BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC CÔNG NGHỆ TP.HCM

THỰC HÀNH CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Biên Soạn:

TS. Nguyễn Thị Hải Bình

www.hutech.edu.vn

THỰC HÀNH CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập:

tailieuhoctap@hutech.edu.vn

MỤC LỤC

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Môn Thực hành cấu trúc dữ liệu cung cấp cho sinh viên những kỹ năng thực hành nâng cao về ngôn ngữ lập trình C. Cung cấp các phương pháp tổ chức và những thao tác cơ sở trên từng cấu trúc dữ liệu, kết hợp với việc phát triển tư duy giải thuật để hình thành nên các ứng dụng thực tế.

Học xong môn này, sinh viên phải nắm được các vấn đề sau:

Mảng một và hai chiều, cài đặt các giải thuật đệ quy.

- Cấu trúc dữ liệu danh sách, các phương pháp cài đặt danh sách. Trong đó, chú trọng vào kiểu dữ liệu danh sách liên kết.
- Cấu trúc dữ liệu Stack và một số bài toán ứng dụng thực tế của Stack.
- Cấu trúc dữ liệu Queue và một số bài toán ứng dụng thực tế của Queue.
- Cấu trúc dữ liệu cây: cây nhị phân, cây nhị phân tìm kiếm, cây nhị phân tìm kiếm cân bằng.

NỘI DUNG MÔN HỌC

- Bài 1: MẢNG – ĐỆ QUY.
- Bài 2: DANH SÁCH LIÊN KẾT.
- Bài 3: CẤU TRÚC QUEUE VÀ STACK.
- Bài 4: CẤU TRÚC CÂY – CÂY NHỊ PHÂN TÌM KIẾM
- Bài 5: CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG

YÊU CẦU MÔN HỌC

Có tư duy tốt về logic và kiến thức toán học cơ bản.

Có kiến thức cơ bản về nhập môn lập trình và kĩ thuật lập trình.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần tập trung tiếp thu bài giảng của môn Cấu trúc dữ liệu và giải thuật, làm tất cả các ví dụ và bài tập.

Người học phải dự học đầy đủ các buổi thực hành tại lớp, nắm và cài đặt được các cấu trúc dữ liệu và thuật toán đã thực hành. Thực hành lại các bài tập cơ bản và nâng cao ở nhà, nghiên cứu tài liệu trước khi đến lớp và gạch chân những vấn đề không hiểu khi đọc tài liệu để đến lớp trao đổi.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

- Điểm quá trình: 30%. Hình thức và nội dung do GV quyết định, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.
- Điểm thi: 70%. Hình thức bài thi thực hành. Nội dung gồm các bài tập liên quan đến kiến thức thuộc bài thứ 1 đến bài thứ 5.

BÀI 1: MẢNG – ĐỆ QUY

Sau khi thực hành xong bài này, sinh viên có thể nắm được:

- *Ôn lại cách viết một chương trình C dạng hàm*
- *Nắm vững cấu trúc dữ liệu mảng 1 chiều*
- *Nắm vững cách thức lập trình đệ quy*

1.1 TÓM TẮT LÝ THUYẾT

1.1.1 Hàm

Cấu trúc chương trình C theo hàm:

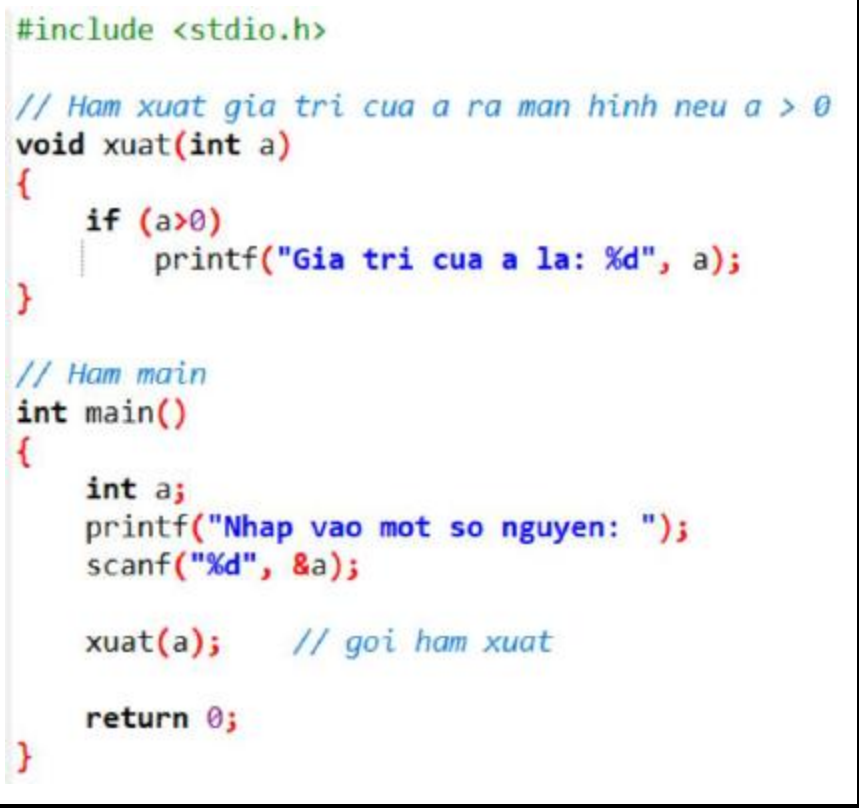
Cấu trúc chương trình C đơn giản (không có hàm con)	**Cấu trúc chương trình C theo hàm**	
	Cách 1	**Cách 2**

Khai báo thư viện
Định nghĩa hằng số
 	
Hàm main()
{
Các câu lệnh;
}
	Khai báo thư viện
	Định nghĩa hằng số
	
	Khai báo các hàm con (nguyên mẫu hàm);
	
	Hàm main()
	{
	Các câu lệnh (bao gồm câu lệnh gọi hàm con ra thực hiện);
	}
	
	Các hàm con
		Khai báo thư viện
		Định nghĩa hằng số
		
		Các hàm con
		
		Hàm main()
		{
		Các câu lệnh (bao gồm câu lệnh gọi hàm con ra thực hiện);
		}

Cách định nghĩa hàm:

- Định nghĩa một hàm bao gồm:
  - Khai báo kiểu hàm,
  - Tên hàm,
  - Các tham số,
  - Các câu lệnh để thực hiện chức năng của hàm.
- 2 loại hàm:
  - Hàm không trả về giá trị

Mẫu hàm	Ví dụ

void Ten_Ham(danh sach tham so neu co)
{
Các câu lệnh;
}
	

```c
#include <stdio.h>

// Ham xuat gia tri cua a ra man hinh neu a > 0
void xuat(int a)
{
    if (a>0)
        printf("Gia tri cua a la: %d", a);
}

// Ham main
int main()
{
    int a;
    printf("Nhap vao mot so nguyen: ");
    scanf("%d", &a);

    xuat(a);    // goi ham xuat

    return 0;
}
```

- Hàm trả về giá trị

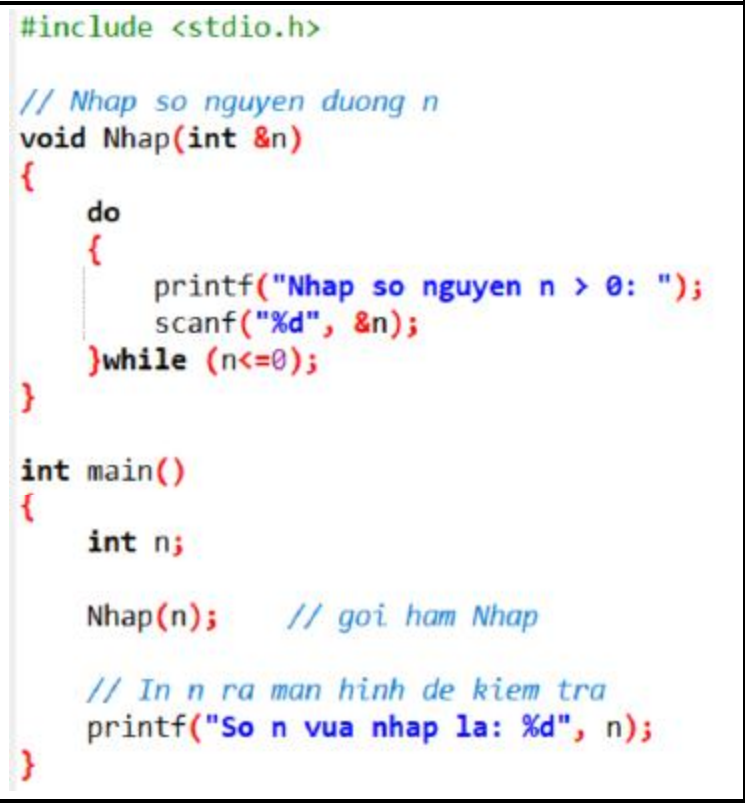
Mẫu hàm	Ví dụ

KieuDL Ten_Ham (danh sach tham so neu co)
{
KieuDL kq;
Các câu lệnh;
return kq;
}
	`#include <stdio.h>`
	
	`// Ham tinh tong hai so nguyen`
	`int TinhTong(int a, int b)`
	`{`
	`int kq;`
	`kq = a + b;`
	`return kq;`
	`}`
	
