Write-up

1. Usage

In this assignment, four python files are created which are gen3ac.py, genbb.py, buildcfg.py and vn.py, corresponding to the four exercises. They should be used separately.

For the first python file, run it with two arguments. The first one indicates the input file which is a c_file, the second one indicates the output file which will also be a c_file (i.e., python gen3ac.py irc.c gen3ac.c).

The second one also takes two arguments. The first one indicates the input file which is a c file in three address code format and the second one indicates the output file which is a .txt file divide the three address code into basic blocks (i.e., python genbb.py gen3ac.c output.txt).

The third one takes one argument which is a .txt file contains the basic block information. The output of this program is a pdf file which shows how CFG looks like. The detail will be described later (i.e., python buildcfg.py output.txt).

The last one takes two arguments. The first one indicates the input file which is a c file in three address code format and the second on indicates the output file which is a c file after local value numbering optimization. (i.e., python vn.py irc.c vnresult.c)

2. Implementation Details

gen3ac.py:
    Take the AST as an input and parse it into three address code format. My code support following items:
    Assignment. Binary Operations. Unary Operations. If-else statements. While-loop statements. For-loop statements. Do-While loop statements. Switch statements. Function calls. Break statements. Continue statements.
    Recursive is used a lot in the process. Global variable are also used. A large parse_item() function is implemented to be responsible for all the parsing functions. Multiple functions named a3c_ are implemented to generate 3 address code to the file.
    To address the break/continue situation, I have a label at the beginning of a loop and a label at the end of the loop. When a break happens, jump to the end of the loop. When the continue happens, jump to the beginning of the loop.

genbb.py:
    Take three address code in and divide it into basic blocks. I implement it in a intuitive way, whenever there is a control signal (i.e., goto or label), a new basic block must be created since there must be a jump instruction into this basic block.

buildcfg.py:
    Take output.txt generated by genbb.py and use it to construct the control flow graph. I used the dot class in graphviz. I stored the basic block information into dot.nodes and linked them using dot.edges. I generated the output pdf file directly. So I don’t need to apply .dot command to get the output figure.

vn.py:

This file is for local value numbering optimization. All the optimization will be inside a local basic block. If a variable shown up on the right side of an equation and it has showed up on the left side before, this can be seen as an optimization point.

3. Test

I tested all the test file in my local machine (Mac OS X) and report some errors or warnings here. I will further explain some of them in the last part of the write-up.

factorial.c: passed compilation with no errors.

irc.c: some warnings.

guess.c: no return value in main function (which is the case in the code) and some warning.

precedence.c: passed compilation with no errors.

skip.c: passed with warnings.

pre_post_fix.c: passed with no errors.

temperature.c: some warnings.

4. Bugs and Flows

My have some bugs to be fixed.

1. I didn't track the type of the temporal variables well. And because of that, there will be some warnings indicate that the type doesn't match.
2. Repeat labels. Because of my implementation details, I will have two labels point to the same location. I haven't solve it.
3. I didn't test if there is a for loop inside another for loop, and both encounter a 'break' situation, what will happen.