

LỜI NÓI ĐẦU

Tài liệu Thực hành cấu trúc dữ liệu và giải thuật trang bị những kiến thức cơ bản nhằm mục tiêu giúp giảng viên giảng dạy thực hành môn học này truyền đạt một cách hiệu quả nhất những kiến thức về cấu trúc dữ liệu và giải thuật đến học viên.

Tài liệu gồm 12 bài học, trong mỗi bài đều có phần tóm tắt lý thuyết, ví dụ demo mẫu, và phần bài tập áp dụng ở lớp cũng như ở nhà. Tài liệu này được biên soạn dùng để giúp giảng viên giảng dạy tốt hơn, giúp sinh viên học tập dễ dàng và hiệu quả hơn.

- Bài 1: Ôn tập ngôn ngữ C
- Bài 2: Các giải thuật tìm kiếm và sắp xếp
- Bài 3: Các giải thuật tìm kiếm và sắp xếp (tt)
- Bài 4: Danh sách liên kết đơn
- Bài 5: Danh sách liên kết đơn (tt)
- Bài 6: Danh sách liên kết đơn (tt)
- Bài 7: Ngăn xếp
- Bài 8: Hàng đợi
- Bài 9: Cây nhị phân - cây nhị phân tìm kiếm
- Bài 10: Cây nhị phân - cây nhị phân tìm kiếm (tt)
- Bài 11: Cây nhị phân - cây nhị phân tìm kiếm (tt)
- Bài 12: Ôn tập

Trong quá trình biên soạn chắc chắn không tránh khỏi những sai sót, tác giả rất mong nhận được sự đóng góp ý kiến của sinh viên và đồng nghiệp. Mọi ý kiến đóng góp tác giả xin ghi nhận và sẽ chỉnh sửa lại để tài liệu này ngày càng hoàn thiện hơn.

Tác giả

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 1: CÁC GIẢI THUẬT TÌM KIẾM VÀ SẮP XẾP	
--	--	---

A. MỤC TIÊU:

- Vận dụng các giải thuật tìm kiếm và sắp xếp dữ liệu vào từng bài toán cụ thể.
- Lập trình được các giải thuật tìm kiếm và sắp xếp thành các chương trình cụ thể chạy được trên máy tính.
- Làm được các bài tập ở cuối mỗi chương.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH

1.1. Giải thuật tìm kiếm tuyến tính

1.1.1. Giải thuật:

B1: $i = 0$; // *Bắt đầu từ phần tử đầu tiên*

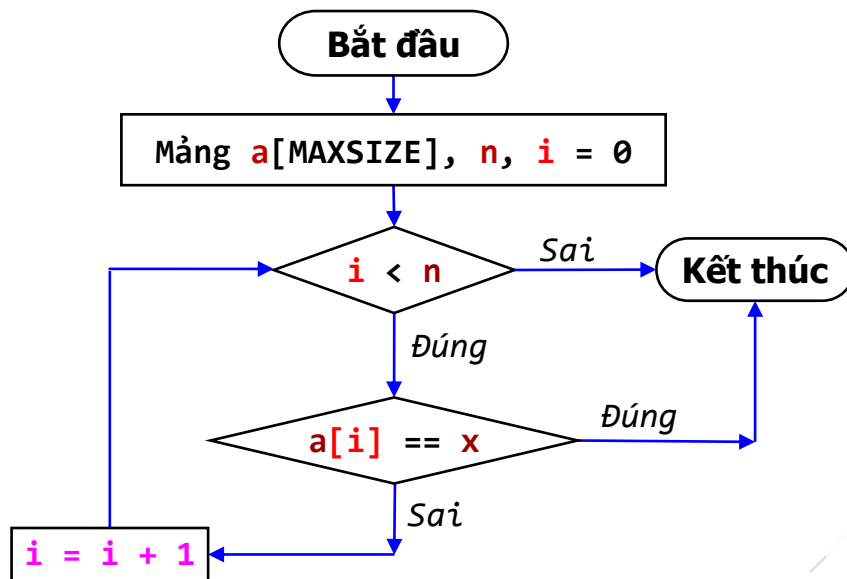
B2: so sánh $A[i]$ với X , có 2 khả năng:

- Nếu $A[i] = X \rightarrow$ Tìm thấy. Dừng
- Nếu $A[i] \neq X \rightarrow$ Sang B3

B3: Tăng biến lặp $i: i = i + 1$; // *Xét phần tử tiếp theo trong mảng*

- Nếu $i > n$: Hết mảng, không tìm thấy. Dừng
- Ngược lại: lặp lại B2

1.1.2. Lưu đồ giải thuật:



1.1.3. Cài đặt giải thuật:

```
int LinearSearch_While(ItemType a[], int n, ItemType x)
{
    int i = 0;
    while( (i < n) && (a[i] != x) )
        i++;
    if(i < n)
        return i; //Tìm thấy
    else
        return -1; //Không tìm thấy x
}
```

Hoặc viết bằng vòng lặp **for**:

```
int LinearSearch_For(ItemType a[], int n, ItemType x)
{
    for(int i = 0; i < n; i++)
        if(a[i] == x)
            return i; //Tìm thấy tại vị trí i
    return -1; //Không tìm thấy
}
```

Hoặc viết bằng cách sử dụng kỹ thuật “lính canh”:

```
int LinearSearch_LinhCanh(ItemType a[], int n, ItemType x)
{
    int i = 0;
    a[n] = x;
    while(a[i] != x)
        i++;
    if(i < n)
```

```

    return i; //Tìm thấy x tại vị trí i
else
    return -1; //Không tìm thấy x
}

```

1.2. Giải thuật tìm kiếm nhị phân

1.2.1. Giải thuật:

B1: Left = 0; Right = n - 1;

B2: Mid = (Left + Right)/2;

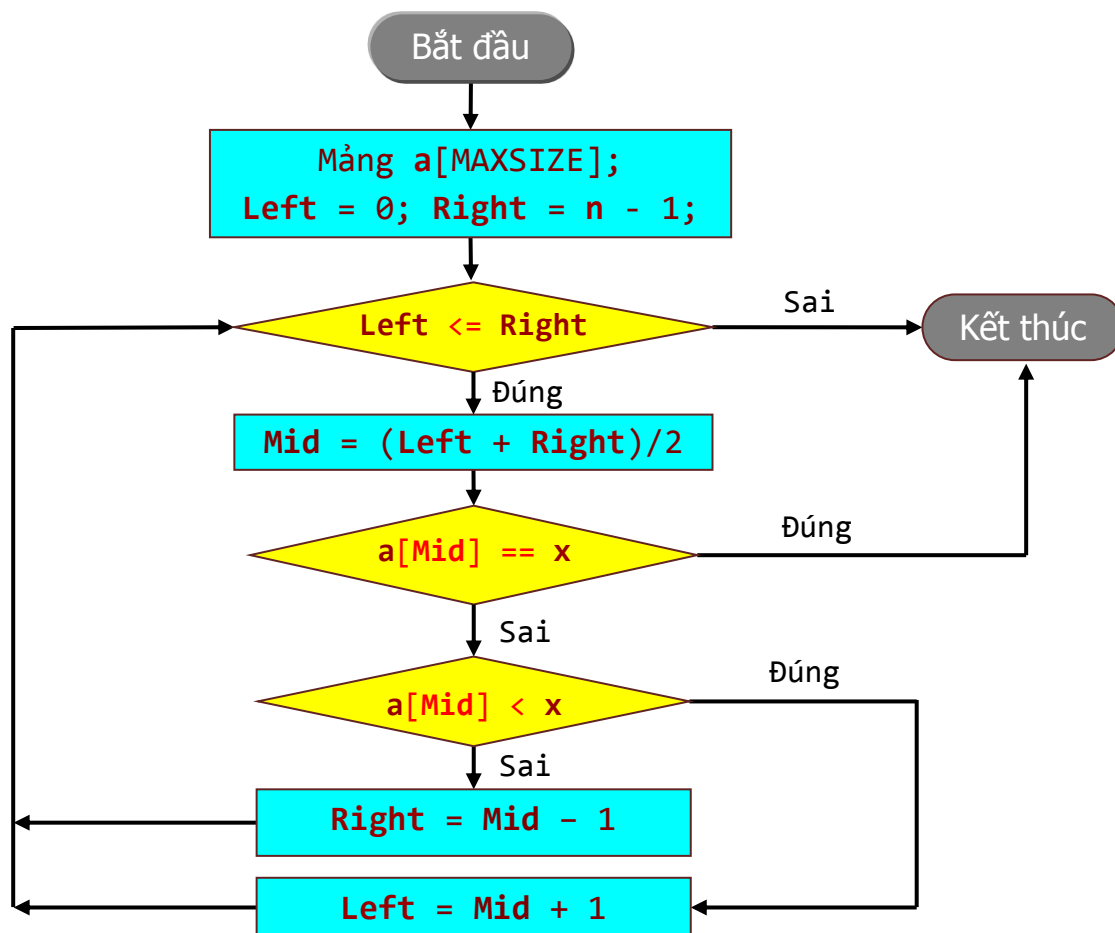
B3: so sánh A[Mid] với X có 3 khả năng xảy ra:

- Nếu A[Mid] = X; //Tìm thấy. Dừng
- Nếu A[Mid] > X; //Tiếp tục tìm trong dãy A[Left] ... A[Mid-1]
Right = Mid - 1;
- Nếu A[Mid] < X; //tìm trong dãy A[Mid + 1] ... A[Right]
Left = Mid + 1;

B4: Nếu Left < Right: Lặp lại B2

Ngược lại: Kết thúc

1.2.2. Lưu đồ giải thuật:



1.2.3. Cài đặt giải thuật (không đệ quy):

```
int BinarySearch(ItemType a[], int n, ItemType x)
{
    int Left = 0, Right = n - 1;
    while(Left <= Right)
    {
        int Mid = (Left + Right) / 2;
        if(x == a[Mid])
            return Mid; //Tìm thấy tại vị trí Mid
        else if(x < a[Mid])
            Right = Mid - 1; //Giảm biên phải xuống đến vị trí Mid-1
        else
            Left = Mid + 1; //Tăng biên trái lên đến vị trí Mid+1
    }
    return -1; //Không tìm thấy x
}
```

1.2.4. Cài đặt giải thuật (đệ quy):

```
int BinarySearch_Recursive(ItemType a[], int Left, int Right,
ItemType x)
{
    if(Left > Right) return (-1);
    int Mid = (Left + Right) / 2;
    if(x == a[Mid]) return Mid;
    if(x < a[Mid]) return BinarySearch_Recursive(a, Left, Mid-1, x);
    else return BinarySearch_Recursive(a, Mid+1, Right, x);
};
```

1.3. Giải thuật sắp xếp đổi chỗ trực tiếp – Interchange Sort

1.3.1. Ý tưởng giải thuật:

Xuất phát từ đầu dãy, tìm tất cả nghịch thế chứa phần tử này, đổi chỗ phần tử này với các phần tử trong cặp nghịch thế. Lặp lại xử lý trên với các phần tử tiếp theo trong dãy.

1.3.2. Cài đặt giải thuật:

```
void InterchangeSort_Ascending(ItemType a[], int n)
{
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(a[i] > a[j])
                Swap(a[i], a[j]); //Gọi hàm hoán vị 2 phần tử.
}
```

Trong đó hàm Swap dùng để hoán vị giá trị của 2 phần tử, được định nghĩa như sau (**Lưu ý**: Hàm này phải viết trước các hàm sắp xếp):

```
void Swap(ItemType &x, ItemType &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

1.4. Giải thuật sắp xếp nổi bọt – Bubble Sort.

1.4.1. Ý tưởng giải thuật:

Nói một cách dễ hiểu là tìm cách đưa những phần tử nhỏ từ cuối dãy về phần đầu dãy, và ngược lại đưa những phần tử lớn từ đầu dãy về phần cuối dãy.

Ý tưởng thực hiện: Xuất phát từ cuối dãy, lần lượt tìm từng cặp phần tử tạo nghịch thế và đổi chỗ chúng với nhau, cho đến khi đưa được phần tử nhỏ nhất trong dãy về vị trí đứng đầu dãy hiện hành, sau đó không xét tới phần tử đó ở bước tiếp theo nữa. Do vậy ở lần lặp thứ i có vị trí đầu dãy sẽ là i .

1.4.2. Cài đặt giải thuật:

```
void BubbleSort_Ascending(ItemType a[], int n)
{
    for(int i = 0; i < n - 1; i++)
        for(int j = n - 1; j > i; j--) //tìm phần tử min kể từ vị trí
                                      $i+1, \dots, n-1$  đưa về vị trí  $i$ 
            if(a[j] < a[j - 1])
                Swap(a[j], a[j - 1]);
}
```

1.5. Bài tập ở lớp.

1. Cho mảng một chiều các số nguyên, viết chương trình tìm một phần tử x trong mảng (nếu có thì trả về vị trí đầu tiên x xuất hiện trong mảng, nếu không thì trả về -1) bằng 2 cách: không sử dụng phần tử cảm canh và sử dụng phần tử cảm canh.
2. Tìm kiếm 1 giá trị x trên mảng đã sắp xếp.
3. Viết chương trình thống kê số phép gán và số phép so sánh trong bài 1 và bài 2.
4. Khai báo CTDL cho một sinh viên (SV) gồm các thông tin sau:

MaSV : số nguyên dương 4 bytes
 HoLot : chuỗi tối đa 25 kí tự
 Ten : chuỗi tối đa 8 kí tự

NamSinh : số nguyên dương 2 bytes

Cài đặt các hàm sau:

- Nhập vào mảng một chiều **a** gồm **n** phần tử, mỗi phần tử lưu trữ thông tin một sinh viên (với $0 < n < 100$).
- Tìm sinh viên có MaSV = **x** trong mảng. Nếu có thì cho biết *thông tin của sinh viên* này, ngược lại thông báo “*Không tìm thấy sinh viên có mã số là x*”.
- Tìm sinh viên có NamSinh = **m** trong mảng. Nếu có thì cho biết *thông tin của sinh viên* này, ngược lại thông báo “*Không tìm thấy sinh viên có năm sinh là m*”.
- Tìm sinh viên có tên là **y** (nếu có nhiều SV có tên là **y** thì chỉ cần cho biết thông tin của SV đầu tiên).
- Giả sử mảng đã sắp xếp tăng dần theo MaSV, Hãy tìm sinh viên có MaSV = **x**.
- Giả sử mảng đã sắp xếp tăng dần theo Ten, Hãy tìm sinh viên có Ten = **y**.
- Sắp xếp mảng tăng dần theo MaSV (bằng các giải thuật: **Interchange Sort, Bubble Sort**).
- Sắp xếp mảng giảm dần theo MaSV (bằng các giải thuật: **Interchange Sort, Bubble Sort**).
- Sắp xếp mảng tăng dần theo Ten (bằng các giải thuật: **Interchange Sort, Bubble Sort**).
- Sắp xếp mảng giảm dần theo Ten (bằng các giải thuật: **Interchange Sort, Bubble Sort**).
- Sắp xếp mảng tăng dần theo Ten, nếu trùng tên thì sắp xếp tăng dần theo HoLot (bằng các giải thuật: **Interchange Sort, Bubble Sort**).
- In ra màn hình danh sách những sinh viên có năm sinh = 1999.

1.6. Bài tập về nhà.

5. Khai báo CTDL cho một trường học (TRUONGHOC) gồm các thông tin:

MaT (Mã trường) : số nguyên dương 4 bytes

TenT (Tên trường) : chuỗi tối đa 50 kí tự

NTL (Năm thành lập) : số nguyên dương 2 bytes

Cài đặt các hàm sau:

- Nhập vào mảng một chiều **a** gồm **n** phần tử, mỗi phần tử là thông tin một trường học (với $0 < n < 100$).
- Tìm trường học có MaT = **x** trong mảng.
- Tìm trường học có TenT = **y** trong mảng.
- Sắp xếp mảng tăng theo MaT (bằng giải thuật: **Interchange Sort**).
- Sắp xếp mảng tăng theo TenT (bằng giải thuật: **Bubble Sort**).
- Tìm trường học có MaT = **x** (sau khi mảng đã sắp tăng theo MaS).

- g. Tìm trường học có TenT = y (nếu có nhiều trường học trùng tên thì chỉ cần tìm trường học đầu tiên).
- h. Liệt kê ra màn hình danh sách những trường học có năm thành lập từ 2000 đến nay.



Lưu ý: Trong quá trình thực hiện lại các Bài tập mẫu theo hướng dẫn, sinh viên không được gõ theo nội dung của bài hướng dẫn từ a → z, *mà phải làm từng chức năng một và chạy thử nghiệm để xem và kiểm tra kết quả cho đến khi thật sự hiểu được bài rồi mới làm tiếp chức năng khác.* Hãy lặp lại việc đó cho các chức năng còn lại

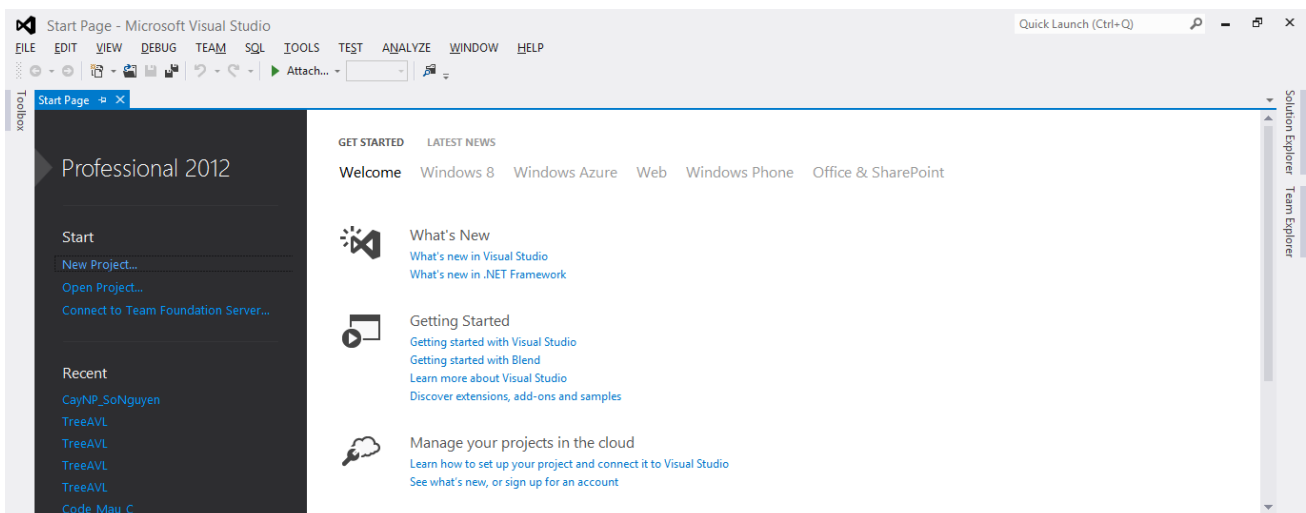


Bài tập mẫu hướng dẫn thực hành

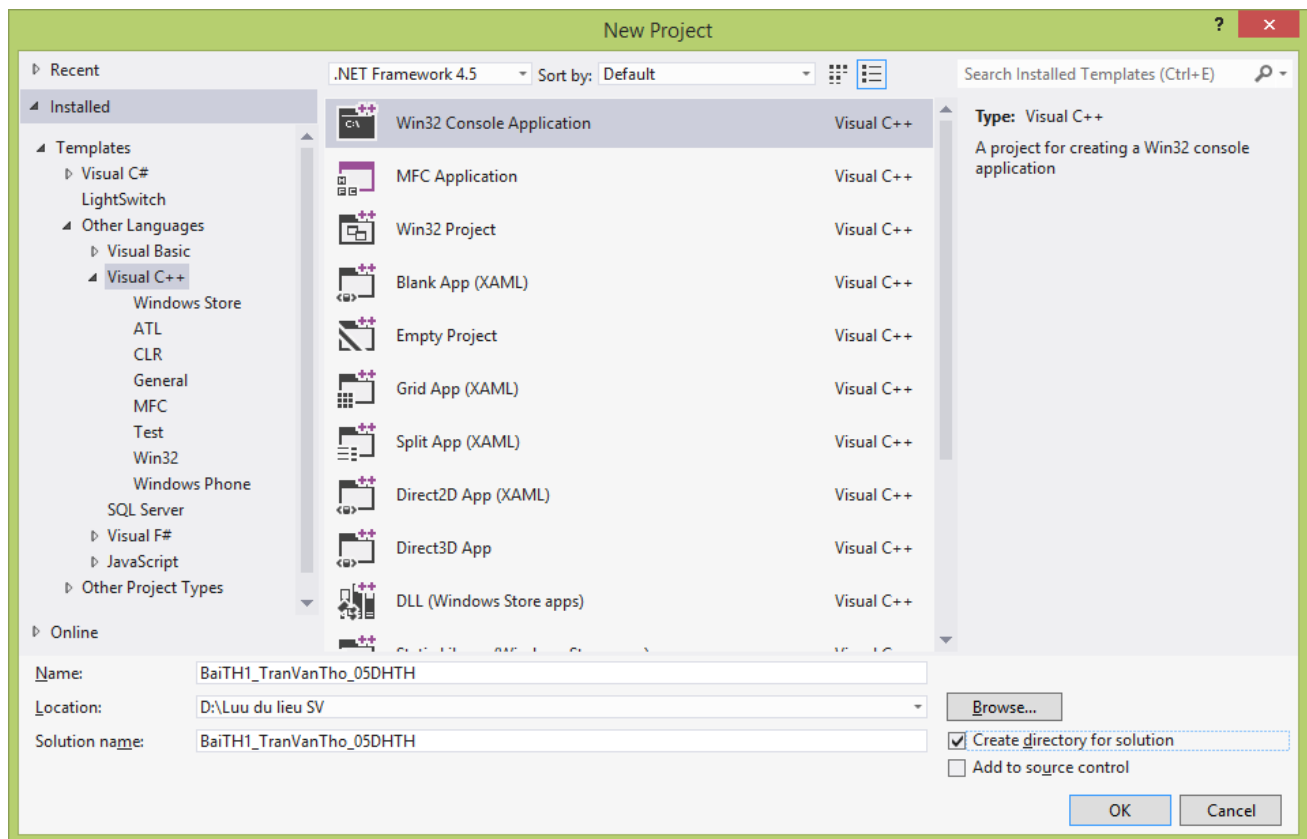
Bài ví dụ 1: Nhập 1 mảng số nguyên, tìm kiếm trên mảng có hay không một giá trị **x**?

Bước 1: Tạo một **Project** mới.

Start → Programs → Microsoft Visual Studio 2012 → xuất hiện màn hình sau:



Chọn **New Project...** → Chọn ngôn ngữ lập trình **Visual C++** → Chọn **Win32 Console Application**

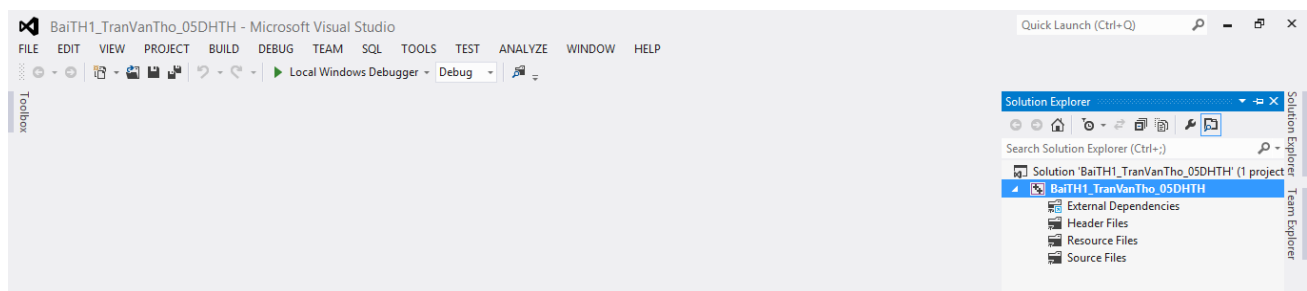


Name: Đặt tên cho Project (ví dụ: BaiTH1_TranVanTho_05DHTH).

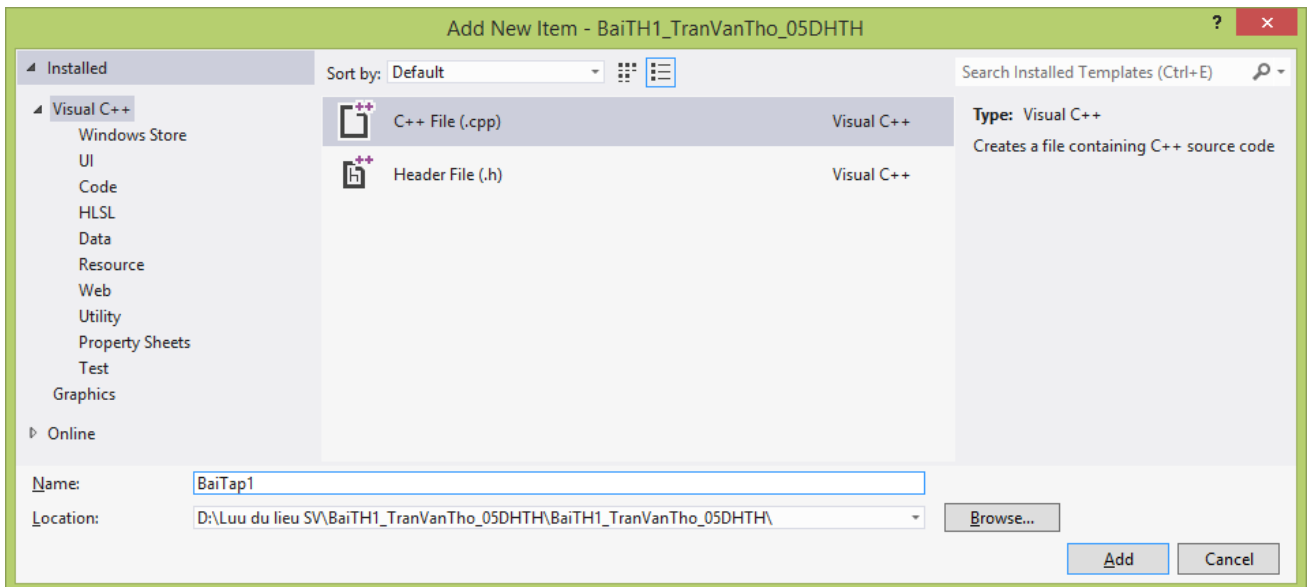
Location: Chọn đường dẫn đến thư mục sẽ chứa project (ví dụ: D:\LuuDuLieuSinhVien).

Solution Name: Thường mặc định giống mục Name.

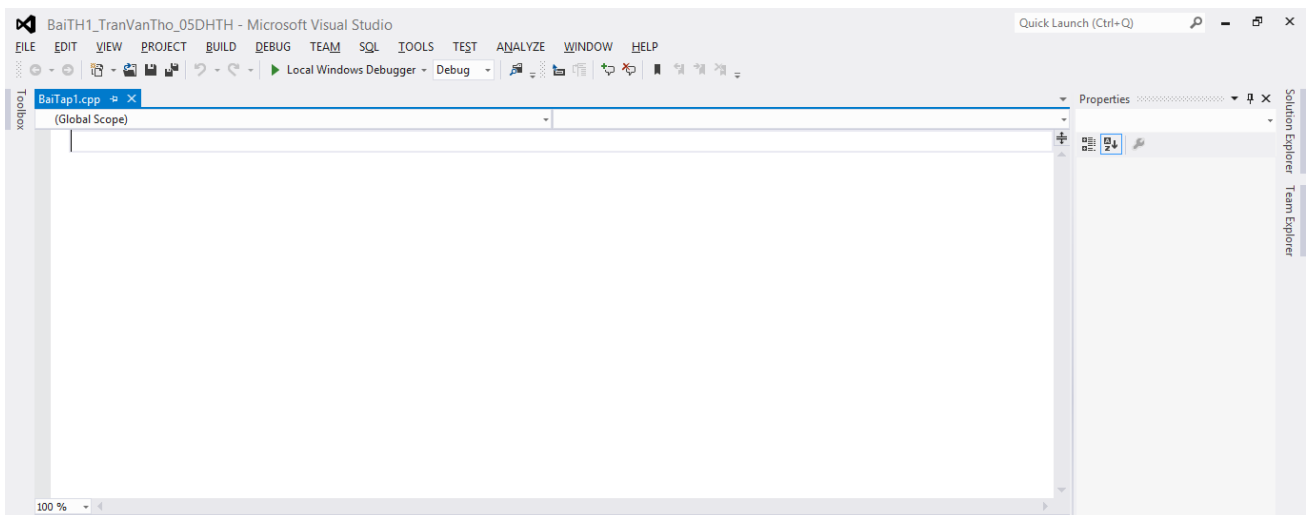
Tiếp tục nhấn OK → Next → Chọn **Empty Project** → Finish → Xuất hiện màn hình sau:



Tiếp tục Right click lên Source Files → Add → New Item... → Chọn **C++ File (.cpp)** → Đặt tên file, ví dụ: BaiTap1 → Nhấn nút **Add**.



Cửa sổ lập trình sẽ như sau:



Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

Bước 3: Khai báo hằng số cho biết số phần tử tối đa của mảng.

```
#define MAXSIZE 100
typedef int ItemType; //Định nghĩa kiểu dữ liệu của một phần tử
```

Bước 4: Viết các hàm nhập, xuất, tìm kiếm như sau:

```
void NhapMangNguyen(ItemType a[], int &n)
{
    do
    {
        printf("Cho biet so phan tu cua mang: ");
        scanf("%d", &n);
    }while(n <= 0);
    for(int i = 0; i < n; i++)
    {
        printf("Nhap phan tu a[%d] = ", i);
```

```
        scanf("%d", &a[i]);
    }
}
//=====
void XuatMangNguyen(ItemType a[], int n)
{
    printf("\nNoi dung cua mang la: ");
    for(int i = 0; i < n; i++)
        printf("%4d", a[i]);
}
//=====
int TimTuyenTinh(ItemType a[], int n, ItemType x)
{
    //Tim gia tri x co trong mang hay khong?
    //Hàm tra ve -1 neu khong tim thay, nguoc lai tra ve vi tri thu i
    for(int i = 0; i < n; i++)
        if(a[i] == x)
            return i;
    return -1;
}
```

Bước 5: Viết hàm main để thực thi chương trình.

