254 Project 5 [README]

Collaborators👫

Name	NetID
Kevin Tusiime	rtusiime
Enting Zhou	ezhou12

Description

In this project, we implemented an online source code and assembly cross-referencing tool. We used ruby to compile a C program, generate it's DWARF information and assembly code, and generate the required HTML, CSS and JavaScript code to render this information in a web page. The web page is also interactive. It highlights lines of assembly or source code when a user clicks their corresponding source code or assembly lines, respectively.

Files

Description:

1. **disassem.rb**: Contains the ruby code to generate the Objdump, DWARF file of the provided c program.
2. **ascii.c**: C program provided by Professor Scott.
3. **header.txt:** HTML, Javascript and HTML code to render the webpage.
4. **priority_queue.c:** Extra C program to be used as test case.
5. **LinkedList.c:** Extra C program to be used as test case.

How to run program

- Compile the program using gcc

```
gcc -g3 -O1 -o <object_file_name> <source_file_name> //NOTE: the resulting HTML files will be named after the object file
```

- run the following command in the terminal

```
ruby disassem.rb <object_file_name> <source_file_name>
```

- Several files will be autogenerated. The webpage is contained in `object_file_disassem.html`.

General Approach

Our ruby program runs shell commands to compile the provided C program and then generate the Objdump and DWARF dump of the compiled program.

We then create four hashmaps:

```
sline2add - Maps a line in the source code to a LIST of assembly address
add2sline - Maps an assembly address to a LIST of source code lines
aline2add - Maps an assembly line to ONE assembly address
add2aline - Maps an assembly address to ONE assembly line
```

We use regular expressions to parse the Objdump and DWARFdump file. We use the table at the bottom of the DWARFDUMP to map each source code line to its corresponding assembly address.
Since the DWARFDump omits some assembly lines, we fill them in based on the OBJDump.

We display our HTML, Javascript and CSS code in one contiguous string block using a nifty ruby feature known as `heredoc`

A few issues that weren't initially obvious:

1. Parsing the addresses and storing them in a hashmap required us to convert it from `string` to `hex` number using `to_i(base=16)` and then to a decimal number. Then whenever we want to display it/print it, we convert it back to a hexadecimal number.
2. While parsing the source code, we also have to count the blank lines (initially we didn't, which had the unintended consequences of mismatching the line→address mapping in the DWARFDump table.
3. We had to find a way to parse from the OBJDump just the assembly code we needed. We noticed that the start address displayed in the example solution corresponds to the first address in the DWARFDump table, and the last address corresponds does not quite correspond to the last address in the table but is quite close. ——→ //to do→ update this with explanation of what we did.
4. We realised that there are three main cases when pattern matching the objdump from the start address and end sequence:
    a. Hex number and function name (for example `0000000004011bf <main>:`) → In this case, we print it to the HTML file and convert `<` and `>` to their HTML equivalent.
    b. If it's an empty line, we just print it.
    c. Hex number and address (for example `4011ab: bf 10 20 40 00`) → In this case, we split the line. The first part will be the address, which we render as button text, and the second part will be the code, which we will put in a span. We will also check if address is the end the end address, if so, we will end the parsing.
5. Contrary to what is mentioned on the assignment page, after extensive testing with other programs, we found that life the lines in the table are actually NOT sorted by assembly code address, to which we changed our code such that the start address is the minimum address in the table, and the end address is the maximum address (initially, we had set the start address to the first address in the table and the end address to the last address in the table).

Testing

We run our program on various C programs of different complexities and even compile our programs with `gcc -g3 -O1 -o ascii ascii.c` to see if the output remains consistent.

Extra credit considerations

We have gone above and beyond the assignment deliverables in the following ways and hope to be awarded some extra credit.

1. Extensive test cases, in terms of other C programs.