TRAINING CUỐI KÌ 1

XUÂN CÔNG NGHỆ - XUÂN TÌNH NGUYỆN 2022

MÔN: NHẬP MÔN LẬP TRÌNH

01

Hàm

Hàm

Không sử dụng hàm

```
int a, b, c, d;

do {
    cout << "Nhap mot so nguyen duong";
    cin >> a;
} while (a <= 0);
do {
    cout << "Nhap mot so nguyen duong";
    cin >> b;
} while (b <= 0);
do {
    cout << "Nhap mot so nguyen duong";
    cin >> c;
} while (c <= 0);
do {
    cout << "Nhap mot so nguyen duong";
    cin >> d;
} while (d <= 0);
```

Có sử dụng hàm

```
int nhap_so_duong(){
    int n;
    do {
        cout << "Nhap mot so nguyen duong";
        cin >> n;
    } while (n <= 0);
    return n;
}
```

```
a = nhap_so_duong();
b = nhap_so_duong();
c = nhap_so_duong();
d = nhap_so_duong();
```

Khái niệm hàm

Một đoạn chương trình có tên, đầu vào và đầu ra.

Có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.

Có thể được gọi nhiều lần với các đối số khác nhau.

Được sử dụng khi có nhu cầu:

- Tái sử dụng.
- Sửa lỗi và cải tiến.

Cú pháp của hàm

Cú Pháp: <**kiểu_trả_về**> <**tên_hàm**>(**[danh sách tham số]**){

<các câu lệnh>

return <**giá_trị_trả_về**>; }

Trong đó

- <**kiểu trả về**> : kiểu dữ liệu bất kỳ của C++ (char, int, long, float,...). Đặc biệt nếu hàm không trả về thì dùng kiểu void.
- <**tên hàm**>**:** theo quy tắc đặt tên định danh.
- <**danh sách tham số**> : tham số hình thức đầu vào giống khai báo biến, cách nhau bằng dấu “,”.
- <**giá trị**> : trả về cho hàm qua lệnh return. Nếu kiểu trả về là void thì không có lệnh này.

Hàm

```
int nhap_so_duong(){
    int n;
    do {
        cout << "Nhap mot so nguyen duong";
        cin >> n;
    } while (n <= 0);
    return n;
}
```

<int> : Kiểu dữ liệu trả về
<nhap_so_duong> : Tên hàm
<return n;> : Giá trị trả về qua biến n;

Tham số hàm

Tham Chiếu

Khai báo hàm sử dụng tham số dưới dạng tham chiếu bằng cách thêm dấu & vào trước tham số đó.
Khi một biến a được truyền vào lời gọi hàm **f(int &x)** làm tham số dưới dạng tham chiếu, thì biến x của hàm **f(int &x)** và biến a thực chất là một. Do đó, nếu hàm thay đổi giá trị của x trong hàm thì đồng nghĩa biến a cũng bị thay đổi theo.

Tham trị

Khi một biến a được truyền vào lời gọi hàm **f(int x)** làm tham số dưới dạng tham trị, thì biến x của hàm **f(int x)** và biến a là hai biến độc lập. Bởi vì khi tham số của hàm là tham trị, hàm này sẽ tạo ra một biến mới và sao chép giá trị của a vào. Do đó, nếu hàm mà thay đổi giá trị của x trong hàm thì không tác động gì tới giá trị của biến a.

Tham số hàm

```
bool phep_chia(int x, int y, double& thuong){
    if (y != 0) {
        thuong = double(x)/y;
        return true;
    } else {
        return false;
    }
}
int main(){
    double thuong;
    if (phep_chia(5, 3, thuong)){
        cout << "Thuong so la " << thuong;
    } else {
        cout << "Khong the chia duoc ";
    }
}
```

Tham trị

Tham chiếu

Bài tập về hàm

Đoạn chương trình bên có đúng không?
Nếu sai, thì phải sửa lại ở những điểm nào?

```
#include <iostream>
int n;
using namespace std;
void kiem_tra(int n) {
    for (int i = 2; i <= sqrt(n); i++)
        if (n % i = 0) cout << false;
    cout << true;
}
int main() {
    cin >> n;
    cout << kiem_tra(n);
    return 0;
}
```

