Mục tiêu

Nắm được các kiến thức cơ bản về danh sách liên kết (Linked list).

Nội dung

Danh sách liên kết

Cần hiểu rõ: mảng có được ngay khi khai báo (mảng tĩnh hay cấp phát động) là danh sách các phần tử được cấp phát vùng nhớ liền kề nhau trong bộ nhớ. Còn danh sách liên kết là danh sách các phần tử mà vùng nhớ nằm rải rác, không liền kề nhau. Do đó có địa chỉ vùng nhớ của 1 phần tử thì không thể truy xuất đến phần tử tiếp theo bằng cách thông thường (nhảy một khoảng bằng kích thước kiểu dữ liệu của phần tử). Như vậy với danh sách liên kết thì mỗi phần tử phải nắm giữ đường dẫn (địa chỉ) của phần từ tiếp theo bằng biến con trỏ.

Hình ảnh minh họa:

Các thao tác cơ bản

- Khai báo thành phần node: bởi vì một phần tử của danh sách liên kết luôn bao gồm ít nhất là 2 thành phần con (1 chứa dữ liệu và 1 con trỏ giữ địa chỉ phần tử tiếp theo) nên phải sử dụng kiểu dữ liệu cấu trúc (structer) cho phần tử danh sách liên kết.

```
//khai báo kiểu cấu trúc cho phần tử dslk
struct Node
{
	int info;
	Node *pNext;
};
```

- Khai báo cấu trúc danh sách liên kết

```
//khai báo struct danh sách liên kết
struct LinkedList
{
	//dslk đơn chỉ cần giữ 2 node đầu và cuối
	//thực tế chỉ cần giữ node đầu
	//giữ node cuối là để tiện thao tác
	Node *pHead, *pTail;
};
```

- Hàm tạo node mới

```
//hàm tạo Node mới
Node* CreateNode(const int &value)
{
	Node *p = new Node();
	if (p != NULL)
	{
		p->info = value;
		//luôn nhớ gán pNext trỏ đến NULL khi mới tạo Node
		p->pNext = NULL;
	}
	return p;
}
```

- Hàm tạo danh sách liên kết

```
//hàm tạo dslk
LinkedList* CreateLinkedList()
{
    LinkedList *ll = new LinkedList();
    if (ll != NULL)
    {
		ll->pHead = ll->pTail = NULL;
    }
    return ll;
}
```

- Hàm thêm node

```
//hàm thêm node vào dslk
bool AddNode(LinkedList *dslk, const int &value)
{
	//tạo node mới để thêm
	Node *p = CreateNode(value);
	if (p == NULL)
	{
		return false;
	}

	//kiểm tra dslk đã có giá trị chưa
	if (dslk->pHead == NULL)
	{
		//gán là node đầu tiên
		dslk->pHead = dslk->pTail = p;
		return true;
	}

	//dslk đã có giá trị thì thêm vào cuối danh sách
	//gán liên kết
	dslk->pTail->pNext = p;
	//gán phần tử cuối là phần tử mới thêm
	dslk->pTail = p;

	return true;
}
```

- Hàm chèn node vào danh sách liên kết

```
//hàm chèn node vào dslk
bool InsertNode(LinkedList *dslk, Node *node, const int &value)
{
	//tạo node mới để thêm
	Node *p = CreateNode(value);
	if (p == NULL)
	{
		return false;
	}

	//kiểm tra node có trong dslk hay không
	//đây cũng là vòng lặp duyệt dslk cơ bản
	Node *temp = NULL;
	for (temp = dslk->pHead; temp != NULL; temp = temp->pNext)
	{
		if (temp == node)
		{
			break;
		}
	}
	//không tìm thấy
	if (temp == NULL)
	{
		return false;
	}
	//tìm thấy thì bắt đầu chèn
	//giữ lại các liên kết
	p->pNext = node->pNext;
	node->pNext = p;

	return true;
}
```

- Hàm xóa các node có giá trị cho trước

```
//hàm xóa node có giá trị cho trước
int DeleteNode(LinkedList *dslk, const int &value)
{
	int sl = 0;
	//cần phải có được node trước node cần xóa nên duyệt bằng 2 pointer
	Node *pOld = dslk->pHead;
	//kiểm tra node đầu tiên có phải là node cần xóa
	while (pOld != NULL && pOld->info == value)
	{
		dslk->pHead = pOld->pNext;
		//nếu dslk chỉ có 1 phần tử
		if (pOld == dslk->pTail)
		{
			dslk->pTail = pOld->pNext;
		}
		delete pOld;
		sl++;
		pOld = dslk->pHead;
	}
	for (Node *p = pOld->pNext; p != NULL; p = p->pNext)
	{
		if (p->info == value)
		{
```

```
                    Node *pTemp = p;
                    //gán lại các liên kết
                    pOld->pNext = p->pNext;
                    p = pOld;
                    delete pTemp;
                    sl++;
            }
            else
            {
                    pOld = p;
            }
        }
        return sl;
}
```

- Hàm xóa danh sách liên kết

```
//Hàm xóa dslk
void DeleteLinkedList(LinkedList *dslk)
{
        Node *p = dslk->pHead;
        while (p != NULL)
        {
                dslk->pHead = p->pNext;
                delete p;
                p = dslk->pHead;
        }
        delete dslk;
}
```

- Hàm phát sinh danh sách liên kết

```
//hàm phát sinh danh sách liên kết ngẫu nhiên
//có giới hạn số lượng phần tử và khoảng giá trị
LinkedList* GenerateLinkedList(const int &slMin, const int &slMax, const int
&gtMin, const int &gtMax)
{
        LinkedList *dslk = CreateLinkedList();
        if (dslk == NULL)
        {
                return NULL;
        }
        //phát sinh số lượng phần tử
        //lưu ý phả gọi hàm srand 1 lần duy nhất đầu hàm main
        int sl = slMin + rand() % (slMax - slMin + 1);

        for (int i=0; i<sl; ++i)
        {
                AddNode(dslk, gtMin + rand() % (gtMax - gtMin + 1));
        }

        return dslk;
}
```

Bài tập

Bài 1

Thực hiện các yêu cầu sau với danh sách liên kết:

- Làm lại các khai báo và hàm cơ bản về danh sách liên kết như code mẫu.
- Tìm phần tử lớn nhất và nhỏ nhất của danh sách liên kết.
- Tính tổng tất cả các phần tử của danh sách liên kết.
- Tính số lượng phần tử có giá trị lớn nhất.

Bài 2

Viết chương trình phát sinh 2 danh sách liên kết (số lượng phần tử ít nhất là 5, nhiều nhất là 100, giá trị mỗi phần tử nằm trong khoảng [-1000, 2000]). Thực hiện các yêu cầu sau:

- Sắp xếp 2 danh sách liên kết theo thứ tự tăng dần.
- Tạo 1 danh sách liên kết là gộp của 2 danh sách liên kết đã sắp xếp sao cho danh sách liên kết gộp cũng có thứ tự tăng dần.

Bài 3

Viết chương trình phát sinh 1 danh sách liên kết (số lượng phần tử ít nhất 10, nhiều nhất là 100, giá trị mỗi phần tử nằm trong khoảng [-50, 50]). Phát sinh 1 giá trị x trong khoảng [-10, 10] sau đó chia danh sách liên kết thành 2 danh sách liên kết sao cho 1 dslk chứa các phần tử có giá trị nhỏ hơn x, dslk còn lại chứ các phần tử còn lại.