1. **#include <isotream> (kiểu như stdio.h trong C, cho phép sử dụng cout, cin,…)**

*cout <<” xxxxxx ”;*

*cout << biến << endl ; (xuống dòng: end line)*

*cin>>x; (scanf trong C)*

*cout<<”gia tri x= “<<x ⬄ printf” gia tri x= , %d “; (hình như thế☺)*

**tham trị và tham chiếu:** nếu hàm ko thay đổi giá trị thì ko cần truyền tham chiếu (& array)

**tạo projet:**

**1.thuviencuaban.cpp: gồm khai báo hàm và viết hàm.** *#include”thuviencuatoi.h”*

**2.thuviencuaban.h: gồm khai báo TÊN CỦA HÀM ( ko có viết code trong đó)**

*khai báo tất cả các thư viện <..> (trừ thư viện “thuviencuaban.h****”***

**3.main.cpp: chứa hàm main:**

#include”thuviencuatoi.h”

**Trong struct: neu ho và tên ko nhập dc, thì trc khi nhập họ và tên**

fflush(stdin);

cout<<”nhap hoc ten, gets(sv.HoTen)

**vector:**

**include <vector>: bản chất là mảng 1 chiều**

điểm yếu: tốn bộ nhớ

mảng: bộ nhớ tĩnh ( vd định nghĩa a[10] ={1,2} thì nó sẽ chứa tối đa 10 phần tử, nhập 2 phần từ sẽ dư ra bộ nhớ =>> lãng phí)

con trỏ : cấp phát bộ nhớ động ( new int…) và thu hồi bộ nhớ động (delete p)

vector: hỗ trợ thao tác mảng động dễ dàng mà ko cần phải cấp phát và thu hồi vùng nhớ ( nó tự động á)

vector <kiểu dữ liệu> tên danh sách bạn cần

vd vector <int> arr={1,2,3};

========= ƯU ĐIỂM =========

1. Vector bản chất là 1 mảng 1 chiều - cụ thể nó là 1 cái mảng động - nó là 1 cái siêu mảng động

+ Vector có thể chứa các kiểu dữ liệu cơ sở(**int**, **float**, **double**, **char**...) và kiểu dữ liệu **do** người lập trình định nghĩa - **struct**...

TẠI SAO NÓ LÀ SIÊU MẢNG ĐỘNG

+ Vector có hết tất cả các tính chất mà mảng 1 chiều(mảng tĩnh và mảng động) và có thêm 1 số tính chất riêng của nó nữa(khắc phục nhược điểm của mảng tĩnh và mảng động)

+ Nếu như chúng ra dùng mảng tĩnh hay mảng động thì KHAI BÁO KÍCH THƯỚC CỦA MẢNG CẦN THAO TÁC TRƯỚC. Còn vector không cần khai báo trước số lượng phần tử cần dùng - bởi vì nó được hỗ trợ cái cơ chế tự động thêm phần tử vào cuối (push\_back())

+ Tự động đi giải phóng vùng nhớ khi chương trình kết thúc

+ Các thao tác thêm, xóa , tìm kiếm, sắp xếp bên mảng 1 chiều (mảng tĩnh hay mảng động) thì chúng ta phải đi cài - đặt - còn bên vector nó sẽ hỗ trợ chúng ta các hàm xử lí tương tự

========= KHUYẾT ĐIỂM =========

- Tốn bộ nhớ hơn mảng tĩnh và mảng động(**do** cơ chế container buộc phải tạo ra thêm các vùng nhớ riêng để quản lí chặt chẽ hơn)

Vector, List, Stack, Queue...: thuộc thư viện chuẩn của C++ - STL(Standard Template Library)

#include<vector>

===================== Một số hàm hỗ trợ trên Vector =====================

1/ resize:

- Hàm này có 2 dạng:

- resize(<số lượng>): Cấp phát mảng với số lượng phần tử cụ thể cho trước.

+ Mặc định khi cấp phát với resize là các phần tử đều mang giá trị mặc định ban đầu là 0(giống hàm calloc bên C)

+ Nếu mảng chưa từng được tạo thì sẽ tạo mảng với số lượng cụ thể đó.

+ Nếu mảng đã được tạo mà tiếp tục resize thì kích thước của mảng sẽ co lại hay dãn ra so với mảng ban đầu.

+ Nếu là co lại thì 1 số phần tử trong mảng ban đầu sẽ bị mất, nếu là dãn ra thì các phần tử lúc đầu vẫn tiếp tục tồn tại

- resize(<số lượng>, <giá trị mặc định>):

+ Cấp phát mảng với số lượng phần tử cụ thể cho trước - với giá trị của các phần tử là giá trị mặc định.

+ Mảng được tạo mới nếu chưa có, nếu đã có rồi mảng sẽ co lại hoặc dãn ra so với mảng ban đầu.

2/ at(<chỉ số index>): Truy xuất đến phần tử có chỉ số index trong vector.

vd:

vector a;

cout << a.at(2); *// Xuất ra phần tử tại vị trí 2 trong vector. Tương đương như: a[2]. Dùng cách nào cũng được.*

3/ size(): Trả về kích thước hiện tại của vector (số lượng các phần tử đang chứa trong vector).

4/ push\_back(<Phần tử có kiểu dữ liệu bất kỳ>): Thực hiện đưa phần tử vào cuối vector.

5/ pop\_back(): Xóa phần tử cuối cùng trong vector.

6/ front(): Trả về phần tử đầu tiên của mảng.

7/ back(): Trả về phần tử cuối cùng của mảng.

8/ capacity(): Trả về kích thước thực sự vector đang lưu trữ (các phần tử).

9/ max\_size(): Trả về kích thước tối đa có thể chứa các phần tử của các container.

10/ begin(): Trả về 1 iterator trỏ đến phần tử đầu tiên trong vector (giống pHead bên List)

11/ end(): Trả về 1 iterator trỏ đến phần tử cuối cùng trong vector (giống pTail bên List)

12/ erase: Xóa phần tử trong vector.

- Hàm này có 2 dạng:

+ Dạng 1: erase(<Tên vector>.begin() + <số nguyên x>): Xóa phần tử tại vị trí x trong mảng (vị trí trong mảng bắt đầu từ 0).

+ Dạng 2: erase(<Tên vector>.begin() + <số nguyên x>, <Tên vector>.begin() + <số nguyên y>): Xóa các phần tử của mảng từ vị trí x cho đến vị trí y - 1.

Lưu ý cả 2 trường hợp trên: Nếu vị trí truyền vào không hợp lệ thì chương trình sẽ bị lỗi.

13/ clear(): Xóa tất cả các phần tử hiện có trong vector, vector sau khi thực hiện câu lệnh này sẽ trở thành vector rỗng.

14/ insert: Thêm phần tử vào trong vector.

- Hàm này có 4 dạng:

+ Dạng 1: insert(<Tên vector>.begin() + <số nguyên x>, <giá trị cần chèn>): Chèn giá trị vào vị trí x trong mảng.

+ Dạng 2: insert(<Tên vector>.begin() + <số nguyên x>, <số lần chèn y>, <giá trị cần chèn>): Chèn y lần giá trị vào vị trí x trong mảng.