```
void main()
{
    ItemType A[MAXSIZE], X;
    int N, kq;
    NhapMangNguyen(A, N);
    XuatMangNguyen(A, N);
    printf("\nCho biet gia tri muon tim kiem: ");
    scanf("%d", &X);
    kq = TimTuyenTinh(A, N, X);
    if(kq == -1)
        printf("\nKhong tim thay gia tri %d trong mang.", X);
    else
        printf("\nTim thay gia tri %d trong mang tai vi tri %d", X, kq);
    getch();
}
```

→ Kết quả chạy thử nghiệm.

```
Cho biet so phan tu cua mang: 5
Nhap phan tu a[0]=7
Nhap phan tu a[1]=-3
Nhap phan tu a[2]=8
Nhap phan tu a[3]=15
Nhap phan tu a[4]=9
Noi dung cua mang la:    7  -3  8  15  9
Cho biet gia tri muon tim kiem: 15

Tim thay gia tri 15 trong mang tai vi tri 3

Cho biet so phan tu cua mang: 5
Nhap phan tu a[0]=6
Nhap phan tu a[1]=12
Nhap phan tu a[2]=-7
Nhap phan tu a[3]=4
Nhap phan tu a[4]=-5
Noi dung cua mang la:    6  12  -7  4  -5
Cho biet gia tri muon tim kiem: 3

Khong tim thay gia tri 3 trong mang.
```

Lưu ý: Có thể viết hàm tạo một mảng ngẫu nhiên các số nguyên như sau:

```
void TaoMangNguyen(ItemType a[], int &n)
{
    do
```

```
{
    printf("Cho biet so phan tu cua mang: ");
    scanf("%d", &n);
}while(n <= 0);
srand((unsigned)time(NULL));
for(int i = 0; i < n; i++)
    a[i] = (rand()%199)-99; //Tạo 1 số ngẫu nhiên trong đoạn [-99, 99]
}
```

→ phải khai báo thêm 2 thư viện sau:

```
#include <stdlib.h>
#include <time.h>
```

→ Sau đó, trong hàm *main* ta thay hàm **NhapMangNguyen** bởi hàm **TaoMangNguyen**.

Bài ví dụ 2: Nhập 1 mảng chứa thông tin các sinh viên (gồm: MaSV – chuỗi 10 ký tự, HoSV – chuỗi 25 ký tự, TenSV – chuỗi 7 ký tự, DiemTB – số thực)

- Cho biết sinh viên có DiemTB lớn nhất?
- Tìm kiếm và cho biết có sinh viên nào có tên y hay không (y nhập từ bàn phím)?

Bước 1: Tạo một **Project** mới → thực hiện giống như Bài 1.

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

Bước 3: Khai báo hằng số cho biết số phần tử tối đa của mảng.

```
#define MAXSIZE 100
```

Bước 4: Khai báo kiểu dữ liệu cấu trúc SINHVIEN.

```
typedef char KeyType;
struct SINHVIEN
{ //Định nghĩa kiểu dữ liệu để Lưu thông tin cho 1 sinh viên
    KeyType MaSV[11];
    char HoSV[26];
    char TenSV[8];
    float DiemTB;
};
typedef SINHVIEN ItemType;
```

Bước 5: Viết các hàm nhập, xuất, tìm kiếm như sau:

```
void NhapThongTinSV(ItemType &x)
{
    printf("\nNhap ma so sinh vien: ");
    fflush();
    gets(x.MaSV);
    printf("Nhap ho sinh vien: ");
    fflush();
    gets(x.HoSV);
    printf("Nhap ten sinh vien: ");
    fflush();
    gets(x.TenSV);
    printf("Nhap diem trung binh sinh vien: ");
    scanf("%f", &x.DiemTB);
}
//=====
```

```
void XuatThongTinSV(ItemType x)
{
    printf("%-15s%-20s%-10s%5.1f", x.MaSV, x.HoSV, x.TenSV, x.DiemTB);
}
//=====
void NhapDanhSachSV(ItemType a[], int &n)
{
    do
    {
        printf("Cho biet so luong sinh vien: ");
        scanf("%d", &n);
    }while(n <= 0);
    for(int i = 0; i < n; i++)
        NhapThongTinSV(a[i]);
}
//=====
void XuatDanhSachSV(ItemType a[], int n)
{
    printf("\nDANH SACH SINH VIEN: ");
    printf("\n%-5s%-15s%-30s%-10s", "STT", "MA SO SV", "HO VA TEN SINH VIEN", "DIEM TB");
    for(int i = 0; i < n; i++)
    {
        printf("\n%-5d", i + 1);
        XuatThongTinSV(a[i]);
    }
}
//=====
void XuatThongTinSV_DiemTB_Max(ItemType a[], int n)
{
    int vtMaxDTB = 0;
    for(int i = 1; i < n; i++)
        if(a[i].DiemTB > a[vtMaxDTB].DiemTB)
            vtMaxDTB = i;
    printf("\nThong tin sinh vien co DiemTB lon nhat\n");
    XuatThongTinSV(a[vtMaxDTB]);
}
//=====
int Tim_TheoTenSV(ItemType a[], int n, char ten[])
{
    for(int i = 0; i < n; i++)
        if(strcmp(a[i].TenSV, ten) == 0)
            return i;
    return -1;
}
```

Bước 6: Viết hàm main để thực thi chương trình.

```
void main()
{
    ItemType A[MAXSIZE];
    int N, kq;
    KeyType X[11];
    NhapDanhSachSV(A, N);
    XuatDanhSachSV(A, N);
    XuatThongTinSV_DiemTB_Max(A, N);
    printf("\nCho biet ten sinh vien muon tim kiem: ");
    fflush();
    gets(X);
    kq = Tim_TheoTenSV(A, N, X);
}
```

```

if(kq == -1)
    printf("\nKhông tìm thấy sinh viên nào có tên %s.", X);
else
{
    printf("\nThông tin sinh viên muốn tìm là: \n");
    XuatThongTinSV(A[kq]);
}
getch();
}

```

→ Kết quả chạy thử nghiệm.

```

Cho biet so luong sinh vien: 3
Nhap ma so sinh vien: 1122334455
Nhap ho sinh vien: Trinh Thanh
Nhap ten sinh vien: Deo
Nhap diem trung binh sinh vien: 7.3928

Nhap ma so sinh vien: 5566447733
Nhap ho sinh vien: Tran Bich
Nhap ten sinh vien: Huu
Nhap diem trung binh sinh vien: 6.7234

Nhap ma so sinh vien: 6611442277
Nhap ho sinh vien: Vu Thi
Nhap ten sinh vien: Tuyet
Nhap diem trung binh sinh vien: 7.8321

DANH SACH SINH VIEN:
STT      Ma so SV      Ho va ten      DiemTB
1         1122334455      Trinh Thanh Deo      7.39
2         5566447733      Tran Bich Huu      6.72
3         6611442277      Vu Thi Tuyet      7.83
Thong tin sinh vien co DiemTB lon nhat
6611442277      Vu Thi Tuyet      7.83
Cho biet ten sinh vien muon tim kiem: Huu

Thong tin sinh vien muon tim la:
5566447733      Tran Bich Huu      6.72

```

Bài ví dụ 3: Trên cơ sở Bài 2. Hãy:

- Viết lại chương trình dạng thực đơn (*menu*).
- Bổ sung thêm chức năng sắp xếp danh sách sinh viên tăng dần theo tên.

Tham khảo thêm ở phần phụ lục

Trường ĐH CNTP TP.HCM
Khoa: Công nghệ Thông tin
Bộ môn: Công nghệ Phần mềm
MSMH: 01201007

BÀI THỰC HÀNH 2:
CÁC GIẢI THUẬT
TÌM KIẾM VÀ SẮP XẾP (tt)



A. MỤC TIÊU:

- Vận dụng các giải thuật sắp xếp dữ liệu vào từng bài toán cụ thể.
- Lập trình được các giải thuật sắp xếp thành các chương trình cụ thể chạy được trên máy tính.
- Làm được các bài tập ở cuối mỗi chương.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none">– Máy tính (có thể là máy bàn hoặc Laptop).– Cấu hình tối thiểu của máy tính:<ul style="list-style-type: none">▪ OS: 9x - XP/Win Vista/Win7/Win8x.▪ RAM: tối thiểu 512 MB.▪ HDD: tối thiểu 10 GB.	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH

2.1. Giải thuật sắp xếp chọn trực tiếp – Selection Sort

2.1.1. Ý tưởng giải thuật:

Chọn phần tử nhỏ nhất trong **n** phần tử ban đầu, đưa phần tử này về vị trí thứ 1 của dãy hiện hành; sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn (**n** – 1) phần tử, bắt đầu từ vị trí thứ 2; lặp lại quá trình đó trên dãy hiện hành...đến khi dãy hiện hành chỉ còn 1 phần tử thì dừng.

2.1.2. Cài đặt giải thuật:

```
void SelectionSort_Ascending(ItemType a[], int n)
{
    for(int i = 0; i < n - 1; i++)
    {
        int min = i;
        for(int j = i + 1; j < n; j++)
            if(a[j] < a[min])
```



```

        min = j; //có phần tử j nhỏ hơn phần tử tại vị trí min
    if(min != i) //nếu có phần tử khác nhỏ hơn phần tử tại vị trí i
        Swap(a[min], a[i]);
    }
}

```

2.2. Giải thuật sắp xếp chèn trực tiếp – Insertion Sort

2.2.1. Ý tưởng giải thuật:

Giả sử có 1 dãy a_0, \dots, a_{n-1} trong đó $(i - 1)$ phần tử đầu tiên a_0, \dots, a_{i-1} đã có thứ tự. Tìm cách chèn phần tử a_i vào vị trí thích hợp của đoạn đã sắp xếp để được một dãy mới a_0, \dots, a_i có i phần tử sắp xếp thứ tự.

Cứ như thế các phần tử tiếp theo cho đến hết dãy. Ta sẽ có một dãy được sắp xếp.

2.2.2. Cài đặt giải thuật:

```

void InsertionSort_Ascending(ItemType a[], int n)
{
    int i, j;
    for(i = 1; i < n; i++)
    {
        ItemType x = a[i];
        for(j = i - 1; j >= 0; j--)
            if(a[j] > x)
                a[j + 1] = a[j];
            else
                break;
        a[j + 1] = x;
    }
}

```

Hoặc có thể viết cách khác như sau:

```

void InsertionSort_Ascending(ItemType a[], int n)
{
    int i, j;
    for(i = 1; i < n; i++)
    {
        ItemType x = a[i];
        for(j = i - 1; (j >= 0 && a[j] > x); j--)
            a[j + 1] = a[j];
        a[j + 1] = x;
    }
}

```

2.3. Giải thuật sắp xếp dựa trên phân hoạch – Quick Sort.

2.3.1. Ý tưởng giải thuật:

Giải thuật QuickSort dựa trên việc phân hoạch dãy ban đầu thành 3 dãy con:

- Dãy con 1: gồm các phần tử $a_0..a_{j-1}$ có giá trị nhỏ hơn x .
- Dãy con 2: gồm các phần tử $a_j..a_i$ có giá trị bằng với x .
- Dãy con 3: gồm các phần tử $a_{i+1}..a_{n-1}$ có giá trị lớn hơn x .

Lặp lại quá trình trên với các dãy con.

2.3.2. Cài đặt giải thuật:

```
void QuickSort_Ascending(ItemType a[], int Left, int Right)
{
    if(Left >= Right) return; //Điều kiện dừng của đệ quy
    int Mid = (Left + Right)/2;
    int i = Left;
    int j = Right;
    ItemType x = a[Mid];
    do
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Swap(a[i], a[j]);
            i++;
            j--;
        }
    }while(i < j);
    QuickSort_Ascending(a, Left, j); //đệ quy nửa bên trái: từ Left → j
    QuickSort_Ascending(a, i, Right); //đệ quy nửa bên phải: từ i → Right
}
```

2.4. Giải thuật sắp xếp trộn trực tiếp – Merge Sort.

2.4.1. Ý tưởng giải thuật:

Phân hoạch dãy ban đầu a_0, a_1, \dots, a_{n-1} thành các dãy con. Sau khi phân hoạch xong, dãy ban đầu sẽ được tách thành hai dãy con (dãy phụ) theo nguyên tắc phân phối đều luân phiên. Rồi trộn từng cặp dãy con của hai dãy phụ thành một dãy với nguyên tắc thứ tự tăng dần. Lặp lại quy trình trên sau một số bước, ta sẽ nhận được một dãy chỉ gồm một dãy con không giảm.

2.4.2. Cài đặt giải thuật:

```
#define min(a, b) (a > b) ? b : a
...
void Merge(ItemType a[], int n, ItemType b[], int nb, ItemType c[],
int nc, int k)
```

```

{
    int ia, ib, ic, jb, jc, kb, kc;
    ia = ib = ic = 0;
    jb = jc = 0;
    while( (nb > 0) && (nc > 0) )
    {
        kb = Min(k, nb);
        kc = Min(k, nc);
        if (b[ib + jb] <= c[ic + jc])
        {
            a[ia++] = b[ib + jb++];
            if(jb == kb)
                while(jc < kc)
                    a[ia++] = c[ic + jc++];
        }
        else
        {
            a[ia++] = c[ic + jc++];
            if(jc == kc)
                while(jb < kb)
                    a[ia++] = b[ib + jb++];
        }
        if( (jb == kb) && (jc == kc) )
        {
            ib += kb;
            ic += kc;
            nb -= kb;
            nc -= kc;
            jb = jc = 0;
        }
    }
}

void MergeSort_Ascending(ItemType a[], int n)
{
    ItemType b[MAXSIZE], c[MAXSIZE];
    int ia, ib, ic, jc, jb; //các chỉ số trên mảng a, b, c
    int k = 1; //độ dài của dãy con khi phân hoạch
    do
    {
        //Phân bố lần lượt k phần tử từ mảng a vào 2 mảng b và c
        ia = ib = ic = 0;
        while(ia < n)

```

```
{
    for(jb = 0; (ia < n) && (jb < k); jb++)
        b[ib++] = a[ia++]; //phân bổ k phần tử vào mảng b
    for(jc = 0; (ia < n) && (jc < k); jc++)
        c[ic++] = a[ia++]; //phân bổ k phần tử vào mảng c
}
Merge(a, n, b, ib, c, ic, k);
k *= 2;
}while(k < n);
}
```

2.5. Bài tập ở lớp.

1. Giả sử ta cần quản lý một danh sách các cuốn sách tại thư viện, mỗi cuốn sách gồm những thông tin sau: Mã số sách (**MaS**: chuỗi tối đa 10 ký tự), Tên sách (**TenS**: chuỗi tối đa 50 ký tự), Năm xuất bản (**NXB**: số nguyên dương 2 bytes).

Hãy viết chương trình cài đặt danh sách trên bằng mảng. Viết các hàm tương ứng với những yêu cầu sau:

- a. Xây dựng cấu trúc dữ liệu SACH theo mô tả.
 - b. Nhập vào mảng một chiều **a** gồm **n** phần tử, mỗi phần tử là thông tin của một SACH (với $0 < n < 100$).
 - c. Tìm cuốn sách có MaS = **x** trong mảng.
 - d. In ra màn hình tất cả các SACH có năm xuất bản là 2015.
 - e. Sắp xếp mảng tăng dần theo MaS (lần lượt với từng giải thuật sau: **Selection Sort, Insertion Sort, Merge Sort, Quick Sort**).
 - f. Sắp xếp mảng tăng dần theo TenS (lần lượt với từng giải thuật sau: **Selection Sort, Insertion Sort, Merge Sort, Quick Sort**).
 - g. Sắp xếp mảng tăng dần theo MaS với giải thuật **Natural Merge Sort**.
 - h. Tìm cuốn sách có MaS = **x** (sau khi mảng đã sắp tăng dần theo MaS).
 - i. Tìm SACH có TenS = **y** (nếu có nhiều SACH thì chỉ cần tìm SACH đầu tiên).
 - j. Sắp xếp tăng dần theo năm xuất bản (NXB), nếu NXB bằng nhau thì sắp xếp tăng dần theo tên sách (TenS).
2. Giả sử ta cần quản lý một danh sách Học sinh, mỗi học sinh gồm những thông tin sau: Mã số (kiểu số nguyên dương 4 bytes), Họ và tên (kiểu chuỗi 30 ký tự), Lớp (kiểu chuỗi 10 ký tự), Điểm tổng kết (kiểu số thực có giá trị từ 0.0 \rightarrow 10.0).

Hãy viết chương trình cài đặt danh sách trên bằng mảng. Viết các hàm tương ứng với những yêu cầu sau:

- k. Xây dựng cấu trúc dữ liệu theo mô tả. (1đ)
- l. Nhập vào danh sách gồm n học sinh. (1đ)
- m. Xuất ra danh sách gồm n học sinh. (1đ)

- n. Tìm xem có học sinh nào có mã số bằng x hay không, x nhập từ bàn phím? Nếu có thì xuất ra thông tin của học sinh vừa tìm được, ngược lại thông báo là không tìm thấy học sinh nào có mã số là x . (1đ)
- o. Sắp xếp danh sách học sinh tăng dần theo **Họ và tên** bằng giải thuật **QuickSort**. Sau đó tìm xem có học sinh nào có Họ và tên trùng với y hay không, y nhập từ bàn phím, hãy sử dụng giải thuật tìm kiếm tốt nhất mà sinh viên đã học? Nếu có thì xuất ra thông tin của học sinh vừa tìm được, ngược lại thông báo là không tìm thấy học sinh nào có Họ và tên là y . (2đ)
- p. Sắp xếp danh sách các học sinh theo từng lớp (trường **Lớp** tăng dần) bằng giải thuật **SelectionSort**. (1đ)
- q. In ra danh sách các học sinh có **Điểm tổng kết** cao nhất của từng lớp. (2đ)
- r. Viết chương trình hoàn chỉnh dạng menu. (1đ)

2.6. Bài tập về nhà.

Với bài tập 5 về nhà của bài học trước (BÀI THỰC HÀNH 1). Hãy sắp xếp MaT, TenT, NTL tăng dần bằng các giải thuật: **Selection Sort**, **Insertion Sort**, **Merge Sort**, **Quick Sort**.

Bài tập mẫu hướng dẫn thực hành

Tham khảo thêm ở phần phụ lục

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 3: DANH SÁCH LIÊN KẾT ĐƠN	
--	--	--

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được danh sách liên kết đơn vào từng bài toán cụ thể.
- Làm được các bài tập ở cuối mỗi chương.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH

3.1. Khái niệm nút.

3.1.1. Cấu tạo nút.

Data	Link
-------------	-------------

- Thành phần dữ liệu (**Data**): Lưu trữ các thông tin về bản thân phần tử.
- Thành phần mối liên kết (**Link**): Lưu trữ địa chỉ của phần tử kế tiếp trong danh sách, hoặc lưu trữ giá trị **NULL** nếu là phần tử cuối danh sách.

3.1.2. Khai báo nút.

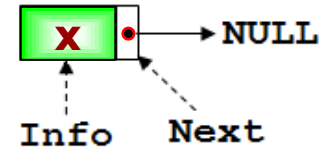
```
typedef int ItemType; //ItemType cũng có thể là kiểu float, SINHVIEN, PHANSO,...
struct SNode
{
    //Định nghĩa kiểu dữ liệu cho 1 nút của DSLK đơn là SNode
    ItemType Info; //Lưu thông tin của nút hiện hành.
    SNode* Next; //Con trỏ chỉ đến nút kế sau.
};
```

3.1.3. Tạo nút chứa giá trị x.

```

SNode* CreateSNode(ItemType x)
{
    SNode* p = new SNode;
    if(p == NULL)
    {
        printf("Không đủ bộ nhớ để cấp phát!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    return p;
}

```



3.1.4. Xuất nội dung của nút.

```

void ShowSNode(SNode* p)
{
    printf("%4d", p->Info);
}

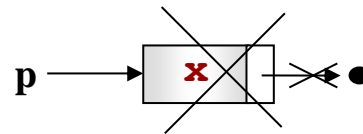
```

3.1.5. Xóa nút khỏi bộ nhớ.

```

void DeleteSNode(SNode* &p)
{
    if(p == NULL) return;
    p->Next = NULL;
    delete p;
    p = NULL;
}

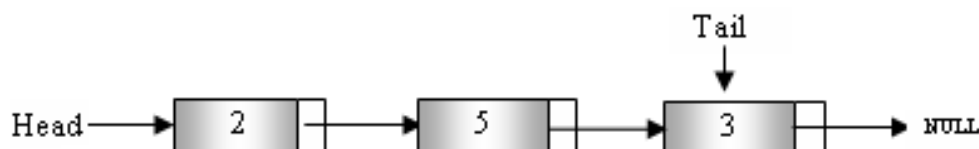
```



3.2. Danh sách liên kết đơn

3.2.1. Khái niệm danh sách liên kết đơn.

- Danh sách liên kết đơn gồm 2 con trỏ Head và Tail. Con trỏ Head trỏ đến nút đầu tiên của danh sách, và con trỏ Tail trỏ đến nút cuối cùng của danh sách.
- Mỗi phần tử liên kết với phần tử đứng kế sau nó trong danh sách, phần tử cuối trỏ **NULL**.



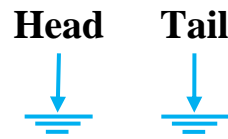
```

struct SList
{
    //Định nghĩa kiểu dữ liệu cho DSLK đơn SList
    SNode* Head; //Lưu địa chỉ nút đầu tiên trong SList
    SNode* Tail; //Lưu địa chỉ của nút cuối cùng trong SList
};

```

3.2.2. Một số phương thức trên danh sách liên kết đơn.

3.2.2.1. Khởi tạo danh sách.



```
void InitSList(SList &s1)
{ //Initialize SList
    s1.Head = NULL;
    s1.Tail = NULL;
}
```

3.2.2.2. Kiểm tra danh sách rỗng.

```
int IsEmpty(SList s1)
{
    if(s1.Head == NULL)
        return 1; //Nếu danh sách rỗng
    else
        return 0; //Nếu danh sách không rỗng
}
```

3.2.2.3. Duyệt danh sách.

```
void ShowSList(SList s1)
{
    if(IsEmpty(s1) == 1)
    {
        printf("\nDanh sach rong!");
        return;
    }
    printf("\nNoi dung cua danh sach la: ");
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
        ShowSNode(p);
}
```

3.2.2.4. Thêm nút p có giá trị x vào đầu danh sách.

```
int InsertHead(SList &s1, SNode* p)
{
    if(p == NULL)
        return 0; //Thực hiện không thành công
    if(IsEmpty(s1) == 1)
    {
        s1.Head = p;
        s1.Tail = p;
    }
    else
    {
        p->Next = s1.Head;
        s1.Head = p;
    }
}
```



```

    }
    return 1; //Thực hiện thành công
}

```

3.2.2.5. Thêm nút p có giá trị x vào cuối danh sách.

```

int InsertTail(SList &s1, SNode* p)
{
    if(p == NULL)
        return 0; //Thực hiện không thành công
    if(IsEmpty(s1) == 1)
    {
        s1.Head = p;
        s1.Tail = p;
    }
    else
    {
        s1.Tail->Next = p;
        s1.Tail = p;
    }
    return 1; //Thực hiện thành công
}

```

3.2.2.6. Tìm kiếm trong danh sách có nút p nào chứa giá trị x.

```

SNode* FindSNode(SList s1, ItemType x)
{
    if(IsEmpty(s1) == 1)
        return NULL;
    SNode* p = s1.Head;
    while( (p != NULL) && (p->Info != x) )
        p = p->Next;
    return p; //có thể NULL: không tìm thấy, hoặc khác NULL: tìm thấy
}

```

3.2.2.7. Thêm nút p có giá trị x vào sau nút q có giá trị y của danh sách.

```

int InsertAfter(SList &s1, SNode* q, SNode* p)
{
    if(q == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    p->Next = q->Next;
    q->Next = p;
    if(s1.Tail == q)
        s1.Tail = p;
    return 1; //Thực hiện thành công
}

```

3.2.2.8. Tạo danh sách (nhập bằng tay).

```

void CreateSList(SList &s1)
{
    int n;

```

```
ItemType x;
InitSList(s1);
do
{
    printf("Cho biet so phan tu cua danh sach (n>0): ");
    scanf("%d", &n);
}while(n <= 0);
for(int i = 1; i <= n; i++)
{
    printf("Nhap phan tu thu %d la: ", i);
    scanf("%d", &x);
    SNode* p = CreateSNode(x);
    InsertTail(s1, p); //Hoặc chèn vào đầu InsertHead(s1, p);
}
}
```

3.2.2.9. Tạo danh sách tự động.

```
void CreateAutoSList(SList &s1)
{
    int n;
    ItemType x;
    InitSList(s1);
    do
    {
        printf("Cho biet so phan tu cua danh sach (n>0): ");
        scanf("%d", &n);
    }while(n <= 0);
    srand((unsigned)time(NULL)); //Thư viện stdlib.h và time.h
    for(int i = 1; i <= n; i++)
    {
        x = (rand()%199)-99; //Tạo 1 số ngẫu nhiên trong đoạn [-99, 99]
        SNode* p = CreateSNode(x);
        InsertTail(s1, p); //Hoặc chèn vào đầu InsertHead(s1, p);
    }
}
```

3.3. Bài tập ở lớp.

- Viết chương trình tạo ra một danh sách đơn *sl* gồm *n* số nguyên (*n*>0):
 - Tạo cấu trúc dữ liệu một nút.
 - Tạo một nút chứa giá trị *x*.
 - Xuất nội dung của nút vừa tạo.
 - Xóa nút vừa tạo.
 - Tạo cấu trúc dữ liệu cho danh sách liên kết đơn *sl*.
 - Thêm phần tử mới vào đầu danh sách *sl*.
 - Xuất danh sách *sl* ra màn hình.
 - Thêm phần tử mới vào cuối danh sách *sl*.
 - Chèn phần tử có giá trị *x* vào sau phần tử có giá trị *y*.

- j. Chèn phần tử có giá trị x vào trước phần tử có giá trị y .
- k. Tìm kiếm một phần tử có giá trị z trong danh sách?
- l. Từ *sl* tạo 2 danh sách mới: *sl1* chứa các số chẵn, *sl2* chứa các số lẻ.
- 2. Cho danh sách đơn *sl* chứa các phân số có 2 thành phần là tử số (TuSo) và mẫu số (MauSo). Hãy:
 - a. Tạo cấu trúc dữ liệu *PhanSo*, *SList*.
 - b. Nhập/xuất danh sách có n phân số.
 - c. Tối giản các phân số.
 - d. Tính tổng/tích các phân số.
 - e. Cho biết phân số lớn nhất, phân số nhỏ nhất.
 - f. Tăng mỗi phân số của danh sách lên 1 đơn vị.
 - g. Xuất các phân số tại các vị trí chẵn.
 - h. Tìm 1 phân số p có trong danh sách *sl* hay không?

3.4. Bài tập về nhà.

- 3. Giả sử có một danh sách đơn *sl* chứa các số nguyên đã có thứ tự tăng. Cài đặt các hàm sau:
 - a. Chèn phần tử x vào danh sách sao cho vẫn giữ nguyên thứ tự tăng.
 - b. In danh sách trên theo thứ tự giảm dần.
- 4. Nối danh sách *sl2* vào sau phần tử có giá trị x trong danh sách *sl1* – nếu không có phần tử x thì thông báo “không có phần tử x ”.
- 5. Cho 2 danh sách đơn chứa các số nguyên là *sl1* và *sl2* có thứ tự tăng. Cài đặt các hàm sau:
 - a. Trộn 2 danh sách trên thành một danh sách *sl* có thứ tự tăng.
 - b. Trộn 2 danh sách trên thành một danh sách *sl* có thứ tự giảm.
 - c. Trộn 2 danh sách trên thành một danh sách *sl* sao cho các phần tử có giá trị chẵn tăng dần, các phần tử có giá trị lẻ giảm dần.
 - d. Trộn 2 danh sách trên thành một danh sách *sl* sao cho các phần tử tại những vị trí chẵn tăng dần, các phần tử tại những vị trí lẻ giảm dần.

Bài tập mẫu hướng dẫn thực hành

1) Hướng dẫn bài thực hành trên số nguyên.

//Write by: Tran Van Tho

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

typedef int ItemType; //Khai bao kieu du lieu moi co ten la ItemType
struct SNode
{
    //Định nghĩa kiểu dữ liệu cho 1 nút của DSLK đơn là SNode
    ItemType Info; //Luu thong tin cua nut
    SNode* Next; //Luu dia chi cua nut ke sau
};
struct SList
{
    //Định nghĩa kiểu dữ liệu cho DSLK đơn là SList
    SNode* Head; //Con tro tro vao nut dau danh sach
    SNode* Tail; //Con tro tro vao nut cuoi danh sach
};
//=====
void InitSList(SList &s1)
{
    //Khoi tao danh sach rong (Initialize SList)
    s1.Head = NULL;
    s1.Tail = NULL;
}
//=====
int IsEmpty(SList s1)
{
    //Trả về 1: nếu danh sách rỗng, Trả về 0: nếu ngược lại.
    if(s1.Head == NULL)
        return 1;
    else
        return 0;
}
//=====
SNode* CreateSNode(ItemType x)
{
    SNode* p = new SNode;
    if(p == NULL)
    {
        printf("\nKhong the cap phat nut moi!");
        getch();
        return NULL;
    }
    p->Info = x; //gan du lieu moi cho nut
    p->Next = NULL; //Chua co nut ke sau
    return p;
}
//=====
void ShowSNode(SNode* p)
{
    printf("%4d", p->Info);
}
//=====
int InsertHead(SList &s1, SNode* p)
{

```

```

if(p == NULL) return 0; //Thực hiện không thành công
if(IsEmpty(s1) == 1)
{
    s1.Head = p;
    s1.Tail = p;
}
else
{
    p->Next = s1.Head; //con tro next của nút mới tro vào vị trí của Head
    s1.Head = p; //con tro Head tro vào vị trí mới là vị trí của p
}
return 1; //Thực hiện thành công
}
//=====
void CreateSList(SList &s1)
{
    int n, kq;
    do
    {
        printf("\nCho biet so phan tu của danh sach: ");
        scanf("%d", &n);
    }while(n < 0);
    for(int i = 1; i <= n; i++)
    {
        ItemType x;
        printf("\nNhap mot so nguyên bat ky: ");
        scanf("%d", &x);
        SNode* p = CreateSNode(x);
        kq = InsertHead(s1, p);
        if(kq == 0)
            printf("\nKhông thêm được nút mới.");
    }
}
//=====
void CreateAutoSList(SList &s1)
{
    int n, kq;
    do
    {
        printf("\nCho biet so phan tu của danh sach: ");
        scanf("%d", &n);
    }while(n <= 0);
    srand((unsigned)time(NULL));
    for(int i = 1; i <= n; i++)
    {
        ItemType x = (rand()%199)-99; //Tạo 1 số trong đoạn [-99, 99]
        SNode* p = CreateSNode(x);
        kq = InsertHead(s1, p);
        if(kq == 0)
            printf("\nKhông thêm được nút mới.");
    }
}
//=====
void ShowSList(SList s1)
{
    if(IsEmpty(s1) == 1)
    {

```

```

        printf("\nDanh sach rong!");
        return;
    }
    printf("\nNoi dung cua danh sach la: ");
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
        ShowSNode(p);
}
//=====
SNode* FindSNode(SList s1, ItemType x)
{
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
        if(p->Info == x)
            return p; //Tra ve con tro co gia tri bang voi x
    return NULL; //Khong co nut nao co gia tri bang voi x
}
//=====
int InsertAfter(SList &s1, SNode* q, SNode* p)
{
    //Them nut p chua gia tri x vao sau nut q chua gia tri y
    if(p == NULL || q == NULL) return 0; //Thực hiện không thành công
    p->Next = q->Next;
    q->Next = p;
    if(s1.Tail == q)
        s1.Tail = p;
    return 1; //Thực hiện thành công
}
//=====
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                MENU                *");
    printf("\n*-----*");
    printf("\n* 1. Tao nut moi va xuat ra noi dung    *");
    printf("\n* 2. Tao danh sach bang tay (nhap lieu) *");
    printf("\n* 3. Tao danh sach tu dong (ngau nhieu) *");
    printf("\n* 4. Xuat noi dung danh sach            *");
    printf("\n* 5. Them nut vao dau danh sach          *");
    printf("\n* 6. Them nut vao cuoi danh sach         *");
    printf("\n* 7. Them nut p chua x vao sau nut q chua y *");
    printf("\n* 8. Tim mot nut chua gia tri x bat ky   *");
    printf("\n* 0. Thoat chuong trinh                 *");
    printf("\n*****");
}
//=====
void Process()
{
    ItemType X, Y;
    SNode *P, *Q;
    int SelectFunction, kq;
    SList SL;
    InitSList(SL);
    do
    {
        ShowMenu();
        do
        {
            printf("\nBan hay chon mot chuc nang (0→8): ");
            scanf("%d", &SelectFunction);

```

```

}while(SelectFunction < 0 || SelectFunction > 8);
switch(SelectFunction)
{
    case 1:
        printf("\nNhập một số nguyên bất kỳ: ");
        scanf("%d", &X);
        P = CreateSNode(X);
        printf("\nNội dung nút mới vừa tạo là: ");
        ShowSNode(P);
        break;
    case 2:
        InitSList(SL);
        CreateSList(SL);
        ShowSList(SL);
        break;
    case 3:
        InitSList(SL);
        CreateAutoSList(SL);
        ShowSList(SL);
        break;
    case 4:
        ShowSList(SL);
        break;
    case 5:
        printf("\nCho biết giá trị nút muốn thêm đầu danh sách: ");
        scanf("%d", &X);
        P = CreateSNode(X);
        kq = InsertHead(SL, P);
        if(kq == 0)
            printf("\nKhông thực hiện được việc thêm đầu danh sách");
        else
            ShowSList(SL);
        break;
    case 6:
        //Giống như thêm vào đầu
        break;
    case 7:
        printf("\nNhập một số nguyên x bất kỳ muốn thêm: ");
        scanf("%d", &X);
        printf("\nNhập một giá trị nút kế trước: ");
        scanf("%d", &Y);
        P = CreateSNode(X); //nút P mới chưa X
        Q = FindSNode(SL, Y); //tìm nút Q chưa giá trị Y
        kq = InsertAfter(SL, Q, P);
        if(kq == 0)
            printf("\nKhông thực hiện được việc thêm %d sau %d", X, Y);
        else
            ShowSList(SL);
        break;
    case 8:
        printf("\nNhập một số nguyên x bất kỳ muốn tìm: ");
        scanf("%d", &X);
        P = FindSNode(SL, X); //tìm nút P chưa giá trị X
        if(P == NULL)
            printf("\nKhông tìm thấy nút nào có giá trị bằng %d", X);
        else
            printf("\nTồn tại nút có giá trị bằng %d", X);
}

```

```
        break;
    }
}while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}
```

2) Hướng dẫn bài thực hành trên cấu trúc Phân số.