Bài tập về hàm

```
#include <iostream>
int n;
using namespace std;
void kiem_tra(int n) {
   for (int i = 2; i <= sqrt(n); i++)
      if (n % i = 0) cout << false;
   cout << true;
}
int main() {
   cin >> n;
   cout << kiem_tra(n);
   return 0;
}
```

```
#include <iostream>
#include <cmath>
int n;
using namespace std;
bool kiem_tra(int n) {
   for (int i = 2; i <= sqrt(n); i++)
      if (n % i == 0) return false;
   return true;
}
int main() {
   cin >> n;
   cout << kiem_tra(n);
   return 0;
}
```

02
Mảng - Chuỗi

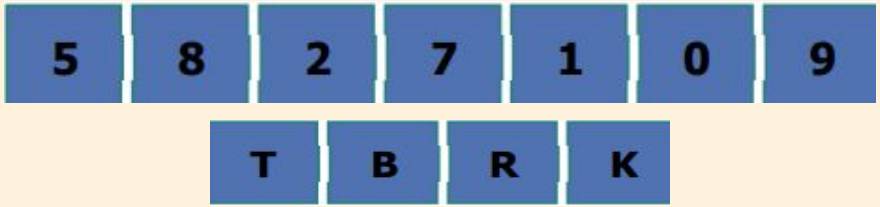
Khái niệm mảng

- 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.
- Ngôn ngữ lập trình C luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng.

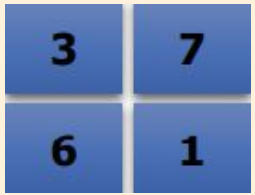
Ví dụ: dãy các số nguyên, dãy các ký tự...

Khái niệm mảng

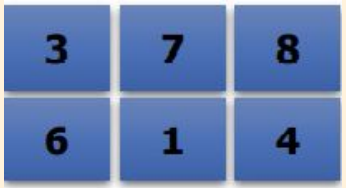
- Mảng 1 chiều gồm 1 dãy các phần tử có cùng kiểu dữ liệu (int, float, char …)



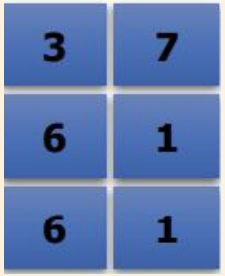
- Mảng 2 chiều (Ma trận) gồm các phần tử trên dòng và các phần tử trên cột



Ma trận dòng = cột = 2



Ma trận dòng < cột Dòng = 2 , cột =3



Ma trận dòng < cột Dòng = 3 , cột =2

Cú pháp của mảng 1 chiều

Cú Pháp: <**Kiểu dữ liệu**> <**Tên biến mảng**>[<**Số phần tử mảng**>];

Trong đó

- **Kiểu dữ liệu** : int, float, char, double…
- **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 1 chiều

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

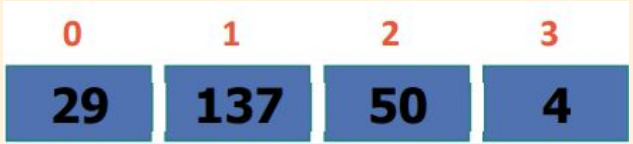
Tên biến mảng: Mang1Chieu

Số phần tử mảng: 30 phần tử

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};

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

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

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 sử dụng con trỏ.

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

void NhapMang(int *A, int n);
```

Cú pháp của mảng 2 chiều

Cú Pháp: <**Kiểu dữ liệu**> <**Tên biến mảng**>[<**Số dòng**>] [<**Số cột**>];

Trong đó

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

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]

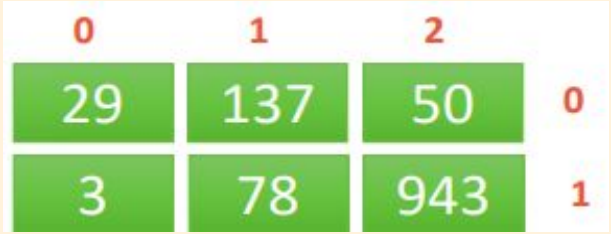
	0
0	29
1	5

Truy xuất phần tử mảng 2 chiều

- Truy xuất phần tử mảng thông qua chỉ số []

<Tên biến mảng>[<Chỉ số dòng>][<Chỉ số cột>]

int A[2][3]



Các truy xuất hợp lệ: A[0][0], A[0][1],..., A[1][2], A[1][3]
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

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

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);
```