+ Dạng 3: insert(<Tên vector>.begin() + <số nguyên x>, <Tên vector a>.begin() + <số nguyên y>, <Tên vector a>.begin() + <số nguyên z>):

---> Lấy các phần tử từ vector a bắt đầu lấy từ vị trí y cho đến vị trí z - 1 và chèn vào vị trí x của vector hiện tại.

+ Dạng 4: insert(<Tên vector>.begin() + <số nguyên x>, <Tên mảng a> + <số nguyên y>, <Tên mảng a> + <số nguyên z>):

----> Lấy các phần tử lấy từ mảng a bắt đầu lấy từ vị trí y đến vị trí z - 1 và chèn vào vị trí x của vector hiện tại.

15/ swap(<vector x>): Hoán đổi các phần tử của vector hiện tại & vector x cho nhau.

1. **Các phép toán:**

**Toán tử gán:** Variablename = expression;

Tên biến = giá trị;

Vd: int a = 2;

⦁ c+ = a ⬄ c = c + a (tương tự cho phép -, \*, /, %)

⦁ a % b : chia lấy phần dư ( 8 % 5 = 3)

A / b : chia lấy phần nguyên ( 8/5 =1)

❶ i++ và ++i

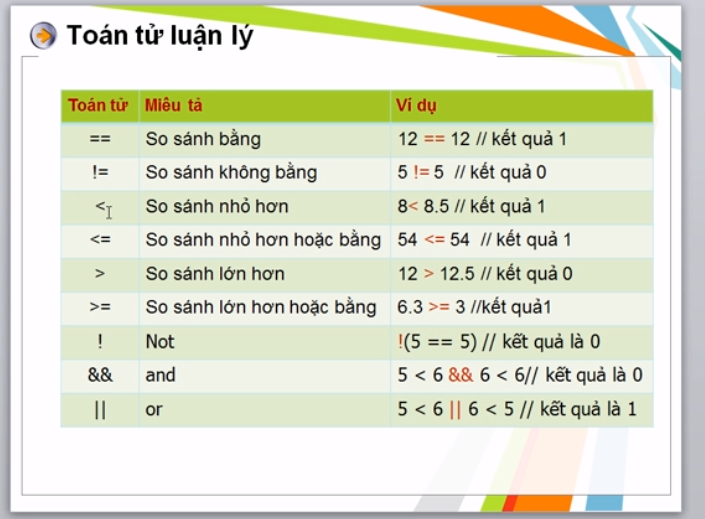
Vd: int i = 10;

a = i++; (1)

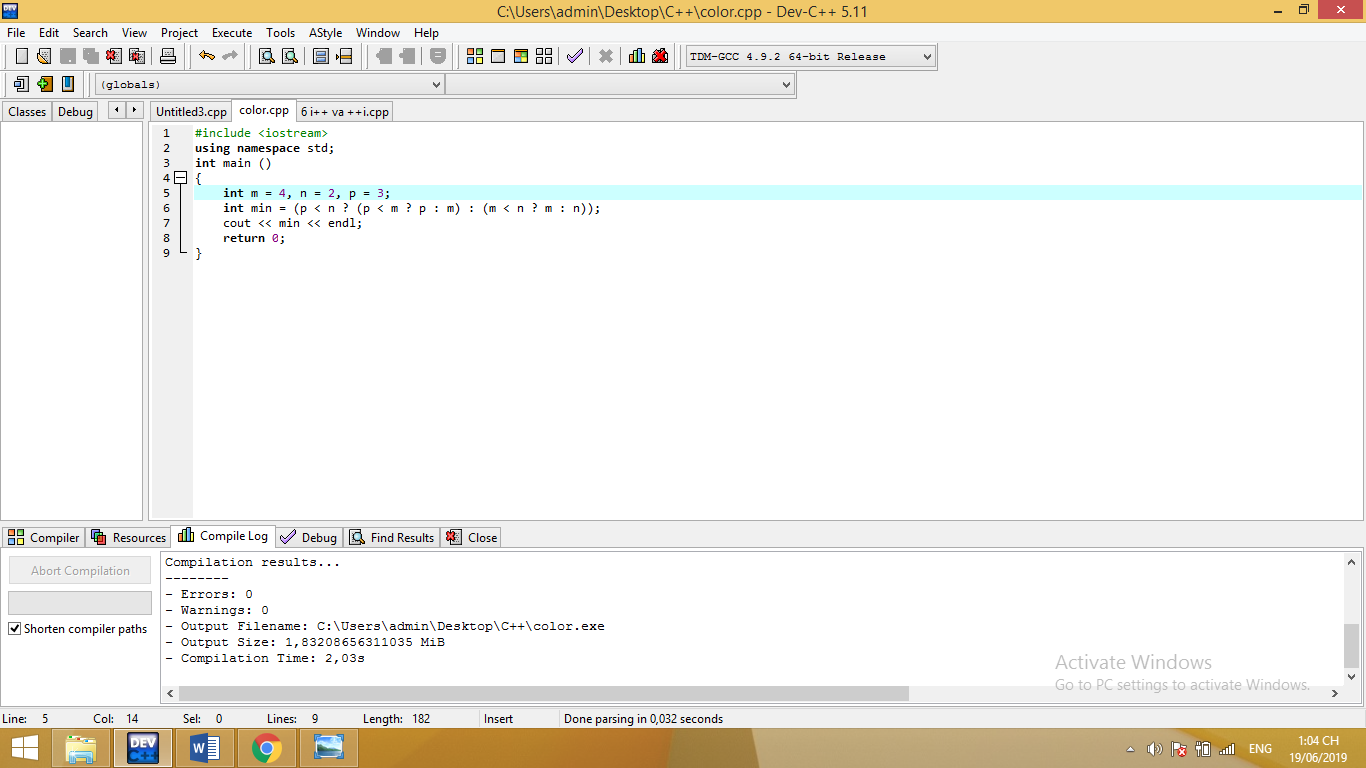
b = ++I; (2)

1. i++ : ( + sau): gán a = 10 trước rồi tăng i lên 1 đơn vị => a = 10, i = 11. ( i > a)
2. ++i : (+ trước): tăng i lên 1 đơn vị trước rồi gán giá trị đó vào b => i =11, b =11 ( i = b)

❷ phép logic:



Thứ tự: ! 🡪 >, >=, <, <= 🡪 ==, != 🡪 && 🡪 ||

❸ Toán tử ? ( toán tử 3 ngôi )

=>> kết quả: min = 2.

**II/ IF…..else:**

If (điều kiện) // if (dieu kien) {

Code }

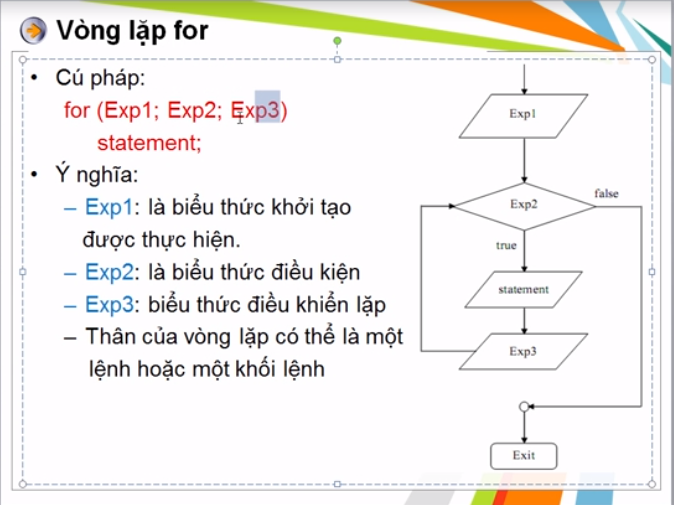
Code…….

