CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
Data Structure and algorithms

TPHCM - 2023

2

CHƯƠNG 1: TỔNG QUAN

Nội dung

- Vai trò của cấu trúc dữ liệu trong một đề án tin học
- Mối quan hệ giữa GT & CTDL
- Các tiêu chuẩn đánh giá cấu trúc dữ liệu
- Kiểu dữ liệu
- Đánh giá độ phức tạp của giải thuật

Vai trò của CTDL trong một đề án tin học

- Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính.
- Một bài toán thực tế bất kỳ đều bao gồm *dữ liệu* và *các yêu cầu xử lý trên dữ liệu đó*
- **Theo bạn: trước khi viết một chương trình để giải quyết một bài toán nào đó trên máy tính thì cần phải làm những việc gì?**

→ Tổ chức biểu diễn các đối tượng dữ liệu → Xây dựng cấu trúc dữ liệu (Chọn CTDL phù hợp).

→ Xây dựng các thao tác xử lý dữ liệu → xây dựng giải thuật

Vai trò của CTDL trong một đề án tin học

Xây dựng cấu trúc dữ liệu

- Sự tổ chức hợp lý của các thành phần dữ liệu,
- Các thành phần dữ liệu thực tế đa dạng, phong phú, quan hệ với nhau=> cần phải tổ chức , xây dựng các cấu trúc thích hợp nhất để:
  - Phản ánh chính xác các dữ liệu thực tế
  - Dễ dàng dùng máy tính để xử lý
- Ví dụ:
  - Mảng (array)
  - Danh sách liên kết (linked list)
  - Ngăn xếp (stack)
  - Hàng đợi (queue)
  - Cây (tree)
  - …

Vai trò của CTDL trong một đề án tin học

Xây dựng cấu trúc dữ liệu

- Dữ liệu có thể là:
  - Input data
  - Trung gian
  - Output

Vai trò của CTDL trong một đề án tin học

Xây dựng các thao tác xử lý dữ liệu

- Từ những yêu cầu thực tế, cần tìm ra các giải thuật tương ứng để xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn → đây là bước *xây dựng giải thuật* cho bài toán.
- Giải thuật hay còn gọi là thuật toán là phương pháp hay cách thức (method) để giải quyết vấn đề.
- Tập các thao tác để truy cập các thành phần dữ liệu.
- Tập các bước *có thể tính toán được* để đạt được kết quả mong muốn

Vai trò của CTDL trong một đề án tin học

Thuật toán:

Ví dụ: Tính tổng các số nguyên lẻ từ 1→n

- B1: S=0
- B2: i=1
- B3: Nếu i=n+1 thì sang B7, ngược lại sang B4
- B4: S=S+i
- B5: i=i+2
- B6: Quay lại B3
- B7: Tổng cần tìm là S

Vai trò của CTDL trong một đề án tin học

Xây dựng các thao tác xử lý dữ liệu

- Cách mô tả giải thuật:
  - Ngôn ngữ tự nhiên (natural language)
  - Sơ đồ (flow chart)
  - Mã giả (pseudo code) thường được sử dụng
- Ví dụ: Mô tả giải thuật giải phương trình
  - $ax+b=0$
  - $ax^2+bx+c=0$

Vai trò của CTDL trong một đề án tin học

Mã giả:

Khai báo thuật toán

Thuật toán <tên TT> (<tham số>)

Input: <dữ liệu vào>

Output: <dữ liệu ra>

<Các câu lệnh>

End <tên TT >

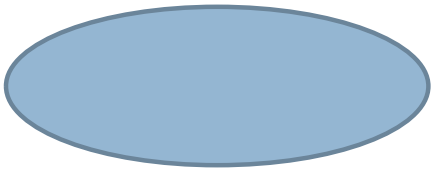
Vai trò của CTDL trong một đề án tin học

VD: Giải thuật giải PT ax+b=0

Thuật toán Giai_PT

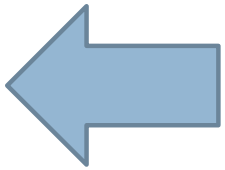
```
Input: hệ số a,b
Output: nghiệm của PT
Begin
        Nhập vào các hệ số a,b
        if (a ≠ 0) PT có nghiệm x=-b/a
        else
            if (b ≠ 0) PT vô nghiệm
            else PT vô số nghiệm
 End Giai_PT
```

Các ký hiệu lưu đồ

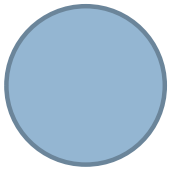


Bắt đầu/ kết thúc

Rẽ nhánh



Giá trị trả về



Điểm nối

Luồng xử lý

Khối xử lý

Nhập/ Xuất

Ký hiệu mã giả

- IF <điều kiện> THEN …ENDIF
- IF <điều kiện> THEN ... ELSE ... ENDIF
- WHILE <điều kiện> DO … ENDWHILE
- DO … UNTIL <điều kiện>
- DISPLAY …
- RETURN …

Ví dụ mô tả giải thuật

Tìm ước số chung lớn nhất của 2 số nguyên dương a và b

- Đầu vào: 2 số nguyên dương a và b
- Đầu ra: ước số chung lớn nhất của a và b

Mô tả bằng mã tự nhiên

Bước 1: Nếu a = b thì kết luận a là ước số chung lớn nhất, kết thúc

Bước 2: Nếu a > b thì a = a – b;