	`// Ham main`
	`int main()`
	`{`
	`int a, b, kq;`
	
	`printf("Nhap vao hai so nguyen: ");`
	`scanf("%d%d", &a, &b);`
	
	`kq = TinhTong(a, b);  // goi ham TinhTong`
	
	`printf("Tong cua %d va %d la: %d", a, b, kq);`
	
	`return 0;`
	`}`

Tham số của hàm:

Loại tham số	Ý nghĩa	Ví dụ

Tham trị	Giá trị của tham số đầu vào không thay đổi sau khi hàm thực hiện xong.	`// Ham tinh tong hai so nguyen`
		`int TinhTong(int a, int b)`
		`{`
		`int kq;`
		`kq = a + b;`
		`return kq;`
		`}`

Loại tham số	Ý nghĩa	Ví dụ
Tham biến	Giá trị của tham số đầu vào thay đổi sau khi hàm thực hiện xong.	

```c
#include <stdio.h>

// Nhap so nguyen duong n
void Nhap(int &n)
{
    do
    {
        printf("Nhap so nguyen n > 0: ");
        scanf("%d", &n);
    }while (n<=0);
}

int main()
{
    int n;

    Nhap(n);    // goi ham Nhap

    // In n ra man hinh de kiem tra
    printf("So n vua nhap la: %d", n);
}
```

1.1.2 Mảng

Mảng là một tập hợp các phần tử cố định có cùng một kiểu dữ liệu, gọi là kiểu phần tử. Kiểu phần tử có thể là có các kiểu bất kỳ: ký tự, số, chuỗi ký tự…; cũng có khi ta sử dụng kiểu mảng để làm kiểu phần tử cho một mảng (trong trường hợp này ta gọi là mảng của mảng hay mảng nhiều chiều). Ta có thể chia mảng làm 2 loại: mảng một chiều và mảng nhiều chiều.

Cách khai báo mảng:

❖ Khai báo tường minh (số phần tử xác định):

Cú pháp: <kiểu dữ liệu> <tên mảng> [<số phần tử >];

Ý nghĩa:

- **<Tên mảng>**: được đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên của biến mảng.
- **[<Số phần tử>]:** là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi nó là kích thước của mảng).

- **<Kiểu dữ liệu >**: là kiểu dữ liệu của mỗi phần tử của mảng

❖ **Khai báo không tường minh (số phần tử không xác định)**

Cú pháp: **<kiểu dữ liệu> <tên mảng> [ ];**

Kiểu khai báo này thường được áp dụng trong các trường hợp: vừa khai báo vừa gán giá trị, hoặc khai báo mảng là tham số hình thức của hàm.

1.1.3 Đệ quy

Khái niệm: Vấn đề đệ quy là vấn đề được định nghĩa bằng chính nó. Điều kiện để bài toán có thể giải được bằng đệ quy là bài toán tồn tại bước đệ quy và phải có điều kiện dừng.

Ví dụ: Bài toán tính tổng S(n) = 1 + 2 + ... + n có thể giải bằng đệ quy. Vì:

- Ta có: S(n – 1) = 1 + 2 + ... + (n – 1) → S(n) = S(n – 1) + n → tồn tại bước đệ quy.
- Khi n = 0: S(0) = 0 → Có điều kiện dừng.

Hàm đệ quy trong ngôn ngữ lập trình C: Một hàm được gọi là đệ quy nếu bên trong thân của hàm đó có lời gọi lại chính nó.

Cấu trúc hàm đệ quy: Một hàm đệ quy thông thường gồm 2 phần như sau:

```
<Kiểu dữ liệu> <Tên hàm>(<Danh sách tham số>
{
        // Phần dừng: xác định điểm kết thúc của thuật toán
        if (<Điều kiện dừng>)
        {
            ...
            return <giá trị>;
        }
        // Phần đệ quy
        Lời gọi tới chính hàm đang định nghĩa.
}
```

1.2 THỰC HÀNH CƠ BẢN

1.2.1 Mảng dãy số nguyên

Bài 1: Viết chương trình thực hiện:

a. Nhập mảng gồm n số nguyên (n > 0).

b. Xuất mảng ra màn hình.

c. Tính tổng và trung bình cộng các phần tử của mảng.

d. Sắp xếp mảng theo chiều tăng dần.

e. Tìm phần tử có giá trị X trong mảng, nếu có cho biết vị trí xuất hiện của X trong mảng.

Hướng dẫn:

- Chương trình minh hoạ nhập xuất mảng số nguyên

```
#include <stdio.h>
#define MAX 100
/*---------------------------------
*Hàm nhập số lượng phần tử cho mảng
*/
void NhapSL(int &n)
{
	do{
		printf ("Nhap so phan tu 0<sl<%d: ", MAX);
		scanf (" %d ", &n);
	}while (n<=0 || n>MAX);
}
/*---------------------------------
*Hàm nhập giá trị cho từng phần tử trong mảng
*/
void NhapMang (int a[], int n)
{
	for (int i = 0; i < n; i ++)
	{
		printf (" a[%d] = ", i);
		scanf (" %d ", &a[i]);
	}
```

```
}
/*---------------------------------
*Hàm xuất các phần tử của mảng ra màn hình
*/
void XuatMang (int a[], int n)
{
	printf ("\nMang gom cac phan tu:\n ");
	for (int i = 0; i < n; i ++)
		printf (" %5d", a[i]);
}
// ---------------------------------
int TinhTong(int a[], int n){...}
float TinhTrungBinhCong(int a[], int n){...}
int TimTuyenTinh(int a[], int n, int X){...}
void SapXep(int a[], int n){...}
int TimNhiPhan(int a[], int n, int X){..}
//================================
int main()
{
    int a[MAX], n;
    NhapMang (a,n);
    XuatMang (a,n);
    //gọi thực hiện tính tổng và trung bình cộng của các phần tử
    int tong = TinhTong(a, n);
    float tbc = TinhTrungBinhCong(a, n);
    printf("Tong cac phan tu = %d\n", tong);
    printf("Trung binh cong cac phan tu = %0.3f\n", tbc);

    int X;
    printf("Nhap gia tri can tim: "); scanf("%d", &X);
    //gọi thực hiện tìm kiếm X bằng phương pháp tìm tuyến tính
    int vt=TimTuyenTinh(a,n,X);
    if (vt==-1)
        printf("Khong tim thay");
```

```
    else
        printf("Tim thay %d tai vi tri %d",X, vt);
    //gọi thực hiện sắp xếp mảng
    sapxep(a,n);
    printf("Mang sau khi sap tang dan:"); XuatMang(a,n);
    printf("Nhap gia tri can tim: "); scanf("%d", &X);
    vt=TimNhiPhan(a,n,X);
    //tương tự...
    return 0;
}
```

Mở rộng: Sau khi chạy thử hết các chức năng, chương trình chạy ra kết quả đúng, bạn hãy viết lại hàm main() theo menu chức năng.

1.2.2 Mảng phần tử kiểu struct

Bài 1: Viết chương trình quản lý thư viện, thông tin mỗi cuốn sách gồm: mã sách (int), tên sách (char[40]), giá (float).

a. Nhập danh sách gồm N cuốn sách.

b. Xuất danh sách các cuốn sách ra màn hình.

c. Tìm cuốn sách có mã là X (Làm theo 2 cách: tìm tuyến tính và tìm nhị phân).

d. Xuất ra các cuốn sách có tên là Y.

e. Xuất các cuốn sách có giá cao nhất (nếu có nhiều sách có giá cao nhất trùng nhau thì xuất hết ra màn hình).

Hướng dẫn:

- Khai báo cấu trúc sách

```
typedef struct Tên_cấu_trúc
{
  //khai báo các biến thành phần của cấu trúc;
  ...
}Tên_cấu_trúc_viết_gọn;
```

```
VD:
typedef struct CuonSach
   {
     int masach;
     char tensach[40];
     float gia;
}Sach;
```

- Chương trình có thể tổ chức gồm các hàm sau:
  - void nhap1Sach(Sach &x)
  - void xuat1Sach(Sach x)
  - void nhapn(int &n)
  - void nhapDS(Sach a[], int n)
  - void xuatDS(Sach a[], int n)
  - int timTuanTu(Sach a[], int n, int X)
  - int timNhiPhan(Sach a[], int n, int X)
- **Hàm nhập 1 cuốn sách:** nhập thông tin cho 1 cuốn sách

  void nhap1Sach(Sach &x)
- **Hàm xuất 1 cuốn sách:** xuất thông tin của 1 cuốn sách

  void xuat1Sach(Sach x)
- Hàm nhập số lượng phần tử của danh sách: nhập số lượng cuốn sách

  void nhapn(int &n)
- **Hàm nhập danh sách các cuốn sách:** dùng 1 mảng một chiều để lưu danh các cuốn sách, mỗi phần tử trong mảng là 1 cuốn sách.

```
   void nhapDS(Sach a[], int n)
   {
     //Nhập thông tin cho từng cuốn sách (Nhập a[i], i=0, 1, ..., n-1) bằng cách  gọi hàm nhập 1 cuốn sách cho phần tử a[i]
     for(int i=0; i<n; i++)
     {
```

```
            printf("Nhap cuon sach thu %d: ", i);
            nhap1Sach(a[i]);
        }
    }
```

- **Hàm xuất danh sách các cuốn sách:** Xuất thông tin từng cuốn sách a[i], i = 0, ..., n-1 bằng cách gọi hàm xuất 1 cuốn sách.
- **Hàm tìm cuốn sách mã là X:** Làm theo hai cách tìm tuyến tính và nhị phân.
  - int timTuanTu(Sach a[], int n, int X)
  - int timNhiPhan(Sach a[], int n, int X)

Lưu ý: X là mã cuốn sách cần tìm do người dùng nhập vào (nhập X trước khi gọi hàm tìm).