Else if (điều kiện 2)

Code…..

Else ……..(code)

1. Switch…case
2. Vòng lặp:



for (exp1; exp2; exp3)

…………………………

Exp1: là định nghĩa biến / vd: int i = 1;

Exp2: điều kiện/ vd: i<=n;

Exp3: điều kiện vòng lặp ( có thể là 1 lệnh ou khối lệnh)/ vd i++

Bất kì exp naò cũng có thể trống, khi cả 3 đều để trống: for(;;;) => lặp vô tận

**While (exp…){**

**Statement………..;}**

**(dùng while ⬄ if dc lặp lại n lần)**

**.for: dung khi biết số vòng lặp trc**

**.while, do..while: ko biết trc số vòng lặp**

**.do..while: lặp lại ít nhất 1 lần ( do nó làm trước rồi mới kiểm tra exp….)**

**break :**

**chỉ dung trong vòng lặp, nếu gặp sẽ thoát khỏi lệnh đó☺**

**vd: for(int i=0, i<=n, i++) {**

**lệnh 1;**

**break; => thoát luôn.**

**Lệnh 2;**

**}**

**Continue : giống như break, nhưng dung để kết thúc vòng lặp hiện tại và bắt đầu vòng lặp tiếp theo… (cái này là lý thuyế, méo biết đúng hay sai☺)**

**Continue thường dc đi kèm với lệnh if**

**Vd:**

**For( ….;…..;…)**

**lệnh 1;**

**Continue; ( nó sẽ bỏ qua lệnh 1 để xét lệnh 2)**

**Lệnh 2;…**

**Vòng lặp lồng nhaoooooo**

**For( exp1; exp2; exp3) {**

**For (exp 3; exp4; exp5){**

**Statement…}**

**statement}**

**Vd:**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**for (int n =0;n<3;n++){ // tổng cộng sẽ in 3 lần (4 dòng hello va bonjour**

**for (int m = 0;m<=3;m++){ // hello 123 sẽ lặp 4 lần ( m <=3)**

**cout<<"hello 123"<<endl;} // hết vòng for 2**

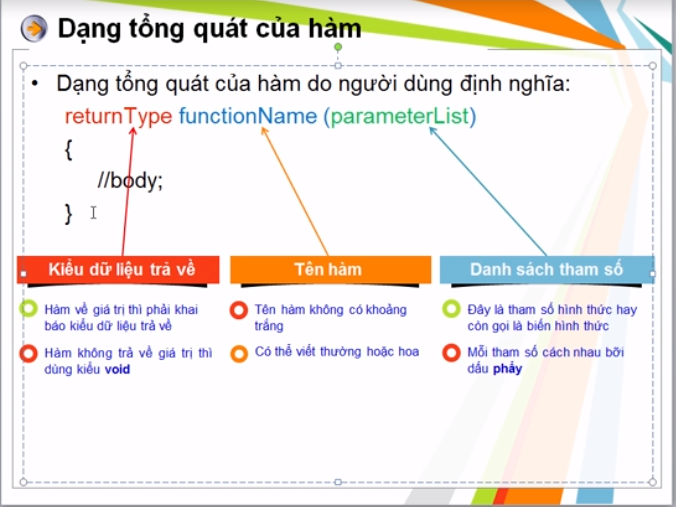
**cout <<"bonjour"<<endl;} // in ra như bình thường ( nó làm lệnh này )**

**return 0;**

**}**

**While lồng nhau, do…while lồng nhau: coi vd**

**Hàm**



Khi một hàm muốn trả về giá trị nào đó, ta dung từ return. ( cho hàm int ….)

Void main thì ko dc dung return!!

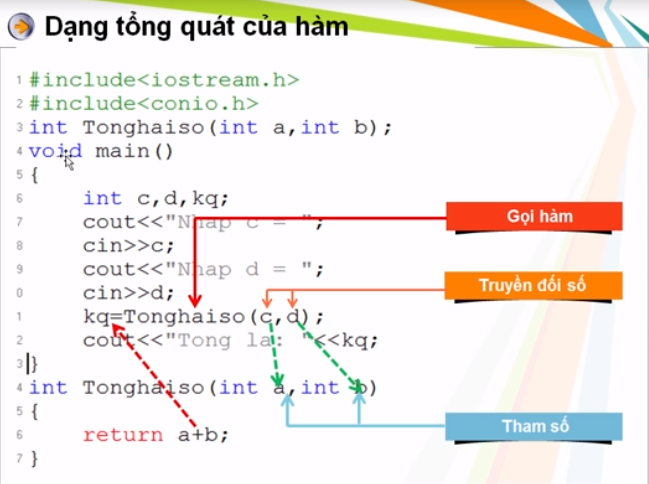
=>> muốn hàm trả về giá trị nào đó: dung int ….., kết thúc là return.

Hàm ko trả giá trị: dung void…., kết thúc ko có return!

Cú pháp: **Tên hàm ( danh sách các tham số)**

**Vd int cong2so (int a, int b){……..**

**}**

****

**Khai báo hàm:**

**Vd int cong2so (int a , int b)**

**(int &a, int &b)**

**(int , int)**

**Định nghĩa hàm: nếu chưa khai báo hàm mà trong hàm main muốn gọi hàm đó thì phải định nghĩa hàm trc hàm main.**

**Nếu khai báo hàm rồi thì có thể định nghĩa hàm ở đâu cũng dc, (cứ để ở cuối cho lành☺)**

**Gán tham số: tong2so (int a = 2, int b = 1)**

**(int a , int b = 1)**

**(int a = 2, int b) ⬄ lỗi => có thể gán giá trị cho 2 biến, biến sau cùng, ko dc gán cho biến đầu tiên ( int a)**

**Đệ quy:**

Hàm đệ quy là hàm mà trong thân hàm có gọi chính nó( hàm a, trong hàm a gọi lại hàm a☺)), có thể gọi tường minh hoặc ko tường minh.

Điều kiện:

Điều kiện dừng: là phần kết thúc thuật toán, ko tính phần đang dc định nghĩa

Tinh theo nguyên tắc thế ngược ( gọi hàm đệ quy, chạy từ đầu cho tới điều kiện dừng rồi thế giá trị vào, từ dk dừng trở lên lại😊, na ná stack

Vd: tính tổng s(n) = 1+2+…+n

Int Sum(int n)

{

If( n == 1 ) return 1; // dk dừng

Return n + Tong(n-1);

}

// truyền n = 5: xét điều kiện: 5 != 1

Chạy ctrinh

B1: 5+ Tong(5-1)

B2: 4 + Tong(4-1)

……

B n-1 : 2 + Tong(1)

B n Tong (1) = 1 // dieu kien dung

Tính toán( sau khi chạy ctrinh) (làm theo thứ tự ngược lại với chạy ctrinh)

B1: Tong(1) = 1 //B n của chạy ctrinh

B2: 2+1 = 3

….

