

# Thực hành cấu trúc dữ liệu và giải thuật

## Chương 1 (9 tiết): Bài toán tìm kiếm và sắp xếp

### Phần 1: Lý thuyết

#### Các giải thuật tìm kiếm nội

- Tìm kiếm tuyến tính
- Tìm kiếm nhị phân

#### Tìm kiếm nhị phân

Bước 1: left = VTĐ; right = VTC;

Bước 2: Trong khi left  $\leq$  right lặp: //đoạn tìm kiếm chưa rỗng

Bước 21: mid = (left+right)/2; // lấy mốc so sánh

Bước 22: Nếu a[mid] = x: //Tìm thấy.

Dừng, vị trí xuất hiện: mid

Bước 23: Nếu a[mid] > x: //tìm x trong dãy con aleft .. amid -1

right = mid - 1;

Ngược lại //tìm x trong dãy con amid +1 .. aright

left = mid + 1;

//Hết lặp

Bước 3: Dừng, không tìm thấy.

Đánh giá giải thuật

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 n/2$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

#### Các phương pháp sắp xếp thông dụng

- Interchange sort
- Selection sort
- Insertion sort
- Bubble sort

- Quick sort

## Phần 2: Chuẩn bị thực hành

Họ và tên SV: .....

MSSV: .....

Lóp: .....

**Câu 1: Vẽ flowchat thuật toán Selection sort và Insertion sort**

[illegible]

Câu 2: Nêu sự giống nhau và khác nhau giữa thuật toán interchange sort và bubble sort

.....

.....

.....

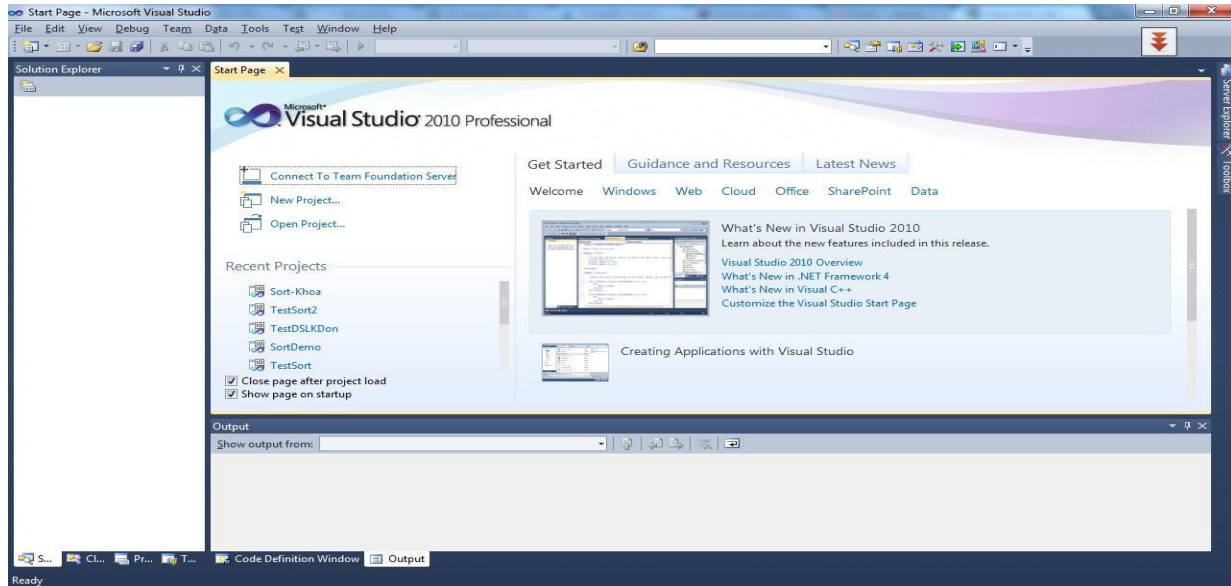
.....

.....

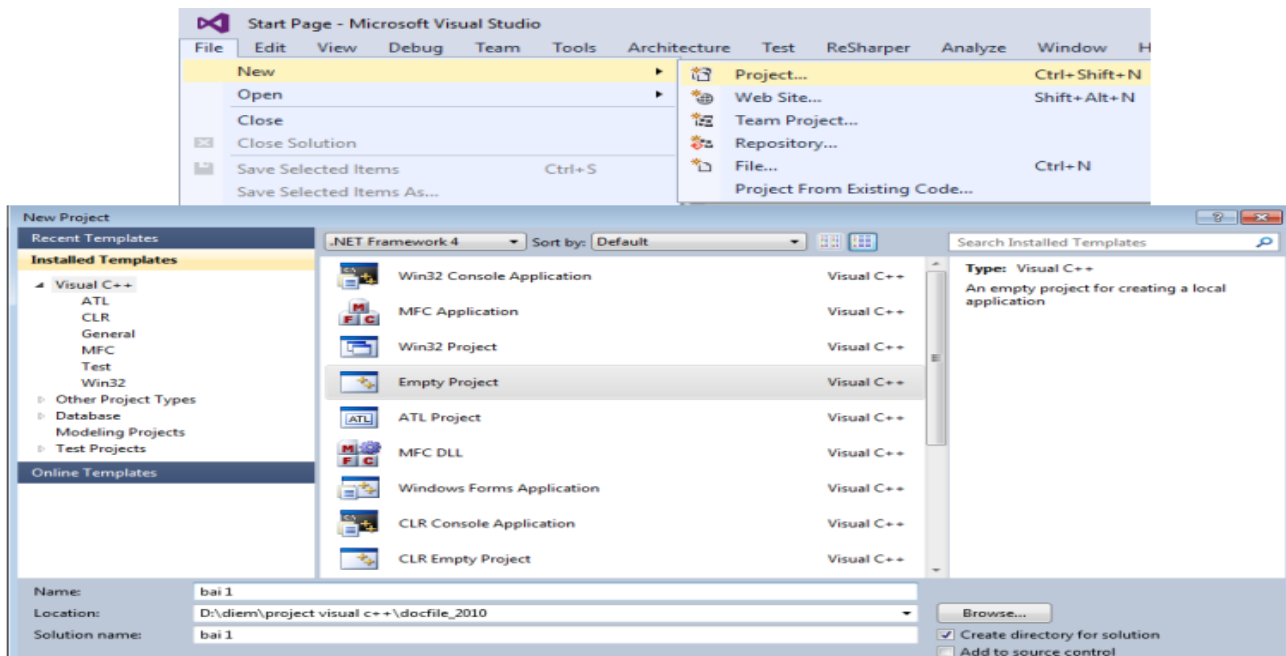
[illegible]

