UIT Trường Đại học Công nghệ Thông tin | Khoa Khoa học và Kỹ thuật Thông tin

BAN HỌC TẬP

Chào mừng đến với buổi training của BHT KH&KTTT và BHT HTTT

NHẬP MÔN LẬP TRÌNH

Trainer: Đặng Trung Hậu (MMCL2021)
Võ Huy Hoàng (MTCL2021)
Trần Lê Gia Bảo (CNTT2021)

NỘI DUNG TRAIN

1. Mảng 1 chiều - Mảng 2 chiều
2. Chuỗi ký tự
3. Con trỏ
4. Cấu trúc

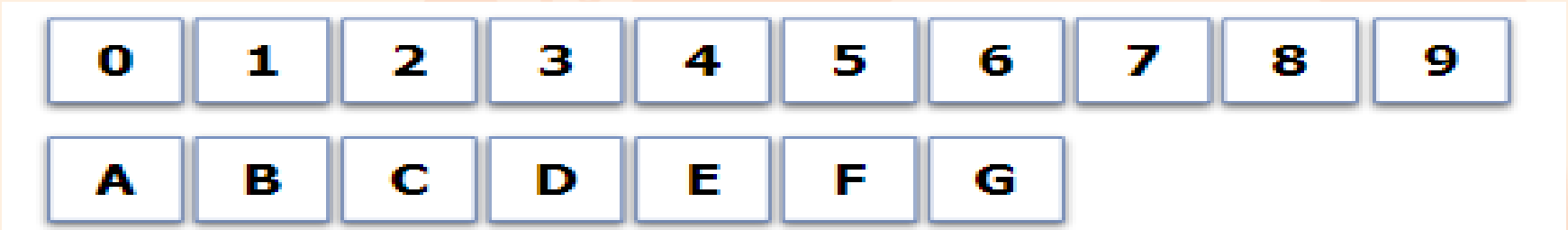
01

MẢNG 1 CHIỀU
MẢNG 2 CHIỀU

KHÁI NIỆM MẢNG

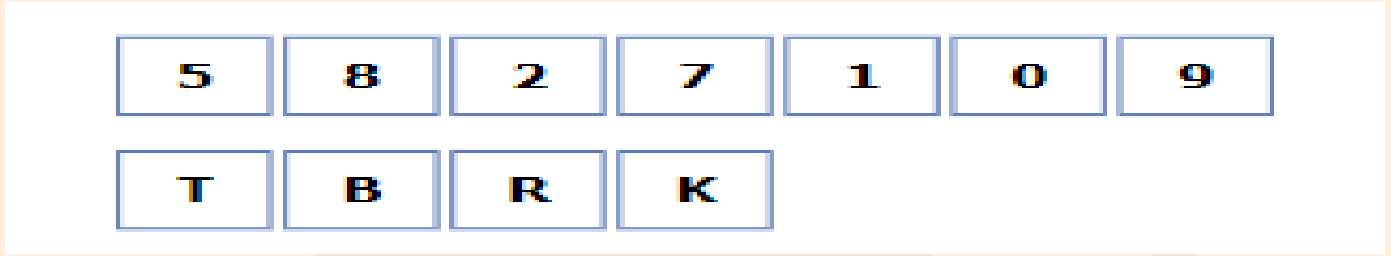
- Biểu diễn một dãy các phần tử có cùng kiểu và mỗi phần tử trong mảng biểu diễn 1 giá trị.
- Kích thước mảng được xác định ngay khi khai báo và không thay đổi.
- Một kiểu dữ liệu có cấu trúc do người lập trình định nghĩa.
- Ngôn ngữ lập trình C/C++ luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng

Ví dụ: Dãy các số nguyên, dãy các kí tự,…

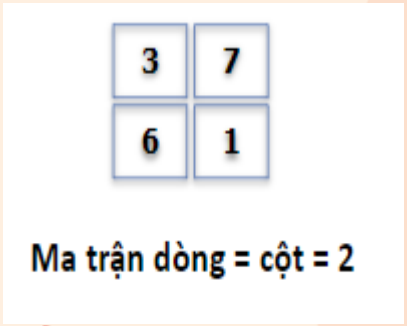


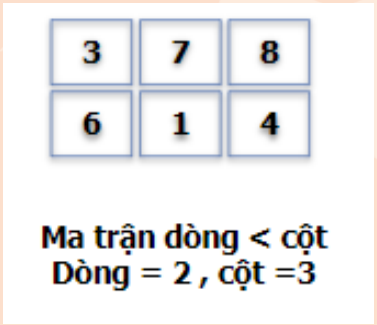
KHÁI NIỆM MẢNG

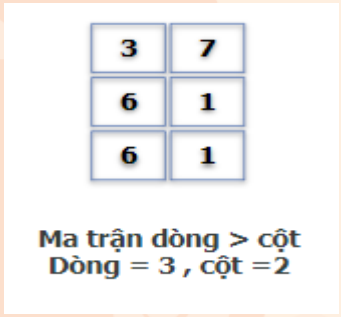
Mảng 1 chiều gồm một dãy các phần tử có cùng kiểu dữ liệu (int, float, char,…).



