CSC-254 Assignment 4

Mingzhe Du, Advaith Suresh

How to run:
Run the command: ruby xref.rb <program_name>
Where the only argument is the name of an compiled c program that was compiled using gcc -g3

Output:
Creates a directory called "html" and writes a file to it called "output.html"
Output.html contains html code that displays assembly code cross-referenced with the c source code.

We use ruby to implement this project. This project contains the following files:

main.rb	This program analyzes the assembly code and the llvm-dwarfdump output
xref.rb	This program uses the information generated by main.rb to generate the index.html
index.html	
others	Like test c programs and object files…

To implement the cross-indexing, we did the project with several steps listed below:

1) Analyze the llvm-dwarfdump output
   In this part, we use two arrays and three hash tables to store information that we need to use later.

array	file_indexes	This array is used to store all the file indexes that are given by llvm-dwarfdump. E.g., index of test.c is 1, then we store 1 into the array.
array	file_names	This array is used to store all the file names. E.g., we store "test.c" into this array.
hash table	lookup_table	This table is used to store the file names, assembly addresses and its corresponding source code line number. E.g., {test.c => {0x0001=>1, 0x0002=>2, 0x0003=>2} func.c => {0x0004=>1, 0x0005=>2, 0x0006=>3}}
hash table	new_lookup_table	This table is used to store the file names, assembly addresses and its corresponding source code. And if one source code line corresponds to multiple assembly code lines, then merging the assembly address. E.g., {test.c => {0x0001=>"int a", [0x0002, 0x0003] => "a=1"}

		func.c => {0x0004=>”int c”, 0x0005=>”int d”, 0x0006=>”c+d”}}
hash_table	unused_source_code	This table is used to store all the source code that doesn’t have corresponding assembly code.

2) Analyze the objdump output
We read the objdump output line by line, and to see if the address of each line is contained in the new_lookup_table, if it is matched, them there is an source code that we need to match with the assembly code. If not, them we just print out the assembly code. During this process, we need to check the table unused_source_code to see if we need to print out the source code line right after its previous source code line.

3) Branch-target linking
Every function call has a corresponding assembly code, so we just iterate the addresses stored in new_lookup_table, and match any addresses called in the assembly code, replacing it with an html anchor. We add any referenced addresses to a destination list. We then iterate through the assembly code again, this time looking for the destinations for each call, and we create a matching anchor at that location. This creates a link between them.