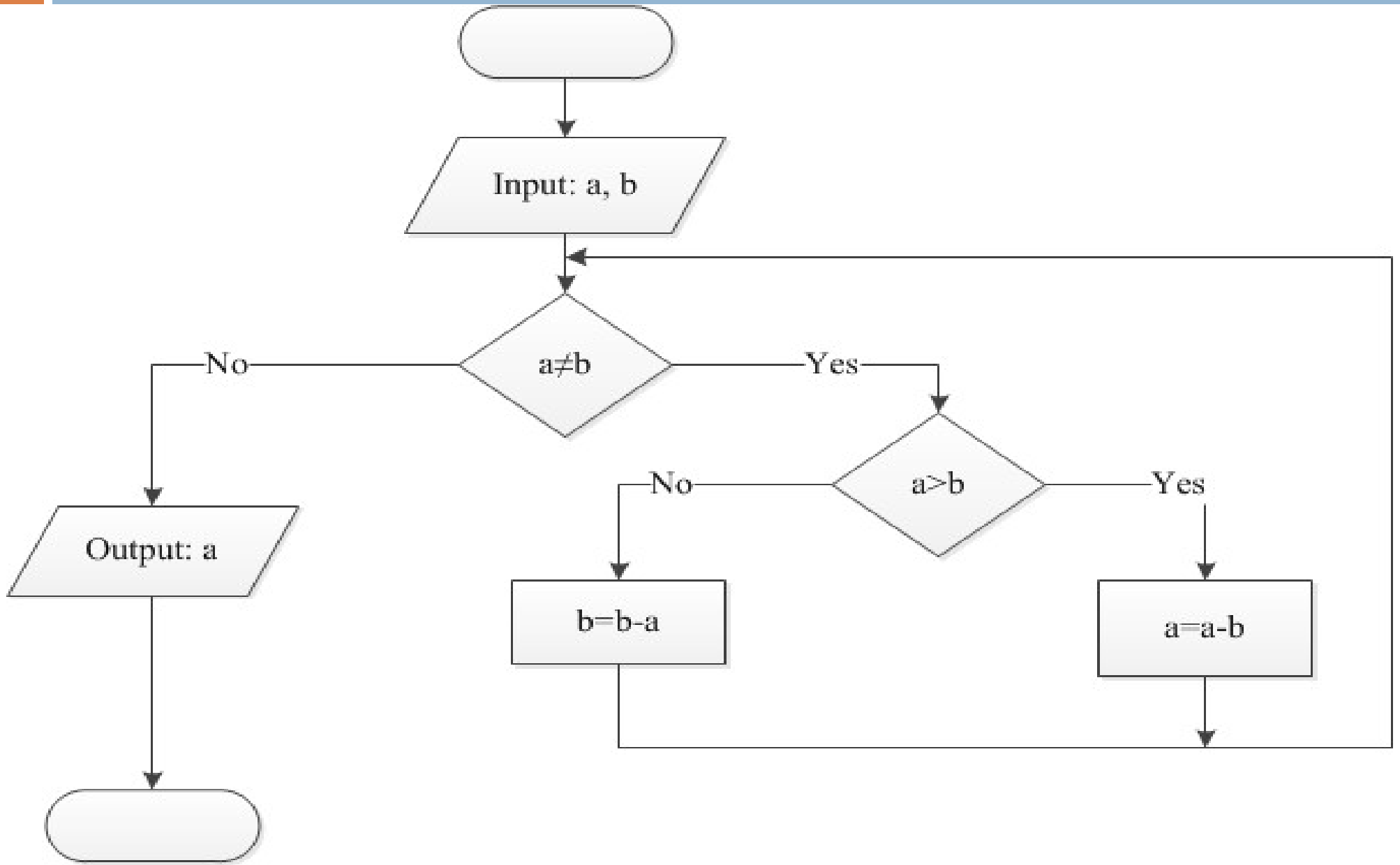
Ngược lại thì b = b – a;

Bước 3: Quay trở lại Bước 1

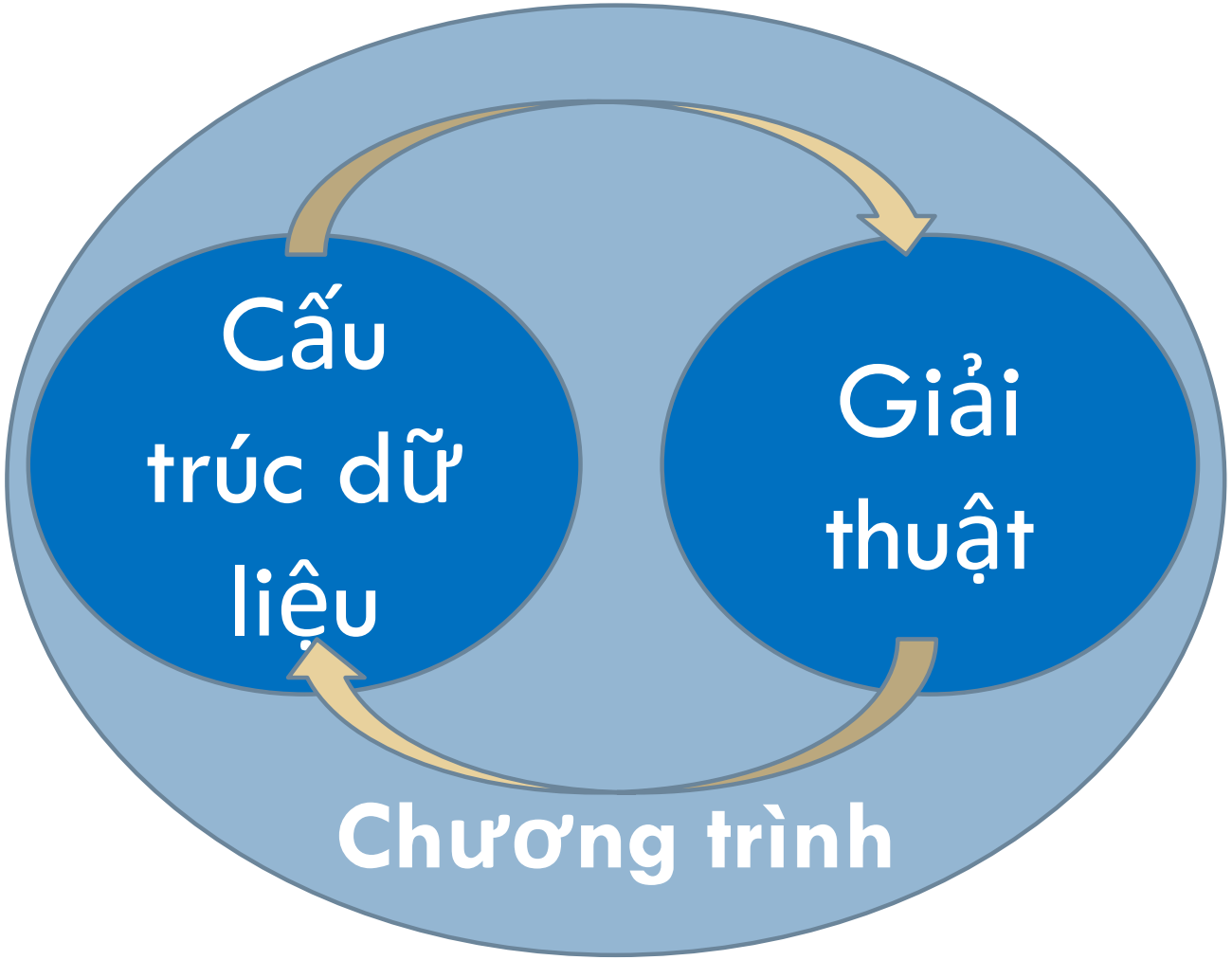
Mô tả bằng mã giả


```
WHILE a ≠ b DO
        IF a>b THEN
                a=a-b
        ELSE
                b=b-a
        ENDIF
ENDWHILE
DISPLAY a
```