//Write by: Tran Van Tho

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

struct PhanSo
{
    int TuSo;
    int MauSo;
};
typedef PhanSo ItemType; //Khái báo kiểu dữ liệu mới là ItemType
struct SNode
{ //Kiểu dữ liệu mới để chứa thông tin của 1 nút
    ItemType Info; //Lưu thông tin của nút
    SNode* Next; //Lưu địa chỉ của nút kế sau
};

struct SList
{ //Kiểu dữ liệu của 1 DSLK Đơn
    SNode* Head; //Con trỏ trỏ vào nút đầu danh sách
    SNode* Tail; //Con trỏ trỏ vào nút cuối danh sách
};
//=====
void InitSList(SList &s1)
{ //Khởi tạo danh sách rỗng (Initialize SList)
    s1.Head = NULL;
    s1.Tail = NULL;
}
//=====
int IsEmpty(SList s1)
{ //Trả về 1: nếu danh sách rỗng, Trả về 0: nếu ngược lại.
    if(s1.Head == NULL)
        return 1;
    else
        return 0;
}
//=====
SNode* CreateSNode(ItemType x)
{
    SNode* p = new SNode;
    if(p == NULL)
    {
        printf("\nKhông thể cấp phát nút mới!");
        getch();
    }
}
```



```

        return NULL;
    }
    p->Info = x; //gan du lieu moi cho nut
    p->Next = NULL; //Chua co nut ke sau
    return p;
}
//=====
void KiemTraDoiDau(PhanSo &x)
{
    if(x.MauSo < 0)
    {
        x.TuSo *= -1;
        x.MauSo *= -1;
    }
}
//=====
int TimUSCLN(int x, int y)
{
    if(x == 0 || y == 0) return 1;
    x = abs(x);
    y = abs(y);
    while(x != y)
        if(x > y)
            x -= y;
        else
            y -= x;
    return x;
}
//=====
void RutGonPhanSo(PhanSo &x)
{
    KiemTraDoiDau(x);
    int usc = TimUSCLN(x.TuSo, x.MauSo);
    x.TuSo /= usc;
    x.MauSo /= usc;
}
//=====
void NhapPhanSo(ItemType &x)
{
    do
    {
        printf("\nNhap tu so: ");
        scanf("%d", &x.TuSo);
    }while(x.TuSo == 0);
    do
    {
        printf("Nhap mau so: ");
        scanf("%d", &x.MauSo);
    }while(x.MauSo == 0);
    RutGonPhanSo(x);
}
//=====
void XuatPhanSo(ItemType x)
{
    printf("%d/%d", x.TuSo, x.MauSo);
}
//=====

```

```
void ShowSNode(SNode* p)
{
    XuatPhanSo(p->Info);
}
//=====
int InsertTail(SList &s1, SNode* p)
{
    if(p == NULL) return 0; //Thực hiện không thành công
    if(IsEmpty(s1) == 1)
    {
        s1.Head = p;
        s1.Tail = p;
    }
    else
    {
        s1.Tail->Next = p; //con tro next của nút Tail tro vào vị trí do p đang tro
        s1.Tail = p; //con tro Tail tro vào vị trí mới là vị trí của p
    }
    return 1; //Thực hiện thành công
}
//=====
void CreateAutoSList(SList &s1)
{
    int n;
    do
    {
        printf("\nCho biet so phan tu của danh sach: ");
        scanf("%d", &n);
    }while(n <= 0);
    srand((unsigned)time(NULL));
    for(int i = 1; i <= n; i++)
    {
        ItemType ps;
        int x, kq;
        do
        {
            // tạo tử số
            x = (rand()%199) - 99; //Tạo 1 số ngẫu nhiên trong đoạn [-99, 99]
        }while(x == 0);
        ps.TuSo = x;
        do
        {
            // tạo mẫu số
            x = (rand()%199) - 99; //Tạo 1 số ngẫu nhiên trong đoạn [-99, 99]
        }while(x == 0);
        ps.MauSo = x;
        RutGonPhanSo(ps);
        SNode* p = CreateSNode(ps);
        kq = InsertTail(s1, p);
        if(kq == 0)
        {
            printf("\nKhong the them phan so ");
            XuatPhanSo(p->Info);
        }
    }
}
//=====
void ShowSList(SList s1)
{

```

```

if(IsEmpty(s1) == 1)
{
    printf("\nDanh sach rong!");
    return;
}
printf("\nNoi dung cua danh sach la: ");
for(SNode* p = s1.Head; p != NULL; p = p->Next)
{
    ShowSNode(p);
    printf(" ");
}
}
//=====
int SoSanhPhanSo(PhanSo p1, PhanSo p2)
{ //Hàm trả về 0: nếu p1==p2; 1: nếu p1>p2; -1: nếu p1<p2
    int kq = (p1.TuSo * p2.MauSo) - (p1.MauSo * p2.TuSo);
    if(kq > 0)
        return 1; //p1 Lớn hơn p2
    else if(kq < 0)
        return -1; //p1 nhỏ hơn p2
    else
        return 0; //p1 bằng với p2
}
//=====
SNode* FindSNode(SList s1, ItemType x)
{
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
        if(SoSanhPhanSo(p->Info, x) == 0)
            return p; //Tra ve con tro co gia tri bang voi x
    return NULL; //Khong co nut nao co gia tri bang voi x
}
//=====
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                MENU                *");
    printf("\n*-----*");
    printf("\n* 1. Tao nut moi va xuat ra noi dung      *");
    printf("\n* 2. Tao danh sach tu dong (ngau nhieu)   *");
    printf("\n* 3. Xuat noi dung danh sach             *");
    printf("\n* 4. Tim mot nut chua gia tri x bat ky    *");
    printf("\n* 0. Thoat chuong trinh                  *");
    printf("\n*****");
}
//=====
void Process()
{
    ItemType X;
    SNode *P;
    int SelectFunction;
    SList SL;
    InitSList(SL);
    do
    {
        ShowMenu();
        do
        {

```

```
printf("\nBan hay chon mot chuc nang (0→4): ");
scanf("%d", &SelectFunction);
}while(SelectFunction < 0 || SelectFunction > 4);
switch(SelectFunction)
{
    case 1:
        NhapPhanSo(X);
        P = CreateSNode(X);
        printf("\nNoi dung nut moi vua tao la: ");
        ShowSNode(P);
        break;
    case 2:
        InitSList(SL);
        CreateAutoSList(SL);
        ShowSList(SL);
        break;
    case 3:
        ShowSList(SL);
        break;
    case 4:
        NhapPhanSo(X);
        P = FindSNode(SL, X); //tim nut p chua gia tri x
        if(P == NULL)
        {
            printf("\nKhong tim thay nut nao co gia tri bang ");
            XuatPhanSo(X);
        }
        else
        {
            printf("\nTon tai nut co gia tri bang ");
            ShowSNode(P);
        }
        break;
}
}while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}
```

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 4: DANH SÁCH LIÊN KẾT ĐƠN (tt)	
--	---	---

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được danh sách liên kết vào từng bài toán cụ thể.
- Làm được các bài tập ở cuối mỗi chương.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**4.1. Một số phương thức trên danh sách liên kết đơn. (tt)****4.1.1. Xóa nút đầu danh sách.**

```

int DeleteHead(SList &s1, ItemType &x)
{
    if(IsEmpty(s1) == 1)
        return 0; //Không thực hiện được
    SNode* p = s1.Head;
    s1.Head = s1.Head->Next;
    if(s1.Head == NULL)
        s1.Tail = NULL;
    x = p->Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}

```

4.1.2. Xóa nút cuối danh sách.

```

int DeleteTail(SList &s1, ItemType &x)
{
    if(IsEmpty(s1) == 1)

```

```
    return 0; //Không thực hiện được
    SNode* q = s1.Head;
    SNode* p = s1.Tail;
    if(p == q)
        s1.Head = s1.Tail = NULL;
    else
    {
        while(q->Next != p) //Tìm nút kế trước nút Tail
            q = q->Next;
        s1.Tail = q;
        s1.Tail->Next = NULL;
    }
    x = p->Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
```

4.1.3. Xóa một nút sau nút q của danh sách.

```
int DeleteAfter(SList &s1, SNode* q, ItemType &x)
{
    if(q == NULL || q->Next == NULL)
        return 0; //Không thực hiện được
    SNode* p = q->Next;
    q->Next = p->Next;
    if(p == s1.Tail)
        s1.Tail = q;
    x = p->Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
```

4.1.4. Xóa nút có giá trị x của danh sách.

```
int DeleteSNodeX(SList &s1, ItemType x)
{
    if(IsEmpty(s1) == 1)
        return 0; //Không thực hiện được
    SNode* p = s1.Head;
    SNode* q = NULL; //sẽ trỏ đến nút kế trước p
    while( (p != NULL) && (p->Info != x) )
    { //Vòng lặp tìm nút p chứa x, q là nút kế trước p
        q = p;
        p = p->Next;
    }
    if(p == NULL) //không tìm thấy phần tử có khóa bằng x
        return 0; //Không thực hiện được
    if(p == s1.Head) //p có khóa bằng x là nút đầu danh sách
        DeleteHead(s1, x);
    else // xóa nút p có khóa x nằm kế sau nút q
        DeleteAfter(s1, q, x);
    return 1; //Thực hiện thành công
}
```

4.1.5. Sắp xếp danh sách (tăng dần trên trường khóa)

```

void SelectionSort_Ascending_SList(SList &s1)
{
    if(IsEmpty(s1) == 1 || s1.Head == s1.Tail) return;
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
    {
        SNode* min = p;
        for(SNode* q = p->Next; q != NULL; q = q->Next)
            if(q->Info < min->Info)
                min = q;
        if(min != p)
            Swap(min->Info, p->Info); //Hoán vị thông tin 1 nút
    }
}

```

4.1.6. Xóa toàn bộ danh sách.

```

int DeleteSList(SList &s1)
{
    if(IsEmpty(s1) == 1)
        return 0; //Không thực hiện được
    while(s1.Head != NULL)
    {
        SNode* p = s1.Head;
        s1.Head = s1.Head->Next;
        delete p; //Hoặc gọi hàm đã viết DeleteSNode(p);
    }
    s1.Tail = NULL;
    return 1; //Thực hiện thành công
}

```

4.2. Bài tập ở lớp.

1. Tiếp theo **Bài tập 1 (BÀI THỰC HÀNH 3)**. Hãy cài đặt các thao tác: sắp xếp danh sách tăng dần, xóa phần tử đầu, xóa phần tử cuối, xóa phần tử có giá trị x , xóa toàn bộ danh sách.
2. Tiếp theo **Bài tập 2 (BÀI THỰC HÀNH 3)**. Hãy cài đặt các thao tác: sắp xếp danh sách để các phân số tăng dần (*tính theo giá trị của chúng*), xóa phân số đầu, xóa phân số cuối, xóa phân số p sau phân số q , xóa một phân số p , xóa những phân số >1 , xóa toàn bộ danh sách.
3. Hãy định nghĩa một DSLK đơn $s1$ để quản lý điểm của một lớp học trong một học kỳ – Mỗi SV trong lớp có các thông tin sau: Mã số sinh viên (MaSV), họ đệm (HoDem), tên (Ten), năm sinh (NamSinh), và điểm kết quả học tập (DiemKQ). Biết rằng mỗi học kì gồm nhiều cột điểm “Điểm tiêu luận (DTL), Điểm giữa kỳ (DGK), Điểm cuối kỳ (DCK), Điểm trung bình môn học (DTB)” với số tín chỉ (SoTC) tương ứng (**Lưu ý: Điểm kết quả là 1 danh sách chứa điểm các môn học của từng sinh viên. Nếu môn học có $SoTC \leq 2$ thì tỷ lệ % điểm được tính là: DTL (30%), DGK (0%) và DCK (70%), ngược lại thì tỷ lệ % điểm được tính là: DTL (20%), DGK (30%) và DCK (50%)**). Hãy cài đặt các hàm sau:
 - a. Cho biết họ tên và điểm trung bình của sinh viên có mã số là x .

- b. Cho biết các thông tin về sinh viên có tên là **x**.
- c. Sắp xếp DSSV theo chiều tăng dần theo MaSV.
- d. Sắp xếp DSSV theo chiều tăng dần của tên.
- e. Thêm một SV sao cho vẫn giữ nguyên thứ tự tăng (có kiểm tra trùng khóa).
- f. Xóa SV có MaSV = **x**.
- g. Xóa tất cả các sinh viên có tên là **x**.
- h. Tạo danh sách mới từ danh sách đã cho sao cho danh sách mới giảm dần theo điểm trung bình.
- i. In danh sách các SV được xếp loại khá.
- j. Cho biết SV có điểm trung bình cao nhất.
- k. Cho biết SV có điểm trung bình thấp nhất.
- l. Cho biết SV có điểm trung bình thấp nhất trong số các SV xếp loại giỏi.
- m. Cho biết SV có ĐTB gần **x** nhất (**x** là số thực).

(Tham khảo thêm ở phần phụ lục)

4.3. Bài tập ở nhà.

4. Hãy định nghĩa DSLK đơn **sl** để lưu trữ một đa thức (gồm nhiều đơn thức) có dạng:

$$p(x) = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n$$

và cài đặt các hàm sau:

- a. Sắp xếp đa thức theo chiều giảm dần của bậc (số mũ).
 - b. Rút gọn đa thức (sau khi đã thực hiện câu **a**).
 - c. Thêm một đơn thức mới vào sao cho vẫn giữ nguyên tính thứ tự của bậc.
 - d. Xóa một đơn thức khi biết bậc.
 - e. Tính giá trị của đa thức khi biết giá trị của **x**.
 - f. Thực hiện việc cộng 2 đa thức P và Q để tạo ra một đa thức mới.
 - g. Thực hiện việc trừ 2 đa thức P và Q để tạo ra một đa thức mới.
 - h. Thực hiện việc nhân 2 đa thức P và Q để tạo ra một đa thức mới.
5. Viết chương trình cộng, trừ, nhân, chia 2 số lớn (Biết mỗi số lớn được lưu trữ bởi 1 DSLK đơn, trong đó mỗi nút của danh sách chỉ chứa một giá trị số $\in [0, 9]$).

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 5: NGĂN XẾP (STACK)	
--	--	---

A. MỤC TIÊU:

- Hiểu được cấu trúc và cơ chế hoạt động của Stack.
- Lập trình và vận dụng được Stack vào từng bài toán cụ thể.
- Làm được các bài tập có ứng dụng Stack.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**5.1. Mô tả Stack.**

- Một stack là một cấu trúc dữ liệu mà việc thêm vào và loại bỏ phần tử chỉ được thực hiện ở một đầu duy nhất (gọi là đỉnh – top của stack).
- Là một loại cấu trúc dữ liệu hoạt động theo nguyên tắc: vào sau ra trước – LIFO (Last In First Out).

5.2. Thao tác trên Stack.**5.2.1. Khai báo Stack**

```

typedef int ItemType; //Định nghĩa kiểu dữ liệu của một phần tử
struct StackNode
{ //Định nghĩa kiểu dữ liệu cho 1 nút của Stack là StackNode
    ItemType Info;
    StackNode* Next;
};
struct Stack
{ //Định nghĩa kiểu dữ liệu cho Stack
    StackNode* Top;

```

```
};
```

5.2.2. Tạo nút mới cho Stack.

```
StackNode* CreateStackNode(ItemType x)
{
    StackNode* p = new StackNode;
    if(p == NULL)
    {
        printf("Không đủ bộ nhớ để cấp phát!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    return p;
}
```

5.2.3. Khởi tạo Stack.

```
void InitStack(Stack &s)
{ //Initialize Stack
    s.Top = NULL;
}
```

Top
↓

5.2.4. Kiểm tra Stack rỗng.

```
int IsEmpty(Stack s) //Stack có rỗng hay không
{
    if(s.Top == NULL)
        return 1; //Nếu Stack rỗng
    else
        return 0; //Nếu Stack không rỗng
}
```

5.2.5. Thêm phần tử mới p có giá trị x vào Stack.

```
int Push(Stack &s, StackNode* p)
{
    if(p == NULL)
        return 0; //Không thực hiện được
    if(IsEmpty(s) == 1)
        s.Top = p;
    else
    {
        p->Next = s.Top;
        s.Top = p;
    }
    return 1; //Thực hiện thành công
}
```

5.2.6. Lấy giá trị x và hủy phần tử đầu Stack.

```
int Pop(Stack &s, ItemType &x)
{
```

```

if(IsEmpty(s) == 1) //Stack rỗng
    return 0; //Không thực hiện được
StackNode* p = s.Top;
s.Top = s.Top→Next;
x = p→Info; //Lấy thông tin của nút ở đỉnh Stack
delete p; //Hủy nút do p trở đến
return 1; //Thực hiện thành công
}

```

5.2.7. Lấy giá trị x của phần tử ở đầu Stack.

```

int Top(Stack s, ItemType &x)
{
    if(IsEmpty(s) == 1) //Stack rỗng
        return 0; //Không thực hiện được
    x = s.Top→Info; //Lấy thông tin của nút ở đỉnh Stack
    return 1; //Thực hiện thành công
}

```

5.3. Bài tập ở lớp.

Ứng dụng Stack (cài đặt theo dạng danh sách liên kết) để viết các chương trình sau:

1. Đổi cơ số cho một số **n** từ hệ thập phân (hệ 10) sang hệ **a** bất kỳ ($a \geq 2$).
2. Khử đệ quy bài toán **Tính giai thừa**.
3. Đảo ngược 1 **chuỗi ký tự** bất kỳ.
4. Khử đệ quy cho bài toán **Tháp Hà Nội**.
5. Khử đệ quy giải thuật sắp xếp **Quick Sort**.

5.4. Bài tập ở nhà.

6. Tính toán giá trị của biểu thức toán học.
7. Kiểm tra cặp ngoặc:

Mỗi dấu “(”, “{”, or “[” đều phải có một dấu đóng tương ứng “)”, “}”, or “]”

- Đúng: `(()){([)]}`
- Sai: `)(()`
- Sai: `({ [] })`

Viết giải thuật nhận một xâu (danh sách) đầu vào gồm các ký tự mở, đóng ngoặc. Kiểm tra xâu có hợp lệ không?

Bài tập mẫu hướng dẫn thực hành

Bài 1: Đổi cơ số cho 1 số nguyên n từ hệ 10 sang hệ a (với $2 \leq a \leq 9$)?

Bước 1: Tạo một **Project** mới → đặt tên: Stack_DoiCoSo_<Tên sinh viên>

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

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

Bước 3: Khai báo cấu trúc dữ liệu cho chương trình.

```
typedef int ItemType;
```

```
struct StackNode
```

```
{
```

```
    ItemType Info;
```

```
    StackNode* Next;
```

```
};
```

```
struct Stack
```

```
{
```

```
    StackNode* Top;
```

```
};
```

Bước 4: Viết các hàm cần thiết cho chương trình như sau:

```
//*****
```

```
StackNode* CreateStackNode(ItemType x)
```

```
{
```

```
    StackNode* p = new StackNode;
```

```
    if(p == NULL)
```

```
    {
```

```
        printf("Không đủ bộ nhớ để cấp phát!");
```

```
        getch();
```

```
        return NULL;
```

```
    }
```

```
    p->Info = x;
```

```
    p->Next = NULL;
```

```
    return p;
```

```
}
```

```
//*****
```

```
void InitStack(Stack &s)
```

```
{ //Initialize Stack
```

```
    s.Top = NULL;
```

```
}
```

```
//*****
```

```
int IsEmpty(Stack s)
```

```
{//Tra ve 1 neu rong, nguoc lai tra ve 0
```

```
    return (s.Top == NULL);
```

```
}
```

```
//*****
```

```
int Push(Stack &s, StackNode* p)
```

```
{
```

```
    if(p == NULL)
```

```
        return 0; //Không thực hiện được
```

```
    if(IsEmpty(s) == 1)
```

```
        s.Top = p;
```

```
    else
```

```
    {
```

```
        p->Next = s.Top;
```

```
        s.Top = p;
```

```
    }
```

```
    return 1; //Thực hiện thành công
```

```
}
```

```

//*****
int Pop(Stack &s, ItemType &x)
{
    if(IsEmpty(s) == 1) //Stack rỗng
        return 0; //Không thực hiện được
    StackNode* p = s.Top;
    s.Top = s.Top->Next;
    x = p->Info; //Lấy thông tin của nút ở đỉnh Stack
    delete p; //Hủy nút do p trở về
    return 1; //Thực hiện thành công
}
//*****
void Stack_DoiCoSo(int n, int a)
{ //Đổi từ hệ 10 sang một hệ số a (a>=2 và a<=9).
    int sodu, x;
    StackNode* p;
    Stack stack;
    InitStack(stack);
    while(n != 0)
    {
        sodu = n%a;
        n /= a;
        p = CreateStackNode(sodu);
        Push(stack, p);
    }
    printf("\nKet qua doi tu he 10 sang he %d la: ", a);
    while(IsEmpty(stack) == 0)
    {
        Pop(stack, x);
        printf("%d", x);
    }
}

```

Bước 5: Viết hàm main để thực thi chương trình.

```

//*****
void main()
{
    int n, heso, sodu;
    ItemType x;
    do{
        printf("Hay nhap mot so he 10: ");
        scanf ("%d", &n);
    }while(n <= 0);
    do
    {
        printf("Hay nhap co so muon doi: ");
        scanf ("%d", &heso);
    }while(heso <= 0);
    Stack_DoiCoSo(n, heso);
    getch();
}

```

➔ **Hãy mở rộng bài toán với cơ số a bất kỳ, ví dụ như $a = 16$?**

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 6: HÀNG ĐỢI (QUEUE)	
--	--	--

A. MỤC TIÊU:

- Hiểu được cấu trúc và cơ chế hoạt động của Queue.
- Lập trình và vận dụng được Queue vào từng bài toán cụ thể.
- Làm được các bài tập có ứng dụng Queue.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH

6.1. Mô tả Queue.

- Một queue là một cấu trúc dữ liệu mà việc thêm phần tử được thực hiện ở cuối (Tail) và xóa phần tử được thực hiện ở đầu (Head).
- Là một loại cấu trúc dữ liệu hoạt động theo nguyên tắc: vào trước ra trước – FIFO (First In First Out).

6.2. Thao tác trên Queue.

6.2.1. Khai báo Queue

```
typedef int ItemType; //Định nghĩa kiểu dữ liệu của một phần tử
struct QueueNode
{
    //Định nghĩa kiểu dữ liệu cho 1 nút của Queue là QueueNode
    ItemType Info;
    QueueNode* Next;
};
struct Queue
{
    //Định nghĩa kiểu dữ liệu cho Queue
    QueueNode* Head;
    QueueNode* Tail;
};
```

6.2.2. Tạo nút mới cho Queue.

```

QueueNode* CreateQueueNode(ItemType x)
{
    QueueNode* p = new QueueNode;
    if(p == NULL)
    {
        printf("Không đủ bộ nhớ để cấp phát!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    return p;
}

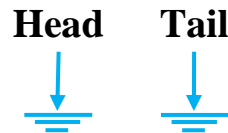
```

6.2.3. Khởi tạo Queue.

```

void InitQueue(Queue &q)
{ //Initialize Queue
    q.Head = NULL;
    q.Tail = NULL;
}

```

**6.2.4. Kiểm tra Queue rỗng.**

```

int IsEmpty(Queue q)
{ //Tra ve 1 neu queue rong, nguoc lai tra ve 0
    if(q.Head == NULL)
        return 1;
    else
        return 0;
}

```

Hoặc có thể viết gọn hơn như sau:

```

int IsEmpty(Queue q)
{ //Tra ve 1 neu queue rong, nguoc lai tra ve 0
    return (q.Head == NULL);
}

```

6.2.5. Thêm phần tử mới p có giá trị x vào Queue.

```

int Insert(Queue &q, QueueNode* p)
{
    if(p == NULL)
        return 0; //Thực hiện không thành công
    if(IsEmpty(q) == 1)
        q.Head = p;
    else
        q.Tail->Next = p;
    q.Tail = p;
    return 1; //Thực hiện thành công
}

```

6.2.6. Lấy giá trị và hủy phần tử đầu Queue.

```
int Remove(Queue &q, ItemType &x)
{
    if(IsEmpty(q) == 1)
        return 0; //Thực hiện không thành công
    QueueNode* p = q.Head;
    q.Head = q.Head→Next;
    if(q.Head == NULL)
        q.Tail = NULL;
    x = p→Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
```

6.2.7. Xem giá trị phần tử ở đầu Queue.

```
int Head(Queue q, ItemType &x)
{
    if(IsEmpty(q) == 1)
        return 0; //Không thực hiện được
    x = q.Head→Info; //Lấy thông tin của nút bị hủy
    return 1; //Thực hiện thành công
}
```

6.2.8. Xem giá trị phần tử ở cuối Queue.

```
int Tail(Queue q, ItemType &x)
{
    if(IsEmpty(q) == 1)
        return 0; //Không thực hiện được
    x = q.Tail→Info; //Lấy thông tin của nút bị hủy
    return 1; //Thực hiện thành công
}
```

6.2.9. Xem nội dung của Queue.

