


MỤC LỤC

Bài 1. Ôn tập C	1
Bài 2. Sắp xếp và Tìm kiếm	16
Bài 3. Sắp xếp và Tìm kiếm (tiếp theo)	24
Bài 4. Danh sách liên kết đơn	31
Bài 5. Danh sách liên kết đơn (tiếp theo)	42
Bài 6. Stack	48
Bài 7. Hàng đợi (Queue).....	55
Bài 8. Cây nhị phân	64
Bài 9. Cây nhị phân tìm kiếm	80
Bài 10. Cây nhị phân tìm kiếm (tiếp theo)	95
Bài 11. Bảng băm	99
Bài 12. Tổng kết	

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 1. ÔN TẬP C	
------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------	------------------------------------------------------------------------------------

A. MỤC TIÊU:

- Cài đặt được bài toán thiết lập và xử lý mảng 1 chiều, 2 chiều.
- Xây dựng được cấu trúc (struct) và các hàm xử lý trên struct theo yêu cầu bài toán.
- Cài đặt được bài toán tổ chức cấu trúc, mảng cấu trúc và các hàm xử lý trên mảng cấu trúc.
- Cài đặt được các hàm xử lý chuỗi.
- Viết chương trình theo dạng cấu trúc hàm con, phân loại hàm, tham số trong hàm, cách gọi hàm.

B. DỤNG CỤ - THIẾT BỊ THỰC HÀNH CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Mảng 1 chiều

- Khai báo mảng:

<Tên kiểu dữ liệu> <Tên mảng> [<Số phần tử>];

- Truy xuất phần tử mảng: **<Tên mảng>[<chỉ số phần tử>]**

Ví dụ: `int a [10] = {2, 5, 6, 1, 0, 6, 7}; // cấp phát 10 nhưng chỉ chèn 7 phần tử`

- Các thao tác xử lý trên mảng 1 chiều

- Duyệt mảng: dùng 1 vòng for
 - Tìm kiếm phần tử thỏa điều kiện (lồng if trong for)
 - Tính toán các phần tử thỏa điều kiện

- Thêm/xóa phần tử: lần lượt di chuyển các phần tử trong mảng để thêm/xóa phần tử theo yêu cầu.
 - Sắp xếp các phần tử: áp dụng các giải thuật sắp xếp đã học
- Chuỗi là mảng 1 chiều các ký tự, để sử dụng các hàm xử lý chuỗi trong C/C++ cần khai báo thư viện **string.h**.

2. Mảng 2 chiều

- Khai báo mảng:

<Tên kiểu dữ liệu> <Tên mảng> [<Số dòng>] [<Số cột>];

- Truy xuất phần tử mảng:

<Tên mảng>[<chỉ số dòng>] [<chỉ số cột>]

Ví dụ: `int a[3][5] = { {2, 5, 6, 1, 0},
 {0, 9, 4, 6, 7},
 {5, 3, 1, 8, 0}};`

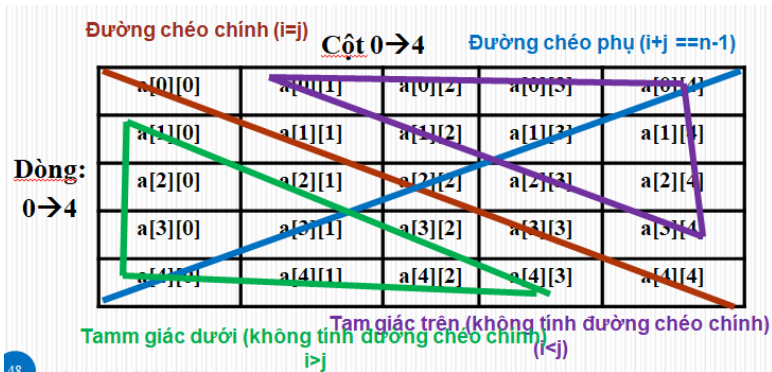
// cấp phát 10 nhưng chỉ chèn 7 phần tử

- Các thao tác xử lý trên mảng 2 chiều
- Duyệt mảng: dùng 2 vòng for
 - Tìm kiếm phần tử thỏa điều kiện (lồng if trong for)
 - Tính toán các phần tử thỏa điều kiện
 - Thao tác/duyet trên dòng cột, đường chéo chính phụ, các biên,...
 - Sắp xếp các phần tử: trên từng dòng, từng cột hoặc trên cả mảng.
- ***Ma trận vuông*** là mảng 2 chiều a có **số dòng bằng số cột**.
- ***Ma trận vuông*** có **số dòng và số cột bằng n** gọi là ***ma trận vuông cấp n***.
- Một số khái niệm trong ma trận vuông:

Cho ma trận vuông a cấp n.

- Đường chéo chính: gồm các phần tử $a[i][j]$ với $i=j, 0 \leq i, j \leq n$

- Đường chéo phụ: gồm các phần tử $a[i][j]$ với $i+j = n-1$, $0 \leq i, j \leq n$
- Tam giác trên: gồm các phần tử $a[i][j]$ với $i < j$, $0 \leq i, j \leq n$
- Tam giác dưới: gồm các phần tử $a[i][j]$ với $i > j$, $0 \leq i, j \leq n$



3. Cấu trúc

- Cấu trúc là cách cho người lập trình tự định nghĩa một kiểu dữ liệu mới bao gồm nhiều thành phần có thể thuộc nhiều kiểu dữ liệu khác nhau. Các thành phần được truy nhập thông qua một tên.

Cú pháp:

```
struct [tên_cấu_trúc]
{
    //khai báo các thành phần
};
```

Ví dụ:

```
struct SinhVien {
    char mssv[50];
    char hoTen[50];
    float diemTB;
};
```

- Các thành phần của cấu trúc được truy nhập thông qua tên biến cấu trúc và tên thành phần.

<Tên biến cấu trúc>.<tên_thành_phần>

Hoặc: *<tên biến con trỏ cấu trúc> -> <tên thành phần>*

4. Mảng cấu trúc

Cú pháp: *<tên cấu trúc> <tên mảng> [<kích thước>];*

Ví dụ

```
SinhVien dssv[100];  
puts("Nhap ma so sinh vien:"); //xuất chuỗi;  
thuộc thư viện stdio.h  
gets(a[2].mssv); // nhập chuỗi, gets khác scanf  
ở chỗ chấp nhận chuỗi các khoảng cách.  
puts("Nhap ho ten:");  
gets(a[2].hoten);  
printf("Nhap so Lan vang mat:");  
scanf("%d",a[2].solanvang);
```

Lưu ý: Cách dùng fflush (stdin)

Xét ví dụ sau:

```
int main() {  
    char ten[100];  
    int tuoi;  
    printf("Ban hay nhap tuoi cho sinh vien : \n");  
    scanf("%d",&tuoi);  
    printf("Ban hay nhap ten cho sinh vien : \n");  
    gets(ten);  
    printf("Thong tin sinh vien vua nhap la : \n");  
    printf("Tuoi cua sinh vien : %d\n",tuoi);  
    printf("Ten cua sinh vien : %s\n",ten);  
    return 0;  
}
```

Chúng ta có thể thấy sau khi nhập tuổi sinh viên thì trình biên dịch đã bỏ qua bước nhập tên cho sinh viên. Vào thời điểm chúng ta sử dụng lệnh scanf để nhập tuổi và kết thúc lệnh scanf bằng phím enter, thì trong bộ nhớ đệm của ta đã lưu kí tự “\n” làm trôi lệnh gets của chúng ta. Đây là lí do tại sao chúng ta cần sử dụng câu lệnh **fflush(stdin);**

5. Đọc/ghi file text (txt) trong C

5.1. Cách 1.

- Đọc/ghi file txt trong C, ta dùng các hàm đọc/ghi trong thư viện `stdio.h`
- Để đọc/ghi file cần khai báo con trỏ FILE , cú pháp mở file:

FILE *fp;

```
FILE *fp;
```

```
fp = fopen("fileopen","mode")
```

Ví dụ:

```
// mở file txt theo đường dẫn để ghi file
```

```
fopen("E:\\cprogram\\newprogram.txt","w");
```

```
// mở file txt theo đường dẫn để đọc file
```

```
fopen("E:\\cprogram\\oldprogram.bin","r");
```

Một số chuẩn mode để đọc/ghi file

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exist, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Ví dụ : Ghi file sử dụng fprintf()


```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;
    fptr = fopen("C:\\program.txt","w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);

    return 0;
}

```

đọc file sử dụng fscanf():

```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt","r")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    fscanf(fptr,"%d", &num);

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}

```

(Nguồn: <https://www.programiz.com/c-programming/c-file-input-output#example-write>)

5.2. Đọc/ghi file dùng thư viện fstream trong namespace std (C++)

- Khai báo thư viện : **#include <fstream>**
- Mở tạo file để đọc/ghi:

```

void open(const char *ten_file, ios::che_do);
//chế độ (phần này không bắt buộc có)

```

Chế độ	Miêu tả
ios::app	Chế độ Append. Tất cả output tới file đó được phụ thêm vào cuối file đó
ios::ate	Mở một file cho output và di chuyển điều khiển read/write tới cuối của file
ios::in	Mở một file để đọc
ios::out	Mở một file để ghi
ios::trunc	Nếu file này đã tồn tại, nội dung của nó sẽ được cắt (truncate) trước khi mở file

Ví dụ:

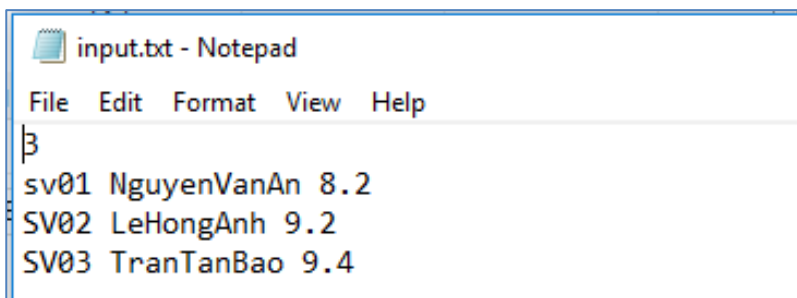
```
ofstream outfile;
outfile.open("file.txt", ios::out | ios::trunc );
//mở một file trong chế độ ghi và muốn cắt
(truncate) nó trong trường hợp nó đã tồn tại
fstream infile;
infile.open("file.txt", ios::out | ios::in );
// mở một file với mục đích đọc và ghi
```

II. Bài tập có hướng dẫn

Yêu cầu: dùng NNLT C/C++ để minh họa, viết chương trình theo dạng cấu trúc (hàm con), tổ chức chương trình theo dạng hiển thị menu các bài toán và cho phép người dùng lựa chọn bài toán cần thực thi.

Cho mảng cấu trúc sinh viên có n phần tử. Hãy đọc/ghi dữ liệu từ file txt. Biết rằng sinh viên có các thông tin mssv, họ tên và điểm trung bình.

1. Cách tổ chức file : input.txt



```
input.txt - Notepad
File Edit Format View Help
  sv01 NguyenVanAn 8.2
SV02 LeHongAnh 9.2
SV03 TranTanBao 9.4
```

Chương trình mẫu:

```
#include "stdafx.h"
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
using namespace std;

struct SinhVien {
    char mssv[50];
    char hoTen[50];
    float diemTB;
};

void xuatDSSV (SinhVien dssv[], int n);
void docFile(SinhVien dssv[], int &n);
void ghiFile (SinhVien dssv[], int n);

void main()
{
    SinhVien dssv[10];
    int n=0;
```

```

        docFile(dssv,n);
        xuấtDSSV(dssv, n);
        ghiFile(dssv,n);
        _getch();
    }
    void xuấtDSSV (SinhVien dssv[], int n)
    {
        for(int i=0; i<n; i++)
            printf(" %s\t%s\t%0.2f\n", dssv[i].mssv,
dssv[i].hoTen, dssv[i].diemTB);
    }
    void docFile(SinhVien dssv[], int &n)
    {
        ifstream in;
        // đường dẫn trực tiếp
        //char *filename =
"E:/4.WORKSPACE/WorkspaceC#/DocGhiFileTxt/Files/
input.txt";

        //đường dẫn gián tiếp
        char *filename = "../Files/input.txt";
        in.open(filename);

        if(in) //nếu file mở không bị lỗi
        {
            /* lần lượt đọc các giá trị trong file vào
cho các biến theo thứ tự, khi gặp khoảng trắng hoặc
xuống dòng là xem như qua 1 giá trị mới */

```

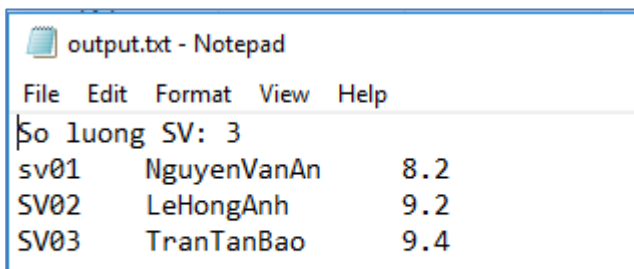
```

        in>>n; //đọc giá trị cho n
        for(int i=0; i<n; i++)
        {
            //đọc các thông tin cho từng SV
            in>>dssv[i].mssv;
            in>>dssv[i].hoTen;
            in>>dssv[i].diemTB;
        }
    }
    in.close();
}

void ghiFile (SinhVien dssv[], int n)
{
    ofstream outFile;
    char * filename = "../Files/output.txt";
    outFile.open(filename);
    if(outFile)
    {
        outFile<<"So luong SV: "<< n<<"\n";
        for(int i=0; i<n; i++)
        {
            outFile<<dssv[i].mssv<<"\t"<<dssv[i].hoTen<<"\t"
            <<dssv[i].diemTB<<"\n";
        }
    }
    outFile.close();
}

```

Kết quả file output.txt



output.txt - Notepad		
File	Edit	Format View Help
So luong SV: 3		
sv01	NguyenVanAn	8.2
SV02	LeHongAnh	9.2
SV03	TranTanBao	9.4

III. Bài thực hành trên lớp

Bài 1. Tạo cấu trúc sinh viên chứa các thông tin: mã số SV, ngày sinh, họ tên, địa chỉ, ngành học, điểm trung bình. Biết rằng ngày sinh thuộc cấu trúc Date có các thông tin (ngày, tháng, năm). Cho mảng 1 chiều a chứa n sinh viên. Hãy viết chương trình thực hiện các yêu cầu sau:

1. Đọc/ghi file từ dữ liệu file txt ($n > 10$, lưu ý: ghi file cần có đầy đủ thông tin điểm chữ và xếp loại).
2. Nhập/xuất thông tin 1 sinh viên.
3. Xuất danh sách sinh viên
4. Tính điểm chữ (theo tín chỉ) dựa vào điểm trung bình của SV, tính theo 2 cách sau:

Cách 1:

Trong học chế tín chỉ, thang điểm đánh giá được chia thành điểm không đạt và điểm đạt.

