Đại Học Công Nghiệp Tp. Hồ Chí Minh
Khoa Công nghệ Thông tin

BÀI TẬP THỰC HÀNH

Môn Cấu trúc Dữ liệu & Giải thuật

1. Ngôn ngữ cài đặt C/C++

2. Danh sách bài tập

Phần I: Bài tập ôn

Bài 1: Ôn tập các thao tác trên mảng một chiều

1. Cho mảng a có n phần tử số nguyên. Viết các hàm thực hiện các công việc: nhập/ xuất mảng, phát sinh mảng, đọc/ ghi mảng vào file.

```
void NhapMang(int a[], int n)
{
	for(int i=0; i<n; i++)
	{
		printf("a[%d]=", i);
		scanf("%d", &a[i]);
	}
}
void PhatSinhMang(int a[], int n)
{
	srand(time(NULL));
	for(int i=0; i<n; i++)
		a[i] = rand();
}
void XuatMang(int a[], int n)
{
	printf("\n");
	for(int i=0; i<n; i++)
		printf("%10d", a[i]);
}
```

```
//Ghi mảng a có n phần tử vào file text
int GhiMangVaoFileText(char* filename, int a[], int n)
{
	FILE* f = fopen(filename, "w");
	if(!f) //Không mở file để ghi được
		return 0;
	for(int i=0; i<n; i++)
		fprintf(f ,"%d\t",a[i]); //Ghi từng phần tử
					a[i] vào file, cách nhau một tab
	fclose(f);
	return 1; //Ghi file thành công trả về 1
}
//Đọc file text vào mảng a
int DocFileTextVaoMang(char* filename, int a[], int &n)
{
	FILE* f = fopen(filename, "r");
	if(!f) //Không mở file được
		return 0;
	int i=0;
	while(!feof(f)) //Trong khi chưa hết file
	{
		fscanf(f,"%d",&a[i]); //Đọc từng PT vào mảng
		i++; //đếm số phần tử
	}
	n = i;
}
```

2. Viết chương trình nhập vào danh sách gồm n sinh viên, in ra những sinh viên có điểm trung bình >=5 và cho biết số lượng sinh viên nam. Thông tin của mỗi sinh viên gồm: mã số, họ tên, điểm trung bình và giới tính.
3. Tính thời gian chạy của chương trình.

```
#include <time.h>
clock_t start, end;

start = clock();

// doạn code cần tính thời gian

end = clock();
double time_used = (end - start)/ CLOCK_PER_SEC;
printf("Thoi gian: %f giay.", time_used);

// CLOCKS_PER_SEC: là một hằng số macro đại diện cho số
clock tick mỗi giây (Windows thì 1 tick = 1/64 giây, nên
CLOCKS_PER_SEC = 64, còn Linux thì 1 tick = 1 ms hay
CLOCKS_PER_SEC = 1000).
```

Phần II: Bài tập tìm kiếm và sắp xếp trên mảng

Bài 2: Viết chương trình cài đặt giải thuật tìm kiếm **tuyến tính**.

Hướng dẫn: Viết các hàm sau:

- Tạo ngẫu nhiên mảng một chiều số nguyên gồm N phần tử:

  void PhatSinhMang (int a[], int N)
- In mảng ra màn hình: **void XuatMang(int a[], int N)**
- Tìm tuyến tính: **int TimTuyenTinh(int a[], int N, int X)**
- **int TimTuyenTinh_CT(int a[], int N, int X)**
- Hàm main( ):
  - Phát sinh mảng a với kích thước N.
  - Xuất mảng xem kết quả phát sinh.
  - Nhập giá trị cần tìm x.
  - Tìm x theo phương pháp tìm tuyến tính.
  - In kết quả tìm: Nếu tìm thấy thì cho biết vị trí tìm thấy, ngược lại in kết quả không tìm thấy.

Bài 3: Viết chương trình cài đặt giải thuật tìm kiếm **nhị phân**.

Hướng dẫn: Viết các hàm sau:

- Tạo ngẫu nhiên mảng một chiều số nguyên gồm N phần tử tăng dần (không cần sắp xếp):

void PhatSinhMangTang (int a[], int N)

- In mảng ra màn hình: **void XuatMang(int a[], int N)**
- Tìm tuyến tính: **int TimNhiPhan(int a[], int N, int X)**
- Hàm main( ):
  - Phát sinh mảng a tăng dần với kích thước N.
  - Xuất mảng xem kết quả phát sinh.
  - Nhập giá trị cần tìm x.
  - Tìm x theo phương pháp tìm nhị phân.
  - In kết quả tìm: Nếu tìm thấy thì cho biết vị trí tìm thấy, ngược lại in kết quả không tìm thấy.