### **Phần 3: Hướng dẫn:**

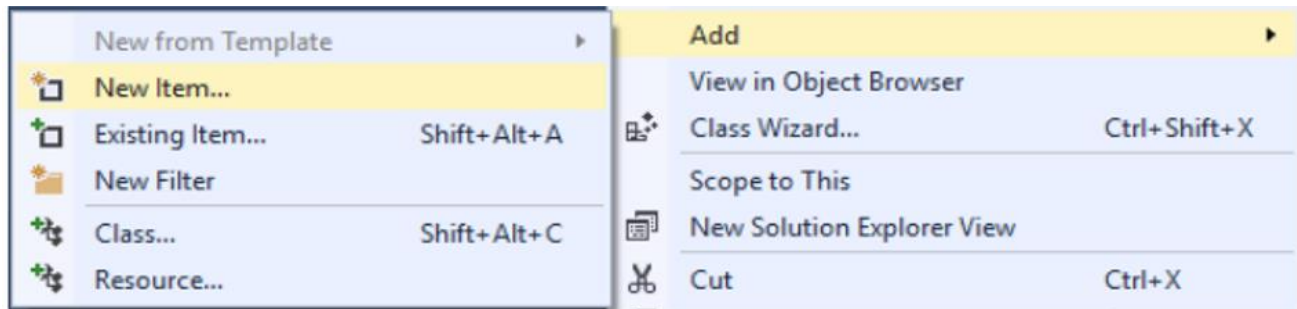
## Mở Visual Studio 2010. Chọn Program File – Microsoft Visual Studio 2010



## Các bước tạo project



Tạo source file bằng cách click phải vào thư mục Source Files trong cửa sổ Solution Explorer (bên phải nhất của giao diện) -> chọn Add->New Item...



Tiếp theo chọn Visual C++ -> chọn C++ file.cpp. Đặt lại tên file hoặc để mặc định (source.cpp). Xong nhấn Add.