+) Điểm không đạt

- 0 - 3,9 thang điểm 10 tương đương điểm F= 0 thang điểm 4.

+) Điểm đạt

- 4,0 - 5,4 thang điểm 10 tương đương điểm D =1 thang điểm 4.
- 5,5 - 6,9 thang điểm 10 tương đương điểm C= 2 thang điểm 4.
- 7,0 - 8,4 thang điểm 10 tương đương điểm B =3 thang điểm 4.
- 8,5 - 10 thang điểm 10 tương đương điểm A= 4 thang điểm 4.

Cách 2:

Sử dụng thang điểm chữ nhiều mức trong đánh giá kết quả học tập của sinh viên	
Thang điểm chữ nhiều mức là thang điểm mà ở một số điểm chữ được chia thành các mức: B+ và B; C+ và C; D+ và D	
+) A	= 4,0 (từ 8,5 đến 10)
+) B+	= 3,5 (từ 8,0 đến 8,4)
+) B	= 3,0 (từ 7,0 đến 7,9)
+) C+	= 2,5 (từ 6,5 đến 6,9)
+) C	= 2,0 (từ 5,5 đến 6,4)
+) D+	= 1,5 (từ 5,0 đến 5,4)
+) D	= 1,0 (từ 4,0 đến 4,9)
+) F	= 0 (dưới 4,0)

5. Tính kết quả xếp loại cho từng SV.

a) Loại đạt:
A (8,5 – 10) Giỏi
B (7,0 – 8,4) Khá
C (5,5 – 6,9) Trung bình
D (4,0 – 5,4) Trung bình yếu
b) Loại không đạt: F (dưới 4,0) Kém

6. Xuất các SV thuộc ngành CNTT
7. Đếm số SV trên 22 tuổi (tính từ năm 2019).
8. Xuất thông tin các SV có quê quán ở “TPHCM” (trong địa chỉ có thông tin tỉnh thành).
9. Thống kê số SV giỏi của Khoa CNTT và Khoa CNTP.
10. Tìm các SV có điểm trung bình cao nhất
11. Tìm xem Khoa CNTT có MSSV “012015” không?
12. Trong danh sách có bao nhiêu ngành học?
13. Thống kê SV theo từng nhóm xếp loại.


IV. Bài thực hành về nhà

Bài 2. Tạo cấu trúc hàng hóa tạp hóa gồm các thông tin: mã hàng, tên hàng, giá bán, ngày sản xuất, (thuộc cấu trúc Date, tự tạo như bài 1), hạn sử dụng (đơn vị tính là tháng), số lượng tồn kho. Cho mảng 1 chiều a chứa n hàng hóa. Hãy viết chương trình thực hiện các yêu cầu sau:

1. Tạo file dữ liệu txt và đọc dữ liệu vào mảng a.
2. Ghi thông tin hàng hóa trong mảng a vào file txt

3. Nhập/xuất thông tin 1 hàng hóa.
4. Xuất các hàng hóa đã hết hạn (tính tại thời điểm tháng 2/2019)
5. Xuất thông tin các hàng hóa gần hết trong kho (số lượng < 5)
6. Đếm số hàng hóa có tên hàng chứa “BanhDanisa”
7. Sắp xếp danh theo hạn sử dụng (hạn gần đến hạn xa)
8. Tổng kho hàng có bao nhiêu mặt hàng?
9. Xuất các mặt hàng có giá bán đồng giá 40.000
10. Đếm số hàng mới sản xuất (trong vòng 1 tháng tính từ tháng 2/2019)
11. Xuất các mặt hàng có hạn sử dụng từ 2 đến 3 năm
12. Xóa các mặt hàng quá hạn sử dụng
13. Thêm 1 hàng hóa mới vào danh sách
14. Tìm mặt hàng có số lượng tồn kho nhiều nhất
15. Tìm mặt hàng có số lượng tồn kho lớn thứ 2 trong danh sách.

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 2. SẮP XẾP & TÌM KIẾM	
------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------	------------------------------------------------------------------------------------

A. MỤC TIÊU:

- Cài đặt được các giải thuật tìm kiếm và sắp trên mảng 1 chiều, 2 chiều, mảng struct.
- Vận dụng các giải thuật vào từng bài toán cụ thể
- Viết chương trình theo dạng cấu trúc hàm con, phân loại hàm, tham số trong hàm, cách gọi hàm.

B. DỤNG CỤ - THIẾT BỊ THỰC HÀNH CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Giải thuật tìm kiếm

▪ Tìm kiếm tuyến tính – Linear Search

Gọi x là giá trị cần tìm và a là mảng chứa dãy số dữ liệu gồm N phần tử. Các bước tiến hành như sau:

Bước 1: $i = 1$;

Bước 2: So sánh $a[i]$ với x, có 2 khả năng:

$a[i] = x$: Tìm thấy. Dừng.

$a[i] \neq x$: Sang bước 3

Bước 3: $i = i + 1$; //xét phần tử kế

Nếu $i > N$: hết mảng, không tìm thấy. Dừng.

Ngược lại: lặp lại bước 2.

▪ Tìm kiếm nhị phân – BinarySearch

Gọi x là giá trị cần tìm và a là mảng chứa dãy số dữ liệu gồm N phần tử đã được sắp, left và right là chỉ số đầu và cuối của đoạn cần tìm,

middle là chỉ số của phần tử nằm giữa của đoạn cần tìm. Các bước tiến hành như sau:

Bước 1: Khởi đầu tìm kiếm trên tất cả các phần tử

$left = 1; right = N;$

Bước 2: $middle = (left + right)/2;$

So sánh $a[middle]$ với x , có 3 khả năng có thể xảy ra:

$a[middle] = x$: Tìm thấy. Dừng.

$a[middle] > x$: Chuẩn bị tìm tiếp x trong dãy con $a_{left} \dots$

$a_{middle-1} : right = middle - 1$

$a[middle] < x$: Chuẩn bị tìm tiếp x trong dãy con $a_{middle+1} \dots$

$a_{right} : left = middle + 1.$

Bước 3:

Nếu $left \leq right$: dãy tìm kiếm hiện hành vẫn còn phần tử.

Lặp lại bước 2.

Ngược lại: dãy tìm kiếm hiện hành hết phần tử. Dừng.

2. Giải thuật sắp xếp

▪ Sắp xếp đổi chỗ trực tiếp – Interchange sort

Bước 1: $i = 1;$ //bắt đầu từ đầu dãy

Bước 2: $j = i + 1;$ //tìm các phần tử $a[j] < a[i], j > i$

Bước 3:

Trong khi $j \leq N$ thực hiện

Nếu $a[j] < a[i]$: $a[i] \leftrightarrow a[j];$ //xét cặp phần tử $a[i], a[j]$

$j = j + 1;$

Bước 4: $i = i + 1;$

Nếu $i < N$: Lặp lại Bước 2.

Ngược lại: Dừng.

▪ Sắp xếp chọn trực tiếp – Selection Sort

Bước 1: $i = 1;$ //bắt đầu từ đầu dãy

Bước 2: Tìm phần tử $a[\text{min}]$ nhỏ nhất trong dãy hiện hành từ $a[i]$ đến $a[N]$;

Bước 3: Hoán vị $a[\text{min}]$ và $a[i]$

Bước 4: Nếu $i < N - 1$: $i = i + 1$; Lặp lại bước 2.

Ngược lại: $N - 1$ phần tử đã được đưa về đúng vị trí.

Dừng.

▪ Sắp xếp nhanh – Quick Sort

Để sắp xếp dãy $a_1 a_2 \dots a_N$ giải thuật Quick Sort dựa trên việc phân hoạch dãy ban đầu thành 3 thành phần:

- Dãy con 1: Gồm các phần tử $a_1 \dots a_i$ có giá trị không lớn hơn x .
- Dãy con 2: Gồm các phần tử $a_i \dots a_N$ có giá trị không nhỏ hơn x .

Với x là giá trị của một phần tử tùy ý trong dãy ban đầu. Sau khi thực hiện phân hoạch, dãy ban đầu được chia thành 3 phần:

1. $a_k < x$, với $k = 1..i$;
2. $a_k = x$, với $k = i..j$;
3. $a_k > x$, với $k = j..N$;

Trong đó dãy con thứ 2 đã có thứ tự, nếu các dãy con 1 và 3 chỉ có 1 phần tử thì chúng cũng đã có thứ tự, khi đó dãy ban đầu đã được sắp. Ngược lại, nếu các dãy con 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ được sắp khi các dãy con 1, 3 có thứ tự. Để sắp xếp dãy con 1 và 1, lần lượt tiến hành phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày ...

Giải thuật để phân hoạch một dãy $a_1 \dots a_r$ thành 2 dãy con:

Bước 1: Chọn tùy ý 1 phần tử $a[k]$ trong dãy là giá trị mốc $1 \leq k \leq r$;

$X = a[k]$; $i = 1$; $j = r$;

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử $a[i]$, $a[j]$ nằm sai chỗ:

Bước 2a: Trong khi $a[i] < x$: $i++$;

Bước 2b: Trong khi $a[j] > x$: $j--$;

Bước 2c: Nếu $i < j$ Hoán vị ($a[i]$, $a[j]$)

Bước 3:

Nếu $i < j$: Lặp lại bước 2.

Nếu $i \geq j$: Dừng.

II. Bài tập hướng dẫn mẫu

Bài 1: Nhập 1 mảng số nguyên, tìm kiếm trong mảng có hay không một giá trị x bằng giải thuật tìm kiếm tuyến tính?

Bước 1: Tạo một Project mới.

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

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

```
#define MAXSIZE 1000
```

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

– Viết hàm nhập mảng số nguyên a có n phần tử:

```
void NhapMangSoNguyen(int a[], int &n)
{
    do
    {
        printf("Cho biet so phan tu cua mang: ");
        scanf("%d", &n);
        if(n <= 0)
            printf("Ban nhap sai roi. Vui long nhap lai!\n");
    } while(n <= 0);
    for(int i = 0; i < n; i++)
    {
        printf("Nhap phan tu a[%d] = ", i);
```

```

        scanf("%d", &a[i]);
    }
}

```

- Viết hàm xuất mảng số nguyên a có n phần tử:

```

void XuatMangSoNguyen(ItemType a[], int n)
{
    printf("\nNoi dung cua mang la: ");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
        //có khoảng trắng sau %d
}

```
- Viết hàm tìm kiếm trong mảng số nguyên a có phần tử nào có giá trị bằng x hay không?

```

int LinearSearch_For(int a[], int n, int x)
{
    //Tìm 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.

Bài 2: Nhập 1 mảng số nguyên, viết chương trình sắp xếp mảng với giải thuật đổi chỗ trực tiếp (**Interchange sort**).

Bước 1, 2, 3: Thực hiện tương tự bài 1.

Bước 4:

- Viết hàm nhập mảng và xuất mảng
- Viết hàm hoán vị (*đổi chỗ*) hai phần tử.

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

- Viết hàm sắp xếp mảng tăng dần bằng giải thuật đổi trực tiếp.