Mảng 2 chiều (ma trận) gồm các phần tử trên dòng và các phần tử trên cột.







CÚ PHÁP KHAI BÁO MẢNG 1 CHIỀU

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

Trong đó:

- Kiểu dữ liệu: int, float, char,…
- Tên biến mảng: 1 ký tự hoặc 1 dãy ký tự viết liền nhau và không có khoảng trắng
- Số phần tử mảng: số lượng các phần tử của mảng

```
char A[10];
```
Kiểu dữ liệu: char

Tên biến mảng: A

Số phần tử mảng: 10 phần tử

```
int Mang1Chieu[30];
```
Kiểu dữ liệu: int

Tên biến mảng: Mang1Chieu

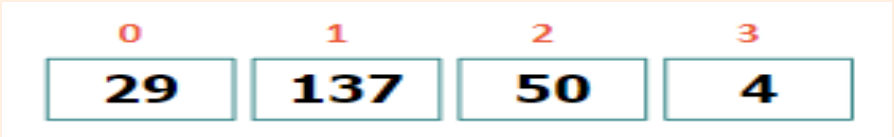
Số phần tử mảng: 30 phần tử

6

KHAI BÁO MẢNG 1 CHIỀU

- Khởi tạo giá trị cho mọi phần tử của mảng

```
int A[4] = {29, 137, 50, 4};
```



- Khởi tạo giá trị cho một số phần tử đầu mảng

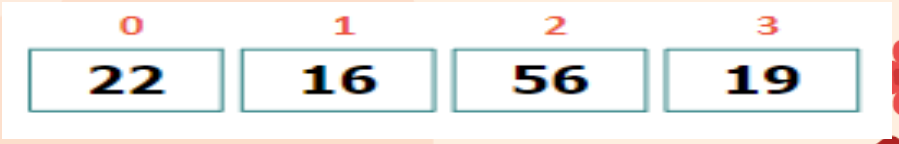
```
int B[4] = {91, 106};
```

- Khởi tạo giá trị 0 cho mọi phần tử của mảng

```
int a[4] = {0};
```

- Tự động xác định số lượng phần tử

```
int a[] = {22, 16, 56, 19};
```



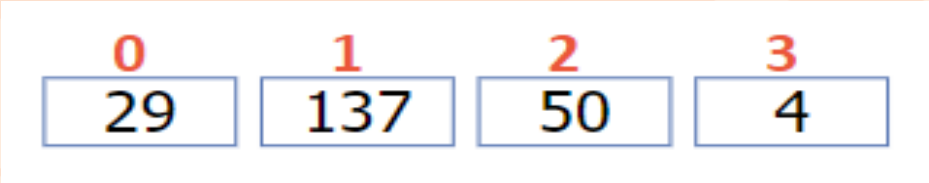
7

TRUY XUẤT PHẦN TỬ MẢNG 1 CHIỀU

- Truy xuất phần tử mảng thông qua chỉ số

<Tên biến mảng> [<chỉ số mảng>]

- Các phần tử mảng là 1 dãy liên tục có chỉ số từ 0 đến <Số phần tử mảng>-1.



int A[4] ;

Các truy xuất hợp lệ: A[0], A[1], A[2], A[3]
Các truy xuất không hợp lệ: A[-1], A[4], A[5]
Giá trị các phần tử mảng: A[0]=29, A[1]=137, A[2]=50, A[3]=4

8

MỘT SỐ LỖI THƯỜNG GẶP

- Khai báo không chỉ rõ số lượng phần tử:

```
int a[];
int a[100];
```

- Số lượng phần tử liên quan đến biến hoặc hằng:

```
int n1 = 10; int a[n1];
int a[10];
const int n2 = 10; int a[n2];
```

- Khởi tạo cách biệt với khai báo:

```
int a[4]; a = {2912, 1706, 1506, 1904};
int a[4] = {2912, 1706, 1506, 1904};
```

- Chỉ số mảng không hợp lệ:

```
a[-1] = 1;
```


9

MỘT SỐ LỖI THƯỜNG GẶP

- Khai báo không chỉ rõ số lượng phần tử:

  `int a[];` => LỖI

  `int a[100];` => OK

- Số lượng phần tử liên quan đến biến hoặc hằng:

  `int n1 = 10; int a[n1];` => LỖI

  `int a[10];` => OK

  `const int n2 = 10; int a[n2];` => OK

- Khởi tạo cách biệt với khai báo:

  `int a[4]; a = {2912, 1706, 1506, 1904};` => LỖI

  `int a[4] = {2912, 1706, 1506, 1904};` => OK

- Chỉ số mảng không hợp lệ:

  `a[-1] = 1;` => LỖI

10

TRUYỀN MẢNG 1 CHIỀU CHO HÀM VÀ LỜI GỌI HÀM

- Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

```
void SapXep(int A[100], int n);
```
Tên hàm: SapXep
Tham số: kiểu mảng số nguyên A và số lượng phần tử mảng n
Giá trị trả về: không có giá trị trả về void

```
int TinhTong(int A[100], int n);
```
Tên hàm: TinhTong
Tham số: kiểu mảng số nguyên A và số lượng phần tử mảng n
Giá trị trả về: kiểu số nguyên int

11

TRUYỀN MẢNG 1 CHIỀU CHO HÀM VÀ LỜI GỌI HÀM

- Mảng có thể thay đổi nội dung sau khi thực hiện hàm.
- Có thể bỏ số lượng phần tử hoặc dùng con trỏ.

```
void NhapMang(int A[], int n);
void Nhapmang(int *A, int n);
```


12

CÚ PHÁP KHAI BÁO MẢNG 2 CHIỀU

Cú pháp: <kiểu dữ liệu> <tên biến mảng>[<Số dòng>] [<Số cột>];

Trong đó:

- Kiểu dữ liệu: int, float, char,…
- Tên biến mảng: 1 ký tự hoặc 1 dãy ký tự viết liền nhau và không có khoảng trắng
- Số dòng, số cột: số lượng phần tử trên mỗi chiều của mảng

```
char A[10][20];
```
Kiểu dữ liệu: char

Tên biến mảng: A

Mảng có 10 dòng và 20 cột

```
int Mang2Chieu[3][5];
```
Kiểu dữ liệu: int

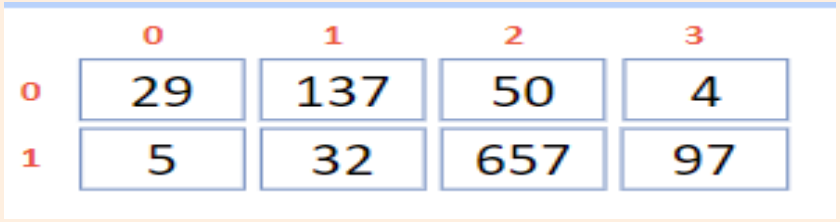
Tên biến mảng: Mang2Chieu

Mảng có 3 dòng và 5 cột

13

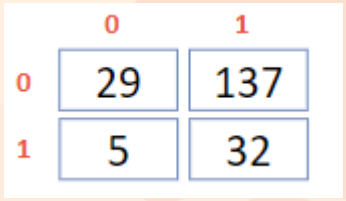
KHAI BÁO MẢNG 2 CHIỀU

```
int A[2][4];
```



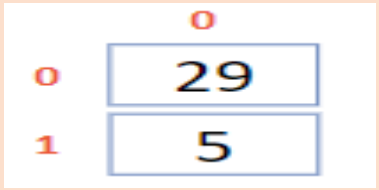
	0	1	2	3
0	29	137	50	4
1	5	32	657	97

```
int B[2][2];
```



	0	1
0	29	137
1	5	32

```
int C[2][1]
```



	0
0	29
1	5

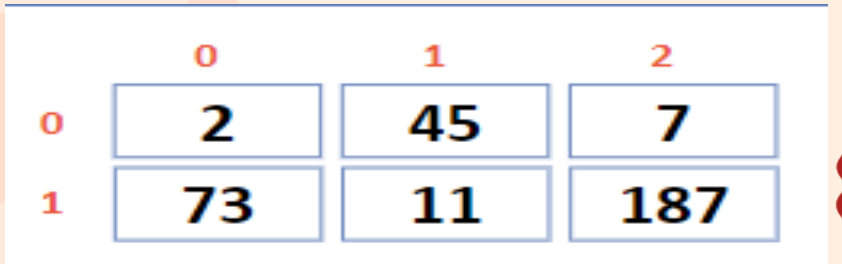
14

CHỈ SỐ MẢNG 2 CHIỀU

- Chỉ số mảng là một giá trị số nguyên int.
- Chỉ số trong mảng 2 chiều gồm chỉ số dòng và chỉ số cột.
  - 0 ≤ chỉ số dòng ≤ số dòng của mảng -1
  - 0 ≤ chỉ số cột ≤ số cột của mảng -1

```
int A[2][3];
```