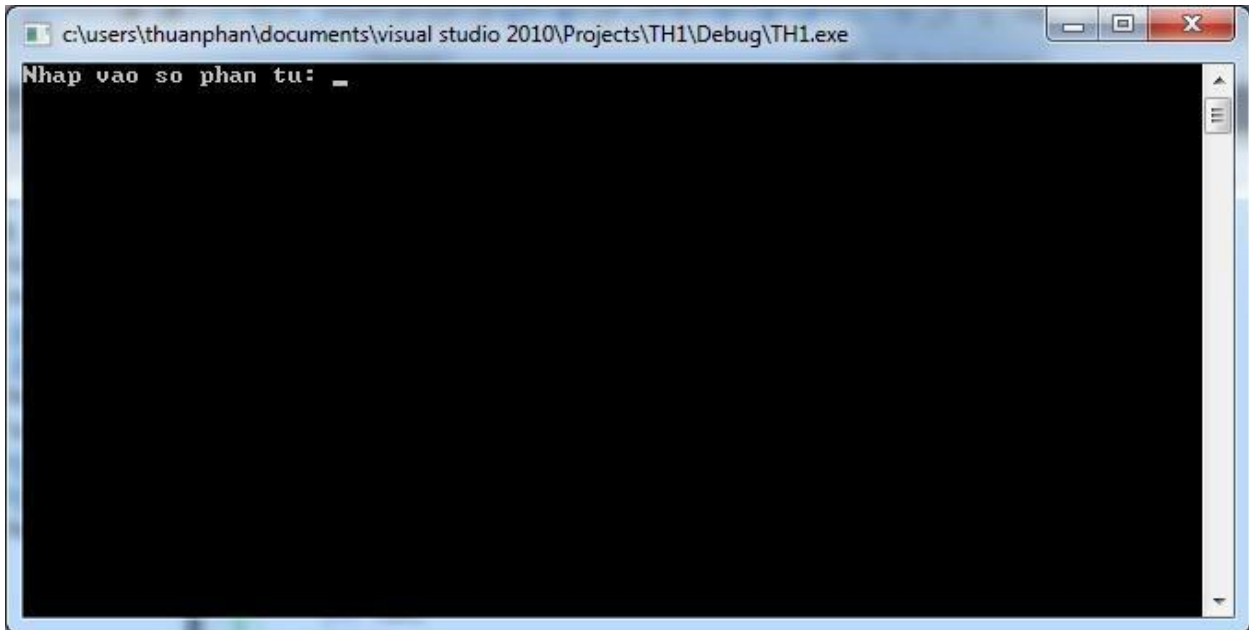


Viết hàm nhập, xuất mảng và sửa lại hàm main

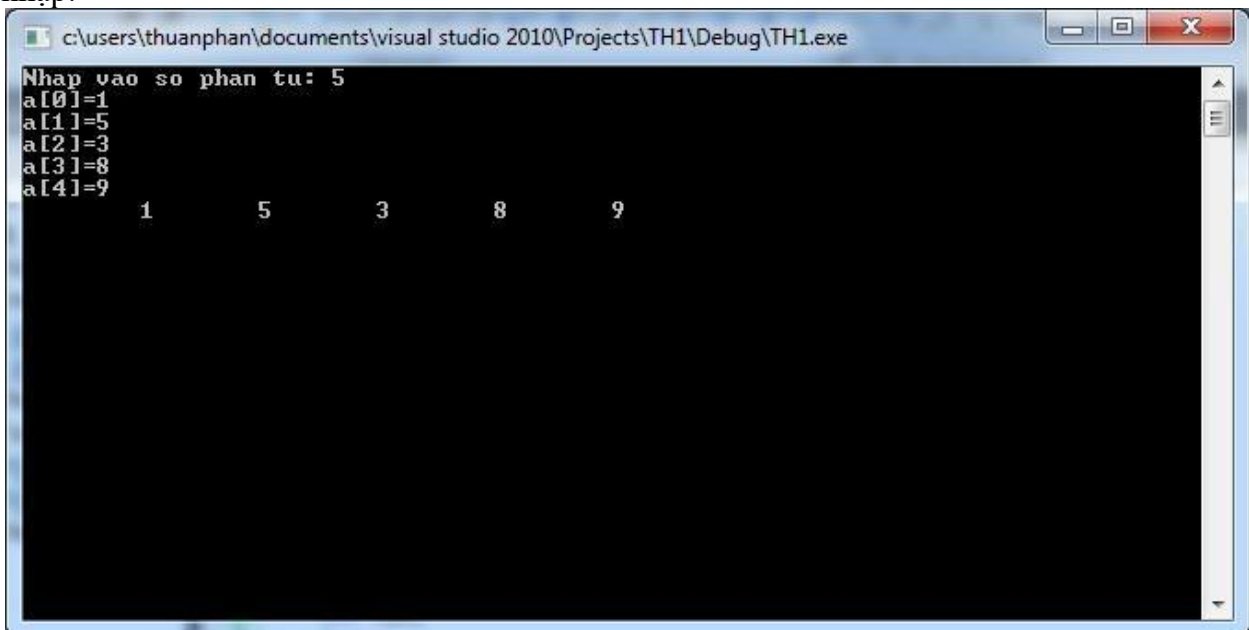
```
#include<stdio.h>
#include<conio.h>
const int max=20;
void nhap(int a[],int n);
void xuat(int a[],int n);
void main()
{
    int a[max],n,m,h[max],c;
    printf("Nhap vao so phan tu: ");
    scanf("%d",&n);
    nhap(a,n);
    xuat(a,n);
    getch();
}
void nhap(int a[],int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }
}
void xuat(int a[],int n)
{
    int i;
    for (i=0;i<n;i++)
        printf("\t%d",a[i]);
    printf("\n");
}
```

Nhấn Ctrl+Shift+B để biên dịch

Nếu chương trình không có lỗi sẽ báo: Build: 1 succeeded, 0 failed. Nhấn F5 để chạy chương trình



Nhập vào số phần tử của mảng. Nhập giá trị từng phần tử. Chương trình in ra mảng đã nhập.



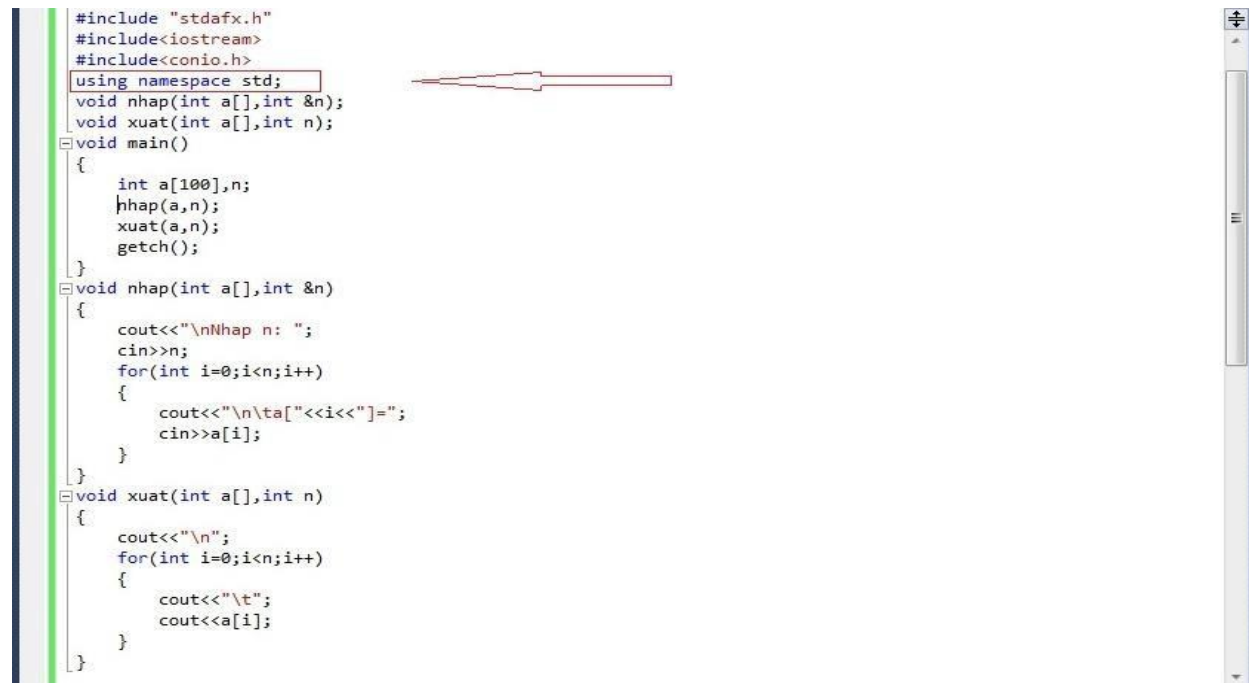
Viết tiếp các hàm:

```
void hoanvi(int &a,int &b); // hoán vị 2 số int a, b

void selectionsort(int a[],int n); // sắp xếp chọn trực tiếp
void insertsort(int a[],int n); // sắp xếp chèn trực tiếp
void interchangesort(int a[],int n); // sắp xếp đổi chỗ trực tiếp
void bubblesort(int a[],int n); // sắp xếp nổi bọt
void quicksort(int a[],int l,int r); // sắp xếp nhanh

// Heap sort
```

Ví dụ chương trình nhập xuất file theo dạng C++



```
#include "stdafx.h"
#include<iostream>
#include<conio.h>
using namespace std;
void nhap(int a[],int &n);
void xuat(int a[],int n);
void main()
{
    int a[100],n;
    nhap(a,n);
    xuat(a,n);
    getch();
}
void nhap(int a[],int &n)
{
    cout<<"\nNhap n: ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cout<<"\n\ta["<<i<<"]=";
        cin>>a[i];
    }
}
void xuat(int a[],int n)
{
    cout<<"\n";
    for(int i=0;i<n;i++)
    {
        cout<<"\t";
        cout<<a[i];
    }
}
```

#### Phần 4: Thực hành

1. Viết chương trình cài đặt thuật toán tìm kiếm tuyến tính và tìm kiếm nhị phân. Input nhập từ 1 file bên ngoài. Output xuất ra 1 file khác.
2. Viết chương trình cài đặt và thử nghiệm thuật toán Sắp xếp nổi bọt (Bubble

sort) cho các số nguyên.