- **Hàm xuất ra các cuốn sách có tên là Y:** Duyệt danh sách, nếu gặp cuốn nào có tên là Y thì xuất thông tin cuốn sách đó ra màn hình. So sánh chuỗi dùng hàm strcmp() hoặc stricmp() trong thư viện <string.h>.
- **Hàm tìm cuốn sách giá lớn nhất:**
  - Bước 1: Tìm giá lớn nhất
  - Bước 2: Duyệt mảng sách, nếu cuốn sách nào có giá = giá lớn nhất (tìm được ở bước 1) thì xuất ra màn hình.

1.2.3 Đệ quy

Bài 1: Nhập số n > 0 và tính S(n) = 1 + 2 + ... + n.

Hướng dẫn:

- Xác định bước đệ quy: S(n) = S(n – 1) + n
- Xác định điều kiện dừng: S(0) = 0

Hàm đệ quy tính tổng S:

```
int Tong(int n)
{
    // Điều kiện dừng
```

```
    if (n == 0)
        return 0;
    // Phần đệ quy
    return Tong(n – 1) + n;
}
```

Bài 2: Nhập số n > 0 và tính số hạng thứ n của dãy Fibonacci.

Hướng dẫn:

- Xác định bước đệ quy: f(n) = f(n – 1) + f(n – 2)
- Xác định điều kiện dừng: f(0) = 1 và f(1) = 1

Hàm đệ quy tính số hạng Fibonacci thứ n:

```
int Fibonacci(int n)
{
    // Điều kiện dừng
    if (n == 0 || n == 1)
        return 1;
    // Phần đệ quy
    return Fibonacci(n – 1) + Fibonacci(n – 2);
}
```

Bài 3: Tìm ước chung lớn nhất (UCLN) của hai số nguyên dương a và b.

Hướng dẫn:

- Xác định bước đệ quy:
  - Nếu a > b: UCLN(a, b) = UCLN(a – b, b)
  - Nếu a < b: UCLN(a, b) = UCLN(a, b - a)
- Xác định điều kiện dừng: Nếu a = b, UCLN(a, b) = a

Hàm đệ quy tìm ước chung lớn nhất của hai số nguyên dương:

```
int UCLN(int a, int b)
{
    // Điều kiện dừng
    if (a == b)
        return a;
```

```
    // Phần đệ quy
    if (a > b)
        return UCLN(a – b, b);
  else
        return UCLN(a, b – a);
}
```

Bài 4: Viết hàm đệ quy tính tổng các phần tử của mảng số nguyên.

Hướng dẫn:

- Nếu mảng có 1 phần tử: tổng chính là giá trị phần tử đó.
- Nếu mảng có n phần tử (n>1), ta tính tổng của (n-1) phần tử đầu, sau đó cộng với phần tử cuối cùng.

Hàm tính tổng các phần tử:

```
int Tong(int a[], int n)
{
  if (n == 1)
      return a[0];
  return Tong(a, n-1) + a[n-1];
}
```

1.3 THỰC HÀNH NÂNG CAO

Bài 1: Hãy làm lại câu 2 với yêu cầu mã sách có kiểu là chuỗi tối đa 10 ký tự.

Bài 2: Viết chương trình (dạng hàm) cho phép:

a. Nhập danh sách gồm n sinh viên (với n nhập từ bàn phím). Mỗi sinh viên gồm các thông tin mã sinh viên, họ tên, ngày sinh, và điểm thi.

b. Xuất danh sách vừa nhập ra màn hình theo định dạng sau:

c. Mã sinh viên | Họ tên | Ngày sinh | Điểm thi

d. Tìm một sinh viên có họ tên là x (x do người dùng nhập vào từ bàn phím).

e. In ra màn hình các sinh viên có điểm thi >= 5.

f. Đếm số lượng sinh viên có điểm thi > 8.

g. Cho biết sinh viên có điểm thi cao nhất.

h. Đếm số lượng sinh viên có điểm thi cao nhất.

i. Sắp xếp danh sách sinh viên theo điểm thi tăng dần.

j. Sắp xếp danh sách sinh viên tăng dần theo mã sinh viên.

Bài 3: Viết chương trình (dạng hàm) cho phép:

a. Nhập danh sách gồm n phân số (với n nhập từ bàn phím).

b. Xuất danh sách vừa nhập ra màn hình.

c. Rút gọn phân số.

d. Tính tổng, hiệu, tích hai phân số.

e. So sánh hai phân số.

f. Tính tổng các phân số có trong dãy.

g. Tìm phân số lớn nhất có trong dãy.

h. Đếm số phần tử lớn nhất có trong dãy.

i. Sắp xếp dãy phân số theo chiều tăng dần.

Bài 4: Viết hàm đệ quy tính n giai thừa: n!

Bài 5: Viết hàm đệ quy tính giá trị của biểu thức: 1^3+2^3+…+n^3

Bài 6: Viết hàm đệ quy tính giá trị của biểu thức: -1+2-3+4-…+(-1)^n n

Bài 7: Viết hàm đệ quy tính giá trị của biểu thức: 1/2+2/3+3/4+…+n/(n+1)

Bài 8: Viết hàm đệ quy tính giá trị của biểu thức: 1×2×3+2×3×4+…+n×(n+1)×(n+2)

Bài 9: Viết hàm đệ quy tính giá trị của biểu thức: 1/(2×3)+1/(3×4)+…+ 1/(n×(n+1))

Bài 10: Viết hàm đệ quy tính C(n, k) biết:

C(n, k) = 1 nếu k = 0 hoặc k = n

C(n, k) = 0 nếu k > n

C(n ,k) = C(n-1, k) + C(n-1, k-1) nếu 0<k<n

Bài 11: Viết hàm đệ quy đổi một số thập phân sang hệ cơ số khác.

Bài 12: Viết hàm đệ quy tìm giá trị max của một mảng bằng đệ quy.

Bài 13: Viết hàm đệ quy xuất ngược một mảng.

Bài 14: Viết hàm đệ quy đếm số lượng các phần tử phân biệt trong một mảng.

Bài 15: Viết hàm đệ quy tính tổng các phần tử chẵn của mảng số nguyên.

Bài 16: Giải bài toán tháp Hà Nội bằng đệ quy.

BÀI 2: DANH SÁCH LIÊN KẾT

Sau khi học xong bài này, học viên có thể:

- *Hiểu được cấu trúc danh sách.*
- *Cài đặt danh sách theo hai phương pháp: cài đặt danh sách theo kiểu kế tiếp và kiểu liên kết.*
- *Hiện thực danh sách liên kết, áp dụng vào bài toán thực tế.*

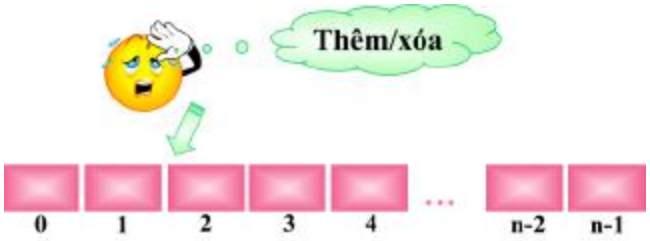
2.1 TÓM TẮT LÝ THUYẾT

2.1.1 Cấu trúc danh sách

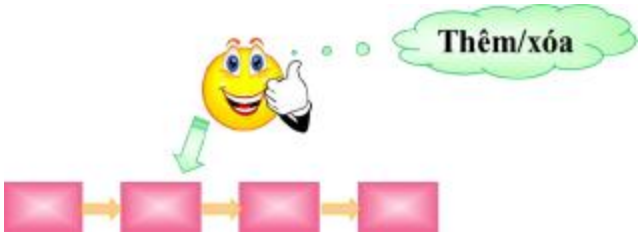
Danh sách là một kiểu dữ liệu trừu tượng gồm nhiều phần tử có cùng kiểu dữ liệu, các nút trong danh sách có thứ tự.

Có hai cách cài đặt danh sách:

- Cài đặt theo kiểu kế tiếp, ta có danh sách kề: MẢNG MỘT CHIỀU.



- Cài đặt theo kiểu liên kết, ta có các loại danh sách liên kết: DSLK ĐƠN, DSLK KÉP, DSLK VÒNG.

