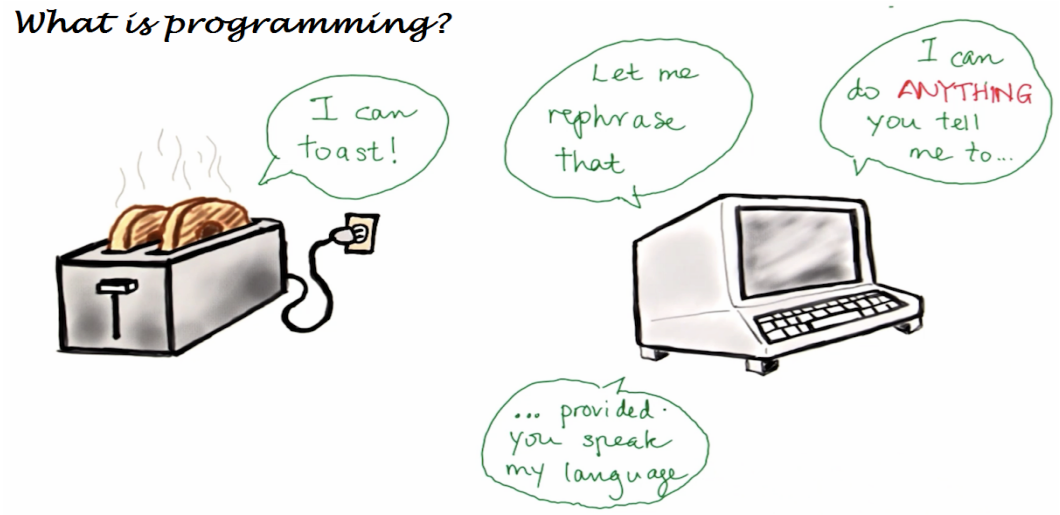


ÔN TẬP HÀM VÀ MỘT SỐ KIỂU DỮ LIỆU

Bộ môn Công nghệ phần mềm

Khoa CNTT&TT – ĐHCT

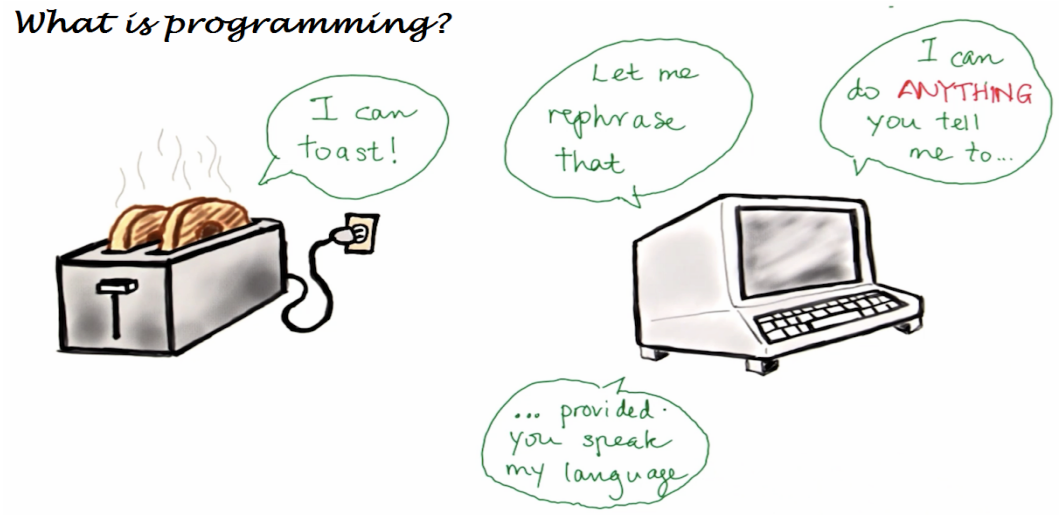
Nội dung

Lập trình căn bản

- Chương trình con/Hàm
- Kiểu mảng
- Kiểu cấu trúc
- Kiểu con trỏ
  - Con trỏ đến cấu trúc

Cấu trúc dữ liệu

- Các phép toán của các CTDL
- Danh sách, ngăn xếp, hàng đợi
- Các cấu trúc dữ liệu
- Cây, các phép toán của các CTDL



Hàm

Khoa CNTT&TT – ĐHCT

Tại sao ta cần dùng hàm?

- Sử dụng chương trình con (hàm):
  - Tránh rườm rà và mất thời gian khi viết lặp đi lặp lại nhiều lần những đoạn chương trình giống nhau.
  - Dễ dàng kiểm tra tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu đính trong chương trình chính sẽ thuận lợi hơn.

Hàm

- Định nghĩa hàm
- Sử dụng/Gọi hàm
- Các phương pháp truyền tham số
- Đặc điểm của hàm đệ quy

Trong CT177: cài đặt các phép toán của các cấu trúc dữ liệu

Hàm

- Định nghĩa hàm

```
<kiểu kết quả>  Tên hàm (
                    [<kiểu t số> <tham số>]
                    [,<kiểu t số><tham số>][…])
{
      [Khai báo biến cục bộ]
      [Các câu lệnh thực hiện hàm]
      [return [<Biểu thức>];]
}
```

Tham số hình thức

- Sử dụng hàm/ Gọi hàm

Tham số thực tế

```
<Tên hàm>([Danh sách các tham số])
```

Hàm

- Định nghĩa hàm

```
<kiểu kết quả>  Tên hàm (
      void                [<kiểu t số> <tham số>]
                          [,<kiểu t số><tham số>][…])
{
      [Khai báo biến cục bộ]
      [Các câu lệnh thực hiện hàm]
      [return [<Biểu thức>];]
}
```

Tham số hình thức

- Sử dụng hàm/ Gọi hàm

Tham số thực tế

```
<Tên hàm>([Danh sách các tham số])
```

Hàm – Ví dụ

- Viết hàm trả về giá trị lớn nhất giữa 2 số nguyên A, B.

Kiểu kết quả *Tên hàm* *Tham số hình thức*

```
int maximum( int A, int B ){
    int tam;
    if(A>=B) tam=A;
    else tam=B;
    return tam;
}
```

Khai báo biến cục bộ

Các câu lệnh thực hiện hàm

Trả về kết quả của hàm

```
int maximum(int A, int B){
    if(A>=B) return A;
    else return B;
}
```

Hàm – Ví dụ

```
#include <stdio.h>
int maximum(int A, int B){
    if(A>=B) return A;
    else return B;
}
```

Định nghĩa hàm maximum

```
int main(){
 int x, y, max;
 scanf("%d%d", &x, &y);

 // Cách 1
 max = maximum(x, y); Gọi hàm maximum
 printf("The maximum of the integers you entered was %d",max);

 // Cách 2                                   Gọi hàm maximum
 printf("The maximum of the integers you entered was %d", maximum(x,y));

 return 0;
}
```

Hàm

Các phương pháp truyền tham số

- Truyền giá trị:
  - Là phương pháp truyền tham số mà sau đó *hàm được truyền* có được một phiên bản được lưu trữ riêng biệt giá trị của các tham số đó.
  - Khi truyền giá trị, thì giá trị gốc (được truyền) sẽ không bị thay đổi cho dù hàm được truyền có thay đổi các giá trị này đi nữa.
- Truyền bằng con trỏ:
  - Là phương pháp truyền tham số mà nó cho phép *hàm được truyền* tham khảo đến vùng nhớ lưu trữ giá trị gốc của tham số.
  - Nếu ta truyền bằng con trỏ thì giá trị gốc của tham số có thể được thay đổi bởi các mã lệnh bên trong hàm.

