ECOTE - final project

Semester: VI

Author: Artur Godlewski

Subject: ECOTE

I. General overview and assumptions

Project consist of creating a program that will be able to pull out of loops (for, while and do/while) unnecessary operations written in C language. Third party software ANTLR 4.7.2 will be used to create parse tree of the code. Project will be written in the C++ language.

Assumptions:

1. Input C code is written correctly - compiles without any errors.
2. Loops created with 'goto' are not considered as proper loops and will not be taken as possible candidates to make changes.
3. Loops with 'goto' instruction or labels inside will be ignored.
4. Any use of any function inside of a loop invalidates it.
5. Use of '*' and '&' operators is also not allowed
6. Only operation '=' where right operands do not change its value will be removed from the loop.
7. Selection statements like If() are considered as both true and false, both paths are considered to be executed.
8. Comment are not preserved as ANTLR does not take them into consideration and some of them could loose meaning after alterations.
9. Formatting of the code does not have to be preserved.

II. Functional requirements

Functional requirement of the project is to be able to remove unnecessary statements inside loops.
Code in the output needs to compile and work identically to the input code.
Only statements inside loops that follow assumption will be removed .

III. Implementation

General architecture

main() - In ECOTE.cpp, handles parameters given at startup or provides the user with means of inputting input and output paths of the file. Passes the path to CodeHandler class which handles code changes and prints its output to output file.

CodeHandler class - reads file content and parses it into a tree by using CLaxer and CParser classes prebuild by ANTLR with use of C.g4 grammar file found in ANTLR project repository. Tree is visited by using simple visitor FunctionVisitors which collects all nodes that start function body. All of those nodes are passed to separate CodeVisitors. After that tree is passed to BackToCode class which visits each node and creates string with code adding rudimentary formatting.

CodeVisitor class - start its walk ether at the start of function body or start of the loop. Visits each subsequent node and keeps track if any statement that was banned (‘*’, ‘goto’ etc.) has been used. Records names of declared variables using NameVisitor. On encountering iteration statement node it passes it and recorded variables to loopHandler.

NameVisitor class - finds variable Name

LoopHandler class - first it visits children of the node by using CodeVisitor. This allows to find any nested loops or banned statements inside loop body (includes nested loops). If no banned statements were found the tree is passed to StatmentOrderVisitor which collects data about statements in two maps - keyed by order and node. Each of the mapped statements is then checked if it can be removed. Changing variables are added to a vector. Checks are done in subsequent order.

0. In StatmentOrderVisitor some statements are marked as unremovable on encounter. Declarations and statements inside conditional statements. (marked statements are not checked, added here for clarity of the process logic)
1. Uses ‘++’, ‘+=’, ‘*= ‘ etc. operators. -> Cannot be removed, var changes.
2. Variable on the left of the operator was used in previous statement on the right. -> Cannot be removed.
3. Variable on the right of the operator is recorded as changing. -> Cannot be removed, var changes.
4. Variable that is used on the right side is used in subsequent statements -> Cannot be removed, var changes.

This checking process is repeated as long as there were new statements marked as unremovable. Nodes marked as removable at the end of this process are then removed from its parent and are reconnected to the parent of the loop node on the left side. This does not fully preserve tree integrity but because tree is traversed from left to right only BackToCode visitor has to encounter those tree manipulations and it does not require fully consistent tree structrure as it is considered only with text in the leaf nodes.

StatementOrderVisitor class - it visits all the nodes and finds statments. Those are fragmented into variables names by use of StatementSplitterVisitor and saved together with its node and a marker. The marker is used to determine if given statement can be removed from the loop. Declarations for example are marked as unremovable at the time of discovery.

StatementSplitterVisitor class - finds variable used on the left of a statement and variables used on the right.

Data structures

- All variable declarations are recorded into vector by CodeVisitor. Current state of this vector is passed to LoopVisitor on loop encounter. It will be starting point for subsequent

CodeVisitor vector. This assures that current vector has all variables declared before currently processed loop.

- All statements inside of a loop are recorder in two maps. This is StatmentOrderVisitor responsibility.
  map<int, tuple<tree::ParseTree*, bool, string, vector<string>>> statmentByOrder
  map<tree::ParseTree*, tuple<int, bool, string, vector<string>>> orderByStatement
  First one is key by order of the statements, starting with statement 0. Value consists of 4 informations: pointer to the statement node, marker - is statement removable, value on the left side of operator, vector of values on the right of the operator.
  In the second map is node pointer and statement order are reversed.
  Two maps allow to quickly find needed information either by order (which is important) or by node (when information about a certain statement is needed).

- LoopVisitor records all names of variables that are considered changing into a vector.

This data structures are needed to determine if a node can be removed.

Input/output description

Both input and output are .c files. Code needs to use proper grammar of the C language.
File path of input/output file can be either typed into console window or the file can be dragged and dropped onto an exe. In this case as well as when output file path is unspecified (giving “” as path), the output file will be named by input file name appended by “Fixed”.

