Report – Phase 4

Project - Compiler

LEWONCZUK Louise – 2020-81806 and MASSARDIER Baptiste – 2020-82265

V2 – 2020/11/26

LEWONCZUK Louise – 2020-81806
MASSARDIER Baptiste – 2020-82265

Project phase 4

Summary

Project phase 4

Introduction

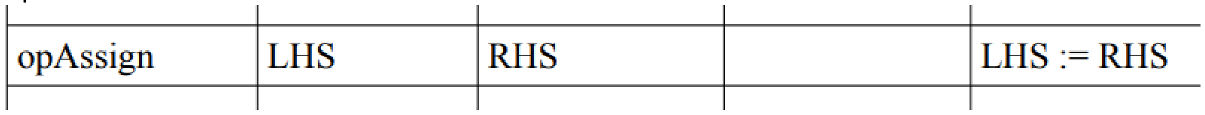
This part is done after having created the AST in the precedent phase thanks to the scanning and the parsing done before. In this phase, the goal is to generate an intermediate code. In order to do that, we have implemented the ToTac function of the class AST.

Project phase 4

Intermediate Code Generation

Statement

`CTacAddr* CAstStatAssign::ToTac(CCodeBlock *cb, CTacLabel *next)`
The implementation is rather simple, as we just need to follow the SnuPL/2 intermediate representation:

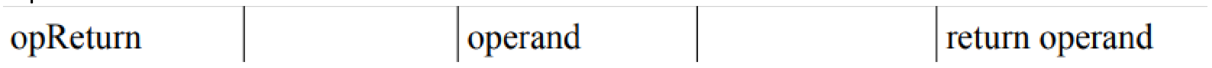


opAssign	LHS	RHS		LHS := RHS

`CTacAddr* CAstStatCall::ToTac(CCodeBlock *cb, CTacLabel *next)`
This function generates for each argument, a TAC corresponding to this argument. At the end, it also generates the TAC to call the function.

opCall	result	target		result := call target

`CTacAddr* CAstStatReturn::ToTac(CCodeBlock *cb, CTacLabel *next)`
If the return expression is not null, then we add it to the TAC, again as in the SnuPL/2 intermediate representation.



opReturn		operand		return operand

`CTacAddr* CAstStatIf::ToTac(CCodeBlock *cb, CTacLabel *next)`
This one is a little trickier, indeed, we have to execute the if body only if the condition is true. So, we made two labels, one if the condition is true, and one if the condition is false. If it is true, then we execute the if body and then jump to the next. If it is false, we jump to the label false and execute only the else part (if exist) and then go to next.

`CTacAddr* CAstStatWhile::ToTac(CCodeBlock *cb, CTacLabel *next)`
This one is event trickier. Indeed, we firstly need to express the while as a if condition. The expression is then a if statement, if the expression is met, then we execute the code one time. Otherwise, we go to next. At the end of the code, we just need to add an unconditional jump to the beginning.

Expression

`CTacAddr* CAstBinaryOp::ToTac(CCodeBlock *cb)`
Depending if the operator is a binary or no, the function does not return the same thing. Indeed, if the operator is not binary, we save the result on a variable and return it. Otherwise, the result is boolean. In this case, the expression is evaluated and the function simply returns a 1 in case the expression is true, a 0 otherwise.

`CTacAddr* CAstBinaryOp::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
If the operator is a relational one, then we evaluate if the result is trus, if it is, we jump to the true label, we go to the false one otherwise.

Project phase 4

If the operator is not a relational one, then it is either opAnd or opOr. In these cases, we evaluate the result lazily.
For the and operator, it is :

```
   if a then goto test_b
   goto lbl_false
 test_b:
   if b then goto lbl_true
   goto lbl_false
```

For the or one, it is the same, but going to label true if a is true, and going to test_b if a is false.