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
```

Khai báo chuỗi ký tự

Các kiểu khai báo chuỗi

- char sName[100];
- char sName[];
- char *sName;

Khởi tạo ký tự

Khởi tạo như mảng thông thường

• Độ dài cụ thể

char s[10] = {'T', 'H', 'C', 'S', ' ', 'A', '\0'};

char s[10] = "THCS A"; // Tự động thêm '\0'

• Tự xác định độ dài

char s[] = {'T', 'H', 'C', 'S', ' ', 'A', '\0'};

char s[] = 'THCS A'; // Tự động thêm '\0'

Nhập xuất chuỗi

• Hàm nhập chuỗi: gets

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); // hàm nhập chuỗi
}
```

Nhập xuất chuỗi

- Hàm xuất chuỗi: puts

Ví dụ: puts(hoten);

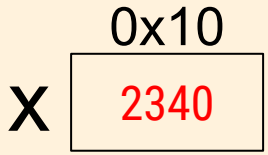
```
void Xuatchuoi(char s[100])
{
        printf("Xuatchuoi");
        puts(s); // hàm xuất chuỗi
}
```

03
Con trỏ

Toán tử & và *

Toán tử **&** đặt trước tên biến sẽ trả về địa chỉ vùng nhớ của biến đó.

Toán tử * đặt trước địa chỉ sẽ trả về giá trị của của vùng nhớ đó.



x =2340

&x = 0x10

*(&x) = 2340

Con trỏ

Con trỏ là một **biến lưu trữ địa chỉ**. Địa chỉ thường là địa chỉ của biến khác.

Khai báo:

<Kiểu dữ liệu>* <Tên biến con trỏ>;

<Tên biến con trỏ> = <Địa chỉ>;

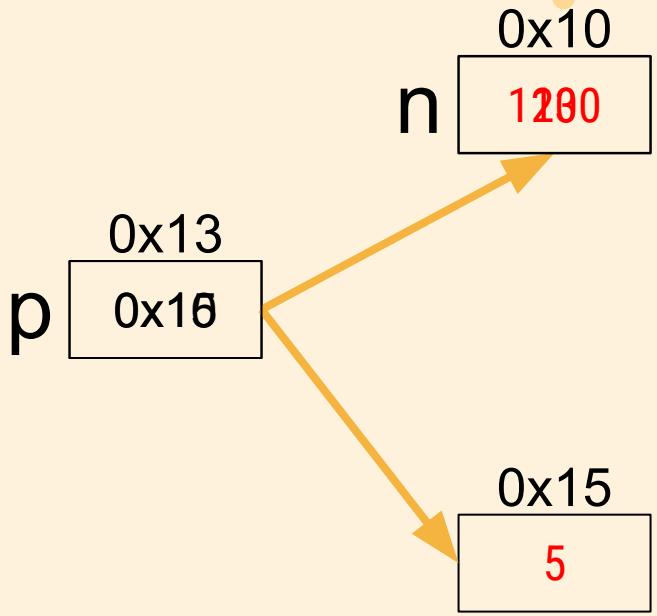
<Tên biến con trỏ> = **new** <Kiểu dữ liệu>; // cấp phát động

VD:

```
int a;          int* p;
p = &a;         p = new int;
```

Ví dụ con trỏ

```
#include <iostream>
using namespace std;

int main() {
    int n = 1230;
    int* p = &n;
    //double* p = &n; <- Sai
    cout << *p << " "; //1230
    *p = 10;
    cout << n; //10
    p = new int(5);
    cout << *p << " "; //5
    return 0;
}
```



Con trỏ và mảng

Cho mảng int arr[6] = {1, 5, 3, 2, 1, 3};

