Agenda

- A C++ program Template
- Standard Output: `cout`, `printf`
- Math expressions and operators
- Comments
- Variables and types, declaration, value assign
- Swap
- Types
- Types conversion

A C++ program template

```
// File name: xxx.cpp
#include <bits/stdc++.h> // Headers

using namespace std;     // Namespaces

int main() {           // Main Function
    /*
        Write your program here
    */

    return 0;
}

```

Standard Output

cout (C++ style - iostream)

```
cout << "Hello world!" << endl;
```

`ostream` 输出流

printf (C style - print format)

```
// printf ("format_string", <var1>, <var2>, ...);
printf("Hello World\n");
printf("%d\n", a);
```

Note: All statements end with `;`

Math expressions and operators

```
5 + 10
(8 * 13 + 5)
(9 / 2 + 4 * 3 - 1 * 6)
```

Frequently used math operators: `+`, `-`, `*`, `/`, `%`

`%` is called *modulo*, i.e. find the remainder. Example: `19 % 4 = 3`

Precedence: `*`, `/`, `%` over `+` and `-`.

Example: `(4 * 5 + 6 / 2) = 23`.

A special note on `/`. The operator `/` works on two integers, and its result is still an integer. Remainders would be truncated. Example: `5 / 2 = 2; -5 / 2 = -2`.

Comments

Comments are words that help a programmer to make notes during programming.

In case we forgot the purpose of a piece of code, comments would help us recall.

It is also convenient for others to understand our code.

> There was a following joke: there are 4 things that a programmer hates most.
>
> 1. Write comments
> 2. Write documentations
> 3. Others don't write comments
> 4. Others don't write documentations.

inline comment

Also called a single line comment. Within a line, whatever after the `//` (double slash) are comments.

```
// This is a comment
4 + 5 // This is another comment
```

block comment

Also called multi-line comment. Whatever contained within the pair of `/*` and `*/` (slash-star and star-slash) are comments.

```
/* This is a
    block comment */
/*
    Block comments can go over lines, and can go not aligned
    just like what's below.
    */
```

Variables

A variable is a value, which has a name, and its value could be changed.

Example:

```cpp
int x;
x = 5;
```

We declared a variable, its name is `x`, and its type is `int` (*integer*). Then, we *assigned* value 5 to this variable `x`.

You can declare a variable using the statement: `<type> <name>;`. You must declare a variable before using it.

You could assign a variable with the operator `=`. Left hand side of `=` is the variable to assign, and what's going on the right side is the value.

Note, the value could be expressed in expressions. Example:

```cpp
int x, y;
x = 4 + 5;
y = x + 3; // 9 + 3
```

Note, we could declare several variables at the same time, if they have the same type.

Usually for short, we could also write `int s = 0;` to combine the declaration and initial value assign at the same time.

Swapping

Here comes the first challenge. We have two variables, and you need to swap their values. In other words, whatever is in variable `x`, should be replaced with the value of `y`, while the value of `y` should be replaced with the original value of `x`.

The most common way is to use a *temporary* variable. Here:

```cpp
int x = 3, y = 4;
int t;
t = x; // t = 3, x = 3, y = 4;
x = y; // t = 3, x = 4, y = 4;
y = x; // t = 3, x = 4, y = 3;
// Finished swapping
```

Variable Basic Types

There are various types in C++, and we'll only cover some basic ones here.

int, long, long long

`int`, or integer, is one of the basic, mostly used types in C++. As you've already seen, it is used to store **integer numbers**.

However, that doesn't mean the type `int` can store some infinity large number. It has a limit: from `-2147483648` up to `2147483647`. Usually referred as $\pm 2 \times 10^9$.

If you want to store some larger numbers, types for larger integers must be used, usually `long` and `long long`. The demerit is they occupy more memory, and they sometimes don't behave well.

Side note: as you might guess, there is a type `short` which stores smaller numbers and occupy less memory. Also, if you don't want to store negative values, you could save the memory in return for larger positive numbers by adding the prefix `unsigned` to those values. For example, `unsigned int` could store value from `0` to `4294967295`.

char

`char`, or character, is another basic type. It is usually used to store character values, like `'a'`, `'b'`, etc.

