



Free University Compiler Project Presentation

Final Presentation

Team A

Project (19517e)

Instructor: Prof. Dr. Elfriede Fehr

Institute of Informatik

FU-Berlin

11.07.2013

Motivation

Project Management

- Free University Compiler-Teams
- Development Environment

Design and Modules

- Lexical Analysis
- Parsing
- Semantic Analysis
- Intermediate Code Generation (TAC Generation)
- Backend - LLVM
- Backend - Exceptions
- Backend - Example program

Evaluation

- Evaluation: Testing levels
- Evaluation: Results
- Summary for milestone 3

GUI Demo

Lessons Learned

Outline

Motivation

Project Management

- Free University Compiler-Teams
- Development Environment

Design and Modules

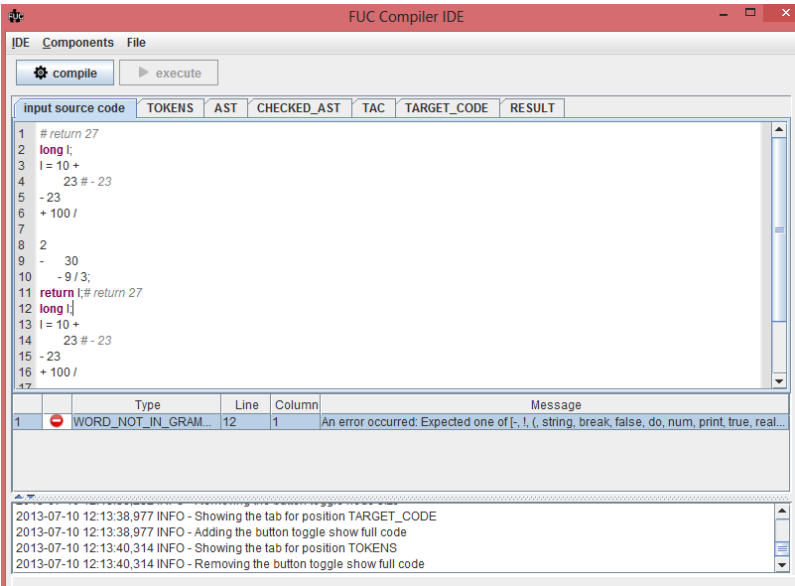
- Lexical Analysis
- Parsing
- Semantic Analysis
- Intermediate Code Generation (TAC Generation)
- Backend - LLVM
- Backend - Exceptions
- Backend - Example program

Evaluation

- Evaluation: Testing levels
- Evaluation: Results
- Summary for milestone 3

GUI Demo

Lessons Learned



The screenshot shows the FUC Compiler IDE interface. At the top, there are menu items 'IDE', 'Components', and 'File'. Below the menu is a toolbar with 'compile' and 'execute' buttons. The main area is divided into tabs: 'input source code', 'TOKENS', 'AST', 'CHECKED_AST', 'TAC', 'TARGET_CODE', and 'RESULT'. The 'input source code' tab is active, displaying the following code:

```
1 # return 27
2 long i;
3 i = 10 +
4     23 # - 23
5 - 23
6 + 100 /
7
8 2
9 - 30
10 - 9 / 3;
11 return i; # return 27
12 long i;
13 i = 10 +
14     23 # - 23
15 - 23
16 + 100 /
17
```

Below the code editor is a table with columns 'Type', 'Line', 'Column', and 'Message'. The first row shows an error:

Type	Line	Column	Message
WORD_NOT_IN_GRAM...	12	1	An error occurred: Expected one of [-, !, (, string, break, false, do, num, print, true, real...

At the bottom of the IDE, there is a log window showing the following messages:

```
2013-07-10 12:13:38,977 INFO - Showing the tab for position TARGET_CODE
2013-07-10 12:13:38,977 INFO - Adding the button toggle show full code
2013-07-10 12:13:40,314 INFO - Showing the tab for position TOKENS
2013-07-10 12:13:40,314 INFO - Removing the button toggle show full code
```

Outline

Motivation

Project Management

- Free University Compiler-Teams
- Development Environment

Design and Modules

- Lexical Analysis
- Parsing
- Semantic Analysis
- Intermediate Code Generation (TAC Generation)
- Backend - LLVM
- Backend - Exceptions
- Backend - Example program

Evaluation

- Evaluation: Testing levels
- Evaluation: Results
- Summary for milestone 3

GUI Demo

Lessons Learned

Project Management

- Planning
 - ▶ We form 7 sub-teams, each sub-team has a leader to communicate with the team leader
 - ▶ We communicate per email, Skype, and meeting each Thursday
 - ▶ Schedule deadline for each milestone (M1, M1, and M3)
 - ▶ Each member has to report his status on the meeting
- Implementation, testing, and documenting
 - ▶ We breakdown the task into sub-tasks (e.g., Lexical Analysis, Parsing,...)
 - ▶ We define many levels of testing, bugs can be reported per email, ticket, etc.
 - ▶ Documentation is in Wiki
- Deployment and maintenance
 - ▶ ANT & CI (next slide)