Mô tả bằng lưu đồ



Mối quan hệ giữa CTDL và Giải thuật



CTDL + Giải Thuật = Chương trình

Ví dụ

- Một chương trình quản lý điểm thi của sinh viên cần lưu trữ các điểm số của 3 sinh viên. Giả sử mỗi sinh viên có 4 điểm số ứng với 4 môn học khác nhau, dữ liệu có dạng bảng như sau:

Sinh viên	Môn 1	Môn 2	Môn3	Môn4
SV 1	7	9	5	2
SV 2	5	0	9	4
SV 3	6	3	7	4

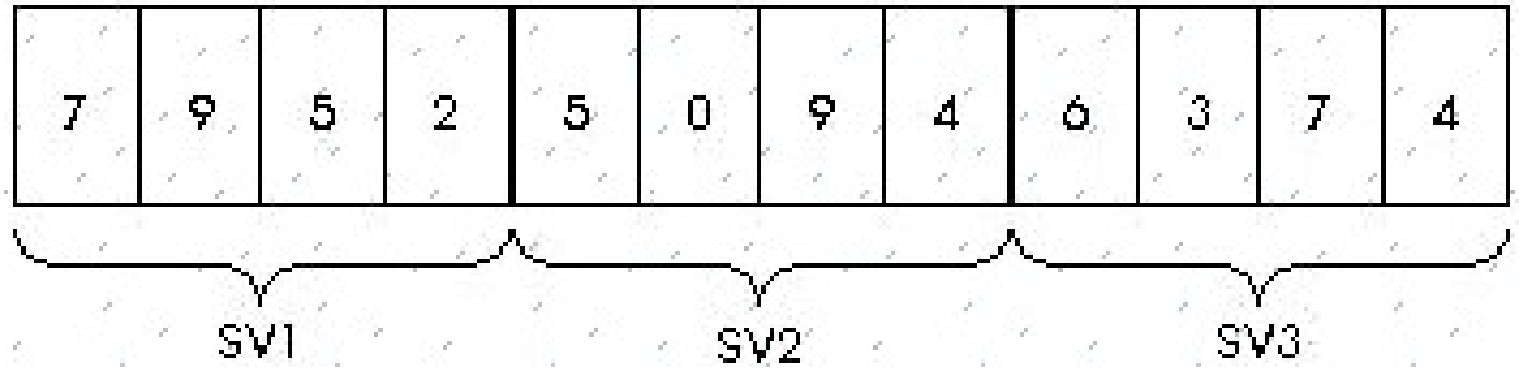
- Chỉ xét thao tác xử lý là xuất điểm số các môn của từng sinh viên.

Ví dụ

- Phương án 1 : Sử dụng mảng một chiều:

  int result [12] = {7, 9, 5, 2, 5, 0, 9, 4, 6, 3, 7, 4};

  - các phần tử sẽ được lưu trữ như sau:



  - Truy xuất điểm số môn j của sinh viên i phải sử dụng một công thức xác định chỉ số tương ứng trong mảng result:

```
result[(i*số cột) + j]
```

Ví dụ


```
void XuatDiem()//Xuất điểm số của tất cả sv
{
  const int so_mon = 4;
  int  sv, mon;
  for (int i=0; i<12; i++)
  {
      sv = i/so_mon;
      mon = i % so_mon;
      printf("Điểm môn  %d của sv %d là: %d”,
      mon, sv, result[i];
  }
}
```

Ví dụ

- Phương án 2 : Sử dụng mảng hai chiều:

  int result[3][4] ={{ 7, 9, 5, 2}, { 5, 0, 9, 4}, { 6, 3, 7, 4 }};

  - các phần tử sẽ được lưu trữ như sau:

	Cột 0	Cột 1	Cột 2	Cột 3
Dòng 0	result[0][0] =7	result[0][1] =9	result[0][2] =5	result[0][3] =2
Dòng 1	result[1][0] =5	result[1][1] =0	result[1][2] =9	result[1][3] =4
Dòng 2	result[2][0] =6	result[2][1] =3	result[2][2] =7	result[2][3] =4

