Agenda

- For loop
- While loop
  - Do-while loop
- Functions

For loop

Loop: A structure that its contents will execute for many times.

With respect to the older programming languages, C++ allows two types of loops: `for` loop and `while` loop.

The syntax of a `for` loop:

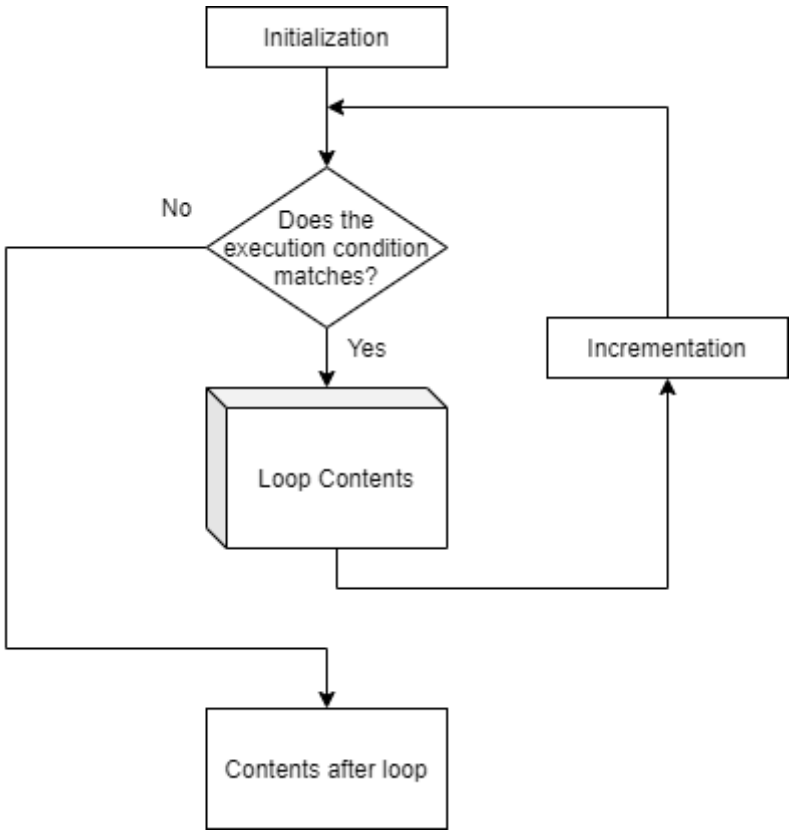
```
for (<init>; <exec_cond>; <increment>) {
    <statements>
}

for (int i = 0; i < N; i++) {
    s = s + i;
}
```

Usually, a **loop variable** is involved in a `for` loop. By counting up/down the value of loop variable, we could control how many times does this loop execute. In the example above, the loop variable is `i`.

Within the `for` parentheses, you need to write the following **three** statements, separated by semi-colon `;`.

- The first statement is the initialization. It is usually used for giving your loop variable an initial value.
- The second statement is the execute condition. It is usually a logical expression (i.e. value is `true` or `false`). If the value is `true`, the for loop will execute, otherwise it will not execute.
- The third statement is the incrementation. An incrementation to your loop variable is executed at the end of each loop execution.

The loop executes as follows:



For example, let's focus on the following loop:

```
s = 0;
for (int i = 0; i < 5; i++) {
    s = s + i;
}
cout<<s<<endl;
```

At first, the variable `i` is set to 0. Check the condition, `i < 5`, since `0 < 5`, we execute the statement `s = s + i`, `s = 0`. Then `i` increments by 1, now `i = 1`.

Then we check the condition again, since `1 < 5`, we keep executing the statement `s = s + i`, and `s = 1` now. `i` increments by 1, now `i = 2`.

We again check the condition where `2 < 5`, and `s = s + i`, and then `s = 3`. `i` increments to `i = 3`.

Keep going, `3 < 5`, `s = 6`, `i = 4`; `4 < 5`, `s = 10`, `i = 5`.

