**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

🙤🙧🟍🙥🙦



**BÁO CÁO ĐỒ ÁN**

**Môn học: CS111.L21**

**GV: Trịnh Quốc Sơn**

**CHỦ ĐỀ:** **VARIABLE IN PROGRAMING LANGUAGE**

**==||==**

**SVTH:**

**19522274- Hồ Thịnh**

**19522351- Trần Trung Tín**

**MỤC LỤC**

[**I.** **Giới thiệu:** 3](#_Toc76152699)

[**II.** **Các nội dung:** 5](#_Toc76152700)

[**1.** **Khái niệm:** 5](#_Toc76152701)

[**2.** **Các thuộc tính của biến:** 5](#_Toc76152702)

[**a.** **Tên biến (name/ identifier):** 5](#_Toc76152703)

[**b.** **Loại (type):** 5](#_Toc76152704)

[**c.** **Giá trị (value):** 6](#_Toc76152705)

[**d.** **Địa chỉ (address):** 6](#_Toc76152706)

[**e.** **Phạm vi:** 7](#_Toc76152707)

[**f.** **Thời gian sống (life time):** 8](#_Toc76152708)

[**3.** **Phân loại biến:** 8](#_Toc76152709)

[**a.** **Biến cục bộ (local variable):** 8](#_Toc76152710)

[**b.** **Biến toàn cục (global variable):** 9](#_Toc76152711)

[**c.** **Biến thanh ghi (register variable):** 10](#_Toc76152712)

[**d.** **Biến tĩnh (static variable):** 10](#_Toc76152713)

[**III.** **Các thao tác với biến:** 11](#_Toc76152714)

[**1.** **Khởi tạo:** 11](#_Toc76152715)

[a. **Khai báo:** gồm khai báo tên biến và kiểu dữ liệu 11](#_Toc76152716)

[**b.** **Gán giá trị:** 13](#_Toc76152717)

[**c.** **Ví dụ:** 13](#_Toc76152718)

[**2.** **Tính toán trên biến:** 14](#_Toc76152719)

[**a.** **Toán tử số học (Arithmetic Operator):** 14](#_Toc76152720)

[**b.** **Toán tử quan hệ (Relative Operator):** 14](#_Toc76152721)

[**c.** **Toán tử logic (Logical Operator):** 15](#_Toc76152722)

[**IV.** **Các lưu ý khi sử dụng biến:** 15](#_Toc76152723)

[**1.** **Biến phải được khai báo:** 15](#_Toc76152724)

[**2.** **Biến tham gia tính toán phải cùng kiểu:** 16](#_Toc76152725)

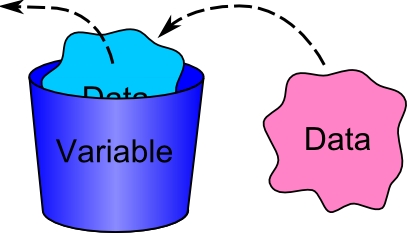
[**3.** **Gán mới xóa cũ:** 16](#_Toc76152726)

[**V.** **Tài liệu tham khảo:** 17](#_Toc76152727)

1. **Giới thiệu:**

Để chương trình luôn biết chính xác dữ liệu cần xử lí ở vị trí nào trong quá trình hoạt động, các ngôn ngữ lập trình cung cấp một công cụ rất quan trọng cho người viết chương trình. Đó là biến nhớ, hay được gọi ngắn gọn là biến.

Trong hầu hết các ngôn ngữ lập trình biến hoạt động giống như một nơi để chứa hoặc là một cái tên mà ở đó chứa những giá trị hoặc những nội dung quan trọng khác có thể sử dụng ở nhiều vị trí khác nhau trong code của bạn.



Trong lập trình phần mềm, biến là tên được sử dụng để chứa một hoặc nhiều giá trị. Thay vì lặp lại các giá trị này ở nhiều vị trí trong code của bạn, biến giữ kết quả của phép tính, lệnh gọi cơ sở dữ liệu, kết quả của truy vấn cơ sở dữ liệu hoặc giá trị khác. Thay vì nhiều phép tính hoặc tải nhiều lệnh gọi cơ sở dữ liệu để truy xuất cùng một dữ liệu, biến được lưu trữ trong bộ nhớ máy tính một lần và được sử dụng ở bất cứ nơi nào cần thiết trong code của bạn.

