Concordia University
Department of Computer Science
and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2018

Final Project Presentation Grading Sheet

Deadline:	April 14-17, 2016
Evaluation:	30% of final grade
Late submission:	not accepted

Instructions

You must deliver an operational version demonstrating the integrated capacities of your compiler. This is about demonstrating that your project has been effectively aimed at solving specific project problems. The tasks involved in building a working compiler have been identified, listed, and attributed some individual marks. The objective of your presentation is to demonstrate by usage the extent to which your compiler is achieving the list of tasks.

During the presentation, you have to do an individual demonstration of each functional requirement as listed on the following grading sheet. For each functional requirement, you are expected to come prepared with at least one test case dedicated to its demonstration. You are thus also graded according to how effectively you can demonstrate that the listed features are implemented. Negative marking will be applied in cases of ineffectiveness of demonstration or lack of preparation, up to a maximum of -10%.

If you cannot really demonstrate the features through execution, you will have to prove that the features are implemented by explaining how your code implements the features, in which case you may be given some marks. Even in such cases, you have to demonstrate that you are well prepared for the presentation, and that you can easily provide clear explanations as questions are asked about the functioning of your code.

The presentation also includes the evaluation of functional requirements and graduate attributes. For each grading element listed, you are given a letter grade. The letter-to-numeric grade correspondence is the following: A:100% — perfect solution, B:75% — partial solution, C:50% — marginal solution, F:0% — not done

