**NGÔN NGỮ LẬP TRÌNH C++**

# MỘT CHƯƠNG TRÌNH CƠ BẢN CỦA C++

## DÙNG CODE BLOCK ĐỂ VIẾT CHƯƠNG TRÌNH C++

|  |  |
| --- | --- |
| * Click double chuột vào biểu tượng của codeblocks để khởi động chương trình. |  |
| * Tạo một file mới * Chọn file |  |
| * Chọn chương trình C++ |  |
| * Lựa chọn ngôn ngữ lập trình C++ |  |
| * Project title: gõ tên của chương trình * **Folder to create project in** : tạo đường dẫn để lưu chương trình. Ta nhấn vào nút … để chọn thư mục. * **Sau khi điền 2 mục trên, Codeblocks tự điền các mục còn lại**. Ta nhấn Next để bắt đầu soạn thảo chương trình. |  |
| * Khai báo 2 thư viện chuẩn trong C++ | #include <bits/stdc++.h>  using namespace std; |
| * Chương trình chính của C++ | Int main()  {  Nội dung chương trình  Return;  { |
| Kết thúc câu lệnh trong C++ là dấu ; | |
| * Sau khi soạn thảo chương trình xong nhấn Bulid chọn Bulid(Ctrl F9) để dịch chương trình. * Lưu ý nếu khi thay đổi trong chương trình thì buộc phải dịch (bulid) lại. * Sau khi dịch chương trình xong, ta chạy chương trình bằng nhấn vào thẻ Bulid chọn Run (Ctrl F10) hay nhấn biểu tượng Run trên thanh công cụ. |  |

## CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH C++

* Một chương trình C++ thông thường sẽ có hai phần cơ bản
  + **Phần I.** Khai báo thư viện cho trình biên dịch biết cần sử dụng các thư viện nào. Phần này được bắt đầu từ từ khóa **#include.**
  + **Phần II.** Hàm main. Hàm main là điểm mà chương trình C++ sẽ bắt đầu chạy chương trình đầu tiên không phụ thuộc vào vị trí hàm main này được đặt ở đâu (ở đầu, ở cuối hay ở giữa của chương trình). Đây là phần bắt buộc phải có của một chương trình C++.
    - Theo sau hàm main là cặp dấu ngoặc đơn () bởi vì nó là một hàm.
    - Nội dung của hàm main tiếp ngay sau phần được bao trong cặp dấu ngoặc nhọn {}.
    - Nội dung của chương trình chính main bao gồm các câu lệnh và kết thúc một câu lệnh là dấu chấm phẩy (;).
* Một chương trình minh họa

|  |  |
| --- | --- |
| #include <bits/stdc++.h>  using namespace std; | Phần khai báo dùng tất cả các thư viện chuẩn của C và C++.  **Đây là dòng bắt buộc** phải có để viết C+ đơn giản hơn.  Khai báo sử dụng thêm std, nhằm giúp trong chương trình ta có thể sử dụng các hàm một cách thuật tiện hơn. |
| int main()  {    cout<<“ xin chao”;  reture 0;  } | main chính  câu lệnh xuất ra nội dung chữ xin chào ra màn hình  lệnh reture để báo kết thúc hàm main và trả số 0 về cho hàm main ( điều này chúng ta sẽ nghiên cứu sau) |

# CÁC KHÁI NIỆM CƠ BẢN

## CÁC KÝ TỰ DÙNG TRONG C

* Ngôn ngữ lập trình C+ được xây dựng trên bộ ký tự sau:
  + - 26 chữ cái hoa : A, B, C…, Z.
    - 26 chữ cái thường: a, b, c, …, z.
    - 10 chữ số : 0, 1, 2,…9.
    - Các ký tự toán học như +, - , \*, /, =
    - Các ký tự đặc biệt: \_, (, ), #, &, %, …

## TỪ KHÓA

* Từ khóa là những từ được Turbo C+ quy định, từ khóa là những từ có một ý nghĩa hoàn toàn xác định. Từ khóa thường được sử dụng để khai báo các kiểu dữ liệu toán tử, hay các lệnh.
* Các từ khóa cơ bản của Turbo C+: int, float, for, while, break,… Ý nghĩa của mỗi từ khóa sẽ được lần lượt giới thiệu ở các mục sau.
* Từ khóa không được viết IN HOA mà phải dùng chữ thường. Ví dụ từ khóa int thì không được viết là INT.

## TÊN

* Từ khóa là những từ được người lập trình tự đặt để dùng xác định các đối tượng khác nhau trong chương trình. Chúng ta có thể đặt tên cho hằng, tên biến, tên hàm, thủ tuc,….
* Tên do người đặt phải tuân thủ các nguyên tắc sau:
* Tên không được trùng với từ khóa.
* Tên có độ dài không quá 32 ký tự
* Tên không được bắt đầu từ một con số và không chứa các ký tự đặt biệt ( kể cả khoảng trống) nhưng được có ký tự gạch chân để phân biệt giữa các từ.
* **Lưu ý** 
  + Trong các tên, chữ hoa và chữ thường được C+ quy ước là khác nhau. Ví dụ tên biến AB và ab là hai tên dùng cho hai biến khác nhau.

# CÁC KIỂU DỮ LIỆU

## KIỂU SỐ NGUYÊN

* Kiểu số nguyên dùng để lưu các giá trị nguyên hay còn gọi là kiểu đếm được.

|  |  |  |
| --- | --- | --- |
| **KIỂU DỮ LIỆU** | **KÍCH THƯỚC** | **VÙNG GIÁ TRỊ** |
| int | 4 byte |  |
| unsigned int | 2 byte |  |
| long | 4 byte | Hay |
| unsigned long | 4 byte | Hay |
| long long | 8 byte | đến |
| unsigned long long |  | 0 đến 18446744073709551615  0 đến |

## KIỂU SỐ THỰC

* Kiểu số thực dùng để lưu các giá trị thực hay các số có dấu chấm thập phân.
* Để sử dụng kiểu số thực, cần khai báo thư viện ***float.h***

|  |  |  |  |
| --- | --- | --- | --- |
| **KIỂU DỮ LIỆU** | **KÍCH THƯỚC** | **VÙNG GIÁ TRỊ** |  |
| float | 4 byte | Từ | 6 vị trí thập phân |
| double | 8 byte | Từ | 15 vị trí thập phân |
| long double | 10 byte | Từ | 19 vị trí thập phân |

## HẰNG SỐ

* Hằng là một ô nhớ dùng để lưu trữ dữ liệu. Trong suốt chương trình giá trị của Hằng số không thay đổi.
* Khai báo Hằng

|  |  |
| --- | --- |
|  | |
|  | Khai báo hằng số tên pi kiểu số thực và có giá trị là 3.14 |
|  | Khai báo hằng số tên x kiểu số nguyên có giá trị là 10 |

# TOÁN TỬ

## PHÉP TOÁN SỐ HỌC

|  |  |  |  |
| --- | --- | --- | --- |
| **Phép toán** | **C+** | **Ví dụ** | |
| Phép cộng | + |  |  |
| Phép trừ |  |  |  |
| Pháp nhân | \* |  |  |
| Phép chia | / |  |  |
| * Nếu phép chia hai số nguyên thì / trở thành phép chia lấy **phần nguyên.** |  | * 5 chia 3 **được 1** dư 2. * Nên kết quả là 1 |
| * Nếu chia hai số nguyên muốn ra số thập phân thì ép kiêu |  |  |
| * Nếu phép chia hai số trong đó có một số thực thì kết quả luôn cho là một **số thực.** |  |  |
| Phép chia lấy phần dư | % |  | * 5 chia 3 **được 1** dư 2. * Nên kết quả là 2. |