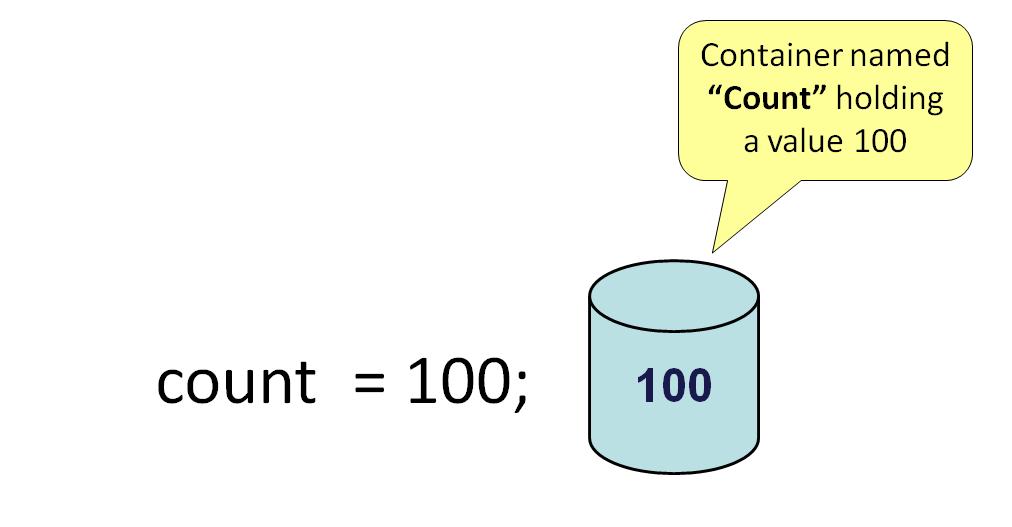
Lấy ví dụ như chúng ta có biến x = 1+1+1+1+1 hoặc x = Function(). Nếu như trường hợp chúng ta không sử dụng/khai báo cái biến “x” ở trên thì khi lập trình, giả sử như chúng ta sử dụng các giá trị cụ thể hoặc một giá trị trả về(như Function() ở trên) ở N chỗ thì chúng ta có phải chúng ta phải gọi lại N lần “1+1+1+1+1” hoặc “Function” không? Điều này làm cho code chúng ta dài ra và có thể gây sai sót, nhầm lẫn.

* Vậy tóm lại, biến cụ thể là gì, khai báo sử dụng biến như thế nào, công dụng/ tầm quan trọng của nó ra sao? Chúng ta cùng tìm hiểu trong bài thuyết trình.

1. **Các nội dung:**
2. **Khái niệm:**

Nói đơn giản, biến là một cái tên tượng trưng (hoặc được tham chiếu tới) thông tin. Gọi nó là biến bởi vì nó có thể thay đổi trong suốt thời gian chương trình chạy nhưng các thao tác trên biến vẫn giữa nguyên.

Các biến trong chương trình máy tính tương tự như “Thùng” hoặc “Phong bì” nơi thông tin có thể được duy trì và tham chiếu.( ảnh minh họa H2.1)



***H2.1. Minh họa cho biến***

1. **Các thuộc tính của biến:**
2. **Tên biến (name/ identifier):**

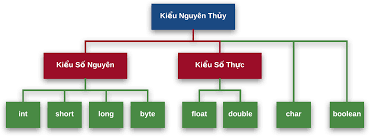
Đại diện cho “tiêu đề” của thông tin/ dữ liệu được lưu trong biến.

Đây là thuộc tính khá quan trọng vì nó giúp chúng ta đối chiếu dữ liệu của chúng ta khi code, vì thế nên chúng ta cần phải đặt tên biến đúng quy tắc, tránh quá ngắn gọn/ tối nghĩa.

1. **Loại (type):**

Tức là kiểu dữ liệu của biến. Bao gồm các kiểu cơ bản như là int, str, float, double,…. Tùy theo ngôn ngữ lập trình.