B n-1 : 4 + 6 = 10

B n: 5+10 = 15

**Đệ quy nhị phân**

* **Ví dụ: Tính số hạng thứ n của dãy Fibonaci được định nghĩa như sau:**
  + **f1 = f0 =1 ;**

**fn = fn-1 + fn-2 ; (n>1)**

**dk dừng: f(0) = 0, f(1) = 1**

int F(int n){

if( n == 0) return 0;

if( n == 1) return 1;

return F(n-1)+ F(n-2); }

// n = 5:

Chạy ctrinh

1. F(5) = F(5-1) + F(5-2)
2. F(4) = F(4-1) + F(4-2)
3. F(3) = F(3-1) + F(3-2)
4. F(2) = F(2-1) + F(2-2) (dk dừng 1 và 0)

Tính toán:

1. 1+0 = 1 ( F1)
2. 1 + 1 = 2 (F2)
3. 2+1 = 3 (F3)
4. 3+2 = 5 (F4)
5. 5 + 3 = 8 (F5)

**Đệ quy hỗ tương: trong hàm x gọi hàm y và hàm y gọi lại hàm x**

**Vd: tìm số hạng thứ n của 2 dãy sau:**

**X(0) = 1;**

**Y(0) = 0;**

**X(n) = x(n-1) + y(n-1) ( n>0)**

**Y(n) = 3\*x(n-1) + 2\*y(n-1) ( n>0)**

**int SoHangCuaDayY(int n) ; ( khai báo hàm trc do có gọi đệ quy tại SoHangCuaDayX)**

*Vd: n = 3:*

*SoHangCuaDayX: X,*

*SoHangCuaDayY : Y,*

**int SoHangCuaDayX(int n){**

**if(n == 0) return 1 ;**

**return SoHangCuaDayX(n-1) + SoHangCuaDayY(n-1) ; }**

1. *X(3) = X(3-1) + Y(3-1) ;*
2. *X(2) = X(2-1) + Y(2-1) ;*
3. *X(1) = X(1-1) + Y(1-1) ;// X(0) = 1 : dk dừng :*

*1+0 = 1 ;*

**int SoHangCuaDayY(int n){**

**if( n==0) return 0 ;**

**return 3\*SoHangCuaDayX(n-1) + 2\* SoHangCuaDayY(n-1) }**

*tương tự X*

**//**

Mảng nhiều chiều:

chiều:

* Một mảng nhiều chiều là một mảng mà những phần tử của nó được xác định bằng nhiều chỉ số.
* Mảng 2 chiều là mảng nhiều chiều đơn giản và sử dụng nhiều nhất.
* Mảng 2 chiều: đó chính là mảng mà các phần tử của nó chính là mảng một chiều.

Mảng 2 chiều giống như một bảng, gồm nhiều dòng và nhiều cột



**m[2][3]**

**Chỉ số**

**cột**

**chỉ số**

**dòng**

Các phần tử được xác định

bằng số dòng và số cột.

Mảng 2 chiều:

Một nhóm các phần tử có

cùng kiểu, chung tên.

**i và j là số nguyên**

Mô hình mảng 2 chiều và liên hệ giữa mảng 1 chiều và 2 chiều: (// trong bài nhập xuất mảng 2 chiều = dạng khai báo mảng 1 chiều:

Với i là chỉ số dòng

j là chỉ số cột Array [ i \* MaxColomns + j]

tham khảo thêm tại link: <https://www.stdio.vn/articles/mang-1-chieu-mang-2-chieu-377>

**Khai báo mảng hai chiều:** giống mảng 1 chiều mà thêm phần số cột

***type arrayName[rows][columns];***

* + **rows**: số dòng
  + **columns**: số cột
* Ví dụ: Khai báo mảng số nguyên 3 dòng 4 cột

 **int a[3][4]**

**Rows: ko cần khai báo tường minh columns: phải khai báo tường minh**

Or: type arrayName [] [columns] **{** {*value 1, value 2,…value N*}**,** {*value 1, value 2,…value N*}**,** {…} **}**

Vd; int array[][5] = {{1,2,3}{3,4,5}{7,8,20,22}}: ma trận có 5 cột và 3 dòng.

Có thể khai báo mảng 2 chiều giống như mảng 1 chiều ( vì mảng 2 chiều có các phần tử là mảng 1 chiều).

Vd: int array[3][4] = {1,2,3,4,5,6,7,8}.

CHUỖI (String)

Là một mảng các ký tự, kết thúc = ký tự NULL (‘0’)

Ký tự NULL (‘0’) dc tự động thêm vào trong chuỗi. ( khi khai báo)

Mỗi ký tự trong chuỗi dc lưu = bảng mã ASCII

**NULL**

**‘o’**

**‘l’**

**‘l’**

**‘e’**

**‘H’**

**01001000**

**01100101**

**01101100**

**01101100**

**01101111**

**00000000**

**101, 65h**

**108, 6Ch**

**108, 6Ch**

**111, 6Fh**

**00000000**

**72, 48h**

**Khai báo chuỗi:**

**Cách 1:** dùng mảng 1 chiều

**char <Tên biến> [Chiều dài tối đa]**

Vd: char string [25].

* + Ý nghĩa khai báo một mảng kiểu ký tự tên là str có 25 phần tử (như vậy tối đa ta có thể nhập 24 ký tự vì phần tử thứ 25 đã chứa ký tự kết thúc chuỗi ‘\0’).
  + *Lưu ý:* Chuỗi ký tự được kết thúc bằng ký tự ‘\0’. Do đó **khi khai báo độ dài của chuỗi luôn luôn khai báo dư 1 phần tử để chứa ký tự ‘\0’**.

**Cách 2: dùng còn trỏ.**

**char \*<Tên biến>**

Vd: char \*string ( giống kiểu int a)

* + - Trong khai báo này, bộ nhớ sẽ dành 2 byte để lưu trữ địa chỉ của biến con trỏ str đang chỉ đến, chưa cung cấp nơi để lưu trữ dữ liệu.

***Trước khi sử dụng phải dùng từ khóa new để cấp phát vùng nhớ.***

***Vd: char \*string;***

***Str = new char[51];*** *// cấp phát cho con trỏ này 51 ký tự*

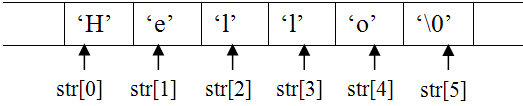
Khởi tạo 1 chuỗi ký tự xác định, ta làm giống như mảng:

**char <Biến>[ ]=<“Hằng Chuỗi”>**

Vd: char str[] = { ‘H’ , ‘E’, ‘L’, ‘L’, ‘O’ }

char str[] = “HELLO”;

char \*str = “HELLO”



Nhập chuỗi: dùng gets( chứa trong #include<stdio.h>)

**char \*gets(char \*s);**

Vd: gets str;

* Hàm gets() đọc các ký tự từ bàn phím vào trong mảng trỏ đến bởi s **cho đến khi nhấn** **Ente**r. *Ký tự null sẽ được đặt sau ký tự cuối cùng của Chuỗi nhập vào trong mảng*. Vd: nhâp abc, r bấm enter: máy sẽ hiểu là chuỗi là abc, còn null là ở cuối cùng( sau c)
* Lưu ý: Khi dùng **cin>>** để nhập dữ liệu cho chuỗi, chương **trình *sẽ tự động ngắt chuỗi khi gặp ký tự khoảng trắng* trong chuỗi**. Do đó, để chuỗi không bị ngắt khi gặp ký tự khoảng trắng, ta sẽ dùng hàm *gets(), hoặc cin.getline()* thay vì hàm cin thông thường.