```
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])
                //Gọi hàm hoán vị 2 phần tử.
                Swap(a[i], a[j]);
}
```

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

III. Bài thực hành trên lớp

Bài 1. Cho mảng 1 chiều a chứa n số nguyên, viết các hàm xử lý sau:

1. Tạo a chứa các phần tử ngẫu nhiên/xuất a.
2. Tìm kiếm x trong a (trả về vị trí của x/ trả lời có hoặc không?) theo giải thuật Linear Search.
3. Sắp xếp a tăng dần theo giải thuật Interchange Sort.
4. Sắp xếp a giảm dần theo giải thuật Interchange Sort.
5. Tìm kiếm x trong a (trả về vị trí của x/ trả lời có hoặc không?) theo giải thuật Binary Search với mảng a tăng/giảm.

6. Sắp xếp a tăng dần theo giải thuật Selection Sort.
7. Sắp xếp a giảm dần theo giải thuật Selection Sort.
8. Sắp xếp a tăng dần theo giải thuật Quick Sort.
9. Sắp xếp a giảm dần theo giải thuật Quick Sort.

Bài 2. Cho ma trận vuông a cấp n, chứa số nguyên. Viết các hàm xử lý theo yêu cầu sau:

1. Tạo a chứa các phần tử ngẫu nhiên/xuất a.
2. Tìm x trên a theo Linear Search.
3. Sắp xếp a tăng dần theo từng dòng.
4. Sắp xếp a giảm dần theo từng cột.
5. Sắp xếp đường chéo chính a tăng dần/giảm dần
6. Sắp xếp a tăng dần theo kiểu zig zag (từ trái qua phải và từ trên xuống dưới)

IV. Bài thực hành về nhà

Bài 3. Tạo cấu trúc sinh viên chứa các thông tin: mã số SV, ngày sinh, họ tên, địa chỉ, ngành học, điểm trung bình. Biết rằng ngày sinh thuộc cấu trúc Date có các thông tin (ngày, tháng, năm). Cho mảng 1 chiều a chứa n sinh viên. Hãy viết chương trình thực hiện các yêu cầu sau:


1. Đọc/ghi file từ dữ liệu file txt ($n > 10$, lưu ý: ghi file cần có đầy đủ thông tin điểm chữ và xếp loại) .
2. Nhập/xuất thông tin 1 sinh viên.
3. Xuất danh sách sinh viên:
4. Sắp xếp danh sách giảm dần theo điểm trung bình.
5. Sắp xếp danh sách tăng dần theo họ tên
6. Sắp xếp danh sách tăng dần theo điểm chữ, những SV nào bằng nhau điểm chữ thì sắp xếp giảm dần theo điểm trung bình.

Bài 4. Tạo cấu trúc hàng hóa tạp hóa gồm các thông tin: mã hàng, tên hàng, giá bán, ngày sản xuất, (thuộc cấu trúc Date, tự tạo như bài 1), hạn sử dụng (đơn vị tính là tháng), số lượng tồn kho. Cho mảng 1 chiều a chứa n hàng hóa. Hãy viết chương trình thực hiện các yêu cầu sau:

1. Tạo file dữ liệu txt và đọc dữ liệu vào mảng a.

2. Ghi thông tin hàng hóa trong mảng a vào file txt
3. Nhập/xuất thông tin 1 hàng hóa.
4. Sắp xếp danh sách tăng dần theo ngày sản xuất.
5. Sắp xếp danh sách giảm dần theo số lượng tồn.
6. Sắp xếp danh sách tăng dần theo hạn sử dụng, những sản phẩm cùng hạn sử dụng thì sắp giảm dần theo ngày sản xuất.

--HẾT--

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 3. SẮP XẾP & TÌM KIẾM (tiếp theo)	
------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------	------------------------------------------------------------------------------------

A. MỤC TIÊU:

- Vận dụng được 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 trên mảng dữ liệu kiểu cấu trúc thành các chương trình cụ thể chạy được trên máy tính.
- Làm được các bài tập ở lớp, cố gắng hoàn thành thêm các bài tập ở nhà.

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

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

Sinh viên xem lại trong **BÀI THỰC HÀNH 1**

II. Bài tập hướng dẫn mẫu

Bài ví dụ 1: Một sinh viên cần lưu trữ những thông tin gồm: MaSV – chuỗi tối đa 10 ký tự, HoSV – chuỗi tối đa 25 ký tự, TenSV – chuỗi tối đa 7 ký tự, DiemTB – số thực.

- Khai báo cấu trúc dữ liệu để lưu trữ thông tin của 1 sinh viên.
- Nhập/xuất mảng một chiều chứa thông tin các sinh viên.
- Cho biết thông tin sinh viên có điểm trung bình lớn nhất?
- Tìm kiếm sinh viên nào có tên y có tồn tại trong mảng danh sách sinh viên hay không (với y nhập từ bàn phím)?

Bước 1: Tạo một Project mới.

Bước 2: Khai báo 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;
//Định nghĩa kiểu dữ liệu của khóa
struct SINHVIEN
{
    /* Định nghĩa kiểu dữ liệu để lưu thông tin cho 1 sinh viên */
    KeyType MaSV[11]; //Mã số sinh viên
    char HoSV[26]; //Họ và họ lót của sinh viên
    char TenSV[8]; //tên sinh viên
    float DiemTB; //Điểm trung bình
};
```

```
typedef SINHVIEN ItemType;
//Định nghĩa kiểu dữ liệu của 1 phần tử
```

Bước 5: Viết các hàm nhập, xuất danh sách sinh viên như sau:

- Viết hàm nhập thông tin của một sinh viên.

```
void NhapThongTinSV(ItemType &x)
{
    //Nhập thông tin của 1 sinh viên
    printf("\nNhap ma so sinh vien: ");
    fflush();
    gets(x.MaSV);
}
```

```

printf("Nhap ho sinh vien: ");
flushall();
gets(x.HoSV);
printf("Nhap ten sinh vien: ");
flushall();
gets(x.TenSV);
printf("Nhap diem trung binh sinh vien: ");
scanf("%f", &x.DiemTB);
}

//=====
Viết hàm xuất thông tin của một sinh viên.
void XuatThongTinSV(ItemType x)
{
    //Xuất thông tin của 1 sinh viên
    printf("%-15s%-20s%-
10s%5.1f", x.MaSV, x.HoSV, x.TenSV, x.DiemTB);
}

//=====
Viết hàm nhập thông tin của một danh sách có n sinh viên.
void NhapDanhSachSV(ItemType a[], int &n)
{
    //Nhập danh sách n sinh viên (kiểm tra điều
    kiện n > 0
}

//=====
Viết hàm xuất thông tin của một danh sách có n sinh viên.
void XuatDanhSachSV(ItemType a[], int n)
{
    //Xuất tiêu đề cho danh sách

```

```
//Xuất danh sách n sinh viên
```

```
}
```

Bước 7: Bổ sung thêm các hàm sau vào chương trình trên:

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

- Viết hàm tìm thông tin của sinh viên có điểm trung bình lớn nhất.

```
ItemType XuatThongTinSV_DiemTB_Max(ItemType a[],  
int n)
```

```
{
```

```
//Trả về sinh viên có điểm trung bình cao nhất
```

```
}
```

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

Viết hàm tìm có hay không một sinh viên có một tên bất kỳ được nhập vào từ bàn phím.

```
int LinearSearch_TheoTenSV(ItemType a[], int n,  
char x[])
```

```
{
```

```
/*Trả về vị trí xuất hiện đầu tiên của sinh viên  
x trong danh sách. Nếu không tồn tại trả về -1*/
```

```
}
```

Bước 8: Viết hàm main để thực thi chương trình.

Bài 2: Trên cơ sở Bài ví dụ 1. Hãy:

- Viết lại chương trình dưới dạng hiển thị thực đơn (*menu*).
- Bổ sung thêm chức năng sắp xếp danh sách sinh viên tăng dần theo tên bằng giải thuật sắp xếp đổi chỗ trực tiếp.

III. Bài tập ở lớp

Bài 1. Dựa vào Bài ví dụ mẫu 1&2. Hãy chỉnh sửa và bổ sung vào chương trình với cấu trúc dữ liệu và các chức năng như sau:

- Khai báo CTDL cho một sinh viên (SV) gồm các thông tin sau:

MaSV : chuỗi tối đa 10 kí tự

HoLotSV : chuỗi tối đa 25 kí tự
 TenSV : chuỗi tối đa 8 kí tự
 Lop : chuỗi tối đa 10 kí tự
 DiemTB : số thực

- b. Đọc file từ dữ liệu file txt chứa thông tin n phần tử, mỗi phần tử lưu trữ thông tin một sinh viên (với $0 < n < 1000$).
- c. Tìm kiếm sinh viên lần lượt theo MaSV, TenSV, Lop bằng giải thuật tìm kiếm tuyến tính.
- d. Tìm kiếm sinh viên lần lượt theo MaSV, TenSV, Lop bằng giải thuật tìm kiếm nhị phân (giả sử mảng đã được sắp xếp tăng dần theo từng trường tương ứng).
- e. Sắp xếp mảng tăng dần theo MaSV bằng các giải thuật: **Interchange Sort, Selection Sort, Quick Sort.**
- f. Sắp xếp mảng tăng dần theo TenSV bằng các giải thuật: **Interchange Sort, Selection Sort, Quick Sort.**
- g. Sắp xếp mảng tăng dần theo Lop bằng các giải thuật: **Interchange Sort, Selection Sort, Quick Sort.**
- h. Sắp xếp mảng giảm dần theo DiemTB bằng các giải thuật: **Interchange Sort, Selection Sort, Quick Sort.**
- i. Sắp xếp mảng tăng dần theo TenSV, nếu trùng tên thì sắp xếp tăng dần theo HoSV bằng các giải thuật: **Interchange Sort, Selection Sort, Quick Sort.**
- j. In ra màn hình danh sách những sinh viên của lớp x.
- k. In ra danh sách những sinh viên có điểm trung bình cao nhất của từng lớp.
- l. Thêm một sinh viên x vào vị trí vt trong danh sách (x và vt do người dùng nhập vào từ bàn phím).
- m. Xóa sinh viên có mã số sinh viên nhập vào từ bàn phím.

IV. Bài tập về nhà

Bài 2. 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 kí tự), Tên sách (TenS: chuỗi tối đa 50 kí tự), Năm xuất bản (NamXB: số nguyên dương 2 bytes).

Sử dụng mảng 1 chiều hãy viết chương trình quản lý các cuốn sách của thư viện. Thực hiện những yêu cầu sau:

- Xây dựng cấu trúc dữ liệu SACH theo mô tả.
- Nhập vào mảng một chiều **a** gồm **n** phần tử, mỗi phần tử là thông tin của một SACH (với $0 < n < 1000$).
- Tìm cuốn sách có MaS = **x** trong mảng.
- In ra màn hình tất cả các SACH có năm xuất bản là **y**.
- Sắp xếp mảng tăng dần theo MaS (lần lượt với từng giải thuật sau: **Interchange Sort, Selection Sort, Quick Sort**).
- Sắp xếp mảng giảm dần theo NamXB (lần lượt với từng giải thuật sau: **Interchange Sort, Selection Sort, Quick Sort**).
- Tìm cuốn sách có MaS = **x** sau khi mảng đã sắp tăng dần theo MaS (HD: dùng giải thuật tìm kiếm nhị phân).
- Sắp xếp giảm dần theo năm xuất bản (NamXB), nếu NamXB bằng nhau thì sắp xếp tăng dần theo tên sách (TenS).

Bài 3. Khai báo CTDL cho một trường học (TRUONGHOC) gồm các thông tin:

MaT (Mã trường)	: chuỗi tối đa 10 kí tự
TenT (Tên trường)	: chuỗi tối đa 50 kí tự
NamTL (Năm thành lập)	: số nguyên dương 2 bytes


Cài đặt các hàm sau:

- Nhập/xuất 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, Selection Sort, Quick Sort**).
- Sắp xếp mảng giảm theo TenT (bằng giải thuật: **Interchange Sort,**

Selection Sort, Quick Sort).

- f. Sắp xếp mảng tăng theo NamTL (bằng giải thuật: **Interchange Sort, Selection Sort, Quick Sort**).
- g. Tìm trường học có MaT = x (sau khi mảng đã sắp tăng theo MaS).
- h. Tìm trường học có TenT = y (nếu có nhiều trường học trùng tên thì chỉ cần tìm trường học đầu tiên).
- i. Liệt kê ra màn hình danh sách những trường học có năm thành lập từ z đến nay.

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 4. DANH SÁCH LIÊN KẾT ĐƠN	
------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------	------------------------------------------------------------------------------------

A. MỤC TIÊU:

- Trình bày được hoạt động của danh sách liên kết đơn: thêm, xóa, tìm kiếm
- Xây dựng được danh sách liên kết đơn với các thao tác cơ bản như: khởi tạo, thêm, tìm kiếm, xử lý tính toán, xuất nội dung ra màn hình.
- Hoàn thành được các bài tập tại lớp, tự làm được các bài tập về nhà

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

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1.1. Khái niệm nút

1.2.1. Cấu tạo nút

info **Link**

- Thành phần dữ liệu (**info**): Lưu trữ các thông tin về bản thân phần tử.
- Thành phần nối 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.