Here's what we want: now `i = 5`, and the executing condition `i < 5` is `false`, therefore we don't execute the loop again, and **jump out** from the loop. We keep executing what's after the loop, which is just print out `s`. We finish this *snippet* (piece of code) by print out `10`.

break

`break` is a special statement that could jump out from the loop, immediately.

```
int s = 0;
for (int i = 0; i < 10; i++) {
    s += i;
    if (i == 5) break;
}
cout<<s<<endl;
```

Once `i=5`, the `if` statement matches and `break` immediately terminates the loop. So the final result of `s` would be $0+1+2+3+4+5=15$.

Note that `break` could only jump out the loop for **one** layer. That means if two `for` loops are nested, only the innermost loop is terminated.

```
for (int i = ....) {
    for (int j = ....) {
        ...;
        break;
        ...;
    }
    ...;
}
```

The `break` only matters loop `j`, and has less impact on loop `i`.

continue

Another keyword / statement used in loop is called `continue`. `continue` is used when you want to end this current loop and start the next loop immediately.

```
int s = 0;
for (int i = 0; i < 10; i++) {
    if (i == 5) continue;
    s += i;
}
cout<<s<<endl;
```

Once `i=5`, `continue` executes and the loop immediately starts the next round, which is `i=6`, with the remaining statement `s+=i` not executed. As a result, the final value of `s` is $0+1+2+3+4+6+7+8+9=40$.

Similar to `break`, `continue` works on only **one** layer, meaning when nested loops occur, it only influences on the innermost loop.

While Loop

A `while` loop has a specific execution condition, same as the second statement of a `for` loop. Syntax for a `while` loop:

```
while (<condition>) {
    <statement>
}
```

As long as the `<condition>` is `true`, the loop will execute forever.

`break` and `continue` also works on `while` loop.

A `for` loop can be translated to a `while` loop as follows:

```
for (expr1; expr2; expr3) {
    statement;
}
// *for* is same as the following *while*
expr1;
while (expr2) {
    statement;
    expr3;
}
```

Infinite Loop

Sometimes for special needs, we want a loop that never breaks. This is called an infinite loop, usually written as the following:

```
while (true) {
    <statements>;
}
```

Another way using `for` is the following, but less frequently:

```
for (;;) {
    <statements>
}
```

Note that unless for special needs (e.g. a web server), programs you write by now should terminate at some moment, meaning if you use an infinite loop, the loop must `break` at some point.

do-while loop

A variation of `while` loop is the `do-while` loop. Experience shows that `do-while` is less frequently used than `for` and `while`.

`for` and `while` all test the execution condition at the top of the loop, i.e. before the statements within are executed. `do-while` is different: it tests the execution condition at the end. Following is its syntax:

```
do
    <statements>;
while (<condition>);
```

Therefore the `<statements>` are guaranteed to execute at least once. If the `<condition>` is true, the loop is executed again.

Note that for `do-while` loop, it is not necessary to have blocks `{}` after the `do` and before the `while`, but it is suggested to do so.

Examples for loops

Question 1. Factorial

In mathematics, the factorial of a natural integer N is defined as:

$$N! = 1 \times 2 \times 3 \times \cdots \times N \ \ (N \geq 1)$$

And specially, $0! = 1$.

Given a number N, calculate its factorial ($N \leq 10$).

```cpp
int N;
cin>>N;
int result = 1;
for (int i = 1; i <= N; i++) {
    result *= i;
}
cout<<result<<endl;
return 0;
```

Question 2. Prime Numbers

Given an integer N, check if it is a prime number.

```cpp
int N;
cin>>N; // Read a number N to identify
if (N <= 1) {
    cout<<"No"<<endl;
    return 0;
}
for (int i = 2; i < N; i++) {
    // Challenge: Could you change that N to an expression of N, such that you could save some time?
    if (N % i == 0) { // If N can be divided by some i smaller than N, it is not a prime number
        cout<<"No"<<endl;
        return 0;
    }
}
// If none of the possible numbers can divide N, N must be a prime number
cout<<"Yes"<<endl;
return 0;
```

Question 3. 3n+1 guessing

There is such a guessing that:

For any natural number larger than 1, say N, we keep doing the following operations:

- If N is an odd number, triple it and plus 1, in other words, change to $3N + 1$.
- If N is an even number, break this number to half.

After some finite number of operations, N will ultimately become 1.