Hàm – Ví dụ

Truyền giá trị

```c
#include <stdio.h>
void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
int main(){
  int A, B;

  scanf("%d%d", &A, &B);

  printf("A=%d, B=%d\n",A,B);
  swap(A, B);
  printf("A=%d, B=%d\n",A,B);

  return 0;
}
```

Truyền bằng con trỏ

```c
#include <stdio.h>
void swap(int *x, int *y){
  int temp = *x;
  *x = *y;
  *y = temp;
}
int main(){
  int A, B;

  scanf("%d%d", &A, &B);

  printf("A=%d, B=%d\n",A,B);
  swap(&A, &B);
  printf("A=%d, B=%d\n",A,B);

  return 0;
}
```

Hàm – Ví dụ

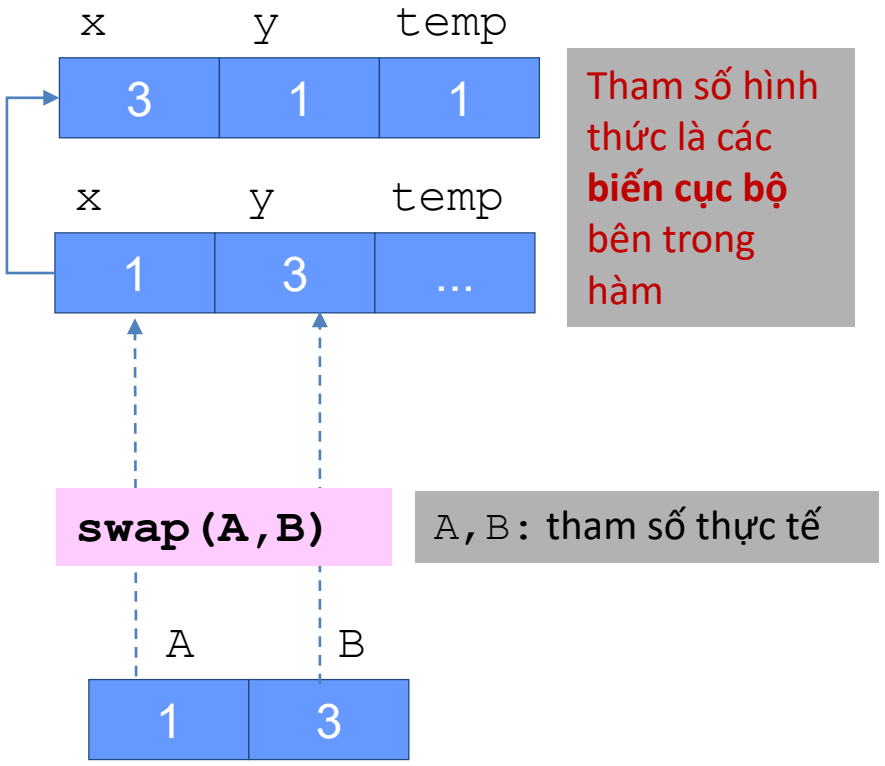
- Xét hàm swap dùng để đổi nội dung 2 biến x, y:

```
#include <stdio.h>
void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}

int main(){
    int A, B;
    scanf("%d%d", &A, &B);
    printf("A=%d, B=%d\n",A,B);
    swap(A, B);
    printf("A=%d, B=%d\n",A,B);
    return 0;
}
```

Định nghĩa hàm

Gọi hàm

Truyền giá trị



Tham số hình thức là các **biến cục bộ** bên trong hàm

Kết quả chạy:

Nhập A, B: 1 3
Trước khi swap: A=1, B=3
Sau khi swap: A=1, B=3

Hàm – Ví dụ

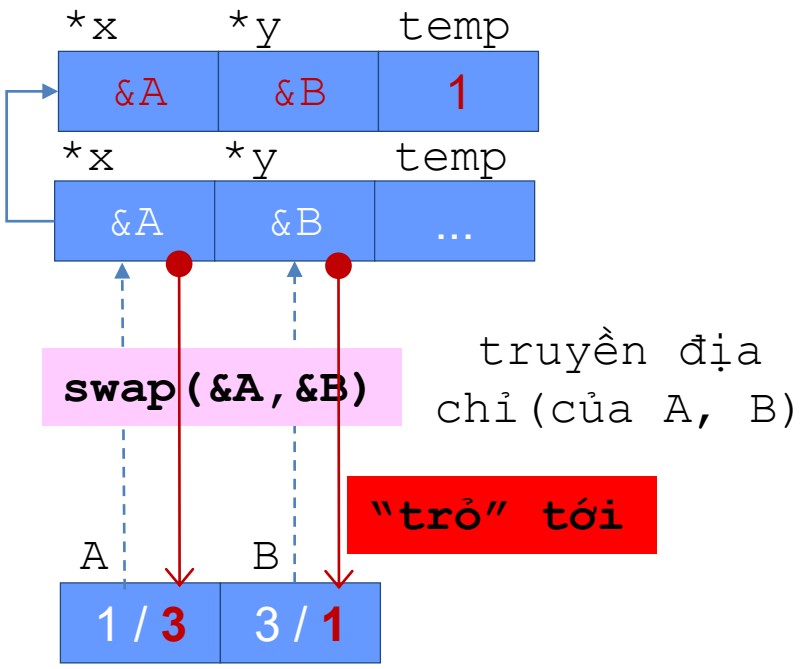
- Xét hàm swap dùng để đổi nội dung 2 biến x, y được truyền bằng con trỏ với dấu * được đặt trước 2 biến x, y:

```
#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main()
{
    int A, B;
    scanf("%d%d", &A, &B);
    printf("A=%d, B=%d\n",A,B);
    swap(&A, &B);
    printf("A=%d, B=%d\n",A,B);
    return 0;
}
```

Định nghĩa hàm

Gọi hàm

Truyền bằng con trỏ



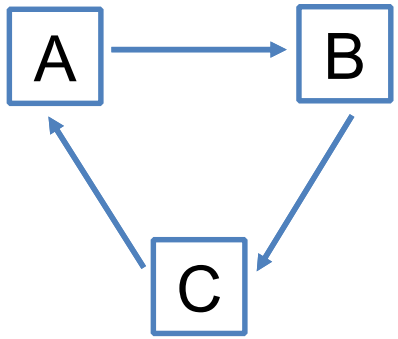
Kết quả chạy:

Nhập A, B:1 3
Trước khi swap: A=1, B=3
Sau khi swap: A=3, B=1

Hàm đệ quy (Recursive function)

- Hàm đệ quy là hàm mà nó gọi chính nó hoặc ở trong một chu trình xoay vòng các lời gọi hàm.
- Nếu một hàm gọi chính nó, ta gọi là hàm đệ quy tức thì (immediate recursion).


```
void A(){
    A();
    return;
}
```




```
void C(){
    A();
    return;
}
void B(){
    C();
    return;
}
void A(){
    B();
    return;
}
```

Hàm đệ quy – Định nghĩa

```
<kiểu kết quả>  Tên hàm ([<kiểu t số> <tham số>]
                          [,<kiểu t số><tham số>][…])
{
   if trường hợp đơn giản
      return giá trị đơn giản;   // trường hợp cơ sở, điều kiện dừng
   else
      gọi hàm với phiên bản đơn giản hơn của bài toán;
}
```

Hàm đệ quy – Ví dụ

```
<kiểu kết quả>  Tên hàm
([<kiểu t số> <tham số>]
[,<kiểu t số><tham số>][…])
{
   if trường hợp đơn giản
      return giá trị đơn giản;
   else
      gọi hàm với phiên bản
      đơn giản hơn của bài toán;
}
```

- Tính n! (n>=0) - đệ quy:

```
n=0   0!                      = 1
n=1   1!                      = 1  * 0!
n=2   2! = 2 * 1              = 2  * 1!
n=3   3! = 3 * 2 * 1          = 3  * 2!
n=4   4! = 4 * 3 * 2 * 1      = 4  * 3!
n=5   5! = 5 * 4 * 3 * 2 * 1  = 5  * 4!
...
```