- Each member chooses his own desired integrated development environment (IDE)
- One rule: The code in the master branch is always executable
- Each member can *push* to master branch
- Each sub-team has their own branch
- Test Driven
 - ▶ We use *ANT* for the build and the automated testing processes
 - ▶ We use *Travis CI* for the integration processes

Outline

Motivation

Project Management

Free University Compiler-Teams

Development Environment

Design and Modules

Lexical Analysis

Parsing

Semantic Analysis

Intermediate Code Generation (TAC Generation)

Backend - LLVM

Backend - Exceptions

Backend - Example program

Evaluation

Evaluation: Testing levels

Evaluation: Results

Summary for milestone 3

GUI Demo

Lessons Learned

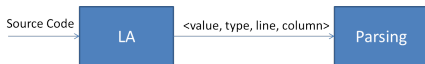
Requirements:

- The modules in the this project shall be able to be used in the project B (JavaBite) and vice versa
- Interfaces shall be provided to use in both projects

Modules

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Intermediate Code Generation(TAC Generation)
5. LLVM & Backend





- Design goals: Simplicity and Robustness
- Divide incoming code into lexical units
- Line separated token calculation, based on delimiters
- Send the token value, type, line, and column to Parser
- + Easy to extend regarding a new token type
- + A token is only calculated when we call *getNextToken()*



We use *Shift-Reduce parsing*, in particular the LR(1) variation

- Generator:
 - Determine the *First sets*, *Follow sets*, and the *lookahead* from the given grammar
 - Generate an LR(1) automaton
- SHIFT & REDUCE:
 - SHIFT: e.g., $ABC \mid xyz \Rightarrow ABCx \mid yz$
 - REDUCE: e.g., If $A \rightarrow xy$, then $Gbxy \mid ijk \Rightarrow GbA \mid ijk$
- + Support *Error Recovery* \Rightarrow user friendly
- + Easy to extend

Semantic Analysis

Goal: The semantic analysis looks for errors in the source file that are not described by the context-free grammar of the source language.

Features:

- Static Type Checking
 - ▶ e.g., this rule $\frac{\vdash e_1 : long \quad \vdash e_2 : long}{\vdash e_1 + e_2 : long}$ is valid
 - ▶ Widening casts/coercions is implicit (e.g., long \rightarrow double)
- *Break* statement, *break* is allowed only within a loop
- Dead Code Detection (e.g., statements after return)
- Array Bounds Check (e.g., accessing the index 11 of the array of range 10)
- Division by zero (e.g., $1/(3 - 3)$)
- Uninitialized Variables, explicitly initialize is required

- IR Generation:
 - ▶ Use *depth-first-search-left-to-right* algorithm for AST(Abstract Syntax Tree) traversal
 - ▶ Each type of the AST node is called by a particular method
 - ▶ Stacks are used to store the results of each method
 - ▶ Status of IRGen is stored in a stack
 - ▶ + Easy to extend
e.g., define a <new AST node type, new method>
 - ▶ + Full support the *Prog* language

What is LLVM?

- „(...) is a collection of modular and reusable compiler and toolchain technologies.“
- Consists of:
 - ▶ C++ libraries (API)
 - ▶ Command line tools
- Highly modular architecture
- Does IR generation, IR optimization, and code generation



Features of LLVM

- Strictly defined semantics + statically typed language allow for easy interaction with other, LLVM compatible languages (C/C++/Haskell) via LLVM IR
- Good optimization support for free (via command line tools, or via API)
- Actively developed and supported
- LLVM IR is mostly cross platform (as opposed to assembler or native machine code)
- Permissive license



- Generated LLVM IR code uses exception handling
- Currently implemented exceptions:
 - ▶ Division by zero
 - ▶ Array out of bounds
- Used C++ standard library functions:

```
declare i8* @__cxa_allocate_exception(i32)
declare void @__cxa_throw(i8*,i8*,i8*)
declare i32 @__gxx_personality_v0(...)
declare i32 @llvm.eh.typeid.for(i8*) nounwind readnone
declare i8* @__cxa_begin_catch(i8*)
declare void @__cxa_end_catch()
```

- No segfault, division by zero triggers exception instead
- Program terminates with information about exception
- Program in Prog:

```
long l;
long zero;
zero = 0;
l = 1 / zero;
print l;
```

- Program in LLVM IR:

```
define i64 @main() {
  %l = alloca i64
  store i64 0, i64* %l
  %zero = alloca i64
  store i64 0, i64* %zero
  store i64 0, i64* %zero
  %tmp1 = alloca i64
  store i64 0, i64* %tmp1
  %zero.0 = load i64* %zero
  %tmp1.0 = invoke i64 (i64, i64)* @div_long(←
    i64 1, i64 %zero.0) to label %tmp1.0.ok ←
    unwind label %UncaughtException
  tmp1.0.ok:
  store i64 %tmp1.0, i64* %tmp1
  %tmp1.1 = load i64* %tmp1
  store i64 %tmp1.1, i64* %l
  %tmp2 = alloca i8*
  %tmp2.0 = getelementptr [1 x i8]* @.←
    string_0, i64 0, i64 0
  store i8* %tmp2.0, i8** %tmp2
  %l.0 = load i64* %l
  %tmp2.1 = call i8* (i64)* @ltoa(i64 %l.0)
  store i8* %tmp2.1, i8** %tmp2
  %tmp2.2 = load i8** %tmp2
  call i32 (i8*, ...)* @printf(i8* %tmp2.2)
  ret i64 0

; (Uncaught exception handler not shown)
}
```

