TELETEXT ASSIGNMENT: REPORT

TOM HOSKER

Preamble

The assignment consists, as per the problem sheet, of three parts:

- The "main course", i.e. the material not in the sub-folders;
- The testing of the above;
- The extension.

I shall now do my best to explain each of these as briefly as possible.

1. Main Course

As for the main course: this needs the least explanation. My program clearly prints the examples exactly as shown in Dr Campbell's examples, as well as a couple of other cases which I wrote myself.[1]

2. Testing

As per the problem sheet, the contents of this section have been copied to a plain text file in the testing folder.

As for the testing: this, in turn, is split into four "tests", each with its own folder inside the `Testing` folder.

2.1. **First Test.** In the folder named `1First_Test`, I try to plug in various silly values into the various functions. Often times, the compiler just yelps, "Don't do that specific thing!" I assume this is exactly what we want to happen in these cases, and so I've commented out the lines in question. I didn't succeed in making my programs crash in an ugly way by this method, which means either my code was perfect to begin with, or I'm not imaginative enough to find the flaws. Hopefully the former, but possibly the latter.

2.2. **Second Test.** Secondly, I went through the same functions, trying to find an `assert` that spat out the wrong answer. Again, I couldn't find anything.

Date: 22 Jan 2018 = 05 Uno $\mathfrak{T}_4$.

[1] A few words on the two sixels font files: as quixotic an approach as, in retrospect, this was, I actually drew the graphics portions of these "by hand" so to speak. Except I couldn't work out an easy way to convert a string of binary to hexadecimal, so I visualised what each bitmap ought to look like and then wrote the hexadecimal directly. This wasn't actually as difficult and tedious as it sounds... It was so, so much worse! A friend described my method as 'impressively masochistic'. But it worked, after much weeping and gnashing of teeth. Anyway, that was how I came by those files, in case you were wondering.

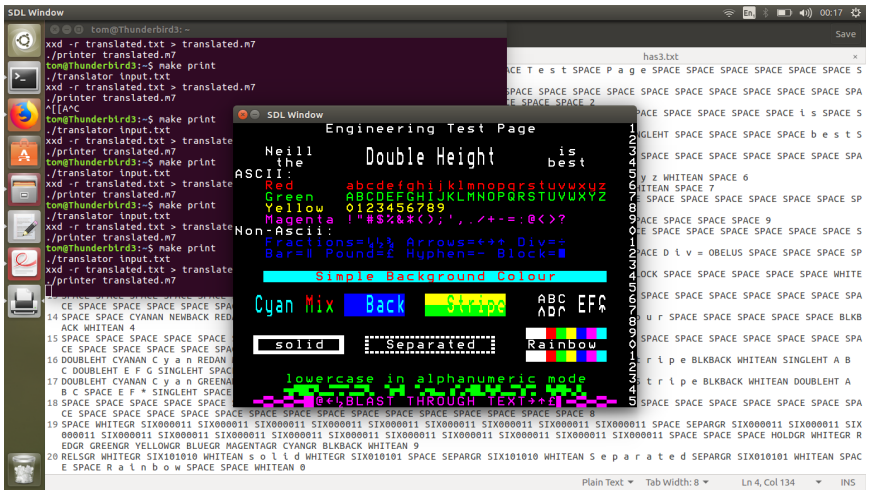

FIGURE 1. An image to prove that my program works: the test teletext page has been modified to include a message of which the marker ought to approve

2.3. **Third Test.** Thirdly, I ran Valgrind on my main course program. It seemed to find a lot of memory leaks; see the screenshot and `valgrind.txt` in the `3Third_Test` folder. I can't really see, from the reports, where they're coming from, but, given that the code I've written doesn't contain any explicit `mallocs`, they must be coming from SDL. And, if **Stack Exchange** is to be believed, SDL memory leaks can be safely ignored.