```
void ShowQueue(Queue q)
{
    if(IsEmpty(q) == 1)
    {
        printf("\nHàng đợi rỗng!");
        return; //Không thực hiện được
    }
    printf("\nNội dung hàng đợi là: ");
    for(QueueNode* p = q.Head; p != NULL; p = p→Next)
        printf("%4d", p→Info);
}
```

6.3. Bài tập ở lớp.

Ứng dụng Queue để (cài đặt theo dạng danh sách liên kết) viết các chương trình sau:

1. Cho dãy số nguyên **a** có **n** phần tử (**n > 0**), hãy:

- Thêm một phần tử vào hàng đợi.
- Xem nội dung hàng đợi.

- c. Xem giá trị phần tử ở đầu hàng đợi.
 - d. Xem giá trị phần tử ở cuối hàng đợi.
 - e. Lấy một phần tử ra khỏi hàng đợi.
 - f. Tạo ngẫu nhiên dãy số nguyên **a** và thêm lần lượt từng phần tử vào hàng đợi.
 - g. Giả sử dãy số nguyên **a** đã có giá trị, hãy thêm lần lượt từng phần tử của **a** vào hàng đợi.
 - h. Lấy ra khỏi hàng đợi và xem toàn bộ nội dung đó.
2. Dựa trên bài toán người sản xuất (NSX)- người tiêu dùng (NTD). NSX sản xuất ra hàng hóa và chất vào kho thứ tự, NTD lấy hàng hóa ra theo thứ tự của queue, cái gì chất vào trước thì tiêu thụ trước (nếu không nó hỏng). Hãy mô phỏng bài toán với các thao tác sau:
- a. Nhập một danh sách **n** (**n**>0) mặt hàng vào kho.
 - b. Xem thông tin tất cả hàng hóa trong kho.
 - c. Xem thông tin mặt hàng chuẩn bị được xuất kho.
 - d. Xuất khỏi kho một mặt hàng và cho xem thông tin của mặt hàng đó.
 - e. Xem thông tin mặt hàng mới vừa nhập vào kho.
 - f. Tìm và xem thông tin của một mặt hàng bất kỳ trong kho.
 - g. Xuất toàn bộ hàng hóa trong kho.
3. Mô phỏng việc xếp hàng mua vé tàu (xe lửa).

6.4. Bài tập ở nhà.

4. Mô phỏng việc xếp hàng khám bệnh.
5. Mô phỏng việc xếp hàng chứng giấy tờ tại 1 phòng công chứng.

Bài tập mẫu hướng dẫn thực hành

Bài 1: Ứng dụng Queue để viết chương trình mô phỏng việc xếp hàng mua vé?

Bước 1: Tạo một **Project** mới → đặt tên: Queue_MuaVe_<Tên sinh viên>

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

Bước 3: Khai báo cấu trúc dữ liệu cho chương trình.

```
#define MAXSIZE 15
struct ThôngTinNguoiMuaVe
{
    char HoTen[31];
    char CMND[MAXSIZE];
}
```

```
};  
typedef ThôngTinNgườiMuaVé ItemType;  
struct QueueNode  
{  
    ItemType Info;  
    QueueNode* Next;  
};  
struct Queue  
{  
    QueueNode* Head;  
    QueueNode* Tail;  
};
```

Bước 4: Viết các hàm cần thiết cho chương trình như sau:

```
//=====
QueueNode* CreateQueueNode(ItemType x)
{
    QueueNode* p = new QueueNode;
    if(p == NULL)
        return NULL;
    p->Info = x;
    p->Next = NULL;
    return p;
}
//=====
void InitQueue(Queue &q)
{ //Initialize Queue
    q.Head = NULL;
    q.Tail = NULL;
}
//=====
int IsEmpty(Queue q)
{
    if(q.Head == NULL)
        return 1;
    else
        return 0;
}
//=====
int Insert(Queue &q, QueueNode* p)
{
    if(p == NULL) return 0;
    if(IsEmpty(q) == 1)
        q.Head = p;
    else
        q.Tail->Next = p;
    q.Tail = p;
    return 1;
}
//=====
int Remove(Queue &q, ItemType &x)
{
    if(IsEmpty(q) == 1)
        return 0;
    QueueNode* p = q.Head;
    q.Head = q.Head->Next;
    if(q.Head == NULL)
```

```

        q.Tail == NULL;
    x = p->Info;
    delete p;
    return 1;
}
//=====
long int KiemTraNhapSo()
{ //Chỉ cho nhập số, kết thúc khi nhấn phím Enter
    long int n;
    int i = 0;
    char num[MAXSIZE], getnum;
    while((getnum = getch()) != 13)
    { //13 là Enter
        if(getnum >= '0' && getnum <= '9' && i < MAXSIZE)
        {
            num[i++] = getnum;
            putchar(getnum);
        }
    }
    num[i] = '\0';
    n = atol(num); //atol: chuyển chuỗi sang số nguyên
    return n;
}
//=====
char* KiemTraCMND()
{ //Chỉ cho nhập số, kết thúc khi nhấn phím Enter
    int i = 0;
    char num[MAXSIZE], getnum;
    while((getnum = getch()) != 13)
    { //13 là Enter
        if(getnum >= '0' && getnum <= '9' && i < MAXSIZE - 1)
        {
            num[i++] = getnum;
            putchar(getnum); //printf("%c", getnum);
        }
    }
    num[i] = '\0';
    return num;
}
//=====
void NhapTTNguoiMuaVe(ItemType &x)
{
    printf("\nNhập họ và tên của người mua vé: ");
    fflush(stdin);
    gets(x.HoTen);
    printf("Nhập số CMND của người mua vé: ");
    fflush(stdin);
    strcpy(x.CMND, KiemTraCMND());
}
//=====
void XemTTNguoiMuaVe(ItemType x)
{
    printf("%-30s%-7s", x.HoTen, x.CMND);
}
//=====
void TaoHangDoi(Queue &q)
{

```

```

int n;
ItemType x;
QueueNode* p;
do
{
    printf("\nCho biet so luong nguoi xep hang mua ve: ");
    n = KiemTraNhapSo();
}while(n <= 0);
for(int i = 1; i <= n; i++)
{
    NhapTTNguoiMuaVe(x);
    p = CreateQueueNode(x);
    Insert(q, p);
}
}
//=====
void XemNoiDungHangDoi(Queue q)
{
    if(IsEmpty(q) == 1)
    {
        printf("\nKhong co nguoi nao dang xep hang.");
        return;
    }
    int stt = 1;
    ItemType x;
    printf("\nDANH SACH NGUOI XEP HANG MUA VE TRONG HANG DOI");
    printf("\n%-5s%-30s%-7s", "STT", "HO VA TEN NGUOI XEP HANG", "SO CMND");
    QueueNode* p = q.Head;
    while(p != NULL)
    {
        printf("\n%-5d", stt++);
        x = p->Info;
        XemTTNguoiMuaVe(x);
        p = p->Next;
    }
}
//=====
QueueNode* TimNguoiMuaVe_TheoTen(Queue q, char ten[])
{
    QueueNode* p = q.Head;
    while(p != NULL)
    {
        if(strcmp(p->Info.HoTen, ten) == 0)
            return p;
        p = p->Next;
    }
    return NULL;
}
//=====

```

Bước 5: Viết hàm main để thực thi chương trình.

```

void ShowMenu()
{
    printf("\n*****");
    printf("\n*                               MENU                               *");
    printf("\n*-----*");
    printf("\n* 1. Tao hang doi *");
    printf("\n* 2. Xem noi dung hang doi *");
}

```

```

printf("\n* 3. Them nguoi xep hang vao hang doi      *");
printf("\n* 4. Goi nguoi trong hang doi mua ve      *");
printf("\n* 5. Tim thong tin nguoi mua ve            *");
printf("\n* 0. Thoat chuong trinh                    *");
printf("\n*****");
}
//=====
void Process()
{
    Queue Q;
    ItemType X;
    int SelectFunction;
    int kq;
    QueueNode* P;
    do
    {
        ShowMenu();
        do
        {
            printf("\nBan hay chon mot chuc nang: ");
            SelectFunction = KiemTraNhapSo();
        } while (SelectFunction < 0 || SelectFunction > 5);
        switch (SelectFunction)
        {
            case 1:
                InitQueue(Q);
                TaoHangDoi(Q);
                break;
            case 2:
                XemNoiDungHangDoi(Q);
                break;
            case 3:
                printf("\nNhap thong tin nguoi mua ve");
                NhapTTNguoiMuaVe(X);
                P = CreateQueueNode(X);
                kq = Insert(Q, P);
                if (kq == 1)
                    printf("\nThem thanh cong!");
                else
                    printf("\nThem khong thanh cong!");
                break;
            case 4:
                kq = Remove(Q, X);
                if (kq == 1)
                {
                    printf("\nThong tin nguoi vua ra khoi hang doi\n");
                    XemTTNguoiMuaVe(X);
                }
                else
                    printf("\nHang doi rong!");
                break;
            case 5:
                char ten[30];
                printf("\nCho biet Ho va ten nguoi mua ve: ");
                fflush(stdin);
                gets(ten);
                P = TimNguoiMuaVe_TheoTen(Q, ten);
        }
    }
}

```

```
        if(P)
        {
            printf("\nThong tin nguoi can tim la: \n");
            XemTTNguoiMuaVe(P->Info);
        }
        else
            printf("\nKhong tim thay nguoi nao co ten %s", ten);
        break;
    case 0:
    }
}while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}
```

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 7: DANH SÁCH LIÊN KẾT ĐÔI	
--	--	--

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được danh sách liên kết đôi vào từng bài toán cụ thể.
- Làm được các bài tập ở cuối mỗi chương.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**7.1. Khái niệm nút.****7.1.1. Cấu tạo nút.**

Prev	Data	Next
-------------	-------------	-------------

- Thành phần dữ liệu (Data): Lưu trữ các thông tin về bản thân phần tử (nút).
- Thành phần mối liên kết (Link): Liên kết đến nút kế trước bởi con trỏ Prev, và đến nút kế sau bởi con trỏ Next.

7.1.2. Khai báo nút.

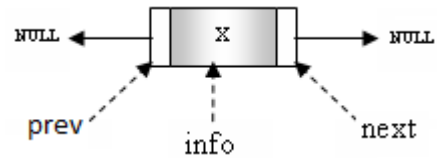
```
typedef int ItemType; //Định nghĩa kiểu dữ liệu của một phần tử
struct DNode
{ //Định nghĩa kiểu dữ liệu cho 1 nút của DSLK đôi là DNode
    ItemType Info;
    DNode* Prev;
    DNode* Next;
};
```

7.1.3. Tạo nút chứa giá trị x.

```

DNode* CreateDNode(ItemType x)
{
    DNode* p = new DNode;
    if(p == NULL)
    {
        printf("Không đủ bộ nhớ để cấp phát !");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    p->Prev = NULL;
    return p;
}

```



7.1.4. Xuất nội dung của nút.

```

void ShowDNode(DNode* p)
{
    printf("%4d", p->Info);
}

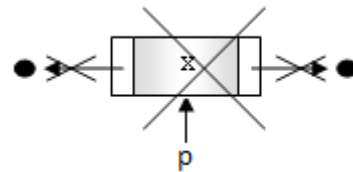
```

7.1.5. Xóa nút khỏi bộ nhớ.

```

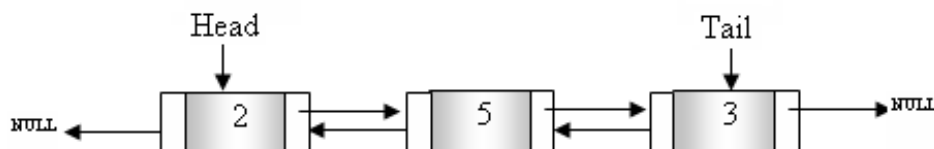
void DeleteDNode(DNode* &p)
{
    if(p == NULL) return;
    p->Prev = NULL;
    p->Next = NULL;
    delete p;
    p = NULL;
}

```



7.2. Khái niệm danh sách liên kết đôi.

- Danh sách liên kết đôi gồm 2 con trỏ Head và Tail. Con trỏ Head trỏ đến nút đầu tiên của danh sách, và con trỏ Tail trỏ đến nút cuối cùng của danh sách.
- Mỗi phần tử liên kết với phần tử đứng kế trước bởi con trỏ Prev và liên kết với phần tử đứng kế sau bởi con trỏ Next.
- Con trỏ Head có liên kết Prev bằng NULL, con trỏ Tail có liên kết Next bằng NULL.



```

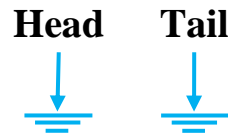
struct DList
{
    //Định nghĩa kiểu dữ liệu cho DSLK đôi
    DNode* Head; //Lưu địa chỉ nút đầu tiên trong List
    DNode* Tail; //Lưu địa chỉ của nút cuối cùng trong List
};

```

7.2.1. Một số phương thức trên danh sách liên kết đôi.

7.2.1.1. Khởi tạo danh sách.

```
void InitDList(DList &d1)
{ //Initialize DList
  d1.Head = NULL;
  d1.Tail = NULL;
}
```

**7.2.1.2. Kiểm tra danh sách rỗng.**

```
int IsEmpty(DList d1)
{
  if(d1.Head == NULL)
    return 1; //Nếu danh sách rỗng
  else
    return 0; //Nếu danh sách không rỗng
}
```

7.2.1.3. Duyệt danh sách (duyet từ đầu đến cuối).

```
void ShowDList(DList d1)
{
  if(IsEmpty(d1) == 1)
  {
    printf("\nDanh sach rong");
    return;
  }
  printf("\nNoi dung duyet xuai cua danh sach la: ");
  for(DNode* p = d1.Head; p != NULL; p = p->Next)
    ShowDNode(p);
}
```

7.2.1.4. Duyệt danh sách (duyet ngược từ cuối về đầu).

```
void ShowDList_Reverse(DList d1)
{
  if(IsEmpty(d1) == 1)
  {
    printf("\nDanh sach rong");
    return;
  }
  printf("\nNoi dung duyet nguoc cua danh sach la: ");
  for(DNode* p = d1.Tail; p != NULL; p = p->Prev)
    ShowDNode(p);
}
```

7.2.1.5. Thêm nút p có giá trị x vào đầu danh sách.

```
int InsertHead(DList &d1, DNode* p)
{
  if(p == NULL)
    return 0; //Thực hiện không thành công
  if(IsEmpty(d1) == 1)
    d1.Head = d1.Tail = p;
  else
  {
```

```
    p→Next = dl.Head;
    dl.Head→Prev = p;
    dl.Head = p;
}
return 1; //Thực hiện thành công
}
```

7.2.1.6. Thêm nút p có giá trị x vào cuối danh sách.

```
int InsertTail(DList &dl, DNode* p)
{
    if(p == NULL)
        return 0; //Thực hiện không thành công
    if(IsEmpty(dl) == 1)
        dl.Head = dl.Tail = p;
    else
    {
        p→Prev = dl.Tail;
        dl.Tail→Next = p;
        dl.Tail = p;
    }
    return 1; //Thực hiện thành công
}
```

7.2.1.7. Thêm nút p có giá trị x vào sau nút q có giá trị y của danh sách.

```
int InsertAfter(DList &dl, DNode* q, DNode* p)
{
    //Chèn p vào sau q
    if(q == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    DNode* r = q→Next;
    q→Next = p;
    p→Next = r;
    p→Prev = q;
    if(r != NULL)
        r→Prev = p;
    else //if(dl.Tail == q)
        dl.Tail = p;
    return 1; //Thực hiện thành công
}
```

7.2.1.8. Thêm nút p có giá trị x vào trước nút q có giá trị y của danh sách.

```
int InsertBefore(DList &dl, DNode* q, DNode* p)
{
    //Chèn p vào trước q
    if(q == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    DNode* r = q→Prev;
    q→Prev = p;
    p→Prev = r;
    p→Next = q;
    if(r != NULL)
        r→Next = p;
    else //if(dl.Head == q)

```

```

        dl.Head = p;
    return 1; //Thực hiện thành công
}

```

7.2.1.9. Tạo danh sách bằng tay.

```

void CreateDList(SList &dl)
{
    int n;
    ItemType x;
    InitDList(dl);
    do
    {
        printf("Cho biet so phan tu cua danh sach (n>0): ");
        scanf("%d", &n);
    }while(n <= 0);
    for(int i = 1; i <= n; i++)
    {
        printf("Nhap phần tử thứ %d là: ", i);
        scanf("%d", &x);
        DNode* p = CreatedNode(x);
        InsertTail(dl, p); //Hoặc chèn đầu InsertHead(dl, p);
    }
}

```

7.2.1.10. Tạo danh sách tự động.

```

void CreateDList_Automatic(SList &dl)
{
    int n;
    ItemType x;
    InitDList(dl);
    do
    {
        printf("Cho biet so phan tu cua danh sach (n>0): ");
        scanf("%d", &n);
    }while(n <= 0);
    srand((unsigned)time(NULL)); //Thư viện stdlib.h và time.h
    for(int i = 1; i <= n; i++)
    {
        x = (rand()%199) - 99; //Tạo 1 số ngẫu nhiên trong đoạn [-99,99]
        DNode* p = CreatedNode(x);
        InsertTail(dl, p); //Hoặc chèn đầu InsertHead(dl, p);
    }
}

```

7.2.1.11. Tìm kiếm nút có giá trị x.

```

DNode* FindDNode(DList dl, ItemType x)
{
    if(IsEmpty(dl) == 1)
        return NULL; //Không tìm thấy
    DNode* p = dl.Head;
    while( (p != NULL) && (p->Info != x) )
        p = p->Next;
}

```

```
    return p;  
}
```

7.3. Bài tập ở lớp.

Hãy ứng dụng DSLK đôi để viết chương trình quản lý các số nguyên với yêu cầu:

1. Nhập n phần tử vào DSLK.
2. Xuất DSLK theo 2 cách duyệt xuôi và duyệt ngược.
3. Thêm một phần tử vào đầu DSLK.
4. Thêm một phần tử vào cuối DSLK.
5. Thêm một phần tử vào sau một nút q của DSLK.
6. Thêm một phần tử vào trước một nút q của DSLK.
7. Thêm một phần tử có giá trị x vào vị trí thứ k của DSLK.
8. Kiểm tra (*tìm kiếm*) một giá trị x có trong DSLK hay không?
9. Tìm phần tử lớn nhất, nhỏ nhất trong danh sách.
10. Tìm phần tử xuất hiện nhiều lần nhất trong danh sách.
11. Cho biết giá trị của phần tử thứ k trong danh sách (k bắt đầu từ 0).
12. Tính tổng các giá trị là bội của 3.
13. Tính tổng các phần tử có giá trị là số nguyên tố trong danh sách.
14. Tính tổng các phần tử xuất hiện duy nhất 1 lần trong danh sách.
15. Đếm số phần tử là số nguyên tố trong danh sách?
16. Đếm số phần tử lớn hơn/nhỏ hơn hai phần tử lân cận?
17. Đếm số lần xuất hiện của phần tử x trong danh sách?
18. Kiểm tra mọi phần tử trong danh sách có phải là số chính phương không?
19. Kiểm tra mọi phần tử có được sắp xếp tăng dần chưa?
20. Cho biết các phần tử chẵn và lẻ có nằm xen kẽ với nhau hay không?
21. Sắp xếp DSLK tăng dần bằng giải thuật sắp xếp nổi bọt (BubbleSort).
22. Thêm một phần tử x vào DSLK đã sắp xếp tăng sao cho danh sách vẫn tăng?

7.4. Bài tập ở nhà. (tiếp theo bài tập ở lớp)

1. Đếm số nút hiện có trên danh sách.
2. Đếm số đường chạy trong danh sách.
3. Tìm số âm/dương nhỏ nhất, lớn nhất (giá trị/địa chỉ).
4. Cho biết các phần tử trong danh sách có được sắp xếp giảm dần?
5. Cho biết các phần tử dương trong danh sách có được sắp xếp tăng?

Bài tập mẫu hướng dẫn thực hành**Bài 1: Ứng dụng DSLK đôi để viết chương trình quản lý danh sách các số nguyên?**

Bước 1: Tạo một **Project** mới → đặt tên: DSLKDoi_SoNguyen_<Tên sinh viên>

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

Bước 3: Khai báo cấu trúc dữ liệu cho chương trình.

```
//=====
typedef int ItemType;
struct DNode
{
    ItemType Info;
    DNode* Next;
    DNode* Prev;
};
struct DList
{
    DNode* Head;
    DNode* Tail;
};
```

Bước 4: Viết các hàm cần thiết cho chương trình như sau:

```
//=====
DNode* CreateDNode(ItemType x)
{
    DNode* p = new DNode;
    if(p == NULL)
    {
        printf("\nKhong du bo nho de cap phat!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    p->Prev = NULL;
    return p;
}
//=====
void InitDList(DList &dl)
{ //Initialize DList
    dl.Head = NULL;
    dl.Tail = NULL;
}
//=====
int IsEmpty(DList dl)
{
    if(dl.Head == NULL)
        return 1; //Nếu danh sách rỗng
    else
        return 0; //Nếu danh sách không rỗng
}
```

```
//=====
int InsertTail(DList &d1, DNode* p)
{
    if(p == NULL) return 0;
    if(IsEmpty(d1) == 1)
        d1.Head = d1.Tail = p;
    else
    {
        p->Prev = d1.Tail;
        d1.Tail->Next = p;
        d1.Tail = p;
    }
    return 1;
}
//=====
void CreateDList_Automatic(DList &d1)
{
    int n;
    printf("Ban hay cho biet so nut: ");
    scanf("%d", &n);
    InitDList(d1);
    srand((unsigned)time(NULL));
    for(int i = 0; i < n; i++)
    {
        ItemType x = rand()%199 - 99;
        DNode* p = CreateDNode(x);
        InsertTail(d1, p);
    }
    printf("\nDa tao thanh cong danh sach!");
}
//=====
void ShowDNode(DNode* p)
{
    printf("%4d", p->Info);
}
//=====
void ShowDList(DList d1)
{
    if(IsEmpty(d1) == 1)
    {
        printf("\nDanh sach rong!");
        return;
    }
    printf("\nNoi dung cua danh sach la: ");
    for(DNode* p = d1.Head; p != NULL; p = p->Next)
        ShowDNode(p);
}
//=====
DNode* FindDNode(DList d1, ItemType x)
{
    if(IsEmpty(d1) == 1)
        return NULL;
    DNode* p = d1.Head;
    while( (p != NULL) && (p->Info != x) )
        p = p->Next;
    return p;
}
```

```
//=====
int CountDNode(DList dl)
{
    int dem = 0;
    for(DNode* p = dl.Head; p != NULL; p = p->Next)
        dem++;
    return dem;
}
//=====
```

Bước 5: Viết hàm main để thực thi chương trình.

```
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                      MENU                      *");
    printf("\n*-----*");
    printf("\n* 1. Tao danh sach lien ket doi                *");
    printf("\n* 2. Xem danh sach lien ket doi                *");
    printf("\n* 3. Tim kiem mot nut co gia tri x              *");
    printf("\n* 4. Them nut p truoc nut q                    *");
    printf("\n* 5. Dem so nut cua danh sach                  *");
    printf("\n* 0. Thoat chuong trinh                        *");
    printf("\n*****");
}
//=====
void Process()
{
    int SelectFunction, dem;
    ItemType x, y;
    DNode *p, *q;
    DList dl;
    InitDList(dl);
    do
    {
        ShowMenu();
        printf("\nBan hay chon mot chuc nang (tu 0 -> 5): ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            case 1:
                CreateDList_Automatic(dl);
                break;
            case 2:
                printf("\nNoi dung danh sach: ");
                ShowDList(dl);
                break;
            case 3:
                printf("\nCho biet gia tri muon tim: ");
                scanf("%d", &x);
                p = FindDNode(dl, x);
                if(p == NULL)
                    printf("\nKhong tim thay nut nao co gia tri bang %d", x);
                else
                {
                    printf("\nThong tin nut muon tim la: ");
                    ShowDNode(p);
                }
            }
        }
    }
}
```

```
        break;
    case 4:
        printf("\nCho biết giá trị nút muốn thêm vào danh sách: ");
        scanf("%d", &x);
        printf("\nCho biết giá trị nút đứng ngay kế trước: ");
        scanf("%d", &y);
        p = CreateDNode(x);
        q = FindDNode(d1, y);
        InsertBefore(d1, q, p);
        break;
    case 5:
        dem = CountDNode(d1);
        printf("\nSố nút của danh sách là: %d", dem);
        break;
    }
} while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}
```


Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 8: DANH SÁCH LIÊN KẾT ĐÔI (tt)	
--	---	---

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được danh sách liên kết đôi vào từng bài toán cụ thể.
- Làm được các bài tập áp dụng danh sách liên kết đôi.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**8.1. Một số phương thức trên danh sách liên kết đôi.****8.1.1. Xóa nút đầu danh sách.**

```

int DeleteHead(DList &dl, ItemType &x)
{
    if(IsEmpty(dl) == 1)
        return 0; //Không thực hiện được
    DNode* p = dl.Head;
    dl.Head = dl.Head->Next;
    if(dl.Head == NULL)
        dl.Tail = NULL;
    else
        dl.Head->Prev = NULL;
    x = p->Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}

```

8.1.2. Xóa nút cuối danh sách.

```

int DeleteTail(DList &dl, ItemType &x)

```

```
{
    if(IsEmpty(dl) == 1)
        return 0; //Không thực hiện được
    DNode* p = dl.Tail;
    dl.Tail = dl.Tail→Prev;
    if(dl.Tail == NULL)
        dl.Head = NULL;
    else
        dl.Tail→Next = NULL;
    x = p→Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
```

8.1.3. Xóa một nút sau nút q của danh sách.

```
int DeleteAfter(DList &dl, DNode* q, ItemType &x)
{ //Hủy nút do p trỏ đến kế sau nút q
    if(q == NULL || q == dl.Tail)
        return 0; //Không thực hiện được
    DNode* p = q→Next;
    DNode* r = p→Next;
    q→Next = r;
    if(r == NULL)
        dl.Tail = q;
    else
        r→Prev = q;
    x = p→Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
```

8.1.4. Xóa một nút trước nút q của danh sách.

```
int DeleteBefore(DList &dl, DNode* q, ItemType &x)
{ //Hủy nút do p trỏ đến kế trước nút q
    if(q == NULL || q == dl.Head)
        return 0; //Không thực hiện được
    DNode* p = q→Prev;
    DNode* r = p→Prev;
    q→Prev = r;
    if(r == NULL)
        dl.Head = q;
    else
        r→Next = q;
    x = p→Info; //Lấy thông tin của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
```

8.1.5. Xóa nút có giá trị x của danh sách.

```
int DeleteDNodeX(DList &dl, ItemType x)
{
```

```

DNode* p = FindDNode(d1, x);
if(p == NULL)
    return 0; //Không thực hiện được
if(p == d1.Head)
    DeleteHead(d1, x);
else if(p == d1.Tail)
    DeleteTail(d1, x);
else
{
    DNode* q = p->Prev;
    DNode* r = p->Next;
    q->Next = r;
    r->Prev = q;
    delete p;
}
return 1; //Thực hiện thành công
}

```

8.1.6. Xóa toàn bộ danh sách.

```

int DeleteDList(DList &d1)
{
    if(IsEmpty(d1) == 1)
        return 0; //Không thực hiện được
    while(d1.Head != NULL)
    {
        DNode* p = d1.Head;
        d1.Head = d1.Head->Next;
        delete p; //Hoặc gọi hàm đã viết DeleteDNode(p);
    }
    d1.Tail = NULL;
    return 1; //Thực hiện thành công
}

```

8.2. Bài tập ở lớp.

1. Tiếp theo bài tập 1 (BÀI THỰC HÀNH 7) vừa học tuần trước, hãy bổ sung những thao tác sau:
 - a. Xóa phần tử đầu danh sách.
 - b. Xóa phần tử cuối danh sách.
 - c. Xóa phần tử p sau một phần tử q của danh sách.
 - d. Xóa phần tử p trước một phần tử q của danh sách.
 - e. Xóa phần tử có giá trị x của danh sách.
 - f. Xóa phần tử nhỏ nhất trong danh sách?
 - g. Xóa phần tử là số nguyên tố trong danh sách?
 - h. Xóa tất cả các phần tử trùng nhau trong DSLK đã được sắp xếp tăng (*chỉ để lại 1 đại diện duy nhất*).
 - i. Xóa toàn bộ danh sách.

2. Ứng dụng DSLK đôi để cài đặt minh họa bài toán quản lý và điều hành tuyến xe lửa. Mỗi nút của danh sách là một đoạn đường gồm các thông tin: ga trước, ga sau, chiều dài và thời gian xe lửa chạy trên đoạn đường đó. Hãy viết chương trình với các chức năng sau:
- Thêm một đoạn đường có ga trước là ga khởi hành.
 - Thêm một đoạn đường có ga sau là ga kết thúc lộ trình.
 - Thêm một đoạn đường bất kỳ vào giữa lộ trình (*vẫn giữ được thứ tự các ga*).
 - Xóa một đoạn đường có ga trước là ga khởi hành.
 - Xóa một đoạn đường có ga sau là ga kết thúc lộ trình.
 - Xóa một đoạn đường bất kỳ trong lộ trình (*vẫn giữ được thứ tự các ga*).
 - Xem lộ trình 1 (*duyet xuoi*) trên toàn tuyến đường.
 - Xem lộ trình 2 (*duyet nguc*) trên toàn tuyến đường.
 - Tính tổng chiều dài của lộ trình.
 - Tính tổng thời gian của lộ trình.
 - Cho biết đoạn đường nào dài nhất và chiều dài là bao nhiêu?
 - Cho biết đoạn đường nào xe lửa di chuyển tốn ít thời gian nhất và thời gian là bao nhiêu?
 - Xem thông tin của đoạn đường thứ k.
 - Hiệu chỉnh thông tin cho đoạn đường thứ k.

8.3. Bài tập ở nhà.

3. Ứng dụng DSLK đôi để cài đặt minh họa bài toán quản lý và điều hành tuyến xe lửa (tiếp theo bài 2). Hãy viết chương trình với các chức năng sau:
- Báo lộ trình: Nhập ga đi và ga đến thì cho biết lộ trình này đi qua những ga nào, tổng chiều dài và tổng thời gian để đi hết lộ trình này.
 - Bổ sung thêm những thông tin cần thiết vào bài toán để có thể thống kê được mỗi Tỉnh/Thành phố có bao nhiêu ga.

Bài tập mẫu hướng dẫn thực hành

Bài 1: Ứng dụng DSLK đôi để viết chương trình quản lý danh sách các số nguyên?

Bước 1: Tạo một **Project** mới → đặt tên: DSLKDoi_SoNguyen_<Tên sinh viên>

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

Bước 3: Khai báo cấu trúc dữ liệu cho chương trình.