⇒ n! = 1 với n=0

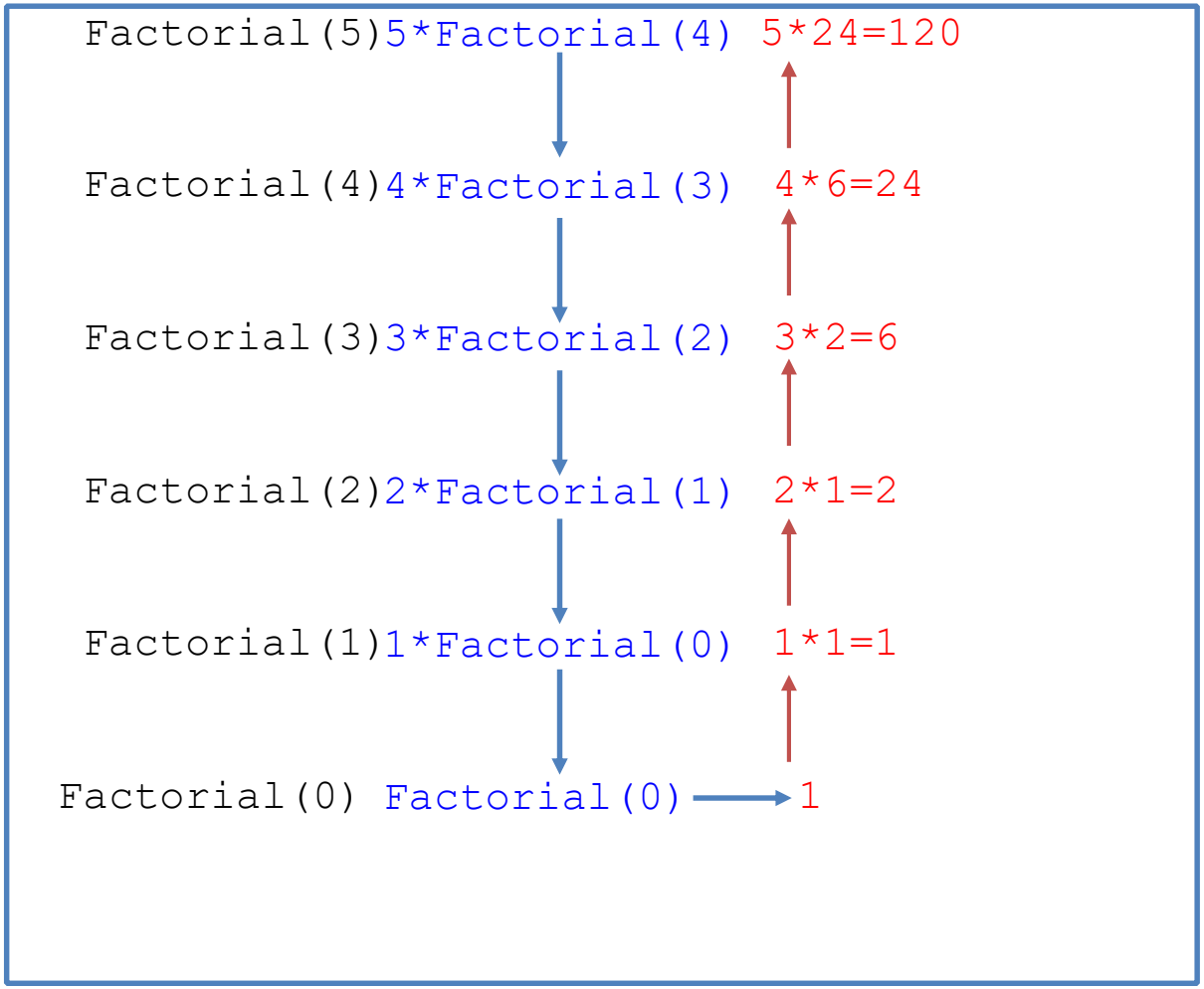
n! = n * (n-1)! với n>0

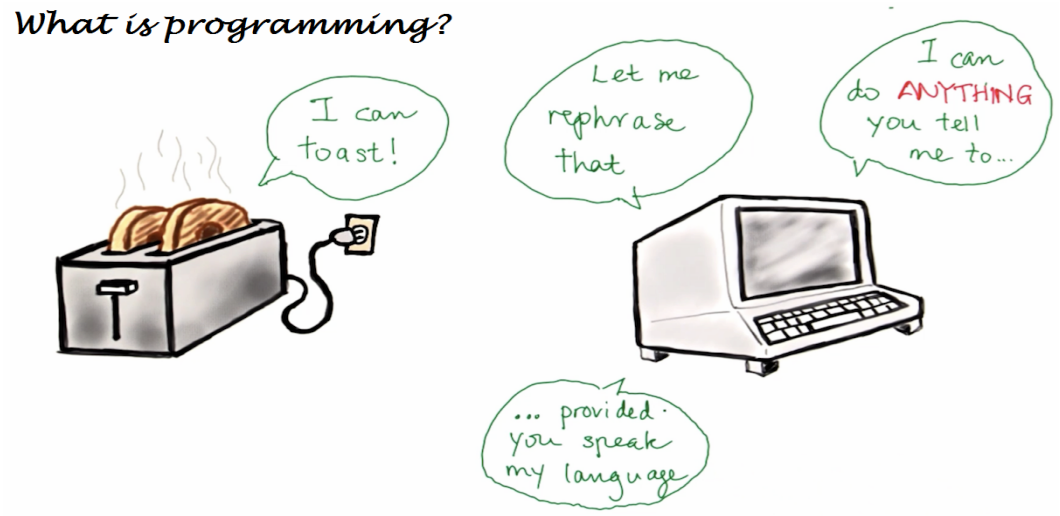
```
int Factorial(int n) {
   if (n == 0) return 1;          //
   return (n * Factorial(n - 1));
}
```

Hàm đệ quy – Ví dụ

- Nếu n=5 thì dãy các lời gọi đệ quy sau sẽ được thực thi:

```
int Factorial(int n) {
    if (n == 0) return 1;
    return (n * Factorial(n - 1));
}
```





Mảng

Khoa CNTT&TT – ĐHCT

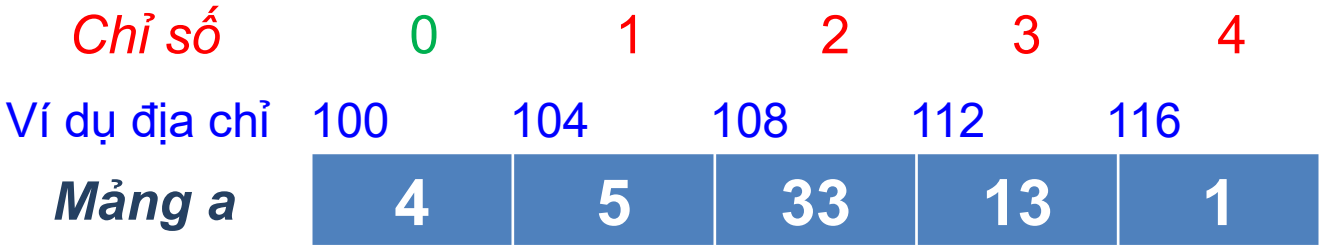
Tại sao ta cần kiểu mảng?

- Trong lập trình đôi khi chúng ta phải đối mặt với việc phải lưu trữ các biến có cùng kiểu dữ liệu như: ***num0***, ***num1***,… và ***num99***
- Mảng giúp nhóm các biến dữ liệu trên thành một biến duy nhất và sẽ được truy cập bằng chỉ số **nums[0]**, ***nums[1]***, … ***nums[99]***.
- Sử dụng mảng giúp người lập trình dễ dàng quản lý dữ liệu hơn thay cho việc khai báo từng giá trị riêng lẻ.