Tên mảng: A
Kiểu dữ liệu của từng phần tử trong mảng: int
Số phần tử tối đa trong mảng: 2*3=6 phần tử
Các chỉ số được đánh số: Chỉ số dòng: 0, 1,
Chỉ số Cột: 0, 1, 2



	0	1	2
0	2	45	7
1	73	11	187

15

TRUY XUẤT PHẦN TỬ MẢNG 2 CHIỀU

- Truy xuất phần tử mảng thông qua chỉ số

  <Tên biến mảng>[<Số dòng>] [<Số cột>];

```
int A[2][3]
```

Các truy xuất hợp lệ: A[0][0], A[0][1], … , A[1][1], A[1][2]
Các truy xuất không hợp lệ: A[-1][0], A[1][4], A[2][0]
Giá trị các phần tử mảng:

A[0][0]=29, A[0][1]=137, A[0][2]=50
A[1][0]=3, A[1][1]=78, A[1][2]=943

0	1	2	
29	137	50	0
3	78	943	1

16

TRUYỀN MẢNG 2 CHIỀU CHO HÀM VÀ LỜI GỌI HÀM

- Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

```
int TinhDCheo(int A[50][50], int n, int m);
```
Tên hàm: TinhDCheo
Tham số: kiểu mảng số nguyên A và số lượng dòng n, số lượng cột m
Giá trị trả về: kiểu số nguyên int

```
void XuatMang(int A[50][50], int n, int m);
```
Tên hàm: XuatMang
Tham số: kiểu mảng số nguyên A và số lượng dòng n, số lượng cột m
Giá trị trả về: Không có kiểu trả về void

17

TRUYỀN MẢNG 2 CHIỀU CHO HÀM VÀ LỜI GỌI HÀM

- Mảng có thể thay đổi nội dung sau khi thực hiện hàm.
- Có thể bỏ số lượng phần tử hoặc sử dụng con trỏ.

```
void NhapMang(int A[][50], int n, int m);
void NhapMang(int (*A)[50], int n, int m);
```


18

02

CHUỖI KÝ TỰ

KHÁI NIỆM CHUỖI KÝ TỰ

- Kiểu char chỉ chứa được một ký tự. Để lưu trữ một chuỗi (nhiều ký tự) ta sử dụng mảng (một chiều) các ký tự.
- Chuỗi ký tự kết thúc bằng ký tự "\0" (null).
- Độ dài chuỗi = kích thước mảng –1.

```
char Hoten[30]; // Dài 29 ký tự
char NgaySinh[9]; // Dài 8 ký tự
```


20

KHAI BÁO CHUỖI KÝ TỰ

Các kiểu khai báo chuỗi:

```
char sName[100];
char sName[];
char *sName;
```

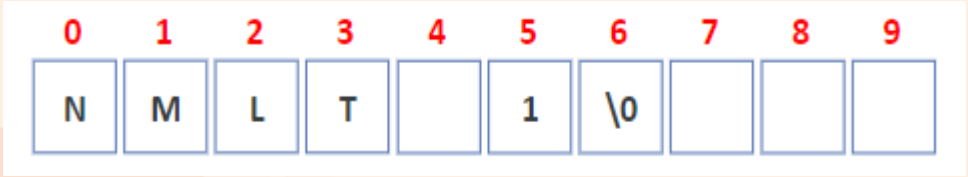

21

KHỞI TẠO CHUỖI KÝ TỰ

Khởi tạo như mảng thông thường:

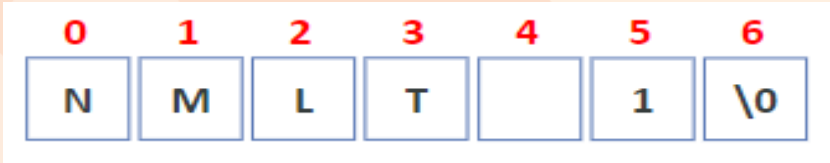
• Độ dài cụ thể:

```
char s[10] = {'N', 'M', 'L', 'T', ' ', '1', '\0'};
char s[10] = "NMLT 1"; // Tự động thêm '\0'
```



• Tự xác định độ dài:

```
char s[] = {'N', 'M', 'L', 'T', ' ', '1', '\0'};
char s[] = "NMLT 1"; // Tự động thêm '\0'
```



22

NHẬP XUẤT CHUỖI KÝ TỰ

- Hàm nhập chuỗi: gets, std::getline

Ví dụ: gets(hoten);

- Hàm tự động thêm ký tự NULL ('\0') vào cuối biến chuỗi.

```
void nhapchuoi(char s[100]) {
printf("Nhap chuoi: ");
gets(s);
// C2: cin.get(s, 100);
// C3: cin.getline(s, 100);
};
```


23

NHẬP XUẤT CHUỖI KÝ TỰ

- Hàm xuất chuỗi: puts, cout, printf

Ví dụ: puts(hoten);

```
void Xuatchuoi(char s[100]) {
printf("Xuatchuoi: ");
puts(s);
// C2: printf("%s", s);
// C3: std::cout << s;
};
```

03

CON TRỎ

CON TRỎ

Con trỏ là gì?