```
//=====
typedef int ItemType;
struct DNode
{
    ItemType Info;
    DNode* Next;
    DNode* Prev;
};
struct DList
{
    DNode* Head;
    DNode* Tail;
};
```

Bước 4: Viết các hàm cần thiết cho chương trình như sau:

```
//=====
DNode* CreatedNode(ItemType x)
{
    DNode* p = new DNode;
    if(p == NULL)
    {
        printf("\nKhong du bo nho de cap phat!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    p->Prev = NULL;
    return p;
}
//=====
void InitDList(DList &d1)
{ //Initialize DList
    d1.Head = NULL;
    d1.Tail = NULL;
}
//=====
int IsEmpty(DList d1)
{
    if(d1.Head == NULL)
        return 1; //Nếu danh sách rỗng
    else
        return 0; //Nếu danh sách không rỗng
}
//=====
int InsertTail(DList &d1, DNode* p)
{
    if(p == NULL) return 0;
    if(IsEmpty(d1) == 1)
        d1.Head = d1.Tail = p;
    else
    {
        p->Prev = d1.Tail;
        d1.Tail->Next = p;
        d1.Tail = p;
    }
}
```

```

    return 1;
}
//=====
void CreatedList_Automatic(DList &d1)
{
    int n;
    printf("Ban hay cho biet so nut: ");
    scanf("%d", &n);
    InitDList(d1);
    srand((unsigned)time(NULL));
    for(int i = 1; i <= n; i++)
    {
        ItemType x = rand()%101 - 50;
        DNode* p = CreatedNode(x);
        InsertTail(d1, p);
    }
    printf("\nDa tao thanh cong danh sach!");
}
//=====
void ShowDNode(DNode* p)
{
    printf("%4d", p->Info);
}
//=====
void ShowDList(DList d1)
{
    if(IsEmpty(d1) == 1)
    {
        printf("\nDanh sach rong!");
        return;
    }
    printf("\nNoi dung cua danh sach la: ");
    for(DNode* p = d1.Head; p != NULL; p = p->Next)
        ShowDNode(p);
}
//=====
DNode* FindDNode(DList d1, ItemType x)
{
    if(IsEmpty(d1) == 1)
        return NULL;
    DNode* p = d1.Head;
    while( (p != NULL) && (p->Info != x) )
        p = p->Next;
    return p;
}
//=====
int DeleteHead(DList &d1, ItemType &x)
{
    if(IsEmpty(d1) == 1)
        return 0; //Không thực hiện được
    DNode* p = d1.Head;
    d1.Head = d1.Head->Next;
    if(IsEmpty(d1) == 1)
        d1.Tail = NULL;
    else
        d1.Head->Prev = NULL;
    x = p->Info; //Lấy ItemType của nút bị hủy
}

```

```

    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
//=====
int DeleteAfter(DList &dl, DNode* q, ItemType &x)
{ //Hủy nút do p trỏ đến sau nút q
    if(q == NULL || q == dl.Tail)
        return 0; //Không thực hiện được
    DNode* p = q->Next;
    DNode* r = p->Next;
    q->Next = r;
    if(r == NULL)
        dl.Tail = q;
    else
        r->Prev = q;
    x = p->Info; //Lấy ItemType của nút bị hủy
    delete p; //Hủy nút do p trỏ đến
    return 1; //Thực hiện thành công
}
//=====

```


Bước 5: Viết hàm main để thực thi chương trình.

```

void ShowMenu()
{
    printf("\n*****");
    printf("\n*                      MENU                      *");
    printf("\n*-----*");
    printf("\n* 1. Tạo danh sách liên kết đôi                *");
    printf("\n* 2. Xem danh sách liên kết đôi                *");
    printf("\n* 3. Xóa nút đầu danh sách                    *");
    printf("\n* 4. Xóa nút kế sau nút q                    *");
    printf("\n* 0. Thoát chương trình                      *");
    printf("\n*****");
}
//=====
void Process()
{
    int SelectFunction, kq;
    ItemType x, y;
    DNode *p, *q;
    DList dl;
    InitDList(dl);
    do
    {
        ShowMenu();
        printf("\nBan hay chon mot chuc nang (tu 0 -> 4): ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            case 1:
                CreateDList_Automatic(dl);
                break;
            case 2:
                printf("\nNoi dung danh sach: ");
                ShowDList(dl);
                break;
            case 3:

```

```
kq = DeleteHead(dl, x);
if(kq == 1)
    printf("\nGia tri nut vua bi xoa o dau danh sach la: %d", x);
else
    printf("\nXoa nut dau danh sach khong thanh cong");
break;
case 4:
    printf("\nCho biet gia tri nut ke truoc nut muon xoa: ");
    scanf("%d", &y);
    q = FindDNode(dl, y);
    kq = DeleteAfter(dl, q, x);
    if(kq == 1)
        printf("\nGia tri nut vua bi xoa khoi danh sach la: %d", x);
    else
        printf("\nXoa nut ke sau nut %d khong thanh cong", y);
    break;
}
} while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}
```


Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 9: CÂY NHỊ PHÂN - CÂY NHỊ PHÂN TÌM KIẾM	
--	--	---

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được cấu trúc dữ liệu cây vào từng bài toán cụ thể.
- Làm được các bài tập áp dụng cây.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

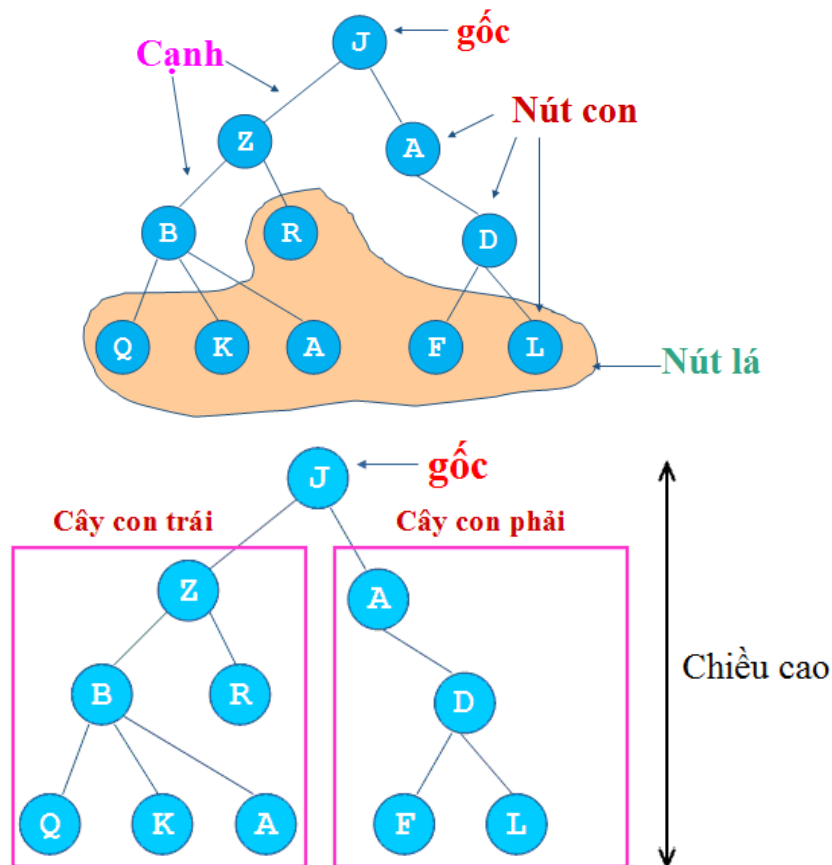
C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**10.1. Cây.****9.1.1. Khái niệm cây.**

Là một tập hợp T các phần tử (gọi là nút của cây) trong đó có 1 nút đặc biệt được gọi là nút gốc, các nút còn lại được chia thành những tập rời nhau T_1, T_2, \dots, T_n theo quan hệ phân cấp trong đó T_i cũng là 1 cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp $i + 1$. Quan hệ này người ta gọi là quan hệ cha-con.

- Cây là tập hợp các nút và cạnh nối các nút đó.
- Có một nút gọi là gốc.
- Các nút còn lại được chia thành những tập rời nhau T_1, T_2, \dots, T_n theo quan hệ phân cấp trong đó T_i cũng là 1 cây.
- Quan hệ one-to-many giữa các nút.
- Có duy nhất một đường đi từ gốc đến nút con.



Thuật ngữ:

- Bậc của một nút: là số cây con của nút đó (nút lá có bậc bằng 0).
- Bậc của một cây: là bậc lớn nhất của các nút trong cây (số cây con tối đa của một nút trong cây). Cây có bậc **n** thì gọi là cây **n**-phân.
- Nút gốc: không có nút cha.
- Nút lá: không có nút con hay nút có bậc bằng 0.
- Nút nhánh: không phải nút lá và nút gốc hay nút có bậc khác 0.
- Các nút có cùng một nút cha gọi là nút anh em (nút đồng cấp).
- Độ dài đường đi từ gốc đến nút **x**: là số nhánh cần đi qua kể từ gốc đến **x**.
- Độ dài đường đi của một cây: được định nghĩa là tổng các độ dài đường đi của tất cả các nút của cây.
- Mức của một nút: là độ dài đường đi từ gốc đến nút đó.
- Chiều cao của một nút: là mức của nút đó cộng thêm 1.
- Chiều cao của một cây: là chiều cao lớn nhất của các nút trong cây.
- Rừng là tập hợp các cây. Như vậy, nếu một cây bị loại bỏ nút gốc có thể cho ta một rừng.

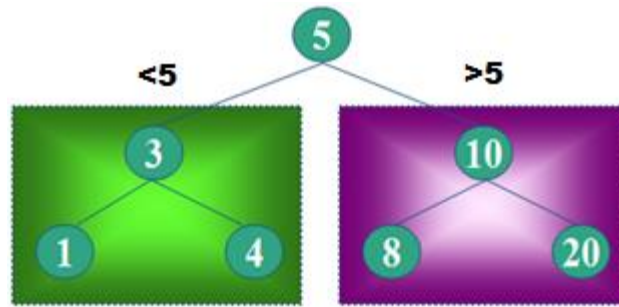
9.1.2. Khái niệm cây nhị phân.

Cây nhị phân là cây mà mỗi nút có tối đa 2 cây con.

10.2. Khái niệm cây nhị phân tìm kiếm.

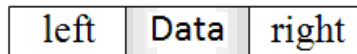
Cây nhị phân tìm kiếm là cây nhị phân mà mỗi nút phải thỏa điều kiện sau:

- Giá trị của tất cả nút con trái < nút gốc.
- Giá trị của tất cả nút con phải > nút gốc.



9.2.1. Khái niệm nút.

9.2.1.1. Cấu tạo nút.



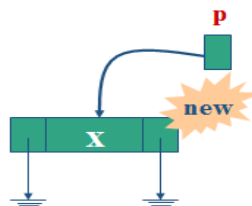
- Thành phần dữ liệu (**Data**): Lưu trữ các thông tin về bản thân phần tử.
- Thành phần mối liên kết (**Link**): Lưu trữ địa chỉ của 2 nút con bên trái (**Left**) và nút con bên phải (**Right**).

9.2.1.2. Khai báo nút.

```

typedef int ItemType; //Định nghĩa kiểu dữ liệu của một phần tử
struct TNode
{
    //Định nghĩa kiểu dữ liệu cho 1 nút của cây nhị phân Là TNode
    ItemType Info;
    TNode* Left;
    TNode* Right;
};
struct BTree
{
    //Định nghĩa kiểu dữ liệu cho cây nhị phân (Cây NPTK)
    TNode* Root;
};
    
```

9.2.1.3. Tạo nút chứa giá trị x.



```

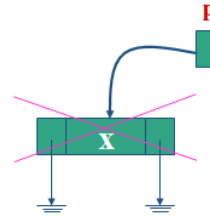
TNode* CreateTNode(ItemType x)
{
    TNode* p = new TNode;
    if(p == NULL)
    {
        printf("Không đủ bộ nhớ để cấp phát nút mới!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Left = NULL;
    p->Right = NULL;
    return p;
}
    
```

9.2.1.4. Xuất nội dung của nút.

```
void ShowTNode(TNode* p)
{
    printf("%4d", p->Info);
}
```

9.2.1.5. Hủy nút.

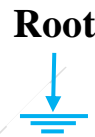
```
void DeleteTNode(TNode* &p)
{
    if(p == NULL) return;
    p->Left = NULL;
    p->Right = NULL;
    delete p;
    p = NULL;
}
```



9.2.2. Các thao tác trên cây nhị phân tìm kiếm.

9.2.2.1. Khởi tạo cây.

```
void InitBTree(BTree &bt)
{
    //Initialize BTree
    bt.Root = NULL;
}
```



9.2.2.2. Thêm nút p làm nút con bên trái nút T.

```
int InsertTNodeLeft(TNode* &T, TNode* p)
{
    if(T == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    if(T->Left != NULL) return 0; //Đã tồn tại nút con trái của T
    T->Left = p;
    return 1; //Thực hiện thành công
}
```

9.2.2.3. Thêm nút p làm nút con bên phải nút T.

```
int InsertTNodeRight(TNode* &T, TNode* p)
{
    if(T == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    if(T->Right != NULL) return 0; //Đã tồn tại nút con phải của T
    T->Right = p;
    return 1; //Thực hiện thành công
}
```

9.2.2.4. Thêm nút p vào cây.

```
int InsertTNode(TNode* &root, TNode* p)
{
    if(p == NULL)
        return 0; //Thực hiện không thành công
    if(root == NULL) //Cây rỗng, nên thêm vào gốc
    {
```

```

    root = p;
    return 1; //Thực hiện thành công
}
if(root→Info == p→Info)
    return 0; //Bị trùng nút
if(p→Info < root→Info)
    InsertTNode(root→Left, p); //Them ben trai
else
    InsertTNode(root→Right, p); //Them ben phai
return 1; //Thực hiện thành công
}

```

9.2.2.5. Tạo cây từ mảng a có sẵn dữ liệu.

```

void CreateBTreeFromArray(BTree &bt, ItemType a[], int n)
{ //Ham tao cay NPTK tu mang a
    InitBTree(bt);
    for(int i = 0; i < n; i++)
    {
        TNode* p = CreateTNode(a[i]);
        InsertTNode(bt.Root, p);
    }
}

```

9.2.2.6. Tạo cây tự động chứa n số nguyên (n > 0).

```

void CreateBTree_Automatic(BTree &bt)
{
    int n;
    ItemType x;
    printf("Cho biet so nut cua cay: ");
    scanf("%d", &n);
    InitBTree(bt);
    srand((unsigned)time(NULL)); //Tạo số mới sau mỗi lần thực hiện
    for(int i = 1; i <= n; i++)
    {
        x = (rand()%199) - 99; //Tạo 1 số ngẫu nhiên trong [-99, 99]
        TNode* p = CreateTNode(x);
        InsertTNode(bt.Root, p);
    }
}

```

9.2.2.7. Duyệt cây theo NLR.

```

void NLR(TNode* root)
{
    if(root == NULL) return;
    printf("%4d", root→Info);
    NLR(root→Left);
    NLR(root→Right);
}

```

9.2.2.8. Duyệt cây theo LNR.

```

void LNR(TNode* root)
{

```

```
    if(root == NULL) return;
    LNR(root→Left);
    printf("%4d", root→Info);
    LNR(root→Right);
}
```

9.2.2.9. Duyệt cây theo LRN.

```
void LRN(TNode* root)
{
    if(root == NULL) return;
    LRN(root→Left);
    LRN(root→Right);
    printf("%4d", root→Info);
}
```

9.2.2.10. Tìm kiếm nút chứa giá trị x.

Cách 1: *Viết dạng đệ quy*

```
TNode* FindTNode(TNode* root, ItemType x)
{
    if(root == NULL) return NULL;
    if(root→Info == x)
        return root; //tìm được khóa thì dừng
    else if(root→Info > x)
        FindTNode(root→Left, x); //tìm x bên nhánh trái
    else
        FindTNode(root→Right, x); //tìm x bên nhánh phải
}
```

Cách 2: *Viết dạng không đệ quy*

```
TNode* FindTNode(TNode* root, ItemType x)
{
    TNode* p = root;
    while(p != NULL)
    {
        if(p→Info == x)
            return p;
        else if(p→Info > x)
            p = p→Left;
        else
            p = p→Right;
    }
    return NULL;
}
```

10.3. Bài tập ở lớp.

1. Cho cây nhị phân tìm kiếm mà mỗi nút là 1 số nguyên. Hãy viết chương trình để thực hiện những chức năng sau:
 - a. Tạo cây NPTK bằng 3 cách (**Cách 1:** Cho trước 1 mảng **a** có **n** phần tử, hãy tạo một cây NPTK có **n** nút, mỗi nút lưu 1 phần tử của mảng. **Cách 2:** Nhập liệu từ bàn phím. **Cách 3:** Tạo ngẫu nhiên tự động).
 - b. Duyệt cây NPTK bằng 6 cách: NLR, LNR, LRN, NRL, RNL, RLN.
 - c. Thêm 1 nút có giá trị **x** vào cây.
 - d. Đếm số nút trên cây.
 - e. Kiểm tra (*tìm kiếm*) 1 nút có giá trị **x** trên cây hay không?
 - f. Cho biết các phần tử trên cây có lớn hơn **x** (**>x**) hay không?
2. Cho cây nhị phân tìm kiếm mà mỗi nút là 1 phân số. Hãy viết chương trình để thực hiện những chức năng sau:
 - a. Tạo cây NPTK bằng 2 cách (nhập liệu từ bàn phím, tạo ngẫu nhiên tự động).
 - b. Duyệt cây NPTK bằng 6 cách: NLR, LNR, LRN, NRL, RNL, RLN.
 - c. Thêm 1 nút là phân số **p** làm con trái của nút **T**.
 - d. Thêm 1 nút là phân số **p** làm con phải của nút **T**.
 - e. Thêm 1 nút là phân số **p** vào cây.
 - f. Đếm số lượng những phân số lớn hơn 1.
 - g. Tối giản tất cả các nút (*phân số*) của cây.
 - h. Tìm kiếm trên cây có nút nào có giá trị bằng với phân số **x** hay không?

10.4. Bài tập ở nhà.

3. Ứng dụng cây nhị phân (*cây biểu thức*) để viết chương trình thực hiện việc biểu diễn và tính giá trị của một biểu thức toán học.

Bài tập mẫu hướng dẫn thực hành

Bài 1: Ứng dụng Cây NPTK để viết chương trình quản lý các số nguyên?

Bước 1: Tạo một **Project** mới → đặt tên: CayNPTK_SoNguyen_<Tên sinh viên>

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình.

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

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

Bước 3: Khai báo cấu trúc dữ liệu cho chương trình.

```
//=====
```

```
typedef int ItemType; //Định nghĩa kiểu dữ liệu của một phần tử
```

```
struct TNode
```

```
{//Định nghĩa kiểu dữ liệu cho 1 nút của cây nhị phân Là TNode
```

```
ItemType Info;
TNode* Left;
TNode* Right;
};
struct BSTree
{ //Định nghĩa kiểu dữ liệu cho cây nhị phân (Cây NPTK)
    TNode* Root;
};
```

Bước 4: Viết các hàm cần thiết cho chương trình như sau:

```
//=====
TNode* CreateTNode(ItemType x)
{
    TNode* p = new TNode;
    if(p == NULL)
    {
        printf("\nKhong du bo nho de cap phat nut moi!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Left = NULL;
    p->Right = NULL;
    return p;
}
//=====
void InitBSTree(BSTree &bt)
{ //Initialize BSTree
    bt.Root = NULL;
}
//=====
int InsertTNode(TNode* &root, TNode* p)
{ //Ham chen 1 nut vao cay NPTK
    if(p == NULL)
        return 0; //Them không thành công
    if(root == NULL) //Cay đang rỗng
    {
        root = p;
        return 1; //Thêm thành công
    }
    if(root->Info == p->Info)
        return 0; //Bi trùng nên không thêm nữa
    if(p->Info < root->Info)
        InsertTNode(root->Left, p);
    else
        InsertTNode(root->Right, p);
    return 1; //Thêm thành công
}
//=====
void CreateBSTreeFromArray(BSTree &bt, ItemType a[], int na)
{ //Ham tao cay NPTK tu mang a
    InitBSTree(bt);
    for(int i = 0; i < na; i++)
    {
        TNode* p = CreateTNode(a[i]);
        InsertTNode(bt.Root, p);
    }
}
```



```
//=====
void ShowTNode(TNode* p)
{
    printf("%4d", p->Info);
}
//=====
void NLR(TNode* root)
{
    //Ham duyet cay theo thu tu NLR
    if(root == NULL) return;
    ShowTNode(root);
    NLR(root->Left);
    NLR(root->Right);
}
//=====
TNode* FindTNode(TNode* root, ItemType x)
{
    if(root == NULL) return NULL;
    if(root->Info == x) return root;
    else if(root->Info > x)
        return FindTNode(root->Left, x);
    else
        return FindTNode(root->Right, x);
}
//=====
```

Bước 5: Viết hàm main để thực thi chương trình.

```
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                MENU                *");
    printf("\n*-----*");
    printf("\n* 1. Tao cay NPTK tu mang                *");
    printf("\n* 2. Duyệt cây theo NLR                  *");
    printf("\n* 3. Tìm kiếm một nút bất kỳ              *");
    printf("\n* 0. Thoát chương trình                  *");
    printf("\n*****");
}
//=====
void Process()
{
    int SelectFunction;
    BSTree bt;
    TNode *p;
    ItemType x;
    ItemType a[] = {10, 8, 9, 5, 4, 17, 15, 16, 20, 11, 12, 13, 19, 7, 25, 6};
    int n = 16;
    InitBSTree(bt);
    do
    {
        ShowMenu();
        printf("\nBan hay chon mot chuc nang (tu 0 -> 3): ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            case 1:
                CreateBSTreeFromArray(bt, a, n);
                break;
        }
    }
}
```

```
case 2:
    printf("\nXem noi dung cua cay bang phep duyệt NLR: ");
    NLR(bt);
    break;
case 3:
    printf("\nCho biet gia tri nut muon tim: ");
    scanf("%d", &x);
    p = FindTNode(bt.Root, x);
    if(p != NULL)
        printf("\nNut muon tim %d co trong cay.", x);
    else
        printf("\nNut muon tim %d khong co trong cay.", x);
    break;
}
} while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}
```

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 10: CÂY NHỊ PHÂN - CÂY NHỊ PHÂN TÌM KIẾM (tt)	
--	--	--

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được cấu trúc dữ liệu cây vào từng bài toán cụ thể.
- Làm được các bài tập áp dụng cây.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**10.1. Các thao tác trên cây nhị phân tìm kiếm.****10.1.1. Kiểm tra nút T có phải là nút lá hay không.**

```

int IsLeaf(TNode* T)
{
    //Tra ve: 1 neu nut la, 0 neu nguoc lai
    if(T == NULL)
        return 0;
    if(T->Left != NULL || T->Right != NULL)
        return 0;
    return 1;
}

```

10.1.2. Xóa nút con bên trái của nút T.

```

int DeleteTNodeLeft(TNode* T, ItemType &x)
{
    //Nút con trái của T phải là nút lá
    if(T == NULL)
        return 0; //Xóa không thành công
    TNode* p = T->Left;
    if(p == NULL)
        return 0; //Xóa không thành công
    if(p->Left != NULL || p->Right != NULL) //p không phải nút lá

```

```
        return 0; //Xóa không thành công
    T→Left = NULL;
    x = p→Info; //Lấy thông tin của nút bị hủy
    delete p;
    return 1; //Xóa thành công
}
```

10.1.3. Xóa nút con bên phải của nút T.

```
int DeleteTNodeRight(TNode* T, ItemType &x)
{ //Nút con phải của T phải là nút Lá
    if(T == NULL)
        return 0; //Xóa không thành công
    TNode* p = T→Right;
    if(p == NULL)
        return 0; //Xóa không thành công
    if(p→Left != NULL || p→Right != NULL) //p không phải nút Lá
        return 0; //Xóa không thành công
    T→Right = NULL;
    x = p→Info; //Lấy thông tin của nút bị hủy
    delete p;
    return 1; //Xóa thành công
}
```

10.1.4. Xóa nút có giá trị x.

```
TNode* FindTNodeReplace(TNode* &p)
{ //Hàm tìm nút q thế mạng cho nút p, f là nút cha của nút q.
    TNode* f = p;
    TNode* q = p→Right;
    while(q→Left != NULL)
    {
        f = q; //Lưu nút cha của q
        q = q→Left; //q qua bên trái
    }
    p→Info = q→Info; //Tìm được phần tử thế mạng cho p là q
    if(f == p) //Nếu cha của q là p
        f→Right = q→Right;
    else
        f→Left = q→Right;
    return q; //trả về nút q là nút thế mạng cho p
}

int DeleteTNodeX(TNode* &root, ItemType x)
{ //Hàm xóa nút có giá trị là x
    if(root == NULL) //Khi cây rỗng
        return 0; //Xóa không thành công
    if(root→Info > x)
        return DeleteTNodeX(root→Left, x);
    else if(root→Info < x)
        return DeleteTNodeX(root→Right, x);
    else
    { //root→Info = x, tìm nút thế mạng cho root
        TNode* p = root;
```

```

    if(root→Left == NULL) //khi cây con không có nhánh trái
    {
        root = root→Right;
        delete p;
    }
    else if(root→Right == NULL) //khi cây con không có nhánh phải
    {
        root = root→Left;
        delete p;
    }
    else
    {//khi cây con có cả 2 nhánh, chọn min của nhánh phải để thế mạng
        TNode* q = FindTNodeReplace(p);
        delete q; //Xóa nút q là nút thế mạng cho p
    }
    return 1; //Xóa thành công
}
}

```

10.1.5. Xóa toàn bộ cây.

```

void DeleteTree(TNode* &root)
{
    if(root == NULL) return;
    DeleteTree(root→Left); //đệ quy xóa cây con trái
    DeleteTree(root→Right); //đệ quy xóa cây con phải
    delete root; //hoặc có thể dùng lệnh DeleteTNode(root);
}

```

10.1.6. Đếm số nút của cây.

```

int CountTNode(TNode* root)
{ //Ham dem so nut hien co trong cay
    if(!root) return 0;
    int n1 = CountTNode(root→Left); //đệ quy trái
    int nr = CountTNode(root→Right); //đệ quy phải
    return (1 + n1 + nr);
}

```

10.1.7. Đếm số nút lá của cây.

```

int CountTNodeIsLeaf(TNode* root)
{ //Ham dem so nut la hien cua cay
    if(!root) return 0;
    if(!root→Left && !root→Right) return 1;
    int n1 = CountTNodeIsLeaf(root→Left); //đệ quy trái
    int nr = CountTNodeIsLeaf(root→Right); //đệ quy phải
    return (n1 + nr);
}

```

10.1.8. Đếm số nút không phải nút lá.

```

int CountTNodeIsNoLeaf(TNode* root)
{ //Hàm đếm số nút không phải là nút lá của cây
    if(!root) return 0;
    if(!root→Left && !root→Right) return 0;

```

```
int n1 = CountTNodeIsNoLeaf(root→Left); //đệ quy trái
int nr = CountTNodeIsNoLeaf(root→Right); //đệ quy phải
return (1 + n1 + nr);
}
```

10.1.9. Đếm số nút trung gian (không phải nút lá cũng không phải nút gốc).

```
int CountTNodeMedium(TNode* root)
{ //Hàm đếm số nút không phải là nút Lá của cây
  int count1 = CountTNode(root);
  if(count1 <= 2) return 0;
  int count2 = CountTNodeIsLeaf(root);
  return (count1 - count2 - 1);
}
```

Hoặc:

```
int CountTNodeMedium(TNode* root)
{ //Hàm đếm số nút không phải là nút Lá của cây
  int n = CountTNodeIsNoLeaf(root);
  if(n == 0) return 0; //không có nút nào không là nút Lá
  return (n - 1); //trừ nút gốc
}
```

10.1.10. Đếm số nút có đúng 2 nút con của cây.

```
int CountTNodeHaveTwoChild(TNode* root)
{ //Hàm đếm số nút có 2 con
  if(!root) return 0;
  if(!root→Left || !root→Right) return 0;
  int n1 = CountTNodeHaveTwoChild(root→Left); //đệ quy trái
  int nr = CountTNodeHaveTwoChild(root→Right); //đệ quy phải
  return (1 + n1 + nr);
}
```

Hoặc:

```
int CountTNodeHaveTwoChild(TNode* root)
{ //Hàm đếm số nút có 2 con
  if(!root) return 0;
  int n1 = CountTNodeHaveTwoChild(root→Left); //đệ quy trái
  int nr = CountTNodeHaveTwoChild(root→Right); //đệ quy phải
  if(root→Left && root→Right)
    return (1 + n1 + nr);
  return (n1 + nr);
}
```

10.1.11. Đếm số nút chỉ có 1 nút con của cây.

```
int CountTNodeHaveOneChild(TNode* root)
{ //Hàm đếm số nút chỉ có 1 con
  if(!root) return 0;
  int n1 = CountTNodeHaveOneChild(root→Left); //đệ quy trái
  int nr = CountTNodeHaveOneChild(root→Right); //đệ quy phải
  if((root→Left && !root→Right) || (!root→Left && root→Right))
    return (1 + n1 + nr);
  return (n1 + nr);
}
```

10.1.12. Đếm số nút có con là nút lá của cây.

```

int CountTNodeHaveChildIsLeaf(TNode* root)
{ //Ham dem so nut co con la nut la
  if(!root) return 0;
  int nl = CountTNodeHaveChildIsLeaf(root→Left); //đệ quy trái
  int nr = CountTNodeHaveChildIsLeaf(root→Right); //đệ quy phải
  if(IsLeaf(root→Left) || IsLeaf(root→Right))
    return (1 + nl + nr);
  return (nl + nr);
}

```

10.1.13. Tính chiều cao của cây.

```

int HighTree(TNode* root)
{ //Ham tinh chieu cao cua cay
  if(!root) return 0;
  int hl = HighTree(root→Left); //đệ quy trái
  int hr = HighTree(root→Right); //đệ quy phải
  if(hl > hr)
    return (1 + hl);
  else
    return (1 + hr);
}

```

10.1.14. Tìm nút có giá trị lớn nhất trong cây nhị phân tìm kiếm.

```

int TNodeMax(TNode* root)
{ //Ham tim nut co gia tri lon nhat cua cay
  TNode* p = root;
  while(p→Right != NULL)
    p = p→Right;
  return (p→Info);
}

```

10.1.15. Tính tổng giá trị các nút của cây.

```

int SumTNode(TNode* root)
{ //Ham tinh tong gia tri cac nut hien co trong cay
  if(!root) return 0;
  int suml = SumTNode(root→Left); //đệ quy trái
  int sumr = SumTNode(root→Right); //đệ quy phải
  return (root→Info + suml + sumr);
}

```

10.1.16. Xuất ra màn hình nội dung các nút ở mức thứ k của cây.