Outline

Motivation

Project Management

- Free University Compiler-Teams
- Development Environment

Design and Modules

- Lexical Analysis
- Parsing
- Semantic Analysis
- Intermediate Code Generation (TAC Generation)
- Backend - LLVM
- Backend - Exceptions
- Backend - Example program

Evaluation

- Evaluation: Testing levels
- Evaluation: Results
- Summary for milestone 3

GUI Demo

Lessons Learned

How good is our program?

- Goal of the test coverage $\geq 90\%$ for each module
- At the meeting each team report their test results & problems
- We perform these testing levels:
 1. Module tests
 2. Integration tests
 - ▶ Runtime Tests
 3. Continuous Integration Tests with *Travis CI*
 4. Cross Testing

Evaluation: Testing levels

1. Module tests: $\geq 90\%$ test coverage
2. Integration tests
 - ▶ All examples in the milestones (e.g., *example.prog*)
 - ▶ Additional test programmes
3. Runtime Tests:
 - ▶ First, execute the generated *LLVM IR* with *lli*
 - ▶ Then, test against exit code(return statement) and the expected output
4. Continuous Integration Tests with *Travis CI*:
Every commit pushed to the master branch is tested with *Travis CI*
5. Cross Testing:
Test the interchangeability of the Free University Compiler and the Javabite modules

Evaluation Results:

- Module Tests Results
- Integration Tests Results
- Runtime Tests Results

Integration Tests Results

An example of the integration test results.

TestSuite: 24 total, 24 passed		2.39 s
		Collapse Expand
TestSuite		2.39 s
M1RuntimeTest		688 ms
testAddProg	passed	387 ms
testSimpleAddProg	passed	109 ms
testMultipleMinusNotation	passed	78 ms
testDoubleDeclaration	passed	78 ms
testMultiplePlusInExp	passed	43 ms
testSimpleMulProg	passed	62 ms
testInvalidIds	passed	60 ms
testIndefReturn	passed	66 ms
testParenthesesProg	passed	85 ms
M2RuntimeTest		285 ms
testPrintProg	passed	112 ms
testAssignmentProg	passed	75 ms
testCondProg	passed	98 ms
M3RuntimeTest		435 ms
testMatrixMultiplicationProg	passed	244 ms
testFibProg	passed	99 ms
testNewtonProg	passed	92 ms
AdditionalRuntimeTest		681 ms
testArrayProg1	passed	65 ms
testArrayProg2	passed	68 ms
testArrayProg3	passed	71 ms
testEmptyProg	passed	69 ms
testRecordProg	passed	92 ms
testSimpleRecordProg	passed	67 ms
testReturnProg	passed	71 ms
testCalendarProg	passed	99 ms
testTestOfLoopsProg	passed	78 ms

The milestone 3 aims to evaluate the two features:

- Loops:
 - Our program can handle loops e.g., *while()*
- Arrays:
 - Our program supports array types e.g., *long[3][4]a*;

Outline

Motivation

Project Management

- Free University Compiler-Teams
- Development Environment

Design and Modules

- Lexical Analysis
- Parsing
- Semantic Analysis
- Intermediate Code Generation (TAC Generation)
- Backend - LLVM
- Backend - Exceptions
- Backend - Example program

Evaluation

- Evaluation: Testing levels
- Evaluation: Results
- Summary for milestone 3

GUI Demo

Lessons Learned

Now, our expert will show you how to use our program.

Outline

Motivation

Project Management

- Free University Compiler-Teams
- Development Environment

Design and Modules

- Lexical Analysis
- Parsing
- Semantic Analysis
- Intermediate Code Generation (TAC Generation)
- Backend - LLVM
- Backend - Exceptions
- Backend - Example program

Evaluation

- Evaluation: Testing levels
- Evaluation: Results
- Summary for milestone 3






GUI Demo

Lessons Learned

What have we learned from the project?

- Good design for the interfaces is mandatory to avoid changes
→ high effort
- Tests must take place in the earlier phase of the project
- Integration test or Cross testing must well organize and begin at the project start
- Respect your leader and his/her decisions, support it, and try to make it a success

References

-  Alfred V. Aho, Jeffrey Ullman, and Ravi Sethi.
Compiler: Prinzipien, Techniken und Werkzeuge.
Pearson Studium, 2. edition, 2008.
-  GitHub
<https://github.com/>.
-  Git Reference
<http://gitref.org/>.
-  Some Notes on Git
<http://java.dzone.com/articles/some-notes-git>.
-  Michael Lee Scott.
Programming language pragmatics.
Morgan Kaufmann Publishers, 3. edition, 2009.

Thank You

Q & A