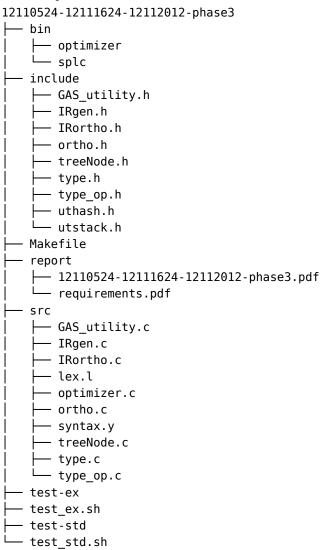
CS323 SPL Parser - Phase 3

陈康睿(A)樊斯特(A)肖佳辰(A)121105241211162412112012

1 Quick Start

1.1 Project Structure



1.2 Environment

Tool	Version	
С	C99	
Bison	3.0.4+	
Flex	2.6.4	

1.3 How to Run

Under the root folder of this project, execute the following commands to build the compiler executable splc under ./bin folder (gcc by default):

- make splc: Build the compiler, and clean the intermediate files.(For Bison 3.0.4)
- make splcb: Build the compiler, and clean the intermediate files.(For Bison 3.8.2)
- make splcc: Build the compiler, reserve the intermediate files.(For Bison 3.8.2)
- make splct: Build the compiler using Tiny C Compiler (TCC).(For Bison 3.8.2)

Execute make clean to clean the ./src and ./bin folder.

For an SPL source file filename.spl, execute ./bin/splc filename.spl to write the output into filename.ir0; or execute ./bin/splc filename.spl result.out to write the output into the given file result.out.

The optimizer is compiled into a separated executable file ./bin/optimizer. Executing ./bin/optimizer filename.ir0 will generate a optimized IR code file filename.ir.

The generated IR code can be executed in the IR simulator with command irsim filename.ir -i [input].

2 Basic Features

2.1 TAC Generation

In Phase 3, our compiler translates SPL into an linear intermediate representation, three-address code that can be executed on the IR simulator.

During translation, the compiler translates the variables into abstract names as v[num], splits combound operations into multiple instructions, and introduces temporary variables t[num] to hold the intermediate results. When reading or writing the arrays and members of a structure, the compiler calculates the offset relative to the pointer, and then accesses the value.

We maintain the intermediate representation in a tree structure. For each grammar specified in the previous phases, we implement a translate function build_[tokenName]_IR_tree in IRgen.h to create an intermediate representation node according to the give node from semantic tree. Each node of the IR tree maintains a stmt attribute and the connections to its children and sibling nodes.

Also, the variables and temporary values are maintained by an orthogonal linked list, supporting local scopes and shadowing definition using a same variable name. The function of the list here is a map from variable name to v[name].

2.2 IR Optimizer

To improve the efficacy of raw IR codes, we developed an optimizer designed to optimize these codes. It will read generated IR codes and organize them into a list, then do operations on list add and delete. We operate optimization in the following steps:

- Eliminate redundent GOTO which just goto the next line.
- Directly assign constant values to variables which may need multi-statements to calculate.
- If two variables store the same value, we use only one to present those.
- Remove the variables that not using after.

3 Bouns Features

3.1 Structure as Parameter

The compiler accepts program containing functions with structure as parameters. Similar to array, the member of a structure is accessed using offset. We implement pass-by-reference feature for structure

parameters, meaning that modifying the structure inside a function will synchronously change the structure in the caller's scope.

3.2 Multi-Dimensional Array as Parameter

The compiler also accepts program containing functions with multi-dimensional arrays as parameters. The implementation is similar to the structure parameter. Therefore, structure with multi-dimensional arrays and another structure as members can also be a function parameter.

3.3 for, continue and break

This feature is implemented in the previous 2 phases so we continue to generate IR for these keywords. for is handled as loop, while break and continue is implemented by adding GOTO instructions.

4 Explanations for Extra Cases

1. test 12111624 1.spl

This program calculates the fibonacci series with matrix acceleration. The time complexity is $O(\log N)$ and the space complexity is O(1).

This program contains bonus features like structure variables in parameters and multi-dimensional arrays as local variables.

Performance comparasion:

n	Recursive Fib	Iterative Fib	Matrix-Accelerated Fib	MA Fib with Optimization
10	1050	130	2808	
20	*	?	3213	
20000	*	*		

2. test_12111624_2.spl

This program is a DFS-based maze solver, outputing 1 if the maze is solvable otherwise 0. The recursive solver function accepts a structure with 2D array member as parameter.

This program is a simple sample showing the feature of continue and break keywords.

This program is an example of Gaussian elimination, containing a function with a 2D matrix as the parameter.

This program is an example showing local scope shadowing. An inner array has the same name as the outer array, accessing inside/outside the block with the same name corresponds to accessing inner/outer array.