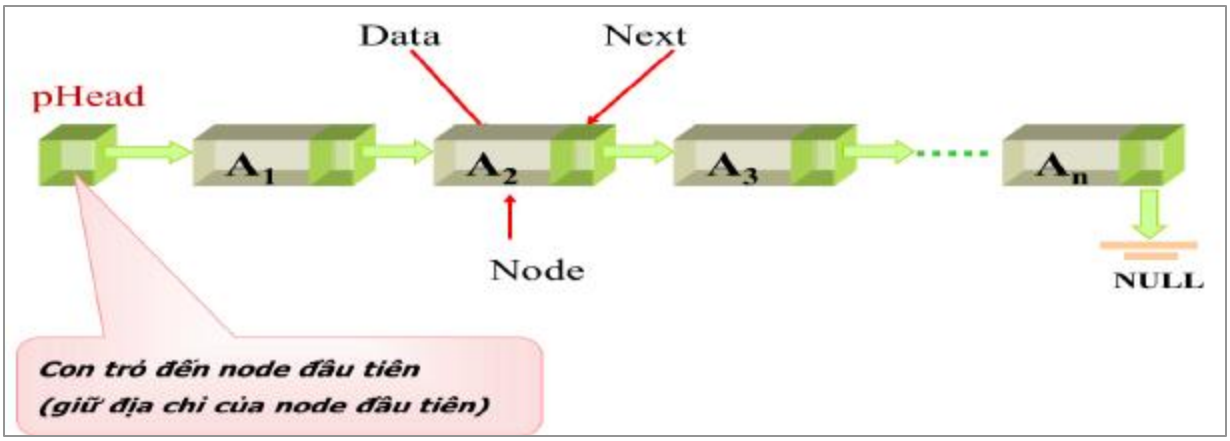


2.1.2 cấu trúc danh sách liên kết đơn

a. Định nghĩa

DSLK đơn là một danh sách các nút, mỗi nút gồm 2 thành phần:

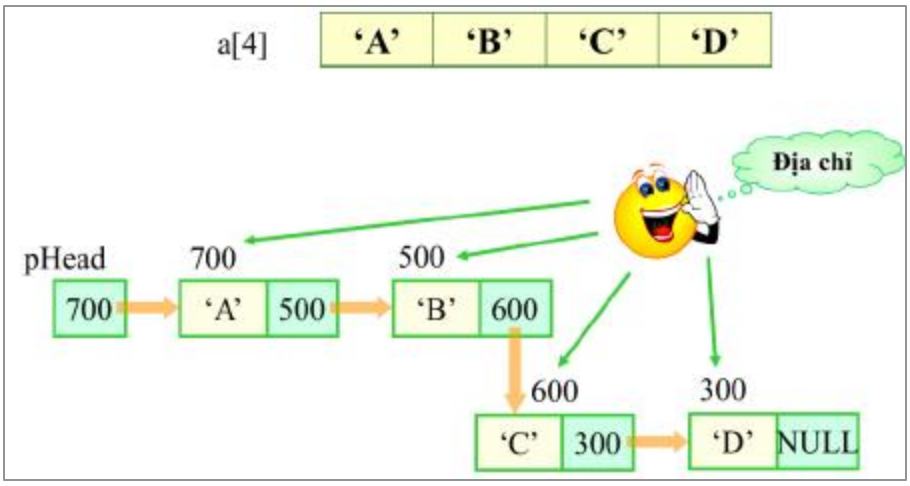
- Phần chứa dữ liệu – **Info (hoặc Data)**
- Phần chỉ vị trí (địa chỉ) của phần tử tiếp theo trong danh sách – **Next**



b. Định nghĩa cấu trúc một nút (tương ứng 1 phần tử trong danh sách)

```
typedef struct node{
    DataType      info;
    struct node *  next;
}Node;
Node*    pHead; //pHead quản lý ds
```

Ví dụ minh họa DSLK đơn:



c. Các thao tác cơ bản trên dslk đơn:

- Khởi tạo danh sách
- Tạo một nút có dữ liệu X
- Thêm một nút vào đầu danh sách
- Thêm một nút vào sau một nút cho trước
- Xóa nút đầu danh sách
- Xóa nút đứng sau một nút cho trước
- Xóa toàn bộ danh sách
- Duyệt danh sách.

2.2 THỰC HÀNH CƠ BẢN

2.2.1 DSLK đơn với dãy số nguyên

Bài 1: Sử dụng cấu trúc dữ liệu danh sách liên kết đơn để lưu một dãy số nguyên, viết chương trình thực hiện:

a. Nhập vào một dãy số nguyên

b. Xuất dãy số nguyên ra màn hình

c. Tìm phần tử có giá trị x. Nếu tìm thấy, chèn thêm phần tử mới có giá trị y vào sau x (với x, y nhập từ bàn phím).

d. Đếm số nút trên danh sách.

e. Cho biết giá trị của node thứ k trong danh sách (k bắt đầu từ 0).

f. Tìm phần tử lớn nhất (nhỏ nhất) trong danh sách.

g. Xóa 1 phần tử có khóa là x.

h. Sắp xếp danh sách tăng dần theo phương pháp Interchange Sort.

Hướng dẫn: *Hướng dẫn sau đây chỉ có tính chất minh họa, sinh viên có thể làm theo cách khác.*

- Khai báo cấu trúc node:

```
struct node
{
	datatype info;		//lưu thông tin của mỗi phần tử trong danh sách
	struct node* next;	//biến con trỏ quản lý, lưu địa chỉ của nút tiếp theo
};
typedef struct node Node;
```

- Vì danh sách có mỗi phần tử là một số nguyên nên kiểu dữ liệu của info là **int**.
- Để thực hiện câu (a) và (b) cần cài đặt các hàm sau:
  - Khởi tạo danh sách liên kết: *void init (Node* &phead)*
  - Kiểm tra danh sách có rỗng hay không: *int isEmpty(Node* phead)*
  - Tạo một node có dữ liệu X: *Node* createNode(int x)*
  - Thêm phần tử mới vào đầu danh sách: *void insertFirst(Node* &phead, int x)*
  - Thêm phần tử mới vào cuối danh sách: *void insertLast(Node* &phead, int x)*
  - Xuất danh sách ra màn hình: *void showList(Node* phead)*
- Hướng dẫn hàm nhập danh sách

Cách 1 Nhập danh sách với số lượng phần tử biết trước, số lượng phần tử do người dùng nhập.

```
void input(Node* &phead)
{
	init(phead); //khởi tạo danh sách liên kết ban đầu chưa có nút nào
	//tạo danh sách gồm n số nguyên
int n, x;
printf("Nhap so luong phan tu: "); scanf("%d", &n);
for(int i=0; i<n; i++)
{
		printf("Nhap so can them vao danh sach: ");
		scanf("%d", &x);
		insertFirst(phead, x);
}
}
```

- Bạn hãy chạy thử với n = 3, thêm lần lượt x = 1, x = 2, và x = 3. Quan sát kết quả và cho nhận xét.

- Thay gọi hàm ***insertFirst(phead, x)*** trong vòng for bằng hàm ***insertLast(phead, x)***, chạy thử với n=3, thêm lần lượt x= 1, x=2, và x=3. Quan sát kết quả và cho nhận xét.

Cách 2: Nhập vào một dãy số nguyên đến khi gặp số 0 thì dừng.

```
void input(Node* &phead)
{
	int x;
	do{
		printf("Nhap gia tri x: "); scanf("%d", &x);
		if(x!=0)
		{
			Node* p=CreateNode(x);
			if (p!=NULL)
				InsertFirst(phead, x);
		}
	}while(x!=0);
}
```

Cách 3:

Hàm main() cài đặt theo menu chức năng cho phép nhập danh sách tùy ý người dùng, số lượng phần tử không biết trước. Khi nhập danh sách, muốn tạo danh sách có bao nhiêu phần tử chỉ cần chọn chức năng thêm bấy nhiêu lần.

```
//Hàm chính
int main()
{
  //khai báo các biến quản lý danh sách
  Node* phead; //biến trỏ đến nút đầu tiên trong danh sách
  init(phead); //khởi tạo danh sách liên kết ban đầu chưa có nút nào
  //dùng một biến nguyên để lưu công việc mà người dùng chọn
  int chon, x;
  do{
	system(cls);
	//nhập chọn lựa của người dùng
	printf("1: Them 1 phan tu vao dau\n");
	printf("2: Them 1 phan tu vao cuoi\n");
```

```
            printf("3: Xuat danh sach\n");
            printf("0: Thoat\n");
            printf("Hay chon cong viec:"); scanf("%d", &chon);
            //thực hiện công việc cho lựa chọn tương ứng
            switch (chon){
                  case 1:   /Thêm đầu
                        printf("Nhap gia tri phan tu can them:");
                  scanf("%d", &x);
            insertFirst(phead, x);
                  break;
                  case 2:   //Thêm cuối
                        printf("Nhap gia tri phan tu can them:");
                  scanf("%d", &x);
            insertLast(phead, x);
                  break;
                  case 3:         //Xuất danh sách
                        showList(phead);
                  break;
                  default: chon=0;
            }
      }while (chon!=0);
   return 0;
}
```

Câu c: Vận dụng 2 hàm:

- Tìm phần tử có giá trị x
- Thêm một phần tử vào sau nút p cho trước.

2.2.2 DSLK đơn với danh sách sinh viên

Bài 1: Viết chương trình quản lý danh sách sinh viên (sử dụng DSLKĐ), thông tin mỗi sv gồm:

- Mã sv - chuỗi tối đa 10 kí tự
- Họ tên - chuỗi tối đa 40 kí tự
- Điểm trung bình - số thực.

Chương trình có các chức năng sau:

a. Nhập danh sách sinh viên.

b. Xuất danh sách sinh viên.

c. Xuất thông tin các sv có DTB>5.

d. Tìm sinh viên có mã là X.

e. Sắp xếp danh sách tăng dần theo điểm trung bình.

f. Thêm một sinh viên vào sau sinh viên có mã là X. (Vận dụng hàm tìm SV có mã X và hàm thêm một nút vào sau nút *p* cho trước).

g. Xóa SV đầu danh sách.

h. Xóa SV cuối danh sách.

i. Xóa toàn bộ danh sách.

j. Xóa SV có mã là X.

