

Bài 5: Các công cụ để tính toán

Tôi đã nói vấn đề này ở phần trước: máy tính của các bạn cũng giống như một cỗ máy tính toán khổng lồ.

Khi mà bạn muốn nghe nhạc, xem film hay chơi game điện tử, máy tính của bạn không làm điều gì khác hơn việc tính toán

Ở phần này tôi sẽ hướng dẫn cho các bạn thực hiện phần lớn những phép tính mà máy tính có thể thực hiện. Các bạn đã biết thế nào là biến số, và ý tưởng là chúng ta sẽ thực hiện những tính toán trên các biến số đó: hãy cho một biến số giá trị nào đó, sau đó hãy nhân nó lên, và giá trị nhận được sẽ đưa vào một biến số khác .v.v...

Kể cả khi bạn không phải là một fan của toán học, các bạn cũng nên biết nội dung của phần hướng dẫn này.

Sự thật là nếu bạn không biết cách thực hiện phép cộng, bạn không thể nào có thể thực hiện việc lập trình. 😊

Nội dung bài học này sẽ gồm:

Những phép toán cơ bản

- Những cách viết rút gọn
 - Thư viện toán học
 - TRẮC NGHIỆM KIẾN THỨC.
-

Những phép toán cơ bản

Những gì chúng ta sẽ tìm hiểu không có gì khác ngoài những phép toán bình thường, máy tính của bạn là một cỗ máy tính toán đơn giản vì nó chỉ có thể làm những phép toán:

- Phép cộng
- Phép trừ
- Phép nhân
- Phép chia
- Phép module (Tôi sẽ giải thích nếu như bạn không biết nó là gì)

Nếu như bạn muốn sử dụng những phép toán phức tạp hơn (bình phương, lũy thừa, logarit, và một số những phép toán khác mà bạn thích) thì bạn phải lập trình ra nó, có nghĩa là bạn sẽ hướng dẫn máy tính làm cách nào để thực hiện những phép toán đó.

May mắn là bạn có thể mượn những quyển sách này trong thư viện của ngôn ngữ C: có rất nhiều function toán học được viết sẵn. Bạn không cần phải viết lại chúng nữa 😊

Chúng ta bắt đầu với phép cộng.

Để thực hiện một phép cộng, chúng ta sử dụng kí tự + (đúng không nhỉ? 🤖)

Bạn cần phải đưa kết quả nhận được vào trong một biến số. Chúng ta sẽ sử dụng biến số «ketqua» dạng int để thực hiện phép tính:

C code:

```
int ketqua = 0;  
ketqua = 5 + 3;
```

Bạn không cần phải có một đầu óc pro về tính toán để có thể hiểu rằng « ketqua » sẽ mang giá trị 8 sau khi ta chạy chương trình. 😊

Chắc chắn là, màn hình sẽ không hiển thị bất cứ điều gì nếu như ta chỉ sử dụng đoạn mã trên, hãy thêm vào một function printf.

Trên màn hình sẽ cho ta :

Console:

```
Ket qua = 8
```

Và đó là phép cộng.

Và với những phép toán khác, cũng tương tự, chỉ cần thay đổi kí tự tính toán :

- Phép cộng: +
- Phép trừ: -
- Phép nhân: *
- Phép chia: /
- Module: %

Nếu như bạn đã từng tính toán trên máy tính của bạn thì chắc hẳn là bạn biết những kí tự này .

"Dấu trừ" tương ứng với dấu gạch ngang "-", "dấu nhân" tương ứng dấu sao "*", "dấu chia" tương ứng dấu slash "/" và "module" sẽ tương ứng với dấu phần trăm "%".

Không có gì đặc biệt khó khăn để sử dụng được chúng. Ở hai phép tính cuối cùng (phép chia và module) có một số khác biệt nhỏ, chúng ta sẽ nói rõ hơn.

Phép chia

Phép chia hoạt động bình thường trên máy tính nếu như không có số dư. Ví dụ, $6 / 3$ bằng 2, máy tính của bạn sẽ cho một kết quả đúng, không hề sai sót.