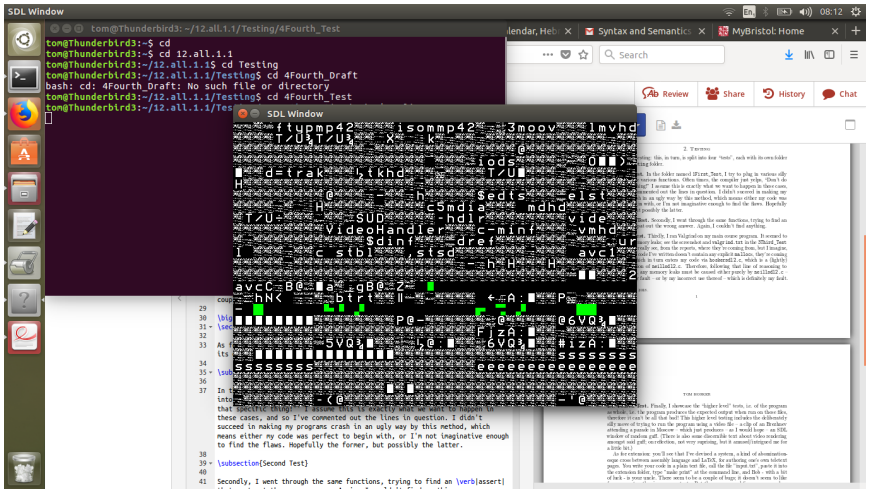

FIGURE 2. Random guff printed out by running my teletext program with a video file as the input

2.4. **Fourth Test.** Finally, I showcase the "higher level" tests, i.e. of the program as a whole. Which is to say: the program produces the expected output when run

on these files, therefore it can't be all that bad! This higher level testing includes the deliberately silly move of trying to run the program using a video file – a clip of Brezhnev attending a parade in Moscow – which just produces – as I would hope – an SDL window of random guff.[2]

3. Extension

As per the problem sheet, the contents of this section have been copied to a plain text file in the extension folder.

As for the extension: you'll see that I've devised a system, a kind of cross between assembly language and LaTeX, for authoring one's own teletext pages.

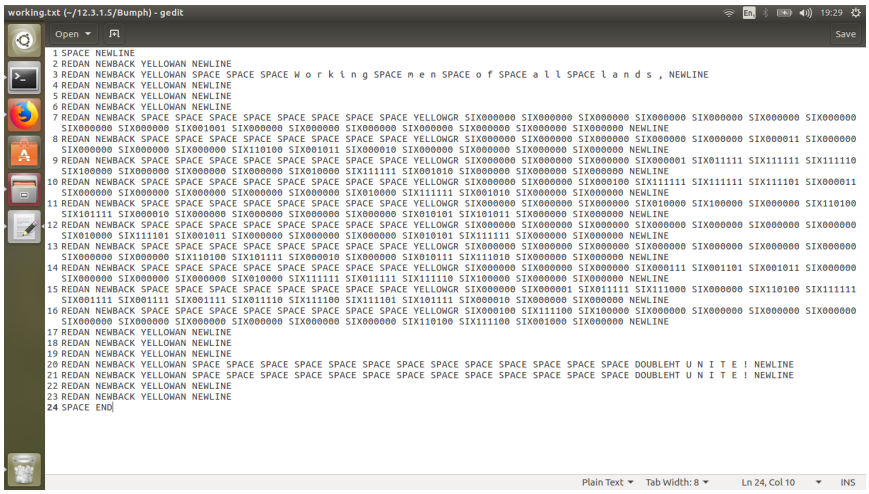

Figure 3. An example of code in the language I've devised

3.1. **Writing the Code.** Using this system, in order to build the teletext page in question, one first has to write it out in the simple language I've devised. Now this language works like so:

- A single character of code prints that character on the page; all the ASCII characters are encoded like this.
- The non-ASCII characters which appear in the font files are encoded by their own commands, e.g. `POUND` prints the £ symbol.
- Spaces are printed using the `SPACE` command.
- Commands and characters are separated using spaces, so, if one wanted to print words, one would have to encode them `l i k e SPACE t h i s .`
- New lines are printed using the `NEWLINE` command. New lines in the code itself may be inserted wherever the user finds most helpful; the translator will simply ignore them.