Hướng dẫn:

- Khai báo cấu trúc node:

```
//ĐỊNH NGHĨA KIỂU DỮ LIỆU SINH VIÊN
typedef struct SinhVien
{
      char masv[10];
      char hoten[40];
      float dtb;
}SV ;
typedef struct node
{
      SV info;       //lưu thông tin của mỗi phần tử trong danh sách là 1 SV
      struct node* next;  //lưu địa chỉ của nút tiếp theo
}Node;
```

- Để làm câu 1, 2 cần cài đặt các hàm sau (lưu ý rằng mỗi phần tử trong danh sách là một sinh viên nên kiểu dữ liệu của info là SV).

  - Nhập thông tin cho 1 sinh viên: *nhap1SV(SV &x)*

- Xuất thông tin cho 1 sinh viên: *xuat1SV(SV x)*
- Khởi tạo danh sách liên kết: *void init (Node* &phead)*
- Kiểm tra danh sách có rỗng hay không: *int isEmpty(Node* phead)*
- Tạo một node có dữ liệu X: *Node* createNode(SV x)*
- Thêm phần tử mới vào đầu danh sách: *void insertFirst(Node* &phead, SV x)*
- Thêm phần tử mới vào cuối danh sách: *void insertLast(Node* &phead, SV x)*
- Xuất danh sách ra màn hình: *void showList(Node* phead).*

- Nhập danh sách sinh viên theo cách 1:

```
void inputList(Node* &phead)
{
int n;
printf("Nhap so luong sinh vien:"); scanf("%d", &n);
for(int i=0; i<n; i++)
{
        SV x;
        printf("Nhap thong tin SV can them:");
        nhap1SV(x);
        insertFirst(phead, x);
        // hoặc insertLast(phead, x);
    }
}
```

2.3 THỰC HÀNH NÂNG CAO

Bài 1: Cài đặt tiếp các chức năng sau:

a. Xóa một sinh viên sau sinh viên có mã là X

b. Xóa tất cả các sinh viên có tên là X.

c. Sắp xếp DSSV tăng dần theo Mã sinh viên

d. In danh sách các SV được xếp loại khá.

e. Cho biết SV có điểm trung bình cao nhất / thấp nhất.

f. Cho biết SV có điểm trung bình thấp nhất trong số các SV xếp loại giỏi.

Bài 2: Làm lại bài tập số 9 của bài thực hành 1 (Mảng – Đệ quy) bằng danh sách liên kết đơn.

Bài 3: Cài đặt các thao tác trên DSLK kép, dữ liệu mỗi nút trong danh sách là một số nguyên.

Bài 4: Cài đặt các thao tác trên DSLK vòng đơn, dữ liệu mỗi nút trong danh sách là một số nguyên.

BÀI 3: CẤU TRÚC QUEUE VÀ STACK

Sau khi thực hành xong bài này, sinh viên có thể nắm được:

- Cấu trúc Stack, Queue, cài đặt các thao tác và ứng dụng.

3.1 TÓM TẮT LÝ THUYẾT

3.1.1 Stack (Ngăn xếp)

Stack có thể được xem là một dạng danh sách đặc biệt trong đó thao tác thêm vào và lấy ra (xóa) một phần tử chỉ diễn ra ở một đầu gọi là đỉnh *Stack*. Trên *Stack* các nút được thêm vào sau lại được lấy ra trước nên cấu trúc *Stack* hoạt động theo cơ chế vào sau ra trước - LIFO (Last In First Out).

Hai thao tác chính trên *Stack:*

- Tác vụ *push* dùng để thêm một phần tử vào đỉnh *Stack.*
- Tác vụ *pop* dùng để xoá đi một phần tử ra khỏi đỉnh *Stack*.

3.1.2 *Queue* (hàng đợi)

Queue (hàng đợi) là một dạng danh sách đặc biệt, trong đó chúng ta chỉ được phép thêm các phần tử vào cuối hàng đợi và lấy ra các phần tử ở đầu hàng đợi. Vì phần tử thêm vào trước được lấy ra trước nên cấu trúc hàng đợi còn được gọi là cấu trúc FIFO (First In First Out). Hai thao tác chính trên hàng đợi:

- insert – thêm nút mới vào cuối hàng đợi.
- remove – dùng để lấy một phần tử ra khỏi hàng đợi.

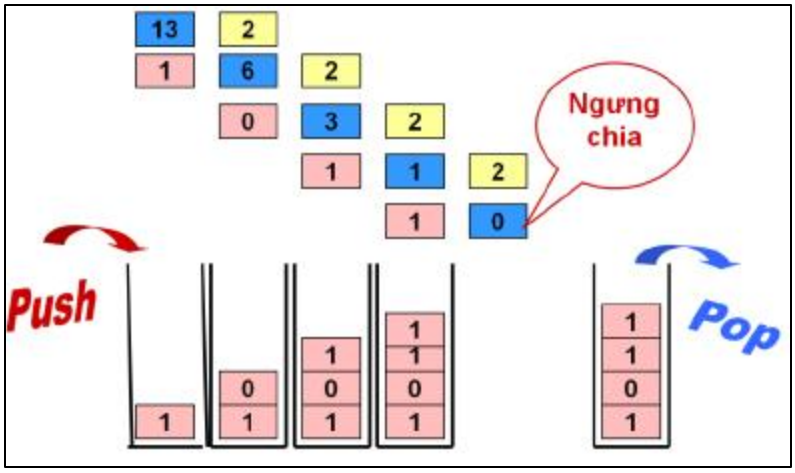
3.2 THỰC HÀNH CƠ BẢN

3.2.1 Ứng dụng Stack

Bài 1: Viết chương trình đổi một số thập phân sang cơ số nhị phân vận dụng *Stack*.

Hướng dẫn:

- Sử dụng Stack để chứa số dư khi chia số thập phân cho 2. Như vậy, Stack sẽ chứa các bit của số ở dạng nhị phân.



- **Bước 1:** Cài đặt Stack và các tác vụ của Stack bằng danh sách liên kết, mỗi phần tử của Stack kiểu int.
  - Khai báo cấu trúc dữ liệu Stack. Thay đổi Datatype cho phù hợp yêu cầu của đề bài.

```
typedef struct node
{
    DataType    info;
    struct node * next;
}Node;
typedef  Node* STACK;
```

- Cài đặt các hàm:
  - Khởi tạo Stack: void Init(STACK &s)
  - Kiểm tra Stack rỗng: int IsEmpty(STACK s)
  - Tác vụ thêm một phần tử vào Stack: void Push(STACK &s, DataType x)

- Tác vụ lấy một phần tử ra khỏi Stack: int Pop(STACK &s, DataType &x)

- **Bước 2.** Vận dụng Stack để đổi một số thập phân *n* sang nhị phân. Mã giả hàm đổi số như sau:

```
void Convert(int n, Stack &s){
    int sodu; //biến chứa số dư khi chia n cho 2
    Lặp chừng nào (n≠ 0)
    {   sodu=n%2;
        Bỏ sodu vào Stack; //gọi hàm Push() hay Pop() ???
        n=n/2;
    }
```

- Xuất số ở dạng nhị phân ra màn hình: Lấy lần lượt các bit trong Stack và xuất ra màn hình.

```
void Output(Stack s){
    Lặp chừng nào (s khác rỗng)
    {
        Lấy bit x từ Stack s;
        Xuất x;
    }
```

- Hàm chính main()

```
int main()
{
    //khai báo các biến quản lý Stack
    STACK s;
    init(s); //khởi tạo Stack ban đầu chưa có gì
    //Nhập số nguyên n cần đổi
    printf("Nhap so o he thap phan: "); scanf("%d", &n);
    //Gọi thực hiện việc đổi số
    Convert(n, s);
    //Xuất số ở dạng nhị phân ra màn hình
    Output(s) ;
    return 0 ;
}
```

Bài 2: Viết chương trình đổi biểu thức trung tố sang hậu tố.

Bài 3: Viết hàm đổi biểu thức trung tố sang tiền tố.

Bài 4: Viết hàm tính giá trị của biểu thức trung tố.

Bài 5: Viết hàm tính giá trị của biểu thức tiền tố.

3.2.2 Ứng dụng Queue

Bài 1: Viết chương trình quản lý danh sách các bệnh nhân chờ khám ở một phòng khám tư, sử dụng cấu trúc Queue để lưu danh sách bệnh nhân tới khám. Chương trình gồm các chức năng:

a. Thêm 1 bệnh nhân vào Queue.

b. Lấy bệnh nhân tiếp theo sẽ khám

c. Cho biết số lượng người đã khám

d. Cho biết số lượng người chưa khám.

e. Xuất danh sách bệnh nhân còn trong hàng đợi chờ khám.

