

The Null Terminator: Rise of the Streams



Asad Sayeed

On the Well-Justified Fear of Invasion by Evil Robots
from the Future.

The Quiz You Loved

```
void printString(LongString s) {  
    int index = 0;  
    while (s[index] != '\0') {  
        Index++;  
    }  
    index--;  
    while (index >= 0) {  
        printf("%c", s[index]);  
        index--;  
    }  
    printf("\n");  
    return;  
}
```

- Many of you forgot that strings have null terminators.
- If you don't check for the null-terminator, you will print out the junk.
- There is no *.length* to save you. This is not Java.



The Terminator is not happy with you for forgetting his li'l buddy, '\0'.

The Quiz You Loved

```
int countchar(LongString s,  
char c) {  
    int count = 0, index = 0;  
    while (s[index] != '\0') {  
        if (s[index] == c)  
            Count++;  
        index++;  
    }  
    return count;  
}
```

- ❏ Better this time, but still forgot to check for the null terminator.
- ❏ Many of you kept forgetting to put variable names.
- ❏ Don't forget the skeleton. This is going in a .c file.

The Quiz You Loved

```
int atoi(LongString s) {
    int val = 0, index = 0;
    while (s[index] != '\0') {
        val *= 10;
        val += (s[index] - '0');
        index++;
    }
    return val;
}
```

- Characters representing numbers are codes. '1' is not 1, and so on. You cannot do math with them.
- C has no exponent operator. You'd need to use math.h. But you don't really need to.
- What we wanted was for you to return an int. You're just implementing atoi, actually.

Obligatory Rescue

- Project 1b is due soon.
 - Thursday to be exact!
- Anyone have any quick, pressing questions?



Tsk tsk, doing your assignment in the last minute...

Buffer

- *stream* – sequence of input or output data. Getting it byte by byte (or whatever unit).
- Disk access. One char written at a time:

```
'H' <write> 'e' <write> 'l'  
<write> 'l' <write> 'o'  
<write> ...
```

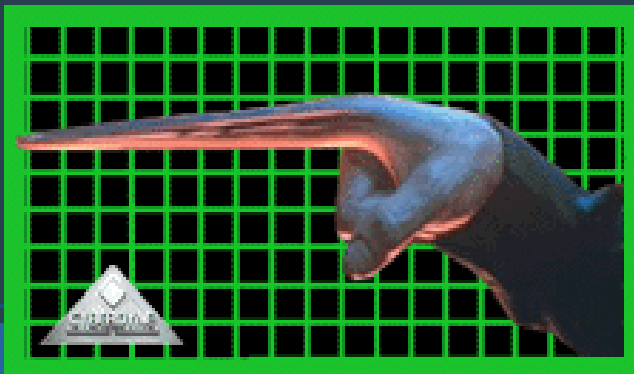
- Is this efficient? No! Writing to disk is slower than memory. Better to store 'Hello' in memory until a set size, and then write it.
- We use a *buffer* to do this (analogous for reading).



Clearly, Arnold is *buffer* than any of us here. We *are* CS students after all...who are we kidding?

File

- *File* – an area on disk (or something that pretends to be an area on disk, like devices in Linux/UNIX).
- *FILE* – a structure used in the C standard library to refer to a stream (which could be a file or *stdin* or whatever).
- *stdin*, *stdout* – standard *FILE* pointers.
- *fsomething* functions (*fgetc*, *fputs*, etc) – use with *FILE* pointers.
- Acquire control of stream with *fopen* and *fclose*, again through *FILE* pointers.



Do you think he needs a manicure?
I think he needs a manicure!

Input, Output

- Character input (byte by byte)
 - *fgetc*, *getchar* (obvious?)
 - *ungetc* (put it BACK onto the stream, like reversing a car only it's not a car but a data stream)

- Character output

 - *fputc*, *putchar*

- String input

 - *fgets*

- String output

 - *fputs*, *printf*

- And many more, but these should be good enough.



Pumping up the output. This is probably string output, rather than puny character output.

Hasta la Vista, section!

- Actually, you're still stuck with me for the rest of the hour.
- We'll study `getFile` from Project 1.
- Then we'll do a worksheet. Yay!
- Wednesday: More Mac fun.

