Use manual for our compiler

Zhangzheng Zheng Qingwen Chen

April 21, 2010

1 Scan and Parse

1.1 Lexical analysis ⟨position: ./src/scanparse/civc.l⟩

Nothing special.

1.2 syntactic analysis ⟨position: ./src/scanparse/civc.y⟩

Nothing special.

2 Phases/sub-phases

2.1 Print ⟨position: ./src/print/print.c⟩

This is the sub-phase right after we build our abstract syntax tree (AST) with lex and yacc. Its purpose is to visualize the AST by printing it to stdout as a Civc program. We traverse the AST, and each time we reach a node, we print out the information stored in that node in the format specified by civc grammar. The printing decision is made according to the attributes of the node, such as type of the node, status of its sons etc. For example, for a while-loop node which has a pred-son as the condition, a body son to indicate the body of the while-loop, we first print out '`while (`' when we reach this node, then traverse to its pred-son and print '`) {` ' after that. Next, we traverse the body-son and print '}' at the end.

In this sub-phase, we also visualize the precedence of binary/unary and cast operations by including the operation in a pair of brackets.

2.2 Context analysis ⟨position: ./src/contextanalysis/contextanalysis.c⟩

Context analysis deals with the following two tasks:

- Find the declaration of each used variable, and add a pointer to the corresponding declaration. If a variable is used without declaration, report semantic error and indicate which line the error is.

- For each function call, find the corresponding declaration and add a pointer in the function-call node to point to its declaration. An error message is generated if no declaration is found.

2.3 Type check ⟨position: ./src/typecheck/typecheck.c⟩

There are three kinds of type checking we should do in this sub-phase: type matching for binary/unary operations as well as assignment, type of return/argument matching in functions. Here is a list about what we have done in this sub-phase:

		Allowed types	Extra restrictions
Binop	$+, -, *, /$	int, float	right matches left
	%	int	
	$>, >=, <, <=$	int, float	
	&&, ||	bool	
Monop	$-$	int, float	right matches left
	!	bool	
Function	arguments	int, float, bool	types of arguments match parameters' in the declaration
	return	int, float	returned type matches its declared one. If the function is declared to return void, no return statement exists in the function body.

2.4 Assembly code generation

This is the final phase of our compiler. We implemented Mileston 8 and 9 together by traversing the AST twice in this phase. The first traversal was used to calculate the offset for *jump* instructions such as *jump*, *branch_f* and *branch_t*. The offset was then used to generate the assembly code.

3 Problems