## PHÉP TOÁN LOGIC

* Trong C, phép so sánh và phép logic trả về giá trị 1 ( ứng với True – Đúng) hay 0 ( ứng với fasle – Sai). Do đó để lưu kết quả của phép so sánh hay logic ta dùng biến int.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phép toán** | **C+** | **Giá trị trả về** | **Ví dụ** |
| Phép Và (AND) | && | * Phép chỉ trả về giá ***ĐÚNG*** (1) khi a và b đều có giá trị đúng (1). |  |
| Phép Hoặc (OR) | || | * Phép chỉ trả về giá ***SAI (0)*** khi a và b đều có giá trị sai (0). |  |
| Phép phủ định (NOT) | ! | * Phép trả về giá đối của a. |  |

## PHÉP SO SÁNH

|  |  |  |
| --- | --- | --- |
| **Phép toán** | **C+** | **Ví dụ** |
| **Có lớn hơn không** | > | có giá trị 0  có giá trị 1 |
| **Có lớn hơn hay bằng nhau không?** | >= | có giá trị 0  có giá trị 1  có giá trị 1 |
| **Có nhỏ hơn không** | < | có giá trị 1  có giá trị 0 |
| **Có nhỏ hơn hay bằng nhau không?** | <= | có giá trị 1  có giá trị 0  có giá trị 1 |
| **Có bằng nhau không** | == | có giá trị 0.  có giá trị 1. |
| **Có khác nhau không** | != | có giá trị 1.  có giá trị 0. |

# BIẾN

## KHAI BÁO BIẾN

* Mọi biến cần phải khai báo trước khi sử dụng. Việc khai báo biến chỉ thực hiện một lần duy nhất, là lần đầu tiên biến xuất hiện trong chương trình với cú pháp

**<kiễu dữ liệu> tên\_biến;**

|  |  |
| --- | --- |
| int a,b,c; | Khai báo 3 biến a, b, c kiểu int; |
| long m,n; | Khai báo 2 biến m, n kiểu long; |
| Char k; | Khai báo biến k kiểu ký tự. |

* Biến kiểu nào thì chỉ được chứa các giá trị của kiểu đó.

## NHẬP GIÁ TRỊ TỪ BÀN PHÍM TRONG C

* Cú pháp nhập giá trị từ bàn phím cho một biến

**Cin>>ten biến;**

## GÁN GIÁ TRỊ BIẾN

* Để gán giá trị biến. Ta dùng cú pháp

**<tên\_biến>=<giá trị của biến>;**

|  |  |
| --- | --- |
| a=5; | Gán số 5 cho biến a.  Vậy sau lệnh này biến a có giá trị là 5. |
| int q=9; | Khai báo biến q kiểu int, và q có giá trị ban đầu là 9; |
| a=b=5; | Gán số 5 cho biến a và b.  Vậy sau lệnh này biến a và b đều có giá trị là 5. |
| Có thể viết gọn | Vậy sau lệnh gán này thì |
| Nghĩa là  Nghĩa là  Nghĩa là  Nghĩa là |

## PHÉP TOÁN TĂNG GIẢM

* C đưa ra hai phép toán để tăng và giảm các biến. Toán tử tăng ++ sẽ cộng 1 vào toán tử của nó. Toán tử giảm -- sẽ trừ đi 1.

|  |  |
| --- | --- |
|  | tương ứng với với câu lệnh  sau câu lệnh th có giá trị là 3. |
|  | tương ứng với với câu lệnh  sau câu lệnh th có giá trị là 1. |
|  | đứng sau biến, thì giá trị biến sẽ được gán cho biến , rồi sau đó giá trị biến mới được tăng lên 1.  Ví dụ , sau câu lệnh này thì |
|  | đứng trước biến, thì giá trị biến sẽ được tăng lên 1 giá trị rồi sau đó gán cho biến .  Ví dụ , sau câu lệnh này thì |

## ĐỊNH NGHĨA KIỂU DỮ LIỆU

|  |  |
| --- | --- |
|  | |
|  | Định nghĩa kiểu dữ liệu mới tên là có kiểu là long long  Sau lệnh này ta dễ dàng khai báo các biến khác kiểu long long với câu lệnh  Khai báo 2 biến a, b với kiểu long long. |

# HÀM XUẤT RA MÀN HÌNH C++

* Cú pháp hàm cout

**Cout<< “chuỗi”<<tên biến 1<<tên biến 2<<endl;**

* + Chuỗi là chuỗi cần xuất ra màn hình
  + Tên biến 1, tên biến 2 là cần xuất giá trị của 2 biến này ra màn hình.
  + Endl dấu hiệu xuống dòng;

|  |  |
| --- | --- |
| Câu lệnh | Màn hình hiển thị |
| **Cout<<“ xin chao ”;** | xin chao\_ |
| **Cout<<“ xin chao ”<<end;** | Xinchao  \_ |
| **cout<< “1”<< “ ”<< “2”** | 1 2 |

# THAO TÁC VỚI FILE DỮ LIỆU

* Trong C++ để thao tác trên kiểu dữ liệu file ta dùng hàm freopen.
* Sử dụng hàm freopen.

**freopen(“Tên file”, <chế độ mở file>);**

* Trong đó:
  + Tên file là tên của tập tin mà ta cần thao tác trên nó.
  + Chế độ mở tệp: là các hằng số được định nghĩa sẳn bởi C++ quy định các chế độ mở đọc, mở ghi hay mở ra để cả đọc và ghi.

|  |  |
| --- | --- |
| “r”, stdin | Mở file để đọc. |
| “w”,stdout | Mở file để ghi. |

* Trong đó:

|  |  |
| --- | --- |
| freopen(“UCLN.INP”,“r”, stdin); | Gắn stdin( đường vào của C++) với file UCLN.INP.  Sau hàm này tất cả các lệnh cin đều được lấy giá trị từ file UCLN.INP. |
| freopen(“UCLN.OUT”,“w”, stdout); | Gắn stdout ( đường xuất của C++) với file UCLN.OUT.  Sau hàm này tất cả các lệnh cout đều được ghi lên file UCLN.OUT. |

* Để tạo file INP, OUT ta nhấn tổ hợp phím **CTRL SHIFT N**. Nhập tên file có đuôi.INP và đuôi.OUT tương ứng.

# CÂU LỆNH ĐIỀU KIỆN

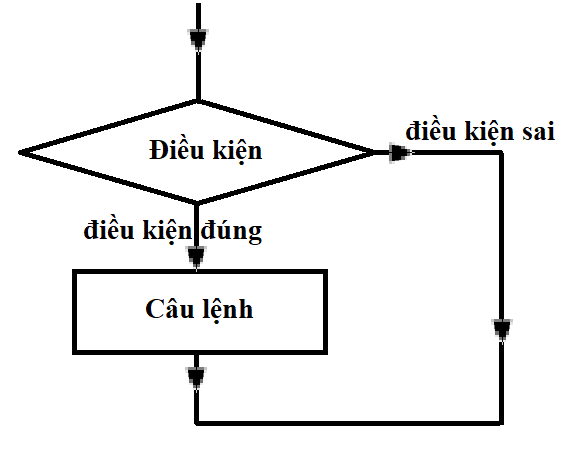
## CÂU LỆNH ĐIỀU KIỆN THIẾU

* Cú pháp câu lệnh điều kiện thiếu

if **(điều kiện)**

Câu lệnh;

* Ý nghĩa



* + Xét điều kiện:
    - Nếu điều kiện đúng (có giá trị 1) thì câu lệnh được thực hiện.
    - Nếu điều kiện sai ( có giá trị 0) thì câu lệnh bị bỏ qua (không được thực hiện)