IV. Functional test cases

title	Input	Output
Banned statements - loop invalidation		

use of goto	for(int r = 0 ; r< 10 ; r++)
	{
	    goto label;
	    x = 12;
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		    goto label;
		    x = 12;
		}

use of label	for(int r = 0 ; r< 10 ; r++)
	{
	    label: x = 12;
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		    label: x = 12;
		}

use of function	for(int r = 0 ; r< 10 ; r++)
	{
	        x = 12;
	        func();
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		        x = 12;
		        func();
		}

use of '*' unary operator	for(int r = 0 ; r< 10 ; r++)
	{
	        x = 12;
	        *z = 7;
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		        x = 12;
		        *z = 7;
		}

use of '&' unary operator	for(int r = 0 ; r< 10 ; r++)
	{
	        z = &y;
	        x = 12;
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		        z = &y;
		        x = 12;
		}

use of function in nested loops	for(int r = 0 ; r< 10 ; r++)
	{
	   x=12;
	   for(int r = 0 ; r< 10 ; r++)
	   {
	        func();
	   }
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		   x=12;
		   for(int r = 0 ; r< 10 ; r++)
		   {
		        func();
		   }
		}

examples of statements that should be removed		

simple	for(int r = 0 ; r< 10 ; r++)
	{
	        x = 12;
	}
		x = 12;
		for(int r = 0 ; r< 10 ; r++)
		{
		}

nested loops	for(int r = 0 ; r< 10 ; r++)
	{
	        x = 12;
	        for(int r = 0 ; r< 10 ; r++)
	        {
	                z=2;
	        }
	}
		x = 12;
		z=2;
		for(int r = 0 ; r< 10 ; r++)
		{
		        for(int r = 0 ; r< 10 ; r++)
		        {
		        }
		}

assignment of variable the does not change	for(int r = 0 ; r< 10 ; r++)
	{
	x = z;
	}
		x = z;
		for(int r = 0 ; r< 10 ; r++)
		{
		}

assignment of variable the does not change chain	for(int r = 0 ; r< 10 ; r++)
	{
	x = z;
	y = x;
	u = y;
	w = u;
	}
		x = z;
		y = x;
		u = y;
		w = u;
		for(int r = 0 ; r< 10 ; r++)
		{
		}

assignment of a statement that does not change	for(int r = 0 ; r< 10 ; r++)
	{
	y1 = z + u;
	y2 = z - u;
	y3 = z * u;
	y4 = z / u;
	y5 = 11 / u;
	y6 = z / u + w * x;
	}
		y1 = z + u;
		y2 = z - u;
		y3 = z * u;
		y4 = z / u;
		y5 = 11 / u;
		y6 = z / u + w * x;
		for(int r = 0 ; r< 10 ; r++)
		{
		}

different kind of loops	for(int r2 = 0 ; r2< 10 ; r2++)
	{
	x = 12;
	}
	
	while(y != 10)
	{
	x = 12;
	y += 1;
	}
	
	do
	{
	x = 12;
	y -= 1;
	}
	while(y != 0);
		x = 12;
		for(int r2 = 0 ; r2< 10 ; r2++)
		{
		}
		
		x = 12;
		while(y != 10)
		{
		y += 1;
		}
		
		x = 12;
		do
		{
		y -= 1;
		}
		while(y != 0);

examples of statements that cannot be removed		
operator that forces value change each iteration	for(int r = 0 ; r< 10 ; r++) { y+=1; }	for(int r = 0 ; r< 10 ; r++) { y+=1; }
assignment of variable that changes each iteration	for(int r = 0 ; r< 10 ; r++) { x = r; }	for(int r = 0 ; r< 10 ; r++) { x = r; }
use of 'x' operator that is effectively 'x=' operator	for(int r = 0 ; r< 10 ; r++) { y = y * x; }	for(int r = 0 ; r< 10 ; r++) { y = y * x; }
assignment of value that changes once, at the end of first iteration	for(int r = 0 ; r< 10 ; r++) { x = y; y = 12; }	for(int r = 0 ; r< 10 ; r++) { x = y; y = 12; }
statement inside conditional statement	for(int r = 0 ; r< 10 ; r++) { if(x == 12) { x =6; } }	for(int r = 0 ; r< 10 ; r++) { if(x == 12) { x =6; } }
assigning value that might change each iteration	for(int r = 0 ; r< 10 ; r++) { if(false) { w = r; } u = w; }	for(int r = 0 ; r< 10 ; r++) { if(false) { w = r; } u = w; }

change of scope	int x =5;
	
	for(int r = 0 ; r< 10 ; r++)
	{
	    int x;
	    x = 12;
	}
		int x =5;
		
		for(int r = 0 ; r< 10 ; r++)
		{
		    int x;
		    x = 12;
		}

assigning statement that uses changing variable	for(int r = 0 ; r< 10 ; r++)
	{
	    y = z * r;
	    u = z * r + 12;
	    w = 12 + r;
	}
		for(int r = 0 ; r< 10 ; r++)
		{
		    y = z * r;
		    u = z * r + 12;
		    w = 12 + r;
		}

more complex examples		

variables are checked if they can be removed in each loop	for(int r = 0 ; r< 10 ; r++)
	{
	    z+=1;
	
	    for(int r = 0 ; r< 10 ; r++)
	    {
	        x = u;
	        y = z;
	    }
	}
		x = u;
		for(int r = 0 ; r< 10 ; r++)
		{
		    z+=1;
		    y = z;
		    for(int r = 0 ; r< 10 ; r++)
		    {
		    }
		}