Mảng 1 chiều là một hằng con trỏ.

```
int* p = arr;
```

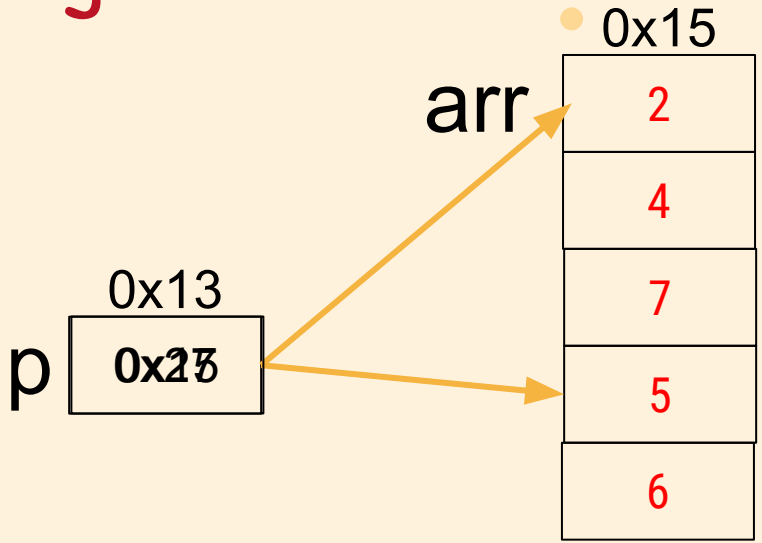
Vậy tên mảng chính là một địa chỉ và địa chỉ đó cũng là địa chỉ của mảng. Địa chỉ mảng là địa chỉ của phần tử đầu tiên của mảng.

Chúng ta có thể sử dụng phép toán + hoặc – trên con trỏ để tương tác với mảng.

```
cout << *(p + 2); //3
```

Ví dụ con trỏ và mảng

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {2, 4, 7, 5, 6};
    int* p = arr;
    cout << *p << " "; //2
    cout << arr[0] << " "; //2
    cout << *(p + 2) << " "; //7
    cout << arr2] << " "; //7
    p = arr + 3;
    cout << *p; //5
    return 0;
}
```



Cấp phát động

Khai báo biến động:

<Kiểu dữ liệu> <Tên biến con trỏ> = **new** <Kiểu dữ liệu> **(** Giá trị **)**;

```
int* p = new int(3);
```

Khai báo mảng động:

<Kiểu dữ liệu> <Tên biến con trỏ> = **new** <Kiểu dữ liệu> **[** Số lượng phần tử **]**;

```
int* arr = new int[5];
```

Lưu ý:

Khi sử dụng xong phải xóa biến để giải phóng vùng nhớ và phải gán bằng **NULL**

```
delete p;
p = NULL;
```

```
delete []arr;
arr = NULL;
```

Ví dụ cấp phát động

```
#include <iostream>
using namespace std;

int main() {
   int* arr = new int[5];
   for (int i = 0; i < 5; i++)
      arr[i] = i + 1;
   for (int i = 0; i < 5; i++)
      cout << *(arr + i) << “ ”; //1 2 3 4 5
   delete []arr;
   arr = NULL;
   return 0;
}
```

Con trỏ và hàm

```cpp
#include <iostream>
using namespace std;

void nhap(int &a)
{
    cout << “Nhap so: ”;
    cin >> a;
}

int main() {
    int a;
    nhap(a);
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

