Mục tiêu

- Áp dụng danh sách liên kết cho các nhu cầu cần mảng động (dễ dàng mở rộng kích thước).
- Đọc ghi file nhị phân.

Nội dung

Giải bài tập tuần 8

Cài đặt cấu trúc Stack

Để đảm bảo nguyên tắc LIFO của Stack ta chỉ cần vận dùng các hàm của dslk sao cho phần tử thêm vào cuối cùng sẽ được lấy ra đầu tiên. Ở đây ta sẽ sử dụng thêm vào đầu dslk và lấy ra từ đầu dslk.

```
//định nghĩa cấu trúc chứa dữ liệu
struct Data
{
	int x;
};

//định nghĩa cấu trúc node
struct Node
{
	Data info;
	Node *pNext;
};

//định nghĩa cấu trúc dslk
struct List
{
	Node *pHead, *pTail;
};

//định nghĩa cấu trúc stack
struct Stack
{
	List l;
};

// 01. Hàm so sánh 2 biến kiểu cấu trúc cho trước
// Đầu vào: biến cấu trúc (info1) và biến cấu trúc (info2)
// Đầu ra:    0 (bằng nhau), -1 (info1 nhỏ hơn info2), 1 (info1 lớn hơn info2)
int CompareData(Data info1, Data info2);

// 02. Hàm khởi tạo DSLK (rỗng)
// Đầu vào:   Tham chiếu đến DSLK cần khởi tạo (l)
// Đầu ra:    Không có
void InitList(List &l);
```

```
// 03. Hàm kiểm tra DSLK cho trước có phải là DSLK rỗng hay không?
// Đầu vào:   DSLK cần kiểm tra (l)
// Đầu ra:    DSLK rỗng hay không (true/false)
bool IsEmptyList(List l);

// 04. Hàm in DSLK cho trước
// Đầu vào: DSLK cần in (l)
// Đầu ra:    Không có
void PrintList(List l);

// 05. Hàm tạo một nút mới với dữ liệu cho trước
// Đầu vào:   Dữ liệu của nút (info)
// Đầu ra:    Con trỏ đến nút đó (trả về NULL nếu không tạo được)
Node* CreateNode(Data info);

// 06. Hàm tìm nút đầu tiên trong DSLK cho trước có dữ liệu cho trước
// Đầu vào:   DSLK (l), dữ liệu của nút cần tìm (info)
// Đầu ra:    Con trỏ đến nút tìm được (trả về NULL nếu không tìm được)
Node* GetNodePointer(List l, Data info);

// 07. Hàm tìm nút có chỉ số (bắt đầu từ 0) cho trước
// Đầu vào:   DSLK (l), chỉ số của nút cần lấy (index)
// Đầu ra:    Con trỏ đến nút tìm được (trả về NULL nếu không tìm được)
Node* GetNodePointer(List l, int index);

// 08. Hàm xác định vị trí của một nút cho trước trong DSLK cho trước
// Đầu vào:   DSLK (l), con trỏ đến nút cần xác định vị trí (pNode)
// Đầu ra:    Thứ tự của nút cho trước (trả về -1 nếu nút này không có trong DSLK)
int GetNodeIndex(List l, Node *pNode);

// 09. Hàm xác định con trỏ đến nút đứng trước của một nút cho trước trong DSLK
// Đầu vào:   DSLK (l), con trỏ đến nút cho trước (pNode)
// Đầu ra:    Con trỏ đến nút tìm được (trả về NULL nếu không tìm được)
Node* GetPreviousNodePointer(List l, Node *pNode);

// 10. Hàm chèn một nút cho trước vào đầu DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cần chèn (pNewNode)
// Đầu ra:    Không có
void AddHead(List &l, Node *pNewNode);

// 11. Hàm chèn một nút có dữ liệu cho trước vào đầu DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), dữ liệu của nút cần chèn (info)
// Đầu ra:    Con trỏ đến nút được chèn (trả về NULL nếu không chèn được)
