Question 1

Incorrect

Mark 0.00 out of 10.00

[Swap]

Trong bài tập này, bạn sẽ được làm quen với cách *truyền tham chiếu* vào hàm. Trước tiên, hãy xem ví dụ sau:

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

void change(int x) {
    x = x * 2;
}

int main() {
    int a = 5;

    cout << "Gia tri cua a truoc khi goi ham change: " << a << endl;
    change(a);
    cout << "Gia tri cua a sau khi goi ham change:   " << a;

    return 0;
}
```

Hàm change trong chương trình trên được viết với mục đích tăng giá trị của một số nguyên lên hai lần. Tuy nhiên, khi thực thi chương trình, ta nhận được kết quả như sau:

```
Gia tri cua a truoc khi goi ham change: 5
Gia tri cua a sau khi goi ham change:   5
```

Có thể thấy giá trị của biến a trước và sau khi gọi hàm change vẫn được giữ nguyên, mặc dù trong hàm change ta đã có tác động để thay đổi giá trị của tham số truyền vào. Mấu chốt vấn đề ở đây là cách chúng ta truyền tham số cho hàm. Cách mà đoạn code trên truyền tham số vào hàm change được gọi là *truyền giá trị*. Cụ thể, quá trình thực thi của đoạn code trên như sau:

- Khi gọi `change(a)`, giá trị của a (giá trị 5) được truyền vào hàm change;
- Một biến mới `int x` được khởi tạo với giá trị bằng 5;
- Mọi hành vi được sử dụng trong hàm change sau đó chỉ tác động lên biến x, do đó biến a không bị thay đổi gì khi kết thúc hàm change. Kể cả khi ta đặt tên tham số của hàm change là a thay vì x thì mọi thứ vẫn hoạt động như trên, ta có hai biến cùng tên a trong hai hàm change và hàm `main`, tuy nhiên chúng là hai thực thể khác nhau.

Để giải quyết vấn đề này, hàm change cần sử dụng một kiểu biến khác, đó là *biến tham chiếu*. Biến tham chiếu có thể được hiểu như "đại diện" của một biến khác. Khi sử dụng biến tham chiếu, ta sẽ truy cập trực tiếp vào vùng nhớ của biến được "đại diện", tức là mọi thay đổi trên biến tham chiếu cũng là những thay đổi trên biến được "đại diện". Để khai báo biến tham chiếu, ta chỉ cần thêm dấu `&` ở trước tên biến và sau kiểu dữ liệu. Ví dụ:

```
double a = 4.18;
double &x = a; // x = 4.18
x = 5.12;      // a = 5.12
a += 1;        // a = 6.12
               // x = 6.12
```

Như vậy, hàm change sẽ được sửa lại như sau:

```
void change(int &x) {
    x = x * 2;
}
```

Kết quả chạy chương trình sau khi sửa hàm change:

```
Gia tri cua a truoc khi goi ham change: 5
Gia tri cua a sau khi goi ham change:   10
```

Bài tập

Viết hàm `void swap(int &a, int &b)` để tráo đổi giá trị của hai biến kiểu nguyên được truyền vào hàm.

Đầu vào

Đầu vào từ bàn phím, gồm hai số nguyên a và b phân tách nhau bởi một dấu cách.

Đầu ra

In ra màn hình giá trị của a và b sau khi gọi hàm `swap(a, b)`. Giá trị của a và b nằm trên một dòng, phân tách nhau bởi một dấu cách.

Lưu ý

Các phần nhập/xuất dữ liệu trong hàm `main` đã được viết sẵn cho bạn. Bạn chỉ cần viết định nghĩa hàm `swap` tại vị trí được yêu cầu trong ô trả lời.

For example:

Input	Result
1 2	2 1

Answer: (penalty regime: 0 %)

Reset answer

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

void swap(int &a, int &b) {
    // complete the function

}

int main() {
    int a, b;
    cin >> a >> b;
    swap(a, b);
    cout << a << " " << b;

    return 0;
}
```

	Input	Expected	Got	
✗	1 2	2 1	1 2	✗
✗	10 20	20 10	10 20	✗

Show differences

▸ **SHOW/HIDE QUESTION AUTHOR'S SOLUTION (CPP)**

Incorrect

Marks for this submission: 0.00/10.00.

Question 2

Not answered

Mark 0.00 out of 10.00

[MaxThree]

Viết hàm `int maxThree(int a, int b, int c)` nhận đầu vào là ba số nguyên, đầu ra là số nguyên lớn nhất trong ba số đó.

