Mục tiêu

Nắm vững các thao tác với danh sách liên kết.

Nội dung

Giải bài tập cũ (Bài 3 tuần 7)

```
//khai báo Node
struct Node
{
	int info;
	Node *pNext;
};

//khai báo dslk
struct List
{
	Node *pHead, *pTail;
};

//hàm khởi tạo dslk
void initList(List &l)
{
	l.pHead = l.pTail = NULL;
}

//hàm tạo Node
Node* CreateNode(const int &x)
{
	Node *p = new Node;
	if (p != NULL)
	{
		p->info = x;
		p->pNext = NULL;
	}
	return p;
}

//hàm thêm node vào cuối dslk
void AddNode(List &l, Node *p)
{
	if (p == NULL)
	{
		return;
	}
	//kiểm tra dslk có rỗng không
	if (l.pHead == NULL)
	{
		l.pHead = l.pTail = p;
	}
	else
	{
		//dslk đã có phần tử thì thêm vào cuối
```

```
			l.pTail->pNext = p;
			l.pTail = p;
		}
}

void Bai3w7(List l, List &l1, List &l2, const int &x)
{
		//duyệt từ đầu ds đến cuối ds
		for (Node *p = l.pHead; p != NULL; p = p->pNext)
		{
			//tạo Node có giá trị để thêm vào ds
			Node *pTemp = CreateNode(p->info);
			//phân chia phần tử
			if (p->info > x)
			{
				//thêm vào dslk 1
				AddNode(l1, pTemp);
			}
			else
			{
				//thêm vào dslk 2
				AddNode(l2, pTemp);
			}
		}
}

void main()
{
		//phát sinh dslk
		List l;
		initList(l);
		//khởi tạo hàm random
		srand((unsigned)time(NULL));
		//phát sinh số lượng phần tử trong khoảng [10, 100]
		int n = 10 + rand() % 91;
		for (int i=0; i<n; ++i)
		{
			//phat sinh node với giá trị trong khoảng [-50, 50]
			Node *pTemp = CreateNode(-50 + rand() % 101);
			AddNode(l, pTemp);
		}

		//phát sinh giá trị x trong khoảng [-10, 10]
		int x = -10 + rand() % 21;

		//chia dslk
		List l1, l2;
		initList(l1);
		initList(l2);
		Bai3w7(l, l1, l2, x);
}
```