```

void ShowTNodeOfLevelK(TNode *root, int k)
{
  if(!root) return;
  if(k == 0) //đến mức cần tìm
    printf("%4d", root→Info);
  k--; //Mức k giảm dần về 0
  ShowTNodeOfLevelK(root→Left, k); //đệ quy trái
  ShowTNodeOfLevelK(root→Right, k); //đệ quy phải
}

```

10.1.17. Xuất ra màn hình nội dung các nút lá ở mức thứ k của cây.

```
void ShowTNodeIsLeafOfLevelK(TNode *root, int k)
{
    if(!root) return;
    if(k == 0 && !root->Left && !root->Right) //đến mức cần tìm
        printf("%4d", root->Info);
    k--; //Mức k giảm dần về 0
    ShowTNodeIsLeafOfLevelK(root->Left, k); //đệ quy trái
    ShowTNodeIsLeafOfLevelK(root->Right, k); //đệ quy phải
}
```

10.1.18. Đếm số lượng nút ở mức thứ k của cây.

```
int CountTNodeOfLevelK(TNode* root, int k)
{
    if(!root) return 0;
    if(k == 0) //đến mức cần tìm
        return 1;
    k--; //Mức k giảm dần về 0
    int nl = CountTNodeOfLevelK(root->Left, k); //đệ quy trái
    int nr = CountTNodeOfLevelK(root->Right, k); //đệ quy phải
    return (nl + nr);
}
```

10.1.19. Đếm số lượng nút lá ở mức thứ k của cây.

```
int CountTNodeIsLeafOfLevelK(TNode *root, int k)
{
    if(!root) return 0;
    if(k == 0 && !root->Left && !root->Right) //đến mức cần tìm
        return 1;
    k--; //Mức k giảm dần về 0
    int nl = CountTNodeIsLeafOfLevelK(root->Left, k); //đệ quy trái
    int nr = CountTNodeIsLeafOfLevelK(root->Right, k); //đệ quy phải
    return (nl + nr);
}
```

10.1.20. Tính tổng giá trị các nút lá ở mức thứ k của cây.

```
int SumTNodeIsLeafOfLevelK(TNode *root, int k)
{
    if(!root) return 0;
    if(k == 0 && !root->Left && !root->Right) //đến mức cần tìm
        return (root->Info);
    k--; //Mức k giảm dần về 0
    int suml = SumTNodeIsLeafOfLevelK(root->Left, k); //đệ quy trái
    int sumr = SumTNodeIsLeafOfLevelK(root->Right, k); //đệ quy phải
    return (suml + sumr);
}
```

10.1.21. Nút có khoảng cách về giá trị gần nhất với phần tử x trong cây.

```
int minDistance(TNode* root, ItemType x)
{
    if(!root)
```



```

        return -1;
    int min = root→Info;
    int mindis = abs(x - min);
    while(root != NULL)
    {
        if(root→Info == x)
            return x;
        if(mindis > abs(x - root→Info))
        {
            min = root→Info;
            mindis = abs(x - min);
        }
        if(x > root→Info)
            root = root→Right;
        else
            root = root→Left;
    }
    return min;
}

```

10.1.22. Nút có khoảng cách về giá trị xa nhất với phần tử x trong cây.

```

int maxDistance(TNode* root, ItemType x)
{
    if(!root)
        return -1;
    TNode* minLeft = root;
    while(minLeft→Left != NULL) //Tìm nút trái nhất
        minLeft = minLeft→Left;

    TNode* maxRight = root;
    while(maxRight→Right != NULL) //Tìm nút phải nhất
        maxRight = maxRight→Right;

    int dis1 = abs(x - minLeft→Info);
    int dis2 = abs(x - maxRight→Info);
    if(dis1 > dis2)
        return minLeft→Info;
    else
        return maxRight→Info;
}

```

10.2. Bài tập ở lớp.

1. Cho cây nhị phân tìm kiếm như ở **Bài tập 1 (BÀI THỰC HÀNH 9)**. Hãy bổ sung các hàm thực hiện những chức năng sau:
 - a. Xóa nút con trái của 1 nút T, xóa nút con phải của 1 nút T.
 - b. Xóa nút có giá trị x trên cây.
 - c. Xuất các phần tử theo chiều giảm dần.
 - d. Đếm số giá trị lớn hơn x, nhỏ hơn x, có giá trị trong đoạn [x, y].
 - e. Đếm số nút, số nút lá của cây.

- f. Đếm số nút có đúng 2 nút con, số nút chỉ có 1 nút con, số nút có con là nút lá của cây.
 - g. Tính chiều cao của cây.
 - h. Tìm nút có giá trị lớn nhất, nhỏ nhất của cây.
 - i. Xuất ra nội dung các nút ở mức k.
 - j. Xuất ra nội dung các nút lá ở mức k.
 - k. Đếm số nút ở mức k.
 - l. Đếm số nút lá ở mức k.
 - m. Tìm phần tử có khoảng cách về giá trị gần nhất với phần tử **x** trong cây (nếu cây rỗng trả về -1).
 - n. Tìm phần tử có khoảng cách về giá trị xa nhất với phần tử **x** trong cây (nếu cây rỗng trả về -1).
 - o. Tính tổng giá trị các nút của cây (dùng đệ quy / không dùng đệ quy).
 - p. Tính tổng giá trị các nút dương, giá trị các nút âm trên cây.
 - q. Tính tổng giá trị các nút là số nguyên tố, là số chính phương, là số hoàn thiện, là số thịnh vượng, là số yếu của cây.
 - r. Ứng dụng cơ chế **Stack**, viết hàm duyệt cây theo NLR.
 - s. Xóa toàn bộ cây.
2. Cho cây nhị phân tìm kiếm như ở **Bài tập 2 (BÀI THỰC HÀNH 9)**. Hãy bổ sung những chức năng sau:
- a. Tìm kiếm 1 phân số **x** trên cây hay không?
 - b. Xóa một phân số **x** trên cây.
 - c. Xóa những phân số >2 (xét theo giá trị).
 - d. Xóa những phân số có mẫu số là số nguyên tố.
 - e. Xóa toàn bộ danh sách.

10.3. Bài tập ở nhà.

3. Tiếp theo **bài 1**. Hãy viết các hàm thực hiện các chức năng sau:
- a. Viết hàm xuất các số hoàn thiện trong cây.
 - b. Viết hàm xuất tất cả các nút trên tầng thứ k của cây. (*)
 - c. Viết hàm xuất tất cả các nút trên cây theo thứ tự từ tầng 0 đến tầng h-1 của cây (với h là chiều cao của cây). (*)
 - d. Đếm số lượng nút lá mà thông tin tại nút đó là giá trị chẵn.
 - e. Đếm số lượng nút có đúng 1 con mà thông tin tại nút đó là số nguyên tố.
 - f. Đếm số lượng nút có đúng 2 con mà thông tin tại nút đó là số chính phương.
 - g. Đếm số lượng nút trên tầng thứ k của cây.
 - h. Đếm số lượng nút nằm ở tầng thấp hơn tầng thứ k của cây.

- i. Đếm số lượng nút nằm ở tầng cao hơn tầng thứ k của cây.
- j. Tính tổng các nút trong cây.
- k. Tính tổng các nút lá trong cây.
- l. Tính tổng các nút có đúng một con.
- m. Tính tổng các nút có đúng hai con.
- n. Tính tổng các nút lẻ.
- o. Tính tổng các nút lá mà thông tin tại nút đó là giá trị chẵn.
- p. Tính tổng các nút có đúng 1 con mà thông tin tại nút đó là số nguyên tố.
- q. Tính tổng các nút có đúng 2 con mà thông tin tại nút đó là số chính phương.
- r. Kiểm tra cây nhị phân T có phải là "cây nhị phân tìm kiếm" hay không?
- s. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng" hay không?
- t. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng hoàn toàn" hay không?

Bài tập mẫu hướng dẫn thực hành

Bài 1: Ứng dụng Cây NPTK để viết chương trình quản lý các số nguyên?

Tiếp theo bài thực hành 10. Bổ sung một số thao tác sau:

```
//=====
int HighTree(TNode* root)
{ //Ham tinh chieu cao cua cay
    if(!root) return 0;
    int hl = HighTree(root->Left);
    int hr = HighTree(root->Right);
    if(hl < hr)
        return 1 + hr;
    else
        return 1 + hl;
}
//=====
int DemNutLaMucK(TNode *root, int k)
{
    if(!root) //kiem tra xem TNode có bằng NULL, nếu bằng thì return 0
        return 0;
    if(k== 0 && !root->Left && !root->Right) //đến mức cần tìm
        return 1;
    k--; //Mức k giảm dần về 0
    int nl = DemNutLaMucK(root->Left,k); //duyet đệ quy trái
    int nr = DemNutLaMucK(root->Right,k); //duyet đệ quy phải
    return (nl + nr);
}
//=====
```

Bước 5: Viết hàm main để thực thi chương trình.

```
void ShowMenu()
```

```

{
    printf("\n*****");
    printf("\n*                      MENU                      *");
    printf("\n*-----*");
    printf("\n* 1. Tao cay NPTK tu mang                *");
    printf("\n* 2. Duyệt cây theo NLR                  *");
    printf("\n* 3. Tìm kiếm một nút bất kỳ            *");
    printf("\n* 4. Tính chiều cao của cây              *");
    printf("\n* 5. Đếm số nút lá ở mức k              *");
    printf("\n* 0. Thoát chương trình                  *");
    printf("\n*****");
}
//=====
void Process()
{
    ...
    int k, high, count, sum, min, max;
    do
    {
        ShowMenu();
        printf("\nBạn hãy chọn một chức năng (từ 0 -> 5): ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            ...
            case 4:
                high = HighTree(bt.Root);
                printf("\nChiều cao của cây là: %d", high);
                break;
            case 5:
                printf("\nCho biết cấp muốn đếm nội dung nút là: ");
                scanf("%d", &k);
                count = DemNutLaMucK(bt.Root, k);
                printf("\nSố lượng nút lá ở mức %d là: %d", k, count);
                break;
        }
    } while(SelectFunction != 0);
}
//=====
void main()
{
    Process();
}

```

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 11: CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG (CÂY AVL)	
--	---	---

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được cấu trúc dữ liệu cây **AVL** vào từng bài toán cụ thể.
- Làm được các bài tập áp dụng cây **AVL**.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**11.1. Cây nhị phân tìm kiếm cân bằng (Cây AVL).****11.1.1. Các thao tác trên cây AVL.****11.1.1.1. Khai báo thư viện và tạo cấu trúc dữ liệu cây.**

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
//=====
#define LH -1    //Cây con trái cao hơn (lệch trái)
#define EH 0    //Cây con trái bằng cây con phải (cân bằng)
#define RH 1    //Cây con phải cao hơn (lệch phải)
//=====
```

Lưu ý:

Cách định nghĩa một kiểu dữ liệu mới

```
typedef <Kiểu dữ liệu cơ bản hoặc kiểu có cấu trúc> ItemType;
```

Ví dụ:

```
typedef int ItemType; //Khai báo kiểu dữ liệu mới có tên là ItemType
```

```
typedef int ItemType;
struct AVLNode
{ //Định nghĩa kiểu dữ liệu cho 1 nút của cây AVL là AVLNode
    int balFactor; //chỉ số cân bằng
    ItemType Info;
    AVLNode* Left;
    AVLNode* Right;
};
struct AVLTree
{ //Định nghĩa kiểu dữ liệu cho cây AVL
    AVLNode* Root;
};
```

11.1.1.2. Cấp phát một nút mới cho cây AVL.

```
AVLNode* CreateAVLNode(ItemType x)
{
    AVLNode* p = new AVLNode;
    if(p == NULL)
    {
        printf("\nKhong du bo nho de cap phat nut moi!");
        getch();
        return NULL;
    }
    p->balFactor = 0;
    p->Info = x;
    p->Left = NULL;
    p->Right = NULL;
    return p;
}
```

11.1.1.3. Hủy vùng nhớ đã cấp phát cho một nút trên cây AVL.

```
void DeleteAVLNode(AVLNode* &p)
{ //Ham xoa 1 nut ra khoi bo nho
    p->Left = NULL;
    p->Right = NULL;
    delete p;
    p = NULL;
}
```

11.1.1.4. Khởi tạo cây AVL.

```
void InitAVLTree(AVLTree &avl)
{ //Initialize AVLTree
    avl.Root = NULL;
}
```

11.1.1.5. Cân bằng lại cây trong trường hợp lệch LL (Left - Left).

```
void RotateLL(AVLNode* &T)
{ //Xoay Left - Left
    AVLNode* T1 = T->Left;
    T->Left = T1->Right;
    T1->Right = T;
    switch(T1->balFactor)
```

```

{
    case LH: //cây T1 lệch trái
        T→balFactor = EH;
        T1→balFactor = EH;
        break;
    case EH: //cây T1 cân bằng
        T→balFactor = LH;
        T1→balFactor = RH;
        break;
}
T = T1;
}

```

11.1.1.6. Cân bằng lại cây trong trường hợp lệch LR (Left - Right).

```

void RotateLR(AVLNode* &T)
{ //Xoay Left - Right
    AVLNode* T1 = T→Left;
    AVLNode* T2 = T1→Right;
    T1→Right = T2→Left;
    T2→Left = T1;
    T→Left = T2→Right;
    T2→Right = T;
    switch(T2→balFactor)
    {
        case LH:
            T→balFactor = RH;
            T1→balFactor = EH;
            break;
        case EH:
            T→balFactor = EH;
            T1→balFactor = EH;
            break;
        case RH:
            T→balFactor = EH;
            T1→balFactor = LH;
            break;
    }
    T2→balFactor = EH;
    T = T2;
}

```

11.1.1.7. Cân bằng lại cây trong trường hợp lệch trái.

```

int BalanceLeft(AVLNode* &T)
{ //Khi cây T lệch bên trái cần cân bằng lại
    AVLNode* T1 = T→Left;
    switch(T1→balFactor)
    {
        case LH:
            RotateLL(T);
            return 2;
        case EH:

```

```
        RotateLL(T);  
        return 1;  
    case RH:  
        RotateLR(T);  
        return 2;  
    }  
    return 0; //Trường balance bị sai  
}
```

11.1.1.8. Cân bằng lại cây trong trường hợp lệch RR (Right - Right).

```
void RotateRR(AVLNode* &T)  
{ //Xoay Right - Right  
    AVLNode* T1 = T→Right;  
    T→Right = T1→Left;  
    T1→Left = T;  
    switch(T1→balFactor)  
    {  
        case RH: //cây T1 lệch phải  
            T→balFactor = EH;  
            T1→balFactor = EH;  
            break;  
        case EH: //cây T1 cân bằng  
            T→balFactor = RH;  
            T1→balFactor = LH;  
            break;  
    }  
    T = T1;  
}
```

11.1.1.9. Cân bằng lại cây trong trường hợp lệch RL (Right - Left).

```
void RotateRL(AVLNode* &T)  
{ //Xoay Right - Left  
    AVLNode* T1 = T→Right;  
    AVLNode* T2 = T1→Left;  
    T1→Left = T2→Right;  
    T2→Right = T1;  
    T→Right = T2→Left;  
    T2→Left = T;  
    switch(T2→balFactor)  
    {  
        case RH:  
            T→balFactor = LH;  
            T1→balFactor = EH;  
            break;  
        case EH:  
            T→balFactor = EH;  
            T1→balFactor = EH;  
            break;  
        case LH:  
            T→balFactor = EH;  
            T1→balFactor = RH;  
    }  
}
```



```

        break;
    }
    T2→balFactor = EH;
    T = T2;
}

```

11.1.1.10. Cân bằng lại cây trong trường hợp lệch phải.

```

int BalanceRight(AVLNode* &T)
{ //Khi cây T lệch bên phải cần cân bằng lại
  AVLNode* T1 = T→Right;
  switch(T1→balFactor)
  {
    case LH:
      RotateRL(T);
      return 2;
    case EH:
      RotateRR(T);
      return 1;
    case RH:
      RotateRR(T);
      return 2;
  }
  return 0;
}

```

11.1.1.11. Thêm nút p có giá trị x.

```

int InsertAVLNode(AVLNode* &T, AVLNode* p)
{
  if(p == NULL)
    return -1; //Nút mới p muốn thêm không tồn tại
  if(T == NULL)
  { //Cây rỗng nên thêm nút mới p vào gốc
    T = p;
    return 2; //Thực hiện thành công nút mới p
  }
  //Cây không rỗng
  int Result;
  if(T→Info == p→Info)
    return 0; //Không thêm được vì tồn tại nút có giá trị bằng x
  if(T→Info > p→Info)
  {
    Result = InsertAVLNode(T→Left, p);
    if(Result < 2)
      return Result;
    switch(T→balFactor)
    {
      case RH:
        T→balFactor = EH;
        return 1;
      case EH:
        T→balFactor = LH;
        return 2;
    }
  }
}

```

```
        case LH:
            BalanceLeft(T);
            return 1;
    }
}
else
{
    Result = InsertAVLNode(T→Right, p);
    if(Result < 2)
        return Result;
    switch(T→balFactor)
    {
        case LH:
            T→balFactor = EH;
            return 1;
        case EH:
            T→balFactor = RH;
            return 2;
        case RH:
            BalanceRight(T);
            return 1;
    }
}
}
```

11.1.1.12. Xóa nút có giá trị x.

```
int SearchStandFor(AVLNode* &p, AVLNode* &q)
{
    int Result;
    if(q→Left)
    {
        Result = SearchStandFor(p, q→Left);
        if(Result < 2)
            return Result;
        switch(q→balFactor)
        {
            case LH:
                q→balFactor = EH;
                return 2;
            case EH:
                q→balFactor = RH;
                return 1;
            case RH:
                return BalanceRight(q);
        }
    }
    else
    {
        p→Info = q→Info;
        p = q;
        q = q→Right;
    }
}
```

```

    }
    return 2;
}
//=====
int DeleteAVLNode(AVLNode* &T, ItemType x)
{ //Xóa nút có Info bằng với x
    int Result;
    if(T == NULL)
        return 0; //Không thực hiện được
    if(T->Info > x)
    {
        Result = DeleteAVLNode(T->Left, x);
        if(Result < 2)
            return Result;
        switch(T->balFactor)
        {
            case LH:
                T->balFactor = EH;
                return 2;
            case EH:
                T->balFactor = RH;
                return 1;
            case RH:
                return BalanceRight(T);
        }
    }
    else if(T->Info < x)
    {
        Result = DeleteAVLNode(T->Right, x);
        if(Result < 2)
            return Result;
        switch(T->balFactor)
        {
            case RH:
                T->balFactor = EH;
                return 2;
            case EH:
                T->balFactor = LH;
                return 1;
            case LH:
                return BalanceLeft(T);
        }
    }
    else
    {
        AVLNode* p = T;
        if(T->Left == NULL)
        {
            T = T->Right;
            Result = 2;
        }
    }
}

```

```
else
{
    if(T→Right == NULL)
    {
        T = T→Left;
        Result = 2;
    }
    else
    {
        Result = SearchStandFor(p, T→Right);
        if(Result < 2)
            return Result;
        switch(T→balFactor)
        {
            case RH:
                T→balFactor = EH;
                return 2;
            case EH:
                T→balFactor = LH;
                return 1;
            case LH:
                return BalanceLeft(T);
        }
    }
}
delete p;
return Result;
}
return Result;
}
```

11.1.1.13. Tìm kiếm một nút có giá trị x trên cây.

```
AVLNode* FindAVLNode(AVLNode* root, ItemType x)
{
    if(!root) return NULL;
    if(root→Info == x) return root;
    else if(root→Info > x)
        return FindAVLNode(root→Left, x);
    else
        return FindAVLNode(root→Right, x);
}
```

11.1.1.14. Xóa toàn bộ cây.

```
int DeleteAVLTree(AVLNode* &root)
{
    //Ham xoa toan bo cay
    if(!root) return 0;
    DeleteAVLTree(root→Left); //Xóa con trái
    DeleteAVLTree(root→Right); //Xóa con phải
    DeleteAVLNode(root); //Xóa nút gốc
    return 1;
}
```

11.1.1.15. Duyệt cây theo cách LNR.

```

void LNR(AVLNode* root)
{
    //Ham duyet cay theo thu tu LNR
    if(!root) return;
    LNR(root→Left); //Duyệt sang nhánh trái
    printf("%4d", root→Info); //Show nội dung nút gốc
    LNR(root→Right); //Duyệt sang nhánh phải
}

```

11.1.1.16. Tạo cây AVL từ mảng a có n phần tử.

```

void CreateAVLTreeFromArray(AVLTree &avl, ItemType a[], int n)
{
    //Ham tao cay NPTK tu mang a
    InitAVLTree(avl);
    for(int i = 0; i < n; i++)
    {
        AVLNode* p = CreateAVLNode(a[i]);
        InsertAVLNode(avl.Root, p);
    }
}

```

Bài tập mẫu hướng dẫn thực hành

Bài 1: Ứng dụng kiến thức về cây AVL, viết chương trình cho phép xây dựng (tạo) cây cũng như những thao tác cơ bản như: duyệt cây, thêm nút, tìm kiếm, ... trên cây AVL mà giá trị của mỗi nút là số nguyên?

Bước 1: Tạo một **Project** mới → đặt tên: CayAVL_SoNguyen_<Tên sinh viên>

Bước 2: Khai báo thêm các thư viện cơ bản cho chương trình. (xem ở trên)

Bước 3: Khai báo cấu trúc dữ liệu cho chương trình. (xem ở trên)

Bước 4: Viết các hàm cần thiết cho chương trình. (xem ở trên)

Bước 5: Viết các hàm để thực hiện chương trình.

```

//=====
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                      MENU                      *");
    printf("\n*-----*");
    printf("\n* 1. Khoi tao cay nhi phan tim kiem can bang *");
    printf("\n* 2. Duyệt LNR *");
    printf("\n* 3. Them nut co gia tri bat ky vao cay *");
    printf("\n* 4. Xoa nut co gia tri bat ky vao cay *");
    printf("\n* 5. Chieu cao cua cay *");
    printf("\n* 6. Tim kiem nut co gia tri x tren cay *");
    printf("\n* 7. Xoa cay *");
    printf("\n* 0. Thoat chuong trinh *");
    printf("\n*****");
}
//=====
void main()

```

```
{
    int SelectFunction, kq, high;
    AVLTree avl;
    AVLNode* p;
    ItemType x;
    ItemType a[]={10, 8, 9, 5, 4, 17, 15, 16, 20, 11, 12, 13, 19, 7, 25, 6};
    int n = 16;
    InitAVLTree(avl);
    do
    {
        ShowMenu();
        printf("\nHay chon mot chuc nang cua chuong trinh: ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            case 1:
                CreateAVLTreeFromArray(avl, a, n);
                break;
            case 2:
                printf("\nNoi dung cua cay voi phep duyiet LNR: ");
                LNR(avl.Root);
                break;
            case 3:
                printf("Nhap gia tri bay ky cho nut muon them: ");
                scanf("%d", &x);
                p = CreateAVLNode(x);
                kq = InsertAVLNode(avl.Root, p);
                if(kq <= 0)
                    printf("\nKhong the them duoc nut co gia tri %d", x);
                else
                {
                    printf("\nNoi dung cua cay sau khi them nut %d la: ", x);
                    LNR(avl.Root);
                }
                break;
            case 4:
                printf("Nhap gia tri bay ky cho nut muon them: ");
                scanf("%d", &x);
                kq = DeleteAVLNode(avl.Root, x);
                if(kq == 0)
                    printf("\nKhong the xoa duoc nut co gia tri %d", x);
                else
                {
                    if(avl.Root == NULL)
                        printf("\nCay rong!");
                    else
                    {
                        printf("\nNoi dung cua cay sau khi xoa nut %d la: ", x);
                        LNR(avl.Root);
                    }
                }
                break;
            case 5:
                high = HighTree(avl.Root);
                printf("\nChieu cao cua cay la: %d", high);
                break;
            case 6:
```

```

printf("Nhap gia tri bay ky cho nut muon them: ");
scanf("%d", &x);
p = FindAVLNode(avl.Root, x);
if(p == NULL)
    printf("\nKhong tim thay nut co gia tri %d", x);
else
    printf("\nCo nut co gia tri %d tren cay", x);
    break;
case 7:
    kq = DeleteAVLTree(avl.Root);
    if(kq == 0)
        printf("\nKhong the xoa cay");
    else
        printf("\nDa xoa toan bo cay");
    break;
}
}while(SelectFunction != 0);
}

```

11.2. Bài tập ở lớp

1. Cho cây AVL mà mỗi nút chứa thông tin **phân số**. Hãy viết chương trình để thực hiện những chức năng sau:
 - a. Tạo cây AVL bằng 2 cách (nhập liệu từ bàn phím, tạo ngẫu nhiên tự động).
 - b. Duyệt cây AVL bằng 6 cách: NLR, LNR, LRN, NRL, RNL, RLN.
 - c. Thêm mới một phân số **x** vào cây.
 - d. Đếm số lượng những phân số lớn hơn 1.
 - e. Tối giản tất cả các nút (*phân số*) của cây.
 - f. Tìm kiếm trên cây có nút nào có giá trị bằng với phân số **x** hay không?
 - g. Liệt kê các nút có cùng mẫu số với phân số **x** (*x nhập từ bàn phím*).
 - h. Xóa một nút là một phân số **x** (*x nhập từ bàn phím*).
 - i. Xóa toàn bộ cây AVL.
2. Cho cây AVL mà mỗi nút chứa thông tin của một **cuốn sách** (gồm những thông tin: MaS, TenS, NamXB). Hãy viết chương trình để thực hiện những chức năng sau:
 - a. Tạo cấu trúc cây AVL.
 - b. Thêm mới một cuốn sách **x** vào cây.
 - c. Duyệt cây AVL bằng 6 cách: NLR, LNR, LRN, NRL, RNL, RLN.
 - d. Đếm số lượng những cuốn sách xuất bản trong năm **n** (*n nhập vào từ bàn phím*).
 - e. Tìm kiếm trên cây có nút nào có tên sách là **y** (*y nhập vào từ bàn phím*)?
 - f. Xóa một nút là một cuốn sách có mã số **z** (*z nhập từ bàn phím*).
 - g. Xóa toàn bộ cây AVL.
3. Cho cây nhị phân tìm kiếm cân bằng (cây AVL) lưu trữ dữ liệu là một từ điển Anh-Việt (Mỗi nút của cây có dữ liệu gồm 2 trường: **word** là khóa chứa một từ tiếng anh,

mean là nghĩa tiếng Việt). Hãy xây dựng cây AVL với những chức năng sau:

- a. Tạo cây AVL từ 1 file từ điển cho trước.
- b. Duyệt cây AVL để xem nội dung theo phép duyệt cây LNR.
- c. Thêm một từ bất kỳ vào cây, duyệt lại cây để xem kết quả.
- d. Xóa một từ bất kỳ khỏi cây, duyệt lại cây để xem kết quả.
- e. Tra cứu nghĩa của 1 từ bất kỳ.
- f. Xóa toàn bộ cây.

11.3. Bài tập ở nhà.

- 4.** Tiếp tục hoàn thiện bài tập 2 như 1 từ điển Anh-Việt thực thụ?

Trường ĐH CNTP TP.HCM Khoa: Công nghệ Thông tin Bộ môn: Công nghệ Phần mềm MSMH: 01201007	BÀI THỰC HÀNH 12: BẢNG BẮM	
--	---	--

A. MỤC TIÊU:

- Vận dụng các cấu trúc bảng băm vào từng bài toán cụ thể.
- Lập trình được các cấu trúc bảng băm thành các chương trình cụ thể chạy được trên máy tính.
- Làm được các bài tập ở cuối mỗi chương.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	<ul style="list-style-type: none"> – Máy tính (có thể là máy bàn hoặc Laptop). – Cấu hình tối thiểu của máy tính: <ul style="list-style-type: none"> ▪ OS: 9x - XP/Win Vista/Win7/Win8x. ▪ RAM: tối thiểu 512 MB. ▪ HDD: tối thiểu 10 GB. 	1	Chiếc	

C. VẬT LIỆU

- Dùng một trong những phần mềm sau: Microsoft Visual C++ 6.0 trở lên, Dev C++, Borland C++ 3.1.
- Không cần mạng Internet.

D. NỘI DUNG THỰC HÀNH**12.1. Mô tả bảng băm****12.1.1. Mô tả dữ liệu**

Bảng băm được mô tả bằng các thành phần sau:

- Có tập khóa (**key**) của các nút trên bảng băm gọi là tập **K**.
- Có tập các địa chỉ (**address**) của bảng băm được gọi là tập **A**.
- Có hàm băm (hash funtion) để ánh xạ một khóa trong tập **K** thành 1 địa chỉ trong tập **A**.

Bảng băm được mô tả bằng hình vẽ như sau:



12.1.2. Các tác vụ trên bảng băm

Bảng băm có thể có các tác vụ sau:

- **Tác vụ khởi động:** Cấp phát bộ nhớ và khởi động các giá trị ban đầu cho bảng băm.
- **Tác vụ tìm kiếm:** Đây là một trong những tác vụ thường được sử dụng nhất của bảng băm. Tác vụ này sẽ tìm kiếm các phần tử trong bảng băm dựa vào khóa của từng phần tử.
- **Tác vụ thêm một phần tử:** Tác vụ này thêm một phần tử mới vào bảng băm.
- **Tác vụ xoá một phần tử:** Tác vụ này được dùng để xoá một phần tử ra khỏi bảng băm.
- **Tác vụ duyệt bảng băm:** Tác vụ này dùng để duyệt qua tất cả các phần tử trên bảng băm.

12.1.3. Các bảng băm thông dụng

Với mỗi loại bảng băm, chúng ta phải xác định tập khóa K, xác định tập địa chỉ A và xây dựng hàm băm.

Khi xây dựng hàm băm chúng ta muốn các khóa khác nhau sẽ ánh xạ vào các địa chỉ khác nhau, nhưng thực tế thì thường xảy ra trường hợp các khóa khác nhau lại ánh xạ vào cùng một địa chỉ, chúng ta gọi là xung đột. Do đó khi xây dựng bảng băm chúng ta phải xây dựng phương án giải quyết sự xung đột trên bảng băm.

Trong chương này ta sẽ nghiên cứu các bảng băm thông dụng như sau với mỗi bảng băm có chiến lược giải quyết sự xung đột riêng.

