Assignment 4

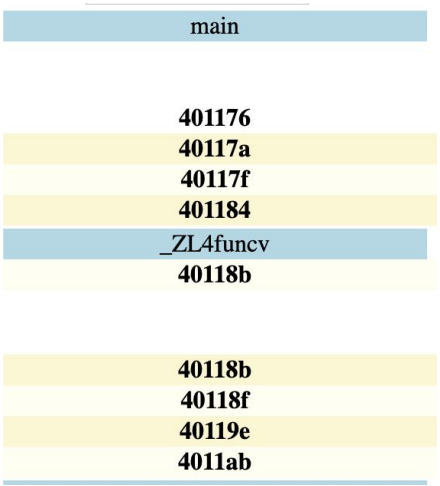
To run

Have an object file with the name, say “x”.
Run: ruby parse.rb “x”

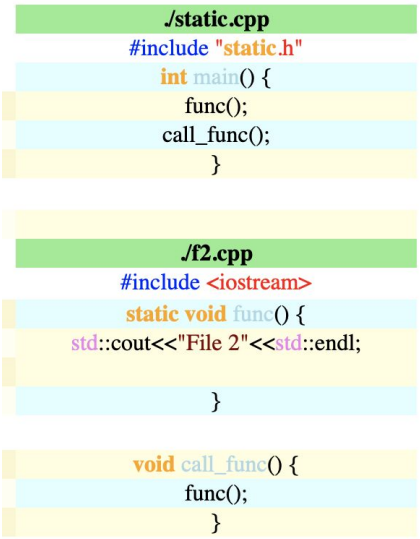
The above command will create (overwrite) HTML/index.html and HTML/“x”.html.
index.html will contain a link to the main function location in “x”.html.

Features

Following is the left hand side of XREF output:



Each bluish banner separates the assembly function blocks. The name inside the bluish banner is the name of the assembly function block that corresponds to the instruction lines that follow.
Following is the right-hand side of the XREF output:



The green banner represents the file from which the lines that follow were taken from. Moreover, if a single line of source corresponds to multiple assembly lines, then we extend the background color of that line to cover all those lines. In the above screenshot, you can see that “std::cout<<”File 2”<<std::endl;” corresponds to two contiguous lines of assembly code because the yellow background color continues in the next line.
We support multi-file compilation. To test this, you can run “ruby parse.rb static”. The object file static is compiled using “g++ -g3 f1.cpp f2.cpp static.cpp -o static”. The object file is called “static” because f1.cpp contains a static C function. Even though the writeup says to “make sure to differentiate between [static functions]”, the functions in the “static” example, while having the same name in the source already had different names in the assembly, so there was no need to explicitly do anything to differentiate between them.
You can also click on a function name corresponding to a callq, jmp etc., and you will be directed to that function.

Extra Credit

We have attempted syntax highlighting. The syntax highlighting feature detects and differentiates keywords, primitive types, macros, macro predicates, function names when definition functions, single-line comments, namespaces (std::) and strings. All this is implemented in the function “highlight” in “parse.rb”. The main bugs arise when a single pattern corresponds to multiple syntax highlighting styles. For example, keywords inside comments will still highlight as keywords and not parts of a comment.
Multi-line comments (/**/) are not supported

We have also implemented search for (source) function names. A small js file, called search.js updates the href attribute of the link “Submit” to “#x” where x is the current value in the input field. As an example, when user types func into the input field, “search.js” changes the href

attribute of the anchor tag surrounding “Submit” to “#func”. When the user clicks “Submit” he is directed to a tag with the already generated id “#func”. This id is generated along with syntax highlighting in the “highlight” function in parse.rb.

Limitations

The function “print_previous_lines” in parse.rb is slower than it should be. This function prints lines that don’t have a corresponding assembly instruction before the source line that does. The reason why this function is slow is because we apply the following algorithm for the lack of a better alternative:

1. For each assembly line:
    a. Find it’s corresponding source line in the right file.
    b. Open the file and iterate backwards from this source line.
        i. For each iteration *check whether the source line has a corresponding assembly instruction*. If it doesn’t then write that source line in the html page, otherwise stop.

The three level nesting of the algorithm, along with the italicized “check whether the source line has a corresponding assembly instruction” are the causes of the inefficiency. We could have used a hash-map to reduce the complexity by a factor, at the expense of memory.