[2]There is also some discernible text about video rendering amongst said guff; on reflection, not very suprising, but it amused/intrigued me for a little bit.

- Other commands, such as for double heighted characters, are made using an appropriate abbreviation; in this case `DOUBLEHT`. See `controlstrings.h` for the full list.
- The code file must either (a) describe at least 25 complete lines of teletext, or (b) be ended with the `END` command, or both. Failure to abide by this rule will cause the city of Bristol to be consumed by fire, the Scots, led by Mel Gibson, to arise in might out of the north and civilisation to perish from this earth – but, seriously, the resulting SDL window isn't going to be pretty.

3.2. **Building the Window.** Now, having authored one's "TeleLatex" code file, one then has to carry out the following steps in order to build the SDL window:

- Put your code into a plain text file;
- Name this file `input.txt`;
- Paste the file into the extension folder;
- **If necessary, build the `printer`, `translator` and `wab` programs using `make trans`;**
- With the appropriate directory selected, run `make print` at the command line.

Carrying out these steps achieves the following:

- The "assembly language" file `input.txt` is translated into the "machine code" file `translated.txt` using the program `translator` which I wrote for this purpose;
- The `translated.txt` plain text file is converted to a binary file, called `translated.m7`, using another program I wrote called `wab`;[3]
- The binary file `translated.m7` is then printed, using the `printer` program, which is identical to the `main` program from before.

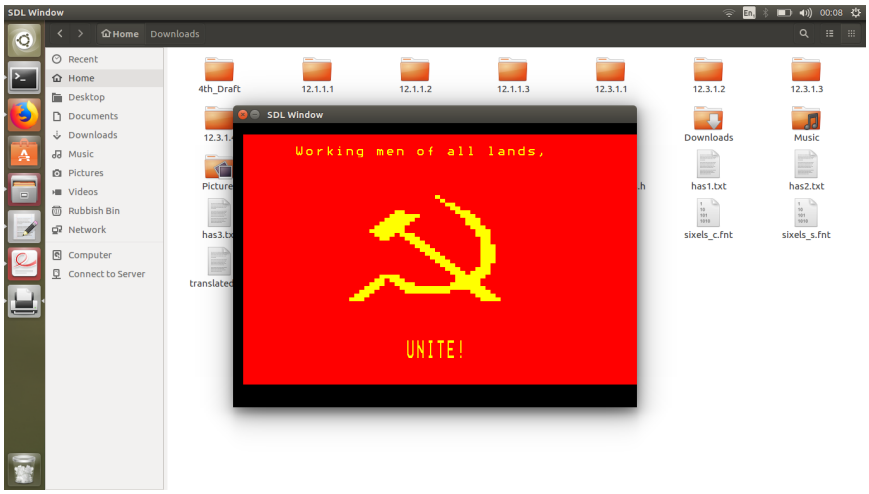

FIGURE 4. Continuing the Russian theme: a teletext page I wrote myself, using the system described

[3]Which stands for Write As Binary.

3.3. **Reflection.** With a pinch of luck, your teletext page should be printed to an SDL window as intended. There did, in earlier versions, seem to be a couple of bugs; it didn't seem to like four spaces together in some contexts. But three spaces and five spaces were okay. Originally I was using Linux's built-in reverse hexdump feature to make `translated.m7` out of `translated.txt`. Once I decided to stop being lazy, and build my own reverse hexdump-er in C, the problem seems to have disappeared. If I'd had more time, I would, of course, have carried out more rigorous tests on my program.

Bumph

There are also a couple of "bumph" folders knocking around. These contain what I call "cognitive scaffolding", i.e. if you're ever thinking, while reading through my code, "How on earth did this stupid berk come up with this?" then the proximate bumph folder would be the place to look. For all other purposes, these can be safely ignored.