Các thao tác với danh sách liên kết

```
// Hàm tìm node đầu tiên trong DSLK cho trước có dữ liệu cho trước
// Đầu vào:   DSLK (l), dữ liệu của node cần tìm (info)
// Đầu ra:    Con trỏ đến node tìm được (trả về NULL nếu không tìm được)
Node* GetNodePointer(List l, Data info)
```

```
{
	Node *p = l.pHead;
	if(p == NULL)
		return p;
	while(p != NULL && CompareData(p->info, info) != 0)
	{
		p = p->pNext;
	}
	return p;
}

// Hàm tìm node có chỉ số (bắt đầu từ 0) cho trước
// Đầu vào:   DSLK (l), chỉ số của node cần lấy (index)
// Đầu ra:    Con trỏ đến node tìm được (trả về NULL nếu không tìm được)
Node* GetNodePointer(List l, int index)
{
	Node *p = l.pHead;
	int i = 0;
	while(p!=NULL && i<index)
	{
		p = p->pNext;
		++i;
	}
	return p;
}

// Hàm xác định vị trí của một node cho trước trong DSLK cho trước
// Đầu vào:   DSLK (l), con trỏ đến node cần xác định vị trí (pNode)
// Đầu ra:    Thứ tự của node cho trước (trả về -1 nếu node này không có trong
DSLK)
int GetNodeIndex(List l, Node *pNode)
{
	if(l.pHead == NULL)
		return -1;
	Node *p = l.pHead;
	int i = 0;
	while(p!=NULL)
	{
		if(p==pNode)
			return i;
		p=p->pNext;
		++i;
	}
	return -1;
}

// Hàm xác định con trỏ đến node đứng trước của một node cho trước trong DSLK
// Đầu vào:   DSLK (l), con trỏ đến node cho trước (pNode)
// Đầu ra:    Con trỏ đến node tìm được (trả về NULL nếu không tìm được)
Node* GetPreviousNodePointer(List l, Node *pNode)
{
	Node *p = l.pHead;
	Node *pOld = NULL;
	while(p!=NULL)
	{
		if(p == pNode)
			return pOld;
```

```
		pOld = p;
		p=p->pNext;
	}
	return NULL;
}

// Hàm chèn một node cho trước vào đầu DSLK
// Đầu vào:	Tham chiếu đến DSLK (l), con trỏ đến node cần chèn (pNewNode)
// Đầu ra:	Không có
void AddHead(List &l, Node *pNewNode)
{
	pNewNode->pNext = l.pHead;
	l.pHead = pNewNode;
	if(l.pTail == NULL)
		l.pTail = l.pHead;
}

// Hàm chèn một node có dữ liệu cho trước vào đầu DSLK
// Đầu vào:	Tham chiếu đến DSLK (l), dữ liệu của node cần chèn (info)
// Đầu ra:	Con trỏ đến node được chèn (trả về NULL nếu không chèn được)
Node* AddHead(List &l, Data info)
{
	Node *p = CreateNode(info);
	if(p!=NULL)
		AddHead(l, p);
	return p;
}

// Hàm chèn một node cho trước vào cuối DSLK
// Đầu vào:	Tham chiếu đến DSLK (l), con trỏ đến node cần chèn (pNewNode)
// Đầu ra:	Không có
void AddTail(List &l, Node *pNewNode)
{
	if(l.pTail==NULL)
	{
		l.pHead = pNewNode;
		l.pTail = pNewNode;
	}
	else
	{
		l.pTail->pNext = pNewNode;
		l.pTail = pNewNode;
	}
	l.pTail->pNext = NULL;
}

// Hàm chèn một node có dữ liệu cho trước vào cuối DSLK
// Đầu vào:	Tham chiếu đến DSLK (l), dữ liệu của node cần chèn (info)
// Đầu ra:	Con trỏ đến node được chèn (trả về NULL nếu không chèn được)
Node* AddTail(List &l, Data info)
{
	Node *p = CreateNode(info);
	if(p!=NULL)
		AddTail(l, p);
	return p;
}
```

```
// Hàm chèn một node cho trước vào sau một node khác cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến node cho trước (q), con trỏ đến
node cần chèn (pNewNode)
// Đầu ra:    Không có
void AddAfter(List &l, Node *q, Node *pNewNode)
{
	Node *p = l.pHead;
	if(p==NULL || q==NULL)
		return;
	while(p!=NULL)
	{
		if(p==q)
			break;
	}
	if(p==NULL)
		return;
	p=q->pNext;
	q->pNext = pNewNode;
	p->pNext = p;
	if(p==NULL)
		l.pTail = pNewNode;
}

// Hàm chèn một node có dữ liệu cho trước vào sau một node khác cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến node cho trước (q), dữ liệu của
node cần chèn (info)
// Đầu ra:    Con trỏ đến node được chèn (trả về NULL nếu không chèn được)
Node* AddAfter(List &l, Node *q, Data info)
{
	if(q==NULL)
		return NULL;
	Node *p = CreateNode(info);
	if(p!=NULL)
	{
		AddAfter(l, q, p);
	}
	if(q->pNext != p)
		return NULL;
	return p;
}

// Hàm chèn một node cho trước vào trước một node khác cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến node cho trước (q), con trỏ đến
node cần chèn (pNewNode)
// Đầu ra:    Không có
void AddBefore(List &l, Node *q, Node *pNewNode)
{
	if(q==NULL || pNewNode==NULL)
		return;
	if(q==l.pHead)
	{
		pNewNode->pNext = l.pHead;
		l.pHead = pNewNode;
		return;
	}
	Node *p = GetPreviousNodePointer(l, q);
	if(p==NULL)
```

```
		return;
	p->pNext = pNewNode;
	pNewNode->pNext = q;
}

// Hàm chèn một node có dữ liệu cho trước vào trước một node khác cho trước
// Đầu vào:	Tham chiếu đến DSLK (l), con trỏ đến node cho trước (q), dữ liệu của node cần chèn (info)
// Đầu ra:	Con trỏ đến node được chèn (trả về NULL nếu không chèn được)
Node* AddBefore(List &l, Node *q, Data info)
{
	if(q==NULL)
		return NULL;
	Node *p = CreateNode(info);
	if(p!=NULL)
	{
		AddBefore(l, q, p);
		if(p->pNext!=q)
			return NULL;
	}
	return p;
}

// Hàm chèn một node cho trước sao cho DSLK cho trước vẫn tăng dần
// Đầu vào: Tham chiếu đến DSLK (l), con trỏ đến node cần chèn (pNewNode)
// Đầu ra:	Không có
void AddAscendingList(List &l, Node *pNewNode)
{
	if(pNewNode==NULL)
		return;
	Node *p = l.pHead;
	while(p!=NULL)
	{
		if(CompareData(p->info, pNewNode->info) > -1)
		{
			AddBefore(l, p, pNewNode);
			return;
		}
	}
	if(p==NULL && l.pTail!=NULL)
	{
		l.pTail->pNext = pNewNode;
		l.pTail = pNewNode;
	}
}

// Hàm chèn một node có dữ liệu cho trước sao cho DSLK cho trước vẫn tăng dần
// Đầu vào: Tham chiếu đến DSLK (l), dữ liệu của node cần chèn (info)
// Đầu ra:	Con trỏ đến node được chèn (trả về NULL nếu không chèn được)
Node* AddAscendingList(List &l, Data info)
{
	Node *p = CreateNode(info);
	if(p!=NULL)
	{
		AddAscendingList(l, p);
	}
	return p;
```

```
}

// Hàm hủy một node đầu DSLK
// Đầu vào:   Tham chiếu đến DSLK (l)
// Đầu ra:    Dữ liệu của node bị xóa
Data RemoveHead(List &l)
{
	Data kq;
	if(l.pHead == NULL)
		return kq;
	Node *p = l.pHead;
	l.pHead = l.pHead->pNext;
	if(l.pHead==NULL)
		l.pTail=NULL;
	kq.x = p->info.x;
	delete p;
	return kq;
}

// Hàm hủy một node cuối DSLK
// Đầu vào:   Tham chiếu đến DSLK (l)
// Đầu ra:    Dữ liệu của node bị xóa
Data RemoveTail(List &l)
{
	Data kq;
	if(l.pTail==NULL)
		return kq;
	Node *p = GetPreviousNodePointer(l, l.pTail);
	kq.x = l.pTail->info.x;
	delete l.pTail;
	l.pTail = p;
	if(p==NULL)
		l.pHead=p;
	return kq;
}

// Hàm hủy một node đứng sau một node cho trước trong DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến node cho trước (q)
// Đầu ra:    Không có
Data RemoveAfter(List &l, Node *q)
{
	if(q==NULL)
		return RemoveHead(l);
	Data kq;
	Node *p = l.pHead;
	while(p!=NULL)
	{
		if(p==q)
			break;
		p=p->pNext;
	}
	if(p!=NULL)
	{
		p = q->pNext;
		if(p!=NULL)
		{
			q->pNext = p->pNext;
```

```
                    kq.x = p->info.x;
                    delete p;
                    if(q->pNext == NULL)
                        l.pTail = q;
                }
        }
        return kq;
}

// Hàm hủy một node có dữ liệu cho trước trong DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), dữ liệu của node cần xóa (info)
// Đầu ra:    Tìm thấy node có dữ liệu info hay không (true/false)
bool RemoveNode(List &l, Data info)
{
        Node *p = GetNodePointer(l, info);
        if(p==NULL)
                return false;
        Node *q = GetPreviousNodePointer(l, p);
        if(q==NULL)
        {
                l.pHead = p->pNext;
                delete p;
                if(l.pHead==NULL)
                        l.pTail = l.pHead;
                return true;
        }
        q->pNext = p->pNext;
        delete p;
        if(q->pNext==NULL)
                l.pTail=q;
        return true;
}
```