- **Bảng băm với giải thuật nối kết trực tiếp:** Mỗi địa chỉ của bảng băm tương ứng với một danh sách liên kết. Các nút bị xung đột được nối kết với nhau trên một danh sách liên kết.
- **Bảng băm với giải thuật nối kết hợp nhất:** Bảng băm loại này được cài đặt bằng danh sách kê, mỗi nút có hai trường: trường Key chứa khóa của nút và trường Next chỉ nút kế bị xung đột. Các nút bị xung đột được nối kết với nhau qua trường liên kết Next.
- **Bảng băm với giải thuật dò tuyến tính:** Ví dụ như khi thêm nút vào bảng băm loại này nếu băm lần đầu bị xung đột thì lần lược dò địa chỉ kế...cho đến khi gặp địa chỉ trống đầu tiên thì thêm nút vào địa chỉ này.

12.1.4. Hàm băm

Hàm băm là hàm biến đổi khóa của nút thành địa chỉ trên bảng băm, là giải thuật nhằm sinh ra các **giá trị băm** tương ứng với mỗi **khối dữ liệu** (có thể là một chuỗi kí tự, một đối tượng trong lập trình hướng đối tượng, v.v...).

- Khóa có thể là khóa ở dạng số hay dạng chuỗi.
- Địa chỉ được tính ra là các số nguyên trong khoảng 0 đến $\text{MAXSIZE} - 1$, với MAXSIZE là số địa chỉ trên bảng băm.
- Hàm băm thường được dùng ở dạng công thức: Ví dụ như công thức $f(\text{Key}) = \text{Key} \% \text{MAXSIZE}$, với MAXSIZE là độ lớn của bảng băm.

12.1.5. Ưu điểm của bảng băm

Bảng băm là một cấu trúc dung hòa tốt giữa thời gian truy xuất và dung lượng bộ nhớ:

- Nếu không có sự giới hạn về bộ nhớ thì chúng ta có thể xây dựng bảng băm với mỗi khóa ứng với một địa chỉ với mong muốn thời gian truy xuất tức thời.
- Nếu dung lượng của bộ nhớ có giới hạn thì tổ chức một số khóa có cùng địa chỉ. Lúc này thời gian truy xuất có bị giảm đi.

Bảng băm được ứng dụng nhiều trong thực tế, rất thích hợp khi tổ chức dữ liệu có kích thước lớn và được lưu trữ ở bộ nhớ ngoài.

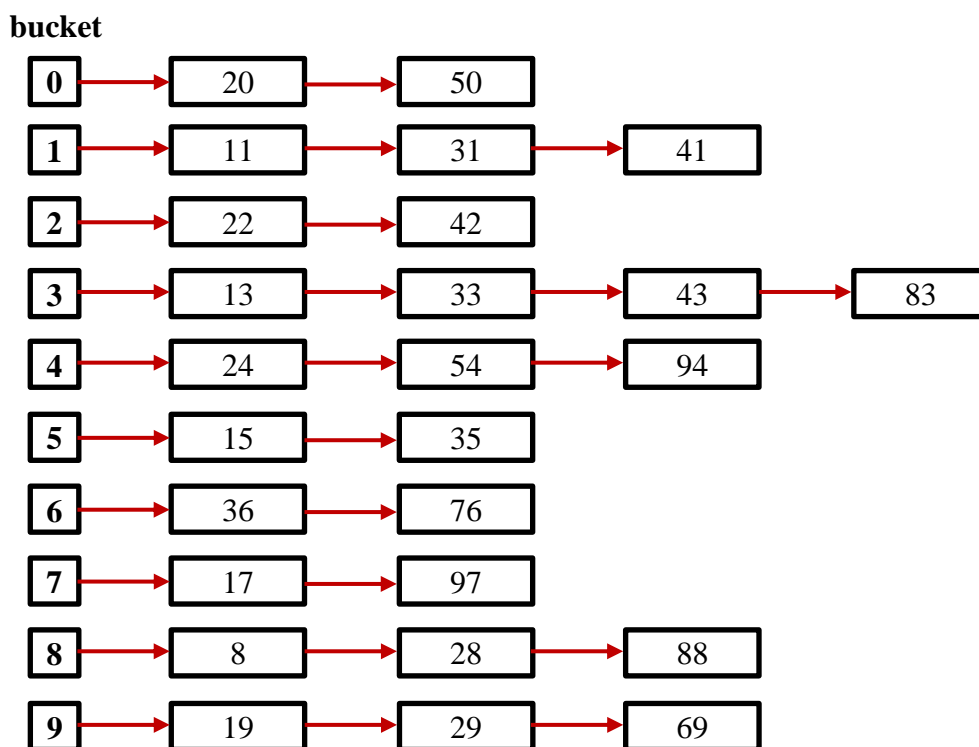
12.2. Bảng băm với giải thuật kết nối trực tiếp**12.2.1. Mô tả**

Bảng băm được cài đặt bằng danh sách liên kết, các nút trên bảng băm được băm thành MAXSIZE danh sách liên kết (từ danh sách 0 đến danh sách MAXSIZE - 1). Các nút bị xung đột tại địa chỉ i được nối kết trực tiếp với nhau qua danh sách liên kết thứ i .

Khi thêm một nút có khóa Key vào bảng băm, hàm băm $f(\text{Key})$ sẽ xác định địa chỉ i trong khoảng 0 đến MAXSIZE - 1 ứng với danh sách liên kết thứ i mà nút này sẽ được thêm vào.

Khi tìm kiếm một nút có khóa Key trên bảng băm, hàm băm $f(\text{Key})$ sẽ xác định địa chỉ i trong khoảng từ 0 đến MAXSIZE - 1 ứng với danh sách liên kết thứ i có thể chứa nút, việc tìm kiếm nút trên bảng băm quy về bài toán tìm kiếm trên danh sách liên kết.

Sau đây là minh họa cho bảng băm có tập khóa K là tập số tự nhiên, tập địa chỉ MAXSIZE có 10 địa chỉ và chọn hàm băm là $f(\text{Key}) = \text{Key} \% 10$.



Bảng băm dùng phương pháp nối kết trực tiếp

12.2.2. Cài đặt

12.2.2.1. Khai báo cấu trúc bảng băm và các hằng số

```
#define MAXSIZE 100
struct HashNode
{ //Định nghĩa kiểu dữ liệu cho 1 nút của Bảng băm
    int Key;
    HashNode *Next;
};
HashNode* bucket[MAXSIZE];
```

12.2.2.2. Hàm băm

```
int HashFunction(int Key)
{
    return (Key % MAXSIZE);
}
```

12.2.2.3. Tìm kiếm một phần tử trên bảng băm

```
// Ham tìm kiếm một khóa k trên bảng băm
int Search(int k)
{
    int b = HashFunction(k);
    HashNode* p = bucket[b];
    while(k > p->Key && p != NULL)
        p = p->Next;
    if(p == NULL || k != p->Key)
        return -1;
    else
        return b;
}
```

12.2.2.4. Thêm vào một phần tử

```
// Thêm nút q chứa khóa k sau nút p
int InsertAfter(HashNode* p, int k)
{
    if(p == NULL)
    {
        printf("Không thêm vào HashNode mới được");
        return 0; //Thực hiện không thành công
    }
    else
    {
        HashNode* q = new HashNode;
        q->Key = k;
        q->Next = p->Next;
        p->Next = q;
    }
    return 1; //Thực hiện thành công
}
```

```

// Tac vu nay chi su dung khi them vao mot bucket co thu tu
void Place(int b, int k)
{
    HashNode *p, *q;
    q = NULL;
    for(p = bucket[b]; p != NULL && k > p->Key; p = p->Next)
        q = p;
    if(q == NULL)
        Push(b, k);
    else
        InsertAfter(q, k);
}
// Them mot nut co khoa la k vao trong bang bam
void Insert(int k)
{
    int b;
    b = HashFunction(k);
    Place(b, k);
}

```

12.2.2.5. Giải phóng và thu hồi lại vùng nhớ đã cấp phát cho một nút

```

// Xoa mot nut trong bo nho
void DeleteHashNode(HashNode* p)
{
    delete p;
}

```

12.2.2.6. Xóa một phần tử

```

// Xoa nut q ke sau nut p
int DeleteAfter(HashNode* p)
{
    HashNode* q;
    int k;
    if(p == NULL || p->Next == NULL)
    {
        printf("\n Khong xoa HashNode duoc");
        return 0;
    }
    q = p->Next;
    k = q->Key;
    p->Next = q->Next;
    DeleteHashNode(q);
    return k;
}
//Xoa mot phan tu co khoa k ra khoi bang bam
void Remove(int k)
{
    int b = HashFunction(k);
    HashNode* p = bucket[b];
    HashNode* q = p;
    while(p != NULL && p->Key != k)

```

```
{
    q = p;
    p = p→Next;
}
if(p == NULL)
    printf("\n Không có HashNode có khóa là: %d", b);
else if(p == bucket[b])
    Pop(b);
else
    DeleteAfter(q);
}
```

12.2.3. Chương trình minh họa

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

#define TRUE 1
#define FALSE 0
#define MAXSIZE 100

struct HashNode
{ //Định nghĩa kiểu dữ liệu cho 1 nút của Bảng băm
    int Key;
    HashNode *Next;
};
HashNode* bucket[MAXSIZE];
//=====
int HashFunction(int Key)
{
    return (Key % MAXSIZE);
}
//=====
// Khởi tạo bảng băm
void InitHashTable()
{ //Initialize HashTable
    for(int b = 0; b < MAXSIZE; b++)
        bucket[b] = NULL;
}
//=====
// Xóa một nút trong bộ nhớ
void DeleteHashNode(HashNode* p)
{
    delete p;
}
//=====
// Kiểm tra một bucket có phải Empty?
int IsEmptyBucket(int b)
{
    if(bucket[b] == NULL)
        return TRUE;
    else
        return FALSE;
}
```

```

}
//=====
// Kiểm tra toàn bộ bảng băm có NULL hay không
int IsEmpty()
{
    for(int b = 0; b < MAXSIZE; b++)
        if(bucket[b] != NULL)
            return FALSE;
    return TRUE;
}
//=====
// Traverse trên từng bucket
void TraverseBucket(int b)
{
    HashNode* p = bucket[b];
    while(p != NULL)
    {
        printf("%4d", p->Key);
        p = p->Next;
    }
}
//=====
// Duyệt qua bảng băm
void Traverse()
{
    for(int b = 0; b < MAXSIZE; b++)
    {
        printf("\n Bucket[%d]=", b);
        TraverseBucket(b);
    }
}
//=====
// Xóa trên một bucket
void ClearBucket(int b)
{
    HashNode* q = NULL;
    HashNode* p = bucket[b];
    while(p != NULL)
    {
        q = p;
        p = p->Next;
        DeleteHashNode(q);
    }
    bucket[b] = NULL;
}
//=====
// Xóa toàn bộ bảng băm
void Clear()
{
    for(int b = 0; b < MAXSIZE; b++)
    {
        ClearBucket(b);
    }
}
//=====
// Thêm một nút vào đầu bucket
void Push(int b, int x)

```

```

{
    HashNode* p = new HashNode;
    p->Key = x;
    p->Next = bucket[b];
    bucket[b] = p;
}
//=====
// Xoa mot nut o dau bucket
int Pop(int b)
{
    if(IsEmptyBucket(b))
    {
        printf("Bucket %d bi rong, khong xoa duoc", b);
        return 0;
    }
    HashNode* p = bucket[b];
    int k = p->Key;
    bucket[b] = p->Next;
    DeleteHashNode(p);
    return k;
}
//=====
// Tac vu them vao bucket mot nut moi sau nut p
int InsertAfter(HashNode* p, int k)
{
    if(p == NULL)
    {
        printf("Khong them vao HashNode moi duoc!");
        return 0; //Thực hiện không thành công
    }
    else
    {
        HashNode* q = new HashNode;
        q->Key = k;
        q->Next = p->Next;
        p->Next = q;
    }
    return 1; //Thực hiện thành công
}
//=====
// Tac vu nay chi su dung khi them vao mot bucket co thu tu
void Place(int b, int k)
{
    HashNode* p, *q;
    q = NULL;
    for(p = bucket[b]; p != NULL && k > p->Key; p = p->Next)
        q = p;
    if(q == NULL)
        Push(b, k);
    else
        InsertAfter(q, k);
}
//=====
// Them mot nut co khoa k vao trong bang bam
void Insert(int k)
{

```



```

    int b;
    b = HashFunction(k);
    Place(b, k);
}
//=====
// Tac vu tim kiem mot khoa trong bang bam
int Search(int k)
{
    int b = HashFunction(k);
    HashNode* p = bucket[b];
    while(p != NULL && k > p->Key)
        p = p->Next;
    if(p == NULL || k != p->Key)
        return -1;
    else
        return b;
}
//=====
// Xoa mot nut ngay sau nút p
int DeleteAfter(HashNode* p)
{
    HashNode* q;
    int k;
    if(p == NULL || p->Next == NULL)
    {
        printf("\n Không xoa HashNode duoc");
        return 0;
    }
    q = p->Next;
    k = q->Key;
    p->Next = q->Next;
    DeleteHashNode(q);
    return k;
}
//=====
// Xoa mot phan tu co khoa k ra khoi bang bam
void Remove(int k)
{
    int b;
    HashNode *p, *q;
    b = HashFunction(k);
    p = bucket[b];
    q = p;
    while(p != NULL && p->Key != k)
    {
        q = p;
        p = p->Next;
    }
    if(p == NULL)
        printf("\n Không co HashNode co khoa la: %d", b);
    else if(p == bucket[b])
        Pop(b);
    else
        DeleteAfter(q);
}
//=====

```

```
// ShowMenu chương trình
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                      MENU                      *");
    printf("\n*-----*");
    printf("\n* 1: Them mot nut vao bang                      *");
    printf("\n* 2: Them ngau nhien nhieu nut vao bang bam      *");
    printf("\n* 3: Xoa mot nut trong bang bam                  *");
    printf("\n* 4: Xoa toan bo bang bam                        *");
    printf("\n* 5: Duyet bang bam                             *");
    printf("\n* 6: Tim kiem tren bang bam                      *");
    printf("\n* 0: Ket thuc chuong trinh                      *");
    printf("\n*****");
}
//=====
// Hàm xử lý chính của chương trình
void Process()
{
    int b, Key, i, n, SelectFunction;
    char c;
    InitHashTable();
    do
    {
        ShowMenu();
        printf("\n Chuc nang ban chon: ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            case 1:
                printf("\n Them mot nut vao trong bang bam");
                printf("\n Nhap gia tri khoa cua nut can them vao: ");
                scanf("%d", &Key);
                Insert(Key);
                break;
            case 2:
                printf("\n Them mot bang ngau nhien nhieu nut vao bang");
                printf("\n So nut ban muon them: ");
                scanf("%d", &n);
                srand(unsigned(time(NULL)));
                for(i = 0; i < n; i++)
                {
                    Key = rand()%100; // 1 số ngẫu nhiên thuộc [0, 99]
                    Insert(Key);
                }
                break;
            case 3:
                printf("\n Xoa mot nut tren bang bam");
                printf("\n Nhap vao khoa cua nut can xoa: ");
                scanf("%d", &Key);
                Remove(Key);
                break;
            case 4:
                Clear();
                break;
            case 5:
                printf("\n Duyet Bang Bam");
        }
    }
}
```

```

        Traverse();
        break;
    case 6:
        printf("\n Tim kiem mot khoa tren bang bam");
        printf("\n Khoa can tim: ");
        scanf("%d", &Key);
        b = Search(Key);
        if(b == -1)
            printf("\n Khong thay");
        else
            printf("\n Tim thay trong bucket %d", b);
        break;
    }
}while(SelectFunction != 0);
}
//=====
// Chương trình chính
void main()
{
    Process();
}

```

12.3. Bảng băm với giải thuật kết nối hợp nhất

12.3.1. Mô tả

Bảng băm trong trường hợp này được cài đặt bằng danh sách liên kết dùng mảng, có MAXSIZE nút. Các nút bị xung đột địa chỉ được nối kết với nhau qua một danh sách liên kết.

Mỗi nút của bảng băm là một mẫu tin có hai trường:

- Trường **Key**: chứa khóa của nút.
- Trường **Next**: con trỏ chỉ nút kế tiếp nếu có xung đột.

Khi khởi động bảng băm thì tất cả trường Key được gán giá trị là NULLKEY, tất cả các trường Next được gán là -1.

Hình vẽ sau mô tả bảng băm ngay sau khi vừa khởi động:

NULLKEY	-1
NULLKEY	-1
NULLKEY	-1
NULLKEY	-1
...	...
NULLKEY	-1

Khi thêm một nút có khóa Key vào bảng băm, hàm băm $f(\text{Key})$ sẽ xác định địa chỉ i trong khoảng từ 0 đến MAXSIZE - 1.

- Nếu chưa bị xung đột thì thêm nút mới tại địa chỉ i này.

- Nếu bị xung đột thì nút mới được cấp phát là nút trống phía cuối mảng. Cập nhật liên kết Next sao cho các nút bị xung đột hình thành một danh sách liên kết.

Khi tìm một nút có khóa Key trong bảng băm, hàm băm $f(\text{Key})$ sẽ xác định địa chỉ i trong khoảng từ 0 đến $\text{MAXSIZE} - 1$, tìm nút khóa Key trong danh sách liên kết xuất phát từ địa chỉ i .

Minh họa:

Sau đây là minh họa cho bảng băm có tập khóa là số tự nhiên, tập địa chỉ có 10 địa chỉ ($\text{MAXSIZE} = 10$), chọn hàm băm $f(\text{Key}) = \text{Key} \% 10$. Hình vẽ sau đây minh họa cho tiến trình thêm các nút 10, 15, 26, 30, 25, 35 vào bảng băm.

Hình (a): Sau khi thêm 3 nút 10, 15, 26 vào bảng băm – lúc này chưa bị xung đột.

Hình (b): Thêm nút 30 vào bảng băm - bị xung đột tại địa chỉ 0 – nút 30 được cấp phát tại địa chỉ 9, trường Next của nút tại địa chỉ 0 được gán là 9.

Hình (c): Thêm nút 25 vào bảng băm - bị xung đột tại địa chỉ 5 – nút 25 được cấp phát tại địa chỉ 8, trường Next của nút tại địa chỉ 5 được gán là 8.

Hình (d): Thêm nút 35 vào bảng băm - bị xung đột tại địa chỉ 5 và địa chỉ 8 – nút 35 được cấp phát tại địa chỉ 7, trường Next của nút tại địa chỉ 8 được gán là 7.

(a)			(b)			(c)			(d)		
0	10	-1	0	10	9	0	10	9	0	10	9
1		-1	1		-1	1		-1	1		-1
2		-1	2		-1	2		-1	2		-1
3		-1	3		-1	3		-1	3		-1
4		-1	4		-1	4		-1	4		-1
5	15	-1	5	15	-1	5	15	8	5	15	8
6	26	-1	6	26	-1	6	26	-1	6	26	-1
7		-1	7		-1	7		-1	7	35	-1
8		-1	8		-1	8	25	-1	8	25	7
9		-1	9	30	-1	9	30	-1	9	30	-1

Hình minh họa việc thêm các khóa vào bảng băm

12.3.2. Cài đặt

12.3.2.1. Khai báo cấu trúc bảng băm và các hằng số

```
#define TRUE 1
#define FALSE 0
#define NULLKEY -1
#define MAXSIZE 100

struct HashNode
{
    //Định nghĩa kiểu dữ liệu cho 1 nút của Bảng băm
    int Key;
    int Next;
}
```

```
};
HashNode HashTable[MAXSIZE];
int avail;
```

12.3.2.2. Hàm băm

```
int HashFunction(int Key)
{
    return (Key % MAXSIZE);
}
```

12.3.2.3. Tác vụ khởi động cho bảng băm

```
void InitHashTable()
{ //Initialize HashTable
    for(int i = 0; i < MAXSIZE; i++)
    {
        HashTable[i].Key = NULLKEY;
        HashTable[i].Next = -1;
    }
    avail = MAXSIZE-1;
}
```

12.3.2.4. Kiểm tra bảng băm rỗng

```
// Kiểm tra bảng băm có rỗng hay không?
int IsEmpty()
{
    for(int i = 0; i < MAXSIZE; i++)
    {
        if (HashTable[i].Key != NULLKEY)
            return FALSE;
    }
    return TRUE;
}
```

12.3.2.5. Tác vụ tìm kiếm

```
// Tìm một khóa có trong bảng băm hay không, tìm thấy trả về địa chỉ
// không thấy trả về MAXSIZE
int Search(int k)
{
    int i = HashFunction(k);
    while(k != HashTable[i].Key && i != -1)
        i = HashTable[i].Next;
    if(k == HashTable[i].Key)
        return i;
    else
        return MAXSIZE;
}
```

12.3.2.6. Chọn nút con để cập nhật khi xảy ra xung đột

```
// Chọn nút con trong phía dưới bảng hash để cập nhật khi xảy ra xung đột
int GetEmpty()
```

```
{  
    while(HashTable[avail].Key != NULLKEY)  
        avail--;  
    return avail;  
}
```

12.3.2.7. Tác vụ thêm một phần tử vào bảng băm

```
// Thêm một nút có khóa k vào bảng băm  
int Insert(int k)  
{  
    int i, j;  
    i = Search(k);  
    if(i != MAXSIZE)  
    {  
        printf("\n khóa %d bị trùng, không thêm vào được", k);  
        return i; //Thực hiện không thành công  
    }  
    i = HashFunction(k);  
    while(HashTable[i].Next >= 0)  
        i = HashTable[i].Next;  
    if(HashTable[i].Key == NULLKEY)  
        j = i; //không có sự đụng độ, first time  
    else  
    {  
        j = GetEmpty();  
        if(j < 0)  
        {  
            printf("\n Bảng băm bị đầy không thêm vào được");  
            return j; //Thực hiện không thành công  
        }  
        else  
            HashTable[i].Next = j;  
    }  
    HashTable[j].Key = k;  
    return j; //Thực hiện thành công  
}
```

12.3.2.8. Tác vụ duyệt bảng băm

```
// Xem chi tiết thông tin bảng băm  
void ViewTable()  
{  
    for(int i = 0; i < MAXSIZE; i++)  
    {  
        printf("\n table[%2d]: %4d %4d", i, HashTable[i].Key,  
            HashTable[i].Next);  
    }  
}
```

12.3.3. Chương trình minh họa

Sinh viên tự làm xem như một bài tập.

12.4. BẢNG BĂM VỚI GIẢI THUẬT DÒ TUYẾN TÍNH

12.4.1. Mô tả

Bảng băm trong trường hợp này được cài đặt bằng một danh sách kê có MAXSIZE nút, mỗi nút của bảng băm là một mẫu tin có một trường Key để chứa khóa của nút.

Khi thêm nút có khóa Key vào bảng băm, hàm băm $f(\text{Key})$ sẽ xác định địa chỉ i trong khoảng từ 0 đến MAXSIZE - 1:

- Nếu chưa bị xung đột thì thêm nút mới tại địa chỉ i này.
- Nếu bị xung đột thì hàm băm lần 1 f_1 sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm lần 2 f_2 sẽ xét địa chỉ kế tiếp nữa... quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm nút vào địa chỉ này.

Khi tìm một nút có khóa Key trong bảng băm, hàm băm $f(\text{Key})$ sẽ xác định địa chỉ i trong khoảng từ 0 đến MAXSIZE - 1, tìm nút khóa Key trong khối đặc chứa các nút xuất phát từ địa chỉ i .

Hàm băm lại của phương pháp dò tìm tuyến tính là truy xuất địa chỉ kế tiếp. Hàm băm lại được biểu diễn bằng công thức sau: $f(\text{Key}) = (f(\text{Key}) + i) \% \text{MAXSIZE}$

Minh họa:

Sau đây là minh họa cho bảng băm có tập khóa là tập số tự nhiên, tập địa chỉ có 10 địa chỉ, chọn hàm băm $f(\text{Key}) = \text{Key} \% 10$.

Hình vẽ sau miêu tả tiến trình thêm các nút 32, 53, 22, 92, 17, 34 vào bảng băm.

Hình (a): Sau khi thêm 2 nút 32 và 53 vào bảng băm – lúc này chưa bị xung đột.

Hình (b): Thêm nút 22 và 92 vào bảng băm - bị xung đột tại địa chỉ 2, nút 22 được cấp phát tại địa chỉ 4, nút 92 được cấp phát tại địa chỉ 5.

Hình (c): thêm nút 17 và 34 vào bảng băm – nút 17 không bị xung đột cấp phát tại địa chỉ 7, nút 34 bị xung đột tại địa chỉ 4, được cấp tại địa chỉ 6.

(a)	(b)	(c)
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9

Hình minh họa việc thêm các khóa vào bảng băm

12.4.2. Cài đặt

12.4.2.1. Khai báo cấu trúc bảng băm và định nghĩa các hằng số

```
#define TRUE 1
#define FALSE 0
#define NULLKEY -1
#define MAXSIZE 100

struct HashNode
{ //Định nghĩa kiểu dữ liệu cho 1 nút của Bảng băm
    int Key;
};
HashNode HashTable[MAXSIZE];
int N;
```

12.4.2.2. Hàm băm

```
int HashFunction(int Key)
{
    return (Key % MAXSIZE);
}
```

12.4.2.3. Tác vụ khởi động

```
void InitHashTable()
{ //Initialize HashTable
    for(int i = 0; i < MAXSIZE; i++)
    {
        HashTable[i].Key = NULLKEY;
    }
    N = 0;
}
```

12.4.2.4. Kiểm tra bảng băm có rỗng hay không

```
int IsEmpty()
{
    if(N == 0)
        return TRUE; //Nếu rỗng
    else
        return FALSE; //Nếu không rỗng
}
```

12.4.2.5. Kiểm tra bảng băm có đầy hay không

```
int IsFull()
{
    if(N == MAXSIZE - 1)
        return TRUE; //Nếu đầy
    else
        return FALSE; //Nếu chưa đầy
}
```


12.4.2.6. Tác vụ tìm kiếm

```

//Tác vụ tìm 1 Key, tìm thay tra ve index, không thay tra ve MAXSIZE
int Search(int k)
{
    int i = HashFunction(k);
    while(HashTable[i].Key != k && HashTable[i].Key != NULLKEY)
    {
        i = i + 1;
        if(i >= MAXSIZE)
            i = i - MAXSIZE;
    }
    if(HashTable[i].Key == k)
        return i;
    else
        return MAXSIZE;
}

```

12.4.2.7. Tác vụ thêm một phần tử vào bảng băm

```

// Thêm khoa k vào bang bam
int Insert(int k)
{
    int i, j;
    if(IsFull() == TRUE)
    {
        printf("\n Bang bam bi day, không thêm vào được");
        return MAXSIZE;
    }
    i = HashFunction(k);
    while(HashTable[i].Key != NULLKEY)
    {
        i++;
        if(i >= MAXSIZE)
            i = i - MAXSIZE;
    }
    HashTable[i].Key = k;
    N++;
    return i;
}

```

12.4.2.8. Tác vụ duyệt bảng băm

```

// Xem chi tiết thông tin bang bam
void ViewTable()
{
    for(int i = 0; i < MAXSIZE; i++)
    {
        printf("\n table[%2d]: %4d", i, HashTable[i].Key);
    }
}

```

12.4.3. Chương trình minh họa

Sinh viên tự làm xem như một bài tập.

12.5. BÀI TẬP

1. Viết một chương trình hiện thực từ điển Anh - Việt, chương trình có cài đặt bảng bấm với phương pháp nối kết trực tiếp. Mỗi nút của bảng bấm có khai báo các trường sau:

- Trường word là khóa chứa một từ tiếng anh.
- Trường mean là nghĩa tiếng Việt.
- Trường Next là con trỏ chỉ nút kế bị xung đột cùng địa chỉ.

Tập khóa là một chuỗi tiếng anh, tập địa chỉ có 26 chữ cái. Chọn hàm bấm sau cho khóa bắt đầu bằng ký tự **a** được bấm vào địa chỉ 0, **b** bấm vào địa chỉ 1, ..., **z** bấm vào địa chỉ 25. Chương trình có những chức năng như sau:

- Nhập vào một từ.
- Xem từ điển theo ký tự đầu.
- Xem toàn bộ từ điển.
- Tra từ điển.
- Xoá một từ.
- Xóa toàn bộ từ điển.
- Thoát khỏi chương trình.

2. Viết chương trình xem thông tin về một sinh viên qua mã số sinh viên.

Thông tin về tất cả sinh viên chứa trong tập tin văn bản, mỗi sinh viên chiếm một dòng văn bản gồm MSSV, họ, tên và điểm các môn học. Chương trình sẽ đọc tập tin văn bản này để tạo ra bảng bấm. Mỗi nút của bảng bấm bao gồm trường MSSV dùng làm khóa, trường line cho biết vị trí dòng văn bản của sinh viên. Khi truy xuất thông tin về một sinh viên chúng ta nhập MSSV, qua bảng bấm chúng ta xác định được vị trí dòng văn bản và đọc thông tin sinh viên qua dòng văn bản này.

Chương trình có các chức năng như sau:

- Xem tất cả MSSV.
- Xem MSSV trong khoảng từ ... đến ...
- Xem thông tin về sinh viên qua MSSV.

PHỤ LỤC – Các chương trình ví dụ mẫu

Bài hướng dẫn 1: Nhập 1 mảng chứa thông tin các sinh viên (gồm: MaSV – chuỗi 10 ký tự, HoSV – chuỗi 25 ký tự, TenSV – chuỗi 7 ký tự, DiemTB – số thực).

- Viết chương trình dạng thực đơn.
- Cho biết sinh viên có DiemTB lớn nhất?
- Tìm kiếm và cho biết có sinh viên nào có tên y hay không (*y nhập từ bàn phím*)?
- Sắp xếp danh sách sinh viên tăng dần theo tên.

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

#define MAXSIZE 100

typedef char KeyType;
struct SINHVIEN
{ //Định nghĩa kiểu dữ liệu để lưu thông tin cho 1 sinh viên
    KeyType MaSV[11];
    char HoSV[26];
    char TenSV[10];
    float DiemTB;
};
typedef SINHVIEN ItemType;

void NhapThongTinSV(ItemType &x)
{
    printf("\nNhap ma so sinh vien: ");
    fflush();
    gets(x.MaSV);
    printf("Nhap ho sinh vien: ");
    fflush();
    gets(x.HoSV);
    printf("Nhap ten sinh vien: ");
    fflush();
    gets(x.TenSV);
    printf("Nhap diem trung binh sinh vien: ");
    scanf("%f", &x.DiemTB);
}

//=====
void XuatThongTinSV(ItemType x)
{
    printf("%-15s%-20s%-10s%5.1f", x.MaSV, x.HoSV, x.TenSV, x.DiemTB);
}

//=====
void NhapDanhSachSV(ItemType a[], int &n)
{
    do
    {
        printf("Cho biet so luong sinh vien: ");
        scanf("%d", &n);
    }while(n <= 0);
    for(int i = 0; i < n; i++)
```

```

        NhapThongTinSV(a[i]);
    }
//=====
void XuatDanhSachSV(ItemType a[], int n)
{
    printf("\nDANH SACH SINH VIEN: ");
    printf("\n%-5s%-15s%-30s%-10s", "STT", "MA SO SV", "HO VA TEN SINH VIEN", "DIEM TB");
    for(int i = 0; i < n; i++)
    {
        printf("\n%-5d", i + 1);
        XuatThongTinSV(a[i]);
    }
}
//=====
void XuatThongTinSV_DiemTB_Max(ItemType a[], int n)
{
    int vtMaxDTB = 0;
    for(int i = 1; i < n; i++)
        if(a[i].DiemTB > a[vtMaxDTB].DiemTB)
            vtMaxDTB = i;
    printf("\nThong tin sinh vien co DiemTB lon nhat\n");
    XuatThongTinSV(a[vtMaxDTB]);
}
//=====
int LinearSearch_Tim_TheoTenSV(ItemType a[], int n, char ten[])
{
    for(int i = 0; i < n; i++)
        if(stricmp(a[i].TenSV, ten) == 0)
            return i;
    return -1;
}
//=====
void HoanVi(ItemType &x, ItemType &y)
{
    ItemType tam;
    tam = x;
    x = y;
    y = tam;
}
//=====
void InterchageSort_SXTang_TheoTen(ItemType a[], int n)
{
    //Bang giai thuat sap xep doi cho truc tiep
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(stricmp(a[i].TenSV, a[j].TenSV) > 0)
                HoanVi(a[i], a[j]);
}
//=====
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                               MENU                               *");
    printf("\n*-----*");
    printf("\n* 1. Nhap danh sach sinh vien                                     *");
    printf("\n* 2. Xuat danh sach sinh vien                                     *");
    printf("\n* 3. Thong tin sinh vien co diem trung binh cao nhat           *");
    printf("\n* 4. Xem thong tin sinh vien co ten bat ky                     *");
}