## CÂU LỆNH ĐIỀU KIỆN ĐỦ

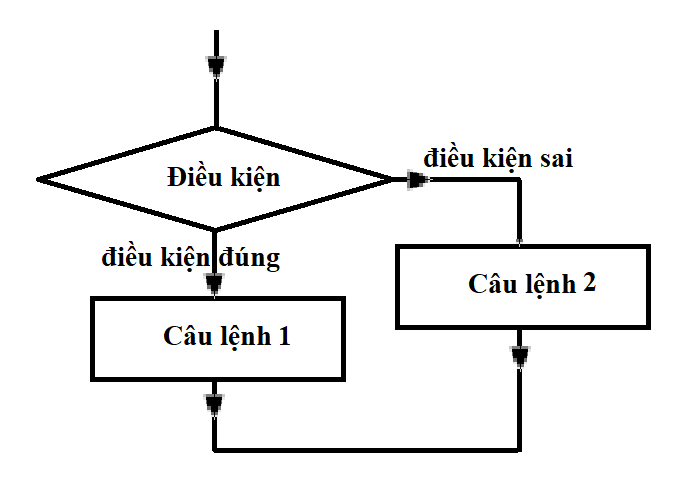
* Cú pháp câu lệnh điều kiện đủ

if **(điều kiện)**

Câu lệnh 1;

else

Câu lệnh 2;



* Ý nghĩa
  + Xét điều kiện:
    - Nếu điều kiện đúng (có giá trị 1) thì **câu lệnh 1** được thực hiện.
    - Nếu điều kiện sai ( có giá trị 0) thì câu lệnh 2 được thực hiện.
* **Lưu ý.** Nếu là nhiều lệnh (khối lệnh) thì khối lệnh cần được đặt trong cặp dấu ngoặc nhọn { }.

# CÂU LỆNH LẶP

## VÒNG LẶP FOR

* Cú pháp

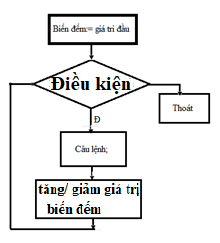
**for ( biến đếm=giá trị đầu; điều kiện lặp; tăng/giảm giá trị biến)**

**{**

**Câu lệnh lặp;**

**}**

* Ý nghĩa.



* **Bước 1.**Biến đếm sẽ được gán giá trị đầu.
* **Bước 2.** Xét điều kiện lặp. **Nếu điều kiện sai** thoát vòng lặp**. Nếu điều kiện đúng** thì câu lệnh lặp sẽ được thực hiện rồi sang bước 3.
* **Bước 3.**Giá trị biến đếm sẽ được tăng hay giảm theo biểu thức tăng giảm.
* **Bước 4.** Quay về bước 2.
* Ví dụ

|  |  |
| --- | --- |
| int i;  for (i=1; i<=10;i++)  cout<<i<<” ”;  count<<endl<<“sau vong lap gia tri bien dem la ”<<i; | Khai báo biến i là biến toàn cục. (nghĩa là biến được sử dụng ở bất kỳ đâu trong chương trình)  Vòng for này sẽ in ra các giá trị từ 1 đến 10.  1 2 3 4 5 6 7 8 9 10  sau vong lap gia tri bien dem la 11 |
| for (int i=1; i<=10;i++)  cout<<i<<” ”;  count<<endl<<“sau vong lap gia tri bien dem la ”<<i; | Khai báo biến i là biến cục bộ của vòng lặp for (nghĩa là i chỉ tồn tại trong vòng lặp)  Vòng for này sẽ in ra các giá trị từ 1 đến 10.  1 2 3 4 5 6 7 8 9 10  C++ sẽ báo lỗi vì không nhận biết biến i. |
| int i;  for (i=1; i<=10;i+=3)  cout<<i<<” ”;  count<<endl<<“sau vong lap gia tri bien dem la ”<<i; | i+=3 nghĩa là i=i+3;  Biến i được tăng mỗi lần 3 đơn vị.  Vòng for này sẽ in ra các giá trị  1 4 7 10  sau vong lap gia tri bien dem la 13 |
| int i;  for (i=10; i>=10;i+=3)  cout<<i<<“ ”; | Vòng lặp này sẽ là vòng lặp vô tận, vì điều kiện luôn đúng, do đó chúng ta cần lưu ý kỹ đến điều kiện lặp và giá trị tăng giảm biến đếm. |
| int i;  for (i=10; i>=10;i-=3)  cout<<i<<‘ ’;  count<<endl<<“sau vong lap gia tri bien dem la ”<<i; | Vòng for này sẽ in ra các giá trị  10 7 4 1  sau vong lap gia tri bien dem la -2 |
| int i=3;  for (int i=10; i>=10;i-=3)  cout<<i<<‘ ’;  count<<endl<<“sau vong lap gia tri bien dem la ”<<i; | Biến i ở trên đã khai báo biến toàn cục  Nhưng vào for ta khai báo biến i, C++ sẽ ưu tiên xem i là biến cục bộ của for, và giá trị của i trong for chỉ là giá trị của biến cục bộ không ảnh hưởng đến biến toàn cục.  Vòng for này sẽ in ra các giá trị  10 7 4 1  sau vong lap gia tri bien dem la 3 (đây là giá trị từ đầu ta khai báo ở biến toàn cục). |

* **Cú pháp tổng quát của for.**

**for ( biến đếm1=giá trị đầu1, biến đếm2=giá trị đầu2; điều kiện lặp; tăng/giảm giá trị của các biến đếm)**

**{**

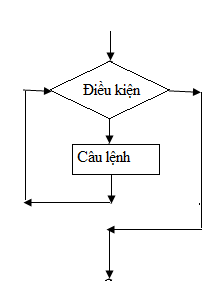
**Câu lệnh lặp;**

**}**

|  |  |
| --- | --- |
| for (int i=0,j=0; i+j<=10;j++,i++)  cout<<i+j<<“ ”; | Vòng for này sẽ in ra các giá trị  0 2 4 6 8 10 |

## VÒNG LẶP WHILE

* Cú pháp



**while (điều kiện lặp)**

**{**

**Câu lệnh lặp;**

**}**

* Ý nghĩa
* Nếu điều kiện đúng câu lệnh sẽ được lặp.
* Sau khi câu lệnh được thực hiện, C++ sẽ xét tiếp điều kiện. Vòng lặp sẽ lặp cho đến khi điều kiện là sai.
* **Lưu ý.** Câu lệnh lặp trong while là khối lệnh, vì trong câu lệnh lặp phải có câu lệnh làm thay đổi điều kiện. Nếu không vòng lặp sẽ thành vòng lặp vô tận.

|  |  |
| --- | --- |
| int i=1;  while (i<5)  {  cout<<i<<“ ”;  i+=1;  } | Vòng lặp sẽ in ra các giá trị  1 2 3 4 |
| int i=1;  while (i<5)  cout<<i<<“ ”; | Đây là vòng lặp vô tận, sẽ in mãi số 1, vì i giữ giá trị 1 và điều kiện lặp luôn đúng |

## LỆNH BREAK VÀ CONTINUE

|  |  |  |
| --- | --- | --- |
| break; | Lệnh break; trong C++ được sử dụng để thoát khỏi vòng lặp trong C++ | |
|  | for (i=1; i<=10;i+=3)  {  cout<<i<<‘ ’;  if i==3  break;  } | Vòng lặp này sẽ in ra màn hình  1 2 3  Vì i=3 C++ thực hiện câu lệnh break làm thoát vòng lặp |
| **Continue;** | Lệnh này nó buộc trở về kiểm tra điều kiện để thực hiện vòng lặp tiếp theo và bỏ qua các lệnh bên trong vòn lặp hiện tại sau lệnh continue.  Đối với vòng lặp for, câu lệnh continue làm cho điều khiển chương trình tăng hoặc giảm biến đếm của vòng lặp. Đối với vòng lặp while và do-while, câu lệnh continue làm cho điều khiển chương trình quay về đầu vòng lặp và kiểm tra điều kiện của vòng lặp. |  |

