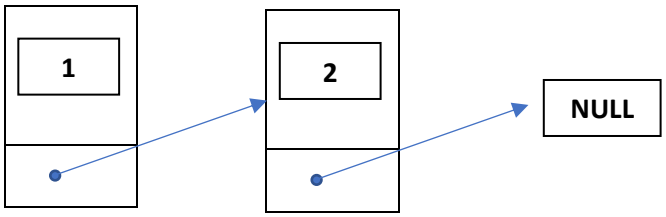
Bước 2:

- Khởi tạo một **node** tạm để lưu dữ liệu mới khởi tạo (nhiệm vụ tránh làm mất dữ liệu có sẵn trong **data**).

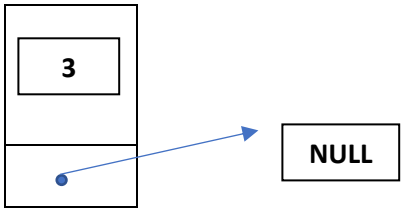
- Tiếp theo đưa con trỏ **first** của danh sách cũ về cuối danh sách bằng cách gán **node *head = first** để tránh làm mất dữ liệu cũ.
- Sau đó **node** tạm vừa được khởi tạo trỏ tới **next =** danh sách cũ đều này giúp cho 2 danh sách mới vào cũ nối lại với nhau.
- Sau đó head trỏ đến next gán them dữ liệu mới vào.

Minh họa hình vẽ:

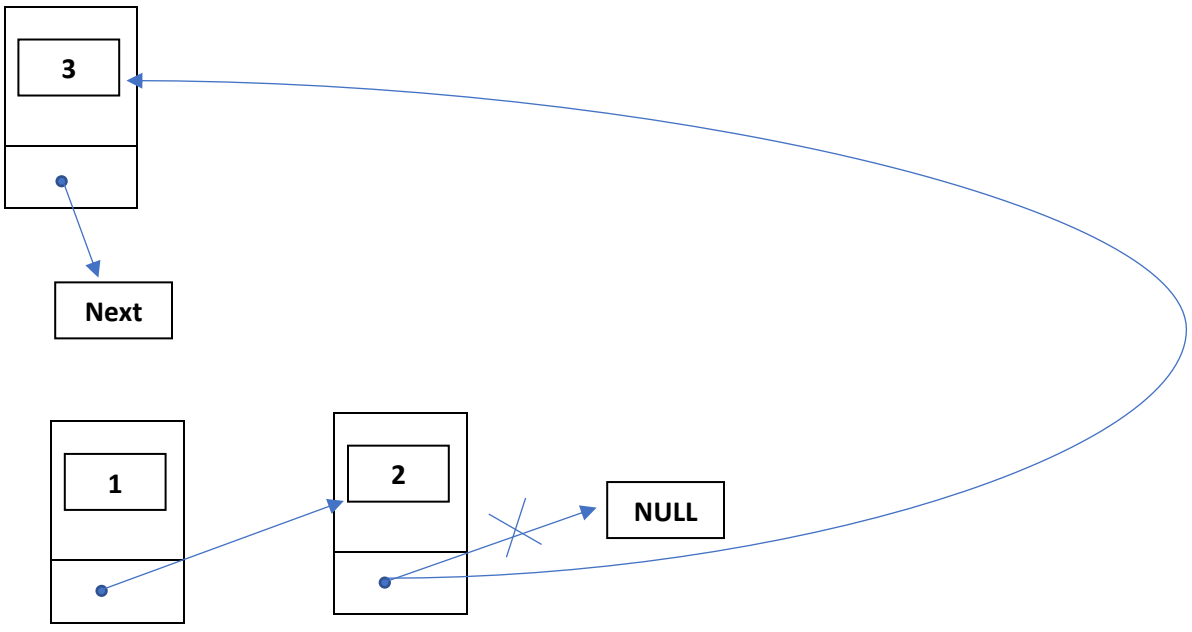
Danh sách cũ:



Danh sách mới:



Chèn số 3 vào cuối danh sách



Code:

```
void List::addTail(int x)
{
	if(first == NULL)
		first = CreateNode(x);
	else if(first != NULL)
	{
		Node *temp = CreateNode(x);
		Node *head = first;
		while(head->next != NULL)
			head = head->next;
		head->next = temp;
	}
}
```

3. **Hàm nhập danh sách và in ra danh sách**

   Hàm nhập được khởi tạo bằng tạo bằng cách tạo 1 vòng lặp liên tục bằng cách them chèn đầu hoặc chèn sau vào.

   Code:

```
void List::Ip_List()
{
	int n,x;
	cin>>n;
	for(int i = 0 ; i < n; i++)
	{
		cin >> x;
		addTail(x);
	}
}
```

Hàm in ra cũng tạo 1 con trỏ duyệt mảng khi gặp NULL thì dừng lại

Code:

```
void List::print_List()
{
	Node *head = first;
	while(head != NULL)
	{
		cout << head->data << " ";
		head = head->next;
	}
}
```

III. Biểu diễn Full danh sách liên kết (Single Linked List)

Ex:

Type Data: Kiểu số nguyên **(int)**

Input:

8

1 2 3 4 5 6 7 8

Output:

1 2 3 4 5 6 7 8

Code:

```
#include <bits/stdc++.h>
#include <iostream>
#include <conio.h>

using namespace std;

class Node
{
	public:
```

```
            int data;
            Node *next;
};

class List
{
      private:
            Node *first;
      public:
            Node *CreateNode(int);
            List()
            {
                  first = NULL;
            }
            void addHead(int);
            void addTail(int);
            void Ip_List();
            void print_List();
};

Node *List::CreateNode(int x)
{
      Node *temp = new Node();
      temp -> data = x;
      temp -> next = NULL;
      return temp;
}


void List::addHead(int x)
{
      if(first == NULL)
            first = CreateNode(x);
      else if(first != NULL)
      {
            Node *temp = CreateNode(x);
            temp->next = first;
            first = temp;
      }
}
```

```
void List::addTail(int x)
{
	if(first == NULL)
		first = CreateNode(x);
	else if(first != NULL)
	{
		Node *temp = CreateNode(x);
		Node *head = first;
		while(head->next != NULL)
			head = head->next;
		head->next = temp;
	}
}

void List::Ip_List()
{
	int n,x;
	cin>>n; // so node khoi tao
	for(int i = 0 ; i < n; i++)
	{
		cin >> x;
		addTail(x);
	}
}

void List::print_List()
{
	Node *head = first;
	while(head != NULL)
	{
		cout << head->data << " ";
		head = head->next;
	}
}

int main()
{
	List f;
	f.Ip_List();
	f.print_List();
	return 0;}
```

IV. Bài tập minh họa thực tế

Bài 1: Để quản lý **nhân viên** trong một cơ quan, ta dùng một danh sách liên kết đơn, mỗi nút của danh sách biểu diễn một đối tượng nhân viên, và có các thông tin sau đây: Mã nhân viên, họ tên, số điện thoại. Kiểu lớp **Nhanvien** để mô tả đối tượng nhân viên được cho như sau:

1. Xây dựng các phương thức của lớp **Nhanvien**: Hàm tạo để khởi tạo giá trị cho các
thuộc tính; Các hàm lấy giá trị của Manv, Hoten, Sdt của lớp **Nhanvien**.
2. Xây dựng lớp **Quanly**, trong đó mỗi đối tượng là 1 danh sách liên kết chứa các phần tử
kiểu Nhanvien, được định nghĩa như sau:

Các phương thức của lớp **Quanly** bao gồm:

a. Nhập dữ liệu cho đối tượng (danh sách liên kết các nhân viên) của lớp Quanly, quá
trình nhập dữ liệu kết thúc khi mã nhân viên nhập vào là rỗng.

b. Sắp xếp các nhân viên theo thứ tự tăng của họ tên nhân viên.

c. Tìm kiếm theo mã nhân viên

d. Hiển thị thông tin về các nhân viên.

Code:

```
#include <bits/stdc++.h>
#include <iostream>
#include <conio.h>

using namespace std;

class NhanVien
{
	private:
		string Manv;
		string Hoten;
		int Sdt;
	public:
		NhanVien()
		{
			Manv = "NULL";
			Hoten = "NULL";
			Sdt = 0;
```

```
            }
            void Ip_Info();
            void Op_Info();
            string get_MNV();
            int len();
            string get_Hoten();
            NhanVien get_NV();
};

int NhanVien::len()
{
      return Manv.length();
}

string NhanVien::get_Hoten()
{
      return Hoten;
}

string NhanVien::get_MNV()
{
      return Manv;
}

void NhanVien::Ip_Info()
{
      cout << "Manv: ";
      getline(cin,Manv);
      if(Manv.length() != 0)
      {
            cout << "Hoten: ";
            cin >> Hoten;
            cout << "Sdt: ";
            cin >> Sdt;
            cin.ignore();
      }
}

void NhanVien::Op_Info()
{
      cout << Manv << "\t\t" << Hoten << "\t\t" << Sdt << endl;
}
```

```
class node
{
	public:
		NhanVien data;
		node *next;
};

class List
{
	private:
		node *f; // first
	public:
		node *createNode(NhanVien x);
		void createList();
		void addHead(NhanVien x);
		void addTail(NhanVien x);
		void Ip_List();
		void Op_List();
		void sort();
		void Search_MSV();
};

node *List::createNode(NhanVien x)
{
	node *temp = new node();
	temp->data = x;
	temp->next = NULL;
	return temp;
}

void List::createList()
{
	node *f = NULL;
}

void List::addHead(NhanVien x)
{
	if(f == NULL)
		f = createNode(x);
	else if(f != NULL)
	{
```

```
            node *temp = createNode(x);
            temp->next = f;
            f = temp;
      }
}

void List::addTail(NhanVien x)
{
      if(f == NULL)
            f = createNode(x);
      else if(f != NULL)
      {
            node *temp = createNode(x);
            node *p = f;
            while(p->next != NULL)
                  p = p->next;
            p->next = temp;
      }
}

void List::Ip_List()
{
      NhanVien x;
      while(1)
      {
            x.Ip_Info();
            if(x.len() == 0)
                  break;
            addTail(x);
      }
}

void List::Op_List()
{
      node *p = f;
      while(p != NULL)
      {
            p->data.Op_Info();
            p = p->next;
      }
}
```

```
void List::sort()
{
	for(node *i = f; i->next != NULL; i = i->next)
	{
		for(node *j = i->next; j != NULL; j = j->next)
		{
			if(i->data.get_Hoten() > j->data.get_Hoten())
			{
				NhanVien t = i->data;
				i->data = j->data;
				j->data = t;
			}
		}
	}
}

void List::Search_MSV()
{
	string m;
	cin >> m;
	node *p = f;
	while(p->data.get_MNV() != m)
		p = p->next;
	p->data.Op_Info();
}

int main()
{
	List l;
	l.Ip_List();
	l.sort();
	l.Op_List();
	l.Search_MSV();
	return 0;
}
```