Bây giờ chúng ta thử thực hiện một phép chia có dư như $5 / 2$.

$5 / 2$ theo như ta tính sẽ cho kết quả là 2.5.

Tuy nhiên! hãy xem kĩ kết quả của đoạn mã này :

C code:

```
int ketqua=0;
ketqua = 5 / 2;
printf ("5 / 2 = %d",ketqua);
```

Console:

```
Ket qua = 2
```

Có một vấn đề lớn ở đây, chúng ta yêu cầu máy tính thực hiện $5 / 2$, chúng ta chờ đợi kết quả là 2.5, nhưng máy tính cho kết quả là 2 !

Có một cái gì đó kì lạ ở đây. Không lẽ máy tính của chúng ta bị ngu ở phép tính này ?

Thực sự, khi máy tính nhận được những số 5 và 2, máy tính của bạn thực hiện phép tính với dạng số tự nhiên, điều đó có nghĩa là máy tính đã làm tròn kết quả, nó chỉ giữ lại phần nguyên (số 2).

🤔 Tôi biết rồi! tại vì biến số « ketqua » mà chúng ta khai báo có dạng int! nếu nó ở dạng double thì nó sẽ chứa một số thực!

Cũng không phải. 😊

Hãy thử lại đoạn mã trên nhưng chúng ta đổi biến số « ketqua » thành double, nó cũng chỉ hiển thị kết quả là 2.

Nếu như ta muốn máy tính hiển thị một kết quả chính xác, chúng ta phải biến đổi những số 5 và 2 đó về dạng số thực, nghĩa là 5.0 và 2.0 (Đối với chúng ta, chúng giống nhau nhưng đối với máy tính, những số thực khác với số tự nhiên và nó sẽ thực hiện phép toán với dạng số thực) :

C code:

```
double ketqua = 0;  
ketqua = 5.0 / 2.0;  
printf ("5 / 2 = %lf", ketqua);
```

Console:

```
Ket qua = 2.500000
```

Mặc dù nó hiển thị một dãy những số 0 ở phía sau nhưng kết quả này hoàn toàn chính xác.

Đặc điểm này của phép chia rất quan trọng, bạn cần chú ý:

$$5 / 2 = 2$$

$$10 / 3 = 3$$

$$4 / 5 = 0$$

Những số trong phép tính phải thuộc dạng số thực :

$$5.0 / 2.0 = 2.5$$

$$10.0 / 3.0 = 3.33333$$

$$4.0 / 5.0 = 0.8$$

Thực tế, nếu ta thực hiện phép tính « $5 / 2$ », ở dạng số tự nhiên. Máy tính sẽ trả lời câu hỏi: « Trong 5, có bao nhiêu lần 2 ? ». Câu trả lời là 2 lần. Giống như vậy, « trong 10, có bao nhiêu lần 3 ? » đáp án là 3 lần ».

Nhưng làm sao để giữ lại số dư của phép chia ?

Và đây chính là công việc của phép module. 😊

Phép module

Module là một phép toán cho ta **số dư của một phép chia**. Module ít được biết đến hơn các phép toán cơ bản còn lại, nhưng nó giúp máy tính có thể thực hiện đầy đủ tất cả những phép toán với những số tự nhiên. Module được biểu thị bởi kí tự %.

Một số ví dụ :

- $5 \% 2 = 1$
- $14 \% 3 = 2$
- $4 \% 2 = 0$

Module $5 \% 2$ là số dư của $5 / 2$, bằng 1. Máy tính tính toán như sau $5 = 2 * 2 + 1$ (module cho kết quả 1).

Tương tự, $14 \% 3$, tính như sau $14 = 3 * 4 + 2$ (module cho kết quả 2).

Cuối cùng, $4 \% 2$, phép chia không có dư nên module sẽ cho kết quả là 0.