Node* AddHead(List &l, Data info);

// 12. Hàm chèn một nút cho trước vào cuối DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cần chèn (pNewNode)
// Đầu ra:    Không có
void AddTail(List &l, Node *pNewNode);

// 13. Hàm chèn một nút có dữ liệu cho trước vào cuối DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), dữ liệu của nút cần chèn (info)
// Đầu ra:    Con trỏ đến nút được chèn (trả về NULL nếu không chèn được)
Node* AddTail(List &l, Data info);

// 14. Hàm chèn một nút cho trước vào sau một nút khác cho trước
```

```
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cho trước (q), con trỏ đến
nút cần chèn (pNewNode)
// Đầu ra:    Không có
void AddAfter(List &l, Node *q, Node *pNewNode);

// 15. Hàm chèn một nút có dữ liệu cho trước vào sau một nút khác cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cho trước (q), dữ liệu của
nút cần chèn (info)
// Đầu ra:    Con trỏ đến nút được chèn (trả về NULL nếu không chèn được)
Node* AddAfter(List &l, Node *q, Data info);

// 16. Hàm chèn một nút cho trước vào trước một nút khác cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cho trước (q), con trỏ đến
nút cần chèn (pNewNode)
// Đầu ra:    Không có
void AddBefore(List &l, Node *q, Node *pNewNode);

// 17. Hàm chèn một nút có dữ liệu cho trước vào trước một nút khác cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cho trước (q), dữ liệu của
nút cần chèn (info)
// Đầu ra:    Con trỏ đến nút được chèn (trả về NULL nếu không chèn được)
Node* AddBefore(List &l, Node *q, Data info);

// 18. Hàm chèn một nút cho trước sao cho DSLK cho trước vẫn tăng dần
// Đầu vào: Tham chiếu đến DSLK (l), con trỏ đến nút cần chèn (pNewNode)
// Đầu ra:    Không có
void AddAscendingList(List &l, Node *pNewNode);

// 19. Hàm chèn một nút có dữ liệu cho trước sao cho DSLK cho trước vẫn tăng dần
// Đầu vào: Tham chiếu đến DSLK (l), dữ liệu của nút cần chèn (info)
// Đầu ra:    Con trỏ đến nút được chèn (trả về NULL nếu không chèn được)
Node* AddAscendingList(List &l, Data info);

// 20. Hàm hủy một nút đầu DSLK
// Đầu vào:   Tham chiếu đến DSLK (l)
// Đầu ra:    Dữ liệu của nút bị xóa
Data RemoveHead(List &l);

// 21. Hàm hủy một nút cuối DSLK
// Đầu vào:   Tham chiếu đến DSLK (l)
// Đầu ra:    Dữ liệu của nút bị xóa
Data RemoveTail(List &l);

// 22. Hàm hủy một nút đứng sau một nút cho trước trong DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), con trỏ đến nút cho trước (q)
// Đầu ra:    Không có
Data RemoveAfter(List &l, Node *q);

// 23. Hàm hủy một nút có dữ liệu cho trước trong DSLK
// Đầu vào:   Tham chiếu đến DSLK (l), dữ liệu của nút cần xóa (info)
// Đầu ra:    Tìm thấy nút có dữ liệu info hay không (true/false)
bool RemoveNode(List &l, Data info);

// 24. Hàm hủy toàn bộ DSLK cho trước
// Đầu vào:   Tham chiếu đến DSLK (l)