Example:

```cpp
char s = 'a';
```

Note, a single character in C++ is quoted by `' '`(single quote).

Single characters can combine together and form *strings*. For example, `"Hello World"` is a string. A string is quoted by `""` (double quotes).

float, double

Other than integers, variables can also store real numbers if they are declared as type `float`.

```cpp
float s = 1.5;
float pi = 3.14159;
```

However, `float` numbers usually have a big error tolerance, meaning the value it stores might have a difference with what you really want. You could shrink this tolerance by using a type with larger memory but wider range, `double`. Usually `double` is the best type for float numbers.

Note, float numbers can do arithmetic operations with integers. When doing so, integers must be converted into float numbers first, and then calculate all numbers in floats.

```cpp
float result = 5 / 2.0;
// result = 2.5, with a small difference;
```

boolean

Boolean is a special type with only two possible values: `true` and `false`. You'll see the usage of this type in the next lecture, when we cover conditional operators, logical operators and `if` statements.

```cpp
bool a = true;
bool b = false;
```

Type conversions

We could actually make conversions between different types. We'll first introduce the two different kinds of type conversions.

Explicit Type Conversion

For any value, you could use `(<type>)` before it, and it would convert into that type. Example:

```
double a = 5 / 2; // a = 2.0
double b = 5 / (double)2; // b = 2.5
double c = 5 / a; // c = 2.5
```

In the first line, first the value of `5/2` is calculated. Since both numbers are integers, we operate an integer division and thus the remainder is truncated.

In the second line, we explicitly turned the second number, `2`, into a double, and now we are doing an integer dividing a float. We must change the integer to float first and do an float division. That's why the value assigned to `b` is 2.5.

Implicit Type Conversion

Sometimes C++ can do type conversions if needed, as you see in the first line of the previous example. Our final value on the right is 2, which is an integer. But the variable-to-assign is a double. C++ then do an implicit type conversion, that secretly changes the integer `2` into float `2.0` and assign `2.0` to `a`.

Actually in the second line of the previous example, the value `5` also faced an implicit type conversion before a float division can be operated.

Not only integers can be converted into floats, floats can also converted into integers, with parts after the decimal truncated. Example:

```
int a = 5.0 / 2.0; // a = 2;
```

`char` and `int` can be converted bidirectionally, using the ASCII code. Note, if the `int` number is larger than 128, which is the maximum size of ASCII chart, it may not behave as expected. Example:

```
char a = 48; // a = '0'
int b = 'a'; // b = 97;
```

Side note, `char` in C++ is actually implemented as `short short`.

Boolean can also convert with `int` bidirectionally. When converting from `boolean` to `int`, `true` goes to `1`, while `false` goes to `0`. From the other direction, only `0` will be considered a `false`, and any other value (e.g. `1`, `-2`, `5`, etc.) will be recognized as `true`.

Homework

In homework, you might want to assign some values from keyboard. You could assign those values by using `cin`.

Example: Read two integers. Add them and double the result, and print it out.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b; // Read 2 integers from keyboard into a and b
    int s = a + b; // Their sum;
    s = s * 2; // double it
    cout << s << endl; // Print s out. Here, "endl" stands for endline, or technically called line feed (回车).
}
```

Homework 1

输入一个三位数，分离出它的百位，十位和个位，翻转后输出。

样例输入1：

```
127
```

样例输出1：

```
721
```

样例输入2：

```
069
```

样例输出2：

```
960
```

Homework 2

鸡兔同笼：鸡有一头二足，兔有一头四足。现有鸡兔同笼，上有`n`首，下有`m`足。问鸡，兔各几何？（其中`n`，`m`为正整数，且确保题目有解。）

样例输入1：

```
14 32
```

样例输出1：

```
12 2
```

样例输入2：

```
500 1000
```

样例输出2:

```
500 0
```

样例输入3:

```
1834 6296
```

样例输出3:

```
520 1314
```

提示:

如果使用`` ` ``cout<<a<<b<<endl;`来输出两个数，两个数中间会没有间隔。因此输出时请使用`cout<<a<<" "<<b<<endl;`来分隔`a`与`b`的结果。