Bài tập

Bài 1

Cài đặt hàm main để thử nghiệm các hàm đã cài đặt ở trên. Các giá trị cần phát sinh ngẫu nhiên, không nhập tay.

Bài 2

Cài đặt hàm sắp xếp danh sách liên kết đơn.

Bài 3

Dựa vào cấu trúc của danh sách liên kết để cái đặt cấu trúc **STACK** với các hàm tương ứng (**Init**, **Pop**, **Push**, **IsEmpty**, **Clear**). [**STACK** là một cấu trúc bộ chứa dữa liệu với nguyên tắc **LIFO** (*Last In First Out*), nghĩa là phần tử nào vào sau cùng thì sẽ được lấy ra đầu tiên ví như cọc để CD, cái CD cuối cùng bỏ vào cọc nằm trên cùng nên được lấy ra đầu tiên].

Bài 4

Dựa vào cấu trúc của danh sách liên kết để cài đặt cấu trúc **QUEUE** với các hàm tương ứng (**Init**, **Pop**, **Push**, **IsEmpty**, **Clear**). [ngược với **STACK**, **QUEUE** là một cấu trúc bộ chứa dữ liệu với nguyên tắc **FIFO** (*First In First Out*), nghĩa là phần tử nào vào đầu tiên thì sẽ được lấy ra đầu tiên ví như ống cầu lông, trái cầu bỏ vào đầu tiên sẽ được lấy ra đầu tiên].