3. Cho một mảng A gồm n số nguyên có giá trị trong khoảng  $[0, n^2 - 1]$ . Viết một thuật toán sắp xếp A có độ phức tạp  $O(n)$ .

4. Viết chương trình cài đặt và thử nghiệm thuật toán Sắp xếp nhanh (Quick Sort) cho các số nguyên. Viết cả dạng sử dụng đệ qui và dạng không sử dụng đệ qui.

5. Viết chương trình cài đặt và thử nghiệm thuật toán Sắp xếp hòa trộn (Merge sort) cho các số nguyên.



**Chương 2: (12 tiết) Danh sách liên kết****Phần 1: Lý thuyết****I. Định nghĩa:**

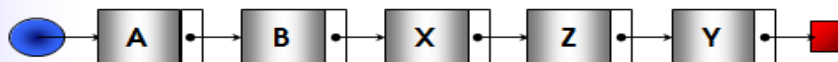
Danh sách là tập hợp gồm một số hữu hạn các phần tử có cùng kiểu dữ liệu. Có 2 cách cài đặt danh sách:

- Cài đặt theo kiểu kế tiếp
- Cài đặt theo kiểu liên kết

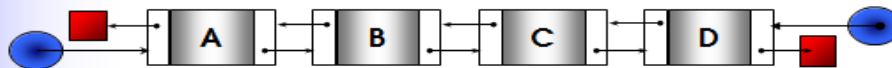
Các dạng danh sách liên kết:

- Danh sách liên kết đơn
- Danh sách liên kết kép
- Danh sách liên kết vòng

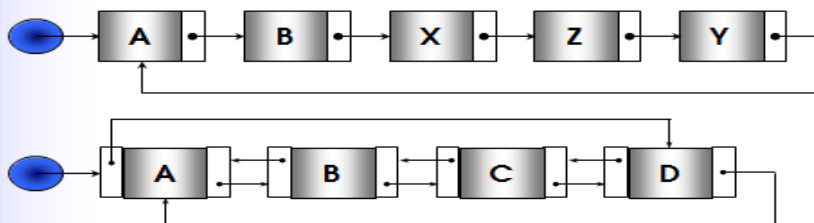
**Danh sách liên kết đơn:** mỗi phần tử liên kết với phần tử đứng sau nó trong danh sách:



**Danh sách liên kết kép:** mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



**Danh sách liên kết vòng :** phần tử cuối danh sách liên kết với phần tử đầu danh sách:

**Các thao tác cơ sở trên danh sách**

- Tạo danh sách rỗng
- Thêm một phần tử vào danh sách
- Tìm kiếm một giá trị trên danh sách

- Trích một phần tử ra khỏi danh sách
- Duyệt danh sách
- Hủy toàn bộ danh sách

**Sắp xếp danh sách****Cách tiếp cận:**

Phương án 1: Hoán vị nội dung các phần tử trong danh sách (thao tác trên vùng data).

Phương án 2 : Thay đổi các mối liên kết (thao tác trên vùng link)

**Phần 2: Chuẩn bị thực hành**

Họ và tên SV: .....

MSSV: .....

Lớp: .....

Câu 1: Nêu ưu nhược điểm của cấu trúc dữ liệu kiểu danh sách động và danh sách kê

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Câu 2: Vẽ flowchat thuật toán bubble sort và selection sort dùng danh sách liên kết đơn



.....

.....

Câu 4: Trình bày thuật toán quick sort không đệ quy bằng danh sách liên kết

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**Phần 3: Hướng dẫn:**

- Cấu trúc dữ liệu của 1 nút
- Cấu trúc dữ liệu của danh sách liên kết đơn

`typedef unsigned int data; // định nghĩa kiểu dữ liệu data là kiểu int`

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#include <string>
#include <conio.h>

typedef unsigned int data;
typedef struct node
{
    data info;
    node* pNext;
};
typedef struct list
{
    node* pHead;
    node* pTail;
};
```

- ❖ Hàm tạo 1 nút có trường info bằng x

```
//tạo nút
node* getnode(data x)
{
    node *p;
    p=new node;
    if(p==NULL)
    {
        cout<<"\nKhong du bo nho";
    }
    p->info=x;
    p->pNext=NULL;
    return p;
}
```

- ❖ Hàm tạo 1 danh sách rỗng

```
void taoList(List &l)
{
    l.phead=NULL;
    l.ptail=NULL;
};
```

- ❖ Hàm thêm 1 phần tử vào đầu danh sách

```
// thêm nút vào danh sách
void AddFirst(List &l, Node *tam)
{
    if(l.pHead==NULL)//xau rong
    {
        l.pHead=tam;
        l.pTail=l.pHead;
    }
    else
    {
        tam->pNext=l.pHead;
        l.pHead->pPre=tam;
        l.pHead=tam;
    }
}
```

- ❖ Hàm thêm 1 phần tử vào cuối danh sách

```
//thêm phần tử vào cuối xâu
void addtail(list &a,node *new_ele)
{
    if(a.pHead==NULL)
    {
        a.pHead=new_ele;
        a.pTail=a.pHead;
    }
    else
    {
        a.pTail->pNext=new_ele;
        a.pTail=new_ele;
    }
}
```

- ❖ Hàm in ra các phần tử trong danh sách

```
//liệt kê các phần tử trong xâu
void viewlist(list a)
{
    node *p;
    while(a.pHead!=NULL)
    {
        p=a.pHead;
        cout<<"\t"<<p->info;
        a.pHead=p->pNext;
    }
}
```

- ❖ Hàm tìm phần tử có khóa x