Con trỏ là **một biến đặc biệt** dùng để lưu **địa chỉ** của một biến khác trong bộ nhớ, và nó có thể trỏ đi khắp bộ nhớ mà ta có thể sử dụng.

Con trỏ là một biến nên nó cũng có tên, địa chỉ và giá trị.

Con trỏ khác biến bình thường ở chỗ giá trị của 1 con trỏ là địa chỉ của một biến khác.

Vì chỉ lưu mỗi địa chỉ bộ nhớ, các biến con trỏ đều có kích thước giống nhau.

Biến thường	Biến con trỏ
int a = 1;	int *ptr = &a;

CON TRỎ

Tác dụng của con trỏ

Sử dụng, quản lý bộ nhớ linh hoạt hơn.

Sử dụng để cấp phát động bộ nhớ.

CON TRỎ

Khai báo con trỏ

Cú pháp khai báo:

```
<Kiểu dữ liệu>* <tên biến>;
```

Trong đó:

- Kiểu dữ liệu là các kiểu dữ liệu có sẵn (int, float…) hoặc là kiểu dữ liệu người dùng tự định nghĩa.
- Dấu *: biểu thị đây là biến con trỏ
- Tên biến: tuân theo qua tắc đặt tên của biến.

Lưu ý: Kiểu dữ liệu của biến con trỏ không phải là kiểu dữ liệu của con trỏ mà là kiểu dữ liệu của giá trị vùng nhớ mà con trỏ trỏ đến.

CON TRỎ

Thao tác với con trỏ

Toán tử &: lấy địa chỉ của biến.

Toán tử *: lấy giá trị tại địa chỉ.

```cpp
#include<iostream>

using namespace std;

int main(){

    int a = 1;

    int *ptr = &a;

    cout<<"Gia tri cua &a la: "<<&a<<endl;
    cout<<"Gia tri cua *ptr la: "<<*ptr<<endl;

    return 0;
}
```

Kết quả:

```
Gia tri cua &a la: 0x7bfe14
Gia tri cua *ptr la: 1
```

CON TRỎ

Thao tác với con trỏ

```cpp
#include<iostream>

using namespace std;

int main(){

    int a = 10;
    cout<<"Truoc: "<<a<<endl;

    int *ptr = &a;
    *ptr = 5;
    cout<<"Sau: "<<a<<endl;

    return 0;
}
```

Kết quả:

```
Truoc: 10
Sau: 5
```

CON TRỎ

Ví dụ:

Khai báo kiểu con trỏ nào thì chỉ có thể lưu địa chỉ biến của con trỏ đó.

```cpp
#include<iostream>

using namespace std;

int main(){

    int a = 1,b, *ptr_a, *ptr_b;
    float *ptr_f;

    ptr_a = &a;
    ptr_b = ptr_a;

    ptr_f = ptr_a;
    ptr_f = &b;

    return 0;
}
```

```
ROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL

C++ vidu.cpp 2
⊗ a value of type "int *" cannot be assigned to an entity of type "float *" C/C++(513) [13, 11]
⊗ a value of type "int *" cannot be assigned to an entity of type "float *" C/C++(513) [14, 11]
```

CON TRỎ

Thao tác với con trỏ

Khi khai báo mà không gán địa chỉ biến con trỏ sẽ lưu một địa chỉ bất kì. Nếu không quản lý được thì sẽ gây lỗi chương trinh.

Con trỏ NULL: là một biến con trỏ không trỏ bất kì địa chỉ nào.

```
1   #include<iostream>
2
3   using namespace std;
4
5   int main(){
6
7       int *ptr = NULL;
8
9       cout<<*ptr;
10      //câu lệnh không chạy vì ptr không có giá trị
11
12      return 0;
13  }
```

CON TRỎ

Hàm nhận đối số là con trỏ

```cpp
#include<iostream>

using namespace std;

void hoan_vi(int *ptr_a, int *ptr_b){
    int tm = *ptr_a;
    *ptr_a = *ptr_b;
    *ptr_b = tm;
}

int main(){

    int a = 10, b = 5;
    cout<<"Truoc: "<<a<<" "<<b<<endl;

    hoan_vi(&a,&b);// tham số là biến con trỏ nên cần truyền vào một địa chỉ
    cout<<"Sau: "<<a<<" "<<b<<endl;

    return 0;
}
```