# CÁC HÀM LẤY GIÁ TRỊ KIỂU DỮ LIỆU

|  |  |  |
| --- | --- | --- |
| * Trả về giá trị lớn nhất của kiểu số nguyên |  |  |
| * Trả về giá trị nhỏ nhất của kiểu số nguyên |  |  |
| * Trả về giá trị lớn nhất của kiểu số long |  |  |
| * Trả về giá trị nhỏ nhất của kiểu số long |  |  |
| * Trả về giá trị lớn nhất của kiểu số long |  |  |
| * Trả về giá trị nhỏ nhất của kiểu số long |  |  |
| * Trả về giá trị lớn nhất của kiểu số char |  |  |
| * Trả về giá trị nhỏ nhất của kiểu số char |  |  |
| * Trả về giá trị lớn nhất của kiểu số unsigned long |  |  |

# CÁC HÀM TOÁN HỌC THÔNG DỤNG

|  |  |  |
| --- | --- | --- |
|  | Trả về giá trị tuyệt đối của số nguyên |  |
|  | Trả về giá trị tuyệt đối của số thực |  |
|  | Trả về giá trị lớn nhất của hai số nguyên và |  |
|  | Trả về giá trị nhỏ nhất của hai số nguyên và |  |
|  | Trả về giá trị lớn nhất, nhỏ nhất của hai số thực và . |  |
|  | Trả về số nguyên là giá trị làm tròn xuống của .  Hay có thể hiểu là lấy phần nguyên của số thực . |  |
|  | Trả về giá trị nguyên là giá trị làm tròn lên của số thực |  |
|  | Trả về số nguyên là giá trị làm tròn của số thực |  |
|  | Trả về giá trị căn bậc 2 của |  |
|  | Trả về giá trị căn bậc 3 của |  |
|  | Trả về phần dư của phép chia 2 số thực. |  |
|  | Đổi giá trị hai biến cho nhau | và |
|  | Tính |  |
|  | Tính |  |

# KIỂU KÝ TỰ - KIỂU CHUỔI

## BẢNG MÃ ASCII

1. **Bảng mã Ascii**

* Là bảng ký tự và bảng mã ký tự của bảng chữ cái la tinh và con số trong thể hiện kiểu bit trong máy tính. Gồm 127 ký tự tương ứng từ 0 đến 126.

1. **Bảng mã ký tự số**

|  |  |
| --- | --- |
| Mã hóa bằng bảng mã trong hệ thập phân | Ký tự con số được thể hiện con |
| 48 | 0 |
| 49 | 1 |
| 50 | 2 |
| 51 | 3 |
| … | … |
| 57 | 9 |

1. **Bảng mã ký chữ cái thường**

|  |  |
| --- | --- |
| Mã hóa bằng bảng mã trong hệ thập phân | Ký tự chữ cái thường được thể hiện con |
| 97 | a |
| 98 | b |
| 99 | c |
| 100 | d |
| … | … |
| 122 | z |

1. **Bảng mã ký chữ cái in hoa**

|  |  |
| --- | --- |
| Mã hóa bằng bảng mã trong hệ thập phân | Ký tự chữ cái in hoa được thể hiện con |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| … | … |
| 90 | Z |

## KIỂU KÝ TỰ

1. **Định nghĩa.**

* Kiểu ký tự là một kiểu dữ liệu có kích thước là 1 byte, dùng để chứa một ký tự.
* Kiểu ký tự luôn đặt giữa hai dấu nháy đơn (‘ ’).

1. **Khai báo**

|  |  |
| --- | --- |
| char b; | Biến b có thể là kiểu ký tự |

1. **Các hàm sử lý trên ký tự**

* Kiểu char trong C++ được lưu trữ trong bộ nhớ là kiểu số nguyên (phạm vi từ ). Do đó ta có thể sử dụng các phép toán của số nguyên đối với kiểu ký tự, tương ứng với giá trị của ký tự trong bảng mã ASCII.

|  |  |  |
| --- | --- | --- |
| **Hàm** | **Kết quả** | **Giải thích** |
| **;** | ; | Ký tự ‘a’ trong bảng mã là 97.  Cộng 1 vào sẽ là 98 → ký tự b; |
| **;** | ; | Ký tự ‘c’ trong bảng mã là 99.  Trừ 2 vào sẽ là 97 → ký tự a; |
| **;** | 99 | Ép kiểu trả ra giá trị bảng mã của ký tự c |
| **;** |  | Ký tự ‘a’ trong bảng mã là 97.  Ký tự ‘c’ trong bảng mã là 99 |
|  |  | Ép kiểu trả về ký tự thứ 97 trong bảng mã. |
|  |  | Đổi từ chữ số về số ta lấy biến chứ ký tự đó -48. |
| **Đổi từ chữ cái in thường sang in HOA** |  | Đổi từ chữ cái thường ra in HOA  Lấy biến |
|  | Kiểm tra giá trị của c có phải là chữ cái không?   * Nếu C là chữ in HOA giá trị trả về là 1 * Nếu C là chữ thường giá trị trả về là 2 * Nếu C không phải chữ giá trị trả về là 0 |  |
| **Đổi từ chữ cái in HOA sang in THƯỜNG** |  | Đổi từ chữ HOA ra in HOA  Lấy biến |
|  | Kiểm tra giá trị của c có phải là chữ số không?   * Nếu C là chữ số từ ‘0’ đến ‘9’ thì trả về giá trí 1 * Nếu C là không phải chữ số thì trả về giá trị 0. |  |

## KIỂU CHUỔI

1. **Định nghĩa**

* Kiểu chuổi trong được coi là một mảng một chiều với các phần tử là một ký tự mà kết thúc của chuổi bởi ký tự null ‘\0’.
* Chiều dài của chuổi khá lớn (không giới hạn), phụ thuộc vào khả năng cấp phát bộ nhớ của máy tính.

1. **Khai báo**

|  |  |
| --- | --- |
| string a; | Khai báo biến a là kiểu chuổi.  Phần tử đầu tiên của chuổi là 0 |
| Char c[50]; | Biến c là một biến chuỗi gồm 50 phần tử ( một mảng gồm 50 phần tử, mỗi phần tử có kiểu là ký tự) |

1. **Lưu ý:**

* Một **string** với chiều dài **n**, những vị trí của các ký tự của **string** có phạm vi từ 0 tới n – 1.
* Có thể truy xuất ký tự trong **string** như mảng với **cú pháp**: **<tên chuỗi> [vị trí]**.
* Có thể gán hai chuỗi **string**, nối chuỗi bằng toán tử “+”.
* So sánh hai chuỗi bằng các **toán tử quan hệ (<, >, ==, !=,…)**.

1. **Các hàm sử lý trên chuổi**

|  |  |  |
| --- | --- | --- |
| Hàm | Ý nghĩa | Ghi chú |
|  | * Lấy các ký tự từ cin vào chuổi đến khi gặp ký tự khoảng trắng. | Chuổi được khai báo  **string chuổi;** |
|  | * Lấy hết các ký tự từ cin vào chuổi. |
|  | * Lấy từ cin ký tự đưa vào chuổi | Chuổi được khai báo  **char chuổi [số lượng];** |
|  | * Trả về số ký tự của chuổi. |  |
|  |
|  | Trích một chuổi con với chiều dài y bắt đầu từ vị trí x | String s=“ABCDEFGH”  s.substr(2,3)=”CDE”  s.substr(2)=”CDEFGH” |
| Nếu không có y, sẽ trích một chuổi từ vị trí x đến cuối chuổi |
|  | Kiểm tra chuổi có trong chuổi không? Nếu có thì trả về vị trí bắt xuất hiện đầu tiên của s1 trong chuổi.  Nếu không sẽ trả về giá trị -1 (giá trị max của unsigned long long) | String s=“ABCDEBCD”  s.find(“CD”)=2  s.find(“BE”)= -1 |
| Chuổi.erase(chuổi.begin()+i); | * Xóa một ký tự thứ i trong chuổi. | String s=“ABCDEFGH”  s.erase(s.begin()+2);  s=“ABDEFGH” |
|  | Xóa tất cả chừa lại i ký tự đầu | s.erase(2);  s=“AB” |
|  | Xóa từ vị trí i, xóa j ký tự | s.erase(1,2);  s=“ ADEFGH” |
|  | Thay thế n ký tự tại vị trí bắt đầu là bằng chuổi . | s=“ ABCDEF”  s.replace(2,3,“123”);  s=“AB123F” |
| Lưu ý lệnh replace là thay thế n ký tự của s bằng str. | s=“ ABCDEF”  s.replace(2,3,“12”);  s=“AB12F” |
| s=“ ABCDEF”  s.replace(2,3,“1234”);  s=“AB1234F” |
|  | So sánh chuổi với str  Nếu chuổi>str kết quả là  Nếu chuổi=str kết quả là  Nếu chuổi<str kết quả là | s="ABCDEF";  cout<<s.compare("ABDEF");  **kết quả là .** |
| s="ABCDEF";  cout<<s.compare("ABB");  **kết quả là .** |
| s="ABCDEF";  string s1="AA";  cout<<s.compare(s1);  **kết quả là .** |
|  | Chèn str vào chuổi từ vị trí | Kết quả |
|  | Đổi của biến số thành chuổi. | Kết quả |