```
//tìm phần tử khóa x
node* search(list a,data x)
{
    node *p;
    p=a.pHead;
    while((p!=NULL)&&(p->info!=x))
        p=p->pNext;
    return p;
}
```

- ❖ Hàm xóa phần tử đầu danh sách

```
//xóa phần tử đầu xâu
data clear(list &a)
{
    node *p;
    p=a.pHead;
    data x=NULL;
    if(p!=NULL)
    {
        x=p->info;
        a.pHead=a.pHead->pNext;
        delete p;
        if(a.pHead==NULL)
            a.pTail=NULL;
    }
    return x;
}
```

- ❖ Hàm hủy 1 nút sau nút q

```
//hủy 1 nút sau nút Q
void DeleteLastQ(List &l,Node *q)
{
    Node *p;
    if(q!=NULL)
    {
        p=q->pNext;
        if(p!=NULL)
        {
            q->pNext=p->pNext;
            if(p==l.pTail)
                l.pTail=q;
            else
                p->pNext->pPre=q;
            delete p;
        }
    }
    else
        DeleteFirst(l);
}
```

**Phần 4: Thực hành:**

1. Cài đặt thuật toán interchange\_sort bằng danh sách liên kết đơn. Input nhập từ file bên ngoài. Output xuất ra 1 file khác.
2. Cài đặt thuật toán bubble\_sort bằng danh sách liên kết đơn. Input nhập từ file bên ngoài. Output xuất ra 1 file khác.
3. Cài đặt thuật toán insertion sort và binary\_insertion\_sort bằng danh sách liên kết kép. Input nhập từ file bên ngoài. Output xuất ra 1 file khác
4. Cài đặt thuật toán merge sort bằng danh sách liên kết kép. Input nhập từ file bên ngoài. Output xuất ra 1 file khác
5. Ứng dụng danh sách liên kết kép, viết chương trình quản lý sinh viên. Gồm các chức năng:
  - Nhập danh sách sinh viên
  - Sửa thông tin sinh viên
  - Xóa sinh viên
  - Tìm thông tin của 1 sinh viên
  - Tìm những sinh viên có điểm trung bình >5

**Hướng dẫn:**

- Cấu trúc dữ liệu của 1 nút (1 sinh viên)
- Cấu trúc dữ liệu của dssv

```
#include "StdAfx.h"
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include <string.h>

typedef struct  SVNode
{
    int info;
    char Hoten[20];
    char MSSV[7];
    float Diemtoan;
    float Diemly;
    float Diemhoa;
    struct  SVNode  *pPre;
    struct  SVNode  *pNext;
}Node;

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



- ❖ Hàm nhập thông tin của 1 sinh viên

```
//nhap thong tin sinh vien
void nhap(List &l,Node *p)
{
    char Hoten[20], Masv[7];
    float diem, tam;
    printf("\n Nhap ho va ten sinh vien :");
    fflush(stdin);
    gets(Hoten);
    strcpy(p->Hoten,Hoten);
    printf("\n Nhap ma so sinh vien:");
    gets(Masv);
    fflush(stdin);
    strcpy(p->MSSV,Masv);
    printf("\n\tNhap diem toan: ");
    scanf("%f",&tam);
    if(tam>0&&tam<10)
    {
        p->Diemtoan=tam;
    }
    else
    {
        printf("\n Nhap diem sai ,vui long nhap lai:");
        scanf("%f",&tam);
    }
    printf("\n\tNhap diem ly: ");
    scanf("%f",&tam);
    if(tam>0&&tam<10)
    {
        p->Diemly=tam;
    }
    else
    {
        printf("\n Nhap diem sai ,vui long nhap lai:");
        scanf("%f",&tam);
    }
    printf("\n\tNhap diem hoa: ");
    scanf("%f",&tam);
    if(tam>0&&tam<10)
    {
        p->Diemhoa=tam;
    }
    else
    {
        printf("\n Nhap diem sai ,vui long nhap lai:");
        scanf("%f",&tam);
    }
}
```

## ❖ Hàm xóa sinh viên có mã x

```
//xóa sinh viên có mã x
void DeleteX(List &l,int x)
{
    Node *p;
    Node *q;
    q=NULL;
    p=l.pHead;
    while(p!=NULL)
    {
        if(p->info==x)
            break;
        q=p;
        p=p->pNext;
    }
    if(q==NULL)
        printf("\n Không có sinh viên nào như vậy.");
    if(q!=NULL)
        DeleteLastQ(l,q);
    else
        DeleteFirst(l);
}
```

### Chương 3: (6 tiết) Cây nhị phân

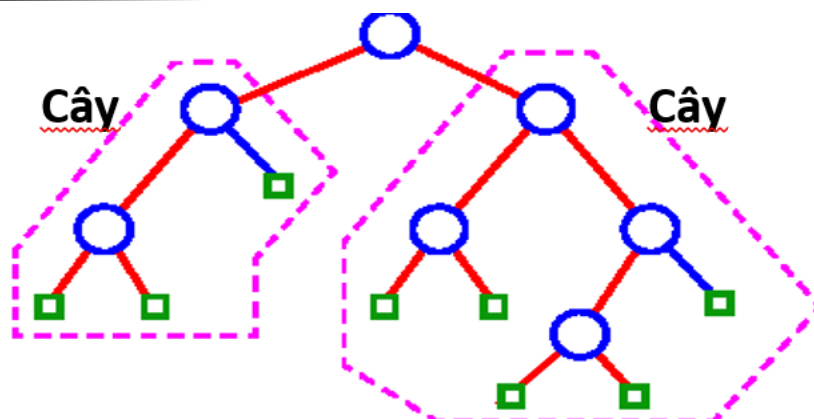
#### Phần 1: Lý thuyết

##### Một số khái niệm cơ bản

- Bậc của một nút : là số cây con của nút đó
- Bậc của một cây : là bậc lớn nhất của các nút trong cây (số cây con tối đa của một nút thuộc cây ). Cây có bậc n thì gọi là cây n-phân.
- Nút gốc : là nút không có nút cha.
- Nút lá : là nút có bậc bằng 0 .
- Nút nhánh : là nút có bậc khác 0 và không phải là gốc .
- Mức của một nút :
  - Mức (gốc (T) ) = 0.
  - Gọi  $T_1, T_2, T_3, \dots, T_n$  là các cây con của  $T_0$
  - $\text{Mức}(T_1) = \text{Mức}(T_2) = \dots = \text{Mức}(T_n) = \text{Mức}(T_0) + 1$ .
- Độ dài đường đi từ gốc đến nút x : là số nhánh cần đi qua kể từ gốc đến x
- Độ dài đường đi tổng của cây : 
$$P_T = \sum_{x \in T} P_x$$
  - trong đó  $P_x$  là độ dài đường đi từ gốc đến X.
- Độ dài đường đi trung bình :  $PI = P_T/n$  (n là số nút trên cây T).
- Rừng cây: là tập hợp nhiều cây trong đó thứ tự các cây là quan trọng.