Lưu ý: chỉ cần viết hàm như đề bài yêu cầu, không cần viết hàm `main()`, không cần viết các câu lệnh `#include, using namespace std ...`

For example:

Test	Input	Result

`int a, b, c;`
`cin >> a >> b >> c;`
`cout << maxThree(a, b, c);`
	4 20 1	20

Answer: (penalty regime: 0 %)

```
1
```

Question 3

Not answered

Mark 0.00 out of 10.00

[FactorialFunction]

Trong quá trình giải quyết các bài toán lập trình, có những phần việc được thực hiện lặp đi lặp lại nhiều lần khiến ta phải viết lại các đoạn code giống nhau nhiều lần. Việc lặp đi lặp lại các đoạn code giống nhau không chỉ gây phiền toái cho lập trình viên, mà còn khiến cho chương trình lủng củng, khó hiểu, khó sửa chữa và nâng cấp.

Hàm là một chuỗi các lệnh dùng để thực hiện một công việc nào đó. Sử dụng hàm sẽ giúp chúng ta tránh được việc viết nhiều lần một đoạn code thực hiện chung một công việc bởi sau khi định nghĩa hàm, mỗi khi cần thực hiện phần việc đó, ta chỉ cần gọi tên hàm để nó thực thi.

Hơn nữa, với những bài toán lớn, ta có thể chia nhỏ thành các bài toán con và viết các hàm để giải quyết từng bài toán con rồi ghép lại thành chương trình hoàn thiện. Phương pháp chia để trị này không chỉ giúp code của chúng ta nhìn gọn gàng và dễ hiểu hơn, mà còn giúp lập trình viên kiểm soát tốt hơn quá trình chạy của chương trình, rất có lợi cho quá trình *debug*.

Hàm có thể nhận vào 0 hoặc nhiều tham số. Một hàm có thể trả về (*return*) một kết quả nào đó, hoặc không trả về gì cả (*void*).

Trong C++, cú pháp để định nghĩa hàm như sau:

```
return_type tên_hàm(arg_type_1 arg_1, arg_type_2 arg_2, ...) {
    // đoạn code thực thi công việc của hàm
    // [nếu <strong>return_type</strong> khác <strong>void</strong>]
    return một giá trị/biến thuộc kiểu <strong>return_type</strong>;
}
```

Trong đó:

- `return_type`: kiểu dữ liệu của kết quả mà hàm trả về;
- `arg_i`: tên tham số (đầu vào) thứ i của hàm;
- `arg_type_i`: kiểu dữ liệu của `arg_i`.

Ví dụ, ta định nghĩa hàm sau đây để tính giá trị lớn nhất trong ba số nguyên:

```
int max_of_three(int a, int b, int c) {
    int max = a;
    if (max < b) max = b;
    if (max < c) max = c;
    return max;
}
```

Bài tập

Viết hàm `long factorial(int n)` trả về kết quả là giai thừa $n!$. Công thức tính giai thừa:

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$$

Đầu vào

Đầu vào từ bàn phím gồm duy nhất một số nguyên n.

Đầu ra

In ra màn hình kết quả của hàm `factorial`👎 .

Lưu ý

Các phần nhập/xuất dữ liệu trong hàm `main` đã được viết sẵn cho bạn. Bạn chỉ cần viết định nghĩa hàm `factorial` tại vị trí được yêu cầu trong ô trả lời.

For example:

Input	Result
4	24

Answer: (penalty regime: 0 %)

Reset answer

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

long factorial(int n) {
    // complete the function
}

int main() {
    int N;
    cin >> N;
    cout << factorial(N);
    return 0;
}
```

▸ SHOW/HIDE QUESTION AUTHOR'S SOLUTION (CPP)

Question 4

Not answered

Mark 0.00 out of 10.00

[Calculator]

Viết hàm `double calculate (double num1, char operat, double num2)` giả lập một máy tính bỏ túi.

Hàm thực hiện tính toán kết quả của phép toán giữa số $num1$ và $num2$ với toán tử $operat$

Biết các toán tử đơn giản thực hiện phép toán lần lượt là:

- Phép trừ $-$
- Phép cộng $+$
- Phép chia $/$
- Phép nhân $*$

Kết quả làm tròn đến chữ số thập phân thứ hai.

Đầu vào: Hai số thực và phép toán cần thực hiện (trong số các phép toán $+, -, *, /$).

Đầu ra: Kết quả phép toán.

For example:

Test	Input	Result
`calculate (num1, operat, num2)`	`2 + 3`	`5.00`

Answer: (penalty regime: 0 %)

```
1
```

Question 5

Not answered

Mark 0.00 out of 10.00

[SumOfDivisors]

Viết hàm tính tổng các ước số dương của số nguyên dương n cho trước.

Đầu vào

Đầu vào từ bàn phím gồm $T + 1$ dòng.

- Dòng đầu tiên chứa số nguyên T là số lượng số cần kiểm tra;
- T dòng tiếp theo, mỗi dòng chứa một số nguyên n $(n < 100)$.

Đầu ra

In ra màn hình T dòng, mỗi dòng ghi ra tổng các ước số dương của n.

For example:

Input	Result
1 6	12

Answer: (penalty regime: 0 %)