# MẢNG

## MẢNG MỘT CHIỀU

1. **Định nghĩa**

* Mảng là một dãy hữu hạn các phần tử có cùng kiểu dữ liệu.
* Trong C++ phần tử đầu tiên luôn được đánh số 0.

1. **Khai báo**

* Kiểu dữ liệu **tên mảng** [số phần tử];

|  |  |
| --- | --- |
| Khai báo | Ý nghĩa |
| int a[10]; | Khai báo một mảng a gồm 10 phần tử, mỗi phần tử đều có kiểu int. |

1. **Các lệnh liên quan đến mảng một chiều**

|  |  |  |
| --- | --- | --- |
| Truy cập vào phần tử thứ i của mảng | a[i]=5 | Gán phần tử thứ i của mảng với giá trị 5 |
| Nhập xuất giá trị của mảng | Ta dùng vòng lặp for, ví dụ mảng a có n phần tử. Nhập giá trị mảng a từ bàn phím.  for(i=0;i<n; i++)  cin>>a[i]; | |
| Lưu ý. Nếu khởi tạo mảng trong hàm main thì giá trị của mãng sẽ có giá trị ngẫu nhiên (Rác) | int main()  {  int a[10];  for (i=0;i<10;i++)  cout<<a[i]<<“ ”;  kết quả là  8 0 4199705 0 8 0 44 0 7018240 0 16 | |
| Nếu khởi tạo mảng trước hàm main thì giá trị của mãng là 0 | int a[10];  int main()  {  for (i=0;i<10;i++)  cout<<a[i]<<“ ”;  kết quả là  0 0 0 0 0 0 0 0 0 0 | |
| Gán tất cả giá trị của mãng cùng 1 giá trị | Fill(mãng,mãng+ số phần tử mãng, giá trị cần gán) | |
| int a[5];  fill(a,a+5,2);  nghĩa là tất cả 5 phần tử của a đều có giá trị là 2 | |

## HÀM SORT MẢNG MỘT CHIỀU

* Cú pháp

|  |
| --- |
|  |

* **Lưu ý.**
* Vị trí đầu là số 0
* Nếu cách sắp xếp để trống thì C++ tự mặc định là sắp xếp tăng dần
* Để sắp xếp giảm dần

|  |  |
| --- | --- |
| bool cmp(int A, int B)  {  return A>B;  } | Xây dựng một hàm cmp |
|  | |

|  |  |  |
| --- | --- | --- |
| Cho mảng | | |
|  |  | Do chỉ số sắp xếp bắt đầu từ 0 và xếp 4 phần tử  Nên xét 4 số 5 3 4 1 để xếp. |
|  |  |  |
| bool cmp(int A, int B)  {  return A>B;  } |  | Sắp xếp giảm dần mảng a |

## MẢNG HAI CHIỀU

1. **Định nghĩa**

* Mảng hai chiều lưu trữ các phần tử theo dạng bảng ( ma trận) gồm dòng và cột. Hình bên là một bảng ( ma trận) gồm m hàng và n cột

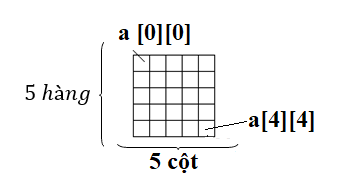
1. **Khai báo**

|  |
| --- |
|  |

* **Lưu ý** ta nên khai báo mảng 2 chiều là biến toàn cục ( trước hàm main) không nên khai báo là biến cục bộ vì sẽ bị giới hạn bởi bộ nhớ cấp phát cho một hàm.

|  |  |  |
| --- | --- | --- |
| int a[4][3] | Khai báo một mảng hai chiều a gồm 4 hàng và mỗi hàng 3 cột |  |
| int b[3][4] | Khai báo một mảng hai chiều b gồm 4 hàng và mỗi hàng 3 cột |  |
| int c[3][3] | Khai báo một mảng hai chiều c gồm 3 hàng và mỗi hàng 3 cột.  Lúc này mảng c được gọi là bảng vuông |  |

1. **Truy xuất đến một phần tử trong mảng**

* Cũng như mảng một chiều, hàng đầu tiên, cột đầu tiên của bảng đều được đánh từ số 0.
* Một ô của bảng sẽ được xác định bởi của ô đó.
* Cú pháp
* Khai báo sẽ khởi tạo một bảng như hình bên.
* Hàng đầu tiên là hàng số 0.
* Cột đầu tiên là cột thứ 0.
* Ô đầu tiên có địa chỉ , ô thứ hai là , cứ như vậy ô cuối cùng hàng 1 là .
* Vậy để truy cập vào ô ở hàng i hàng j trong mảng a. ta dùng tên .
* Để duyệt các phần tử của mảng thường ta sẽ dùng hai vòng lặp for để duyệt.

1. **Khởi tạo giá trị ban đầu cho mảng hai chiều**

* Khởi tạo giá trị ban đầu như mảng một chiều.

|  |  |  |
| --- | --- | --- |
|  | sẽ tạo một bảng có các giá trị ban đầu như sau: |  |
|  | Khởi tạo giá trị ban đầu theo từng nhóm là giá trị các dòng |
|  | Khởi tạo các giá trị ban đầu, các phần tử còn lại có giá trị ngẫu nhiên nhưng thường là 0 |  |
|  | Khởi tạo tất cả giá trị ban đầu của a là |  |

* Lưu ý việc khởi tạo giá trị ban đầu cho bảng chỉ được chấp thuận trong khi khai báo bảng.

|  |  |
| --- | --- |
|  | Sẽ bị báo lỗi. |
|  |

# CHƯƠNG TRÌNH CON

## CHƯƠNG TRÌNH CON TRONG C++

* Dựa vào tham số và kiểu trả về, có thể có các loại hàm sau:
  + Hàm **không có tham số** và **không có kiểu trả về**
  + Hàm **không có tham số** nhưng **có kiểu trả về**
  + Hàm **có tham số** nhưng **không có kiểu trả về**
  + Hàm **có tham số** và **có kiểu trả về**

|  |  |  |
| --- | --- | --- |
| Hàm không có tham số và không có kiểu trả về. | Void tên\_hàm  {  Các câu lệnh;  } | void nhap()  {  freopen(“a.INP”,“r”, stdin);  cin>>a;  } |
| Cách gọi hàm  Tên\_hàm(); | void main()  {  int a;  nhap();  } |
| **Hàm không có tham số nhưng có kiểu trả vè** | Kiểu\_trả\_về tên\_hàm  {  Các câu lệnh;  Return giá\_trị\_trả\_về;  } | Int nhap()  {  int n;  freopen(“a.INP”,“r”, stdin);  cin>>n;  return n;  } |
| Cách gọi hàm  Biến=Tên\_hàm(); | Void main()  {  int a;  a = nhap();  } |
| Hàm có tham số nhưng không có kiểu trả về | Void Tên( kiểu TS biến)  {  Các câu lệnh;  } | void Luy\_thua (int a)  {  S=1;  For( int i=1;i<=a;i++)  s=s\*2;  } |
| Cách gọi hàm  Tên\_hàm(biến); | Int main()  {  int s;  luythua(n);  return 0;  }  Lúc này giá trị n sẽ được truyền vào biến a.  Và khi gọi hàm luythua. Ta có S là kết quả của . |
| **Hàm có tham số và có kiểu trả về.** | Kiểu hàm  {  Các câu lệnh;  Return giá\_trị\_trả\_về;  } | int luythua(int a)  {  int s;  For( int i=1;i<=a;i++)  s=s\*2;  return s;  } |
| Cách gọi hàm  Biến=Tên\_hàm(biến); | Cout<<“2 mu n”<<luythua(n); |