  - Truy xuất điểm số môn j của sinh viên i cũng chính là phần tử nằm ở vị trí (dòng i, cột j) trong mảng: result[i][j]

Ví dụ


```
void XuatDiem()//Xuất điểm số của tất cả sv
{
  const int so_mon = 4, so_sv = 3;
  for ( int i=0; i<so_sv; i++)
      for ( int j=0; j<so_mon; j++)
          printf("Điểm môn %d của sv %d
  là:%d",j,i,result[i][j];
}
```

Ví dụ

- Phương án 3 : Sử dụng mảng một chiều, các phần tử có kiểu cấu trúc:

```
typedef struct DiemSV{
    int mon1, mon2, mon3, mon4;
};
DiemSV result [3];
```

  - các phần tử sẽ được lưu trữ như sau:

Result[0].mon1=9	Result[0].mon2=9	Result[0].mon3=5	Result[0].mon4=2
Result[1].mon1=5	Result[1].mon2=0	Result[1].mon3=9	Result[1].mon4=4
Result[2].mon1=6	Result[2].mon2=3	Result[2].mon3=7	Result[2].mon4=4

  - Truy xuất điểm số môn j của sinh viên i phải sử dụng một công thức xác định chỉ số tương ứng trong mảng result:

```
result[(i*số cột) + j]
```

Ví dụ


```
void XuatDiem()//Xuất điểm số của tất cả sv
{
   const int so_sv = 3;
   for (int i=0; i<so_sv; i++)
   {
        printf("Điểm môn 1 của sv %d là: %d”,i,
        result[i].mon1);
        printf("Điểm môn 2 của sv %d là: %d”,i,
        result[i].mon2);
        printf("Điểm môn 3 của sv %d là: %d”,i,
        result[i].mon3);
        printf("Điểm môn 4 của sv %d là: %d”,i,
        result[i].mon4);
   }
}
```

Các tiêu chuẩn đánh giá CTDL

Một cấu trúc dữ liệu tốt phải thỏa mãn các tiêu chuẩn sau:

- **CTDL phải phản ánh đúng thực tế**: Đây là yếu tố quyết định tính đúng đắn của bài toán. Cần xem xét trạng thái biến đổi của dữ liệu nhằm chọn CTDL phù hợp
- **CTDL phải phù hợp với các thao tác trên đó**: Giúp tăng tính hiệu quả của bài toán. Mỗi CTDL có các thao tác tương ứng. Nên chọn CTDL sao cho các thao tác được xử lý đơn giản và tự nhiên. Việc phát triển các thuật toán đơn giản, tự nhiên hơn => chương trình đạt hiệu quả cao hơn về tốc độ xử lý.
- **CTDL phải tiết kiệm tài nguyên hệ thống:** CTDL chỉ nên sử dụng tài nguyên hệ thống vừa đủ để đảm nhiệm được chức năng của nó. Loại tài nguyên cần quan tâm là : CPU và bộ nhớ.

Các kiểu dữ liệu

- Máy tính chỉ có thể lưu trữ dữ liệu ở dạng nhị phân.
- Nếu muốn phản ánh được dữ liệu đa dạng, thì cần phải xây dựng những *phép ánh xạ*, những *qui tắc tổ chức phức tạp* che lên tầng dữ liệu nhị phân thô sơ.
- Nhằm đưa ra những khái niệm logic về hình thức lưu trữ khác nhau đượoc gọi là kiểu dữ liệu.
  - Các kiểu dữ liệu cơ sở
  - Các kiểu dữ liệu có cấu trúc
  - Kiểu dữ liệu con trỏ
  - Kiểu tập tin

Các kiểu dữ liệu

Định nghĩa kiểu dữ liệu

Kiểu dữ liệu **T** được xác định bởi một bộ **<V,O>**

- V : tập các giá trị hợp lệ mà một đối tượng kiểu T có thể lưu trữ
- O : tập các thao tác xử lý có thể thi hành trên đối tượng kiểu T.
- Dữ liệu lưu trữ chiếm số bytes trong bộ nhớ gọi là kích thước của kiểu dữ liệu

Các kiểu dữ liệu

Định nghĩa kiểu dữ liệu

- Ví du:

Giả sử có kiểu dữ liệu **Alphabet**= <V,O> với

V = { a-z,A-Z}

O = { lấy mã ASCII của ký tự, biến đổi ký tự thường thành ký tự hoa …}

Giả sử có kiểu dữ liệu **số nguyên** = <V,O> với

V = { -32768..32767}

O = { +, -, *, /, %}

Các kiểu dữ liệu

Định nghĩa kiểu dữ liệu

- Ví du:

Giả sử có kiểu dữ liệu **Contact**= <V,O> với

V = Họ và tên, địa chỉ, số điện thoại}

O = { Gán giá trị, so sánh}

Giả sử có kiểu dữ liệu **ContactList** = <V,O> với

V = { danh sách các đối tượng có kiểu **Contact**}

O = { Thêm, xóa, sửa, so sánh, tìm kiếm, sắp xếp}

Các kiểu dữ liệu

Các thuộc tính cần quan tâm của một kiểu dữ liệu

- Tên kiểu dữ liệu
- Miền giá trị của dữ liệu
- Kích thước dữ liệu
- Tập các toán tử, thao tác xử lý tác động lên đối tượng có kiểu dữ liệu

Các kiểu dữ liệu

Các kiểu dữ liệu cơ bản

- Kiểu không rời rạc: số nguyên, ký tự, logic , liệt kê, miền con
- Kiểu có thứ tự rời rạc: số thực

VD: Kiểu dữ liệu trong C

Tên kiểu	Kích thước	Miền giá trị
char	1 byte	-128 -> 127
unsign char	1 byte	0 ->255
int	2 byte	-32768 -> 32767
unsign int	2 byte	0 - -> 65335
long	4 byte	-2^{32} -> 2^{31}
unsign long	4 byte	0 -> 2^{32}
float	4 byte	3.4E-38....3.4E38
double	8 byte	1.7E-308....1.7E308
long double	10 byte	3.4E-4932....1.1E4932

Các kiểu dữ liệu

Các kiểu dữ liệu có cấu trúc

- Kiểu chuỗi ký tự
- Kiểu mảng
- Kiểu mẫu tin (cấu trúc)
- Kiểu union

Các kiểu dữ liệu

Các kiểu dữ liệu có cấu trúc

Kiểu chuỗi ký tự: là kiểu dữ liệu có cấu trúc đơn giản nhất và thường các ngôn ngữ lập trình đều định nghĩa nó như một kiểu cơ bản.

