Description of the tests

In order to incrementally build the compiler, a list of program tests has been proposed. This table summarizes what are the new elements of the Asl language that appear in each test with respect to the previous one.

Some indications on modifying compiler components are also given, but should be taken *just as a guide*.

Tests for parsing and type check

The following tests can help you to define the grammar of the whole ASL language and to complete the semantic analysis (*SymbolsVisitor* and *TypeCheckVisitor* modules). Some changes in the ASL grammar will produce errors executing *g++* on *CodeGenVisitor.cpp*. Adjust this module when necessary.

Test	New structures / new checks	Changes in Asl grammar	Changes in SymbolsVisitor	Changes in TypeCheckVisitor	Adjust in CodeGenVisitor
`jp_chkt_01`	Other basic types, new operators, values, parenthesis, ... Use of *type coercion*	Update rule *type* and rule *expr*. New tokens, floats, ...	Update variable declaration (new basic types). See module *TypesMgr*	Type check of new expressions. *Type coercion* from *int* to *float* (update both .h and .cpp)	
`jp_chkt_02`	Multiple variable declarations in a single line. (new token *true*)	Update variable declaration	Update variable declaration	(new value *true*)	Fix access to the text of the *first* variable to avoid compilation error
Additional statements					
`jp_chkt_03`	*while* and *return* statements. *else* in conditionals. (new operator *!=*)	Update rule *statement*		Type check of new statements (update both .h and .cpp) Fix access to *statement* in *if* children (there are two now)	Fix access to *statement* children in *if* (there are two now) to avoid compilation error
Function declaration with parameters					
`jp_chkt_04`	Functions may have *parameters* and return values	Update function declaration (use new rules). Update return statement	Update *visitFunction* to create the appropriate function type. See module *TypesMgr*. **Hint:** add new visit methods to process parameters		
Array access					
`jp_chkt_05`	Array access in expressions and in *l-value* expressions	Update rules *left_expr* and *expr*		Add new visitors for array access in *left_expr* and in *expr* (update both .h and .cpp)	Modify the visitor for rule *left_expr* to use the new visit method

Test	New structures / new checks	Changes in Asl grammar	Changes in SymbolsVisitor	Changes in TypeCheckVisitor	Adjust in CodeGenVisitor
Function call					
`jp_chkt_06`	Call to function as a form of expression. Functions without parameters	Update rule *expr* to accept function calls		Add new visitors for function call. Type check the expression appropriately (update both .h and .cpp)	
`jp_chkt_07`	Call to function with parameters, without errors in arguments. No *main* function detection (automatic)			Type check arguments (*actual* parameters)	
`jp_chkt_08`	Call to procedures with parameters, without errors in arguments. Inappropriate calls to procedures and functions, regardless of arguments	Update statements		Type check of function call and compute expression type. Type check arguments (*actual* parameters)	
`jp_chkt_09`	Incorrect use of a function. *l-value* expressions in *read*			Compute properly the *IsLValue* decoration of the identifiers (probably nothing)	
Array declaration					
`jp_chkt_10`	Array declarations	New structured *type*. **Hint:** use 2 rules, *type* and *basic_type*	Add new visit methods to process types. See module *TypesMgr*. Note: remember that functions only return *basic* types (update both .h and .cpp)	Probably nothing	
`jp_chkt_11`	Use of *read* statement			Probably nothing if type check of array access is correct	
Return statement					
`jp_chkt_12`	Extensive use of *return* statement in procedures and		Note: remember that the first declaration of an identifier *in a*	Update type check of *return* statement. **Hint:** the return type of the current	

Test	New structures / new checks	Changes in Asl grammar	Changes in SymbolsVisitor	Changes in TypeCheckVisitor	Adjust in CodeGenVisitor
	functions, with and without *type coercion*.		*given scope* prevails over the rest (on *that scope*)	function can be obtained from SymTab in *visitFunction*, and saved/retrieved with the methods *setCurrentFunctionTy* and *getCurrentFunctionTy* (available in TypeCheckVisitor).	
`jp_chkt_13`	Check undeclared identifiers				
Errors in parameters					
`jp_chkt_14`	Check that parameters must be declared in the *local scope*, with re-declarations of symbols in the same scope. Also, function calls may have errors due to the type of the *arguments*		Nothing if *visitFunction* was properly updated in the test `jp_chck_04`. Note: remember that the first declaration of an identifier *in a given scope* prevails over the rest (on that *scope*)	In a function call, remember that arguments (*actual parameters*) must always be processed. Now, also type check *actual* vs. *formal* parameters See module *SemErrors*	
`jp_chkt_15`	Similar to `jp_chck_14` and and previous ones: function calls with parameters and arrays involved				
`jp_chkt_16`	Similar to `jp_chck_15` and the previous ones: function calls with parameters and arrays involved				
`jp_chkt_17`	Similar to `jp_chck_16` and the previous ones: function calls with				

