Agenda

- Relational Operators, Logical Operators
- `if` statements
  - Block `if`
  - `if - else if - else`
  - `switch - case` statements (not quite important)
  - `?:` expression
- Short-hand Arithmetic

Comparisons

Statements to compare two values of equal type.

Operators include the following 6:

```
a == b; // Double Equals, if two expressions are equal
a != b; // Not equal, if two expressions are not equal
a > b; // If a is greater than b
a < b; // If a is less than b
a >= b; // If a is greater than or equal to b
a <= b; // If a is less than or equal to b
```

Of the 6 statements above, the result is a boolean value, i.e. `true` or `false`.

If what the expression states is real, the result is `true`, otherwise the result is `false`.

Example:

```
1 == 2; // false
2 > 1; // true
4.5 > 3.2; // true
'c' >= 'b'; // true
'a' < 'S'; // false, guess why?
```

Sometimes we need to match a lot of conditions together, so here we have three more operators that works with boolean types.

&&

Logical AND, or sometimes called double `&` (pronounced 'double-and' or 'double-ands'), or just 'and', connects two boolean values, and the result is `true` if **both** two operands are `true`, i.e.

```
true && true -> true;
true && false -> false;
false && true -> false;
false && false -> false;
```

||

Logical OR, or double `|` (pronounced 'double-or'), or just called 'or', also connects two boolean values, and the result is `true` if **either** one of the two operands is `true`, i.e.

```
true && true -> true;
true && false -> true;
false && true -> true;
false && false -> false;
```

!

Logical NOT, or simply called 'not'. Unlike the previous two operators, NOT works on a single operand, and inverts its value.

```
!true -> false;
!false -> true;
```

Exercise

```
(2 < 3) || (4 >= 5)

(4.5 > 4) && ((1.0 - 2.0) < 0) && ((6*7)>(5*8)) && false

! (true && false) || ! (true || false)
```

Note

Arithmetic operators have precedence over relational. Relational operators have precedence over the logical operators.

Within logical operators, `NOT` over `AND` over `OR`.

If you are not sure, just add parentheses.

IF

If statements are used to make decisions. Syntax as follows:

```
if (/* expression */) /*statement*/;
if (1 + 1 == 2) cout<<"helloworld"<<endl;
```

If the 'expression' part is true, the program will execute the statement, otherwise the statement will not be executed.

Block If

Sometimes it is needed to write more than one expression. At this case, we use something called **block**, or officially called *compound statement*.

A block is surrounded by `{` and `}`. For example,

```
if (2 > 1) {
    cout<<"Yes!"<<endl;
    cout<<"2 is bigger than 1!"<<endl;
}
```

If-else

There is an optional branch for `if` statement, that is `else`. Statement following `else` is executed if the conditional expression is `false`. Example

```
if (1 > 2) {
    cout<<"Math doomed!"<<endl;
} else {
    cout<<"Math is safe!"<<endl;
}
```

It is also possible to write another `if` statement after `else`, which is what we usually called `else-if`.

```
if (a > b) {
    cout<<"a is larger than b"<<endl;
} else if (a < b) {
    cout<<"a is smaller than b"<<endl;
} else {
    cout<<"a is equal to b"<<endl;
}
```

switch-case

`switch` statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values. This is not frequently used, and can be fully replaceable by `if` statements. Syntax as follows:

```
switch (/*expression*/) {
    case (/*expr1*/):
        /*statements*/;
    case (/*expr2*/):
        /*statements*/;
    case (/*expr3*/):
        /*statements*/;
    default:   // In case that none of the expression matches
        /*statements*/
}
```

Example usage:

```
char ch;
cin>>ch;
switch (ch) {
    case ('A'):
        cout<<"Grade A"<<endl;
        break;
    case ('B'):
        cout<<"Grade B"<<endl;
        break;
    case ('C'):
        cout<<"Grade C"<<endl;
        break;
    default:
        cout<<"Grade Unaccepted"<<endl;
}
```

?: expression

There is a special expression that looks just like `if`, sometimes used frequently, if assigning some values. It is called "question-mark expression". Syntax:

```
int a = (expr) ? true_value : false_value
```

Here, value of `a` will be assigned to `true_value` if the `expr` is true, but will be assigned to `false_value` if the `expr` is false. Here's a more straightforward example:

```
int N;
cin>>N;
cout<< (N % 2 == 0) ? "even" : "odd" << endl;
```

If the value inserted is an even number, print "even", otherwise print "odd".