## TRUYỀN THAM SỐ, TRUYỀN THAM TRỊ

* Trong C++ hỗ trợ những cách truyền đối số cho hàm là:
* Truyền bằng tham trị (pass by value)
* Truyền bằng địa chỉ (pass by address hoặc pass by pointer)
* Truyền bằng tham chiếu (pass by reference)

1. **Khái niệm tham số, đối số và tham trị.**

* Tham số của hàm là những biến được khai báo trong khai báo hàm. Tham số đóng vai trò là giá trị đầu vào của hàm số.
* Đối số là giá trị được truyền vào hàm mỗi khi thực hiện lời gọi hàm. Đối số phải có kiểu dữ liệu phù hợp với tham số của hàm.

|  |  |
| --- | --- |
| Void foo (int a,int b)  {  Câu lệnh;  } | Hai biến a,b được gọi là tham số của hàm foo. |
| int main()  {  foo(1,2);  } | 1 và 2 là hai đối số của hàm foo.  Khi đó giá trị 1 sẽ được tiếp nhận và lưu trữ bởi biến a  Giá trị 2 sẽ được tiếp nhận và lưu trữ bởi biến b. |

* Truyền đối số cho hàm có thể truyền dưới dạng hằng ( giá trị cụ thể), biến, biểu thức nhưng tham số của hàm chỉ sẽ nhận giá trị, ta gọi là **truyền tham trị**.
* Lúc này nếu đối số cho hàm là một biến thì tham số chỉ lấy giá trị của biến, và sau khi hàm được gọi giá trị của biến đối số không đổi.

|  |  |
| --- | --- |
| int tang(int a)  {  a++;  return a;  }  Void main()  {  int ;  cout<<tang(x)<< “ ”<<x;  } | Ta được kết quả là  6 5  Vì x được truyền tham trị  Nên tham số a nhận giá 5  Còn đối số x không bị thay đổi giá trị nên vẫn là 5. |

1. **Truyền tham chiếu cho hàm**

* Nếu muốn giá trị của đối số là biến cũng thay đổi sau khi lời gọi hàm thì ta cần truyền theo kiểu tham chiếu. Lúc này tham số cho hàm phải được định nghĩa dưới dạng địa chỉ ( hay con trỏ) – phần này chúng ta nghiên cứu kỹ hơn ở phần con trỏ, để làm điều này chúng ta dùng toán tử &.
* Lưu ý
  + Khi đã khai tham số dưới dạng địa chỉ thì phép truyền gọi là truyền tham chiếu.
  + Đối với phép truyền tham chiếu thì đối số chỉ có thể biến.

|  |  |
| --- | --- |
| int tang(int &a)  {  a++;  return a;  }  Void main()  {  int ;  cout<<tang(x)<< “ ”<<x;  } | Ta được kết quả là  6 6  Vì x được truyền tham chiếu  Nên tham số a nhận giá 5 và lúc này x và a như nhau.  a có giá trị 6 thì x cũng có giá trị 6 |

1. **Truyền mảng cho hàm**

* Mảng có thể được truyền vào hàm như là các tham số.
* Bản chất của việc truyền mảng vào hàm là ta truyền địa chỉ của phần tử đầu tiên cho hàm chứ không phải truyền toàn bộ mảng. Do đó các mảng luôn được **truyền con trỏ** (**pass by pointer**) vào hàm. Nó tương tự truyền tham chiếu, có nghĩa là các mảng có thể **thay đổi giá trị** bởi lời gọi hàm.

|  |  |
| --- | --- |
| Void changeArr (int b[])  {  a[0]=99;  }  Int main()  {  int a[5]={1;2;3;4;5};  changeArr(a);  return 0;  } | Kết quả sau lời gọi hàm changeArr  Mảng a là 99 2 3 4 5. |

# CON TRỎ (POINTER)

## KHÁI NIỆM

* Trong ngôn ngữ C++, con trỏ (pointer) là biến lưu trữ địa chỉ bộ nhớ của những biến khác.
* Biến con trỏ chỉ dùng chứa địa chỉ của một ô nhớ mà con trỏ trỏ tới. Nhưng khi ta truy cập đến biến con trỏ thì biến con trỏ sẽ trả về giá trị của biến mà biến con trỏ trỏ tới.

|  |  |
| --- | --- |
|  | Trong sẽ tạo     * Biến a có địa chỉ là 0x61ff08 * Vậy con trỏ khi trỏ đến biến a sẽ cho giá trị là 0x61ff08 |

## KHAI BÁO

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | * Khai báo một biến cho trỏ ch, dùng để trỏ đến vùng nhớ có kiểu char. |
|  | * Khai báo một biến con trỏ i, dùng để trỏ đến vùng nhớ có kiểu số nguyên. * Khai báo một biến j là biến chứa số nguyên |

## CÁC PHÉP TOÁN VỚI BIẾN CON TRỎ

|  |  |  |  |
| --- | --- | --- | --- |
| Gán giá trị cho biến con trỏ |  |  | 5  0x61ff08  5  0x61ff08  100 |
| Giá trị của biến con trỏ |  |
| Thay đổi giá trị của biến | Khi đã gán biến con trỏ với một ô nhớ, thì khi giá trị của con trỏ thay đổi thì giá trị của biến cũng thay đổi theo |

## SỰ TƯƠNG QUAN GIỮA CON TRỎ VÀ MẢNG MỘT CHIỀU

* Bản chất khi ta khai báo một mảng một chiều, C++ sẽ cung cấp một số các ô nhớ liên tiếp trong bộ nhớ. Do đó chỉ cần xác định được địa chỉ của ô chứa phần tử đầu tiên của mảng ta dễ dàng suy ra các địa chỉ các ô kế tiếp.
* Do đó có thể sử dụng một biến con trỏ để lưu trữ địa chỉ của phần tử đầu tiên trong mảng, lúc này biến con trỏ sẽ tương đương với một mảng một chiều mà nó đang trỏ tới.

|  |  |
| --- | --- |
|  | Khai báo một mảng a gồm 5 phần tử và một biến con trỏ arr. |
|  | Cả 2 cách đều cho biến con trỏ trỏ đến địa chỉ của ô đầu tiên của mảng |
|  |
|  | Tương đương |
|  | Tương đương |

## CON TRỎ NULL

* Con trỏ NULL hay con trỏ trỏ vào NULL là con trỏ không trỏ vào đâu cả.
* Con trỏ NULL khác với con trỏ chưa được khởi tạo, khi con trỏ được khai báo nhưng chưa được khởi tạo giá trị con trỏ sẽ chứa giá trị rác.

|  |  |
| --- | --- |
|  | * Khởi tạo một biến con trỏ a; * Lúc này do chưa có giá trị nên con trỏ a sẽ có giá trị rác nào đó. |
|  | * Gán biến con trỏ a là NULL. Trong định nghĩa giá trị NULL là 0. |
|  |

## CẤP PHÁT ĐỘNG

* Con trỏ phát huy sức mạnh trong việc cấp phát động. Vậy cấp phát động là gì?
* Để tìm hiểu cấp phát động là gì chúng ta cùng xem một bài toán “Nhập một mảng gồm các số tự nhiên lớn hơn 0, khi nhập số 0 thì kết thúc quá trình nhập rồi sắp xếp các số đã nhập theo thứ tự tăng dần”
* Nhận xét ta nhận thấy bài toán chưa cho cận trên của mảng, dẫn đến tình trạng nếu ta khai báo mảng chỉ có 100 phần tử thì bài toán nhập 101 phần tử thì sao?
* Để giải quyết bài toán ta cần dùng cấp phát động.
* do ta cấp phát động cho biến, mảng,.. Tuy nhiên việc sử dụng một mảng động thông qua việc cấp phát động bằng con trỏ khá phức tạp nên chúng ta sẽ sử dụng kiểu vectơ cũng được xem là kiểu mảng động thông qua con trỏ.