Vd như trong Java: (ảnh H2.2.b)

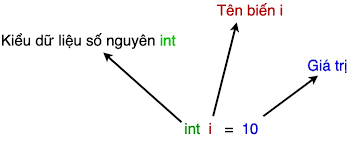


***H2.2.b. Kiểu dữ liệu trong Java***

1. **Giá trị (value):**

Là dữ liệu mà biến lưu trữ.

Giá trị của biến có thể là một giá trị cụ thể (một số, một chuỗi,…) hoặc là một địa chỉ trong bộ nhớ. (ảnh minh họa H2.2.c)



***H2.2.c. Minh họa giá trị của biến***

Trong ví dụ a:=a+1 thì biến a bên trái chứa giá trị của a bên phải sau khi đã cộng thêm giá trị 1, còn a bên phải là tham chiếu đến địa chỉ của biến a trước đó.

1. **Địa chỉ (address):**

Là vị trí của biến trong bộ nhớ. Một chương trình được thực thi bởi một bộ xử lý có hai phần chính code và dữ liệu. Phần code là code của bạn đã viết và phần dữ liệu chứa các biến bạn đang sử dụng trong chương trình.

Tất cả code và biến được tải vào bộ nhớ (thường là RAM) và bộ xử lý thực thi code từ đó. Mỗi phân đoạn trong bộ nhớ có một địa chỉ, cho dù nó chứa code hay biến đó là cách để bộ xủ lý truy cập code và các biến.

Chúng ta có thể kiểm tra địa chỉ của một biến thông qua các tóan tử được định nghĩa trong từng ngôn ngữ lập trình. Vd như trong C/C++ sử dụng toán tử “&”

Một biến có thể có các địa chỉ khác nhau tại các thời điểm khác nhau trong quá trình thực thi.

**Ví dụ:** Các biến cục bộ của chương trình con có thể có các địa chỉ khác nhau bởi các lệnh gọi khác nhau.

***Code minh họa:***

#include<iostream>

using namespace std;

int main(){

int a;

cout<<&a;

return 0;

system(“pause”);

}

Kết quả thu được từ chương trình: 0x61febc

1. **Phạm vi:**

Mọi biến đều có một khu vực để hoạt động, ngoài khu vực này biến không còn giữ giá trị trước đó của nó nữa và xem như không tồn tại. Khu vực hoạt động này gọi là “phạm vi của biến”.

Tùy theo phạm vi của biến mà ta chia biến thành 2 loại:

* Biến cục bộ (local variable)
* Biến toàn cục (global variable)

1. **Thời gian sống (life time):**

Thời gian từ lúc khai báo đến lúc chương trình rời khỏi phạm vi của biến.

Trong khoảng thời gian đó ta có thể truy xuất giá trị của biến, lifetime phụ thuộc vào cách nó được định nghĩa và khai báo.

1. **Phân loại biến:**

Dựa vào phạm vi truy xuất biến mà ta chia biến thành hai loại cơ bản (a và b), ngoài ra còn 2 loại khác ngoại lệ.

1. **Biến cục bộ (local variable):**

Là các biến được khai báo trong một khối code thuộc lớp lưu trữ tự động. Biến cục bộ chỉ tồn tại và có thể sử dụng trong khối code đó khi khối code đó được thực thi.

Các khối code ở đây có thể là: thân của một hàm,thân của vòng lặp,….

***Code minh họa:***

#include<stdio.h>

Using namespace std;

int main(){

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

printf(“\nExample 1”);

}

Print(“%d”, i);

Return 0;

}

Kết quả nhận được: eror: “i” undeclared (first use in this function print(“%d”,i)

1. **Biến toàn cục (global variable):**

Các biến được khai báo ngoài các hàm thì được gọi là biến toàn cục. Biến toàn cục có thể truy xuất ở mọi hàm trong chương trình. Tồn tại cho đến khi chương trình kết thúc.

***Code minh họa:***

void display();

int n = 5;

int main(){

++n;

display();

return 0;

}

void display(){

++n;

printf(“n=%d”, n);

}

Kết quả nhận được: n=7

1. **Biến thanh ghi (register variable):**

Các biến có tính chất giống như biến cục bộ nhưng lưu trong thanh ghi. Truy xuất nhanh hơn.