Bài 4: Bổ sung **Bài 2, Bài 3** sao cho chương trình phải xác định được số lần so sánh và vị trí tìm thấy (nếu có) của phần tử cần tìm.

Gợi ý: Thay đổi 2 hàm tìm trong **Bài 2, Bài 3** như sau:

- Tìm tuyến tính có chèn vào giá trị ss tính số lần so sánh:
  int TimTuyenTinh(int a[], int N, int X, int &ss)
  int TimTuyenTinh_CT(int a[], int N, int X, int &ss)
- Tìm nhị phân có chèn vào giá trị ss tính số lần so sánh:
  int TimNhiPhan(int a[], int N, int X, int &ss)
- Hàm main():
  - Phát sinh mảng tăng a với kích thước N cho trước.
  - Xuất mảng xem kết quả phát sinh.
  - Nhập giá trị cần tìm x
  - Tìm x theo 2 phương pháp
  - In kết quả tìm: Gồm vị trí (nếu tìm thấy x) và số lần so sánh cho từng phương pháp.

Bài 5: Áp dụng giải thuật tìm kiếm cho bài toán tổng quát: Nếu dãy không có thứ tự thì tìm tuyến tính, ngược lại thì áp dụng phương pháp tìm nhị phân.

Hướng dẫn: viết các hàm sau:

- Hàm phát sinh mảng ngẫu nhiên: **void PhatSinhMang(int a[], int N)**
- In mảng ra màn hình: **void XuatMang(int a[], int N)**
- Hàm tìm tuyến tính: **int TimTuyenTinh(int a[], int N, int X)**

- Hàm tìm nhị phân cho trường hợp dãy tăng: **int TimNhiPhan(int a[], int N, int X)**
- Tìm nhị phân cho trường hợp dãy giảm dần: **int TimNhiPhan2(int a[], int N, int X)**
- Hàm kiểm tra mảng có thứ tự tăng (trả về ***true***: nếu tăng, ngược lại trả về ***false***)
  bool KiemTraTang(int a[], int N)
- Hàm kiểm tra mảng có thứ tự giảm (trả về ***true***: nếu giảm, ngược lại trả về ***false***)
  bool KiemTraGiam(int a[], int N)

i) Hàm chính (main()):

- Phát sinh mảng a với kích thước N cho trước.
- Xuất mảng xem kết quả phát sinh.
- Nhập giá trị cần tìm x
- Kiểm tra nếu mảng có thứ tự tăng thì gọi hàm **TimNhiPhan**
  Ngược lại, nếu mảng có thứ tự giảm thì gọi hàm **TimNhiPhan2**
  Trường hợp còn lại thì gọi hàm **TimTuyenTinh** (*mảng không có thứ tự*)
- In kết quả

Bài 6: Cài đặt các giải thuật sắp xếp theo các phương pháp:

1. Chọn trực tiếp.
2. Chèn trực tiếp.
3. Đổi chỗ trực tiếp.
4. Nổi bọt.
5. **Quicksort.**

Theo yêu cầu sau:

- *Dữ liệu thử phát sinh ngẫu nhiên*
- *In ra màn hình kết quả chạy từng bước của từng giải thuật.*
- *Tính số lần so sánh và số phép gán của từng giải thuật.*
- *Tính thời gian chạy.*

Bài 7: Cho mảng 1 chiều quản lý thông tin các sinh viên của 1 lớp học (tối đa 50 sinh viên). Mỗi sinh viên gồm các thông tin: MSSV, họ và tên, giới tính, địa chỉ và điểm trung bình. Viết chương trình thực hiện các yêu cầu sau:

1. Nhập danh sách sinh viên.
2. In ra danh sách sinh viên.

3. Xóa 1 sinh viên với mã số x cho trước khỏi danh sách.
4. Sắp xếp danh sách sinh viên theo thứ tự tăng dần của điểm trung bình
5. Sắp xếp danh sách sinh viên theo thứ tự tăng dần của họ và tên

Hướng dẫn:

- Khai báo cấu trúc thông tin sinh viên:

```
struct ttsinhvien
{   char MSSV[10], hoten[30];
    int gioitinh; //1: nữ, 0: nam
    char diachi[50];
    float dtb;
};
typedef struct ttsinhvien SINHVIEN;
```