1.2.2. Khai báo nút

```
typedef int ItemType;
//ItemType cũng có thể là kiểu
float, SINHVIEN, PHANSO
```

```

struct Snode //Single Node
{
    ItemType info;
    //Lưu thông tin của nút hiện hành.
    SNode* next; //Con trỏ chỉ đến nút kế sau.
};

```

1.2.3. Tạo nút chứa giá trị x

```

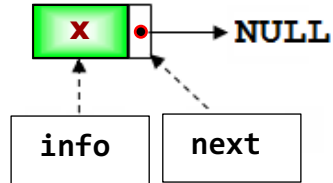
SNode* createSNode(ItemType x)

```

```

{
    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;
}

```



1.2.4. Xuất nội dung của nút

```

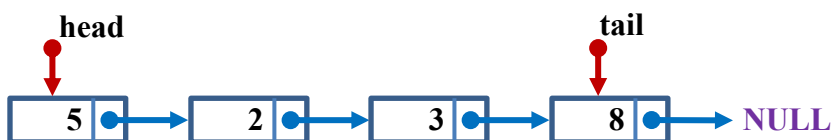
void showSNode(SNode* p)
{
    printf("%d ", p->info);
}

```

1.2. Danh sách liên kết đơn

1.2.1. 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**.



- Khai báo kiểu dữ liệu SList để lưu trữ danh sách liên kết đơn, sử dụng 2 con trỏ Head để quản lý nút đầu và Tail để quản lý nút cuối:

```

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
};
  
```

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

a. Khởi tạo danh sách

Do danh sách chưa có phần tử nào, nên con trỏ đầu head (*lưu địa chỉ của nút đầu tiên*) và con trỏ cuối tail (*lưu địa chỉ của nút cuối cùng*) đều bằng NULL.



b. Kiểm tra danh sách rỗng

Kiểm tra danh sách liên kết đơn đã có phần tử nào hay chưa? Nghĩa là kiểm tra con trỏ đầu Head (*hoặc con trỏ cuối Tail*) có bằng NULL hay không? Hàm trả về 1 nếu danh sách liên kết đơn chưa có phần tử nào (*rỗng*), ngược lại (*không rỗng*) thì hàm sẽ trả về 0.

c. Duyệt danh sách để xuất nội dung ra màn hình

- **Bước 1:** Nếu danh sách rỗng thì: Thông báo danh sách rỗng và không thực hiện.

- **Bước 2:** Gán $p = \text{Head}$; // p lưu địa chỉ của phần tử đầu trong SList

- **Bước 3:**

Lặp lại trong khi (con trỏ p còn khác NULL) thì thực hiện:

+ Xuất nội dung của phần tử tại con trỏ p .

+ $p = p \rightarrow \text{Next}$; // Xét phần tử kế

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

- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.

- **Bước 2:** Nếu SList rỗng thì

+ $\text{Head} = p$;

+ $\text{Tail} = p$;

- **Bước 3:** Ngược lại

+ $p \rightarrow \text{Next} = \text{Head}$;

+ $\text{Head} = p$;

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

- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.

- **Bước 2:** Nếu SList rỗng thì:

+ $\text{Head} = p$;

+ $\text{Tail} = p$;

- **Bước 3:** Ngược lại

+ Tail→Next = p;

+ Tail = p;

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

- **Bước 1:** Nếu phần tử q hoặc phần tử muốn thêm p không tồn tại thì không thực hiện.

- **Bước 2:**

+ p→Next = q→Next;

+ q→Next = p;

+ Nếu q là phần tử cuối thì: Tail = p;

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

- **Bước 1:** Nếu danh sách rỗng thì: trả về NULL;

- **Bước 2:** Gán p = Head; *//địa chỉ của phần tử đầu tiên trong DSLK đơn*

- **Bước 3:**

Lặp lại trong khi (p != NULL và p→Info != x) thì thực hiện:

p = p→Next; *//xét phần tử kế sau*

- **Bước 4:** Trả về p; *//nếu (p != NULL) thì p lưu địa chỉ của phần tử có khóa bằng x, hoặc NULL là không có phần tử cần tìm.*

h. **Tạo danh sách (nhập giá trị từ bàn phím)**

```
void CreateSList(SList &s1)
```

```
{
```

```
    int n;
```

```
    ItemType x;
```

```
    InitSList(s1);
```

```
    do
```

```
    {
```

```
        printf("Sophan tu cua danh sach (n>0):");
```

```

        scanf("%d", &n);
    } while(n <= 0);
    for(int i = 1; i <= n; i++)
    {
        printf("Nhập phần tử thứ %d là: ", i);
        scanf("%d", &x);
        SNode* p = CreateSNode(x);
        InsertTail(s1, p);
        //Hoặc chèn vào đầu InsertHead(s1, p);
    }
}

```

i. Tạo danh sách tự động

```

void CreateAutoSList(SList &s1)
{
    int n;
    ItemType x;
    InitSList(s1);
    do
    {
        printf("Số phần tử của danh sách (n>0):");
        scanf("%d", &n);
    } while(n <= 0);
    srand(time(NULL));
    //Thư viện stdlib.h và time.h
    for(int i = 1; i <= n; i++)
    {
        //Tạo 1 số ngẫu nhiên trong đoạn [-99, 99]
        x = (rand()%199)-99;
    }
}

```

```

    SNode* p = CreateSNode(x);
    //Hoặc chèn vào đầu InsertHead(sl, p);
    InsertTail(sl, p);
}
}

```

II. Bài tập hướng dẫn mẫu

Bài 1: Xây dựng một danh sách liên kết đơn chứa các số nguyên.

- Cài đặt các phương thức thêm đầu, thêm cuối và thêm sau một phần tử vào DSLK
- Cài đặt phương thức cho biết phần tử x có tồn tại trong danh sách hay không?
- Cài đặt phương thức xóa phần tử của DSLK: xóa đầu, xóa sau
- Cài đặt phương thức xuất danh sách ra màn hình
- Xây dựng hàm main (dạng menu) minh họa

Bước 1: Tạo một Project mới.

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

```

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

```

Bước 3: Định nghĩa các kiểu dữ liệu: thông tin của một nút, kiểu nút. Định nghĩa kiểu dữ liệu SList

```

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
    };

```

Bước 4: Viết các hàm cơ bản xử lý nút:

```

//=====
Viết hàm tạo mới (cấp phát) một nút lưu trữ giá trị là x.
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;
}
//=====

```


- Viết hàm hiển thị thông tin của một nút.

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

Bước 5: Viết các hàm cơ bản để thực hiện chương trình:

//=====

Viết hàm hiển thị menu của chương trình.

```
void ShowMenu()
{
    printf("\n*****");
    printf("\n*          MENU          *");
    printf("\n*-----*");
    printf("\n*1. Tao nut va xuat ra noi dung *");
    printf("\n* 2. Huy nut vua tao          *");
    printf("\n* 0. Thoat                    *");
    printf("\n*****");
}
```

III. Bài tập ở lớp

Bài 1. Sinh viên thực hiện lại Bài tập mẫu. Sau đó hãy bổ sung thêm các chức năng sau:

- Thêm phần tử mới vào cuối danh sách *sl*.
- Chèn thêm phần tử có giá trị *x* vào trước phần tử có giá trị *y*.
- Viết hàm xóa các phần tử lớn hơn *x* trong *dslk*
- Viết hàm xóa các phần tử chẵn trong *dslk*
- Sắp xếp *dslk* tăng dần, giảm dần
- Cho biết trong *dslk* có bao nhiêu số nguyên tố
- Tính tổng các số chính phương trong *dslk*
- Tìm phần tử nhỏ nhất, phần tử lớn nhất trong *dslk*

- i. Cho biết trong dskl có bao nhiêu phần tử lớn hơn gấp đôi phần tử phía sau nó.
- j. Từ *sl* tạo 2 danh sách mới: *s11* chứa các số chẵn, *s12* chứa các số lẻ.

Hướng dẫn:

- + Khởi tạo 2 danh sách *s11* và *s12* rỗng.
- + Lặp lại trong khi *sl* còn phần tử, kiểm tra:
 - Nếu phần tử có giá trị chẵn thì thêm vào cuối *s11*.
 - Nếu phần tử có giá trị lẻ thì thêm vào cuối *s12*.

Bài 2. Cho danh sách đơn *sl* chứa các phân số, biết mỗi phân số có 2 thành phần là tử số (TuSo) và mẫu số (MauSo). Hãy viết chương trình cho phép:

- 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 lớn hơn 1 trong danh sách liên kết
- h. Viết hàm trả về Snode chứa phân số *p* trong danh sách liên kết. Nếu không có *p* trong DSLK thì trả về NULL

IV. Bài tập về nhà

Bài 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.
- c. Nối danh sách *s12* vào sau phần tử có giá trị *x* trong danh sách *s11* – nếu không có phần tử *x* thì thông báo “không có phần tử *x*” với *s12* là danh sách chứa các số nguyên

Bài 4. Cho 2 danh sách đơn chứa các số nguyên là *s11* và *s12* có thứ tự


tăng. Cài đặt các hàm sau:

- a. Trộn 2 danh sách trên thành một danh sách *sl* có thứ tự tăng.

Hướng dẫn:

- + Khởi tạo danh sách *sl* rỗng.
 - + Lặp lại trong khi *sl1* và *sl2* vẫn còn phần tử, kiểm tra:
 - Nếu phần tử của *sl1* có giá trị nhỏ hơn hoặc bằng phần tử của *sl2* thì thêm phần tử này vào cuối *sl*.
 - Ngược lại: Nếu phần tử của *sl2* có giá trị nhỏ hơn phần tử của *sl1* thì thêm phần tử này vào cuối *sl*.
 - + Nếu danh sách nào còn phần tử thì thêm tất cả các phần tử đó vào cuối *sl*.
- 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.

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 5. DANH SÁCH LIÊN KẾT ĐƠN (tiếp theo)	
------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------	------------------------------------------------------------------------------------

A. MỤC TIÊU:

- Cài đặt được danh sách liên kết đơn để lưu dữ liệu là các cấu trúc do người dùng định nghĩa
- Xử lý tính toán trên danh sách liên kết đơn đã xây dựng

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

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

Một số phương thức trên danh sách liên kết đơn (tiếp theo)

a. Xóa nút đầu danh sách

- **Bước 1:** Nếu danh sách rỗng thì: Không thực hiện gì cả.
 - **Bước 2:** Khi (Head != NULL) thì: thực hiện các công việc sau:
 - + **Bước 2.1:** p = Head;
 - + **Bước 2.2:** Head = Head→Next;
 - + **Bước 2.3:** Nếu (Head == NULL) thì: Tail = NULL;
 - + **Bước 2.4:** Lưu lại thông tin nút bị xóa;
 - + **Bước 2.5:** delete p;
- //Hủy nút do p trở đến (con trở p)*

b. Xóa nút cuối danh sách

- **Bước 1:** Nếu (Tail == NULL) thì: không thực hiện gì cả.
- **Bước 2:** Khi (Tail != NULL) thì: thực hiện các công việc sau:
 - + **Bước 2.1:** Gán q = Head; và p = Tail;

- + **Bước 2.2:** Nếu $(p == q)$ thì: $Head = Tail = NULL$;
- + **Bước 2.3:** Ngược lại: Tìm đến nút kế trước nút Tail.
- + **Bước 2.4:** $Tail = q$;
- + **Bước 2.5:** $Tail \rightarrow Next = NULL$;
- + **Bước 2.6:** Lưu lại thông tin nút bị xóa;
- + **Bước 2.7:** delete p; //Hủy nút do p trở đến

c. Xóa một nút sau nút q của danh sách

- **Bước 1:** Nếu $(q == NULL \text{ hoặc } q \rightarrow Next == NULL)$ thì: không thực hiện.
- **Bước 2:** Khi $(q != NULL \text{ và } q \rightarrow Next != NULL)$ thì: thực hiện các công việc sau:
 - + **Bước 2.1:** $p = q \rightarrow Next$;
 - + **Bước 2.2:** $q \rightarrow Next = p \rightarrow Next$;
 - + **Bước 2.3:** Nếu $(p == Tail)$ thì: $Tail = q$;
 - + **Bước 2.4:** Lưu lại thông tin nút bị xóa;
 - + **Bước 2.5:** delete p; //Hủy nút do p trở đến

d. Xóa nút có giá trị x của danh sách

- **Bước 1:** Nếu danh sách rỗng thì: Không thực hiện gì cả.
- **Bước 2:** Tìm phần tử p có khóa bằng x, và q là phần tử đứng kế trước p.
- **Bước 3:** Nếu $(p == NULL)$ thì: Không có nút chứa x nên không thực hiện.
- **Bước 4:** //Ngược lại $(p != NULL) \rightarrow$ nghĩa là tồn tại nút p chứa khóa x
 - + Nếu $(p == Head)$ thì: //p là nút đầu danh sách \rightarrow xóa đầu
DeleteHead(sl, x);
 - + Ngược lại:
DeleteAfter(sl, q, x); //Xóa nút p chứa x kế sau nút q

e. Xóa toàn bộ danh sách

- **Bước 1:** Nếu danh sách rỗng thì: Không thực hiện gì cả.
- **Bước 2:** Lặp lại trong khi (danh sách còn phần tử) thì thực hiện lần lượt những việc sau:
 - + **Bước 2.1:** Gán $p = \text{Head}$;
 - + **Bước 2.2:** Gán $\text{Head} = \text{Head} \rightarrow \text{Next}$; //Cập nhật con trỏ Head
 - + **Bước 2.3:** delete p; //Hủy nút do p trỏ đến
- **Bước 3:** Gán $\text{Tail} = \text{NULL}$; //bảo toàn tính nhất quán khi danh sách rỗng

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

- Do danh sách liên kết đơn chỉ có một chiều đi từ phần tử đầu tới phần tử kế tiếp nên chỉ áp dụng được các thuật toán sắp xếp thỏa tính chất như vậy như: InterchangeSort, SelectionSort, ...
- Có thể thay thế $a[i]$ trong mảng bằng $i \rightarrow \text{Info}$, $a[i+1]$ thay thế bằng $(i \rightarrow \text{Next}) \rightarrow \text{Info}$, $i++$ thì thay bằng $i = i \rightarrow \text{Next}$
- Các bước của thuật toán áp dụng giống như trong mảng.