**cin.getline**(chuỗi, số ký tự tối đa);

\*Ví dụ:

char \*str;

str = new char [30];

cin.getline(str, 30); = gets str;

Xuất chuỗi:

Để xuất chuỗi ra màn hình, ta dùng puts() trong thư viện <stdio.h>

**int puts(const char \**s*);**

Hoặc dùng cout**: cout<<str; = puts(str);**

**Các hàm thao tác trên chuỗi:**

**Khai báo thư viện <string.h> #include<string.h>**

**Hàm strcpy(string coppy): strcpy(char \*đích, char \*nguồn);**

Tác dụng: sao chép nội dung chuỗi nguồn vào chuỗi đích, nội dung chuỗi đích sẽ bị xóa.

#include <iostream.h>

#include <stdio.h>

#include <string.h> // str1 là chuỗi nguồn

void main() // str2 là chuỗi

{

char str1[20], str2[20];

cout<<“Nhap chuoi 1:"; gets(str1);

**strcpy(str2,str1);**

cout<<"\nXuat chuoi 2:"; puts(str2);

}

=>> nó sẽ xóa chuỗi trong old str1, lưu new str1 = chuỗi của str2

Vd: str1: I love u

Str2: je t’aime

Khi dùng strcpy(str2, str1): sẽ xuất ra màn hình: je t’aime

Char str[20]

Muốn xóa 1 hay n ký tự đầu tiên: strcpy(str,str+n) ( xóa 1 ký tự thì n = 1)

………………………………từ cuối chuỗi lên đầu chuỗi( từ phải qua trái)” strcpy(str+l-1, str) (l là dộ dài chuỗi nhập vào) ( -1 là do NULL)

**STRNCPY: ( string n coppy)**

* Chép n ký tự từ chuỗi nguồn sang chuỗi đích. Nếu chiều dài nguồn < n thì hàm sẽ điền khoảng trắng cho đủ n ký tự vào đích.

strncpy(char \*đích, char \*nguồn, int n);

#include <iostream.h>

#include <stdio.h>

#include <string.h>

void main()

{

char str1[20], str2[20];

cout<<“Nhap chuoi 1:"; gets(str1);

cout<<"\n Nhap chuoi 2: ", gets(str2);

**strncpy(str2,str1,5***); // coppy 5 ký tự đầu tiên của chuỗi str1 vào chuỗi str2*

cout<<"\nXuat chuoi 2:"; puts(str2);

}

Vd: Input: str1: I love u

str2: je t’aime

=>> output: str2: I lovaime

=>> strncpy : coppy 5 ký tự đầu của str1 và dán đề lên 5 ký tự đầu của str2. Nếu chuỗi str2 ko có hoặc rỗng thì nó sẽ coppy 5 ký tự đầu của str1 vào, khi đó cout của str2 sẽ là 5 ký tự đầu của str1

**Strcat**

* Nối chuỗi s2 vào cuối chuỗi s1

strcat(s1, s2)

strcat(char s1[],char s2[]);

vd:

#include <iostream>

#include<stdio.h>

#include<string.h>

#define MAX 50

using namespace std;

main(){

char str1[MAX], str2[MAX];

cout<<"Nhap chuoi 1: ", gets(str1);

cout<<"\n Nhap chuoi 2: ", gets(str2);

strcat(str1, str2); // nối chuỗi 2 vào chuỗi 1

cout<<"\n\n\n\nChuoi 1 bay h la: "<<str1<<endl;

cout<<"Chuoi 2 bay h la: "<<str2;

}

Input: str1 i love u

str2 je t’aime

output: str1 i love uje t’aime

str2 je t’aime

**strncat**

* Nối n ký tự đầu tiên của chuỗi s2 vào cuối chuỗi s1

strncat(char s1[],char s2[],int n);

#include <iostream>

#include<stdio.h>

#include<string.h>

#define MAX 50

using namespace std;

main(){

char str1[MAX], str2[MAX];

cout<<"Nhap chuoi 1: ", gets(str1);

cout<<"\n Nhap chuoi 2: ", gets(str2);

strncat(str1,str2,5); // *gán 5 ký tự đầu tiên của chuỗi str2 GHÉP vào đầu của chuỗi str1*

cout<<"\n\n\nstrncat(str1,str2,5)";

cout<<"\nchuoi 1 tro thanh: "<<str1<<endl;

cout<<"chuoi 2 : "<<str2;

}

**strlen**

* Tính độ dài chuỗi s

strlen(char \*s);

trong thư viện **<ctype.h>**

* So sánh 2 chuỗi s1 và s2 theo nguyên tắc thứ tự từ điển. Phân biệt chữ hoa và thường.Trả về :
  + 0 : nếu s1 bằng s2.
  + =1: nếu s1 lớn hơn s2.
  + =-1: nếu s1 nhỏ hơn s2.

**strcmp(char s1[],char s2[]);**

* + So sánh n ký tự đầu tiên của s1 và s2, giá trị trả về tương tự hàm strcmp()

**strncmp(char s1[],char s2[], int n);**

* + So sánh chuỗi s1 và s2 nhưng không phân biệt hoa thường, giá trị trả về tương tự hàm strcmp()

**stricmp(char s1[],char s2[]);**

* So sánh n ký tự đầu tiên của s1 và s2 nhưng không phân biệt hoa thường, giá trị trả về tương tự hàm strcmp()

**strnicmp(char s1[],char s2[], int n);**

* Chuyển ký tự thường sang ký tự hoa

**toupper( int ch );**

* Chuyển ký tự hoa sang ký tự thường

**tolower( int ch );**

* + Hàm **islower(s[i])**: kiểm tra xem s[i] có phải là ký tự thường hay không
  + Hàm **isupper(s[i]):** kiểm tra xem s[i] có phải là ký tự hoa hay không
  + Hàm **isalpha(s[i]):** kiểm tra xem s[i] có phải là ký tự hay không

Khai báo chuỗi:

Dùng mảng:

* Mảng các Chuỗi là một mảng ký tự hai chiều. Kích thước của chỉ số thứ nhất là số chuỗi và kích thước của chỉ số thứ hai xác định chiều dài lớn nhất của mỗi chuỗi.
* Ví dụ: **char str[5][80];**

Khai báo và khởi tạo mảng các Chuỗi

char arrayList[][length] = { constantString1,

constantString2,

...

constantStringN};

Ví dụ:

char arrayList[][10] = {“AAAA”, “BBB”, “CC”};

Ý nghĩa: mảng str chứa 5 Chuỗi, mỗi Chuỗi có chiều dài tối đa là 79 ký tự ( ký tự cuối là NULL)

Dùng con trỏ

* ta có thể dùng mảng của các con trỏ. Mỗi con trỏ sẽ chứa địa chỉ của Chuỗi
* Ví dụ:

*char \*str[20];*

* + Mảng con trỏ đến các chuỗi

char \*listArr[] = {“Pascal”, “C/C++”, “CSharp”, “Java”};

ý nghĩa: con trỏ listArr có 4 (hoặc 5 – nhớ test lại nha vì cái này t ko biết chắc) con trỏ, mỗi con trỏ sẽ trỏ vào 1 chuỗi

**CON TRỎ**

Địa chỉ biến:

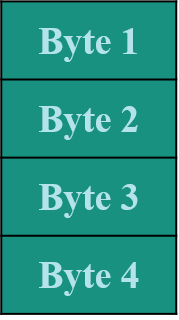
* Thông tin của một biến bao gồm:
  + Tên biến (vd a)
  + Kiểu dữ liệu của biến ( v dint)
  + Giá trị của biến ( vd 10)

(int a = 10; )

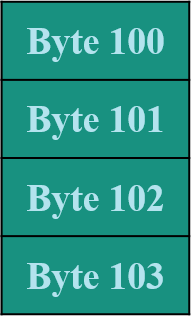
* Mỗi biến sẽ được lưu trữ tại một vị trí xác định trong ô nhớ, nếu kích thước của biến có nhiều byte thì máy tính sẽ cấp phát một dãy các byte liên tiếp nhau, địa chỉ của biến sẽ lưu byte đầu tiên trong dãy các byte này

//Khi khai báo 1 biến, ctrinh sẽ dành ra bộ nhớ. Giả sử ô nhớ như dãy nhà phố dài giống như đặt địa chỉ cho căn nhà ( ở tầng trệt ⬄ byte đầu tiên lưu địa chỉ) , các căn nhà tiếp theo là các biến khác, nhà 4 tầng ⬄ int, nhà 2 tầng ⬄ char,….

**Địa chỉ biến x địa chỉ biến a**



Các ô nhớ của biến x



Các ô nhớ của biến a

* Địa chỉ của biến luôn luôn là một **số nguyên (hệ thập lục phân)** dù biến đó chứa giá trị là số nguyên, số thực hay ký tự, …
* Cách lấy địa chỉ của biến :*&tênbiến*
* Ví dụ:

int x=7;

cout<<"Dia chi cua bien x = "<<&x<<endl;

=>> 0x23fe3c ( đây là địa chỉ của x)

* Một con trỏ là 1 **biến chứa một địa chỉ bộ nhớ**. *Địa chỉ này là vị trí* của một đối tượng khác trong bộ nhớ.
* Nếu một biến chứa địa chỉ của một biến khác, biến thứ nhất được gọi là trỏ đến biến thứ hai.

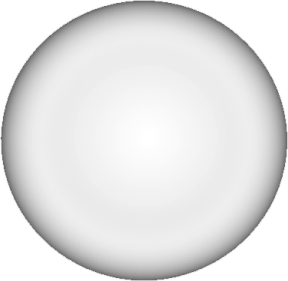
**10**

**Giá trị biến a = 10**

**aPtr**

**a**

**Con trỏ trỏ đến vùng nhớ của biến a**



// biến aPtr trỏ tới địa chỉ của a

Khai báo con trỏ: **type \*pointerVariable;**

//Type là kiểu dữ liệu của biến mà con trỏ trỏ đến ( char, int, float, double,…..)

int **\***a;

**a**

(….địa chỉ biến dc trỏ…)

Toán tử con trỏ:

* **Toán tử &** là toán tử 1 ngôi, trả về địa chỉ bộ nhớ của toán hạng của nó.
* **Toán tử &** dùng để gán địa chỉ của biến cho biến con trỏ
* Cú pháp:

**<Tên biến con trỏ>=&<Tên biến>**

* **Vd: Ví dụ:**

int a = 25, x; // gán giá trị a = 25, tạo 1 biến x;

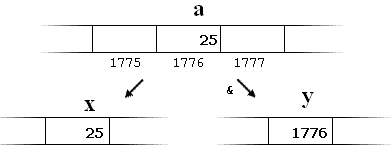
int \*y; // tạo một con trỏ y;

x = a; // gán giá trị x = giá trị của a ( x = 25)

y=&a; // gán con trỏ y = địa chỉ của a, khi này con trỏ y = địa chỉ a)

*y=a;//sai // con trỏ y là biến chứa địa chỉ nên ko thể gán với biến mang giá trị ( nước sông ko phạm nước giếng ☺*

*vd hình ảnh : a có giá trị bàng 25, dc lưu tại địa chỉ 1776 ; khi x = a và y = &a, ta có hình minh họa*



* Toán tử \* là toán tử một ngôi trả về giá trị tại địa chỉ con trỏ trỏ đến.
* Cú pháp:

*\* <Tên biến con trỏ>*

**Ví dụ:**

**z=\*y ;** *// gán giá trị của a = giá trị của con trỏ p trỏ đến. Vd: con trỏ y trỏ đến giá trị a ở bài trên, với a = 25. Địa chỉ của a = 1776 ⬄ con trỏ y = 1776, khi gán z = \*y thì z =\*y= 25 ( tính chất bắc cầu, con trỏ y là cái cầu☺)*

**a=p;//sai**

***CÁC THAO TÁC TRÊN CON TRỎ:***

Gán con trỏ

Có thể dùng phép gán để gán giá trị của một con trỏ cho một con trỏ khác có cùng kiểu

**Ví dụ:**

int x=10;

int \*p1, \*p2;

p1 = &x;

p2 = p1;

Sau khi đọan lệnh trên được thực hiện, cả hai p1 và p2 cùng trỏ đến biến x.

**PHÉP TOÁN SỐ HỌC CON TRỎ:**

Chỉ có 2 phép toán số học là phép + và –

Khi cộng hoặc trừ 1 con trỏ với 1 sốp nguyên n, kết quả trả về là 1 CON TRỎ. Con trỏ mới này chỉ đến vùng nhớ mới, cách vùng nhớ cũ 1 số nguyên lần kích thước của KIỂU DỮ LIỆU của nó.

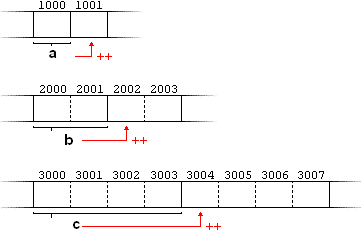
***Ví dụ :***

short \*b;  
long \*c;

Các con trỏ b, c lần lượt trỏ tới ô nhớ **2000** và **3000**.

Cộng các con trỏ với một số nguyên:

b = b + 1;//con trỏ b dời đi 2 byte   
c = c + 1; //con trỏ c dời đi 4 byte



Lưu ý: cả hai toán tử tăng (++) và giảm (--) đều có quyền ưu tiên lớn hơn toán tử \*

* Ví dụ: \*p++;

Lệnh \*p++ tương đương với \*(p++) : thực hiện là tăng p (**địa chỉ ô nhớ mà nó trỏ tới** chứ không phải là giá trị trỏ tới).

**CẤP PHÁT ĐỘNG**

* Các chương trình trước đây kích cỡ vùng nhớ khai báo là cố định và không thể thay đổi trong thời gian chương trình chạy.
* Tuy nhiên chúng ta cần một lượng bộ nhớ mà kích cỡ của nó chỉ có thể được xác định khi chương trình chạy, ví dụ như trong trường hợp chúng ta nhận thông tin từ người dùng để xác định lượng bộ nhớ cần thiết.???
* Giải pháp ở đây chính là *bộ nhớ động*
* C/C++ hỗ trợ hai hệ thống cấp phát động: một hệ thống được định nghĩa bởi C và một được định nghĩa bởi C++.