- Viết các hàm sau:

void Nhap1SV(SINHVIEN &sv); //Nhập thông tin 1 sinh viên

void NhapDSSV(SINHVIEN dssv[], int &n); //Nhập danh sách sinh viên

void Xuat1SV(SINHVIEN sv); //Xuất thông tin 1 sinh viên

void XuatDSSV(SINHVIEN dssv[], int n); //Xuất danh sách sinh viên

int TimSV(SINHVIEN dssv[], int n, char maso[]); //Tìm sinh viên

void XoaSV(SINHVIEN dssv[], int n, char maso[]); //Hàm xóa

void SapTheoDTB(SINHVIEN dssv[], int n); //Sắp xếp theo điểm tb

void SapTheoHoTen(SINHVIEN dssv[], int n); //Sắp xếp theo họ tên

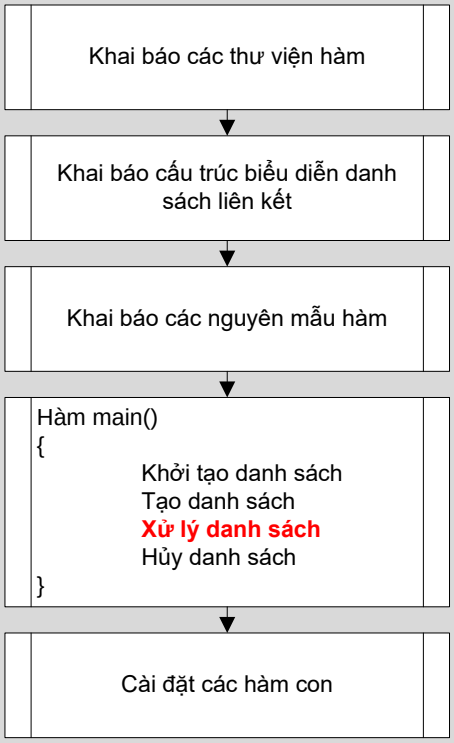
void Hoanvi(SINHVIEN &a, SINHVIEN &b); // Hoán vị 2 sinh viên

Lưu ý: Dùng hàm ***stricmp()*** để so sánh 2 chuỗi

- Hàm chính (main()):
  - Nhập danh sách sinh viên.
  - Xuất danh sách.
  - Nhập mã số sinh viên (x) cần xóa.
  - Xóa x.
  - Xem kết quả sau khi xóa.
  - Sắp xếp theo điểm trung bình, xuất và xem kết quả.
  - Sắp xếp theo họ tên, xuất và xem kết quả.

Phần III: Bài tập danh sách liên kết

Cấu trúc tổng quát của chương trình:



Chương trình mẫu: Nhập và xuất danh sách liên kết đơn các số nguyên

```
#include <iostream.h>
#include <stdlib.h>
//-------------------------------------------
struct ttNODE
{
     int Data;
     struct ttNODE *pNext;
};
typedef struct ttNODE NODE;

//-------------------------------------------

struct ttList
{
     NODE *pHead, *pTail;
};
typedef struct ttList LIST;

//-------------------------------------------

void KhoiTao(LIST &L);
void Huy(LIST &L);
NODE *TaoNode(int x);
void ThemDau(LIST &L, NODE *p);
void Nhap(LIST &L);
void Xuat(LIST L);
```

```
//-------------------------------------------

void main()
{
	LIST L;
	Nhap(L);
	cout<<"\nDanh sach vua nhap: ";
	Xuat(L);

	//Tiếp tục xử lý các yêu cầu

	Huy(L);
}

//-------------------------------------------

void KhoiTao(LIST &L)
{
	L.pHead=L.pTail=NULL;
}

//-------------------------------------------

void Huy(LIST &L)
{
	NODE *p;
	p = new NODE;
	while(L.pHead!=NULL)
	{
		p=L.pHead;
		L.pHead=L.pHead->pNext;
		delete p;
	}
}

//-------------------------------------------

NODE *TaoNode(int x)
{
	NODE *p;
	p=new NODE;
	if(p==NULL)
	{
		cout<<"Khong cap phat duoc vung nho, ket thuc";
		exit(0);
	}
	p->Data=x;
	p->pNext=NULL;
	return p;
}

//-------------------------------------------
```

```
void ThemDau(LIST &L, NODE *p)
{
	if(L.pHead==NULL)
	{
		L.pHead=L.pTail=p;
	}
	else
	{
		p->pNext=L.pHead;
		L.pHead=p;
	}
}

//-----------------------------------------

void Nhap(LIST &L)
{
	int x;
	NODE *p;
	KhoiTao(L);

	do{
		cout<<"Nhap gia tri vao danh sach (Nhap 0 ket thuc): ";
		cin>>x;
		if(x==0)
			break;
		p=TaoNode(x);
		ThemDau(L,p);
	}while(true);
}

//-----------------------------------------

void Xuat(LIST L)
{
	NODE *p=L.pHead;
	while(p!=NULL)
	{
		cout<<p->Data<<" ";
		p=p->pNext;
	}
}

//-----------------------------------------
```