## KIỂU VECTOR (MẢNG ĐỘNG)

### Định nghĩa

* Không giống như [array (mảng)](https://topdev.vn/blog/4-ung-dung-cua-array-from/), chỉ một số giá trị nhất định có thể được lưu trữ dưới một tên biến duy nhất. Vector giống dynamic array (mảng động) nhưng có khả năng tự động thay đổi kích thước khi một phần tử được chèn hoặc xóa tùy thuộc vào nhu cầu của tác vụ được thực thi, với việc lưu trữ của chúng sẽ được vùng chứa tự động xử lý. Các phần tử vector được đặt trong contiguous storage (bộ nhớ liền kề) để chúng có thể được truy cập và duyệt qua bằng cách sử dụng iterator.
* Một số lưu ý
* Bạn không cần phải khai báo kích thước của mảng ví dụ int A[100]…, vì vector có thể tự động nâng kích thước lên.
* Nếu bạn thêm 1 phần tử vào vector, thì nó sẽ tự động tăng kích thước của nó lên để dành chỗ cho giá trị mới này.
* Vector cho bạn biết số lượng các phần tử mà bạn đang lưu trong đó.
* Dùng số phần tử âm vẫn được trong vector ví dụ , , rất tiện trong việc cài đặt các giải thuật.
* Không có vector nào không được sắp xếp trong . Các phần tử vector được đặt trong bộ nhớ liền kề để chúng có thể được truy cập và di chuyển qua các iterator. Trong vector, dữ liệu được chèn vào cuối. Việc chèn một phần tử vào cuối sẽ mất thời gian chênh lệch, vì đôi khi có thể cần mở rộng vector. Việc xóa phần tử cuối cùng chỉ mất thời gian không đổi vì không xảy ra thay đổi kích thước. Chèn và xóa ở đầu hoặc giữa vector là tuyến tính theo thời gian.

### Khai báo

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | * Khai báo một vector (mảng động) có tên là a, với mỗi phần tử có kiểu là int; |

### Các hàm trên Vector

|  |  |  |
| --- | --- | --- |
|  | * Đưa giá trị vào vị trí sau cùng của vectơ. * Nếu giá trị x không phù hợp với kiểu của vecto sẽ bị bỏ qua | * Vì giá trị 2.3 không phải số nguyên nên không được đưa vào trong mảng.   Lúc này mảng a chỉ còn 1 phần tử là 24 |
|  | * Trả về số lượng các phần tử đang được sử dụng của vecto. |
|  | * Xóa đi phần tử cuối cùng của mảng. |
|  | * Tạo ra một mảng gồm phần tử với mỗi phần tử đều có giá trị là . * Nếu mảng a đã có, thì C++ sẽ thay thế mảng đã có bằng mảng mới này. | Mảng a sẽ chỉ còn 2 phần tử là **1 1**  Mảng a sẽ chỉ còn 5 phần tử là  **2 2 2 2 2** |
|  | * Đặt con trỏ đến phần tử đầu tiên của vectơ | Vậy mảng a sẽ là  24 25 **3** 36 27  Vậy mảng a sẽ là  **30** 24 25 3 36 27 |
|  | * Đặt con trỏ đến phần tử cuối của vectơ |
|  | * Chèn giá trị y vào vị trí của con trỏ trỏ đến. |
|  | * Chèn n lần giá trị y vào vị trí con trỏ đang trỏ đến. | Vậy mảng a sẽ là  30 **4 4** 24 25 3 36 27 |
|  | * Chèn từ các phần tử vị trí đến vị trí vào vị trí con trỏ đang trỏ. * Lưu ý. là vị trí do con trỏ trỏ tới * Phần tử  **không được chèn vào.** | Mảng a đang là 24 25 3 6 2 1  Mảng a đang là  24 25 3 6 **24 25**  2 1 |
|  | * Xóa các phần tử tử vị trí con trỏ trỏ đến, đến vị trí thêm y vị trí | Mảng a đang là  24 25 3 3 26 27  Mảng a sẽ là **25 3 3 26 27**  Mảng a đang là 24 25 3 3 26 27  Mảng a sẽ là **25 3 3 26 27**  Mảng a đang là  24 25 3 3 26 27  Mảng a sẽ là **3 3 26 27.** |
|  | * Xóa tất cả các phần tử của mảng. |  |
|  | Thay đổi kích thước của vecto bằng n.  Nếu nhỏ hơn kích thước đang có của vecto thì những phần tử thừa sẽ bị bỏ đi để vecto đó có kích thước đúng bằng n.  Ngược lại nếu lớn hơn kích thước đang có, thì các giá trị mặc định sẽ được thêm vào cho đủ n phần tử ( Ví dụ vecto kiểu int thì thêm vào số 0, còn kiểu char thì sẽ thêm vào giá trị trống. | |
| Duyệt mảng, in các phần tử của mảng | for(int i=0;i<a.size();i++)  cout<<a[i]<<" "; | |

### Hàm sort cho mảng động

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| Cho mảng | | |
|  |  | Sắp xếp từ phần tử đầu đến phần tử cuối |
|  |  |  |
| bool cmp(int A, int B)  {  return A>B;  } |  | Sắp xếp giảm dần mảng a |

### Khai báo Vecto hai chiều

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | Tạo vecto 2 chiều rỗng. |
|  | Bị lỗi. Vì 2 dấu ngoặc sát nhau ( phải có 1 dấu cách) |

# ITERATORS

## Định nghĩa

* Iterator là một con trỏ được sử dụng để đại diện cho một phần tử nào đó.
* Được sử dụng đại diện cho các biến được trỏ đến để thực hiện các thao tác thêm, sửa, xóa, …

## Khai báo

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | Khai báo một vector (mảng động) có tên là a, với mỗi phần tử có kiểu là int; |
|  | Khai báo một con trỏ tên là it được sử dụng đại diện cho một phần tử của một vectơ với mỗi phần tử của vectơ là một int. |
| Nếu con trỏ iterators it này được trỏ tới phần tử đầu tiên của vecto a thì mọi thay đổi liên quan đến con trỏ it cũng sẽ làm thay đổi trên phần tử đầu tiên của vecto a. |

## Cách sử dụng

|  |  |  |
| --- | --- | --- |
|  | | |
| ; | Gán con trỏ iterators it với phần tử đầu tiên của vecto a. |  |
| ; | Gán con trỏ iterators it với phần tử cuối cùng của vecto a. |

# KIỂU MAP

## Định nghĩa

* Kiểu map trong C++ là một tập hợp các phần tử được sắp xếp theo thứ tự cụ thể, mà mỗi phần tử trong đó được hình thành bởi sự kết hợp của một cặp khóa và giá trị (key và Value) với mỗi khóa (key) là duy nhất trong map.
* Trong map, các **khóa** (key) được sử dụng để sắp xếp và xác định **giá trị** (value) tương ứng được liên kết với nó. Mỗi khóa trong map là duy nhất và không được phép trùng lặp. Các giá trị trong map thì có thể trùng lặp, chúng có thể thay đổi giá trị, cũng như là được chèn hoặc xóa khỏi map.
* Về mặt nội bộ, các phần tử trong map luôn được sắp xếp theo khóa của nó theo thứ tự cụ thể một cách nghiêm ngặt, được chỉ ra bởi đối tượng so sánh nội bộ của map. Nếu ta thêm các phần tử mới không theo thứ tự cụ thể vào một map, chúng sẽ tự động sắp xếp lại theo khóa trước khi được lưu trữ nội bộ.