Thường được dùng khi làm việc với hệ thống nhúng hay các chương trình cần tối ưu hóa về hiệu năng. Nhưng mà các chương trình này phải được tối ưu trước nếu không thì việc dùng biến này cũng không có ý nghĩa nhiều về mặt tối ưu.

Ví dụ khai báo biến thanh ghi trong C/C++: register int x = 5;

1. **Biến tĩnh (static variable):**

Là biến tồn tại cho đến khi chương trình kết thúc dù cho nó là biến toàn cục hay biến cục bộ.

Biến tĩnh luôn tham chiếu đến môi trường cấp cao nhất của nó. Biến tĩnh cũng giúp tạo mã mô-đun dễ dàng hơn nhiều vì lập trình viên có thể tìm ra phạm vi chỉ bằng cách nhìn vào code. Ngược lại, phạm vi hoạt động đòi hỏi người lập trình phải lường trước tất cả các bối cảnh động có thể xảy ra.

Trong hầu hết các ngôn ngữ lập trình bao gồm C, C++ và java, các biến luôn nằm trong phạm vi tĩnh. Tức là ràng buộc của một biến có thể được xác định bằng văn bản chương trình và độc lập với ngăn xếp lệnh gọi hàm thời gian chạy.

***Code minh họa:***

void display();

int main(){

display();

display();

}

void display(){

static int c =1;

c+=5;

print(“%d”, c);

}

Kết quả nhận được: 6 11

Để tổng hợp trong phạm vi tĩnh, trình biên dịch đầu tiên tìm kiếm trong khối code hiện tại, sau đó trong các biến toàn cục, sau đó trong các phạm vi nhỏ hơn liên tiếp.

1. **Các thao tác với biến:**
2. **Khởi tạo:**
3. **Khai báo:** gồm khai báo tên biến và kiểu dữ liệu

Khai báo tên biến, tên biến phải tuân theo qui tắc đặt tên của ngôn ngữ lập trình.

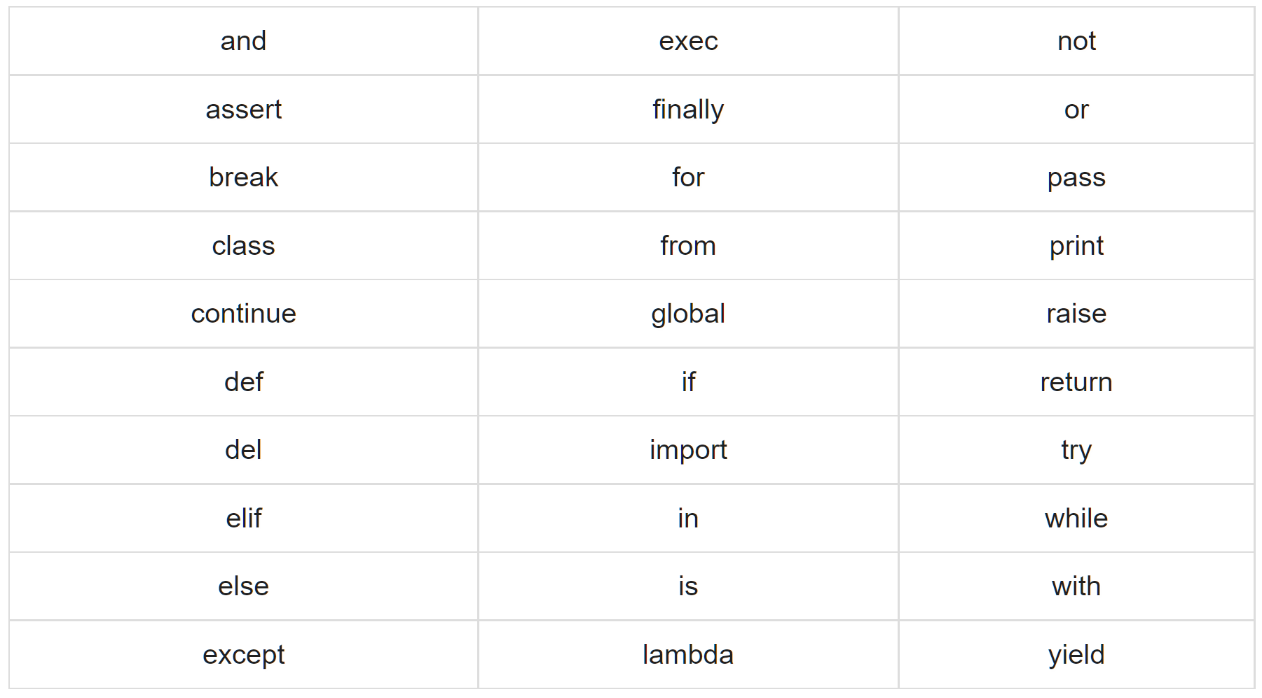
Quy tắc đặt tên:

* Gồm các kí tự chữ, số, dấu gạch dưới,..
* Không được bắt đầu bằng 1 chữ số.
* Không được trùng với những từ định nghĩa sẵn.
* Không chứa khoảng trắng và các kí tự đặc biệt.

Ví dụ về một số từ định nghĩa sẵn trong một số ngôn ngữ lập trình:

* Khai báo dữ liệu biến

***H3.1.a-1. Keyword in C/C++***



***H3.1.a-2. Keyword in Python***

Tùy theo ngôn ngữ lập trình mà cách khai báo của chúng khác nhau. Như trong ngôn ngữ PHP, khi khai báo biến thì tên biến sẽ bắt đầu bằng dấu “$” theo sau đó mới đến tên mình muốn đặt. Còn đối với ngôn ngữ lập trình Ruby thì trong tên biến có thể bắt đầu bằng một số kí tự đặt biệt, các kí tự này được gọi là sigil, các kí tự sigil dùng để chỉ ra phạm vi hoạt động của biến (“$” cho biến toàn cục, “@” cho biến instance,…).

Chúng ta có thể khai báo biến dưới dạng tường minh hoặc ngầm định (không tường minh). Một lệnh khai báo tường minh trong đa số các ngôn ngữ lập trình đều bắt người lập trình phải chỉ rõ kiểu dữ liệu cho một biến. Người lập trình có thể gán giá trị cho biến trong câu lệnh khai báo, nếu kiểu của giá trị được gán khác với kiểu của biến thì chương trình sẽ tự động ép kiểu cho giá trị và gán lại cho biến. Khai báo biến kiểu ngầm định (khai báo biến không tường minh) là biến được khai báo mà không cần chỉ ra kiểu dữ liệu , kiểu dữ liệu hoặc phạm vi của biến sẽ được chương trình xác định dựa vào biểu thức được gán khi khai báo biến.

1. **Gán giá trị:**

Đa phần thì việc gán giá trị cho biến ở các ngôn ngữ lập trình đa phần là giống nhau, sử dụng toán tử “=”.

1. **Ví dụ:**

* C/C++/Java: <kiểu dữ liệu> <tên biến> = <giá trị>.
* int a = 10;
* float square = 14.4;
* Python: <tên biến> = <giá trị>
* count = 10
* str = ”Python”
* PHP: $<tên biến> = <giá trị>
* $hello = ”Hello World”
* Ruby: <sigil><tên biến> = <giá trị>
* $name = ”Tin”
* @temp = 1

1. **Tính toán trên biến:**

Các tính toán trên biến thường dựa trên các toán tử cơ bản (operator). Mỗi ngôn ngữ lập trính sẽ có một số điểm khác biệt trong các toán tử. Sau đây sẽ là phần liệt kê các toán tử cơ bản mà ở các ngôn ngữ lập trình.

1. **Toán tử số học (Arithmetic Operator):**

Các toán tử số học thông thường là các phép cộng trừ nhân chia chúng ta hay sử dụng để tính toán.