// Đầu ra:    Không có
void RemoveList(List &l);
```

```
// 25. Hàm xóa tất cả các nút có khóa k cho trước
// Đầu vào:   Tham chiếu đến DSLK (l), giá trị khóa k.
// Đầu ra:    Số lượng nút đã bị xóa
int RemoveKeys(List &l, Data k);

// 26. Hàm sắp xếp dslk đơn theo thuật toán đổi chỗ trực tiếp
// Đầu vào:   Tham chiếu đến DSLK (l).
// Đầu ra:    Không có.
void SortList(List &l)
{
	Node *p = l.pHead;
	if(p == NULL || p->pNext == NULL)
		return;
	for(; p->pNext->pNext!=NULL; p=p->pNext)
	{
		for(Node *q=p->pNext; q->pNext!=NULL; q=q->pNext)
		{
			if(p->info.x > q->info.x)
			{
				HoanViNode(l, p, q);
			}
		}
	}
}

// 27. Hàm gộp 2 dslk đơn cho trước thành 1 dslk đơn đảm bảo thứ tự tăng.
// Đầu vào:   Tham chiếu đến 2 DSLK (l1, l2).
// Đầu ra:    dslk đã được gộp
List CombineList(const List &l1, const List &l2)
{
	List l;
	InitList(l);
	if(l1.pHead == NULL)
	{
		if(l2.pHead == NULL)
		{
			return l;
		}
		else
		{
			l.pHead = l2.pHead;
			l.pTail = l2.pTail;
			return l;
		}
	}
	else
	{
		l.pHead = l1.pHead;
		l.pTail = l1.pTail;
	}
	Node *p = l2.pHead;
	while(p!=NULL)
	{
		AddAscendingList(l, p);
		p=p->pNext;
	}
```

```
	return l;
}

// 28. Hàm đảo dslk
// Đầu vào:   Tham chiếu đến DSLK (l).
// Đầu ra:    Không có
void ReverseList(List &l)
{
	if(l.pHead==NULL)
		return;
	Node *p = l.pHead;
	Node *q = NULL;
	Node *oldTail = l.pTail;
	while(p!=oldTail)
	{
		q = p->pNext;
		p->pNext = NULL;
		l.pTail->pNext = p;
		l.pTail = p;
		p = q;
	}
}

//29. Hàm hoán vị 2 node trên dslk
//Đầu vào: dslk l, node p và node q thuộc dslk, node p đứng trước node q
//Đầu ra: không có
void HoanViNode(List &l, Node *p, Node *q)
{
	//lấy các node trước và sau node cần hoán vị
	Node *pPre = GetPreviousNodePointer(l, p);
	Node *pNext = p->pNext;
	Node *qPre = GetPreviousNodePointer(l, q);
	Node *qNext = q->pNext;

	if(pPre != NULL)
	{
		pPre->pNext = q;
	}
	else
	{
		l.pHead = q;
	}
	if(pNext == q)
	{
		q->pNext = p;
	}
	else
	{
		q->pNext = pNext;
		qPre->pNext = p;
	}
	p->pNext = qNext;
	if(qNext == NULL)
	{
		l.pTail = p;
	}
}
```

```
//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////STACK/////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
//dùng dslk luôn nhớ phải khởi tạo trước.
void InitStack(Stack &s)
{
	InitList(s.l);
}

//kiểm tra Stack rỗng hay không
bool Stack_IsEmpty(const Stack &s)
{
	return IsEmptyList(s.l);
}

//Pop nghĩa là lấy phần tử được đưa vào cuối cùng cho Stack (LIFO)
Data Stack_Pop(Stack &s)
{
	Data info;
	if(!Stack_IsEmpty(s))
	{
		info = RemoveHead(s.l);
	}
	return info;
}

//đưa phần tử mới vào Stack
bool Stack_Push(Stack &s, const Data &info)
{
	Node *p = CreateNode(info);
	if(p==NULL)
		return false;
	AddHead(s.l, p);
	return true;
}

//xóa tất cả các phần tử mà Stack đang chứa.
void Stack_Clear(Stack &s)
{
	RemoveList(s.l);
}
```

