**getch and ungetch**

It is often the case that a program cannot determine that it has read enough input until it has read too much. One instance is collecting characters that make up a number: until the first non-digit is seen, the number is not complete. But then the program has read one character too far, a character that it is not prepared for.

The problem would be solved if it were possible to “un-read” the unwanted character. Then, every time the program reads one character too many, it could push it back on the input, so the rest of the code could behave as if it had never been read. ***getch*** delivers the next input character to be considered; ***ungetch*** remembers the characters put back on the input, so that subsequent calls to **getch** will return them before reading new input.

**ungetch** puts the pushed-back characters into a shared buffer - a character array. **getch** reads from the buffer if there is anything there, and calls **getchar** if the buffer is empty.

What getch does is:

if there is a character already in **buf**

move the counter back one and then

return the character that was in the buffer's last used spot

else

get a character using getchar

ungetch just checks to see if the buffer is full, and if it isn’t, puts a character in it, updating the position of the end of the buffer.

Think of the input stream (which is generally the stuff you're typing at the keyboard) as a long queue of characters. If you want to read 10 characters, you pluck them off of the input stream, one by one.  
  
So you have a function that asks the user to enter a number, and he types this: 1234+  
  
Until you start reading what he typed (using scanf(), getchar(), fgets(), or a similar function), you can't know what he entered. The easy thing to do is use getchar() to read a character. So you do that, and get back '1'. Keep doing this, and eventually you'll see a '+'.  
  
Now, the user did nothing wrong, because your program told him to use '+' to add numbers together, so that's what he did. Your problem is that you have absolutely no idea how long the number he entered was. In this case, its 4 digits, but it could have been 1, or 10, or anything. The crux of the matter is that you cannot know that the number is 4 digits until you read *5* digits. You know the number's done when you read a non-digit character.  
  
Of course, here's the problem: now that you've read the '+', it's no longer available to be read. If you do nothing, the '+' "disappears". It would be as if the user never typed it. Ideally, the next time you call getchar() (or getch() in the case of this program), the '+' would still be there. That's the point of ungetch(): it puts the '+' back so it can be read by the next call to getch(). Now, the next time you call getch() you'll get that '+'. It's like you never accidentally read it in the first place. The '+' is pushed back onto the input stream (or, at least, it looks that way). That's what the pushing back is referring to.