CS 7B - Fall 2019 - CS7B Quiz 10/28/19

Write responses on separate paper.

1. Consider the following code (assume all necessary libraries are included.)

```
1  char* strcopy(const char* s) {
       if (s == 0) return 0;
3      int n{0};
       while (s[n++] != '\0');
5      char* pc = new char[n + 1];
       for (int i = 0; s[i]; ++i) {
7          pc[i] = s[i];
           cout << "pc = " << pc << endl;
9      }
       pc[n] = 0;
11     //delete[] s;
       return pc;
13 }

15 int main() {
       string s;
17     char* cstr;
       while (cin >> s && s != "quit") {
19         cstr = strcopy(&s[0]);
           cout << cstr << "\n";
21         delete[] cstr;
       }
23 }
   /* sample output
25 catdog
   pc = c-----L**%
27 pc = ca----L**%
   pc = cat---L**%
29 pc = catd--L**%
   pc = catdo-L**%
31 pc = catdog-L**%
   catdog*/
```

   (a) Do we need the `const` modifier on the input to `strcopy()`? If so, why? If not, why not? And, if not, is it still appropriate? Why?
   ANS: The `const` modifier is not essential because there is no explicit promise from the calling context that `strcpy()` will not change its input and without the `const` mopdifier, you could, in fact, change the input–but, clearly, the intent is not to change the input, only to copy it, so the `const` modifier is appropriate.

   (b) On line 19, `&s[0]` is passed as the argument. Explain what this is and why it is used here.
   ANS: This is the memory address of the first character in the `string s`. If `s` were a simple array of `char`, then we could just pass `s`, since, for arrays, the name of the array is the address of the first character in the array–but...

   (c) Is the memory allocated for a string object necessarily contiguous in C++17? What would happen here if it weren't contiguous?
   ANS: ...for an STL type string, depending on whether you're using ISO C++11 or later, the memory may not even be contiguous, in which case this program could crash.

   (d) What happens if we pass `strcopy()` an empty string?
   ANS: It'll do the same thing it does when `s` is not the empty string. You can't pass a literal empty string to the function, so to test this you'll need to change the conditional of the `while` loop:

```
while (getline(cin,s) && s != "quit") { //..
}
```

(e) Is the while loop on line 4 guaranteed to terminate? How does it terminate? What is the value of `n` if and when the loop terminates?
ANS: The way this loop: `while (s[n++] != 0);` would terminate is by encountering a `NUL` character in memory (this is equivalent to a byte with all zeros). Such bytes are often found in random memory, but they're not guaranteed to exist, so it may happen that there is no zero anywhere in the memory beyond where we start searching, in which case `n` itself may overflow and wrap around. If it does terminate, `n` will be the number of bytes between where it started looking for a zero and where it found one.

(f) What happens on line 5?
`ANS:` Memory for `n+1 chars` is allocated on the heap and the memory address of the first is assigned to the variable `pc`, which, of course, is a pointer to a `char`.

(g) How does the for loop on line 6 terminate?
ANS: If `S[i]` is the NUL character, this is interpreted as `false` and the `for-loop` terminates.

(h) What is happening on line 10. Does this make sense?
ANS: Yes, it makes sense to add a NUL terminator to the string, so that when you print it, or whatever, it has a way of knowing when it's done.

(i) Why is line 11 commented out?
ANS: Line 11 would attempt to delete memory allocated on the stack. Only heap memory can be deleted this way.

(j) Explain the sample output at the end of the code.
Ans: The user has typed int "catdog" which is passed to `strcopy()` via `&s[0]`, the address of the 'c' in "catdog." The function then counts the number of characters from that memory location until the first instance of a NUL byte and allocates enough memory for one character more than that (line 5.) Then the `for-loop` in `strcopy()` copies the characters one at a time (line 7) and outputs the string to the console after each character is copied, showing the growing "catdog" in `pc`, each time up to the NUL character, including whatever random bytes are in that patch of memory. Then a NUL character is oplace at the end of the string, so that when the output of `strcopy()` is returned, no miscellaneous characters are included in the printout.

(k) Is the way memory is allocated on the heap here at all problematic? What could go wrong?// **ANS:** It's dangerous for a function to allocate memory for something and then relinquish responsibility for freeing that memory to some other function. Some code could be inserted in between the allocation and the deletion that also uses that function to allocate the memory, but then forgets to free it. For instance:

```
int main() {
    string s;
    char* cstr, *cstr2;
    cin >> s;
    while (1) {
        cstr = strcopy(&s[0]);
        cstr2 = strcopy(cstr);
        delete[] cstr;
    }
}
```

has a memory leak because `strcopy()` is called twice but the memory is only freed for one of the allocations.