Bài 8: Cho danh sách liên kết đơn gồm các phần tử là số nguyên, viết chương trình thực hiện các yêu cầu sau:

1. Thêm một phần tử vào đầu danh sách.

   void ThemDau(LIST &l, NODE *p);

2. Xuất danh sách ra màn hình.

   void Xuat(LIST l);

3. Liệt kê các phần tử mang giá trị chẵn.

```
void XuatChan(LIST l)
{
        NODE *p=l.pHead;
        while(p!=NULL)
        {
                Nếu p->Data chẵn
                {
                        in giá trị p->Data
                }
                p=p->pNext;
        }
}
```

4. Tính tổng các phần tử mang giá trị chẵn.

```
int TongChan(LIST l)
{
        NODE *p=l.pHead;
        int S=0;
        while(p!=NULL)
        {
                Nếu p->Data chẵn
                        S=S+ p->Data;
                p=p->pNext;
        }
        return S;
}
```

5. Tìm phần tử có giá trị lớn nhất.

```
NODE *TimMax(LIST l)
{
    NODE *max=l.pHead;
    for(NODE *p=l.pHead->pNext; p!=NULL; p=p->pNext)
    {
        Nếu giá trị của max < giá trị của p thì
        {
            gán lại max = p
        }
    }
    return max;
}
```

6. Đếm số lượng số nguyên tố trong danh sách.

   bool LaSNT(int x); //Kiểm tra x có phải là số nguyên tố

   int DemSNT(LIST l);//Đếm số lượng số nguyên tố trong danh sách

7. Xóa phần tử nhỏ nhất trong danh sách (Nếu trùng chỉ xóa phần tử nhỏ nhất đầu tiên).

   NODE *TimMin(LIST l);//Tìm node có giá trị nhỏ nhất

   void XoaDau(LIST &l);//Xóa node đầu của danh sách

   void XoaCuoi(LIST &l);//Xóa node cuối của danh sách

   void Xoap(LIST &l, NODE *p);//Xóa node p

   void XoaMin(LIST &l);//Xóa phần tử nhỏ nhất trong danh sách

8. Nhập vào phần tử X, xóa phần tử đứng sau và đứng trước phần tử X trong danh sách.

   NODE *TimX(LIST l, int X);//Tìm X

   void XoakTruocp(LIST &l, NODE *p, NODE *k);//Xóa k trước p

   void XoakSaup(LIST &l, NODE *p, NODE *q);//Xóa k sau p

9. Tách danh sách thành 2 danh sách, sao cho:
   - Danh sách thứ nhất chứa các phần tử là số nguyên tố.
   - Danh sách thứ hai chứa các phần tử còn lại.

```
void Tach(LIST l, LIST &l1, LIST &l2)
{
    KhoiTao(l1);
    KhoiTao(l2);
    NODE *p=l.pHead, *pAdd;
```

```
        while(p)
        {
                int k = p->Data;
                pAdd=TaoNode(k);
                Nếu k là số nguyên tố thì
                        ThemDau(l1, pAdd);
                Ngược lại
                        ThemDau(l2, pAdd);

                p trỏ đến node kế tiếp
        }
}
```

<u>Bài 9:</u> Cho danh sách liên kết đơn quản lý thông tin của các sinh viên của 1 lớp học. Mỗi sinh viên gồm các thông tin: MSSV, họ và tên, giới tính, địa chỉ và điểm trung bình. Viết chương trình thực hiện các yêu cầu sau:

1. Nhập n sinh viên vào danh sách
2. Thêm 1 sinh viên vào danh sách.
3. In ra danh sách sinh viên.
4. Xóa 1 sinh viên với MSSV cho trước khỏi danh sách.
5. Sắp xếp danh sách sinh viên theo thứ tự tăng dần của điểm trung bình.
6. Liệt kê các sinh viên có điểm trung bình >=5.0
7. Đếm số lượng sinh viên nam.
8. Cập nhật điểm trung bình của một sinh viên thông qua mã số sinh viên.

Bài tập làm thêm: Cài đặt lại câu 1 của phần II dùng danh sách liên kết kép.

Phần IV: Bài tập cây nhị phân tìm kiếm

<u>Bài 10:</u> Khai báo cấu trúc dữ liệu cây nhị phân và viết chương trình thực hiện các yêu cầu sau:

1. Nhập và duyệt cây theo các thứ tự: trước, giữa và sau.
2. Tìm node có giá trị x trên cây.
3. Tìm node có giá trị nhỏ nhất.
4. Tìm node có giá trị lớn nhất.
5. Tính độ cao của cây.
6. Đếm số nút lá của cây.
7. Đếm số nút có đúng 2 cây con.

8. Đếm số nút có đúng 1 cây con.
9. Xóa nút có giá trị x.