23.2 — Input with istream

ALEX DECEMBER 21, 2020

The iostream library is fairly complex -- so we will not be able to cover it in its entirety in these tutorials. However, we will show you the most commonly used functionality. In this section, we will look at various aspects of the input class (istream).

The extraction operator

As seen in many lessons now, we can use the extraction operator (>>) to read information from an input stream. C++ has predefined extraction operations for all of the built-in data types, and you've already seen how you can overload the extraction operator for your own classes.

When reading strings, one common problem with the extraction operator is how to keep the input from overflowing your buffer. Given the following example:

```
1 char buf[10];
  std::cin >>
2 buf;
```

what happens if the user enters 18 characters? The buffer overflows, and bad stuff happens. Generally speaking, it's a bad idea to make any assumption about how many characters your user will enter.

One way to handle this problem is through use of manipulators. A manipulator is an object that is used to modify a stream when applied with the extraction (>>) or insertion (setw (in the iomanip.h header) that can be used to limit the number of characters read in from a stream. To use setw(), simply provide the maximum number of characters to read as a parameter, and insert it into your input statement like such:

```
1 #include <iomanip.h>
  char buf[10];
2 std::cin >> std::setw(10) >>
  buf;
```

This program will now only read the first 9 characters out of the stream (leaving room for a terminator). Any remaining characters will be left in the stream until the next extraction.

Extraction and whitespace

The one thing that we have omitted to mention so far is that the extraction operator works with "formatted" data -- that is, it skips whitespace (blanks, tabs, and newlines).

Take a look at the following program:

```
int main()
{
    char ch;
    while (std::cin >>
ch)
        std::cout << ch;

    return 0;
}
```

When the user inputs the following:

```
Hello my name is Alex
```

The extraction operator skips the spaces and the newline. Consequently, the output is:

```
HellomynameisAlex
```

Oftentimes, you'll want to get user input but not discard whitespace. To do this, the istream class provides many functions that can be used for this purpose.

One of the most useful is theget() function, which simply gets a character from the input stream. Here's the same program as above using get():

```
int main()
{
    char ch;
    while
(std::cin.get(ch))
        std::cout << ch;

    return 0;
}
```

Now when we use the input:

```
Hello my name is Alex
```

The output is:

```
Hello my name is Alex
```

std::get() also has a string version that takes a maximum number of characters to read:

```
int main()
{
    char strBuf[11];
    std::cin.get(strBuf, 11);
    std::cout << strBuf <<
'\n';

    return 0;
}

```

If we input:

```
Hello my name is Alex
```

The output is:

```
Hello my n
```

Note that we only read the first 10 characters (we had to leave one character for a terminator). The remaining characters were left in

the input stream.

One important thing to note about get() is that it does not read in a newline character! This can cause some unexpected results:

```
int main()
{
    char strBuf[11];
    // Read up to 10 characters
    std::cin.get(strBuf, 11);
    std::cout << strBuf << '\n';

    // Read up to 10 more
characters
    std::cin.get(strBuf, 11);
    std::cout << strBuf << '\n';
    return 0;
}
```

If the user enters:

```
Hello!
```

The program will print:

```
Hello!
```

and then terminate! Why didn't it ask for 10 more characters? The answer is because the first get() read up to the newline and then stopped. The second get() saw there was still input in the cin stream and tried to read it. But the first character was the newline, so it stopped immediately.

Consequently, there is another function calledgetline() that works exactly like get() but reads the newline as well.

```
int main()
{
    char strBuf[11];
    // Read up to 10 characters
    std::cin.getline(strBuf, 11);
    std::cout << strBuf << '\n';

    // Read up to 10 more
characters
    std::cin.getline(strBuf, 11);
    std::cout << strBuf << '\n';
    return 0;
}
```

This code will perform as you expect, even if the user enters a string with a newline in it.

If you need to know how many character were extracted by the last call of getline(), usegcount():

```
int main()
{
    char strBuf[100];
    std::cin.getline(strBuf, 100);
    std::cout << strBuf << '\n';
    std::cout << std::cin.gcount() << " characters were read" <<
std::endl;

    return 0;
}
```

A special version of getline() for std::string

There is a special version of getline() that lives outside the istream class that is used for reading in variables of type std::string. This special version is not a member of either ostream or istream, and is included in the string header. Here is an example of its use:

```
#include <string>
#include <iostream>

int main()
{
    std::string strBuf;
    std::getline(std::cin,
strBuf);
    std::cout << strBuf << '\n';

    return 0;
}
```

A few more useful istream functions

There are a few more useful input functions that you might want to make use of:

ignore() discards the first character in the stream.
ignore(int nCount) discards the first nCount characters.
peek() allows you to read a character from the stream without removing it from the stream.
unget() returns the last character read back into the stream so it can be read again by the next call.
putback(char ch) allows you to put a character of your choice back into the stream to be read by the next call.

istream contains many other functions and variants of the above mentioned functions that may be useful, depending on what you need to do. However, those topics are really more suited for a tutorial or book focusing on the standard library (such as the excellent "The C++ Standard Library" by Nicolai M. Josuttis).

Next lesson
23.3 Output with ostream and ios

Back to table of contents

Previous lesson
23.1 Input and output (I/O) streams

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ coc

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

74 COMMENTS

Newest