Khái niệm về Mảng

- Mảng là tập hợp các phần tử có cùng một kiểu dữ liệu.
- Các phần tử trong mảng được lưu trữ nối tiếp nhau trong bộ nhớ
- Các phần tử trong mảng có thể được truy cập thông qua các chỉ số

Các phần tử trong mảng




```
a[0] = 4;
a[1] = 5;
a[2] = 33;
a[3] = 13;
a[4] = 1;
```

4, 5, 33, 13, 1 là các giá trị dữ liệu thực sự trong mảng.

0, 1, 2, 3, 4 là các chỉ số để chỉ thứ tự của các phần tử

100, 104, 108, 112, 116 là địa chỉ bắt đầu của các vùng nhớ lưu các giá trị 4, 5, 33, 13, 1 tương ứng

Mảng 1 chiều

- Khai báo
- Truy xuất một phần tử của mảng
- Thao tác trên toàn bộ mảng
- Truyền mảng làm tham số của hàm

Trong CT177: hiểu cách tổ chức dữ liệu của cấu trúc danh sách, ngăn xếp, hàng đợi

Mảng 1 chiều

- Mảng đơn hay mảng một chiều là một *mảng tuyến tính.*
- Chỉ có một ***chỉ số*** để biểu diễn vị trí phần tử, mỗi phần tử trong mảng một chiều *không phải là một mảng khác*.

Chỉ số	0	1	2	3	4
Mảng a	**5**	**4**	**33**	**13**	**1**

Mảng 1 chiều – Khai báo

- ***Cú pháp khai báo***:

```
<data-type> <array_name> [size];
```

- ***Ví dụ khai báo mảng***:

```
int arr[100]; // số phần tử xác định : 100
```

- ***Vừa khai báo vừa gán giá trị:***

```
int iarr[3] = {2, 3, 4} ;
char carr[20] = “c4learn” ;
float farr[3] = {12.5,13.5,14.5} ;
```

- ***Xác định số phần tử của mảng:***

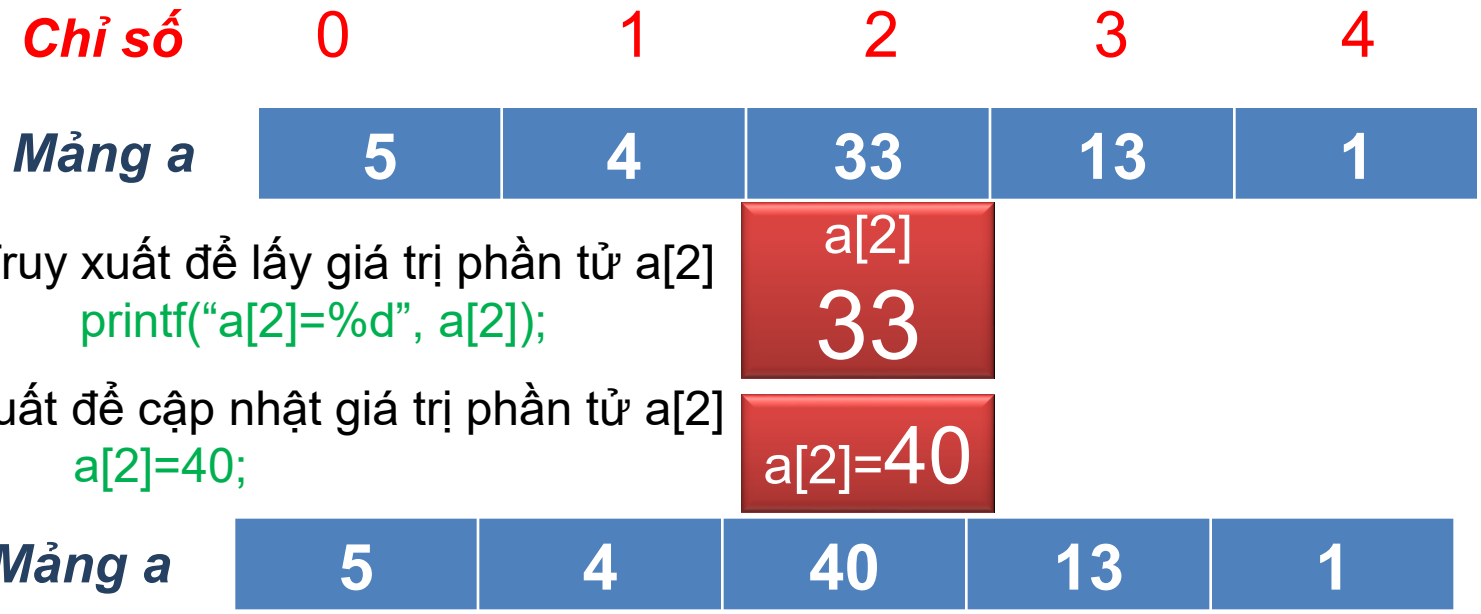
```
size = sizeof(<tên mảng>)/sizeof(<kiểu phần tử>)
```

VD: sizeof(iarr)/sizeof(int) => 3

Mảng 1 chiều – Truy xuất

- Mảng đơn hay mảng một chiều là một *mảng tuyến tính.*
- Truy xuất phần tử của mảng.

```
<array_name>[index]
```



Mảng 1 chiều – Thao tác trên mảng

- Ta thường dùng vòng lặp ***for*** để thao tác các phần tử trong mảng 1 chiều

```c
#include <stdio.h>

int main () {
    int a[10]; /* Khai báo mảng a 10 số nguyên */
    int i,m; /* m<=10 */

    /* Khởi tạo các phần tử mảng a bằng cách nhập từ bàn phím*/
    printf("So phan tu cua mang ");
    scanf("%d",&m);
    for ( i = 0; i < m; i++ ) {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }

    /* Xuất các phần tử của mảng */
    for (i = 0; i < m; i++ ) {
        printf("a[%d] = %d\n", i, a[i] );
    }
    return 0;
}
```

Mảng 1 chiều - Truyền mảng vào tham số của hàm

Trong định nghĩa hàm

- ***Cách 1:*** Truyền như *mảng có kích thước*

```
void myFunction(int param[10]) {
...
}
```

- ***Cách 2:*** Truyền như *mảng không có kích thước xác định*

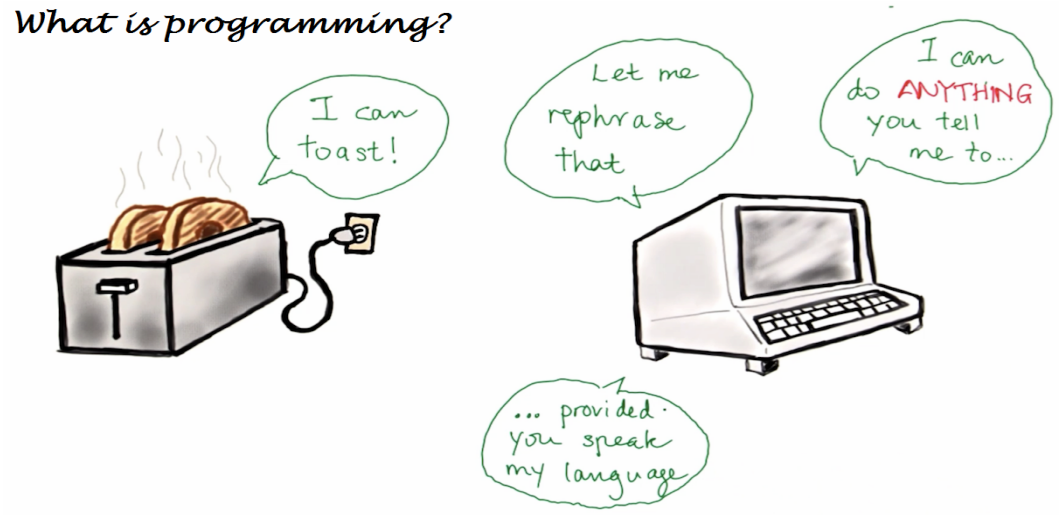
```
void myFunction(int param[]) {
        ...
}
```

Trong gọi hàm

```
// với arr là mảng đã được khai kháo int arr[10];
myFunction(arr); // ⇔ myFunction(&arr[0])
```

Mảng 1 chiều - Truyền mảng vào tham số của hàm

```c
#include <stdio.h>

/* Khai báo hàm */
double getAverage(int arr[], int size);

int main () {
        /* Một mảng 5 phần tử */
        int balance[5] = {1000, 2, 3, 17, 50};
        double avg;

        /* Truyền mảng như một tham số */
        avg = getAverage(balance, 5 ) ;
                                &balance[0]   Truyền địa chỉ
        /* Xuất kết quả trả về */
        printf( "Average value is: %f ", avg );
        return 0;
}
```



Kiểu cấu trúc

Khoa CNTT&TT – ĐHCT

Tại sao ta cần kiểu cấu trúc?

- **1** nhân viên được mô tả bởi tập các thuộc tính:
  - **id**: kiểu chuỗi
  - **name**: kiểu chuỗi
  - **dob**: kiểu chuỗi
  - **gender**: kiểu ký tự
  - **salary**: kiểu số

- Khai báo các biến để lưu trữ 1 nhân viên:
  - `char id[8];`
  - `char name[50];`
  - `char dob[10];`
  - `char gender;`
  - `float salary;`

Nhận xét:

Khó quản lý khi có nhiều biến, chương trình lớn.

Truyền quá nhiều tham số cho hàm

Tìm kiếm, sắp xếp, sao chép, … khó khăn

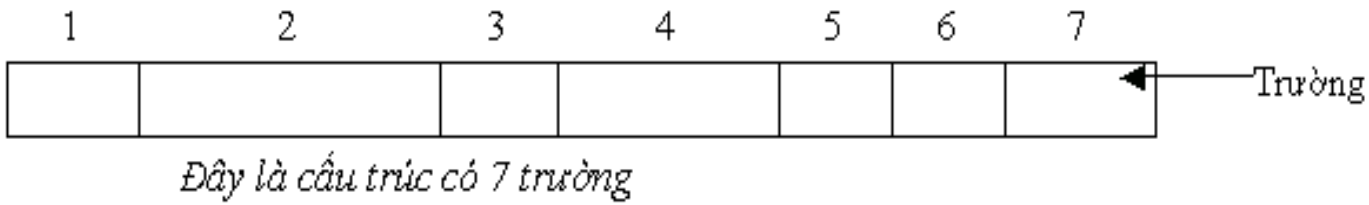
Ý tưởng:

Gom những thông tin của cùng 1 đối tượng thành một kiểu dữ liệu mới => Kiểu Cấu Trúc (Structure)

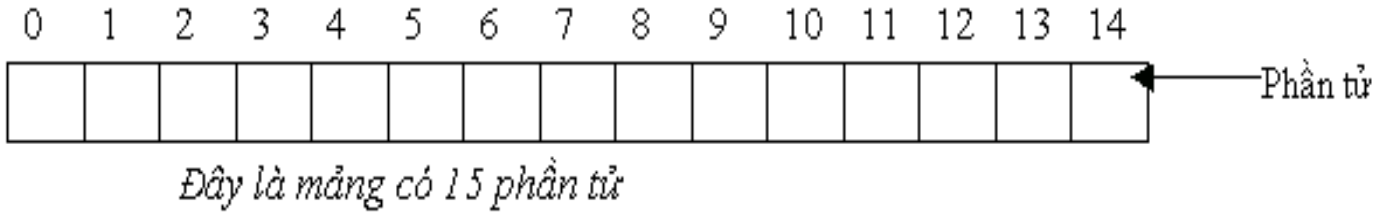
Kiểu cấu trúc

- Là kiểu dữ liệu bao gồm nhiều thành phần có kiểu khác nhau, mỗi thành phần được gọi là một trường (field).
- Nó khác với kiểu mảng gồm các phần tử có cùng kiểu.
- Ví dụ:

1 struct:



1 mảng:



Kiểu cấu trúc

- Định nghĩa kiểu cấu trúc
- Khai báo biến kiểu cấu trúc
- Truy xuất từng trường/thành phần của biến kiểu cấu trúc
- Gán dữ liệu kiểu cấu trúc
- Cấu trúc phức tạp
  - Thành phần của cấu trúc là cấu trúc khác
  - Thành phần của cấu trúc là mảng
  - Mảng cấu trúc

Trong CT177: hiểu cách tổ chức dữ liệu của các cấu trúc dữ liệu cơ bản và cây

Định nghĩa kiểu cấu trúc – Khai báo biến kiểu cấu trúc

- Định nghĩa kiểu cấu trúc:

```
struct <tên kiểu cấu trúc>
{
   <kiểu dữ liệu> <tên thành phần 1>;
   …
   <kiểu dữ liệu> <tên thành phần n>;
};
```

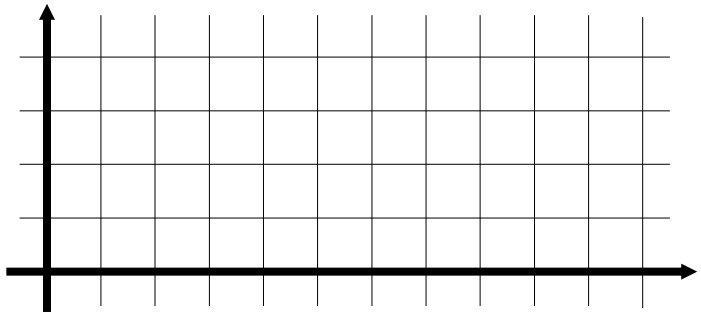
Cách 1

- Khai báo biến kiểu cấu trúc:

```
struct <tên kiểu cấu trúc> <tên biến>;
```

- Ví dụ:

```
struct Diem2D {
   int X;
   int Y;
};
struct Diem2D diem2D1, diem2D2;
```



Định nghĩa kiểu cấu trúc – Khai báo biến kiểu cấu trúc

- Định nghĩa kiểu cấu trúc:

```
typedef struct
{
   <kiểu dữ liệu> <tên thành phần 1>;
   …
   <kiểu dữ liệu> <tên thành phần n>;
} <tên kiểu cấu trúc>;
```

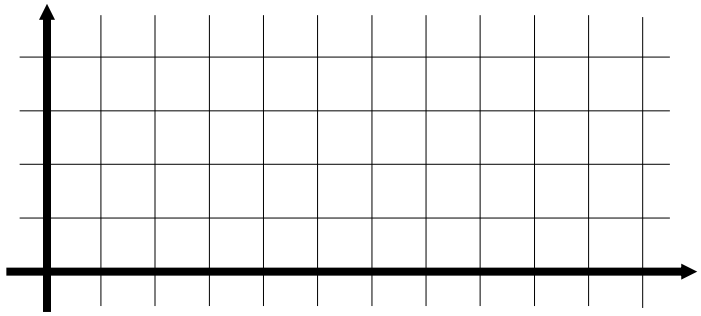
- Khai báo biến kiểu cấu trúc:

```
<tên kiểu cấu trúc> <tên biến>;
```

Cách 2

- Ví dụ:

```
typedef struct {
   int X;
   int Y;
} Diem2D;
Diem2D  diem2D1, diem2D2;
```



Định nghĩa kiểu cấu trúc – Khai báo biến kiểu cấu trúc

- Định nghĩa kiểu cấu trúc:

```
typedef struct
{
<kiểu dữ liệu> <tên thành phần 1>;
…
<kiểu dữ liệu> <tên thành phần n>;
} <tên kiểu cấu trúc>;
```