void nhap(int* a)
{
    cout << “Nhap so: ”;
    cin >> *a;
}

int main() {
    int a;
    nhap(&a);
    return 0;
}
```

Con trỏ và hàm

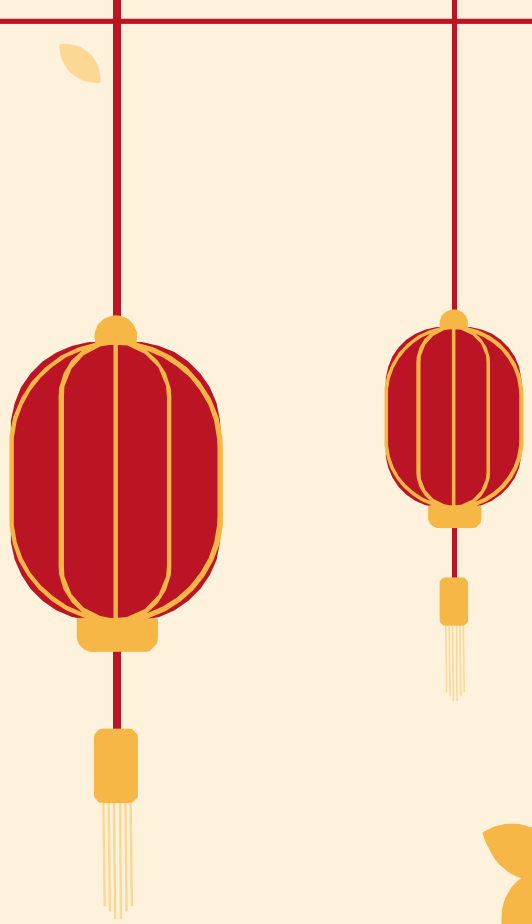
```
#include <iostream>
using namespace std;

void fun1(int* p)
{
    *p = 5;
    cout << *p << " "; // 5
}

int main() {
    int n = 10;
    int* p = &n;
    cout << *p << " "; // 10
    fun1(p);
    cout << *p << " "; // 5
    return 0;
}
```

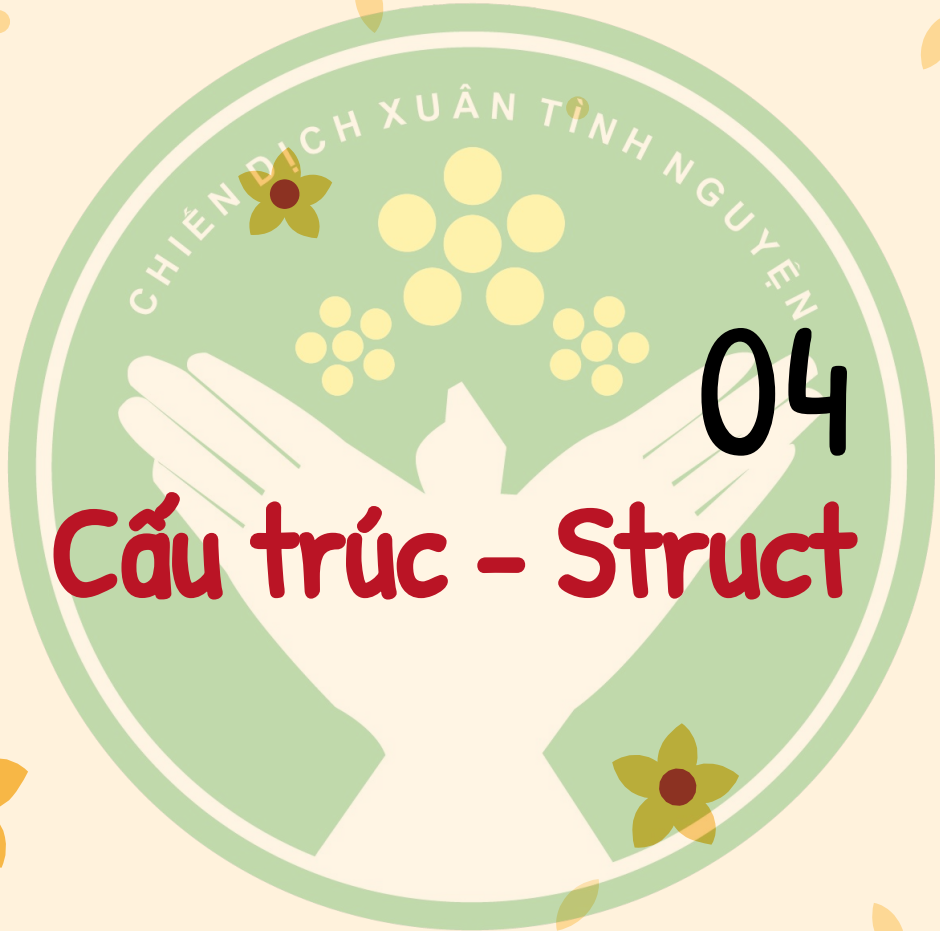
Con trỏ và hàm