II. Bài tập ở lớp

Bài 1. Xây dựng cấu trúc danh sách liên kết đơn lưu các số nguyên và cài đặt thao thác

- a. Nhập danh sách liên kết đơn từ file text chứa các số nguyên, các số cách nhau bằng một khoảng trắng (không cho biết số lượng phần tử): VD: 2 3 4 5 12 13
- b. Xóa phần tử đầu
- c. Xóa phần tử sau một phần tử đã cho
- d. Xóa các phần tử có giá trị bằng x
- e. Xóa toàn bộ danh sách.
- f. Xóa các phần tử chẵn trong danh sách liên kết.

Bài 2. Thông tin bài hát gồm

- Tên bài hát là một chuỗi tối đa 50 ký tự

- Tác giả là một chuỗi tối đa 40 ký tự
- Ca sĩ là một chuỗi tối đa 40 ký tự
- Thời lượng là một số nguyên (đơn vị là giây)

Sử dụng danh sách liên kết đơn lưu trữ và quản lý một playlist nhạc với các chức năng như sau:

- Đọc danh sách list nhạc từ file định dạng txt
- In danh sách các bài hát ra màn hình
- Để nghe hết tất cả bài hát trong playlist phải cần bao nhiêu thời gian.
- Thêm một bài nhạc mới vào đầu playlist/cuối playlist
- Xóa một bài nhạc khỏi danh sách
- Kiểm tra xem bài hát có tên X có trong playlist không?
- Sắp xếp các bài hát trong playlist theo thứ tự từ điển của tên bài hát
- Sắp xếp các bài hát thứ tự giảm dần của tên ca sĩ
- Đưa một bài hát trong playlist lên đầu

Bài 3. Hãy định nghĩa một DSLK đơn *sl* để quản lý điểm của một lớp học trong một học kỳ, biết rằng thông tin của mỗi sinh viên trong lớp học gồm có:

- Mã số sinh viên (maSV) – chuỗi tối đa 10 ký tự,
- Họ đệm (hoDem) – chuỗi tối đa 25 ký tự,
- Tên sinh viên (tenSV) – chuỗi tối đa 8 ký tự,
- Năm sinh (namSinh) – số nguyên dương
- Điểm kết quả học tập (diemKQ).

Mỗi học kỳ sinh viên phải học nhiều môn học lý thuyết gồm các thông tin:

- Mã môn học, tên môn học, số tín chỉ
- Điểm tiểu luận (DTL)
- Điểm cuối kỳ (DCK),

– Điểm môn học: được xác định bằng 30% DTL + 70% DCK

Lưu ý: **Điểm kết quả học tập** của sinh viên là trung bình cộng điểm các môn học sinh viên đã đăng ký với trọng số là số tín chỉ.

Hãy viết một chương trình quản lý sinh viên theo mô tả với các chức năng như sau:

- a. Cho biết họ tên và điểm kết quả học tập 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 sinh viên.
- e. Thêm một sinh viên sao cho vẫn giữ nguyên thứ tự tăng dần của mã số sinh viên (*có kiểm tra trùng khóa*).
- f. Xóa sinh viên 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 kết quả học tập.
- i. In danh sách các sinh viên được xếp loại khá (sinh viên xếp loại khá nếu thỏa điều kiện: $7.0 \leq \text{điểm kết quả học tập} \leq 8.5$)
- j. Cho biết sinh viên có điểm kết quả học tập cao nhất.
- k. Cho biết sinh viên có điểm kết quả học tập thấp nhất.
- l. Cho biết sinh viên có điểm kết quả học tập thấp nhất trong số các sinh viên xếp loại giỏi.
- m. Cho biết sinh viên có điểm kết quả học tập gần **x** nhất (*x là số thực*).

III. Bài tập về nhà

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

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

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/trừ/nhân 2 đa thức P và Q để tạo ra một đa thức mới.

Bài 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 $\in [0, 9]$*).

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 6. STACK	
------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------	--

A. MỤC TIÊU:

- Định nghĩa, cài đặt được cấu trúc Stack, Queue sử dụng danh sách liên kết đơn
- Vận dụng được Stack, Queue trong một số bài toán cụ thể.

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

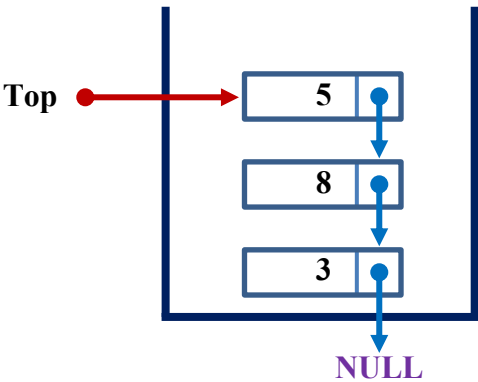
STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

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).



2. Thao tác trên Stack sử dụng DSLK đơn

a. Khai báo kiểu dữ liệu 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;
};

```

b. Tạo nút mới cho Stack: Tương tự như tạo Snode trong danh sách liên kết đơn.

c. Khởi tạo Stack

Sau khi khởi tạo thì Stack phải rỗng, nên con trỏ Top sẽ trỏ NULL.



d. Kiểm tra Stack rỗng.

Kiểm tra Stack có rỗng hay không? Nếu rỗng nghĩa là con trỏ Top có giá trị NULL, sẽ trả về 1, ngược lại trả về 0.

e. Thêm phần tử mới p có giá trị x vào Stack (Push): tương tự phương thức thêm một phần tử và đầu danh sách liên kết

- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.

- **Bước 2:** Nếu Stack rỗng thì

+ Top = p;

- **Bước 3:** Ngược lại

+ p → Next = Top;

+ Top = p;

f. **Lấy giá trị và hủy phần tử đầu Stack (Pop):** tương tự phương thức xóa phần tử đầu của danh sách liên kết

- **Bước 1:** Nếu ngăn xếp rỗng thì: Không thực hiện gì cả.

- **Bước 2:** Khi (Top != NULL) thì: thực hiện các công việc sau:

+ **Bước 2.1:** p = Top;

+ **Bước 2.2:** Top = Top → Next;

+ **Bước 2.3:** Lưu lại thông tin nút bị xóa vào một tham số x;

+ **Bước 2.4:** delete p;

//Hủy nút do p trở đến (con trỏ p)

g. **Lấy giá trị của phần tử ở đầu Stack**

- **Bước 1:** Nếu Stack rỗng thì: Không thực hiện gì cả.

- **Bước 2:** Khi (Top != NULL) thì: thực hiện công việc sau:

+ Lấy thông tin của nút ở đỉnh của Stack gán vào một tham số x để sử dụng.

h. **Xem nội dung của Stack**

- **Bước 1:** Nếu ngăn xếp rỗng thì: Thông báo ngăn xếp rỗng và không thực hiện.

- **Bước 2:** Gán p = Top; //p lưu địa chỉ của phần tử đỉnh Stack

- **Bước 3:**

Lặp lại trong khi (con trỏ p còn khác NULL) thì thực hiện:

+ Xuất nội dung của phần tử tại con trỏ p.
 + $p = p \rightarrow \text{Next}$; //Xét phần tử kế

II. Bài tập hướng dẫn mẫu

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_<Họ 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:

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

Viết hàm tạo mới (cấp phát) một nút thông tin x.

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

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

Viết hàm khởi tạo stack (làm rỗng stack).

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

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

Viết hàm kiểm tra stack có rỗng hay không.

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

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

Viết hàm thêm một nút p vào vị trí đỉnh của stack.

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

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

Viết hàm lấy ra và hủy một nút ở vị trí đỉnh của stack.

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

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

Viết hàm ứng dụng stack để đổi cơ số của một số nguyên dương n từ hệ 10 sang hệ a.

```
void Stack_DoiCoSo(int n, int a): xem lại thuật  
toán trong slide lý thuyết
```

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

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

```
void main()
```

```
{
```

```
    int n, coso;
```

```
    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", &coso);
```

```

}while(coso <= 0);
Stack_DoiCoSo(n, coso);
getch();
}

```

III. Bài tập ở lớp

Bài 1. Đổi cơ số cho 1 số nguyên n từ hệ 10 sang hệ 16

Bài 2. Ứng dụng Stack (*cài đặt theo dạng danh sách liên kết đơn*) để viết chương trình thực hiện việc kiểm tra sự hợp lệ của các cặp dấu ngoặc.

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

Ví dụ:

- Đúng: $((()))\{([()])\}$
- Sai: $)(())$
- Sai: $(\{[])\}$

Viết giải thuật nhận một chuỗi chứa các ký tự mở, đóng ngoặc. Kiểm tra chuỗi đã cho có hợp lệ về việc mở và đóng ngoặc không?

Bài 3. Xây dựng chương trình chuyển một chuỗi biểu thức dạng trung tố sang hậu tố. Trong biểu thức gồm:

- Các phép toán: +, -, *, / và ^ (lũy thừa)
- Các giá trị là các số nguyên n từ 1 tới 9
- Dấu (,)

Ví dụ: $1 + (2 * 4 - (4 + 6 / 2))$

IV. Bài tập về nhà

Bài 4.

1. Đảo ngược 1 **chuỗi ký tự** bất kỳ được nhập từ bàn phím bằng cách dùng Stack.
2. Ứng dụng ngăn xếp Stack để chuyển từ biểu thức trung tố (infix) sang hậu tố (postfix) và tính giá trị biểu thức.

Hướng dẫn:

Biểu thức hậu tố (Postfix) là biểu thức được biểu diễn bằng cách đặt các toán tử ra sau các toán hạng.

Một vài ví dụ minh họa:

Infix	Postfix
$A / B - C * D$	$A B / C D * +$
$A / (B - C * D)$	$A B C D * - /$

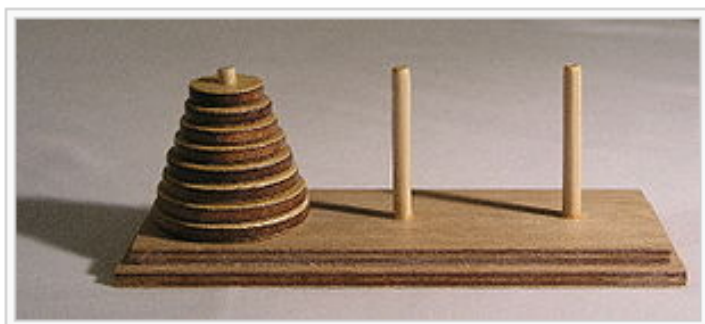
Yêu cầu bài toán được tách thành 2 bước:

- Bước 1: chuyển biểu thức trung tố thành biểu thức hậu tố
 - Bước 2: từ biểu thức hậu tố thu được ở bước 1, thực hiện tính toán và in kết quả cho người dùng.
3. Giải bài toán Tính giai thừa bằng Stack.
 4. Giải bài toán Tháp Hà Nội bằng Stack.

Mô tả bài toán:

Một nhà toán học Pháp sang thăm Đông Dương đến một ngôi chùa cổ ở Hà Nội thấy các vị sư đang chuyển một chồng đĩa quý gồm 64 đĩa với kích thước khác nhau từ cột A sang cột C theo quy tắc:

- Mỗi lần chỉ chuyển 1 đĩa.
- Khi chuyển có thể dùng cột trung gian B.
- Trong suốt quá trình chuyển các chồng đĩa ở các cột luôn được xếp đúng (đĩa có kích thước bé được đặt trên đĩa có kích thước lớn).



--- HẾT ---

Trường: **ĐH CNTP TP.HCM**
Khoa: **Công nghệ thông tin**
Bộ môn: **Công nghệ phần mềm.**
MH: **TH Cấu trúc dữ liệu & giải thuật**

BÀI 7. 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:

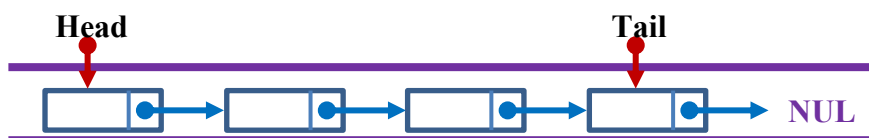
STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

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).



2. Thao tác trên Queue

a. Khai báo kiểu dữ liệu 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;
    };

```

b. 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;
}

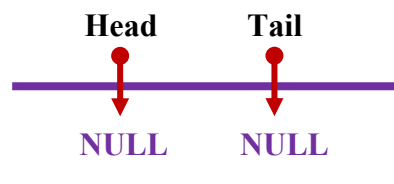
```

c. Khởi tạo Queue rỗng

```

void initQueue(Queue &q) //initialize Queue
{
    q.Head = NULL;
    q.Tail = NULL; }

```



d. Kiểm tra Queue rỗng.

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

e. 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
}
```

- f. Lấy ra giá trị **x** và hủy phần tử ở 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
}
```

- g. Xem giá trị **x** của phần tử ở đầu Queue.

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

- h. Xem giá trị **x** của phần tử ở cuối Queue.

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

i. Duyệt và 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);
}
```

II. Bài tập hướng dẫn mẫu

Bài 1: Hãy viết chương trình mô phỏng một Queue lưu trữ các số nguyên?