Con trỏ hàm

- ✓ Tình huống chúng ta cần sắp xếp dslk chứa các sinh viên như sau

```
struct SinhVien
{
	char *MSSV;
	char *HoTen;
};

struct Node
{
	SinhVien info;
	Node *pNext;
};
```

```
struct List
{
	Node *pHead;
	Node *pTail;
};

void SortList(List &l)
{
	Node *p = l.pHead;
	if(p == NULL || p->pNext == NULL)
		return;
	for(; p->pNext->pNext!=NULL; p=p->pNext)
	{
		for(Node *q=p->pNext; q->pNext!=NULL; q=q->pNext)
		{
			if(strcmp(p->info.HoTen, q->info.HoTen))
			{
				//hoán vị vị trí 2 node p và q
			}
		}
	}
}
```

✓ Giả sử bây giờ chúng ta lại có nhu cầu sắp xếp sinh viên theo mssv thì sao? Chỉ thay đổi chỗ highlight nhưng lại phải viết nguyên hàm mới. Giải pháp ở đây là sử dụng con trỏ hàm.

```
typedef int (*SoSanh)(const SinhVien&, const SinhVien&);

void SortList(List &l, SoSanh sfunc)
{
	Node *p = l.pHead;
	if(p == NULL || p->pNext == NULL)
		return;
	for(; p->pNext->pNext!=NULL; p=p->pNext)
	{
		for(Node *q=p->pNext; q->pNext!=NULL; q=q->pNext)
		{
			if(sfunc(p->info, q->info) > 0)
			{
				//hoán vị vị trí 2 node p và q
			}
		}
	}
}

int SoSanhHoTen(const SinhVien &sv1, const SinhVien &sv2)
{
	return strcmp(sv1.HoTen, sv2.HoTen);
}

int SoSanhMSSV(const SinhVien &sv1, const SinhVien &sv2)
{
	return strcmp(sv1.MSSV, sv2.MSSV);
}

void main()
{
```

```
        List l;
        //...
        //Sắp xếp ds sinh viên theo học tên
        SortList(l, SoSanhHoTen);
        //Sắp xếp ds sinh viên theo mssv
        SortList(l, SoSanhMSSV);
}
```

Đọc ghi file kiểu nhị phân

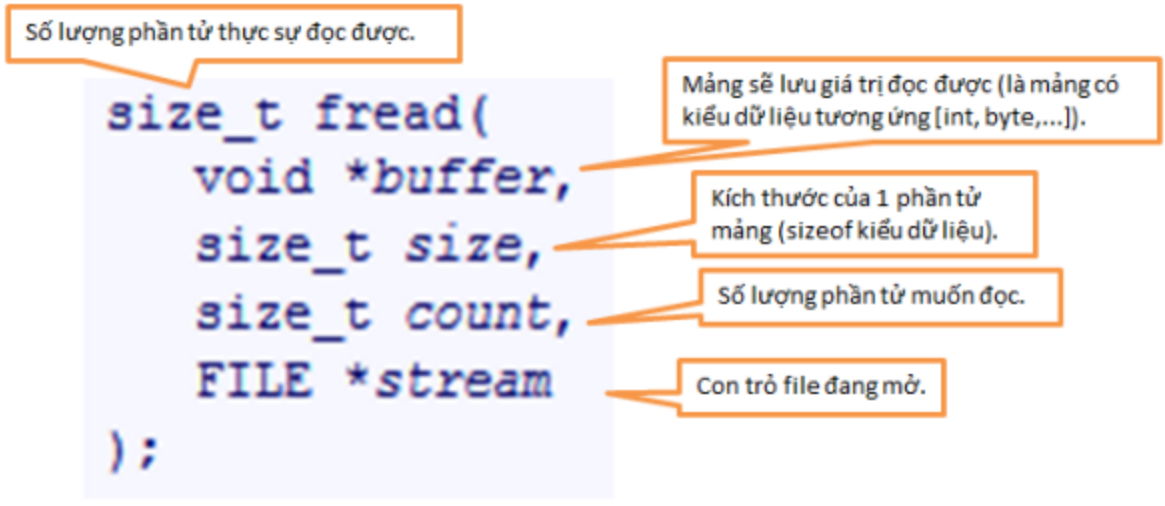
- ✓ Dữ liệu ghi lên file theo các byte nhị phân giống như bộ nhớ, trong quá trình nhập xuất, dữ liệu không bị thay đổi (nghĩa là trên file có 4 bytes lưu 1 số int thì khi đọc vào cũng là 1 số int 4 bytes).
- ✓ Khi đọc đến cuối file thì sẽ gặp ký tự kết thúc file EOF (kiểm tra bằng hàm feof).