Và không nói gì nhiều hơn về module, tôi chỉ giải thích với những bạn nào chưa biết. 😊

Và tôi có thêm một tin tốt nữa, chúng ta đã biết tất cả những phép toán cơ bản. 😊

Những tính toán sử dụng biến số

Vấn đề này khá lý thú, bạn đã biết cách sử dụng các phép toán cơ bản, chúng ta hãy luyện tập bằng cách tính toán với nhiều biến số, bạn có thể thực hiện :

C code:

```
ketqua = so1 + so2;
```

Đoạn mã này tính tổng của các biến số so1 và so2, sau đó kết quả sẽ đưa vào biến số ketqua. Ah! tôi có một ý tưởng cho bạn đây, bây giờ bạn có khả năng thực hiện một công cụ tính toán nhỏ. Chắc mà, tôi bảo đảm các bạn có thể làm được ! 😊

Hãy tưởng tượng một chương trình đòi hỏi người sử dụng nhập vào 2 số hạng. Đó là giá trị của 2 biến số, sau đó bạn hãy thực hiện tổng của 2 biến số này, kết quả là giá trị của biến số « ketqua ». Sau đó bạn hiển thị nó lên màn hình. Chương trình này khá đơn giản, hãy luyện tập với nó. 😊

Đây là kết quả:

C code:

```
int main(int argc, char *argv[])
{
    int ketqua = 0, so1 = 0, so2 = 0;

    // Chúng ta yêu cầu người sử dụng nhập vào giá trị của so1 và so2 :

    printf ("Gia tri so thu 1 : ");
    scanf ("%d", &so1);
    printf ("Gia tri so thu 2 : ");
    scanf ("%d", &so2);

    // Thực hiện phép tính :

    ketqua = so1 + so2;

    // Và ta hiển thị lên màn hình :

    printf ("%d + %d = %d\n", so1, so2, ketqua);

    return 0;
}
```

Console:

```
Gia tri so thu 1 : 30
Gia tri so thu 2 : 25
30 + 25 = 55
```

Các bạn có thể thử chương trình này với bất kì số hạng nào (nếu số hạng đó không nằm ngoài giới hạn của biến số dạng int), máy tính của bạn sẽ hoàn thành phép tính với vận tốc ánh sáng 😊.

Tôi khuyên bạn hãy tạo thêm các chương trình sử dụng những phép toán khác (phép trừ, phép nhân...)

Bạn hoàn toàn có thể thực hiện với nhiều biến số hơn nữa, không vấn đề gì nếu như ta dùng 3 biến số cùng lúc:

C code:

```
ketqua = so1 + so2 + so3;
```

Phương pháp viết rút gọn

Như tôi đã nói với các bạn, chúng ta không còn phép toán nào mới để học nữa, đó là tất cả. 😊

Chúng ta không cần thêm các phép toán nào khác vì chúng ta có thể tạo ra chúng.

Thật khó tin nếu như tôi nói rằng một game 3D chỉ sử dụng không gì khác ngoài phép cộng và phép trừ. Nhưng đó hoàn toàn là sự thật. 😊

Và cũng giống như dưới đây, trong C có những phương pháp giúp ta viết ngắn gọn những phép toán.

Tại sao phải dùng phương pháp viết rút gọn ?

Tại vì chúng ta thường xuyên phải lặp lại một phép toán. Bạn sẽ hiểu rõ hơn những gì tôi nói. Với cái gọi là increment.

Incrementing (Phương pháp tăng giá trị)

Bạn sẽ thấy rằng bạn sẽ phải thường xuyên tăng giá trị một biến số lên 1. Ví dụ như biến số của bạn là « sohang ». Và ta làm như sau:

C code:

```
sohang = sohang + 1;
```

Và chuyện gì diễn ra ở đây? Chúng ta lấy sohang + 1, và sau đó chúng ta đưa giá trị nhận được vào chính « sohang ». Nếu số hạng này có giá trị ban đầu là 4 thì nó sẽ thành 5, nếu là 8 thì sẽ thành 9...