- Khai báo biến:

```
<tên kiểu cấu trúc> <tên biến>;
```

- Ví dụ:

```
typedef struct {
  int X;
  int Y;
} Diem2D;
Diem2D  diem2D1, diem2D2;
```

- Định nghĩa kiểu cấu trúc:

```
struct <tên kiểu cấu trúc>
{
<kiểu dữ liệu> <tên thành phần 1>;
…
<kiểu dữ liệu> <tên thành phần n>;
};
```

- Khai báo biến:

```
struct <tên kiểu cấu trúc> <tên biến>;
```

- Ví dụ:

```
struct Diem2D {
  int X;
  int Y;
};
struct Diem2D diem2D1, diem2D2;
```

Kết hợp định nghĩa kiểu cấu trúc và khai báo biến kiểu cấu trúc

- Kết hợp định nghĩa kiểu cấu trúc và khai báo biến kiểu cấu trúc:

```
struct <tên kiểu cấu trúc>
{
   <kiểu dữ liệu> <tên thành phần 1>;
   …
   <kiểu dữ liệu> <tên thành phần n>;
}<tên biến 1>, <tên biến 2>;
```

Cách 3

- Ví dụ:

```
struct Diem2D {
   int X;
   int Y;
}diem2D1, diem2D2;
```

Truy xuất từng trường của biến cấu trúc

- Cú pháp:

```
<tên biến kiểu cấu trúc>.<tên thành phần i>;
```

- Ví dụ:

C1

```
struct Diem2D {
   int X;
   int Y;
} ;
struct Diem2D diem2D1;
```

C2

```
typedef struct {
   int X;
   int Y;
} Diem2D;
Diem2D diem2D1;
```

C3

```
struct Diem2D {
   int X;
   int Y;
} diem2D1;
```

```
scanf("%d",&diem2D1.X);
scanf("%d",&diem2D1.Y);
```

Nhập

```
printf("%d",diem2D1.X);
printf("%d",diem2D1.Y);
```

Xuất

Gán dữ liệu kiểu cấu trúc

- Gán toàn bộ cấu trúc

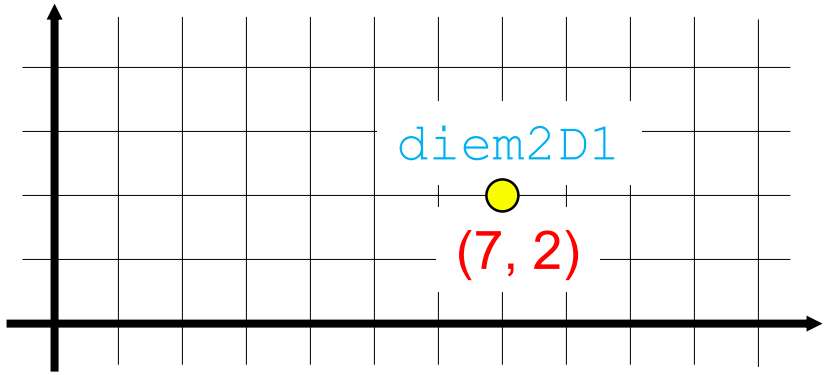
<biến cấu trúc đích > = < biến cấu trúc nguồn >;

- Gán từng trường của biến cấu trúc

<biến cấu trúc đích >.<tên thành phần> = < giá trị >;

- Ví dụ:

```
struct Diem2D {
   int X;
   int Y;
} diem2D1={7,2}, diem2D2;
```




```
diem2D2=diem2D1;
```

```
diem2D2.X=diem2D1.X;
diem2D2.Y=diem2D1.Y;
```

Cấu trúc phức tạp

Thành phần của cấu trúc là cấu trúc khác

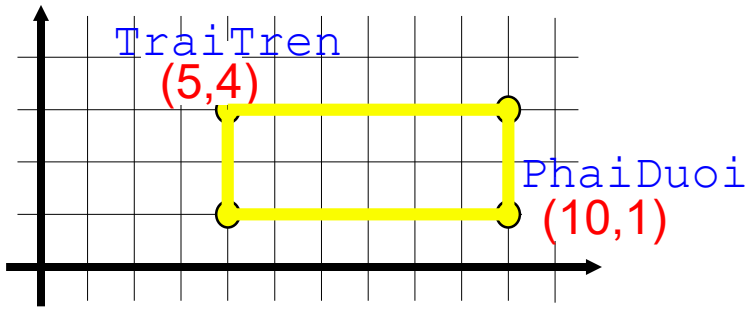
- Định nghĩa, khai báo:

```
typedef struct {
   int X;
   int Y;
} Diem2D;
typedef struct {
   Diem2D TraiTren;
   Diem2D PhaiDuoi;
} HinhCN;
HinhCN  hinhchunhat1;
```

```
struct Diem2D {
   int X;
   int Y;
};
struct HinhCN {
   struct Diem2D TraiTren;
   struct Diem2D PhaiDuoi;
};
struct HinhCN hinhchunhat1;
```

- Truy xuất từng trường (gán dữ liệu):

```
hinhchunhat1.TraiTren.X=5;
hinhchunhat1.TraiTren.Y=4;
```



Cấu trúc phức tạp

Thành phần của cấu trúc là mảng

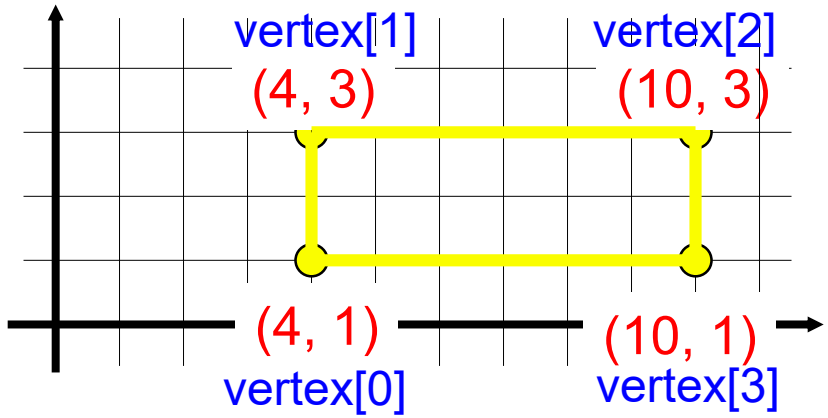
- Định nghĩa, khai báo:

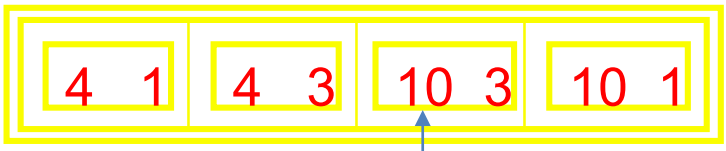
```
typedef struct {
   double X, Y;
} point;
typedef struct {
   point vertex[4];
} square;
square Sq;
```

```
struct point{
   double X, Y;
};
struct square{
   struct point vertex[4];
};
struct square Sq;
```

- Truy xuất từng trường (gán dữ liệu):

```
Sq.vertex[1].X=4;
Sq.vertex[1].Y=3;
```





Cấu trúc phức tạp

Mảng của cấu trúc

- Định nghĩa, khai báo

```
typedef struct {
   char  hoten[30];
   char  ngaysinh[11];
   float diemTBTL;
} Sinhvien;
Sinhvien dssv[10];
```

```
struct Sinhvien {
   char  hoten[30];
   char  ngaysinh[11];
   float diemTBTL;
};
struct Sinhvien dssv[10];
```

	hoten	ngaysinh	diemTBTL
(dssv) dssv[0]	"Nguyen Van An"	"01/02/2000"	3.25
(dssv+1) dssv[1]	…	…	…
(dssv+2) dssv[2]	…	…	…
(dssv+3) dssv[3]	…	…	…