**Cấp phát động kiểu C**: coi pp bài giảng

Cấp phát động kiểu C++:

* C++ cung cấp hai toán tử cấp phát bộ nhớ động: new và delete.
  + Toán tử new cấp phát bộ nhớ và trả về một con trỏ đến byte đầu tiên của vùng nhớ được cấp phát.
  + Toán tử delete thu hồi vùng nhớ được cấp phát trước đó bởi toán tử **new**.
* Cú pháp:

p = new type; vd: int \*p = new int

delete p; vd: delete p;

* + p là một biến con trỏ nhận địa chỉ của vùng nhớ được cấp phát đủ lớn để chứa 1 đối tượng có kiểu là type
* Con trỏ void là một lọai con trỏ đặc biệt mà có thể trỏ đến bất kỳ kiểu dữ liệu nào. ( con trỏ phò☺))
* Cú pháp:

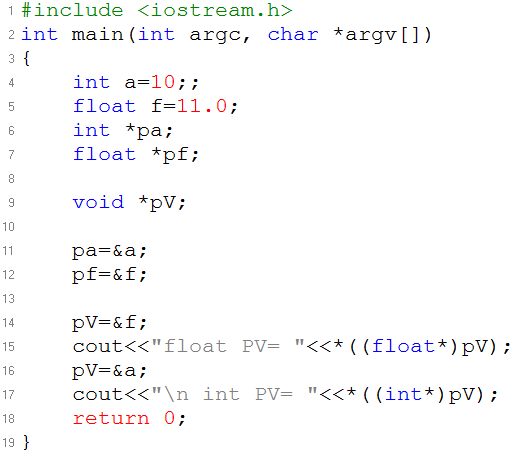
void \*pointerVariable;

* Ví dụ:

void \*p;

p = &a; // p trỏ đến biến nguyên a

p = &f; //p trỏ đến biến thực f



* Tuy nhiên, ta cũng có thể ép kiểu con trỏ về đúng kiểu tương ứng khi dùng trong các biểu thức.
* Ví dụ:
  + Nếu p đang trỏ đến biến nguyên a, để tăng giá trị của biến a lên 10 ta phải dùng lệnh sau:

(int\*)\*p + 10;

* + Nếu p đang trỏ đến biến thực f, để tăng giá trị của biến f lên 10 ta phải dùng lệnh sau:

(float\*)\*p + 10;

Con trỏ NULL

* Một con trỏ hiện hành không trỏ đến một địa chỉ bộ nhớ hợp lệ thì được gán giá trị NULL
* NULL được định nghĩa trong <cstdlib>
* Ví dụ:

#include <iostream.h>

void main()

{

int \*p;  
cout <<“Gia tri con tro p tro den la: “<< \*p;

}

* Kết quả của chương trình trên là:

NULL POINTER ASSIGNMENT

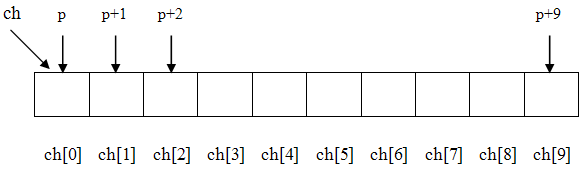
**CON TRỎ VÀ MẢNG**

* **Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ:**
  + Những phần tử của mảng được xác định bằng chỉ số trong mảng và cũng có thể được xác định qua biến con trỏ.
  + Tên của một mảng tương đương với địa chỉ phần tử đầu tiên của nó, tương tự một con trỏ tương đương với địa chỉ của phần tử đầu tiên mà nó trỏ tới.

Vd:

Char ch[10], \*p;

p = ch;



* p được gán địa chỉ của phần tử đầu tiên của mảng ch.

p = ch;

tham chiếu phần tử thứ 3 trong mảng ch, ta dùng một trong 2 cách sau:

ch[2] hoặc \*(p+2)

* Truy cập các phần tử mảng bằng con trỏ

|  |  |
| --- | --- |
| ***Kiểu mảng*** | ***Kiểu con trỏ*** |
| **&<Tên mảng>[0]** | **<Tên con trỏ >** |
| **&<Tên mảng> [<Vị trí>]** | **<Tên con trỏ> + <Vị trí>** |
| **<Tên mảng>[<Vị trí>]** | **\*(< Tên con trỏ > + <Vị trí>)** |

Vd: mảng arr[5] = { 10, 20, 30, 40, 50}

main()

{

int arr[5], \*p;

p = arr; // gán con trỏ p vào địa chỉ a[0]

\*p = 10; // gán a[0] = 10

p++, \*p = 20; // tại vị trí p+1 (⬄ a[1] gán \*p = a[1] = 20)

p = &arr[2], \*p = 30; // con trỏ p trỏ địa chỉ a[2] ⬄ a[2] = 30

p = arr + 3, \*p = 40; // phép toán cộng con trỏ: arr+3 ⬄ a[3]

p = arr; // gán p = địa chỉ mảng arr ( arr[0] )

\*(p+4) = 50; // tang lên 4 : (p+4) ⬄ p = arr +4

for(int i = 0; i<5; i++)

cout<<arr[i]<<" , ";

}

* Mỗi biến con trỏ là một biến đơn. Ta có thể tạo mảng của các con trỏ với mỗi phần tử của mảng là một con trỏ.
* Cú pháp:

**type \*pointerArray[elements];**

* + **type**: kiểu dữ liệu mà các con trỏ phần tử trỏ đến.
  + **pointerArray**: tên mảng con trỏ.
  + **elements**: số phần tử của mảng con trỏ.
  + **Vd**:

int \*p[5]; //tạo mảng gồm 5 con trỏ

int a=6;

p[0] = &a; p[2] = p[0]; // gán p[0] = địa chỉ biến a, p[2]=p[0]: <=>p[2]=&a

int b;

b = \*p[0]; // gán giá trị b = \*p[0]

**CẤU TRÚC DỮ LIỆU**

* Một cấu trúc là một tập các biến được tham chiếu thông qua một tên chung. Những biến tạo nên cấu trúc được gọi là các thành viên (**members**).
* Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng là **cùng kiểu** còn các phần tử của kiểu cấu trúc có thể **có kiểu khác nhau**.

***structureName***: Tên của cấu trúc

***type***: Kiểu dữ liệu của thành viên tương ứng

***Member1,..., memberN***: Tên các biến thành viên của cấu trúc

***varNames***: Tên các biến cấu trúc phân cách nhau bằng dấu phẩy.

**struct structureName**

**{**

**type member1;**

**type member2;**

**...**

**type memberN;**

**.. .**

**} [varNames];**

Cấu trúc thực chất là một kiểu dữ liệu do người dùng định nghĩa bằng cách gom nhóm các kiểu dữ liệu cơ bản có sẵn trong C/C++ thành một kiểu dữ liệu phức hợp nhiều thành phần.

* Ví dụ:

struct Sach

{

int MaSach;

char NhanDe[100];

char TacGia[100];

char NhaXuatBan[100];

int NamXuatBan;

};

Khai báo biến kiểu Sách:

struct Sach s1,s2;

* Ví dụ:

struct Sach

{

int MaSach;

char NhanDe[100];

char TacGia[100];

char NhaXuatBan[100];

int NamXuatBan;

};

Khai báo biến kiểu Sách:

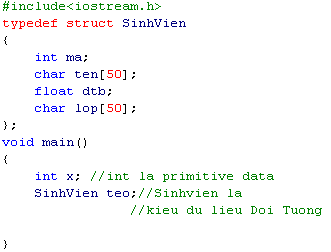
struct Sach s1,s2;

* Từ khóa **typedef** dùng để định nghĩa một tên mới cho một kiểu dữ liệu đã có.
* Dạng tổng quát của dùng typedef là

**typedef existingType newType;**

* + ***existingType***: là kiểu dữ liệu nào đã tồn tại
  + ***newType***: tên mới của kiểu dữ liệu

khi khai báo typedef rồi thì trong hàm main, khi gọi là kiểu cấu trúc, ko cần struct Sach….