Thao tác này sẽ được sử dụng lại. Những nhà lập trình đều là những người đặc biệt lười biếng, hầu như họ đều không hứng thú với việc viết lại một cái nào đó đã có (việc này khá mệt nhọc!). 😊

Và họ đã tạo ra một cách viết rút gọn gọi là **increment**. Đoạn mã sau cũng biểu thị điều tương tự với đoạn mã ta vừa thấy ở trên :

C code:

```
sohang++;
```

Đoạn mã này khá ngắn so với những gì ta thấy trước đó, nó có nghĩa là « thêm 1 vào biến số sohang ». Chỉ cần viết tên biến số, sau đó thêm vào hai dấu +, và đừng quên dấu chấm phẩy đặt ở cuối cùng.

Chỉ là như vậy, chúng ta sẽ gặp lại nó thường xuyên vì có rất nhiều trường hợp cần sử dụng phương pháp này.

⚠ Nếu bạn chịu để ý một tí, bạn sẽ thấy dấu hiệu này được tìm thấy trong « C++ ». Và với con mắt của một nhà lập trình, bạn có thể hiểu được ý nghĩa của nó! C++ có nghĩa là ngôn ngữ C « được tăng thêm một cấp » 🤖, mặc dù vậy so với C nó không hề hơn.

Decrementing (Phương pháp giảm giá trị)

Đơn giản có thể hiểu là phương pháp này trái ngược hoàn toàn với increment. Chúng ta sẽ giảm giá trị của biến số đi 1.

Và chúng ta cũng sẽ sử dụng nó thường xuyên như increment.

Nếu như ta viết nó đầy đủ :

C code:

```
sohang = sohang - 1;
```

Thì đây là dạng rút gọn :

C code:

```
sohang--;
```

Hẳn là bạn có thể tự đoán ra được, ở vị trí ta đặt ++, thì thay thế bằng --. Nếu biến số có giá trị là 6 ban đầu, thì nó sẽ thành 5 sau khi thực hiện decrement.

Những dạng viết rút gọn khác

Trong C còn nhiều cách viết rút gọn khác cũng hoạt động tương tự. Tất cả các phép toán cơ bản : + - * / đều có phương pháp viết rút gọn.

Nó giúp ta phải viết lại tên của biến số cùng một dòng.

Và nếu bạn muốn tăng lên 2 lần giá trị của một biến số :

C code:

```
sohang = sohang * 2;
```

Bạn có thể viết dưới dạng rút gọn :

C code:

```
sohang *= 2;
```

Nếu số đó là 5 lúc ban đầu thì sau đó nó sẽ trở thành 10.

Với những phép toán cơ bản khác cũng hoạt động y như vậy, đây là một chương trình làm ví dụ :

C code:

```
int sohang = 5;
sohang += 4; // sohang tro thanh 9...
sohang -= 3; // ... sohang tro thanh 2
sohang *= 5; // ... sohang tro thanh 25
sohang /= 3; // ... sohang tro thanh 1
sohang %= 3; // ... sohang tro thanh 2 (vi  $5 = 1 * 3 + 2$ )
```

Đó là những cách viết rút gọn mà bạn sẽ sử dụng trong một ngày nào đó, 😊 hãy nắm vững increment vì đây là phương pháp viết rút gọn được sử dụng nhiều nhất. 😊

Thư viện toán học

Trong ngôn ngữ C, tồn tại một số cái gọi là những thư viện « standard », đó là những thư viện « cơ bản » luôn sẵn sàng để sử dụng, và được sử dụng thường xuyên.

Tôi xin nhắc lại, thư viện là tập hợp những function đã được viết sẵn bởi những nhà lập trình khác trước đó để tránh việc phải viết lại.

Chúng ta đã từng sử dụng function *printf* và *scanf* trong thư viện «stdio.h» .

Chúng ta phải biết rằng còn nhiều thư viện khác nữa, trong đó có «math.h», nó chứa một số lớn những function toán học đã được viết trước.

⚠ Ngoài phép toán cơ bản mà bạn đã biết, thì thư viện toán học chứa những phép toán phức tạp khác mà tôi chắc là bạn sẽ cần đến, ví dụ như là các hàm lũy thừa (nếu như bạn không biết đây là gì thì có thể bạn còn quá trẻ hay là bạn học toán vẫn chưa đủ).