Bước 1: Tạo một Project mới.

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

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

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

```
typedef int 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)
{
    ...
}
//=====
void initQueue(Queue &q)
{
    //initialize Queue
    ...
}
//=====
int isEmpty(Queue q)
{
    ...
}
//=====
int insert(Queue &q, QueueNode* p)
{
    ...
}
//=====

```

```

int remove(Queue &q, ItemType &x)
{
    ...
}

//=====
void createQueue_Automatic(Queue &q)
{
    int n;
    printf("\nBan hay cho biet so phan tu cua hang
doi: ");
    scanf("%d", &n);
    initQueue(q);
    srand((unsigned)time(NULL));
    //Thư viện stdlib.h và time.h
    for(int i = 1; i <= n; i++)
    {
        //Tạo 1 số ngẫu nhiên trong đoạn [-99,99]
        ItemType x = (rand()%199) - 99;
        QueueNode* p = createQueueNode(x);
        insert(q, p); //Thêm vào đầu của Queue
    }
}

//=====
void showQueue(Queue q)
{
    if(isEmpty(q) == 1)
    {
        printf("\nHang doi rong.");
    }
}

```

```

        return;
    }
    printf("\nNOI DUNG CUA HANG DOI:\n");
    QueueNode* p = q.Head;
    while(p != NULL)
    {
        ItemType x = p->Info;
        printf("%4d", x);
        p = p->Next;
    }
}

//=====

```

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

III. Bài tập ở lớp

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

Bài 1. Thực hiện lại bài tập mẫu.

Bài 2. Bổ sung thêm vào Bài 1 những chức năng sau:

- Thêm một phần tử vào hàng đợi.
- Xem giá trị phần tử ở đầu hàng đợi.
- Xem giá trị phần tử ở cuối hàng đợi.
- Lấy một phần tử ra khỏi hàng đợi.
- Lấy ra khỏi hàng đợi toàn bộ nội dung và cho phép xem nội dung đó.

Bài 3. Dựa trên bài toán nhập xuất hàng hóa. Sau khi công ty nhập hàng hóa về, chúng sẽ được đưa ngay vào kho để bảo quản. Khi có khách hàng đến mua hàng thì bộ phận bán hàng sẽ lấy hàng hóa từ kho ra để giao cho khách (theo nguyên tắc của queue - cái gì vào trước thì sẽ được lấy ra trước). Một hàng hóa gồm có các thông tin: Mã hàng, Tên hàng,

Đơn vị tính (ĐVT) và Số lượng. Hãy cài đặt chương trình để mô phỏng bài toán với các thao tác sau:

- a. Nhập một danh sách có 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.

IV. Bài tập về nhà


Bài 4. Hãy phân tích và viết chương trình để mô phỏng việc xếp hàng mua vé tàu (xe lửa). Sinh viên tự tìm hiểu và phân tích bài toán (*có thể hỏi thêm giảng viên để được hướng dẫn thêm.*). Biết thông tin người mua vé tàu gồm: Số CMND/Số căn cước, Họ và tên, Năm sinh.

Bài 5. Hãy phân tích và viết chương trình để mô phỏng việc xếp hàng để khám bệnh. Sinh viên tự tìm hiểu và phân tích bài toán (*có thể hỏi thêm giảng viên để được hướng dẫn thêm.*)

Bài 6. Hãy phân tích và viết chương trình để mô phỏng việc xếp hàng để chứng giấy tờ tại một phòng công chứng. Sinh viên tự tìm hiểu và phân tích bài toán (*có thể hỏi thêm giảng viên để được hướng dẫn thêm.*)

Lưu ý: Cả 3 bài tập về nhà đều thực hiện với các chức năng sau:

- Thêm một phần tử vào hàng đợi.
- Xem thông tin của hàng đợi.
- Xem thông tin phần tử ở đầu hàng đợi.
- Xem thông tin phần tử ở cuối hàng đợi.
- Tìm và xem thông tin của một phần tử trong hàng đợi.
- Lấy một phần tử ra khỏi hàng đợi và xem thông tin.
- Lấy ra khỏi hàng đợi toàn bộ nội dung và cho phép xem nội dung.

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 8. CÂY NHỊ PHÂN	
------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------	------------------------------------------------------------------------------------

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	Computer	1	1	

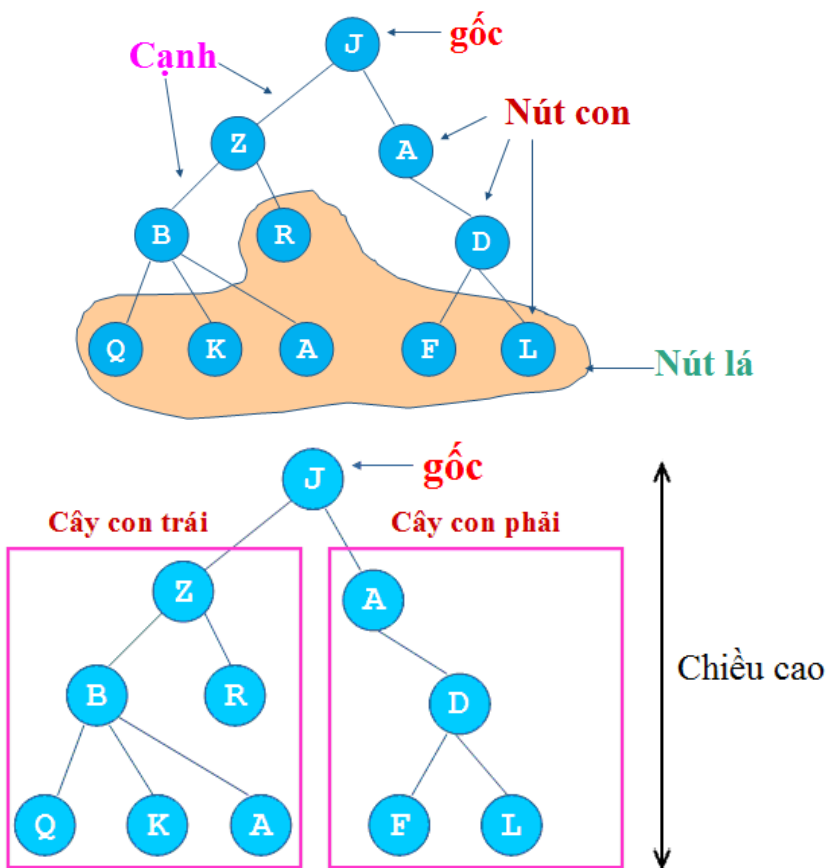
C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

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.

2. Cây nhị phân

a. Khái niệm

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

b. Cấu trúc của một nút



Mỗi nút (phần tử) của cây nhị phân ứng với một biến động gồm ba thành phần:

- Thông tin (dữ liệu) lưu trữ tại nút: **Info**.
- Địa chỉ nút gốc của cây con trái trong bộ nhớ: **Left**.
- Địa chỉ nút gốc của cây con phải trong bộ nhớ: **Right**.

3. Các thao tác trên cây nhị phân

a. 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; };

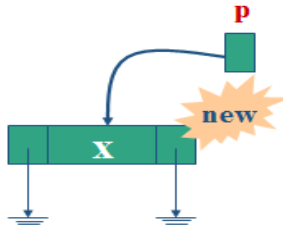
```

```

struct BTree
{
    /* Định nghĩa kiểu dữ liệu cho cây nhị phân
    (Cây NPTK) */
    TNode* Root;
};

```

b. Tạo nút mới chứa giá trị x



```

TNode* createTNode(ItemType x)
{
    TNode* p = new TNode;
    if(p == NULL)
    {
        printf("Khong du bo nho cap phat !");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Left = NULL;
    p->Right = NULL;
    return p;
}

```

c. Xuất nội dung của nút

```

void showTNode(TNode* p)

```

```

{
    printf("%4d", p→Info);
}

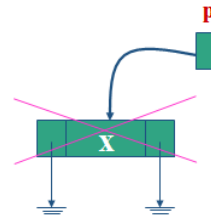
```

d. Hủy nút

```

void deleteTNode(TNode* &p)
{
    if(p == NULL) return;
    p→Left = NULL;
    p→Right = NULL;
    delete p;
    p = NULL;
}

```



e. Khởi tạo cây

```

void initBTree(BTree &bt)
{
    //initialize BTree
    bt.Root = NULL;
}

```



f. Kiểm tra cây rỗng

```

int isEmpty(BTree bt)
{ // BTree is empty or not?
    return (bt.Root == NULL) ? 1 : 0;
}

```

g. 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
}

```

- h. 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
}

```

- i. 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→Left == NULL) //Chưa có con trái
        insertTNode(root→Left, p); //Thêm bên trái
}

```

```

else if(root→Right == NULL) //Chưa có con phải
    insertTNode(root→Right, p);
    //Them ben phải
else //Đã có con trái và con phải
{
    int x = rand()%2; //Tao so chan hoac le
    if(x == 0)
        insertTNode(root→Left, p);
        //Them ben trái
    else //if(x == 1)
        insertTNode(root→Right, p);
        //Them ben phải
    }
    return 1; //Thực hiện thành công
}

```

j. Duyệt cây theo traverseNLR

```

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

```

k. Duyệt cây theo traverseLNR

```

void traverseLNR(TNode* root)
{
    ...
}

```


l. Duyệt cây theo traverseLRN

```
void traverseLRN(TNode* root)
{
    ...
}
```

m. Tìm kiếm nút chứa giá trị x

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

n. 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
    ...
}
```

Gợi ý: Nút lá là nút có cả 2 con trỏ Left và Right đồng thời **NULL**.

- Nếu con trỏ T= **NULL** thì hàm trả về 0.

- Nếu con trỏ T->Left= **NULL** và T->Right= **NULL** thì hàm trả về 1.
Ngược lại hàm trả về 0.

o. Xóa nút con bên trái của nút T

```
int deleteTNodeLeft(TNode* T, ItemType &x)
{ //Nút con trái của T phải là nút Lá. Xóa thành
  công trả về 1, ngược lại trả về 0.
  ...
}
```

Gợi ý:

- Nếu con trỏ T= NULL thì hàm trả về 0.
- Gán p= T->Left.
- Nếu con trỏ p = NULL thì hàm trả về 0.
- Nếu con trỏ p->Left <> NULL và p->Right <> NULL thì hàm trả về 0.
- Gán x= T->Info.
- Xóa p.
- Hàm trả về 1.

p. Xóa nút con bên phải của nút T

```
int deleteTNodeRight(TNode* T, ItemType &x)
{
  ...
}
```

Gợi ý:

- Nếu con trỏ T= **NULL** thì hàm trả về 0.
- Gán p= T->Right.
- Nếu con trỏ p = **NULL** thì hàm trả về 0.
- Nếu con trỏ p->Left <> **NULL** và p->Right <> **NULL** thì hàm trả về 0.
- Gán x= T->Info.
- Xóa p.

- Hàm trả về 1.

q. Đếm số nút của cây

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

r. Đếm số nút lá của cây

```
int countTNodeIsLeaf(TNode* root)
{ //Ham dem so nut la hien cua cay
    ...
}
```

Gợi ý: Tương tự hàm **countTNode**

- Nếu root = **NULL** thì hàm trả về 0.
- Nếu root->Left = **NULL** và root->Right = **NULL** thì hàm trả về 1.
- Tính cnl = Số lượng nút nhánh con trái.
- Tính cnr = Số lượng nút nhánh con phải.
- Hàm trả về cnl + cnr.

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