* Khai báo sinh viên: SinhVien sv1;
* Không cần dùng thêm từ khóa struct

* Truy cập biến cấu trúc:
  + Dùng toán tử dấu chấm (dot operator) để truy cập các thành viên của một biến cấu trúc.
  + **Cú pháp:** *tên biến . biến thành phần*

**varNames.memberName**

* + - **Ví dụ:**

strcpy(sv1**.**MaSV, “SV01”);

strcpy(sv1**.**HoTen,“Nguyen Van A”);

***MẢNG CẤU TRÚC***

* Mảng cấu trúc là một mảng mà mỗi phần tử là một biến kiểu cấu trúc.
* Để khai báo một mảng các cấu trúc, trước hết phải khai báo cấu trúc, sau đó khai báo một mảng của cấu trúc đó.

Ví dụ:

typedef struct DANHSACH

{

char hoten[25];

float toan,ly,hoa;

};

DANHSACH bangdiem[50];//mảng 50 phần tử kiểu DANHSACH

COI THEM TẠI PP BÀI 81…

CON TRỎ KIỂU CẤU TRÚC:

* Một biến con trỏ có thể trỏ đến một biến kiểu cấu trúc.
* Cú pháp khai báo một con trỏ cấu trúc

**structureName \*structurePointers;**

Ví dụ:

typedef struct diem

{

int x;

int y;

};

**diem \*p;//p là con trỏ cấu trúc.**

***. biến thành -> trong con trỏ.***

**CẤU TRÚC LỒNG NHAOOO:**

**Kiểu Tập tin**

* + File nhị phân (binary file): dữ liệu ghi trên tệp tin theo các byte nhị phân như trong bộ nhớ.
  + File văn bản (text file): là file có cấu trúc như các file văn bản chuẩn của Dos, hay file văn bản trong chương trình soạn thảo notepad của Windows.

// coi pp bài giảng ( bài 91)

* Trong quá trình lập trình với file chúng ta có các thao tác xử lý:
  + Mở file
  + Đọc file
  + Ghi dữ liệu vào file
  + Đóng file
* Trong C++ để xử lý file chúng ta dùng thư viện fstream
* Chúng ta có thể dùng ifstream để đọc file, ofstream để ghi file. Ngoài ra có thể dùng **fstream để đọc và ghi file**. Thường chúng ta sử dụng fstream.
* Để mở tập tin dùng hàm (phương thức) open

fstream myFile;

myFile.open(filePath, mod);

* + *filePath*: đường dẫn file
  + *mod*: chế độ mở file.
* Các mod trong C++
  + **ios::in**: mở file để đọc.
  + **ios::out**: mở file có sẵn để ghi
  + **ios::binary**: mở file ở chế độ nhị phân.
  + **ios:ate**: mở file và đặt con trỏ file vào cuối file.
  + **ios::app** : mở file và ghi dữ liệu vào cuối file. Nếu file không tồn tại thì tạo file mới.
  + **ios::trunc**: mở file, xóa bỏ nội dung trong file vừa mở.
  + Sau khi thao tác với file xong, chúng ta sẽ đóng file bằng lệnh:
  + **f.close()**
  + Việc đóng file sẽ giúp chúng ta bảo toàn được dữ liệu đang tiến hành lưu trữ và tránh khỏi một số lỗi không đáng có
* Nếu mở file mà không ghi rõ chế độ mở file, trình biên dịch sẽ chuyển về chế độ mở file text.
* Nếu muốn mở ở chế độ binary, ta cần phải ghi rõ ràng mod là ios::binary.
* Nếu mở file dưới chế độ binary sẽ phải đọc file theo dạng binary.
* Sử dụng hàm write và read để đọc và ghi file dưới dạng binary.
* Đọc dữ liệu từ file

// myFile là 1 biến thuộc kiểu fstream

* + myFile.read(address, size);
    - **address**: Kiểu dữ liệu char\*, là địa chỉ mà dữ liệu sẽ lưu vào sau khi đọc lên.
    - **size**: Kiểu dữ liệu int, là số lượng byte trong bộ nhớ được đọc vào từ file.
* Ghi dữ liệu vào file
  + myFile.write(address, size);
    - **adress**: Kiểu dữ liệu char\*, là địa chỉ vùng nhớ của dữ liệu được lưu vào file.
    - **size**: Kiểu dữ liệu int, là số byte vùng nhớ mà nó sẽ ghi.
* Nếu chúng ta muốn ghi các kiểu khác ( khác char) vào file nhị phân thì ta phải dùng cú pháp sau đây:

**reinterpret\_cast<dataType>(value)**

* + dataType sẽ là kiểu dữ liệu mà chúng ta muốn ép về
  + value sẽ là giá trị mà chúng ta muốn ép
  + Ví dụ:

**int x = 1;**

**file.write(reinterpret\_cast< char \* >(&x), sizeof(x));**

* **Đọc dữ liệu từ file**
  + **f >> data; // cin>> ( hơi ngược nha vì cái này là đọc, nó giống kiểu cout<<)**
    - f là tên biến file.
    - data là tên biến lưu dữ liệu. Có thể có các kiểu như int, float, long, char…
    - Khi dữ liệu được đọc có kiểu khác với biến dữ liệu được lưu trữ.
      * Biến lưu trữ dữ liệu kiểu số không đọc vào kiểu kí tự.
      * Biến lưu trữ dữ liệu kiểu số được tự động chuyển kiểu qua lại giữa các kiểu (float sang int, int sang float) bằng cơ chế chuyển kiểu ngầm định.
* **Ghi dữ liệu vào file**
  + **f << data; // cout<< ( giống cin>>, hơi ngược đời☺)**
    - f: là tên biến file.
    - data: biến dữ liệu chứa data cần ghi vào file. Có thể có các kiểu như int, float, long, char…
    - Thư viện fstream hỗ trợ ghi dữ liệu char, int, long.., (nhóm dữ liệu primitive). Tuy nhiên lại không hỗ trợ ghi kiểu dữ liệu mở rộng (struct…)
  + Ngoài ra chúng ta có thể sử dụng hàm get và hàm getline để lấy thông tin từ file
* Kiểm tra kết thúc file

**f.eof()** sẽ kiểm tra xem đã kết thúc file chưa?

Con trỏ chỉ vị:

Hàm dịch chuyển con trỏ chỉ vị trong file

Seekg(x,y);

X: số byte cần dịch chuyển ( là 1 số nguyên)

+âm: dịch về tay trái ( đầu file )

+ dương: dịch về bên phải ( cuối file )

+ n : số byte dịch

Y: vị trí bắt đầu dịch:

+y = 0: đầu file

+y = 1: vị trí hiện tại

+y = 2: cuối file

. Vd: 1 2345 , file.seekg(3,0): từ đầu file dịch 3 byte: 1 2