Lưu ý

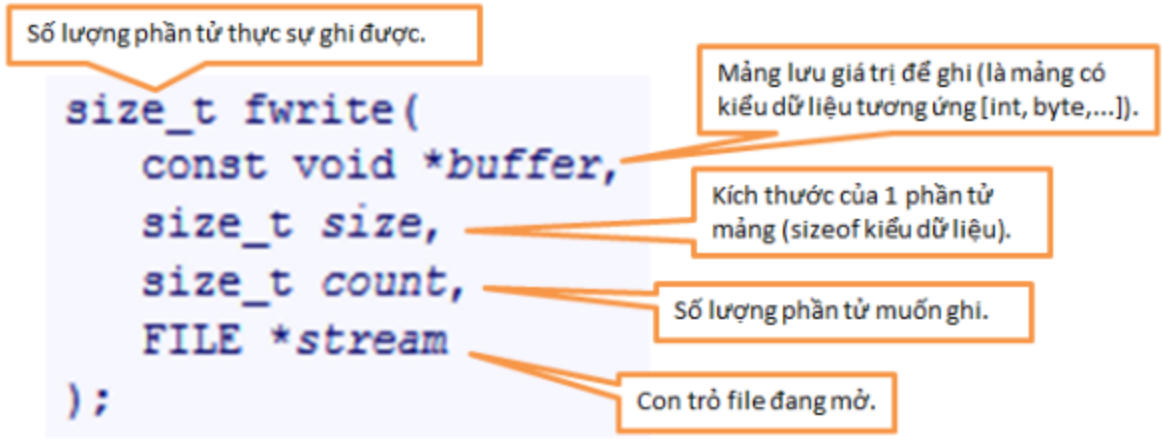
- Như đã bàn, bản chất file là tập hợp các byte, vì thế file ghi theo kiểu nhị phân hay văn bản là như nhau, chỉ có ý nghĩa dữ liệu là khác nhau (tức là vẫn có thể dễ dàng đọc file theo kiểu khác, vẫn lấy được dữ liệu nhưng việc có được ý nghĩa dữ liệu hay không mới là điều quan trọng).
- Ví dụ: file được ghi theo kiểu văn bản số n= 2345, khi mở để đọc theo kiểu nhị phân thì ta lại được giá trị số là: 892613426.

Thao tác đọc ghi file kiểu nhị phân

- ✓ Đọc



- ✓ Ghi



✓ Ví dụ

```c
#include <stdio.h>
#include <conio.h>

void main()
{
	FILE* f = NULL;
	if((f = fopen("test.txt", "w+b"))!=NULL)
	{
		int n=2345;
		//để ghi giá trị ra file cần có mảng làm bộ đệm
		int k[1];
		k[0] = n;
		//ghi giá trị ra file
		fwrite(k, sizeof(int), 1, f);
		//dịch chuyển đầu đọc về đầu file
		fseek(f, 0, SEEK_SET);
		//đọc giá trị vừa ghi
		fread(k, sizeof(int), 1, f);
		//sau khi hoàn thành thì đóng file
		fclose(f);
		int t = k[0];
		printf("%d", t);
		getch();
	}
}
```

Bài tập

Bài 1

a) Cài đặt thêm hàm chia dslk thành 2 phần có số lượng phần tử gần bằng nhau (chia đôi, nếu số phần tử lúc đầu không chẵn thì hơn kém nhau 1 phần tử).

b) Cài đặt thêm hàm chia dslk thành 2 phần dựa vào con trỏ p làm mốc cho trước.

Bài 2 (tính thành 10 bài)

Cài đặt chương trình sử dụng dslk để quản lý một lớp học. Mỗi sinh viên trong lớp học có các thông tin sau:

- Họ tên: tối đa 30 ký tự.
- Mã số sinh viên: tối đa 8 ký tự. Mỗi sinh viên có mã số duy nhất, không trùng nhau.
- Điểm trung bình: số thực trong khoảng từ 0 đến 10.
- Các chức năng mà chương trình yêu cầu:
  - 1, Khởi tạo lớp học.
  - 2, Thêm sinh viên vào lớp học.
  - 3, Xuất danh sách sinh viên của lớp học ra màn hình với số thông tin yêu cầu (chỉ xuất tên, chỉ xuất tên và mã số, xuất hết thông tin). Khuyến khích sử dụng con trỏ hàm.
  - 4, Đọc, ghi dữ liệu Lớp học ra file nhị phân.
  - 5, Sắp xếp sinh viên trong lớp theo tiêu chí: mssv, họ tên, điểm trung bình. Khuyến khích sử dụng con trỏ hàm.
  - 6, Đuổi học một số sinh viên (theo mssv, theo điểm trung bình (những sinh viên <5 thì đuổi chẳng hạn)).
  - 7, Xóa lớp học (đuổi hết sinh viên).
  - 8, Thêm một sinh viên vào lớp học đã được sắp xếp theo tiêu chí cụ thể (mssv, họ tên, điểm trung bình). Khuyến khích sử dụng con trỏ hàm.
  - 9, Tách 1 lớp học ra thành 2 lớp có số lượng sinh viên gần bằng nhau.
  - 10, Gộp 2 lớp học thành 1 lớp học.