Inputting a Character

Sometimes you want to read only a single character of input. For example, some programs display a menu of items for the user to choose from. Often the selections are denoted by the letters A, B, C, and so forth. The user chooses an item from the menu by typing a character. The simplest way to read a single character is with `cin` and the `>>` operator, as illustrated in Program 3-20.

Program 3-20

```
// This program reads a single character into a char variable.
#include <iostream>
using namespace std;

int main()
{
    char ch;

    cout << "Type a character and press Enter: ";
    cin >> ch;
    cout << "You entered " << ch << endl;
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
Type a character and press Enter: A [Enter]
You entered A
```

Using `cin.get`

As with string input, however, there are times when using `cin >>` to read a character does not do what you want. For example, because it passes over all leading whitespace, it is impossible to input just a blank or [Enter] with `cin >>`. The program will not continue past the `cin` statement until some character other than the spacebar, tab key, or [Enter] key has been pressed. (Once such a character is entered, the [Enter] key must still be pressed before the program can continue to the next statement.) Thus, programs that ask the user to `"Press the Enter key to continue."` cannot use the `>>` operator to read only the pressing of the [Enter] key.

In those situations, the `cin` object has a built-in function named `get` that is helpful. Because the `get` function is built into the `cin` object, we say that it is a *member function* of `cin`. The `get` member function reads a single character, including any whitespace character. If the program needs to store the character being read, the `get` member function can be called in either of the following ways. In both examples, assume that `ch` is the name of a `char` variable that the character is being read into.

```
cin.get(ch);
ch = cin.get();
```

If the program is using the `cin.get` function simply to pause the screen until the [**Enter**] key is pressed and does not need to store the character, the function can also be called like this:

```
cin.get();
```

Program 3-21 illustrates all three ways to use the `cin.get` function.

Program 3-21

```
// This program demonstrates three ways
// to use cin.get() to pause a program.
#include <iostream>
using namespace std;

int main()
{
    char ch;

    cout << "This program has paused. Press Enter to continue.";
    cin.get(ch);
    cout << "It has paused a second time. Please press Enter again.";
    ch = cin.get();
    cout << "It has paused a third time. Please press Enter again.";
    cin.get();
    cout << "Thank you!";
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
This program has paused. Press Enter to continue. [Enter]
It has paused a second time. Please press Enter again. [Enter]
It has paused a third time. Please press Enter again. [Enter]
Thank you!
```

Mixing `cin >>` and `cin.get`

Mixing `cin >>` with `cin.get` can cause an annoying and hard-to-find problem. For example, look at Program 3-22.

Program 3-22

```
// This program demonstrates a problem that occurs
// when you mix cin >> with cin.get().
#include <iostream>
using namespace std;

int main()
{
    char ch;                          // Define a character variable
    int number;                       // Define an integer variable

```

(program continues)

Program 3-22 *(continued)*

```
11       cout << "Enter a number: ";
12       cin >> number;                // Read an integer
13       cout << "Enter a character: ";
14       ch = cin.get();               // Read a character
15       cout << "Thank You!\n";
16       return 0;
17   }
```

Program Output with Example Input Shown in Bold

```
Enter a number: 100 [Enter]
Enter a character: Thank You!
```

When this program runs, line 12 lets the user enter a number, but it appears as though the statement in line 14 is skipped. This happens because `cin >>` and `cin.get` use slightly different techniques for reading data.

In the example run of the program, when line 12 executed, the user entered 100 and pressed the [**Enter**] key. Pressing the [**Enter**] key causes a newline character (`'\n'`) to be stored in the keyboard buffer, as shown in Figure 3-5. The `cin >>` statement in line 12 begins reading the data that the user entered, and stops reading when it comes to the newline character. This is shown in Figure 3-6. The newline character is not read, but remains in the keyboard buffer.

Figure 3-5

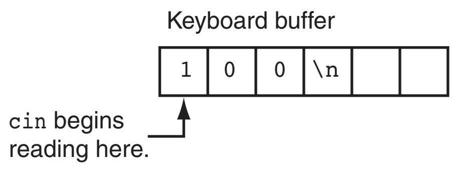
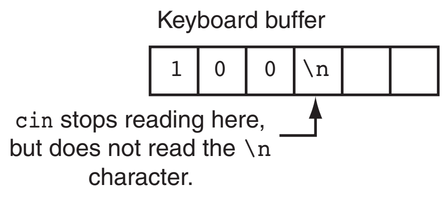


Figure 3-6



When the `cin.get` function in line 14 executes, it begins reading the keyboard buffer where the previous input operation stopped. That means that `cin.get` reads the newline character, without giving the user a chance to enter any more input. You can remedy this situation by using the `cin.ignore` function, described in the following section.

Using `cin.ignore`

To solve the problem previously described, you can use another of the `cin` object's member functions named `ignore`. The `cin.ignore` function tells the `cin` object to skip one or more characters in the keyboard buffer. Here is its general form:

```
cin.ignore(n, c);
```

The arguments shown in the parentheses are optional. If used, *n* is an integer and *c* is a character. They tell `cin` to skip `n` number of characters, or until the character *c* is encountered. For example, the following statement causes `cin` to skip the next 20 characters or until a newline is encountered, whichever comes first:

```
cin.ignore(20,'\n');
```

If no arguments are used, `cin` will skip only the very next character. Here's an example:

```
cin.ignore();
```

Program 3-23, which is a modified version of Program 3-22, demonstrates the function. Notice that a call to `cin.ignore` has been inserted in line 13, right after the `cin >>` statement.

Program 3-23

```
 1  // This program successfully uses both
 2  // cin >> and cin.get() for keyboard input.
 3  #include <iostream>
 4  using namespace std;
 5
 6  int main()
 7  {
 8      char ch;
 9      int number;
10
11      cout << "Enter a number: ";
12      cin >> number;
13      cin.ignore();              // Skip the newline character
14      cout << "Enter a character: ";
15      ch = cin.get();
16      cout << "Thank You!\n";
17      return 0;
18  }
```

Program Output with Example Input Shown in Bold

```
Enter a number: 100 [Enter]
Enter a character: Z [Enter]
Thank You!
```