Kết quả:

```
Truoc: 10 5
Sau: 5 10
```

CON TRỎ

Hàm nhận đối số là con trỏ

Truyền tham chiếu

```
5   void hoan_vi(int &a, int &b){
6       int tm = a;
7       a = b;
8       b = tm;
9   }
```

Truyền tham trị với đối số là con trỏ

```
5   void hoan_vi(int *ptr_a, int *ptr_b){
6       int tm = *ptr_a;
7       *ptr_a = *ptr_b;
8       *ptr_b = tm;
9   }
```

CON TRỎ

Các toán tử dùng cho con trỏ

Toán tử - : khoảng cách của 2 con trỏ trên bộ nhớ (tính theo số phần tử)

```cpp
#include<iostream>

using namespace std;

int main(){

    int a = 1, b = 2;
    int *ptr_a = &a; //0x7bfe04
    int *ptr_b = &b; //0x7bfe00
    cout<<ptr_a - ptr_b<<endl; // kết quả 1

    float c = 1.2f;
    float *ptr_c = &c;
    cout<<ptr_a - ptr_c; // Lỗi do 2 con trỏ không cùng kiểu

    return 0;
}
```

CON TRỎ

Các toán tử dùng cho con trỏ

Toán tử + : tăng địa chỉ của con trỏ lên một đơn vị lên tùy theo kích thước kiểu dữ liệu

```cpp
#include<iostream>

using namespace std;

int main(){

    int a;
    int *ptr_a = &a;

    cout<<ptr_a<<endl; //0x7bfe14
    cout<<ptr_a+1<<endl;
    cout<<ptr_a+2<<endl;
    cout<<ptr_a+3<<endl;

    return 0;
}
```

Kết quả:

```
0x7bfe14
0x7bfe18
0x7bfe1c
0x7bfe20
```

```cpp
#include<iostream>

using namespace std;

int main(){

    double a;
    double *ptr_a = &a;

    cout<<ptr_a<<endl; //0x7bfe10
    cout<<ptr_a+1<<endl;
    cout<<ptr_a+2<<endl;
    cout<<ptr_a+3<<endl;

    return 0;
}
```

Kết quả:

```
0x7bfe10
0x7bfe18
0x7bfe20
0x7bfe28
```

CON TRỎ

Con trỏ và mảng một chiều

Mảng là một dãy các biến được lưu liên tiếp nhau trên vùng nhớ.
Địa chỉ của các phần tử trong mảng liên tiếp nhau.

```cpp
#include<iostream>

using namespace std;

int main(){

    int arr[5] = {2,3,1,4,10};

    cout<<&arr[0]<<endl;
    cout<<&arr[1]<<endl;
    cout<<&arr[2]<<endl;
    cout<<&arr[3]<<endl;
    cout<<&arr[4]<<endl;

    return 0;
}
```

Kết quả:

```
0x7bfe00
0x7bfe04
0x7bfe08
0x7bfe0c
0x7bfe10
```

- Ta có thể sử dụng con trỏ để thao tác với phần từ mảng.

CON TRỎ

Con trỏ và mảng một chiều

```
1   #include<iostream>
2
3   using namespace std;
4
5   int main(){
6
7       int arr[5] = {2,3,1,4,10};
8
9       cout<<arr<<endl;
10      cout<<arr+1<<endl;
11      cout<<arr+2<<endl;
12      cout<<arr+3<<endl;
13      cout<<arr+4<<endl;
14
15      return 0;
16  }
```

Kết quả:

```
0x7bfe00
0x7bfe04
0x7bfe08
0x7bfe0c
0x7bfe10
```

arr tương đương với &arr[0]
arr+1 tương đương với &arr[1]

arr là một con trỏ hằng.

Con trỏ hằng tương tự biến hằng. Con trỏ trỏ đến địa chỉ mà không thể thay đổi địa chỉ đó. Tuy nhiên vẫn có thể thay đổi giá trị lưu ở địa chỉ đó.

CON TRỎ

Cấp phát động

Cấp phát tĩnh:
- Khai bao biến, cấu trúc, mảng,…
- Cần phải biết trước kích thước bộ nhớ để lưu trữ, không thể thay đổi kích thước, không thể giải phóng bộ nhớ không còn sử dụng khi chương trinh còn chạy.

Cấp phát động:
- Không cần thiết phải biết trước kích thước bộ nhớ để lưu trữ.
- Có thể cấp phát thêm bộ nhớ khi chương trinh đang chạy.
- Có thể giải phóng bộ nhớ nếu không sử dụng.

CON TRỎ

Cấp phát động: Cấp phát

Toán tử new: Cấp phát một địa chỉ ô nhớ cho con trỏ
Ta sử dụng bộ nhớ được cấp phát qua biến con trỏ

```cpp
#include<iostream>

using namespace std;

int main(){

    // Cú pháp <Kiểu dữ liệu> *<tên con trỏ> = new <tên con trỏ>
    int *ptr = new int;

    *ptr = 9;
    cout<<"Gia tri tai dia chi ptr la: "<<*ptr;

    return 0;
}
```

Kết quả:

```
Gia tri tai dia chi ptr la: 9
```

CON TRỎ

Cấp phát động: Cấp phát

Mảng tĩnh:

```cpp
#include<iostream>

using namespace std;

int main(){

    int arr[10];

    for (int i = 0; i < 10; i++)
    {
        cout<<arr[i];
    }

    return 0;
}
```

Mảng động:

```cpp
#include<iostream>

using namespace std;

int main(){

    int n = 10;
    int *arr = new int[n];

    for (int i = 0; i < n; i++)
    {
        cout<<arr+i;
    }
    return 0;
}
```

CON TRỎ

Cấp phát động: Giải phóng

Toán tử delete: Giải phóng vùng nhớ đã cấp phát cho con trỏ bằng toán tử new.

```cpp
#include<iostream>

using namespace std;

int main(){

    int *ptr = new int;
    *ptr = 6;
    cout<<*ptr<<endl;

    //Cú pháp: detlete <tên con trỏ>;
    delete ptr;
    cout<<*ptr<<endl;

    return 0;
}
```

Kết quả:

```
6
9968224
```

CON TRỎ

Cấp phát động: Giải phóng

Toán tử delete chỉ **giải phóng bộ nhớ được cấp phát**. Nhưng con trỏ vẫn sẽ **trỏ về địa chỉ vùng nhớ đã được giải phóng** đó. Đó là "**con trỏ lạc**".

Ta vẫn có thể thao tác trên vùng nhớ đó nhưng:

- Kết quả sẽ ngẫu nhiên.
- Dễ gây nguy hiểm cho chương trình.
- Để tránh trường hợp con trỏ lạc, sau khi delete, ta sẽ gán NULL cho con trỏ.

```
int *ptr = new int;
*ptr = 6;
cout<<*ptr<<endl;

delete ptr;
ptr = NULL;
cout<<*ptr<<endl;// Câu lệnh không chạy vì ptr không có giá trị
```

CON TRỎ

Bài tập

Kết quả của đoạn chương trinh sau:

```cpp
#include<iostream>

using namespace std;

int main(){

    int a = 5;
    int *ptr = &a; // ptr = 0x7bfe1c

    cout<<*&ptr<<endl;
    cout<<**&ptr<<endl;

    return 0;
}

```

Kết quả:

```
0x7bfe1c
5
```

CON TRỎ

Bài tập

Chương trình sẽ lỗi ở đâu:

```
1   #include<iostream>
2
3   using namespace std;
4
5   int main(){
6
7       int arr[5] = {1,2,3,4,5};
8
9       cout<<&arr;
10
11      for (int i = 0; i < 5; i++)
12      {
13          cout<<arr<<endl;
14          arr += 1;
15      }
16
17      return 0;
18  }
```

Dòng 14 sẽ lỗi do arr là con trỏ hằng, không thể thay đổi giá trị của con trỏ hằng

CON TRỎ

Bài tập

Kết quả đoạn chương trinh sau:

```cpp
#include<iostream>

using namespace std;

void ham(int *a){
    a = new int;
    *a = 10;
}

int main(){

    int a = 5;
    int *ptr_a = &a;
    cout<<*ptr_a<<endl;

    ham(ptr_a);
    cout<<*ptr_a<<endl;

    return 0;
}
```

Kết quả:

```
5
5
```

04

CẤU TRÚC

ĐẶT VẤN ĐỀ

Ví dụ: Thông tin của 1 sinh viên bao gồm tên, tuổi, giới tính, điểm số
Yêu cầu: lưu thông tin của n sinh viên

- Nhận xét
  - Tốn nhiều bộ nhớ
  - Truyền tham số cho hàm quá nhiều
  - Đặt tên biến rất khó khăn, dễ nhầm lẫn

- Giải Quyết
  => kiểu CẤU TRÚC(STRUCT)

KHÁI NIỆM

- Kiểu cấu trúc là kiểu dữ liệu tự định nghĩa.
- Kiểu cấu trúc bản chất là một kiểu dữ liệu do người lập trình định nghĩa một đối tượng.

```
struct diem
{
    int toan;
    int van;
    int tin;
};
struct in4
{
    int ID;
    diem a;
};
int main()
{
    in4 b = {1234,9,8,7};
    cout <<"ID: " << b.ID <<endl<<"toan: " << b.a.toan <<
" van: " << b.a.van << " tin: " << b.a.tin;
    return 0;
}
```

```
struct TS
{
    int width;
    int length;
};
void nhap(TS &HCN)
{
    cout << "Enter width: ";cin >> HCN.width;
    cout <<"Enter length: ";cin >> HCN.length;
}
int main()
{
    TS HCN;
     nhap(HCN);
    cout << "DIEN TICH HCN LA: << HCN.width * HCN.length;
    return 0;
}
```