Trong C các hàm xử lý chuỗi được đặt trong thư viện string.lib.

VD: char S[10] ;// chuỗi ký tự S có chiều dài tối đa là 10 (kể cả ký tự kết thúc)

char S[] = ”ABCDEF” ;

char *S = “ABCDEF”;

Các kiểu dữ liệu

Các kiểu dữ liệu có cấu trúc (tt)

Kiểu mảng: là kiểu dữ liệu trong đó mỗi phần tử của nó là một tập hợp có thứ tự các giá trị có cùng cấu trúc được lưu trữ liên tiếp nhau trong bộ nhớ.

Mảng một chiều :

<Kiểu dữ liệu> <Tên biến> [<Số phần tử>];

Mảng nhiều chiều :

<Kiểu dữ liệu> <Tên biến> [<Số phần tử 1>] [<Số phần tử 2>]….;

Các kiểu dữ liệu

Các kiểu dữ liệu có cấu trúc (tt)

Kiểu mẫu tin: Kiểu mẫu tin cũng tương tự như mảng nhưng mỗi phần tử của nó là tập hợp các giá trị có thể khác cấu trúc.

Kiểu mẫu tin thường được dùng để mô tả những đối tượng có cấu trúc phức tạp.

Ví dụ :
```
struct PERSON{
    char Hoten[];
    int NamSinh;
    char NoiSinh[];
    char GioiTinh;      // 0:Nữ, 1: Nam
    char DiaChi[];
    };
```

Các kiểu dữ liệu

Kiểu dữ liệu con trỏ

Cho trước kiểu T = <V,O>. Kiểu con trỏ ký hiệu $\mathbf{T_p}$ chỉ đến các phần tử có kiểu **T** được định nghĩa như sau:

$T_p = <V_p, O_p>$ Trong đó:

T_p = {{các địa chỉ có thể lưu trữ những đối tượng kiểu T}, NULL}

O_p = {các thao tác định địa chỉ của một đối tượng kiểu T khi biết con trỏ chỉ đến đối tượng đó}

Kiểu con trỏ là kiểu dữ liệu cơ sở dùng lưu địa chỉ của một đối tượng dữ liệu khác.

Biến thuộc kiểu con trỏ T_p là biến mà giá trị của nó là địa chỉ của một vùng nhớ ứng với một biến kiểu T, hoặc là giá trị NULL

Các kiểu dữ liệu

Kiểu dữ liệu con trỏ (tt)

- Kích thước biến con trỏ tùy thuộc vào quy ước số byte trong từng mô hình bộ nhớ và từng ngôn ngữ lập trình cụ thể.
- Biến con trỏ trong C++ có kích thước 2 hoặc 4 bytes tùy vào con trỏ NEAR hay FAR
- Cú pháp định nghĩa dữ liệu kiểu con trỏ
  typedef <kiểu cơ sở> * <kiểu con trỏ>;

Các thao tác cơ bản trên kiểu con trỏ:

- Khi một biến con trỏ 'p' lưu trữ địa chỉ của đối tượng x ta nói "p trỏ đến x"
- Gán địa chỉ của một vùng nhớ con trỏ p:
  p = <địa chỉ>;
  p = <địa chỉ> + <giá trị nguyên>
- Truy xuất (xem) nội dung của đối tượng p trỏ đến (*p)

Các kiểu dữ liệu

Kiểu dữ liệu tập tin

- Tập tin là kiểu dữ liệu đặc biệt, kích thước tối đa của tập tin phụ thuộc không gian đĩa
- Việc đọc, ghi dữ liệu trên tập tin là mất thời gian, không an toàn dữ liệu
- Thông thường chuyển dữ liệu trong tập tin (một phần hay toàn bộ) vào bộ nhớ trong để xử lý.

Các kiểu dữ liệu

Ví dụ: Để mô tả thông tin của một đối tượng sinh viên, cần quan tâm đến các thông tin sau:

- Mã sinh viên: chuỗi 12 ký tự
- Tên sinh viên: chuỗi 30 ký tự
- Ngày sinh: kiểu ngày tháng
- Nơi sinh: chuỗi 255 ký tự
- Điểm thi: số thực

Các kiểu dữ liệu

Các kiểu dữ liệu cơ sở cho phép mô tả một số thông tin như :

- `float Diemthi;`

Các thông tin khác đòi hỏi phải sử dụng các kiểu có cấu trúc như :

- `char masv[12];`
- `char tensv[30];`
- `char noisinh[255];`

Các kiểu dữ liệu

Để thể hiện thông tin về ngày tháng năm sinh cần phải xây dựng một kiểu bản ghi,

```
typedef struct tagDate
{
   int ngay;
   int thang;
   int thang;
} Date;
```

Các kiểu dữ liệu

- kiểu dữ liệu thể hiện thông tin về một sinh viên :

```
typedef struct tagSinhVien
    {
     char masv[12];
     char tensv[30];
     char noisinh[255];
     Date ngaysinh;
     float Diem thi;
    } SinhVien;
```

Nhắc lại các kiểu dữ liệu C++

- Kiểu cơ sở: Số nguyên, số thực và kiểu logic
- Kiểu mảng, chuỗi
- Kiểu có cấu trúc