Hướng dẫn:

- Cài đặt cấu trúc dữ liệu Bệnh nhân: số thứ tự, họ tên, tuổi. Họ tên và tuổi do người dùng nhập, số thứ tự do chương trình tự phát sinh theo số người đến khám.
- Cài đặt một Queue chứa các bệnh nhân chờ khám.

```
typedef struct benhnhan{
      ...
} BenhNhan ;
typedef struct node
{
      BenhNhan      info;
      struct node *  next;
}Node;
typedef struct QUEUE
{
      Node*     pHead;
      Node*     pTail;
}Queue;
```

- Cài đặt các thao tác trên Queue: Init, IsEmpty, Insert, Remove.
- Cài đặt các chức năng theo mô tả của bài tập, chương trình phải viết theo menu chức năng.

BÀI 4: CẤU TRÚC CÂY – CÂY NHỊ PHÂN TÌM KIẾM

Sau khi thực hành xong bài này, sinh viên có thể nắm được:

- *Cấu trúc cây nhị phân.*
- *Cấu trúc cây nhị phân tìm kiếm.*

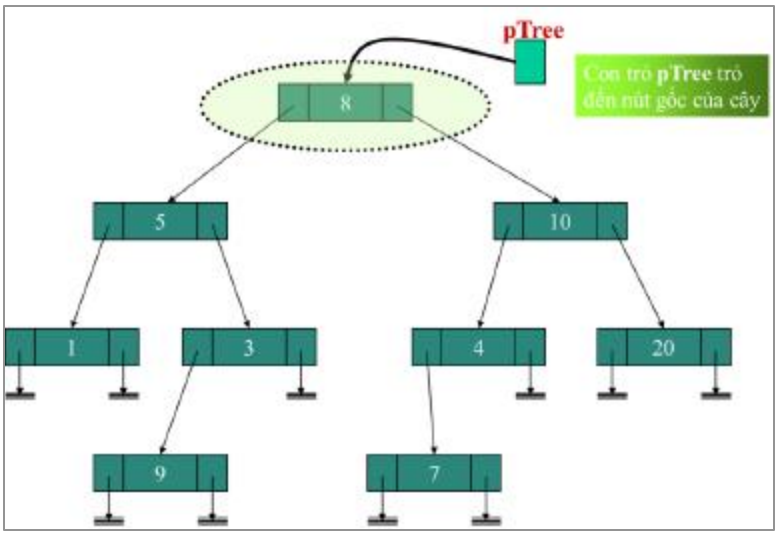
4.1 TÓM TẮT LÝ THUYẾT

4.1.1 Định nghĩa

Cây nhị phân là cấu trúc cây đơn giản nhất, mỗi nút có tối đa 2 nút con.

Tại mỗi nút gồm các 3 thành phần:

- Phần data: chứa giá trị, thông tin...
- Liên kết đến nút con trái (nếu có)
- Liên kết đến nút con phải (nếu có).



Cây nhị phân tìm kiếm là cây nhị phân mà mỗi nút thoả:

- Giá trị của tất cả nút con trái < giá trị của nút cha
- Giá trị của tất cả nút con phải > giá trị của nút cha

4.1.2 Các thao tác

Các thao tác cơ bản trên cây:

- CreateNode, FreeNode, Init, IsEmpty
- InsertLeft, InsertRight
- DeleteLeft, DeleteRight
- PreOrder, InOrder, PostOrder
- Search
- ClearTree.

4.2 THỰC HÀNH CƠ BẢN

4.2.1 Cây nhị phân

Bài 1: Cho trước 1 mảng a có n phần tử (mảng số nguyên/ mảng cấu trúc có một trường là khóa), hãy tạo một cây nhị phân có n node, mỗi nút lưu 1 phần tử của mảng.

a. Cài đặt hàm xuất giá trị các nút trên cây theo thứ tự: LNR, NLR, LRN.

b. Đếm số node trên cây.

c. Đếm số node lá.

Hướng dẫn:

- Định nghĩa cấu trúc dữ liệu cho một nút trên cây

```
typedef struct node
{
    int   info;
    struct node *  left;
    struct node *  right;
} Node;
```

- Biến quản lý cây: Node* proot; //pTree giữ nút gốc của cây.
- Cài đặt hàm nhập mảng số nguyên.
- Hướng dẫn hàm tạo cây nhị phân từ mảng số nguyên

```
void CreateTree(Node* & proot, int a[], int begin, int end )
{
      if (begin>end) return;
      int  i = (begin+end)/2;
      //tạo nút gốc có dữ liệu a[i]
      proot = CreateNode(a[i]);
      //tạo cây con trái chứa các giá trị ở nửa trái của mảng
      CreateTree(proot ->left, a, begin, i-1);
       //tạo cây con trái chứa các giá trị ở nửa trái của mảng
      CreateTree(proot ->right, a, i+1, end);
}
//trong hàm main() : begin=0, end=n-1 ;
```

- Duyệt cây theo thứ tự trước:
  - Xét nút gốc trước và xuất giá trị nút gốc
  - Duyệt cây con trái
  - Duyệt cây con phải

```
void NLR(Node * proot){
      if (proot!=NULL){
            printf("%4d", proot ->info); //xuất giá trị nút gốc
            NLR(proot ->left); //duyệt cây con trái
            NLR(proot ->right); //duyệt cây con phải
      }
}
```

4.2.2 Cây nhị phân tìm kiếm

Bài 2: Cài đặt cây nhị phân tìm kiếm, mỗi nút chứa một số nguyên, với các thao tác:

a. Cài đặt các thao tác xây dựng cây: Init, IsEmpty, CreateNode

b. Cài đặt thao tác cập nhật: Insert, Remove

c. Cài đặt thao tác xuất các giá trị trên cây

d. Chương trình viết theo menu chức năng.

Hướng dẫn:

a. Cài đặt các thao tác xây dựng cây: Init, IsEmpty, CreateNode

```
typedef struct node{
	int info;
	struct node *left;
	struct node *right;
}Node;
void Init(Node* &proot){ proot=NULL ;}
Node* CreateNode(int x){
	NODEPTR p=new Node;
	p->info=x; p->left=NULL;	p->right=NULL;
	return p;
}
```

b. Cài đặt thao tác cập nhật: Insert, Remove

```
void Insert(NODEPTR &proot, int x) {
	if (isEmpty(proot)  //nếu cây rỗng, thêm trực tiếp vào cây
		proot = CreateNode(x);
	else
		if (x==proot->info) //nếu đã có x trong cây
			return;
		if (x<proot->info)
			Insert(proot->left, x);
		else
			Insert(proot->right, x);
}
int Remove(NODEPTR &proot, int x) {
	if ( proot == NULL)
		return FALSE; //không tìm thấy nút cần xoá
	if (proot->info >x) //tìm và xóa bên trái
		return Remove(proot->left, x);
```

```
	if (proot->info <x) //tìm và xóa bên phải
		return Remove(proot->right, x);
//nếu (proot->info==x)
	Node* p, f, rp;
	p = proot;		//p biến tạm trỏ đến proot
	//trường hợp proot có 1 cây con
	if ( proot->left == NULL)	//có 1 cây con
		proot = proot->right;
	else if (proot->right == NULL)	//có 1 cây con
		proot = proot->left;
	else { //TH pTree có 2 cây con chọn nút nhỏ nhất bên cây con phải
		f = p;	//f để lưu cha của rp
		rp = p->right;	//rp bắt đầu từ p->right
		while ( rp->left != NULL)	{
			f = rp;  //lưu cha của rp
			rp = rp->left; //rp qua bên trái
		} //kết thúc khi rp là nút có nút con trái là null
		p->info = rp->info; //đổi giá trị của p và rp
		if ( f == p) //nếu cha của rp là p
			f->right = rp->right;
		else //f != p
			f->left = rp->right;
		p = rp; // ptrỏ đến phần tử thế mạng rp
	}
	delete p;	//xoá nút p
	return TRUE;
```

c. Cài đặt thao tác xuất các giá trị trên cây: xuất theo NLR hoặc LNR hoặc LRN.

- Viết hàm main() gọi thực hiện các chức năng theo menu.

Bài 3: Viết hàm thực hiện các chức năng sau trên cây nhị phân tìm kiếm:

a. Xuất danh sách tăng dần và giảm dần

b. Kiểm tra xem cây có phải là cây nhị phân đúng

c. Kiểm tra xem cây có phải là cây nhị phân đầy đủ

d. Xác định nút cha của nút chứa khoá x

e. Đếm số nút lá, nút giữa, kích thước của cây

f. Xác định độ sâu/chiều cao của cây

g. Tìm giá trị nhỏ nhất/lớn nhất trên cây

h. Tính tổng các giá trị trên cây

BÀI 5: CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG

Sau khi thực hành xong bài này, sinh viên có thể nắm được:

- *Cài đặt cây nhị phân tìm kiếm cân bằng.*
- *Cài đặt các thao tác xoay, thêm nút, xóa nút.*