```
int countTNodeHaveTwoChild(TNode* root)
{
    //Ham dem so nut co du 2 con
    ...
}
```

Gợi ý:

- Nếu root= **NULL** thì hàm trả về 0.
- Nếu root->Left= **NULL** hoặc root->Right= **NULL** thì hàm trả về 0.
- Tính cnl=Số lượng nút nhánh con trái.
- Tính cnr=Số lượng nút nhánh con phải.
- Hàm trả về cnl + cnr + 1.

t. Tính tổng giá trị các nút của cây

```
int sumTNode(TNode* root) //Tính tong gia tri
cac nut hien co trong cay
{
}
```

Gợi ý:

- Nếu root= **NULL** thì hàm trả về 0.
- Tính suml = Tổng giá trị các nút nhánh con trái.
- Tính sumr = Tổng giá trị các nút nhánh con phải.
- Hàm trả về suml + sumr + root->Info.

u. Tính chiều cao của cây

```
int highTree(TNode* root)
{ //Ham tinh chieu cao cua cay
    ...
}
```

Gợi ý:

- Nếu root= **NULL** thì hàm trả về 0.
- Tính hl = Chiều cao nhánh con trái.
- Tính hr = Chiều cao nhánh con phải.
- Nếu hl > hr thì: Hàm trả về hl + 1.
- Ngược lại: Hàm trả về hr + 1.

II. Bài tập hướng dẫn mẫu

Bài 1. Ứng dụng Cây NP để viết chương trình quản lý các số nguyên?

Bước 1: Tạo một Project mới.

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 BTree
{
    //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)
{
    ...
}
//=====
```

```

void initBTree(BSTree &bt)
{ //initialize BSTree
    ...
}
//=====
int insertTNodeLeft(TNode* &T, TNode* p)
{
    ...
}
//=====
int insertTNodeRight(TNode* &T, TNode* p)
{
    ...
}
//=====
int insertTNode(TNode* &T, TNode* p)
{
    ...
}
//=====
void createBSTree_FromArray(BTree &bt, ItemType
a[], int na)
{ //Ham tao cay NPTK tu mang a
    ...
}
//=====

```

```

void showTNode(TNode* p)
{
    ...
}

//=====

void traverseNLR(TNode* root)
{
    //Ham duyet cay theo thu tu traverseNLR
    ...
}

//=====

TNode* findTNode(TNode* root, ItemType x)
{
    ...
}

```

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

```

//=====

int highTree(TNode* root)
{
    //Ham tinh chieu cao cua cay
    ...
}

//=====

```

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

III. Bài tập ở lớp

Bài 1. Cho cây nhị phân chứa các số nguyên (mỗi nút là 1 số nguyên) như Bài tập mẫu 1. Hãy hoàn thiện chương trình với những chức năng sau:

- a. Tạo cây NP bằng 3 cách (**Cách 1:** Cho trước 1 mảng **a** có **n** phần tử, hãy tạo một cây NP 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 NP bằng 6 cách: `traverseNLR`, `traverseLNR`, `traverseLRN`, `traverseNRL`, `traverseRNL`, `traverseRLN`.
- c. Thêm 1 nút có giá trị **x** làm con trái của nút có giá trị **y** của cây.
- d. Thêm 1 nút có giá trị **x** làm con phải của nút có giá trị **y** của cây.
- e. Đếm số nút trên cây.
- f. Kiểm tra (*tìm kiếm*) 1 nút có giá trị **x** có tồn tại trên cây hay không?
- g. Liệt kê các nút có giá trị có lớn hơn **x**.
- h. Thực hiện một số thao tác xử lý tính toán trên cây như: Đếm số nút trên cây/ số nút lá/ số nút có 1 con/ số nút có 2 con/..., Tính tổng các nút trên cây/ tổng nút lá/ tổng nút có 1 con/ tổng nút có 2 con/..., tính chiều cao, ...

Bài 2. Cho cây nhị phân mà mỗi nút là 1 phân số. Hãy viết chương trình để thực hiện những chức năng sau:

- a. Tạo cây NP bằng 2 cách (nhập liệu từ bàn phím, tạo ngẫu nhiên tự động).
- b. Duyệt cây NP bằng 6 cách: `traverseNLR`, `traverseLNR`, `traverseLRN`, `traverseNRL`, `traverseRNL`, `traverseRLN`.
- 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. Đếm số lượng những phân số lớn hơn 1.
- f. Tối giản tất cả các nút (*phân số*) của cây.
- g. Tìm kiếm trên cây có nút nào có giá trị bằng với phân số **x** hay không?

B. Bài tập về nhà


Bài 3. Tiếp theo bài tập 2. Hãy bổ sung thêm những chức năng sau:

- a. Tính tổng các phân số.
- b. Tìm phân số nhỏ nhất.

- c. Tìm phân số lớn nhất.
- d. Liệt kê các phân số có tử số lớn hơn mẫu số.
- e. Liệt kê các phân số có tử số nhỏ hơn mẫu số.
- f. Liệt kê các phân số có tử số và mẫu số đồng thời là các số nguyên tố.
- g. Liệt kê các phân số ở mức k (k được nhập từ bàn phím).
- h. Đếm số lượng phân số ở mức k (k được nhập từ bàn phím).
- i. Tính tổng các phân số ở mức k (k được nhập từ bàn phím).

Bài 4. Hãy viết lại các hàm duyệt cây với việc khử đệ quy bằng cách ứng dụng **Stack**.

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 9. 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	Computer	1	1	

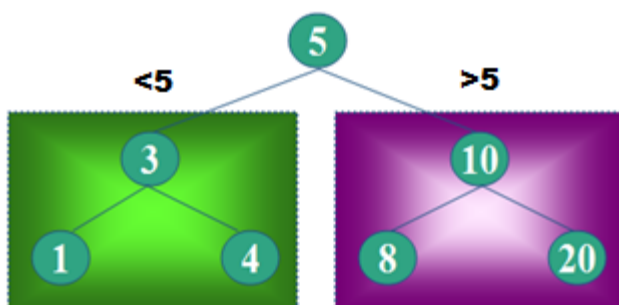
C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. 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.



2. Cấu trúc của một nút



Mỗi nút (phần tử) của cây nhị phân ứng với một biến động gồm ba thành phần:

- Thông tin (dữ liệu) lưu trữ tại nút: **Info**.
- Địa chỉ nút gốc của cây con trái trong bộ nhớ: **Left**.
- Địa chỉ nút gốc của cây con phải trong bộ nhớ: **Right**.

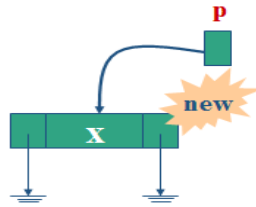
3. Các thao tác trên cây nhị phân tìm kiếm

a. 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 BSTree
{
    /* Định nghĩa kiểu dữ liệu cho cây nhị phân
    (Cây NPTK) */
    TNode* Root;
};
```

b. Tạo nút mới chứa giá trị x



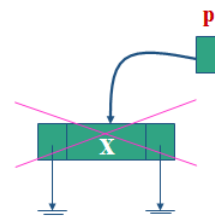
```
TNode* createTNode(ItemType x)
{
    TNode* p = new TNode;
    if(p == NULL)
    {
        printf("Khong du bo nho cap phat!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Left = NULL;
    p->Right = NULL;
    return p;
}
```

c. Xuất nội dung của nút

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

d. Hủy nút

```
void deleteTNode(TNode* &p)
{
    if(p == NULL) return;
```



```

    p→Left = NULL;
    p→Right = NULL;
    delete p;
    p = NULL;
}

```

e. Khởi tạo cây

```

void initBSTree(BSTree &bst)
{
    //initialize BTree
    bst.Root = NULL;
}

```

f. Kiểm tra cây rỗng

```

void isEmpty(BSTree bst)
{
    //initialize BTree
    return (bst.Root == NULL)? 1 : 0;
}

```



g. 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 phải
return 1; //Thực hiện thành công
}

```

- h. Tạo cây từ một file chứa **n** số nguyên (**n** > 0)

```

void createBSTree_FromFile(BSTree &bst, char
fileName[])
{
    //Ham tao cay NPTK tu file
    FILE *f;
    f = fopen(fileName, "rt");
    if(!f) return;
    int n;
    fscanf(f, "%d", &n);
    ItemType x;
    initBSTree(bst);
    for(int i = 0; i < n; i++)
    {
        fscanf(f, "%d", &x);
        TNode* p = createTNode(x);
        insertTNode(bst.Root, p);
    }
}

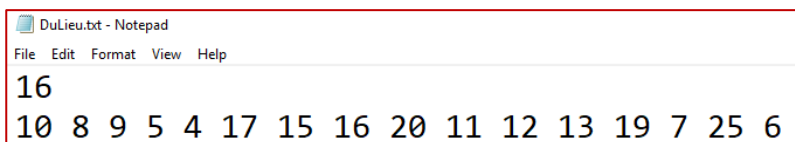
```

```

        fclose(f);
    }

```

Lưu ý: Cấu trúc file text như sau: dòng đầu lưu một số nguyên n là số lượng phần tử, dòng thứ hai lưu n số nguyên cách nhau bằng khoảng trắng, ví dụ như sau:



```

DuLieu.txt - Notepad
File Edit Format View Help
16
10 8 9 5 4 17 15 16 20 11 12 13 19 7 25 6

```

i. Duyệt cây theo traverseNLR

```

void traverseNLR(TNode* root)
{
    ...
}

```

j. Duyệt cây theo traverseLNR

```

void traverseLNR(TNode* root)
{
    ...
}

```

k. Duyệt cây theo traverseLRN

```

void traverseLRN(TNode* root)
{
    ...
}

```

l. Tìm kiếm nút chứa giá trị x

```

TNode* findTNode(TNode* root, ItemType x)
{
    ...
}

```

m. 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
    }

```

```

    }
}

```

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

```

int maxTNode(TNode* root)
{
    //Ham tìm nút có giá trị lớn nhất của cây
    ...
}

```

Gợi ý: Nút lớn nhất là nút bên phải cùng của cây.

- Cho p=root.
- Trong khi p->Right <> NULL thì p=p->Right.
- Hàm trả về p->Info.

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

```

void showTNodeOfLevelK(TNode *root, int k)
{
    ...
}

```

Gợi ý: Theo nguyên tắc của phép duyệt cây NLR

- Nếu root=NULL thì dừng (bằng lệnh *return*).
- Nếu k=0 thì xuất thông tin của root->Info.
- Giảm k xuống 1 đơn vị.
- Gọi đệ quy cho nhánh con trái với root->Left ở mức k.
- Gọi đệ quy cho nhánh con phải với root->Right ở mức k.

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

```

void showTNodeIsLeafOfLevelK(TNode *root, int k)
{
    ...
}

```

Gợi ý: Tương tự như hàm **showTNodeOfLevelK**

- Nếu root=NULL thì dừng (bằng lệnh *return*).
- Nếu k=0 và root→Left=NULL và root→Right=NULL thì xuất thông tin của root→Info.
- Giảm k xuống 1 đơn vị.
- Gọi đệ quy cho nhánh con trái với root→Left ở mức k.
- Gọi đệ quy cho nhánh con phải với root→Right ở mức k.

v. **Đếm số lượng nút ở mức thứ k của cây**

```
int countTNodeOfLevelK(TNode* root, int k)
{
    ...
}
```

Gợi ý:

- Nếu root=NULL thì hàm trả về 0.
- Nếu k=0 thì hàm trả về 1.
- Giảm k xuống 1 đơn vị.
- Tính cnl=Số lượng nút của nhánh con trái với root→Left ở mức k.
- Tính cnr=Số lượng nút của nhánh con phải với root→Right ở mức k.
- Hàm trả về cnl + cnr.

II. Bài tập hướng dẫn mẫu

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

Bước 1: Tạo một Project mới

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)
{
    ...
}
//=====
void initBSTree(BSTree &bst)
{ //initialize BSTree
    ...
}

```

```

//=====
int insertTNode(TNode* &T, TNode* p)
{
    ...
}

//=====
void createBSTree_FromArray(BTree &bt, ItemType
a[], int na)
{//Ham tao cay NPTK tu mang a
    ...
}

//=====
void showTNode(TNode* p)
{
    ...
}

//=====
void traverseNLR(TNode* root)
{//Ham duyet cay theo thu tu traverseNLR
    ...
}

//=====
int countTNodeIsLeafOfLevelK(TNode *root, int k)
{
    ...
}

//=====

```

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

III. Bài tập ở lớp

Bài 1. Cho cây nhị phân tìm kiếm chứa các số nguyên (*mỗi nút là 1 số nguyên*) như Bài tập mẫu 1. Hãy hoàn thiện chương trình với những chức năng sau:

- a. Tạo cây NP bằng 3 cách (**Cách 1:** Cho trước 1 mảng **a** có **n** phần tử, hãy tạo một cây NP 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 NP bằng 6 cách: `traverseNLR`, `traverseLNR`, `traverseLRN`, `traverseNRL`, `traverseRNL`, `traverseRLN`.
- c. Thêm 1 nút có giá trị **x** vào cây.
- d. Tìm kiếm 1 nút có giá trị **x** trên cây hay không.
- e. Xóa nút có giá trị **x** trên cây.
- f. Xuất các phần tử theo chiều giảm dần.
- g. Đếm số giá trị lớn hơn **x**, nhỏ hơn **x**, có giá trị trong đoạn **[x, y]**.
- h. Tìm nút có giá trị lớn nhất, nhỏ nhất của cây.
- i. Xuất ra nội dung các nút ở mức **k**/ nội dung các nút lá ở mức **k**/ nội dung các nút chỉ có 1 con ở mức **k**/
- j. Đếm số nút ở mức **k**/ số nút lá ở mức **k**/ số nút chỉ có 1 con ở mức **k**/
- k. Tính tổng giá trị các nút dương/ giá trị các nút âm trên cây.