Kiểu số nguyên

Stt	Tên kiểu	Ghi chú	Kích thước
1	char	Ký tự	1 byte
		Số nguyên	1 byte
2	unsigned char	Số nguyên dương	1 byte
3	short	Số nguyên	2 bytes
4	unsigned short	Số nguyên dương	2 bytes
5	int	Số nguyên	4 bytes
6	unsigned int	Số nguyên dương	4 bytes
7	long	Số nguyên	4 bytes
8	unsigned long	Số nguyên dương	4 bytes

Kiểu số thực

Stt	Tên kiểu	Ghi chú	Kích thước
1	float		4 bytes
2	double		8 bytes
3	long double		8 bytes

Kiểu luận lý

Stt	Tên kiểu	Ghi chú	Kích thước
1	bool	Gồm 2 giá trị: **true** hoặc **false**	

Kiểu mảng 1 chiều

Giá trị	5	7	10	…	3	11	2
Vị trí	*0*	*1*	*2*	…		*n-2*	*n-1*

- **Khai báo**

<KDL> <Tên mảng> [<Số phần tử max>];

VD: int a[100];

- **Gán giá trị ban đầu**

VD1: int a[100] = {0};

VD2: int a[5] = {3, 6, 2, 10, 17};

hoặc: int a[] = {3, 6, 2, 10, 17};

Kiểu mảng 1 chiều

Phát sinh ngẫu nhiên <time.h>

- Khởi tạo phát sinh ngẫu nhiên

```
srand((unsigned int)time(NULL));
```

- Phát sinh giá trị ngẫu nhiên

```
int x = rand()%k;
```

k: Số nguyên dương

x ∈ [0..k-1]

VD: Phát sinh 1 số nguyên có giá trị từ 0 đến 50

```
srand((unsigned int)time(NULL));
int x = rand()%51;
```

Bài tập 1

1. Cài đặt hàm nhập mảng số nguyên, n phần tử
2. Cài đặt hàm phát sinh n phần tử ngẫu nhiên cho mảng số nguyên
3. Cài đặt hàm phát sinh n phần tử ngẫu nhiên có giá trị tăng dần cho mảng số nguyên

Kiểu chuỗi ký tự

- **Khai báo**

char <Tên chuỗi> [<Số ký tự max>] ;

VD: char hoten[50];

- **Xem lại các hàm**

- gets()
- strcpy, strcat, strlen
- strcmp, stricmp
- strchr, strstr
- …

Tên hàm	Công dụng
strcmp(s1,s2)	So sánh 2 chuỗi s1 và s2
strcpy(dest,src)	Sao chép src vào dest
strstr(s1,s2)	Kiểm tra s2 có nằm trong chuỗi s1 khong?
itoa(n,s,c)	Đổi số nguyên n thành chuỗi ở dạng cơ số c
atoi(sn) ,atof(sf), atol(sl)	Đổi 1 chuỗi thành số
gets(st)	Nhập một chuỗi
puts(st)	Xuất một chuỗi

Kiểu mảng – Khai báo kiểu con trỏ

- **Mảng 1 chiều**

<KDL> *<Tên mảng>;

VD: int *a;

- **Chuỗi ký tự**

char *<Tên chuỗi>;

VD: char *hoten;

Kiểu mảng – Khai báo kiểu con trỏ

Lưu ý khi sử dụng phải cấp phát biến con trỏ bằng lệnh **new**, hủy bằng lệnh **delete**

VD:

```
int *a;
int n = 10;
a = new int[n];
…..
delete a;
```

Bài tập 2

Viết lại các hàm trong Bài tập 1 dùng khai báo kiểu con trỏ

Kiểu dữ liệu có cấu trúc


```
struct tên_struct
{
        khai báo các thuộc tính;
};
typedef struct tên_struct tên_kiểu;
```

Ví dụ kiểu dữ liệu có cấu trúc


```
struct ttDate
{
    char thu[9];
    unsigned char ngay;
    unsigned char  thang;
    int nam;
};
typedef struct ttDate DATE;
```

Truy cập thành phần có cấu trúc

- **Biến cấu trúc kiểu tĩnh**

<tên biến>.thành phần cấu trúc

VD:

```
DATE d;

d.nam = 2012;
```

Bài tập 3

Viết hàm nhập và hàm xuất thông tin của một sinh viên gồm các thông tin:

- Mã số
- Họ tên
- Điểm trung bình

Truy cập thành phần có cấu trúc

- **Biến cấu trúc kiểu con trỏ**

<tên biến> → thành phần cấu trúc

VD:

DATE *d;

d → nam = 2012;

Bài tập 4

Viết lại các hàm trong Bài tập 3 sử dụng khai báo biến kiểu con trỏ cấu trúc

Đánh giá độ phức tạp của giải thuật

- Phân tích thuật toán
  - Tính đúng
  - Tính đơn giản
  - Không gian
  - Thời gian chạy của thuật toán

Đánh giá độ phức tạp giải thuật

Đánh giá độ phức tạp của giải thuật

- Là công việc ước lượng thời gian thực hiện của thuật toán để so sánh tương đối các thuật toán với nhau
- Trong thực tế, thời gian thực hiện còn phụ thuộc cấu hình máy, dữ liệu đưa vào, …
- Để ước lượng thời gian thực hiện thuật toán xem xét 2 trường hợp
  - Trường hợp tốt nhất: T_{min}
  - Trường hợp xấu nhất: T_{max}
- Với T_{min} và T_{max} → thời gian thực hiện trung bình của thuật toán T_{avg}

Đánh giá độ phức tạp của giải thuật

- Đánh giá thời gian chạy của thuật toán
  - Là số các thao tác cơ bản được thực hiện; phép tính số học, logic cơ bản hoặc một câu lệnh "đơn" (như lệnh gán, tăng, giảm biến…)
  - Các câu lệnh sau được xem như được thực hiện với cùng một thời gian
    - `y = m * x + b`
    - `c = 5 / 9 * (t - 32 )`
    - `z = f(x) + g(y)`