5.1 TÓM TẮT LÝ THUYẾT

5.1.1 Khái niệm

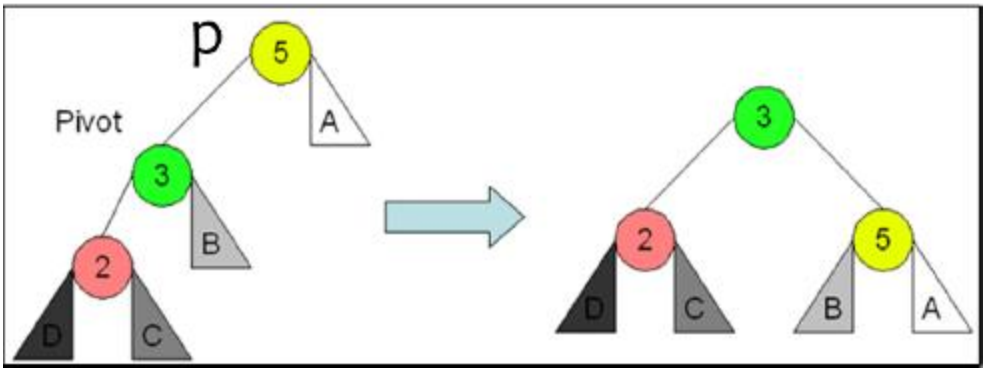
- Cây AVL do Adelson Velski và Landis xây dựng vào năm 1962.
- Cây AVL thỏa mãn điều kiện sau:
  - Là cây nhị phân tìm kiếm.
  - Tại mỗi nút p: chiều sâu của cây con bên phải và chiều sâu của cây con bên trái chênh nhau không quá 1.
- **Chỉ số cân bằng (bf - balance factor)** của nút trên cây AVL:
  - **bf = height (nút con trái) – height (nút con phải)**
  - bf(p) = 0 → nút p cân bằng
  - bf(p) = 1 → nút bị lệch trái
  - bf(p) = -1 → nút bị lệch phải

5.1.2 Các thao tác

- Kế thừa các thao tác của cây nhị phân tìm kiếm.
- Tuy nhiên, để đảm bảo tính chất của cây nhị phân tìm kiếm cân bằng, chúng ta cần cài đặt hai thao tác xoay cơ bản sau: xoay trái và xoay phải. Ngoài hai thao

tác cơ bản đó, cần cài đặt thêm thao tác cân bằng cây và các thao tác cập nhật cây (thêm và xóa nút) cũng cần được cài đặt lại.

- Tác vụ xoay phải quanh nút p:



  - Điều kiện: Nút p phải có nút con trái.
  - Các thao tác:
    - Nếu p = NULL → thoát
    - Nếu p không có nút con trái →thoát
    - Ngược lại, thực hiện thao tác xoay:

      Giả sử pivot là nút con trái của p.

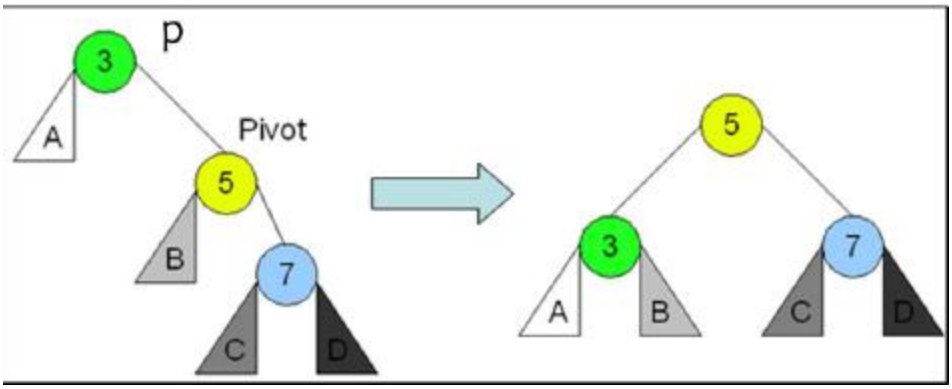
      Chuyển nút con phải của pivot thành nút con trái của p.

      Chuyển p thành nút con phải của pivot.

      Cập nhật chiều cao của nút p và nút pivot.

      Gốc mới của cây con đang xét là pivot.

- Tác vụ xoay trái quanh nút p:



- Điều kiện: Nút p phải có nút con phải.
- Các thao tác:
  - Nếu p = NULL → thoát
  - Nếu p không có nút con phải →thoát
  - Ngược lại, thực hiện thao tác xoay:

    Giả sử pivot là nút con phải của p.

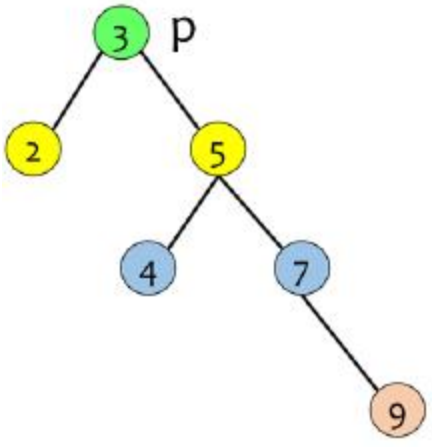
    Chuyển nút con trái của pivot thành nút con phải của p.

    Chuyển p thành nút con trái của pivot.

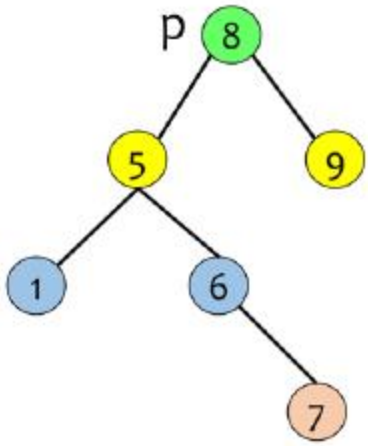
    Cập nhật chiều cao của nút p và nút pivot.

    Gốc mới của cây con đang xét là pivot.

- Thao tác cân bằng cây:
- Trường hợp 1: Nếu chỉ số cân bằng (balance factor) của nút p lớn hơn 1 và chỉ số cân bằng của nút con trái của p lớn hơn hoặc bằng 0 → chỉ cần thực hiện xoay phải.
- Trường hợp 2: Nếu chỉ số cân bằng (balance factor) của nút p lớn hơn 1 và chỉ số cân bằng của nút con trái của p nhỏ hơn 0 → cần thực hiện xoay trái tại nút con trái của p, sau đó xoay phải tại p.



Cần thực hiện xoay trái

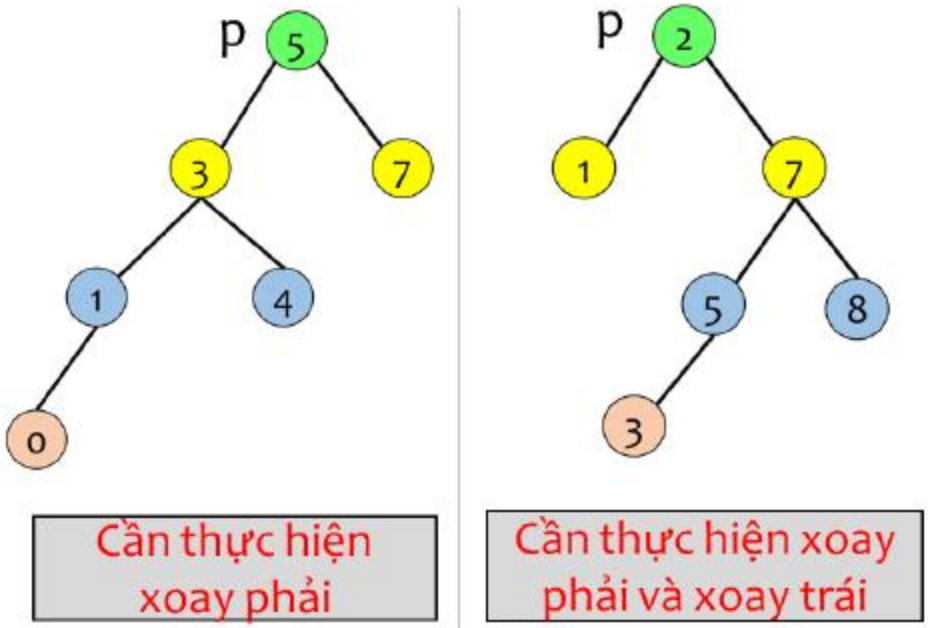


Cần thực hiện xoay trái và xoay phải

```
if (balanceFactor(p) > 1)
{
    if (balanceFactor(p->left) < 0)
        p->left = rotateLeft(p->left);

    p = rotateRight(p);
}
```

- Trường hợp 3: Nếu chỉ số cân bằng (balance factor) của nút p nhỏ hơn -1 và chỉ số cân bằng của nút con phải của p nhỏ hơn hoặc bằng 0 → chỉ cần thực hiện xoay trái.
- Trường hợp 4: Nếu chỉ số cân bằng (balance factor) của nút p nhỏ hơn -1 và chỉ số cân bằng của nút con phải của p lớn hơn 0 → cần thực hiện xoay phải tại nút con phải của p, và sau đó xoay trái tại p.




```
if (balanceFactor(p) < -1)
{
    if (balanceFactor(p->right) > 0)
        p->right = rotateRight(p->right);

    p = rotateLeft(p);
}
```

5.2 THỰC HÀNH CƠ BẢN

Bài 1: Cài đặt cây nhị phân tìm kiếm cân bằng, mỗi nút chứa một số nguyên, với các thao tác:

a. Cài đặt các thao tác xây dựng cây: Init, IsEmpty, CreateNode

b. Cài đặt thao tác cập nhật: Insert, Remove

c. Viết hàm tạo cây nhị phân cân bằng từ một mảng các số nguyên.

Chương trình viết theo menu chức năng.