Bài 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: `traverseNLR`, `traverseLNR`, `traverseLRN`, `traverseNRL`, `traverseRNL`, `traverseRLN`.
- c. Thêm 1 nút là phân số **p** vào cây.
- d. Tìm kiếm 1 phân số **x** có trên cây hay không?
- e. Xóa một phân số **x** trên cây.
- f. Xóa những phân số >2 (*xét theo giá trị*).

- g. Xóa những phân số có mẫu số là số nguyên tố.
- h. Tính tổng các phân số.
- i. Tìm phân số nhỏ nhất.
- j. Tìm phân số lớn nhất.
- k. Liệt kê các phân số có tử số lớn hơn mẫu số.
- l. Liệt kê các phân số có tử số nhỏ hơn mẫu số.
- m. Liệt kê các phân số có tử số và mẫu số đồng thời là các số nguyên tố.
- n. Liệt kê các phân số ở mức k (*k được nhập từ bàn phím*).
- o. Đếm số lượng phân số ở mức k (*k được nhập từ bàn phím*).
- p. Tính tổng các phân số ở mức k (*k được nhập từ bàn phím*).

IV. Bài tập về nhà


Bài 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 nằm ở tầng thấp hơn tầng thứ k của cây.
- h. Đếm số lượng nút nằm ở tầng cao hơn tầng thứ k của cây.
- i. Tính tổng các nút lẻ.
- j. Tính tổng các nút lá mà thông tin tại nút đó là giá trị chẵn.
- k. Tính tổng các nút có đúng 1 con mà thông tin tại nút đó là số nguyên tố.
- l. Tính tổng các nút có đúng 2 con mà thông tin tại nút đó là số chính

phương.

- m. Kiểm tra cây nhị phân T có phải là "cây nhị phân tìm kiếm" hay không?
- n. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng" hay không?
- o. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng hoàn toàn" hay không?

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 10. CÂY NHỊ PHÂN TÌM KIẾM (tiếp theo)	
------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------	------------------------------------------------------------------------------------

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	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Khái niệm cây nhị phân tìm kiếm
2. Các thao tác trên cây nhị phân tìm kiếm

a. 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);
}

```

- b. 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;
}
```

- c. 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;
}

```

II. Bài tập ở lớp

Bài 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:

- Tìm phần tử có khoảng cách về giá trị gần nhất với phần tử x trong cây (nếu cây rỗng trả về -1).
- Tìm phần tử có khoảng cách về giá trị xa nhất với phần tử x trong cây (nếu cây rỗng trả về -1).
- Đếm số nút của cây (dùng đệ quy / không dùng đệ quy).
- Đếm số nút là số nguyên tố, là số chính phương, là số hoàn thiện, là số thịnh vượng, là số yếu của cây.
- Tính tổng giá trị các nút của cây (dùng đệ quy / không dùng đệ quy).
- 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.

g. Xóa toàn bộ cây.

Bài 2. Cho cây nhị phân tìm kiếm như ở *Bài tập 2 (BÀI THỰC HÀNH 9)*. Hãy bổ sung những chức năng sau:


- a. Đếm có bao nhiêu phân số có cả tử số và mẫu số đều là các số nguyên tố.
- b. Xóa toàn bộ danh sách.

IV. Bài tập về nhà

Bài 3. Cho cây nhị phân tìm kiếm 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 NPTK với những chức năng sau:

- a. Tạo cây NPTK từ 1 file từ điển cho trước.
- b. Duyệt cây NPTK để xem nội dung theo phép duyệt cây traverseLNR.
- 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. Bổ sung hay chỉnh sửa nghĩa của 1 từ bất kỳ.
- g. Xóa toàn bộ cây.

--- HẾT ---

Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. MH: TH Cấu trúc dữ liệu & giải thuật	BÀI 11. BẢNG BẤM	
------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------	------------------------------------------------------------------------------------

A. MỤC TIÊU:

- Vận dụng các cấu trúc bảng bấm vào từng bài toán cụ thể.
- Lập trình được các cấu trúc bảng bấm 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	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Mô tả bảng bấm

a. Mô tả dữ liệu

Bảng bấm được mô tả bằng các thành phần sau:

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

Bảng bấm được mô tả bằng hình vẽ như sau:



b. Các tác vụ trên bảng bấm

Bảng bấm có thể có các tác vụ sau:

- **Tác vụ khởi tạo:** Cấp phát bộ nhớ và khởi tạo 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.

c. 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 kết nối trực tiếp:*** Mỗi địa chỉ của bảng băm tương ứng với một danh sách liên kết. Các nút bị xung đột được kết nối với nhau trên một danh sách liên kết.
- ***Bảng băm với giải thuật kết nối hợp nhất:*** Bảng băm loại này được cài đặt bằng danh sách kề, 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 kết nối 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.

d. 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.

e. 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.

2. Bảng băm với giải thuật kết nối trực tiếp

a. Mô tả

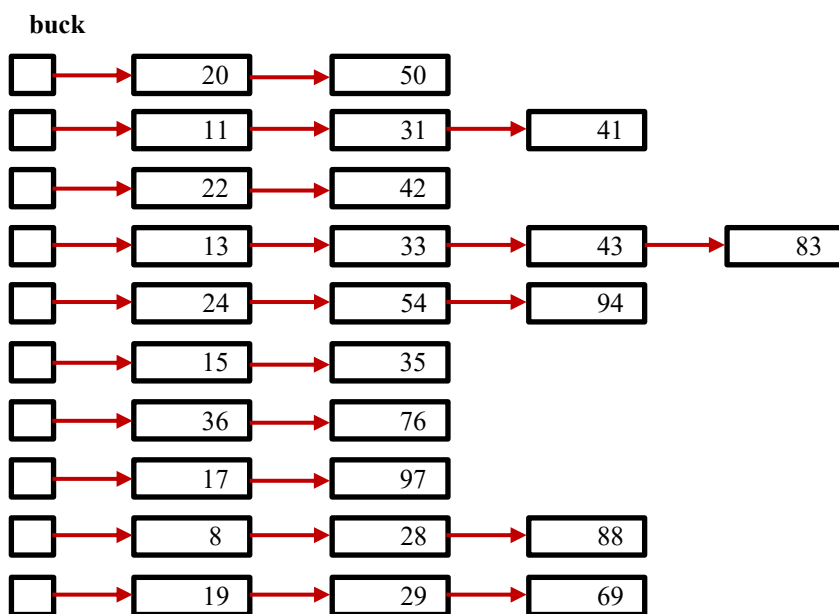
Bảng băm được cài đặt bằng danh sách liên kết, các nút trên bảng băm được băm thành MAXSIZE danh sách liên kết (từ danh sách 0 đến

danh sách MAXSIZE - 1). Các nút bị xung đột tại địa chỉ i được kết nối 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

b. Cài đặt

- *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];

```

- *Hàm băm*

```

int hashFunction(int Key)
{
    return (Key % MAXSIZE);
}

```
- *Tìm kiếm một phần tử trên bảng băm*

```

// Hàm tìm kiếm một khóa k trên bảng băm
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;
}

```
- *Thêm vào một phần tử*

```

// Thêm nút q chứa khóa k sau nút p
int insertAfter(HashNode* p, int k)

```

```

{
    if(p == NULL)
    {
        printf("Khong them duoc vao HashNode");
        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);
```

```
}
```

- *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;
```

```
}
```

- *Xoá 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 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 = 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 Khong co HashNode co khoa
la: %d", b);
    else if(p == bucket[b])
        pop(b);
    else
        deleteAfter(q);
}

```

c. 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;
}
//=====

```

```

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

//=====
// Kiem tra mot bucket co phai Empty?
int isEmptyBucket(int b)
{
    if(bucket[b] == NULL)
        return TRUE;
    else
        return FALSE;
}

//=====
// Kiem tra toan bo bang bam co NULL hay khong
int isEmpty()
{
    for(int b = 0; b < MAXSIZE; b++)
        if(bucket[b] != NULL)
            return FALSE;
    return TRUE;
}

//=====
// Traverse tren tung bucket
void traverseBucket(int b)
{
    HashNode* p = bucket[b];

```

```

    while(p != NULL)
    {
        printf("%4d", p→Key);
        p = p→Next;
    }
}

//=====
// Duyệt qua bang bam
void traverseHashTable()
{
    for(int b = 0; b < MAXSIZE; b++)
    {
        printf("\n Bucket[%d]=", b);
        traverseBucket(b);
    }
}

//=====
// Xoa tren mot 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;
    }
    //=====
    // Xoa toan bo bang bam
    void Clear()
    {
        for(int b = 0; b < MAXSIZE; b++)
        {
            ClearBucket(b);
        }
    }
    //=====
    // Them mot nut vao dau 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)
{
    ...
}

//=====
// Tac vu nay chi su dung khi them vao mot
// bucket co thu tu
void place(int b, int k)
{
    ...
}

//=====
// Them mot nut co khoa k vao trong bang bam
void insert(int k)
{
    ...
}

//=====

```

```

// Tac vu tim kiem mot khoa trong bang bam
int search(int k)
{
    ...
}

//=====
// Xoa mot nut ngay sau nút p
int deleteAfter(HashNode* p)
{
    ...
}

//=====
// Xoa mot phan tu co khoa k ra khoi bang bam
void remove(int k)
{
    ...
}

//=====
// Hàm xử lý chính của chương trình
//=====
// Chương trình chính
void main()
{
    ...
}

```

3. Bảng băm với giải thuật kết nối hợp nhất
 - a. 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 kết nối 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 tạo 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 tạo:

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 $\text{MAXSIZE} - 1$.

- 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

b. Cài đặt

- 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;
```

- **Hàm băm**

```
int hashFunction(int Key)
```

```
{
```

```
    return (Key % MAXSIZE);
```

```
}
```

- **Tác vụ khởi tạo 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;
```

```
}
```

- **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;
}

```

- **Tác vụ tìm kiếm**

/ Tìm một khóa có trong bảng băm hay không, tìm
thay tra về địa chỉ không thay tra 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;
}

```

- **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;
    }

```

- *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
}

```

▪ **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);
    }
}

```

c. Chương trình minh họa

Sinh viên tự làm xem như một bài tập.

4. Bảng băm với giải thuật dò tuyến tính

a. 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 $\text{MAXSIZE} - 1$:

- 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 $\text{MAXSIZE} - 1$, tìm nút khóa Key trong khối đặc chứa các nút xuất phát từ địa chỉ i .

Hàm băm lại của giải thuật 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 phát 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

b. Cài đặt

- *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;
```

- *Hàm băm*

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

- *Tác vụ khởi tạo*

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

- *Kiểm tra bảng băm có rỗng hay không*

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

- *Kiểm tra bảng băm có đầy hay không*

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

- *Tác vụ tìm kiếm*

//Tác vụ tìm 1 Key, tìm thay tra về index, không thay tra về 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;
}

```

- *Tác vụ thêm một phần tử vào bảng băm*

// Thêm khóa k vào bảng băm

```

int insert(int k)
{
    int i, j;
    if(isFull() == TRUE)
    {
        printf("\n Bảng băm bị đầy, 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;
}

```

▪ *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", i,
        HashTable[i].Key);
    }
}
```

c. **Chương trình minh họa**

Sinh viên tự làm xem như một bài tập.

II. Bài tập ở lớp

Bài 1. Cài đặt lại ba loại bảng băm như mô tả ở phần tóm tắt lý thuyết ở trên.

Bài 2. Viết một chương trình hiện thực từ điển Anh - Việt, chương trình có cài đặt bảng băm với giải thuật kết nối trực tiếp. Mỗi nút của bảng băm 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 băm sau cho khóa bắt đầu bằng ký tự **a** được băm vào địa chỉ 0, b băm vào địa chỉ 1, ..., z băm 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.
- Xoá toàn bộ từ điển.
- Thoát khỏi chương trình.

III. Bài tập về nhà

Bài 3. Viết một chương trình hiện thực việc tra cứu thông tin sinh viên bằng cách ứng dụng bảng băm với 3 giải thuật: Kết nối trực tiếp, Kết nối hợp nhất, Dò tuyến tính.

Biết rằng thông tin của tất cả sinh viên chứa trong tập tin văn bản, mỗi sinh viên chiếm một dòng gồm các trường: MSSV, họ, tên và điểm các môn học. Chương trình sẽ đọc tập tin văn bản này để tạo ra bảng băm. Mỗi nút của bảng băm 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 băm 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.

--- HẾT ---