## Khai báo kiểu map

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **Map <char,int> a;** | * Khai báo một map tên a gồm 2 kiểu * Key là kiểu char * Value kiểu int; |

## Gán giá trị kiểu map

tên\_biến\_map [key]=value;

|  |  |
| --- | --- |
|  |  |

## Truy cập giá trị trong map

* Để truy cập đến một phần tử trong map thông qua phần tử key của phần tử đó. Nếu key đó có trong map thì giá trị Value của phần tử đó sẽ được trả về, ngược lại nếu phần tử đó không có trong map thì giá trị trả về là 0.

|  |  |  |
| --- | --- | --- |
| Map <char,int> a;  cout<<a[‘a’]<<endl;  Cout<<a[‘b’]<<endl;  Cout<<a[‘c’]<<endl; | **Kết quả là**  **2**  **3**  **0** |  |

## Liệt kê các phần tử của map

* Để liệt kê các phần tử của map, ta sử dụng con trỏ iterator để duyệt qua từng phần tử của map
  + Để truy xuất phần tử key ta sử dụng toán tử
  + Để truy xuất phần tử value ta sử dụng toán tử

|  |  |
| --- | --- |
| map <char,long> a;  map <char,long>::iterator it;  for (it=a.begin();it!=a.end();it++)  cout<<it->first<<" "<<it-second<<endl; |  |
|  |

## Một số hàm xử lý trên map

|  |  |  |
| --- | --- | --- |
|  | Trả về kích thước của map | ;  kết quả cho là **3** |
|  | Chèn thêm một cặp giá trị vào map. Lưu ý giá trị key khi được chèn sẽ tự được sắp xếp | ; |
|  | Xóa phần tử có key | ; |
|  | Xóa phần tử con trỏ iterator đang chỉ tới | ; |
|  | Trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy iterator trỏ về “end” của map | ;  Lúc này con trỏ it sẽ chỉ đến phần tử b. (lúc này it->second trả về giá trị 3) |
|  | Trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí “end” của map | ;  Lúc này con trỏ it sẽ chỉ đến phần tử b. (lúc này it->second trả về giá trị 3) |
|  | Trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí “end” của map. | Lúc này con trỏ it sẽ chỉ đến phần tử c. (lúc này it->second trả về giá trị 4) |

# KIỂU SET

## Định nghĩa

* Set là một loại associative containers để lưu trữ các phần tử không bị trùng lặp (unique elements), và các phần tử này chính là các khóa (keys).
* Khi duyệt set theo iterator từ begin đến end, các phần tử của set sẽ tăng dần theo phép toán so sánh. Mặc định của set là sử dụng phép toán less ( tăng dần), bạn cũng có thể viết lại hàm so sánh theo ý mình.
* Set được thực hiện giống như cây tìm kiếm nhị phân (Binary search tree).
* Set là một kiểu tập hợp các giá trị khác nhau và các giá trị một khi đã ở trong set sẽ không thể bị thay đổi (const), tuy nhiên vẫn có thể thêm hoặc xóa phần tử, vậy nên rất hữu ích trong các dạng bài cần lược bỏ giá trị giống nhau. Đồng thời theo mặc định, các giá trị của set sẽ tự động được sắp xếp theo thứ tự tăng dần.

|  |  |
| --- | --- |
| Mảng đề cho | Khi đưa vào set |
| 1 3 2 4 | 1 2 3 4 |
| 1 1 3 3 2 | 1 2 3 |
| 1 1 1 1 | 1 |

## Khai báo

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | Khai báo một set kiểu nguyên có tên là a. Set a sẽ chứa các số nguyên khác nhau |

## Các vấn đề liên quan đến set

|  |  |
| --- | --- |
| Ghi dữ liệu từ mảng vào set | {    } |
| * Để liệt kê các phần tử của set, ta sử dụng con trỏ iterator để duyệt qua từng phần tử của set |  |

## Các hàm liên quan đến set

|  |  |  |
| --- | --- | --- |
|  | Trả về độ dài của set |  |
|  | Thêm phần tử vào set |  |
|  | Thêm phần tử từ vị trí của con trỏ đang trỏ đến | Lúc này set a được thêm vào 1 phần tử là số 6  Lúc này set a được thêm vào 3 phần tử là số 6 7 8  Lúc này set a được thêm vào 2 phần tử là số 8 9 |
|  | Xóa phần tử từ set |  |
|  | Xóa phần tử ở vị trí của con trỏ đang trỏ đến  Lưu ý. Con trỏ phải đang trỏ đến vị trí của set. | 1 2 3 4  Xóa 1 phần tử là phần tử it trỏ đến ( phần tử đầu tiên). Vậy a là 2 3 4 |
|  | xóa hết tất cả các phần tử của set |  |
|  | Trả về số lần xuất hiện của khóa trong container. Nhưng trong set, các phần tử chỉ xuất hiện một lần, nên hàm này có ý nghĩa là sẽ return 1 nếu khóa có trong container, và 0 nếu không có |  |
|  | Trả về it trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy iterator trỏ về vị trí “end” của set |  |
|  | Trả về it đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí “end” của set |  |
|  | trả về true (1) nếu set không có phần tử, false (0) nếu có |  |

## Set được sắp xếp theo thứ tự giảm dần

|  |  |
| --- | --- |
| Tạo một hàm trước khi khai báo set | struct cmp  {  bool operator () (int a, int b)  {  Return a>b;  }  } ; |
| Khai báo set |  |
| Lưu ý lúc khai báo biết con trỏ cần cùng kiểu với set |  |

# KIỂU PAIR

## Định nghĩa

* Kiểu pair trong C++ đơn giản chỉ là tập hợp 2 phần tử (first và second).
* Đây cũng là kiểu dữ liệu được dùng để chứa mỗi cặp khóa và giá trị (key, value) trong map.

## Khai báo

using namespace std;

pair<first\_type, second\_type> tên\_biến;

|  |  |
| --- | --- |
| **pair<char, int> a;** | * Khai báo một pair gồm 2 kiểu * First là kiểu char * Second là kiểu int |

## Gán và truy cập giá trị kiểu pair

tên\_biến\_pair.first = value;

tên\_biến\_pair.second = value;

|  |  |
| --- | --- |
| pair<char, int> a;  a.first = ‘m’;  a.second = 3;  cout << a.first << endl;  cout << a.second << endl; | **Kết quả là**  **m**  **3** |

# PHÉP TOÁN TRÊN BIT

## PHÉP TOÁN LOGIC

|  |  |  |  |
| --- | --- | --- | --- |
| **Phép toán** | **C** | **Giá trị trả về** | **Ví dụ** |
| Phép Và (AND) | && | * Phép chỉ trả về giá ***ĐÚNG*** (1) khi a và b đều có giá trị đúng (1). |  |
| Phép Hoặc (OR) | || | * Phép chỉ trả về giá ***SAI (0)*** khi a và b đều có giá trị sai (0). |  |
| Phép Hoặc loại trừ (XOR) | ^ | * Phép chỉ trả về giá ***SAI (0)*** khi a và b có cùng giá trị, nhận giá trị ***ĐÚNG (1)*** khi a và b khác giá trị. |  |
| Phép phủ định (NOT) | ! | * Phép trả về giá đối của a. |  |