Đánh giá độ phức tạp của giải thuật

- Thời gian chạy của thuật toán
  - Đánh giá như thế nào
    - Thực nghiệm
    - Xấp xỉ (pp hình thức)

Đánh giá độ phức tạp giải thuật

Phương pháp thực nghiệm:

- Cài đặt thuật toán, chọn các bộ dữ liệu thử, thống kê các thông số nhận được
- Ưu điểm: Dễ thực hiện
- Một số nhược điểm:
  - Phụ thuộc vào ngôn ngữ lập trình dung để cài đặt
  - Phụ thuộc vào trình độ của người cài đặt
  - Phụ thuộc vào việc chọn ra bộ dữ liệu thử đặc trưng cho tất cả các trường hợp: : khó khăn và tốn nhiều chi phí
  - Phụ thuộc vào phần cứng khi thực thi thuật toán

Độ phức tạp của thuật toán

Xấp xỉ tiệm cận

- Đánh giá theo hướng xấp xỉ tiệm cận thông qua khái niệm toán học Big-O (O-lớn) hay O()
- Ưu điểm: Đây là phương pháp đánh giá ít phụ thuộc vào môi trường cũng như phần cứng.
- Nhược điểm: Phức tạp
- Thời gian chạy của 1 thuật toán sẽ phụ thuộc vào kích thước của dữ liệu đầu vào (thường gọi là N).
- Trong các dữ liệu đầu vào cùng 1 kích thước, thời gian chạy của thuật toán cũng thay đổi và đánh giá theo 3 trường hợp:
  - Trường hợp tốt nhất
  - Trường hợp xấu nhất
  - Trường hợp trung bình

Độ phức tạp của thuật toán

- Gọi n là số lượng dữ liệu nhập vào và t là thời gian thực hiện 1 lệnh
- Tình f(n) là tổng số lần thực hiện các lệnh

  ➔ Thời gian thực hiện thuật toán là T=f(n)*t
- Cho n → vô cực thì f(n) → g(n)
- O(g(n)) gọi là độ phức tạp của thuật toán

Độ phức tạp của thuật toán

- Độ phức tạp tính toán của giải thuật: O(f(n))
- Quy tắc xác định độ phức tạp:
  - Quy tắc bỏ hằng số:

    T(n) = O(c.f(n)) = O(f(n)) với c là một hằng số dương
  - Quy tắc lấy max:

    T(n) = O(f(n)+ g(n)) = O(max(f(n), g(n)))
  - Quy tắc cộng:

    T1(n) = O(f(n)) , T2(n) = O(g(n)) ➔ T1(n) + T2(n) = O(f(n) + g(n))
  - Quy tắc nhân:

Đoạn chương trình có thời gian thực hiện T(n)=O(f(n)). Nếu thực hiện k(n) lần đoạn chương trình với k(n) = O(g(n)) thì độ phức tạp sẽ là O(g(n).f(n))

Đánh giá độ phức tạp của giải thuật

- Một số kết quả Big-O quan trọng:
  - Hàm đa thức:
    - $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$
    - Khi đó f(x) là $O(x^n)$
  - Hàm giai thừa:
    - $f(n) = n!$ là $O(n^n)$
  - Logarit của hàm giai thừa:
    - $f(n) = \log n!$ là $O(n \log n)$
  - Hàm điều hòa
    - $H(n) = 1 + 1/2 + 1/3 + .. + 1/n$ là $O(\log n)$

Thời gian chạy của các lệnh

- Lệnh gán

  X = <biểu thức>

  Thời gian chạy của lệnh gán bằng thời gian thực hiện biểu thức ~ O(1)

- Lệnh lựa chon

  if (điều kiện) → T0(n)

  lệnh 1 → T1(n)

  else

  lệnh 2 → T2(n)

  Thời gian: T0(n) + max (T1(n), T2(n))

Thời gian chạy của các lệnh

- Lệnh lặp: for, while, do-while

$$\sum_{i=1}^{X(n)} (T_0(n) + T_i(n))$$

X(n): Số vòng lặp

T0(n): Điều kiện lặp

Ti(n): Thời gian thực hiện vòng lặp thứ i

Đánh giá độ phức tạp của giải thuật

- Sự phân lớp các độ phức tạp của thuật toán

Độ phức tạp	Thuật ngữ/tên phân lớp
O(1)	Độ phức tạp hằng số
O(logn)	Độ phức tạp logarit
O(n)	Độ phức tạp tuyến tính
O(nlogn)	Độ phức tạp nlogn
$O(n^a)$	Độ phức tạp đa thức
$O(a^n)$, a > 1	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

Đánh giá độ phức tạp của giải thuật

- Ví dụ, xét hàm sau:
  - Lệnh *printf* ngoài vòng lặp có độ phức tạp hằng O(1) , vì không phụ thuộc vào N
  - Số lệnh *printf* trong vòng lặp bằng với kích thước mảng, do đó vòng lặp có độ phức tạp O(N)
  - Tổng cộng: Hàm f có độ phức tạp: O(1) + O(N)

→ Độ phức tạp: O(N)

```
void f(int a[], int n)
{
   printf("n = %d", n) ;
   for (int i = 0; i < n; i++ )
      printf("%d  ",a[i]);
}
```

Đánh giá độ phức tạp của giải thuật

- Ví dụ, đánh giá giải thuật tìm số lớn nhất của n số với giải thuật sau:
  - B0: Nhập n và dãy n số nguyên a[0], a[1],….,a[n]
  - B1: max=a[0[; i=1;
  - B2: Trong khi j<n lặp lại các bước sau:
    - B2.1: Nếu a[i]>max thì max=a[i];
    - B2.2: i=i+1
  - B3: In max

  → Đánh giá trường hợp dữ liệu xấu nhất (dãy tăng)

Tính các bước?