Tuy nhiên trong một số ngôn ngữ lập trình thì sẽ có một số điểm khác biệt. điển hình như python chúng ta có thể sử dụng toán tử cộng để cộng hai chuỗi lại với nhau(vd),…

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| + | Cộng hai toán hạng | C/C++: a=10; b=1; c=a+b; 🡪 c=11  Python: s1=py s2=thon s3 = s1+s2  🡪 s3=python |
| - | Trừ hai toán hạng | a=100 b=20 c=a-b 🡪 c=80 |
| \* | Nhân hai toán hạng | d=a\*b 🡪 d=2000 |
| / | Chia lấy nguyên | e=a/b 🡪 e= 5 |
| % | Chia lấy dư | f= a%b 🡪 f=0 |

1. **Toán tử quan hệ (Relative Operator):**

Toán tử quan hệ cơ bản gồm các phép so sánh như là lớn hơn, nhỏ hơn, lớn hơn bằng nhỏ hơn bằng, bằng, khác.

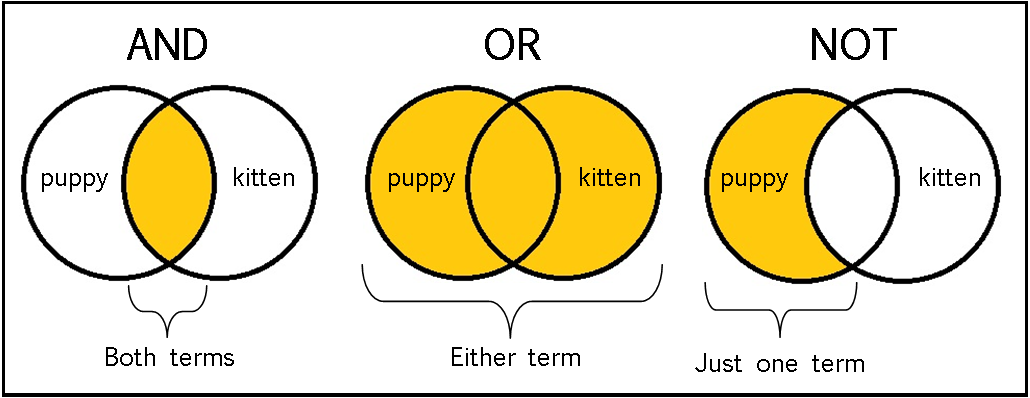
Trong các ngôn ngữ lập trình thì các toán tử này trả về giá trị boolean tức là True/False.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| p | q | p<q | p<=q | p>q | p>=q | p==q | p!=q |
| 0 | 1 | True | True | False | False | False | True |
| 1 | 0 | False | False | True | True | False | True |
| 3 | 3 | False | True | False | True | True | False |
| 2 | 6 | True | True | False | False | False | True |

1. **Toán tử logic (Logical Operator):**

Các toán tử logic bao gồm các phép “and”, “or”, “not”. (H3.2.c)

Dùng để kết nối các điều kiện hay các quan hệ lại vói nhau để tạo ra các ràng buộc.

* 

***H3.2.c. Các phép toán logic***

1. **Các lưu ý khi sử dụng biến:**
2. **Biến phải được khai báo:**

***Code minh họa:***

#include<iostream>

using namespace std;

int main():{

cout<<mssv;

return 0;

}

Kết quả nhận được: error: ‘mssv’ was not declared in this scope cout<<mssv;

1. **Biến tham gia tính toán phải cùng kiểu:**

***Code minh họa:***

#include<iostream>

#include<string>

using namespace std;

int main():{

string name;

int b=19522274;

cout<<”What’s your name?”;

getline(cin,name);

cout<<”Hello”<<name+b;

return 0;

system(“pause”);

}

Kết quả nhận được: error: no match for operator’+’

1. **Gán mới xóa cũ:**

***Code minh họa:***

#include<iostream>

int main(){

int a=10;

a=1000;

print(“%d”,a);

return 0;

}

Kết quả nhận được: 1000

1. **Tài liệu tham khảo:**
2. Biến cục bộ và biến toàn cục: <https://nguyenvanhieu.vn/bien-cuc-bo-va-bien-toan-cuc/>
3. Phạm vi của biến: <https://cpp.daynhauhoc.com/1/7-pham-vi-cua-bien/>
4. Variables and Data Types: <https://ftms.edu.my/v2/wp-content/uploads/2019/02/PROG0101_CH04.pdf>
5. Computer Programming – Operators: <https://www.tutorialspoint.com/computer_programming/computer_programming_operators.htm>
6. Static and Dynamic Scoping: <https://www.geeksforgeeks.org/static-and-dynamic-scoping/>