##### Cây nhị phân

Định nghĩa: Cây nhị phân là cây mà mỗi nút có tối đa 2 cây con



### Duyệt cây nhị phân

Có 3 kiểu duyệt chính có thể áp dụng trên cây nhị phân:

- Duyệt theo thứ tự trước (NLR)- Preorder
- Duyệt theo thứ tự giữa (LNR)- Inorder
- Duyệt theo thứ tự sau (LRN)- Postorder

### Cây nhị phân tìm kiếm (BST)

- Định nghĩa: cây nhị phân tìm kiếm (BST) là cây nhị phân trong đó tại mỗi nút, khóa của nút đang xét lớn hơn khóa của tất cả các nút thuộc cây con trái và nhỏ hơn khóa của tất cả các nút thuộc cây con phải.
- Nếu số nút trên cây là  $N$  thì chi phí tìm kiếm trung bình chỉ khoảng  $\log_2 N$ .

### Một số thao tác trên cây nhị phân

- Tạo cây
- Duyệt cây
- Thêm nút mới vào cây
- Hủy nút đã có

### Phần 2: Chuẩn bị thực hành

Họ và tên SV: .....

MSSV: .....

Lớp: .....

Câu 1: Cho ví dụ một số bài toán dùng cấu trúc cây trong thực tế

.....

.....

.....

Câu 2: Nêu cách chuyển đổi một cây tổng quát thành 1 cây nhị phân

.....

.....

.....

.....

.....

.....

.....

Câu 3: Trình bày code giả tạo cây, duyệt cây theo 3 cách Preorder, Postorder, Inorder

.....

.....

.....

.....

.....

.....

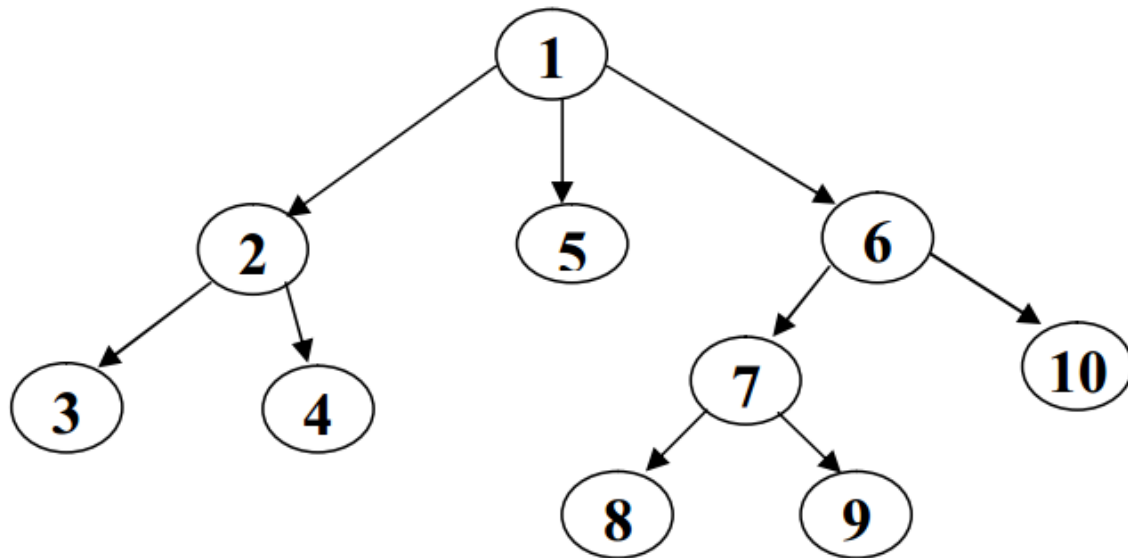
.....

.....

### **Phần 3: Thực hành**

**Câu 1:**

Sử dụng khai báo cây bên dưới trong tất cả các phần của bài tập 1. Chú ý rằng bạn phải viết chương trình test tất cả các phần đã được cài đặt.



Viết phần thân chương trình cho các hàm được khai báo bên dưới:

*//Tạo một cây mới từ một file chứa một dãy các số tương ứng với giá trị tại các nút của cây được đọc theo thứ tự pre-order, nằm trên cùng một dòng, và ngăn cách nhau bởi dấu cách trống*

```
Tree createTree(char *filename);
```

*//In giá trị các nút của cây được duyệt theo thứ tự preorder*

```
void Preorder(Tree t);
```

*// In giá trị các nút của cây được duyệt theo thứ tự postorder.*

```
void Postorder(Tree t);
```

