**getline(cin, string) vs cin.getline(char, literal).**

*cin.getline()* is a member function of the input stream object. *getline()* is a member function of the string class.

Let's first look at how to use *getline()*:

#include<iostream>

using namespace std;

int main()

{

string str;

getline(cin,str);

cout << str;

return 0;

}

Use *getline()* as the name suggests to read the line. This string may include spaces.

Let's look at the input and output results:

Input: abc def ghhgjgh

Output: abc def ghhgjgh

The function of this program is to output the input string to the original integer, including spaces in the middle. The first line of the execution result is the input and the second line is the output.

Let's take another look at the use of *cin.getline*:

#include<iostream>

using namespace std;

int main()

{

char str[20];

cin.getline(str,20);

cout << str;

return 0;

}

The parameter of *cin.getline()* is an array of characters and the number of elements we want to read (unless we have special requirements for entering the number of characters, then the number of elements we usually take when declaring a character array is the size of the array). The function of this program is the same as the previous one.

Pass an input string for uncorrupted output and look at the input and output results:

Input: abc def hjhj

Output: abc def hjhj

We can see that the effect of this execution of *cin.getline()* is the same as that of *getline()*, both can output a string and can output spaces. But compared to *getline()*, when using *cin.getline()*, you have to pass in a character array, and also you need to specify the length of the character array, how long the operation string is. So I personally still prefer *getline()*.

For *cin.getline()*, let's modify the program a bit:

#include<iostream>

using namespace std;

int main()

{

char str[20];

cin.getline(str,5);

cout << str;

return 0;

}

When we change the second parameter of *cin.getline()* to 5, the program will only read the first 4 characters of our input string and output the result:

Input: abcdefgrtt fhg

Ouput: abcd

In other words, we can limit the number of valid characters entered by the user by setting the number of elements read in *cin.getline()*.

Let's see how *cin* reads data in C++. When *cin* reads a line, it will end when it encounters a space or newline. That is, when we want to output the string we entered intact, if *cin* is used directly, then it will only output the character before the first space, not the original string.

To conclude, let's look at an example:

#include<iostream>

using namespace std;

int main()

{

string str;

cin >> str;

cout << str;

return 0;

}

Look at the input and output results again:

Input: abc def ghjhjh

Output: abc

You can see from the output that it only outputs the characters before the first space, and since there is no read-write cycle in the program, the program will only output the characters before the first space.

Let's write a function that reads in a loop:

#include<iostream>

using namespace std;

int main()

{

string str;

while(cin >> str)

{

cout << str;

}

return 0;

}

Look at the input and output results:

Input: abc def fergfg

Output: abcdeffergfg

We can see from the output results that this time it will be read in a loop, but we found that the spaces in the original string were not output, which is a disadvantage of using *cin*. *cin* ends with spaces or exchange behaviour, and will not output spaces.