`CTacAddr* CAstSpecialOp::ToTac(CCodeBlock *cb)`
This function just follow the SnuPL/2 intermediate representation:

opAddress	result	operand		result := &operand
opCast	result	operand		result := (type)operand
opDeref	*result*	*operand*		*result := *operand – not used*
opWiden	*result*	*operand*		*result := (type)operand – not used*
opNarrow	*result*	*operand*		*result := (type)operand – not used*

`CTacAddr* CAstStatement::ToTac(CCodeBlock *cb, CTacLabel *next)`
This function is given.

`CTacAddr* CAstUnaryOp::ToTac(CCodeBlock *cb)`
Depending on the unary operator, the output of this function is a boolean or a constant so firstly we check the operand.

`CTacAddr* CAstUnaryOp::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
This function called the precedent one and in the case opNot swap the argument ltrue and lfalse.

`CTacAddr* CAstFunctionCall::ToTac(CCodeBlock *cb)`
This function looks at the number of arguments of the function that is called and for each argument it creates a new instruction. After that, it creates the instruction to call those parameters.

`CTacAddr* CAstFunctionCall::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
This function calls the precedente one and saves it. Then it looks if this value is true then goto ltrue label otherwise goto lfalse label.

`CTacAddr* CAstDesignator::ToTac(CCodeBlock *cb)`
This function returns the symbol's name of cb.

`CTacAddr* CAstDesignator::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
This function calls the precedente one and saves it. Then it looks if this value is true then goto ltrue label otherwise goto lfalse label.

Project phase 4

`CTacAddr* CAstConstant::ToTac(CCodeBlock *cb)`
This function returns the constant.

`CTacAddr* CAstConstant::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
This function looks at the constant value as a condition and if it's true then goto ltrue label otherwise goto lfalse label.

`CTacAddr* CAstStringConstant::ToTac(CCodeBlock *cb)`
This function returns the symbol of cb.

`CTacAddr* CAstStringConstant::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
This function calls the precedence function to return the symbole cb.

Translation of array access

`CTacAddr* CAstArrayDesignator::ToTac(CCodeBlock *cb)`
This function is the most complex one. It deals with an array of designators. It computes the pointer of the data and references it to the pointer calculated by the function. The different steps are the following. Firstly, if needed, we link a pure array to a pointer. After that we use the formula of the dragon book as follows: A[i1][i2]...[ik] = (base) + ((... (i1 n2+ i2)n3+ i3) ... nk+ ik) *w

The base corresponds to the element located at &A + DOFS(A) ; nk= DIM(&A, k + 1) for k in range (1, n-1) ; w is the size of the basetype. To do that, we create some temporary variables in order to store the different designators contained in the array.

At last, thanks to CTacReference, we reference the array location.

`CTacAddr* CAstArrayDesignator::ToTac(CCodeBlock *cb, CTacLabel *ltrue, CTacLabel *lfalse)`
This function calls the precedente one and saves it. Then it adds two instructions: one for the equal and one Goto lfalse.

Project phase 4

Evaluation

During the test phase, we got several weird issues with tests files. Indeed, our parser starts receiving tUndefined at each line return. However, it's working with olds test files. By comparing two same files, one from phase 2 and one from this phase, I saw that the new one did not have carriage return at the end of each line. As we use the compiled scanner, we sadly can not resolve this issue.

Project phase 4

Conclusion

This project brought us knowledge on many different aspects. The most important contribution was on the management of complex code. Indeed, with this intermediate code generation we need to take into account the output of the AST with precision and caution. Moreover, this part is primordial for the code generation we will have to code in the last phase. However, the part of the code needed a lot of visualization and understanding of the CTac class so it was hard to implement some function especially the one for the array designator.

For the organization and way to do, we have improved too. We know each other more and especially our strength and weakness so we can split the work more efficiently.

For this phase we had decided to use 3 grace days as the deadline had been moved from the 23th at 23:59 and we submitted this work on the 26th.