Cấu trúc phức tạp

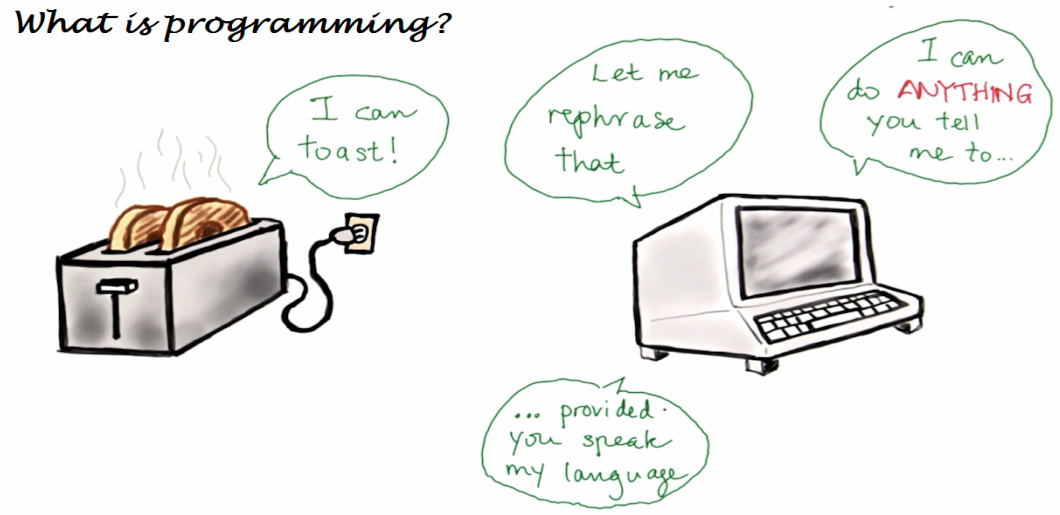
Mảng của cấu trúc

- Truy xuất từng trường

	hoten	ngaysinh	diemTBTL
(dssv) dssv[0]	""		0
(dssv+1) dssv[1]	...	...	...
(dssv+2) dssv[2]	...	...	...
(dssv+3) dssv[3]	...	...	...

```
for (i=0; i<10; i++) {
        dssv[i].diemTBTL = 0;
        strcpy(dssv[i].hoten, "");
        //...
}
```

```
for (i=0; i<10; i++) {
        (dssv+i)->diemTBTL = 0;
        strcpy((dssv+i)->hoten, "");
        //...
}
```



Con trỏ

Khoa CNTT&TT – ĐHCT

Con trỏ

- Con trỏ là 1 biến đặc biệt: chứa địa chỉ của một ô nhớ (hay 1 biến khác)

  vs. biến thường: chứa dữ liệu.

- Khi con trỏ chứa địa chỉ của 1 ô nhớ (biến), *ta nói là*: con trỏ đang trỏ tới (hay tham chiếu tới) ô nhớ (biến) đó.

- **Tính chất**: con trỏ trỏ tới một ô nhớ (biến)
  ⇒ có thể truy xuất ô nhớ (biến) thông qua con trỏ

Con trỏ

- Khai báo con trỏ
- Các phép toán trên con trỏ
- Con trỏ đến con trỏ
- Con trỏ đến cấu trúc
  - Con trỏ đến biến kiểu cấu trúc
  - Mảng động các phần tử kiểu cấu trúc
- Truyền tham số bằng con trỏ

Trong CT177:

- Cài đặt các phép toán (của các cấu trúc dữ liệu) dùng phương pháp truyền con trỏ
- Hiểu cách tổ chức dữ liệu của các cấu trúc dữ liệu cơ bản và cây

Con trỏ

- Cú pháp khai báo:

```
<kiểu dữ liệu> *<tên con trỏ>;
```

  - Kiểu dữ liệu của một con trỏ: xác định *kiểu biến* hay *kiểu của dữ liệu của vùng nhớ* mà con trỏ có thể trỏ tới.
  - Kiểu dữ liệu của con trỏ có thể là bất kỳ kiểu gì.
  - Tên con trỏ: đặt theo qui tắc đặt tên biến.
  - Có thể khai báo nhiều con trỏ trong một câu lệnh.

- Ví dụ:

```
int *p;
float *x, *z;
```

Chú ý: Kích thước của các con trỏ luôn bằng nhau, không phụ thuộc vào kiểu dữ liệu của con trỏ.

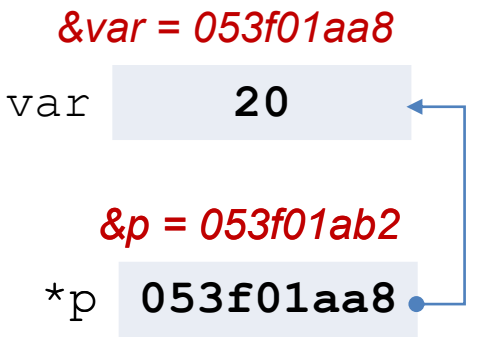
Con trỏ

- Các phép toán trên con trỏ

Phép toán	Ý nghĩa
* *(defererence)*	1. Dùng để khai báo một con trỏ. 2. Truy xuất vùng nhớ (biến) con trỏ đang trỏ đến.
& *(reference)*	Lấy địa chỉ của một biến (địa chỉ vùng nhớ được cấp phát cho biến)

```
int main (){
    int  var = 20;
    int  *p;

    p = &var; //cho p tham chiếu tới var
    printf("%x\n", &var);
    printf("%x\n", p);
    printf("%d\n", *p);
    printf("%x\n", &p);
}
```

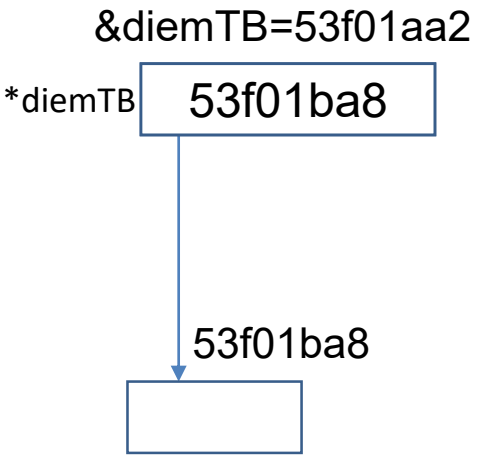
⇒ 53f01aa8
⇒ 53f01aa8
⇒ 20
⇒ 53f01ab2



Con trỏ

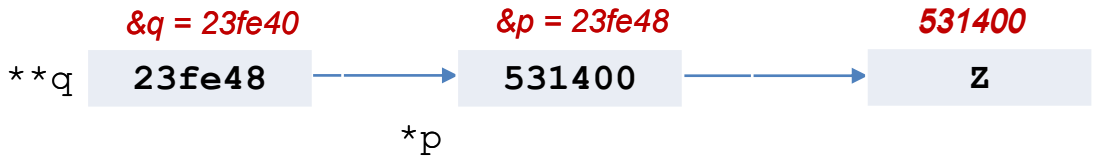
- Con trỏ không chỉ có thể trỏ đến một biến mà còn có thể trỏ đến các vùng nhớ được cấp phát "động".
- Cấp phát vùng nhớ động:
  - `void* malloc(size_t size)`
  - `void* calloc(size_t num, size_t size)`
- Thu hồi vùng nhớ được cấp phát động:
  - `void free(void *ptr)`

```
int main() {
    float *diemTB;
    diemTB = (float*)malloc(sizeof(float));
    ...
    free(diemTB);
}
```



Con trỏ đến con trỏ

```
int main(){
        char *p;
        char **q;
        q=&p;
        (*q)=(char *)malloc(sizeof(char));
        **q='Z';  // //*p='Z';
        printf("&p=%x, p=%x, *p=%c\n", &p, p, *p);
        printf("&q=%x, q=%x, *q=%x, **q=%c", &q, q, *q, **q);
        return 0;
}
```



&p=23fe48, p=531400, *p=Z
&q=23fe40, q=23fe48, *q=531400, **q=Z

Con trỏ đến cấu trúc

- Một con trỏ có thể trỏ đến một biến/vùng nhớ có dữ liệu thuộc bất kỳ kiểu gì

  ⇒ một con trỏ có thể trỏ đến một cấu trúc hoặc sử dụng như là một mảng động các phần tử kiểu cấu trúc.