Trong trường hợp, chúng ta muốn thực hiện những phép tính lũy thừa trong C! Làm sao tính một số mũ 2 ? Bạn có thể viết 5^2 trong đoạn mã của bạn, nhưng máy tính sẽ không hiểu cái đó là gì cả... Ít nhất bạn phải giải thích cho nó bằng cách sử dụng những thư viện toán học !

Để có thể sử dụng những function trong thư viện toán học, chúng ta bắt buộc phải thêm preprocessor directives ở đầu chương trình:

C code:

```
#include <math.h>
```

Một khi mà bạn đã làm điều đó, bạn có thể sử dụng tất cả các function trong thư viện này.

Tôi nghĩ là tôi nên giới thiệu chúng với bạn. Tôi sẽ không làm một list đầy đủ ở đây, vì nó có rất nhiều, các ngón tay đáng thương của tôi sẽ phỏng lên trước khi kết thúc phần hướng dẫn này. 😊 Tôi chỉ hướng dẫn các bạn một số function chính, những function có vẻ quan trọng nhất.

⚠️ Có thể trình độ toán học của bạn không đủ để hiểu làm cách nào có thể viết được những function này. Nếu đúng là như vậy, bạn không cần phải lo lắng gì cả. Chỉ cần đọc, việc này không có hại cho bạn đâu.
Và tôi sẽ cho bạn một lời khuyên miễn phí: hãy chú tâm vào những tiết toán, họ không nói như tôi ở đây đâu. 🤔

fabs

Function này sẽ trả về giá trị tuyệt đối của một số, trong toán học viết là $|x|$.

- Nếu bạn đưa function này giá trị là -53, nó sẽ trả về giá trị 53.
- Nếu bạn đưa function này giá trị là 53, nó sẽ trả về giá trị 53.

C code:

```
double giatri_tuyetdoi = 0, sohang = -27;  
giatri_tuyetdoi = fabs(sohang); // gia tri tuyet doi cua sohang se la 27
```

Function này sẽ **trả về một số dạng double** vì vậy biến số của bạn đưa vào cũng phải thuộc dạng double.

⚠️ Trong thư viện « `stdlib.h` » cũng có một function tương tự gọi là « `abs` », nó cũng hoạt động như vậy, chỉ trừ việc nó sử dụng những số nguyên « `int` » và nó trả về giá trị dạng số nguyên `int`.

ceil

Function này sẽ **trả về giá trị dạng số nguyên** nếu như ta đưa cho nó một số thực.

Đó là một dạng làm tròn. Nó sẽ luôn cho một **số nguyên có giá trị lớn hơn**.

Ví dụ, nếu như ta cho nó giá trị là 26.512, function sẽ trả lại 27.

Nó **sử dụng và trả lại giá trị dạng double**:

C code:

```
double lamtronLen = 0, sohang = 52.71;  
lamtronLen = ceil(sohang); // lamtronLen se bang 53
```

floor

Trái ngược với function ceil, function này cho ta **số nguyên có giá trị nhỏ hơn**.
Nếu như ta cho nó 37.91, function *floor* sẽ trả lại 37.

C code:

```
double lamtronXuong = 0, sohang = 37.91;  
lamtronXuong = floor(sohang); // giá trị của lamtronXuong sẽ bằng 37
```

pow

Function này cho phép tính lũy thừa một số. Chúng ta phải chỉ ra cho nó 2 giá trị: số hạng và cấp lũy thừa của số đó. Đây là cấu trúc của function này:

C code:

```
pow(sohang, capLuyThua);
```

Ví dụ, « 2 lũy thừa 3 » (chúng ta thường ghi là 2^3 trên máy tính), là phép toán $2 * 2 * 2$, cho kết quả là 8 :

C code:

```
double ketqua = 0, sohang = 2;  
ketqua = pow(sohang, 3); // ketqua sẽ được  $2^3 = 8$ 
```

Bạn có thể sử dụng function này để tính bình phương của một số.

sqrt

Function này **tính căn bậc 2** của một số, nó cho ta **giá trị dạng double**.