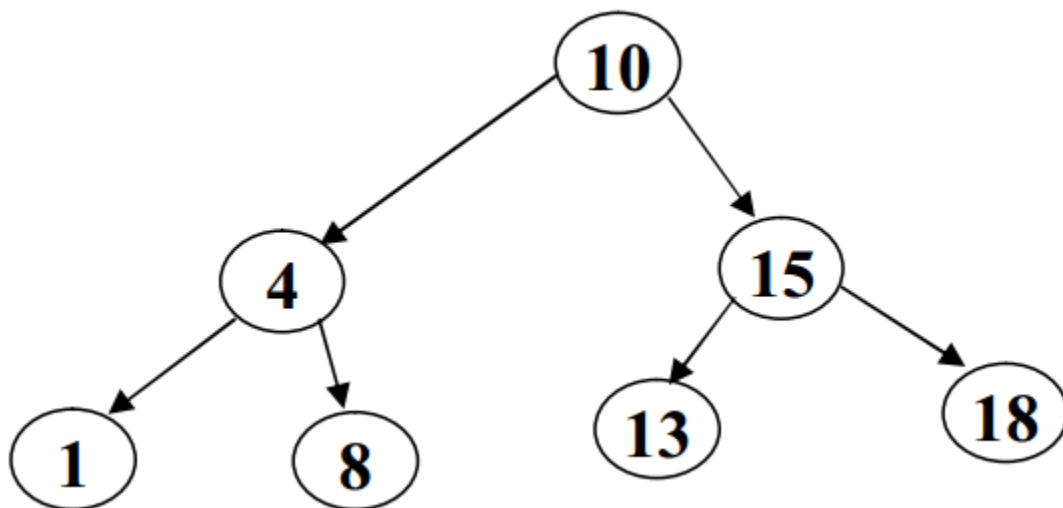
## Bài 2:

Sử dụng khai báo cây nhị phân bên dưới cho tất cả các phần trong bài tập 2. Chú ý rằng bạn phải viết chương trình test tất cả các phần đã được cài đặt

```
struct Node{
```

```
int data;
```

```
Node *left;  
Node *right;  
};  
typedef struct BinaryTreeADT *BinaryTree;  
struct BinaryTreeADT{  
Node *root;  
};
```



Viết phần thân chương trình cho các hàm được khai báo bên dưới:

*// Tạo một cây nhị phân mới từ một file chứa các số nguyên tương ứng với các nút được duyệt theo thứ tự in-order, nằm trên cùng một dòng và ngăn cách nhau bởi dấu cách trống*

```
BinaryTree createTree(char *filename);
```

*//In dữ liệu trong tất cả các nút duyệt theo level-order*

```
void TraversalLevel(BinaryTree t);
```

*//Trả về true nếu cây nhị phân có cây con trái, false trong trường hợp ngược lại*

```
bool hasLeft(BinaryTree t)
```

*//Trả về true nếu cây nhị phân có cây con phải, false trong trường hợp ngược lại*

bool hasRight(BinaryTree t)

*// Trả về cây con trái trong trường hợp cây có cây con trái, NULL trong trường hợp ngược lại.*

BinaryTree Left(BinaryTree t);

*//Trả về cây con phải trong trường hợp cây có cây con phải, NULL trong trường hợp ngược lại.*

BinaryTree Right(BinaryTree t)



**Phân phụ lục:**

Một số bài toán mẫu

## 1. Bài toán tìm kiếm

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

int LinearSearch(int a[], int n, int x);
int LinearSearchCoLinhCanh(int a[], int n, int x);
void main()
{
    int i = 0,j,t;
    int c = 10;
    int a[10];
    clock_t start1,afinish2;
    FILE *fp = fopen ("array.txt","rt");    //Open file;
    int fflush(FILE *fp);                  //Lam sach vung dem;
    while(1)
    {
        fscanf(fp,"%d",&a[i]);    //Doc du lieu tu file vao mang;
        ++i;
        if(!feof(fp))            //Kiem tra xem het tep chua;
            fseek(fp,1,SEEK_CUR);
        //Chuyen con tro vi tri; xuat phat tu vi tri hien tai
        else
            break;
    }

    printf("\nCac phan tu cua mang a la: ");
    for(i= 0;i<10;i++)                //Xuat mang
        printf("%d ",a[i]);
    printf("\n");
```

```
fclose(fp);
//Close file;

printf("\nGia tri x can tim trong mang la: %d\n",c);
    //Tim kiem thong thuong
start1 = clock();
for(int k = 0; k < 10000000; k++)
j = LinearSearch(a,10,c);
finish1 = clock();
    //Tim kiem co linh canh
start2 = clock();
for(int m = 0; m < 10000000; m++)
t = LinearSearchCoLinhCanh(a,10,c);
finish2 = clock();

printf("\nTim kiem thong thuong\n");
printf("Vi tri thu %d \n",j);
printf("Start = %d \n",start1);
printf("Finish = %d \n",finish1);
double time1 = (double)(finish1 - start1)/CLOCKS_PER_SEC;
printf("=> Thoi gian tim kiem %.4f times ", (double)(time1 * 1000));
printf("\n");

printf("\nTim kiem co linh canh\n");
printf("vi tri thu %d \n",t);
printf("Start = %d \n",start2);
printf("Finish = %d \n",finish2);
double time2 = (double)(finish2 - start2)/CLOCKS_PER_SEC;
printf("=> Thoi gian tim kiem %.4f times ", (double)(time2 * 1000));
printf("\n");

if(time1 < time2)
    printf("\n=> Thuat toan tim kiem thong thuong tim nhanh hon thuat
toan tim kiem co linh canh\n");
else
    if(time1 > time2)
```

```
        printf("\n=> Thuật toán tìm kiếm có lĩnh canh tìm nhanh hơn
thuật toán tìm kiếm không có lĩnh canh\n");
    else
        printf("\n=> Thuật toán tìm kiếm thông thường giống thuật
toán tìm kiếm có lĩnh canh\n");
    getch();

}

//Tìm kiếm thông thường
int LinearSearch(int a[], int n, int x)
{
    int i=0;
    while(i<n && a[i]!=x) i++;
    if (i<n) return i;
    return -1;
}