Simulate these operations, and calculate how many operations are needed before it changes to 1.

```cpp
int N;
cin>>N;
```

```
3
4  int count = 0;
5  while (N > 1) {
6      if (N % 2 == 1) {
7          // odd number
8          N = N * 3 + 1;
9      } else {
10         // even number
11         N = N / 2;
12     }
13     count++;
14 }
15 cout<<count<<endl;
16 return 0;
```

Extra problem (HW1): If we insert $N = 987654321$, the result is actually 1, which is unreasonable. Why? Fix this piece of code such that it works for $N \leq 10^9$.

Function

As you've already written for a few times, statements you write are within a special block called `main()`.

```
1  int main() {
2      <statements>;
3      return 0;
4  }
```

This `main()` is called a **function**.

In mathematics, a function takes in some values, does some calculations/operations, and returns a value out. This is exactly what a function does in C++.

```
1  <return_type> <function_name> ( <parameters> ) {
2      <statements>;
3  }
```

A function contains four key things.

- Name. A defined function should have a name. Once a name is given, we could **call** this function using its name. (*call* is to execute a function).
- Return type. It indicates the type of value that a function returns.
- Parameters. They are the values that **passes** into a function. This is optional.
- Statements. They are the statements within a function that defines the process to finally get a result.

Let's write a simple function first. The following function `abs()` returns the absolute value of an integer (except the least number that `int` represents, which is -2147483648).

```
1  int abs(int N) {
2      if (N >= 0) return N;
3      return -N;
4  }
```

The first `int` before `abs` indicates that `abs()` function returns an integer. `abs` is the name of this function. What's inside the parentheses is `int N`, indicates that there's only 1 parameter, its type is `int`, and its name is `N`.

The keyword `return` is somewhat similar to `break`, but it works on a function, and tells what value to return. When the `return` statement is executed, the function terminates immediately.

There is also a special type called `void`, in case you want to return nothing. We'll see examples of `void` when we learned pointers.

Let's see another function that takes multiple parameters:

```
int add(int A, int B) {
    return A + B;
}
```

This function takes in two parameters `A` and `B`. They are separated using a comma `,`. Note that all parameters passed have a different name, while their types must be indicated individually.

Here's how you call a function:

```
int C = add(5, 10);
int D = abs(-8);
int E = abs(-C);
if (abs(-2) > 1) {
    //...
}
```

Another thing I would like to point out is that you must declare a function before you use it. For example, the following function call is illegal.

```
int main() {
    ...
    int N = try(5, 10);
}

int try(int A, int B) {
    ...
}
```

If you want to write the function afterwards (say for some easy-readable purposes), you may use something called pre-declaration, i.e. declare this function at first, but finish it afterwards. Note the function name, return type and parameters' type must be the same, but parameters' name could be different or even omitted.

```cpp
int try(int A, int B); // Function pre-declaration
// you could omit the name by writing "int try(int, int);"

int main() {
    ...
    int N = try(5, 10);
}

int try(int A, int B) {
    ...
}
```

It is possible if you declare a function first and never write how it executes.

Example for function: power

We define $\text{power}(a, b) = a^b$. Write a function `power()` in C++. (Assume $b \geq 0$)

```cpp
int power(int a, int b);

int main() {
    int N, M;
    cin>>N>>M;
    cout<<power(N, M)<<endl;
}

int power(int a, int b) {
    int result = 1;
    while (b > 0) {
        result *= a;
        b--;
    }
    return result;
}
```

Homework 1. Fix 3n+1 guessing

Fix the $3n + 1$ guessing program, such that it works for all possible numbers from $1 < N \leq 10^9$.

Homework 2. Number of digits

Given a number $0 \leq N \leq 10^9$, calculate its number of digits. Now, please write this problem as a **function**, in other words:

```
int num_of_digits(int N) {
    // Write your program here
}

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

Finish this snippet such that it works as expected.

输入样例1：

```
12345
```

输出样例1：

```
5
```

输入样例2：

```
987654321
```

输出样例2：

```
9
```

输入样例3：

```
716354
```

输出样例3：

```
6
```

输入样例4：

```
0
```

输出样例4：

```
1
```

Question 3. Triangle

输入一个正整数$1 \leq N \leq 20$，输出一个N层的三角形。比如$N = 5$时输出如下：

```
    #
   ###
  #####
 #######
#########
```