KHAI BÁO KIỂU CẤU TRÚC

Cú Pháp

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

Ví Dụ

```
struct ngaysinh
{
    int d;
    int m;
    int y;
};
```

KHỞI TẠO CHO BIẾN CẤU TRÚC

CÚ PHÁP:

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

VÍ DỤ:

```
struct DIEM
{
        int x;
        int y;
};struct DIEM diem1,diem2;
// C++ có thể bỏ struct
```

TYPEDEF

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

VÍ DỤ:

```
struct diem
{
        int x;
        int y;
};
diem x1,x2;
```

KHỞI TẠO GIÁ TRỊ CHO BIẾN CẤU TRÚC

CÚ PHÁP

```
struct <tên kiểu cấu trúc>
{
<kiểu dữ liệu> <tên thành phần 1>;
...
<kiểu dữ liệu> <tên thành phần n>;
}
<tên biến> = {<giá trị 1>,...,<giá trị n>};
```

VÍ DỤ:

```
struct diem
{
  int x;
  int y;
}
diem a = {10,9};
```

TRUY XUẤT DỮ LIỆU KIỂU CẤU TRÚC

ĐẶC ĐIỂM:

- Không thể truy xuất một cách trực tiếp
- Thông qua toán tử thành phần cấu trúc hay còn gọi là toán từ chấm (dot operation)

CÚ PHÁP:

<tên biến cấu trúc>.<tên thành phần>

VÍ DỤ:

```
struct birthday
{
    int d;
    int m;
        int y;
};
birthday x={1,1,2003);
cout << x.d << "/" << x.m <<"/" << x.y;
```

GÁN DỮ LIỆU KIỂU CẤU TRÚC

CÓ 2 CÁCH:

- C1: <Biến cấu trúc đích> = <Biến cấu trúc nguồn>;
- C2: <Biến cấu trúc đích>. <tên thành phần> = <giá trị>;

VÍ DỤ:

```
struct birthday
{
    int d;
    int m;
    int y;
}; x={1,1,2003},y;
C1 : x = y;

C2 : x.d = y.d;
```

CẤU TRÚC PHỨC TẠP

Thành phần của cấu trúc là cấu trúc khác

```
struct birthday
{
    int d;
    int m;
    int y;
};
struct info
{
    int ID;
    struct birthday a;
    struct birthday b;
    struct birthday c;

};
    info.a.d = 1;
    info.b.m = 2;
```

```
struct diem
{
    int x;
    int y;
};
struct HCN
{
    struct diem a;
    struct diem b;
}hcn1;
hcn1.a.x = 10;
hcn1.b.y = 20;
```

CẤU TRÚC PHỨC TẠP

CẤU TRÚC ĐỆ QUY(TỰ TRỎ)

```
struct Person
{
    char hoten[50]
    struct person *father,*mother;
};
struct node
{
     int a;
     node *b;
};
```

TRUYỀN CẤU TRÚC CHO HÀM

- Giống như truyền dữ liệu cơ sở:
  - Tham trị (không thay đổi khi kết thúc hàm)
  - Tham chiếu
  - Con trỏ

Ví dụ :

```
struct Diem
{
    int x,y;
};
void xuat(int x, int y){…};
void xuat1(Diem diem){…};
void xuat2(Diem &diem){…};
void xuat3(Diem *diem){…};
```

CÁC LƯU Ý VỀ CẤU TRÚC

- Kiểu cấu trúc được định nghĩa để làm khuôn dạng còn biến cấu trúc được khai báo để sử dụng khuôn dạng đã định nghĩa
- Trong C++, có thể bỏ từ khóa struct khi khai báo biến (hoặc sử dụng typedef)
- Khi nhập các biến kiểu số thực trong cấu trúc phải nhập thông qua một biến trung gian

```
struct DIEM
{
 float x,y;
}d1;
float temp;
cin >> temp;
d1.x = temp;
```

BÀI TẬP:

Kết quả của đoạn code dưới đây là:

```
#include<iostream>

using namespace std;

struct Student {
    char gender;
    int age;
    double gpa;
};

int main() {
    cout << sizeof(Student);
    return 0;
}
```

KẾT QUẢ: 16

BÀI TẬP:

Kết quả của đoạn code dưới đây là:

```
#include<iostream>

using namespace std;

struct MyStruct {
};

int main() {
    cout << sizeof(MyStruct);
    return 0;
}
```

KẾT QUẢ : 1

XIN CẢM ƠN CÁC BẠN ĐÃ LẮNG NGHE