Test	New structures / new checks	Changes in Asl grammar	Changes in SymbolsVisitor	Changes in TypeCheckVisitor	Adjust in CodeGenVisitor
	parameters, assignment to to an array, ...				
`jp_chkt_18`	Function and procedure calls may have a wrong number of arguments			Type check function and procedure calls to detect also these errors. See module *SemErrors*	
`jp_chkt_19`	Use of operator *modulo* (%)	(Update rule *expr*)		Type check the operator *modulo*	
Operations on arrays					
`jp_chkt_20`	Array assignments, arrays as parameters, *type coercion*, ... Similar to previous tests			Probably nothing	

Tests for code generation

The following tests can help you to complete the code generation (*CodeGenVisitor* module). Some changes may involve your ASL grammar and/or type check. Adjust *Asl.g4* and *TypeCheckVisitor* module if necessary.

Test	New structures / new code	Adjust in Asl grammar	Adjust in TypeCheckVisitor	Changes in CodeGenVisitor
`jp_genc_01`	Multiple variable declarations in a single line. Other basic types, new operators, values, parenthesis,... *Type coercion* in some expressions. Extend `write` statement with new types			Update variable declaration. Update code for `write` statement. Generate code for new expressions, with *coercion* `int` $\rightarrow$ `float` in *arithmetic* operators. (Update both .h and .cpp). See module *code*
`jp_genc_02`	Use of `while` statement. New operators >, < (without *type coercion*). Boolean values `true` and `false`	If necessary, add value `false`	If necessary, type check the new value	Generate code for `while` statement, and for new expressions
Function call with parameters				
`jp_genc_03`	Functions may have *parameters* and return values. Call to a function as a form of expression (only parameters of *basic types*). Use of `return expr` statement (without *type coercion*)			Update function declaration (`visitFunction`). Add new visitors for function call, and generate code for this expression appropriately (update both .h and .cpp). See module *code*

Test	New structures / new code	Adjust in Asl grammar	Adjust in TypeCheckVisitor	Changes in CodeGenVisitor
`jp_genc_04`	Call to a procedure with *parameters* of *basic types* (without *type coercion*). New operator <= with *type coercion*			Update procedure call to allow *parameters*. Relational operators with *coercion* `int` → `float`
Array declaration and access				
`jp_genc_05`	Use of *array* type in declarations of local variables. Array access in expressions, and in *l-value* expressions. `if-then-else` statement. Function calls with *type coercion* in parameters	If necessary, complete the `if` statement with `else` branch	If necessary, complete the type check of the `if` statement	Update *assignment* statement: now a value can be assigned to an array position. Update `if` statement. Update function call to allow *coercion* in actual *parameters*
`jp_genc_06`	*Type coercion* in assignments, and in actual parameters of procedure calls. Use of `read` statement into non `int` expressions. Call to functions discarding the result (like a procedure)			Update *assignment* to allow *coercion*. Update `read` statement. Update code generation in "procedure" calls
Arrays as parameters				
`jp_genc_07`	Use of parameters of type `array`: access to the value and to the address of an array position. Procedure calls with actual parameters of type array (passed by *reference*)			Update procedure call to allow actual *parameters* of type *array* (passed by *reference*). Only local arrays are passed.
`jp_genc_08`	Function calls with actual parameters of type array (passed by *reference*)			Update function call to allow actual *parameters* of type *array* (passed by *reference*). Only local arrays are passed
`jp_genc_09`	Definition and use of the function `factorial`			Probably nothing
`jp_genc_10`	Definition and use of the function `prod_escalar` (*dot product* of two arrays)			Probably nothing, but check the pass of arrays as parameters
`jp_genc_11`	Extend `write` statement with *char* expression. Operator *modulo*. Values of type *char*			Update `write` statement. Add *modulo* operator in *arithmetic* expressions. Generate code for new values
Array assignment				

Test	New structures / new code	Adjust in Asl grammar	Adjust in TypeCheckVisitor	Changes in CodeGenVisitor
`jp_genc_12`	Array assignment (*a* = *b*), where *a,b* may be local variables and/or parameters			Update assignment statement
`jp_genc_13`	Additional checks of arrays passed as parameters, where the array is not the first param.			Probably nothing
`jp_genc_14`	Unary operator +			Generate code for this expression