Short-hand Arithmetic

In C++, there are some other arithmetic operators that will save us some time.

++ and --

`++` and `--` are operators for (usually int) variables. It is used to increment / decrement the variable value by 1. For example, `a++` means increment `a` by 1, i.e. `a = a + 1`.

There is a little difference between `a++` and `++a`, when they are compounded with other statements. See the following example:

```
int a,b,c;
a = 5;
b = a++;

a = 5;
c = ++a;
```

It might be quite curious that `b=5`, but `c=6` at the end. The difference is the timing when `a` gets increments. `a++` increments **after** the whole statement is executed, while `++a` increments **before** the value is used for other purposes. However, if used individually, `++a` and `a++` have no difference.

+=, -=, etc.

Another category is for assigning variables. They include `+=`, `-=`, `*=`, `/=`, and `%=` for now (In future, more will be introduced).

```
a += 1 <=> a = a + 1 <=> a++

t -= 5 <=> t = t - 5

c %= 17 <=> c = c % 17
```

Homework

Question 1. 双十一活动

Dollar Tree是美国一家著名连锁商店，其特点是贩卖廉价的小物品，多数物品仅需1美元即可买到。虽然现实中的Dollar Tree也会有2块、3块的物品，**在这题里我们假设这家Dollar Tree里全都是1美元的物品。**

因为赶上了疫情，今年的Dollar Tree旧金山唐人街店的营业业绩显得非常惨谈，老板想要趁着“双十一”带一波货，否则过了这个月就连员工工资都发不出来了。

今年Dollar Tree的优惠策略是这样的：玩家每购买`M`件物品，其中的一件物品可以免费赠送！换言之，顾客可以用`M-1`美元购买本来`M`美元的东西。

老板想知道，假设一位顾客购买了`N`件物品的话，经过优惠后，他实际需要花费多少钱呢？你可以假设`M`和`N`都是正整数，且不用考虑美国的消费税。

输入数据的第一行为两个数，其中第一个数代表`N`，即顾客购买的件数，第二个数代表`M`，即每`M`件物品其中一件免费。你只需要输出一个数，即优惠后顾客实际需要支付的费用，单位为美元。

输入样例1：

```
12 3
```

输出样例1：

```
8
```

说明：客户购买了12件物品，按照每3件免1件的政策，一共可以享受4次，也就是免掉4件。

输入样例2：

```
4 6
```

输出样例2：

```
4
```

说明：每6件免1件，但是顾客连6件都没有买到，所以按照原价支付。

输入样例3：

```
77435 8
```

输出样例3：

```
67756
```

Question 2. 绝对值

输入一个数，计算它的绝对值。你可以默认这个数在`int`范围内。

输入样例1：

```
200
```

输出样例1：

```
200
```

输入样例2：

```
-177
```

输出样例2：

```
177
```

输入样例3：

```
0
```

输出样例3：

```
0
```

输入样例4：

```
-2147483648
```

输出样例4：

```
2147483648
```

Question 3. 三角形

输入三角形的三条边长度`a`，`b`，`c`（均为正整数），判断它能否成为三角形的三条边长。如果可以的话，再判断它能否成为直角三角形的三条边长。

如果它可以做成直角三角形，输出`Right Triangle`。如果它只能做出非直角的三角形，输出`Triangle`，如果它不能做出三角形，输出`Not a Triangle`。

输入样例1：

```
5 3 4
```

输出样例1：

```
Right Triangle
```

输入样例2：

```
13 12 5
```

输出样例2：

```
Right Triangle
```

输入样例3：

```
6 7 8
```

输出样例3：

```
Triangle
```

输入样例4：

```
6 18 7
```

输出样例4：

```
Not a Triangle
```

输入样例5：

```
13 83 85
```

输出样例5：

```
Triangle
```

输入样例6：

```
6 6 6
```

输出样例5：

```
Triangle
```