Identification

Student Name	Evaluator Name	Evaluator Signature	Presentation Time

	Functional Requirements	letter		
1	USER INTERACTION		5	
1.1	input interface: accepts user-provided file name as opposed to hard-coded file name		1	
1.2	output interface: clarity/usefulness of standard output, clarity/usefulness of alternate output to different files		2	
1.3	all errors are reported in synchronized order, even if different phases are implemented and errors are found in different phases		2	
2	LEXICAL ANALYSIS		5	
2.1	tokenizing			
1.1.1	integers and floating point numbers (valid/invalid numbers according to assignment 1 handout)		1	
1.1.2	comments: inline comments, block comments, unending block comments, nested block comments		1	
1.2	error detection/reporting/recovery			
2.2.1	error detection: detecting all lexical errors in a given program		1	
2.2.2	error reporting: accurate reporting of errors, including line number and useful description of the error		1	
2.3	output			
2.3.1	output of token stream in a separate file		1	
3	SYNTACTIC ANALYSIS		30	
3.1	parsing			
3.1.1	variable declarations: int, float, class types, array, array of class types		1	
3.1.2	program function		1	
3.1.3	free functions		2	
3.1.4	member function definitions		2	
3.1.5	class declarations: data member declarations, method declarations, inheritance list		2	
3.1.6	complex expressions (all arithmetic, relational and logic operators in one expression)		3	
3.1.7	conditional statement, including nested if without brackets		2	
3.1.8	loop statement, including nested for without brackets		2	
3.1.9	get(var) / put(expression) / return(expression) statements		2	
3.1.10	access to class members, including multiply nested and including arrays		3	
3.1.11	access to arrays: uni- and multi-dimensional, using expressions as index		2	
3.2	error detection/reporting/recovery			
3.2.1	error detection: detect all syntax errors in a program		1	
3.2.2	error reporting: accurate reporting of errors, including line number and useful description of the error		1	
3.2.3	error recovery: implementation of an effective syntax error recovery mechanism		2	
3.3	output			
4.3.1	output a derivation of the compiled program in a separate output file		2	
4.3.2	output the AST of the compiled program in a separate output file		2	
5	SYMBOL TABLE CREATION AND SEMANTIC CHECKING		30	
5.1	symbol table creation			
5.1.1	AST tree traversal that triggers semantic actions		3	
5.1.2	global scope symbol table.		1	
5.1.3	entry in the global table for each class declared. Local tables for classes.		1	
5.1.4	entry in the appropriate table for each function definition (free functions and member functions). Local tables for each.		1	
5.1.5	entry in the appropriate table for each variable defined in a scope, i.e. class data members or function's local variables.		1	
5.2	semantic/type checking			
5.2.1	type checking on expressions, assignment and return statements.		3	
5.2.2	checking of type and number of parameters upon a function call		2	
5.2.3	use of an array variable should be made using the same number of dimensions as declared in the variable declaration.		1	
5.2.4	expressions used as an index must be of integer type.		1	
5.2.5	circular class dependencies (through data members or inheritance) should be detected and not be allowed		2	
5.2.6	the "." Operator should be used only on variables of a class type.		1	
5.2.7	forward references for classes/free functions		1	
5.2.8	the type of the operand of a return statement must be the same as declared in its function's declared return type		1	
5.2.9	undeclared function: call to a function that is not declared (free function or member function)		1	
5.2.10	undefined function: declaring a member function that does not have a corresponding function definition		1	
5.2.11	undeclared variable: use of a local variable name for which there is no declaration		1	
5.2.12	undeclared data member: reference to a data member that is not declared (including in superclasses or deeply nested)		3	
5.2.13	multiply declared variable: an identifier cannot be declared twice in the same scope.		1	
5.3	error detection/reporting/recovery			
5.3.1	error detection : detect all semantic errors in a program		1	
5.3.2	error reporting : accurate reporting of errors, including line number and useful description of the error		1	
5.4	output			
5.4.1	output of the symbol table structure of the compiled program in a separate file		2	
6	CODE GENERATION		40	
6.1	memory allocation			
6.1.1	allocate memory for basic types (integer, float).		1	
6.1.2	allocate memory for arrays of basic types.		1	
6.1.3	allocate memory for objects.		1	
6.1.4	allocate memory for objects with inheritance.		1	
6.1.5	allocate memory for objects having object members.		1	
6.1.6	allocate memory for arrays of objects.		1	
6.2	functions			
6.2.1	branch to a function's code block, execute the code block, branch back to the calling function.		2	
6.2.2	branch to a function that has been branched upon.		1	
6.2.3	pass parameters as local values to the function's code block.		1	
6.2.4	upon function resolution, pass the return value back to the calling function.		1	
6.2.5	function call stack mechanism and recursive function calls.		2	
6.2.6	call to member functions.		2	
6.2.7	call to deeply nested member function.		1	
6.2.8	passing/returning an array or object to/from a function		1	
6.3	statements			
6.3.1	assignment statement: correct assignment of the resulting value of an expression to a variable, independently of what is the expression.		2	
6.3.2	conditional statement: correct implementation of branching mechanism, including for imbricated conditional statements.		2	
6.3.3	loop statement: correct implementation of branching mechanism, including for imbricated loop statements.		2	
6.3.4	input/output statement: get()/put()		2	
6.4	access to elements of aggregate data types (array, object)			
6.4.1	arrays of basic types (integer and float), access to an array's elements, single or multidimensional.		1	
6.4.2	arrays of objects, access to an array's object elements, single or multidimensional.		1	
6.4.3	objects, access to members of basic types.		1	
6.4.4	objects, access to members of array types, as well as the elements of the array.		1	
6.4.5	objects, access to members of object types, as well as the elements of the object.		1	
6.4.6	objects, access to the members of a superclass.		1	
6.5	expressions			
6.5.1	computing the value of an entire expression.		2	
6.5.2	expressions involving all of: arithmetic, relational and logic operators in one expression		2	
6.5.3	expression involving an array factor whose indexes are themselves expressions.		1	
6.5.4	expression involving an object factor referring to deeply nested object members.		1	
6.5.5	memory allocation for temporary results.		1	
6.6	output			
6.6.1	output of the generated code of the compiled program in a separate file		2	
	Functional Requirements — Total		110	

Graduate attributes		letter		
Attribute 1: Knowledge-base for Engineering	Indicator 1.2: Show competence in tackling advanced engineering problems: Demonstrate understanding of the theoretical basis of the implementation.		2	
Attribute 2: Problem analysis	Indicator 2.1: Problem identification and formulation: Demonstrate that the implementation follows the original specifications. Demonstrate that the problem is clearly and completely understood,		2	
	Indicator 2.2: Modeling: Explain what models were used to analyze and implement the lexical/syntactical/semantic specifications.		2	
Attribute 4: Design	Indicator 4.1: Problem identification and information gathering: Demonstrate that the solution is well-adapted to the problem, and that unstated parts of the problem were uncovered as part of the development process.		2	
	Indicator 4.3: Architectural and detailed design: Description of the rationale and structure of the architectural design and detailed design justified against project requirements/constraints.		2	
Attribute 5: Use of Engineering tools	Indicator 5.2: Ability to evaluate and select appropriate tools: Justified adoption of tools in the project (e.g. programming language, compiler, IDE, libraries, project management tools, grammar analysis tools, etc).		1	
	Indicator 5.3: Ability to use tools: Proficient use of particular tools for the analysis and implementation.		2	
Attribute 7: Communication skills	Indicator 7.4: Oral presentation: Structure and demonstrated preparation of presentation, using appropriate presentation techniques. Demonstrated knowledge of code base/clarity of explanations.		2	
	Graduate Attributes — Total		15	

Grand Total		125	