```
#include <iostream>
using namespace std;

void fun1(int* p)
{
    p = new int(20);
    cout << *p << “ ”; // 20
}

int main() {
    int n = 10;
    int* p = &n;
    cout << *p << “ ”; // 10
    fun1(p);
    cout << *p << “ ”; // 10
    return 0;
}
```

Con trỏ và hàm

```
#include <iostream>
using namespace std;

int* nhap(int n)
{
   int* p = new int[n];
   for (int i = 0; i < n; i++)
     p[i] = i + 1;
   return p;
}

int main() {
   int* arr, n;
   cin >> n;
   arr = nhap(n);
   return 0;
}
```





04

Cấu trúc - Struct

Khái niệm Struct

- Kiểu cấu trúc (struct) là kiểu dữ liệu tự định nghĩa.
- Kiểu cấu trúc cho phép chúng ta định nghĩa một đối tượng có nhiều thuộc tính.

```
#include <iostream>
using namespace std;

struct bithday
{
   int d;
   int m;
   int y;
}
struct info
{
   int ID;
   birthday b;
};

int main()
{
   info a = {7812, 1, 1, 2002};
   cout << “ID = ” << a.ID << “\n dd/mm/yy = ”
<< a.b.d << “/” << a.b.m << “/” << a.b.y;
   return 0;
}
```

```
const int MAX = 50;
struct OnTap{
     char x[MAX], y[MAX]
     int z;
};
typedef struct OnTap ONTAP;

int main()
{
     ONTAP data = {y = {‘B’, ’H’, ‘T’},
x = “KH&KTTT”, z = 2021};
     cout << data.x;
}
```

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 DIEM
{
    int x;
    int y;
};
```

Khai báo 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>;
```

Sử dụng 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;
};
struct DIEM diem1, diem2;
```

Ví dụ:

```
typedef struct diem
{
    int a;
    int b;
};
diem x1, x2;
```

Khởi tạo giá trị cho biến

<Tên biến> = {<Giá trị 1>, <Giá trị 2>,...};

Ví dụ:

```
struct Diem
{
      int x;
      int y;
};
Diem x1 = {31,28};
```

Truy xuất dữ liệu kiểu cấu trúc

Đặc điểm:

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

Cú pháp : <tên biến cấu trúc>.<tên thành phần>

Ví dụ :

```
struct Diem
{
        int a;
        int b;
} diem;
diem x1 = {10,20};
cout << x1.a << " " << x1.b;
```

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 DIEM
{
      int a;
      int b;
} x1 = {2, 3},  x2;

C1 : x1 = x2;
C2:  x1.a = x2.a;
        x2.b = x1.b / 3;
```

Cấu trúc phức tạp

Thành phần của cấu trúc là cấu trúc khác.

```
struct Point{
        int x;
        int y;
};
struct Triangle{
        struct Point canh1;
        struct Point canh2;
        struct Point canh3;
};
Triangle.canh1.x = 165;
Triangle.canh2.y = 215;
```

Cấu trúc phức tạp

Cấu trúc tự trỏ (đệ quy):

```
struct OnTap
{
	int Diem;
	OnTap *pNext;
};
struct Node
{
	int a;
	Node *b;
}
```

Truyền cấu trúc cho hàm

Giống như truyền dữ liệu cơ sở:

- Tham trị (không thay đổi sau khi kết thúc hàm)
- Tham chiếu
- Con trỏ

Ví dụ:

```
struct Diem{
    int x;
    int y;
};
void xuat1(int x, int y) { };
void xuat2(Diem x1) { };
void xuat3(Diem &x1) { };
void xuat4(Diem *x1) { };
```

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 về cấu trúc

Dòng lệnh nào cần thêm để in ra ID của number1 ?

Info -> ID

```
struct ThongTin
{
        int ID;
        char CapBac[20];
};
typedef struct ThongTin thongtin;

int main
{
     thongtin number1 = {712346, “Truong Phong”};
     thongtin *Info = &number1;
     cout<<____;
}
```