- Ví dụ:

```
typedef struct {
   char hoten[30];
   char ngaysinh[11];
   float diemTBTL;
} Sinhvien;
```

Con trỏ trỏ đến biến kiểu cấu trúc

```
Sinhvien sv1;
Sinhvien *p = &sv1;
```

Mảng động các phần tử kiểu cấu trúc

```
Sinhvien *dssv;
dssv = (Sinhvien*)malloc(
            sizeof(Sinhvien)*10);
//hoặc
dssv = (Sinhvien*)calloc(10,
              sizeof(Sinhvien));
```

Con trỏ đến cấu trúc

Con trỏ trỏ đến biến kiểu cấu trúc

- Truy xuất các phần tử của cấu trúc: có 2 cách
  - Dùng toán tử * (dereference) kết hợp với toán tử . (dot)

```
(*p).diemTBTL = 8.5;
strcpy((*p).hoten, "TCAN");
strcpy((*p).ngaysinh, "23/12/78");
```

  - Dùng toán tử -> (arrow operator)

```
p->diemTBTL = 8.5
strcpy(p->hoten, "TCAN");
strcpy(p->ngaysinh, "23/12/78");
```

```
typedef struct {
   char hoten[30];
   char ngaysinh[11];
   float diemTBTL;
} Sinhvien;
```

```
Sinhvien sv1;
Sinhvien *p = &sv1;
```

```
Sinhvien sv1;
Sinhvien *p;
p = &sv1;
```

Con trỏ đến cấu trúc

Mảng động các phần tử kiểu cấu trúc

```
dssv = (Sinhvien*)malloc(
    sizeof(Sinhvien)*10);
```

- Lưu ý: khi dùng con trỏ như là mảng, một phần tử của mảng (truy xuất bằng toán tử []) là một cấu trúc chứ không phải là một con trỏ.

```
for (i=0; i<10; i++) {
  dssv[i].diemTBTL = 0;
  strcpy(dssv[i].hoten, "");
  //...
 }
```

Phần tử thứ i của mảng, là một cấu trúc

sai

```
dssv[i]->diemTBTL = 0;
strcpy(dssv[i]->hoten, "");
```

- Hoặc:

```
for (i=0; i<10; i++) {
  (dssv+i)->diemTBTL = 0;
  strcpy((dssv+i)->hoten, "");
  //...
 }
```

một con trỏ, trỏ tới phần tử thứ i của mảng

CT101 – Lập trình căn bản

Khoa CNTT&TT – ĐHCT

Hàm - Lưu ý

```c
#include <stdio.h>
int maximum(int A, int B){
    if(A>=B) return A;
    else return B;
}
int main(){
 int x, y, max;
 scanf("%d%d", &x, &y);
 max = maximum(x, y);
 printf("The maximum of the
 integers you entered was %d",max);
 return 0;
}
```

```c
#include <stdio.h>
int maximum(int A, int B);
int main(){
 int x, y, max;
 scanf("%d%d", &x, &y);
 max = maximum(x, y);
 printf("The maximum of the
 integers you entered was %d",max);
 return 0;
}
int maximum(int A, int B){
    if(A>=B) return A;
    else return B;
}
```

Hàm - Lưu ý

- Xác định phạm vi của các biến A, B, x, y, max?

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

2.  int maximum(int A, int B){
3.      if(A>=B) return A;
4.      else return B;
5.  }
6.  int main(){
7.      int x, y, max;
8.      scanf("%d%d", &x, &y);
9.      max = maximum(x, y);
10.     printf("The maximum of the integers you entered was %d",max);
11.     return 0;
12. }
```

Định nghĩa hàm maximum

Gọi hàm maximum

Hàm đệ quy - Ví dụ (tính n!)

- Tính n! (n>=0)

  Cách 1 – không đệ quy:

  n! = 1 với n=0

  n! = 1 * 2 * 3 * ... * n với n>0

  Cách 2 – đệ quy:

  n! = 1 với n=0

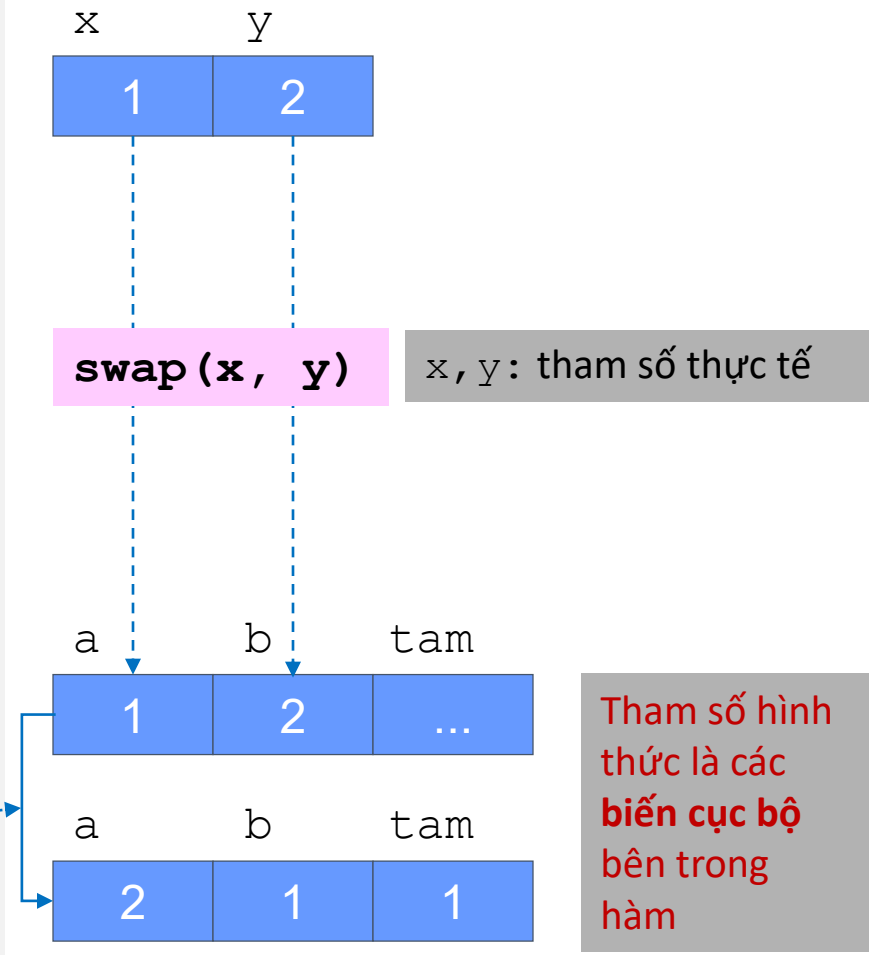
  n! = n * (n-1)! với n>0

Hàm – Truyền giá trị

```
int main() {
    int x = 1, y = 2;
    swap(x, y);
    printf("x=%d, y=%d", x, y);
    return 0;
}
```

tham số hình thức

```
void swap(int a, int b) {
    int tam;
    tam = a;
    a = b;
    b = tam;
}
```



Hàm – Truyền con trỏ

```
int main() {
    int x = 1, y = 2;
    swap(&x, &y);
    printf("x=%d, y=%d", x, y);
    return 0;
}
```

```
void swap(int *a, int *b) {
    int tam;
    tam = *a;
    *a = *b;
    *b = tam;
}
```