Hướng dẫn:

a. Cài đặt các thao tác xây dựng cây: Init, IsEmpty, CreateNode

- Khác với cây nhị phân tìm kiếm, mỗi nút trong cây nhị phân tìm kiếm cân bằng cần phải đảm bảo chỉ số cân bằng của nút chỉ nhận một trong ba giá trị 0, 1, hoặc -1. Do vậy, đối với mỗi nút trong cây nhị phân tìm kiếm cân bằng, ta sẽ duy trì một biến height để ghi nhận chiều cao của nút. Chiều cao của nút sẽ được dùng để tính chỉ số cân bằng, từ đó, dùng để kiểm tra tính cân bằng của cây.

```
typedef struct node{
      int info;
      struct node *left;
      struct node *right;
int height;  // chứa chiều cao của nút, sử dụng để tính chỉ số cân bằng, kiểm //tra tính cân bằng của cây.
}Node;

void Init(Node* &proot){ proot=NULL ;}

Node* CreateNode(int x){
      NODEPTR p = new Node;
      If(p==NULL) return 0;
      p->info = x;
p->left = NULL;
p->right = NULL;
p->height = 1;
      return p;
}
```

b. Cài đặt thao tác cập nhật: Insert, Remove

- Để xây dựng được cây cân bằng, ta cần kiểm tra được chỉ số cân bằng của mỗi nút trong cây.
- Khi thực hiện thao tác thêm hoặc xóa nút, ngay sau khi thêm hoặc xóa, cần kiểm tra lại tính cân bằng của cây và thực hiện việc cân bằng lại cây ngay lập tức nếu cần thiết.
- Để tính được chỉ số cân bằng tại mỗi nút, trước hết ta xây dựng một số hàm sau đây:
  - Hàm getHeight dùng để lấy thông tin về chiều cao của một nút,
  - Hàm balanceFactor dùng để tính chỉ số cân bằng của một nút,

```
// Hàm lấy thông tin về chiều cao của một nút
int getHeight(node *p)
{
	if (p == NULL)
		return 0;
	return p->height;
}

// Hàm tính chỉ số cân bằng của một nút
int balanceFactor(node *p)
{
	if (p == NULL)
		return 0;

	return getHeight(p->left) - getHeight(p->right);
}

// Hàm tìm giá trị lớn nhất giữa hai số nguyên
int max(int a, int b)
{
	return a > b ? a : b;
}
```

- Trước khi cài đặt hàm để thực hiện việc thêm và xóa nút, ta cần cài đặt hàm thực hiện tác vụ xoay trái, xoay phải, và cân bằng cây.
  - Hàm rotateLeft: thực hiện tác vụ xoay trái tại một nút cho trước.

- Hàm rotateRight: thực hiện tác vụ xoay phải tại một nút cho trước.
- Hàm balance: thực hiện tác vụ cân bằng cây như mô tả ở phía trên.

```
// Hàm thực hiện tác vụ xoay trái tại nút p
node *rotateLeft(node *p)
{
	// TH1: Không xoay được do nút p rỗng
	if (p == NULL)
		return NULL;

	// TH2 : Không xoay được do nút p không có cây con phải
	if (p->right == NULL)
		return NULL;

	// TH3: Thực hiện xoay
	node *pivot = p->right;

	// chuyển cây con trái của pivot thành cây con phải của p
	p->right = pivot->left;
	// chuyển p thành cây con trái của pivot
	pivot->left = p;
	// cập nhật chiều cao của nút p và pivot
	p->height = 1 + max(getHeight(p->right), getHeight(p->left));
	pivot->height = 1 + max(getHeight(pivot->right), getHeight(pivot->left));

	return pivot;
}

//Hàm thực hiện tác vụ xoay phải tại nút p
node *rotateRight(node *p)
{
	// TH1: Không xoay được do nút p rỗng
	if (p == NULL)
		return NULL;

	// TH2: không xoay được do nút p không có cây con
	if (p->left == NULL)
```

```
            return NULL;

      // TH3: Thực hiện xoay
      node *pivot = p->left;

      // chuyển nút con phải của pivot thành nút con trái của
      p->left = pivot->right;
      // chuyển p thành nút con phải của pivot
      pivot->right = p;
      // cập nhật chiều cao của nút p và pivot
      p->height = 1 + max(getHeight(p->right), getHeight(p->left));
      pivot->height = 1 + max(getHeight(pivot->right), getHeight(pivot->left));

      return pivot;
}

// Cân bằng cây tại nút p
node * balance(node *p)
{
      if (balanceFactor(p) < -1)
      {
            if (balanceFactor(p->right) > 0)
            {
                  p->right = rotateRight(p->right);
            }
            p = rotateLeft(p);
      }
      else if (balanceFactor(p) > 1)
      {
            if (balanceFactor(p->left) < 0)
            {
                  p->left = rotateLeft(p->left);
            }
            p = rotateRight(p);
      }
      return p;
}
```

- Thêm một nút vào cây: Khi thêm một nút vào cây nhị phân tìm kiếm cân bằng AVL, cần thực hiện các thao tác sau:
  - Thêm nút x vào cây như trên cây nhị phân tìm kiếm BST
  - Cập nhật chiều cao của các nút bị ảnh hưởng
  - Cân bằng lại cây

```
// Thêm một nút vào cây và cân bằng cây
node *insert(node *p, int x)
{
	if (p == NULL)
		return createNode(x);

	if (x < p->data)		// add to the right side
		p->left = insert(p->left, x);
	else if (x > p->data)	// add to the left side
		p->right = insert(p->right, x);

	p->height = 1 + max(getHeight(p->left), getHeight(p->right));

	return balance(p); // balance the tree
}
```

- Xóa một nút: Khi xóa một nút khỏi cây nhị phân tìm kiếm cân bằng AVL, cần thực hiện các thao tác sau:
  - Thực hiện thao tác xóa nút như trên cây nhị phân tìm kiếm BST
  - Cập nhật chiều cao của các nút bị ảnh hưởng
  - Cân bằng lại cây

```
//Tìm nút có giá trị nhỏ nhất
node * minValueNode(node* p)
{
   node* current = p;

   while (current && current->left != NULL)
      current = current->left;
```

```
    return current;
}

// Xóa một nút
node * removeNode(node *root, int x)
{
      // Bước 1: thực hiện xóa như trên cây nhị phân tìm kiếm
      if (root == NULL)
            return root;

      // Nếu nút cần xóa có giá trị nhỏ hơn nút đang xét, tìm và xóa bên trái
      if (root->data > x)
            root->left = deleteNode(root->left, x);

      // Nếu nút cần xóa có giá trị lớn hơn nút đang xét, tìm và xóa bên phải
      else if (root->data < x)
            root->right = deleteNode(root->right, x);

      // Nếu nút cần xóa có giá trị bằng nút đang xét, xóa nút đang xét
    else
    {
      // Trường hợp nút cần xóa không có con hoặc chỉ có một con
      if (root->left == NULL)
      {
            node *temp = root->right;
            delete root;
            return temp;
      }
            else if (root->right == NULL)
            {
                  node *temp = root->left;
                  delete root;
                  return temp;
            }
            // Trường hợp nút cần xóa có hai con: Tìm nút có giá trị nhỏ nhất ở //nhánh
con bên phải của nút cần xóa.
```

```
            node *temp = minValueNode(root->right);

            // Chuyển nội dung của nút tìm được vào nút cần xóa
        root->data = temp->data;

        // Xóa nút temp
        root->right = deleteNode(root->right, temp->data);
       }
       // Nếu cây chỉ có duy nhất một nút, dừng
       if (root == NULL)
            return root;

       // Bước 2 : cập nhật chiều cao của nút đang xét
       root->height = 1 + max(getHeight(root->left), getHeight(root->right));

// Cân bằng cây trước khi thoát khỏi hàm
       return balance(root);

}
```

c. Viết hàm tạo cây nhị phân cân bằng từ một mảng các số nguyên.

```
node * createAVL(int a[], int n)
{
       node *root = NULL;


       for (int i = 0; i < n; i++)
            root = insert(root, a[i]);


       return root;
}
```

- Viết hàm main() gọi thực hiện các chức năng theo menu.

Bài 2: Viết chương trình kiểm tra một cây nhị phân có phải là cây nhị phân tìm kiếm cân bằng.

5.3 KIỂM TRA THỰC HÀNH

Chúc các bạn thi tốt!

TÀI LIỆU THAM KHẢO

1. Dương Anh Đức, Trần Hạnh Nhi (2000). Cấu trúc dữ liệu & thuật toán. Đại học Khoa học Tự nhiên, Tp. Hồ Chí Minh.
2. Nguyễn Trung Trực (1992). Cấu trúc dữ liệu. Khoa CNTT, Trường ĐH Bách Khoa TPHCM.
3. Lê Minh Hoàng. (1999 - 2002). Giải thuật & lập trình. Đại học Sư Phạm Hà Nội.
4. Richard Neapolitan and Kumarss (2004). Foundations of Algorithms Using C++ Pseudocode. Jones and Bartlett Publishers.
5. Nguyễn Văn Linh (2003). Giải thuật. Đại học Cần thơ.
6. Nguyễn Hà Giang (2008). Bài giảng Cấu trúc dữ liệu và Giải thuật. Đại học Kỹ thuật Công nghệ, Tp. Hồ Chí Minh.
7. ThS. Văn Thị Thiên Trang, Cấu trúc dữ liệu và giải thuật (2019). Đại học Kỹ thuật Công nghệ, Tp. Hồ Chí Minh.