```
1
```

Question 6

Not answered

Mark 0.00 out of 10.00

[Prime]

Viết chương trình kiểm tra xem số nguyên $number$ có phải là một số nguyên tố không.

Đầu vào: Một số nguyên $number$.

Đầu ra: trả về "Prime" nếu số đầu vào là số nguyên tố, ngược lại, trả về "Not a prime".

Dữ liệu vào nhập từ bàn phím và kết quả được in ra màn hình.

For example:

Input	Result
3	Prime.

Answer: (penalty regime: 0 %)

```
1
```

Question 7

Not answered

Mark 0.00 out of 10.00

[BinaryRectangle]

Viết hàm `void binaryRectangle(int nRows, int nCols)` để in ra hình chữ nhật có $nRows$ hàng và $nCols$ cột với các cạnh của hình chữ nhật là số 1 và bên trong hình chữ nhật là số 0.

Chú ý: Bạn CHỈ phải viết hàm.

For example:

Input	Result
5 5	11111 10001 10001 10001 11111

Answer: (penalty regime: 0 %)

```
1
```

Question 8

Not answered

Mark 0.00 out of 10.00

[PrintArrow]

Viết hàm `void printArrow(int n, bool left)` để in một hình mũi tên, trong đó `n` là bậc của mũi tên, và `left` quy định hướng của mũi tên. Nếu `left` là `true` thì hướng của mũi tên sẽ quay sang trái, là `false` thì sẽ quay sang phải.

Chú ý: Bạn chỉ phải viết định nghĩa hàm ***printArrow*** mà không phải viết hàm ***main().***

For example:

Test	Input	Result

`int n, left;`
`cin >> n >> left;`
`printArrow(n, left);`
	`5 0`	`*****`
		`  ****`
		`   ***`
		`    **`
		`     *`
		`    **`
		`   ***`
		`  ****`
		`*****`

`int n, left;`
`cin >> n >> left;`
`printArrow(n, left);`
	`5 1`	`    *****`
		`   ****`
		`  ***`
		` **`
		`*`
		` **`
		`  ***`
		`   ****`
		`    *****`

Answer: (penalty regime: 0 %)

```
1
```

Question 9

Not answered

Mark 0.00 out of 10.00

[HammingDistance]

Khoảng cách Hamming giữa hai số nguyên là số lượng vị trí khác nhau giữa hai dãy bits tương ứng của chúng. Ví dụ: Khoảng cách Hamming giữa **1 (1)** và **4 (100)** là 2.

Cho hai số nguyên `x` và `y`, hãy viết hàm `int hammingDistance(int x, int y)` trả về khoảng cách Hamming giữa hai số này.

Khoảng cách Hamming là cái tên được đặt theo tên của Richard Hamming, người giới thiệu lý thuyết này trong tài liệu có tính cơ sở của ông về *mã phát hiện lỗi và sửa lỗi* (*error-detecting and error-correcting codes*). Nó được sử dụng trong kỹ thuật viễn thông để tính số lượng các bit trong một từ nhị phân (*binary word*) bị đổi ngược, như một hình thức để ước tính số lỗi xảy ra trong quá trình truyền thông, và vì thế, đôi khi, nó còn được gọi là **khoảng cách tín hiệu** (*signal distance*). Việc phân tích trọng số Hamming của các bit còn được sử dụng trong một số ngành, bao gồm lý thuyết tin học, lý thuyết mã hóa, và mật mã học. Tuy vậy, khi so sánh các dãy ký tự có chiều dài khác nhau, hay các dãy ký tự có xu hướng không chỉ bị thay thế không thôi, mà còn bị ảnh hưởng bởi dữ liệu bị lồng thêm vào, hoặc bị xóa đi, phương pháp đo lường phức tạp hơn, như khoảng cách Levenshtein (*Levenshtein distance*) là một phương pháp có tác dụng và thích hợp hơn.

For example:

Test	Input	Result

`int x, y;`
`cin >> x >> y;`
`cout << hammingDistance(x, y);`
	`1 4`	`2`

Answer: (penalty regime: 0 %)

```
1
```

Question 10

Not answered

Mark 0.00 out of 10.00

[SumOfDigits]

Viết hàm *sumOfDigits*🥐 trả về tổng các chữ số của số nguyên dương n.

For example:

Test	Result
sumOfDigits(1234)	10

Answer: (penalty regime: 0 %)

Reset answer

```
int sumOfDigits(int n)
{
    // Student code
}
```

Back to Course