C code

```
double ketqua = 0, sohang = 100;  
ketqua = sqrt(sohang); // ketqua trở thành 10
```

sin, cos, tan

Đây là 3 function được sử dụng trong lượng giác.
Cách hoạt động của chúng như nhau, **trả về giá trị dạng double**.

Chúng ta phải cho nó giá trị **radian**.
(một radian bằng 180 độ)

asin, acos, atan

Đây là những function arc sinus (arcsin), arc cosinus (arccos) và arc tangente (arctan), các function lượng giác khác.

Cũng hoạt động tương tự như trên, **trả về giá trị dạng double**.

exp

Function tính exponential, hay còn gọi là **lũy thừa cơ số e**. **Trả về giá trị dạng double**.

VD: $\exp(4) = e^4$

log

Function tính logarit tự nhiên, là logarit cơ số **e**. (chúng ta thường ghi là « ln ») ngày trước còn đi học ông thầy bảo đọc cái này là « lóc nê be »

log10

Function này tính logarit cơ số 10 của một số. ông thầy bảo đọc là « lóc mười »

Lời kết:

Tóm lại, tôi đã không nói về các function khác. 😊 (Thật sự là tôi không biết là chúng được dùng để làm gì 😊)

Với những function này bạn có thể sử dụng cho phần lớn các trường hợp liên quan đến toán học.

Xin nói thêm lần nữa, nếu bạn không hiểu những điều tôi nói ở trên thì cũng không có gì nghiêm trọng cả vì những phép toán này chúng ta không nhất thiết cần đến. Trừ khi bạn phải làm một chương trình tính toán một vấn đề khoa học nào đó. 😊

Dù gì bạn cũng nên nắm vững function **floor**, **ceil**, và **pow**, nó rất cần thiết cho chúng ta trong tính toán.

Bài hướng dẫn chi tiết về các công thức toán tôi xin nhường cho những thầy giáo dạy toán. Nếu bạn vẫn còn đi học, tôi cho bạn một lời khuyên chân thành: Hãy học tốt môn toán, điều đó sẽ giúp ích rất nhiều trong lập trình. Nhưng chúng ta rất ít khi phải tính lũy thừa và tiếp tuyến khi viết chương trình, nó phụ thuộc vào chương trình mà chúng ta viết, tôi nói lại.

Ví dụ, nếu có ai trong các bạn hứng thú với những việc liên quan đến 3D (tôi sẽ hướng dẫn về 3D sau), bạn cần phải có một số hiểu biết về hình học không gian (đồ thị, vec-tơ...)

TRẮC NGHIỆM KIẾN THỨC.

❓ "Dấu nhân" trên máy tính là kí hiệu nào ?

- A. *
- B. +
- C. /
- D. -
- E. %

❓ Một câu hỏi đơn giản về module

Kết quả sẽ là bao nhiêu: $17 \% 5$?

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4
- F. 5
- G. 15

❓ "Ketqua" sẽ mang giá trị bao nhiêu ?

C code:

```
int ketqua = 0;
```

```
ketqua = (8 / 3) - 2;
```

- A. -2
- B. 0
- C. 1
- D. 2

❓ Phương pháp này là gì ?

C code:

```
sohang++;
```

- A. **Increment**
- B. **Increasing**
- C. **Supplementation**

❓ Biến số "sohang" sẽ mang giá trị bao nhiêu ?

C code:

```
int sohang = 4;  
sohang--;  
sohang *= 4;  
sohang %= 12;  
sohang += 1;
```

- A. **1**
- B. **4**
- C. **12**
- D. **14**

❓ Function nào sau đây sẽ làm tròn 5.47 thành 5 ?

- A. **pow**
- B. **ceil**
- C. **floor**
- D. **sqrt**

Đáp án:

- 1- A
 - 2- C
 - 3- B
 - 4- A
 - 5- A
 - 6- C
-