//Tìm kiếm có lĩnh canh
int LinearSearchCoLinhCanh(int a[], int n, int x)
{
    int i=0;
    a[n] = x;
    while(a[i]!=x) i++;
    if (i<n) return i;
    return -1;
}
```

## Bài 2: Bài toán quick sort dùng danh sách liên kết

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <list>
using namespace std;

typedef struct node
{
    int data;
```

```
        node* Next;
    }Node;
typedef struct list
{
    Node* pHead;
    Node* pTail;
}List;
void init(List& l)
{
    l.pHead = l.pTail = NULL;
}
Node* GetNode(int a)
{
    Node* p = new Node;
    if (p == NULL) exit(0);
    p->data = a;
    p->Next = NULL;
    return p;
}
void AddTail(List& l, Node* p)
{
    if (l.pHead == NULL)
        l.pHead = l.pTail = p;
    else
    {
        l.pTail->Next = p;
        l.pTail = p;
    }
}

void docfile(List& l)
{
    Node* p;
    string filename("LIST.txt");
    fstream input_file(filename);
    if (!input_file.is_open())
    {
        cout << "Could not open the file - ";
    }
    while (!input_file.eof())
    {
```

```
        int n;
        input_file >> n;
        p = GetNode(n);
        AddTail(l, p);
    }
    input_file.close();
}

void output(List l)
{
    if (l.pHead != NULL)
    {
        Node* Node = l.pHead;
        while (Node != NULL)
        {
            cout << Node->data << ' ';
            Node = Node->Next;
        }
    }
}

void SListQSort(List& l) {
    Node* X, * p;
    List l1, l2;
    if (l.pHead == l.pTail) return;

    init(l1);    init(l2);
    X = l.pHead;
    l.pHead = X->Next;
    while (l.pHead != NULL) {
        p = l.pHead;
        l.pHead = p->Next;
        p->Next = NULL;
        if (p->data <= X->data) AddTail(l1, p);
        else AddTail(l2, p);
    }
    SListQSort(l1);
    SListQSort(l2);
    if (l1.pHead != NULL)
    {
        l.pHead = l1.pHead;
        l1.pTail->Next = X;
    }
}
```

```
    }
    else
        l.pHead = X;
    if (l2.pHead != NULL)
    {
        X->Next = l2.pHead;
        l.pTail = l2.pTail;
    }
    else
        l.pTail = X;
}

// SListAppend(l, l1);
// AddFirst(l, X);
// SListAppend(l, l2);
//}

int main()
{
    List l;
    init(l);
    input(l);
    cout << "Doc du lieu tu file\n";
    output(l);
    cout << endl;
    SListQSort(l);
    cout << "Merge Sort: \n";
    output(l);

    return 0;
}
```

### Bài 3: Thao tác trên cây nhị phân

```
#include <stdio.h>

#include <conio.h>

struct data //tao struct
```

```
{  
    int ms;  
  
};  
  
struct tagnode  
{  
    data so;  
    tagnode *left, *right;  
};  
  
void init_tree(tagnode* &root) // tao cay rong  
{  
    root = NULL;  
}  
  
tagnode *make_node(int x) // tao 1 nut moi  
{  
    tagnode *p = new tagnode;  
    p->so.ms = x;  
    p->left = p->right = NULL;  
    return p;  
}  
  
tagnode *insert_node(tagnode* &root, int x) // chen pt vao cay  
{  
    tagnode *p = make_node(x);  
    tagnode *q, *f;  
    if(root == NULL)
```

```
{  
    root = p;  
}  
  
else  
  
{  
    q = root;  
    f = NULL;  
    while(q != NULL)  
    {  
        f = q;  
        if(x < q -> so.ms)  
        {  
            q = q -> left;  
        }  
        else  
        {  
            q = q -> right;  
        }  
    }  
    if(x < f -> so.ms)  
    {  
        f -> left = p;  
    }  
    else  
    {
```



```
        f->right = p;
    }
}

return p;
}

tagnode *search_node(tagnode *root , int x)// tìm 1 pt
{
    tagnode *p = root;
    while(p!=NULL)
    {
        if(p->so.ms == x)        return p;
        else if(x < p->so.ms)    p = p->left;
        else                    p = p->right;
    }
    return NULL;
}

int del_node(tagnode* &root , int x)// xoa 1 nut
{
    tagnode *p , *q , *f;
    p = root;
    f = NULL;
    while(p!=NULL)
    {
        if(p->so.ms == x)    break;
        else
```

```
{
    f = p;
    if(x < p->so.ms)    p = p->left;
    else    p = p->right;
}
}
if(p == NULL)    return 0;
else
{
    if(p->left != NULL && p->right != NULL)
    {
        q = p->right;
        f = p;
        while(q->left != NULL)
        {
            f = q;
            q = q->left;
        }
        p->so.ms = q->so.ms;
        p = q;
    }
    if(p->left != NULL)    q = p->left;
    else    q = p->right;
    if(p == root)    root = q;
    else
```

```
    {
        if(p=f->left)    f->left = q;
        else    f->right = q;
    }
    delete p;
    return l;
}
}

void input_tree(tagnode* &root) // tao cay
{
    int n,x;
    root = NULL;
    printf("\nSo phan tu : ");
    scanf("%d" , &n);
    for(int i=0; i<n; i++)
    {
        printf("Phan tu thu %d : ",i);
        scanf("%d" , &x);
        insert_node(root , x);
    }
}

void NLR(tagnode *root) // duyet cay
{
    if(root!=NULL)
    {
```

```
        printf("%d" , root->so);

        NLR(root->left);

        NLR(root->right);

    }

}

void del_tree(tagnode* &root) // xoa cay
{
    if(root!=NULL)
    {
        del_tree(root->left);

        del_tree(root->right);

        delete root;

        root = NULL;
    }
}

void main()// ham chinh
{
    printf("\n tao cay\n");

    tagnode *root,*s;root = NULL;

    input_tree(root);

    while(1)
    {
        printf("\n-----bang chon-----\n");

        printf("\n1 : duyet cay\n");

        printf("\n2 : them 1 pt\n");
```

```
        printf("\n3 : xoa 1 pt\n\n");
        printf("\n4 : xoa cay\n\n");
        int a,x;
        printf("ban chon chuc nang ? :");
        scanf("%d",&a);
        switch(a)
        {
            case 1:    NLR(root);break;
            case 2:    printf("\n\n pt can them la: ");
                      scanf("%d",&x);
                      insert_node(root , x);
                      printf("\n\nsau khi them : ");
                      NLR(root);break;
            case 3:    printf("\n\n pt can xoa la: ");
                      scanf("%d",&x);
                      del_node(root , x);
                      printf("\n\nsau khi xoa : ");
                      NLR(root);break;
            case 4:    del_tree(root);
                      printf("\n\nsau khi xoa : ");
                      NLR(root);break;
        }
        getch();
    }
}
```