Đánh giá độ phức tạp của giải thuật

- Ví dụ, đánh giá số lần thực hiện câu lện x=x+1 trong đoạn chương trình sau:

```
for (i= 1;i<=n;i++)
     for (j= 1;j<=i;j++)
       x=x+i;
```

Đánh giá độ phức tạp của giải thuật

- **PHÉP TOÁN TÍCH CỰC (BEST PROXY)**
  - Trong một thuật toán, ta chú ý đặc biệt đến một phép toán gọi là phép toán tích cực. Đó là phép toán mà số lần thực hiện không ít hơn các phép toán khác
  - Ví dụ 1:

```
s = 0;
for (i=0; i<=n;i++){
        p =1;
        for (j=1;j<=i;j++)
                p=p * x / j;
        s = s+p;
}
```

- Phép toán tích cực là p = p * x / j ➔ Số lần thực hiện phép toán là $0+1+2+\ldots+n = n(n-1)/2 = n^2/2 - n/2$ ➔ Vậy độ phức tạp của thuật toán là $O(n^2/2 - n/2)$ ➔ $= O(n^2/2)$: sử dụng quy tắc lấy max ➔ $=O(n^2)$ sử dụng quy tắc bỏ hằng số

Đánh giá độ phức tạp của giải thuật

- **PHÉP TOÁN TÍCH CỰC (BEST PROXY)**
  - Trong một thuật toán, ta chú ý đặc biệt đến một phép toán gọi là phép toán tích cực. Đó là phép toán mà số lần thực hiện không ít hơn các phép toán khác
  - Ví dụ 2:

```
1. s = 1; p = 1;
2. for (i=1;i<=n;i++) {
3.                 p = p * x / i;
4.                 s = s + p;
5. }
```

Phép toán tích cực là p = p * x / j ➔ Số lần thực hiện phép toán là n
➔ Vậy độ phức tạp của thuật toán là O(n)

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
s=n*(n-1) /2;
```

Hãy xác định độ phức tạp.

=> Độ phức tạp của thuật toán là O(1), nghĩa là thời gian tính toán không phụ thuộc vào n

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
s=0;
for (i= 1;i<=n;i++)
  s=s+i;
```

Hãy xác định độ phức tạp.

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
1. for (i= 1;i<=n;i++)
2.     for (j= 1;j<=n;j++)
3.                 //Lệnh
```

Hãy xác định độ phức tạp.

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
1. for (i= 1;i<=n;i++)
2.     for (j= 1;j<=m;j++)
3.                 //Lệnh
```

Hãy xác định độ phức tạp.

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
1. for (i= 1;i<=n;i++)
2.                  //lệnh1
3. for (j= 1;j<=m;j++)
4.                  //lệnh 2
```

Hãy xác định độ phức tạp.

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
1. for (i= 1;i<=n;i++)
2.    for (j= 1;j<=m;j++) {
3.        for (k= 1;k<=x;k++)
4.            //lệnh
5.         for (h= 1;h<=y;h++)
6.            //lệnh
7. }
```

Hãy xác định độ phức tạp.

Đánh giá độ phức tạp của giải thuật

- Cho đoạn mã dưới đây:

```
1. for (i= 1;i<=n;i++)
2.   for (u= 1;u<= m;u++)
3.     for (v= 1;v<=n;v++)
4.       //lệnh
5. for (j= 1;j<=x;j++)
6.   for (k= 1;k<=z;k++)
7.     //lệnh
```

Hãy xác định độ phức tạp.

BÀI TẬP ĐỘ PHỨC TẠP

Bài 1: Tính thời gian thực thi của f(n) trong đoạn code sau:

```
void f(int n) {
      for (int i = 0; i < n; ++i)
            g();
}
```

Bài 2: Tính thời gian thực thi của f(n) trong đoạn code sau:

```
void f(int n) {
   if (n > 0) {
      for (int j=0; j<n; j++)
            cout << j << endl;
      f(n/2);
   }
}
```

BÀI TẬP ĐỘ PHỨC TẠP

Bài 3: Tính thời gian thực thi của f(n) trong đoạn code sau:

```
void f(int n) {
  if (n > 0) {
     for (int j=0; j<n; j++) cout << j << endl;
     f(n-1);
  }
}
```

Bài 4: Tính thời gian thực thi của f(n) trong đoạn code sau:

```
void f(int n, int x) {
  for (i=1; i<= n; i++)
    for (j=1; j<=n; j++)
       --x;
}
```

BÀI TẬP ÔN TẬP 1

- Các thao tác trên cấu trúc dữ liệu mảng một chiều:
  - Nhập/xuất mảng (hoặc phát sinh ngẫu nhiên - random)
  - Tìm vị trí của giá trị X trong mảng
  - Sắp xếp mảng có thứ tự
  - Ghi dữ liệu của mảng ra file text.
  - Nộp bài: MASV_HOVATEN_BAI1.CPP trên hộ thống LMS

BÀI TẬP ÔN TẬP 2

Thiết kế CTDL và thao tác xử lý trên CTDL nhằm quản lý thông tin thí sinh đăng ký dự thi đại học:

Gồm các thông tin sau:

1. Số báo danh
2. Trường dự thi
3. Khối thi
4. Điểm môn 1
5. Điểm môn 2
6. Điểm môn 3
7. Điểm cộng

Gồm các thao tác xử lý sau:

1. Nhập thôn g tin từng thí sinh
2. Tính tổng điểm của tin từng thí sinh
3. Sắp xếp danh sách thí sinh theo SBD, theo tổng điểm
4. Tìm kiếm thí sinh với thông tin cho trước
5. In danh sách K thí sinh

Nộp bài: MASV_HOVATEN_BAI2.CPP trên hộ thống LMS