```

```

printf("\n* 5. Sắp xếp danh sách sinh viên tăng dần theo tên      *");
printf("\n* 0. Thoát                                              *");
printf("\n*****");
}
//=====
void Process()
{
    ItemType A[MAXSIZE];
    int N, kq, SelectFunction;
    char X[11];
    do
    {
        ShowMenu();
        do
        {
            printf("\nBạn hãy chọn 1 chức năng (0 -> 5): ");
            scanf("%d", &SelectFunction);
        } while (SelectFunction < 0 || SelectFunction > 5);
        switch (SelectFunction)
        {
            case 1:
                NhapDanhSachSV(A, N);
                break;
            case 2:
                XuatDanhSachSV(A, N);
                break;
            case 3:
                XuatThongTinSV_DiemTB_Max(A, N);
                break;
            case 4:
                printf("\nCho biết tên sinh viên muốn tìm kiếm: ");
                fflush();
                gets(X);
                kq = LinearSearch_Tim_TheoTenSV(A, N, X);
                if (kq == -1)
                    printf("\nKhông tìm thấy sinh viên nào có tên %s.", X);
                else
                {
                    printf("\nThông tin sinh viên muốn tìm là: \n");
                    XuatThongTinSV(A[kq]);
                }
                break;
            case 5:
                InterchangeSort_SXTang_TheoTen(A, N);
                XuatDanhSachSV(A, N);
        }
    } while (SelectFunction != 0);
}
//=====
void main()
{
    Process();
}

```

Bài hướng dẫn 2: Thực hiện lại bài tập 4 (**BÀI THỰC HÀNH 1**), bổ sung thêm các giải thuật tìm kiếm nhị phân, sắp xếp BubbleSort?

```

...
//=====
int LinearSearch_TimSV_TheoMaSV(ItemType a[], int n, KeyType maso[])
{
    for(int i = 0; i < n; i++)
        if(stricmp(a[i].MaSV, maso) == 0)
            return i;
    return -1;
}
//=====
int BinarySearch_TimSV_TheoMaSV(ItemType a[], int n, KeyType *maso)
{
    int Left = 0, Right = n - 1, Mid;
    while(Left <= Right)
    {
        Mid = (Left + Right)/2;
        if(stricmp(a[Mid].MaSV, maso) == 0)
            return Mid;
        else if(stricmp(a[Mid].MaSV, maso) > 0)
            Right = Mid - 1;
        else
            Left = Mid + 1;
    }
    return -1;
}
//=====
void HoanVi(ItemType &x, ItemType &y)
{
    ItemType tam;
    tam = x;
    x = y;
    y = tam;
}
//=====
void InterchangeSort_SXTang_TheoMaSV(ItemType a[], int n)
{
    //Bang giai thuat sap xep doi cho truc tiep
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(stricmp(a[i].MaSV, a[j].MaSV) > 0)
                HoanVi(a[i], a[j]);
}
...
//=====
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                      MENU                      *");
    printf("\n*-----*");
    printf("\n* 1. Nhap danh sach sinh vien.                  *");
    printf("\n* 2. Xuat danh sach sinh vien.                   *");
    printf("\n* 3. Tim sinh vien theo ma so bang Linear Search. *");
    printf("\n* 4. Tim sinh vien theo ma so bang Binary Search. *");
    ...
    printf("\n* 0. Thoat khoi chuong trinh.                  *");
}

```

```

printf("\n*****");
}
//=====
void Process()
{
    ItemType A[MAXSIZE];
    int N, Y, kq, SelectFunction;
    KeyType X[11];
    char Ten[10];
    do
    {
        ShowMenu();
        do
        {
            printf("\nBan hay chon mot chuc nang: ");
            scanf("%d", &SelectFunction);
        }while(SelectFunction < 0);
        switch(SelectFunction)
        {
            case 1:
                NhapDanhSachSV(A, N);
                break;
            case 2:
                XuatDanhSachSV(A, N);
                break;
            case 3:
                printf("\nNhap ma so sinh vien muon tim: ");
                fflush();
                gets(X);
                kq = LinearSearch_TimSV_TheoMaSV(A, N, X);
                if(kq == -1)
                    printf("\nKhong tim thay sinh vien co ma so %s.", X);
                else
                {
                    printf("\nThong tin sinh vien muon tim la:\n ");
                    XuatThongtinSV(A[kq]);
                }
                break;
            case 4:
                //Su dung giai thuat Interchange de s.xep DSSV tang dan theo MaSV
                InterchangeSort_SXTang_TheoMaSV(A, N);
                printf("\nNhap ma so sinh vien muon tim: ");
                fflush();
                gets(X);
                kq = BinarySearch_TimSV_TheoMaSV(A, N, X);
                if(kq == -1)
                    printf("\nKhong tim thay sinh vien co ma so %s.", X);
                else
                {
                    printf("\nThong tin sinh vien muon tim la:\n ");
                    XuatThongtinSV(A[kq]);
                }
                break;
            //case 5: ...
        }
    }while(SelectFunction != 0);
}

```

```
//=====
void main()
{
    Process();
}
```

Bài hướng dẫn 3: Đọc ghi file từ mảng 1 chiều, mảng 2 chiều, mảng cấu trúc?

a. Đọc ghi file từ mảng 1 chiều các số nguyên:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXSIZE 100
//=====
void ReadFromFile(int a[], int &n)
{
    FILE *f;
    f=fopen("FileData1D.txt","rt");
    if(!f) return;
    fscanf(f, "%d", &n);
    for(int i=0; i<n; i++)
        fscanf(f, "%d", &a[i]);
    fclose(f);
}
//=====
void WriteToFile(int a[],int n)
{
    FILE *f;
    f = fopen("FileData1D.txt","wt");
    if(!f) return;
    fprintf(f, "%d\n", n);
    for(int i=0; i<n; i++)
        fprintf(f, "%d ", a[i]);
    fclose(f);
}
//=====
void EnterPositiveInteger(int &x)
{
    do
    {
        printf("Please enter a positive integer: ");
        scanf("%d", &x);
    }while(x <= 0);
}
//=====
void CreateArray1D(int a[], int &n)
{
    EnterPositiveInteger(n);
    srand((unsigned)time(NULL));
    for(int i = 0; i < n; i++)
        a[i] = (rand()%199) - 99; //Tạo 1 số ngẫu nhiên trong đoạn [-99, 99]
}
//=====
void ShowArray1D(int a[], int n)
{

```



```

    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
}
//=====
void main()
{
    int A[MAXSIZE];
    int N;
    CreateArray1D(A, N);
    printf("\nMang ban dau:\n");
    ShowArray1D(A, N);
    WriteToFile(A, N);
    ReadFromFile(A, N);
    printf("\nKet qua doc file:\n");
    ShowArray1D(A, N);
    getch();
}

```

b. Đọc ghi file từ mảng 2 chiều các số nguyên:

```

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXSIZE 100
//=====
void ReadFromFile(int a[][MAXSIZE], int &m, int &n)
{
    FILE *f;
    f=fopen("FileData2D.txt", "rt");
    if(!f) return;
    fscanf(f, "%d%d\n", &m, &n);
    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
            fscanf(f, "%d", &a[i][j]);
        fscanf(f, "\n");
    }
    fclose(f);
}
//=====
void WriteToFile(int a[][MAXSIZE], int m, int n)
{
    FILE *f;
    f = fopen("FileData2D.txt", "wt");
    if(!f) return;
    fprintf(f, "%d %d\n", m, n);
    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
            fprintf(f, "%4d", a[i][j]);
        fprintf(f, "\n");
    }
    fclose(f);
}
//=====
void EnterPositiveInteger(int &x)
{

```

```
do
{
    printf("Please enter a positive integer: ");
    scanf("%d", &x);
}while(x <= 0);
}
//=====
void CreateArray2D(int a[][MAXSIZE], int &m, int &n)
{
    EnterPositiveInteger(m);
    EnterPositiveInteger(n);
    srand((unsigned)time(NULL));
    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
            a[i][j] = (rand()%199) - 99; //Tạo 1 số ngẫu nhiên thuộc [-99, 99]
}
//=====
void ShowArray2D(int a[][MAXSIZE], int m, int n)
{
    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
            printf("%4d", a[i][j]);
        printf("\n");
    }
}
//=====
void main()
{
    int A[MAXSIZE][MAXSIZE];
    int M, N;
    CreateArray2D(A, M, N);
    printf("\nMang ban dau:\n");
    ShowArray2D(A, M, N);
    WriteToFile(A, M, N);
    ReadFromFile(A, M, N);
    printf("\nKet qua doc file:\n");
    ShowArray2D(A, M, N);
    getch();
}
```

c. Đọc ghi file từ mảng dữ liệu có cấu trúc?

```
#include <conio.h>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <string.h>

#define fio "Products.txt"
#define fbin "Products.dat"
struct Product
{
    char ProductID[11]; //Mã số sản phẩm
    char ProductName[51]; //Tên sản phẩm
    int Quantity; //Số lượng
    double Cost; //Giá
};
```

```

void Input(Product prd[], int &n);
void WriteBinary(Product prd[], int n);
void write(Product prd[], int n);
void Sort_Cost(Product prd[], int n);
void ReadBinary(Product prd[], int *n);
void Output(Product prd[], int n);
void Search_ID(Product prd[], int n, char id[]);
void Search_Name(Product prd[], int n, char ProductName[]);
void Search(Product prd[], int n);
void MainMenu()
{
    printf("\n*****");
    printf("\n*                MENU                *");
    printf("\n*-----*");
    printf("\n* 1: Input                        *");
    printf("\n* 2: Read and sort by Cost        *");
    printf("\n* 3: Search information product   *");
    printf("\n* 0: exit                        *");
    printf("\n*-----*");
}
// Ham main
void main()
{
    Product prd[100];
    int n, i;
    FILE *f = fopen(fio, "r");
    int select;
    do
    {
        MainMenu();
        printf("Enter the number to work: ");
        scanf("%d", &select);
        switch(select)
        {
            case 1:
            {
                printf("1: INPUT\n");
                Input(prd, n);
                write(prd, n);
                break;
            }
            case 2:
            {
                printf("2: READ AND SORT BY COST\n");
                printf("Befor sort:\n-----\n");
                ReadBinary(prd, &n);
                Output(prd, n);
                Sort_Cost(prd, n);
                write(prd, n);
                printf("After sort:\n-----\n");
                ReadBinary(prd, &n);
                Output(prd, n);
                break;
            }
            case 3:
            {

```

```

        printf("SEARCH\n");
        Search(prd, n);
        break;
    }
    case 0:
        return;
    default:
        printf("Error select !");
    }
} while (select != 0);
}
void EnterInformationProduct(Product &x, int i)
{
    printf("Enter the ProductID of products %d: ", i);
    fflush();
    gets(x.ProductID);
    printf("\tEnter the name of products %d: ", i);
    fflush();
    gets(x.ProductName);
    printf("\tEnter the Quantity of products %d: ", i);
    scanf("%d", &x.Quantity);
    printf("\tEnter the Cost of products %d: ", i);
    scanf("%lf", &x.Cost);
}
void ShowTitle()
{
    printf("%-10s %-10s %-15s %-10s %-10s\n", "Order", "ProductID", "Name",
"Quantity", "Cost");
}
void ShowInformationProduct(Product x, int i)
{
    printf("%-10d %-10s %-15s %-10d %-10.2lf\n", i, x.ProductID,
x.ProductName, x.Quantity, x.Cost);
}
void WriteInformationProduct(FILE *f, Product x, int i)
{
    fprintf(f, "%-10d %-10s %-15s %-10d %-10.2lf\n", i, x.ProductID,
x.ProductName, x.Quantity, x.Cost);
}
// Ham Input
void Input(Product prd[], int &n)
{
    printf("Enter the number of products: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        EnterInformationProduct(prd[i], i+1);
}
// Ghi file
void WriteBinary(Product prd[], int n)
{
    FILE *f = fopen(fbin, "wb");
    if(f == NULL)
        printf("Error load file!");
    else
        fwrite(prd, sizeof(Product), n, f);
    fclose(f);
}

```

```

void write(Product prd[], int n)
{
    FILE *f = fopen(fio, "w");
    if(f == NULL)
    {
        printf("Error load file");
        return;
    }
    fprintf(f, "%-10s %-10s %-15s %-10s %-10s\n", "Order", "ProductID",
    "Name", "Quantity", "Cost");
    for (int i = 0; i < n; i++)
        WriteInformationProduct(f, prd[i], i + 1);
    fclose(f);
    WriteBinary(prd, n);
    printf("Input and write success to file!");
}
void Swap(Product &x, Product &y)
{
    Product temp = x;
    x = y;
    y = temp;
}
void Sort_Cost(Product prd[], int n)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (prd[i].Cost > prd[j].Cost)
                Swap(prd[i], prd[j]);
}
void Output(Product prd[], int n)
{
    ShowTitle();
    for (int i = 0; i < n; i++)
        ShowInformationProduct(prd[i], i+1);
}
void ReadBinary(Product prd[], int *n)
{
    FILE *f = fopen(fbin, "rb");
    fseek(f, 0, SEEK_END); //Nhảy về cuối file, di chuyển đi 0 vị trí
    (*n) = (ftell(f)+1)/sizeof(Product); //ftell(); tra về vị trí hiện tại của con trỏ
    // SEEK_CUR: di chuyển bắt đầu từ vị trí hiện tại của con trỏ, chỉ dùng trong fseek()
    fseek(f, 0, SEEK_SET); //Nhảy về đầu file, di chuyển đi 0 vị trí
    fread(prd, sizeof(Product), (*n), f);
    fclose(f);
}
void Search_ID(Product prd[], int n, char id[])
{
    int check = 0;
    for (int i = 0; i < n; i++)
    {
        if (strcmp(id, prd[i].ProductID) == 0)
        {
            check = 1;
            ShowTitle();
            ShowInformationProduct(prd[i], i + 1);
            break;
        }
    }
}

```

```
        if (check == 0 && i == n - 1)
            printf("Not found Product have ProductID is %s !\n", id);
    }
}
void Search_Name(Product prd[], int n, char ProductName[])
{
    int check = 0;
    for (int i = 0; i < n; i++)
    {
        if (strcmp(ProductName, prd[i].ProductName) == 0)
        {
            check = 1;
            ShowTitle();
            ShowInformationProduct(prd[i], i + 1);
            break;
        }
        if (check == 0 && i == n - 1)
            printf("Not found Product have ProductName is %s !\n", ProductName);
    }
}
void SubMenu()
{
    printf("\n\t 1: ProductID");
    printf("\n\t 2: Name");
    printf("\n\t 3: Exit Search");
}
void Search(Product prd[], int n)
{
    int select;
    do
    {
        SubMenu();
        printf("\nEnter the type you want Search: ");
        scanf("%d", &select);
        switch(select)
        {
            case 1:
            {
                printf("Enter ProductID want Search: ");
                char id[11];
                gets(id);
                Search_ID(prd, n, id);
                break;
            }
            case 2:
            {
                printf("Enter ProductName want Search: ");
                char ProductName[51];
                gets(ProductName);
                Search_Name(prd, n, ProductName);
                break;
            }
            case 3: return;
            default: printf("Error select !");
        }
    } while (select != 3);
}
```

Bài hướng dẫn 4: Thực hiện bài tập 3 (**BÀI THỰC HÀNH 4**), Chương trình quản lý điểm của sinh viên?*// QLSV_Diem.cpp : Defines the entry point for the console application.**// Write by: Tran Van Tho*

#include <conio.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

struct DiemMH

{

float DTL;

float DGK;

float DCK;

float DTB;

};

typedef char KeyType;

struct KetQua

{

KeyType MaMH[11];

char TenMH[50];

int SoTC;

DiemMH Diem;

};

struct SNodeKQ

{

KetQua Info;

SNodeKQ *Next;

};

struct SListKQ

{

SNodeKQ *Head;

SNodeKQ *Tail;

};

struct SinhVien

{

KeyType MaSV[11];

char HoDem[25];

char Ten[8];

SListKQ DiemKQ;

};

//=====

typedef SinhVien ItemType;

struct SNode

{

ItemType Info;

SNode *Next;

};

struct SList

{

SNode *Head;

SNode *Tail;

};

//=====

```
SNodeKQ *CreateNodeKQ(KetQua x)
{
    SNodeKQ *p = new SNodeKQ;
    if(p == NULL)
    {
        printf("\nKhong du bo nho de cap phat SNode!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    return p;
}
//=====
SNode *CreateSNode(ItemType x)
{
    SNode *p = new SNode;
    if(p == NULL)
    {
        printf("\nKhong du bo nho de cap phat SNode!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Next = NULL;
    return p;
}
//=====
void InitSListKQ(SListKQ &s1)
{
    s1.Head = NULL;
    s1.Tail = NULL;
}
//=====
int IsEmpty(SListKQ s1)
{
    if(s1.Head == NULL)
        return 1;
    else
        return 0;
}
//=====
void InitSList(SList &s1)
{
    s1.Head = NULL;
    s1.Tail = NULL;
}
//=====
int IsEmpty(SList s1)
{
    if(s1.Head == NULL)
        return 1;
    else
        return 0;
}
//=====
int InsertTailSListKQ(SListKQ &s1, SNodeKQ* p)
```



```

{
    if(p == NULL) return 0;
    if(IsEmpty(s1) == 1) //danh sách kết quả học tập rỗng
        s1.Head = s1.Tail = p;
    else
    {
        s1.Tail->Next = p;
        s1.Tail = p;
    }
    return 1;
}
//=====
void NhapDiemKQ(KetQua &x)
{
    printf("\nCho biet ma mon hoc: ");
    fflush();
    gets(x.MaMH);
    printf("Cho biet ten mon hoc: ");
    fflush();
    gets(x.TenMH);

    /*Phát sinh ngẫu nhiên số tín chỉ
    srand((unsigned)time(NULL));
    x.SoTC = rand()%5+1;
    printf("So tin chi: %d", x.SoTC);*/

    printf("Cho biet so tin chi: ");
    scanf("%d", &x.SoTC);

    x.Diem.DTL = (rand()%101*0.1); //Phát sinh ngẫu nhiên điểm tiểu luận
    x.Diem.DCK = (rand()%101*0.1); //Phát sinh ngẫu nhiên cuối kỳ
    if(x.SoTC <= 2)
    {
        x.Diem.DGK = 0;
        x.Diem.DTB = x.Diem.DTL*0.3 + x.Diem.DCK*0.7;
    }
    else
    {
        x.Diem.DGK = (rand()%101*0.1); //Phát sinh ngẫu nhiên điểm giữa kỳ
        x.Diem.DTB = x.Diem.DTL*0.2 + x.Diem.DGK*0.3 + x.Diem.DCK*0.5;
    }
    printf("Diem mon hoc: %.1f\n", x.Diem.DTB);
}
//=====
void NhapDiemSV(SListKQ &s1)
{
    InitSListKQ(s1);
    int sm, kq;
    do
    {
        printf("\nCho biet so luong mon hoc: ");
        scanf("%d", &sm);
    }while(sm <= 0);
    for(int i = 1; i <= sm; i++)
    {
        KetQua x;
        NhapDiemKQ(x);
    }
}

```

```

        SNodeKQ* p = CreateNodeKQ(x);
        kq = InsertTailSlistKQ(s1, p);
        if(kq == 0)
            printf("\nKhong the them ket qua hoc tap nay!");
    }
}
//=====
void NhapTTSV(SinhVien &x)
{
    printf("\nCho biet ma sinh vien: ");
    fflush();
    gets(x.MaSV);
    printf("Cho biet ho va ho lot: ");
    fflush();
    gets(x.HoDem);
    printf("Cho biet ten: ");
    fflush();
    gets(x.Ten);
    NhapDiemSV(x.DiemKQ);
}
//=====
float TinhDTB(SListKQ slkq)
{
    int sotec = 0;
    float td = 0;
    for(SNodeKQ* p = slkq.Head; p; p = p->Next)
    {
        KetQua x = p->Info;
        sotec += x.SoTC;
        td += x.Diem.DTB * x.SoTC;
    }
    return (sotec > 0) ? (td/sotec) : 0;
}
//=====
void XuatKetQua(KetQua x)
{
    printf("%-12s%-35s%-10d%-8.1f%-8.1f%-8.1f%-8.1f", x.MaMH, x.TenMH, x.SoTC,
x.Diem.DTL, x.Diem.DGK, x.Diem.DCK, x.Diem.DTB);
}
//=====
void XuatDiemKQ(SListKQ slkq)
{
    printf("Ket qua hoc tap nhu sau:");
    int i = 1;
    printf("\n%-5s%-12s%-35s%-10s%-8s%-8s%-8s%-8s", "STT", "MSMH", "TEN", "MON",
HOC", "SO TC", "DTL", "DGK", "DCK", "DTB");
    for(SNodeKQ* p = slkq.Head; p != NULL; p = p->Next)
    {
        printf("\n%-5d", i++);
        XuatKetQua(p->Info);
    }
}
//=====
void XuatTTSV(SinhVien x)
{
    float dtb = TinhDTB(x.DiemKQ);
    printf("%-12s%-23s%-10s%10.1f\n", x.MaSV, x.HoDem, x.Ten, dtb);
}

```

```

        XuatDiemKQ(x.DiemKQ);
    }
}
//=====
int InsertTail(SList &s1, SNode* p)
{
    if(p == NULL) return 0;
    if(IsEmpty(s1) == 1)
        s1.Head = s1.Tail = p;
    else
    {
        s1.Tail->Next = p;
        s1.Tail = p;
    }
    return 1;
}
//=====
void CreateSList(SList &s1)
{
    int n, kq;
    do
    {
        printf("\nCho biet so luong sinh vien: ");
        scanf("%d", &n);
    }while(n <= 0);
    InitSList(s1);
    for(int i = 1; i <= n; i++)
    {
        SinhVien x;
        NhapTTSV(x);
        SNode* p = CreateSNode(x);
        kq = InsertTail(s1, p);
        if(kq == 0)
            printf("\nKhong the them sinh vien nay!");
    }
}
//=====
void ShowSList(SList s1)
{
    if(IsEmpty(s1) == 1)
    {
        printf("\nKhong co sinh vien nao!");
        return;
    }
    int i = 1;
    printf("\nDANH SACH SINH VIEN: ");
    printf("\n%-5s%-12s%-33s%-10s", "STT", "MA SO SV", "HO VA TEN SINH VIEN", "DTB");
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
    {
        printf("\n\n%-5d", i++);
        XuatTTSV(p->Info);
    }
}
//=====
SNode* TimTheoMaSV(SList s1, char pMaSV[])
{
    for(SNode* p = s1.Head; p != NULL; p = p->Next)
        if(stricmp(p->Info.MaSV, pMaSV) == 0)

```

```

        return p;
    return NULL;
}
//=====
void ShowMenu()
{
    printf("\n*****");
    printf("\n*                MENU                *");
    printf("\n*-----*");
    printf("\n* 1. Nhập thông tin sinh viên và điểm    *");
    printf("\n* 2. Xuất thông tin sinh viên và điểm    *");
    printf("\n* 3. Tìm thông tin sinh viên             *");
    printf("\n* 0. Thoát chương trình                 *");
    printf("\n*****");
}
//=====
void Process()
{
    ItemType x, y;
    KeyType masv[11];
    SNode *p, *q;
    int SelectFunction;
    SList sl;
    InitSList(sl);
    do
    {
        ShowMenu();
        printf("\nVui lòng chọn một chức năng: ");
        scanf("%d", &SelectFunction);
        switch(SelectFunction)
        {
            case 1:
                CreateSList(sl);
                ShowSList(sl);
                break;
            case 2:
                ShowSList(sl);
                break;
            case 3:
                printf("\nCho biết mã số sinh viên muốn tìm: ");
                fflush();
                gets(masv);
                q = TimTheoMaSV(sl, masv);
                if(q)
                    XuatTTSV(q->Info);
                else
                    printf("\nKhông có sinh viên nào có mã số %s", masv);
            }
        }while(SelectFunction != 0);
    }
//=====
void main()
{
    Process();
}

```

Bài hướng dẫn 5: Ứng dụng Stack để giải 1 số bài toán: Tính giai thừa, Đảo chuỗi, Quick Sort, Tháp Hà Nội?**a. Tính giai thừa:**

```

long int Stack_TinhGiaiThua(int n)
{ // Tính n!
    ItemType x;
    StackNode* p;
    Stack stack;
    InitStack(stack);
    while(n > 0)
    { // Thêm Lần Lượt nội dung vào stack
        p = CreateStackNode(n);
        if(Push(stack, p) == 0)
            break;
        n--;
    }
    long int kq = 1;
    while(1)
    { // Lấy Lần Lượt nội dung từ stack
        if(Pop(stack, x) == 0)
            break;
        kq *= x;
    }
    return kq;
}

```

b. Đảo chuỗi:

```

void Stack_DaoChuoi(char *chuoi)
{
    StackNode* p;
    Stack stack;
    InitStack(stack);
    int len = strlen(chuoi);
    for(int i = 0; i < len; i++)
    {
        p = CreateStackNode(chuoi[i]);
        Push(stack, p);
    }
    printf("\nChuoi dao nguoc la: ");
    while(!IsEmpty(stack))
    {
        ItemType ch;
        Pop(stack, ch);
        printf("%c", ch);
    }
}

```

c. Khử đệ quy QuickSort:

```

struct Item
{
    int Left, Right;
};
typedef Item ItemType;

```

```
//=====
ItemType CopyInfoToStackNode(int left, int right)
{
    ItemType temp;
    temp.Left = left;
    temp.Right = right;
    return temp;
}
//=====
void Stack_QuickSort(int a[], int Left, int Right, int &solan)
{
    Stack S;
    StackNode* p;
    ItemType M, N;
    int i, j, Mid, x;

    InitStack(S);
    M = CopyInfoToStackNode(Left, Right);
    p = CreateStackNode(M);
    Push(S, p);

    while(!IsEmpty(S))
    {
        Pop(S, N);
        i = Left = N.Left;
        j = Right = N.Right;
        Mid = (Left + Right)/2;
        x = a[Mid];
        do
        {
            while(a[i] < x) i++;
            while(a[j] > x) j--;
            if(i <= j)
            {
                Swap(a[i], a[j]);
                i++;
                j--;
            }
            solan++;
        }while(i < j);
        if(i < Right)
        {
            M = CopyInfoToStackNode(i, Right);
            p = CreateStackNode(M);
            Push(S, p);
        }
        if(j > Left)
        {
            M = CopyInfoToStackNode(Left, j);
            p = CreateStackNode(M);
            Push(S, p);
        }
    }
}
```

d. Khử đệ quy Tháp Hà Nội:

```
struct Item
```

```

{
    int SoDia, Nguon, Dich;
};
typedef Item ItemType;
//=====
ItemType SaoChepStackNode(int SoDia, int Nguon, int Dich)
{
    ItemType temp;
    temp.SoDia = SoDia;
    temp.Nguon = Nguon;
    temp.Dich = Dich;
    return temp;
}
//=====
void Stack_HanoiTower(int n, int A, int B, int C)
{
    int solan = 0;
    ItemType x, y;
    StackNode* p;
    Stack S;
    InitStack(S);
    y = SaoChepStackNode(n, A, C);
    p = CreateStackNode(y);
    Push(S, p);
    while(!IsEmpty(S))
    {
        Pop(S, x); //Biến x có 3 phần: {n, Nguon, Dich}
        if(x.SoDia == 1)
            printf("\nLan %2d: Chuyen tu cot %d den cot %d", ++solan, x.Nguon, x.Dich);
        else
        {
            int temp = 6 - (x.Nguon + x.Dich);
            y = SaoChepStackNode(x.SoDia - 1, temp, x.Dich);
            p = CreateStackNode(y);
            Push(S, p);
            y = SaoChepStackNode(1, x.Nguon, x.Dich);
            p = CreateStackNode(y);
            Push(S, p);
            y = SaoChepStackNode(x.SoDia - 1, x.Nguon, temp);
            p = CreateStackNode(y);
            Push(S, p);
        }
    }
}

```