Program Notes

The program uses `new` to create a new string for holding the selected characters. One awkward possibility is that an uncooperative user requests a negative number of characters. In that case, the function sets the character count to zero and eventually returns the null string. Another awkward possibility is that an irresponsible user requests more characters than the string contains. The function protects against this by using a combined test:

\[ i < n \land \text{str}[i] \]

The `i < n` test stops the loop after `n` characters have been copied. The second part of the test, the expression `str[i]`, is the code for the character about to be copied. If the loop reaches the null character, the code is zero, and the loop terminates. The final `while` loop terminates the string with the null character and then sets the rest of the allocated space, if any, to null characters.

Another approach for setting the size of the new string is to set `n` to the smaller of the passed value and the string length:

```c
int len = strlen(str);
n = (n < len) ? n : len;  // the lesser of n and len
char * p = new char[n+1];
```

This ensures that `new` doesn't allocate more space than what's needed to hold the string. That can be useful if you make a call like `left("Hi!", 32767)`. The first approach copies the "Hi!" into an array of 32767 characters, setting all but the first three characters to the null character. The second approach copies "Hi!" into an array of four characters. But, by adding another function call (`strlen()`), it increases the program size, slows the process, and requires that you remember to include the `cstring` (or `string.h`) header file. C programmers have tended to opt for faster running, more compact code and leave a greater burden on the programmer to use functions correctly. The C++ tradition, however, places greater weight on reliability. After all, a slower program working correctly is better than a fast program that works incorrectly. If the time taken to call `strlen()` turns out to be a problem, you can let `left()` determine the lesser of `n` and the string length directly. For example, the following loop quits when `m` reaches `n` or the end of the string, whichever comes first: