Project Compilers 2020 (Mano Marichal & Joren Van Borm)

Werkt met Python 3.6+

Overview

Onderaan staat voor alle delen van de taak wat we wel of niet hebben gedaan.

- main.py: compileerd een enkele file & genereerd een dot en png file van de AST. Verdere uitleg onder **Installing and running - Compiling a file**
- run_benchmarks.py: genereert voor alle c files in **./test_IO/CompilersBenchmark** de llvm ir, mips, dotfiles en de ast als png
- run_test_files.py: genereert voor alle c files in **./test_IO/extra_tests** de llvm ir, mips, dotfiles en de ast als png
- clean.py: verwijdert alle gegenereerde files uit **./test_IO/**

De tests die we hebben gedaan hebben we vergeleken met gcc om te checken of ze klopten.

Installing and running:

(assuming a linux-based system)

Clone git repository

First, open a terminal where you'd like to clone the repository, then run:

```
git clone https://github.com/shano19/compilers-2020.git
```

Then navigate to the repository with `cd ./compilers-2020`.

Install LLVM

```
sudo apt-get install llvm
```

Install pip

```
sudo apt-get install python3-pip
```

Install virtualenv using pip3

```
sudo pip3 install virtualenv
```

Create virtual environment

`virtualenv venv`

Active virtual environment:

`source venv/bin/activate`

When you're done using the compiler, deactivate it again using the `deactivate` command. (or just quit the terminal)

Install prerequisites:

`pip3 install -r requirements.txt`

Run the test files

`python3 run.py`

Some of these test files will print to stderr when warnings (or errors) are encountered. Some of them won't compile at all because they're testing error detection.

Compiling a file

`python3 ./src/main.py <filename>`

The `-cf` flag can be added after `<filename>` to enable constant folding. The `-n` flag can similarly be added to suppress warnings.

PART 1 (C -> LLVM):

Project 1)

- 1 Expression Parser
  - 1.1 Grammar:
  - [x] (mandatory) Binary operations + , - , * , and /
  - [x] (mandatory) Binary operations > , < , and ==
  - [x] (mandatory) Unary operators + and -
  - [x] (mandatory) Brackets to overwrite the order of operations
  - [x] (optional) Binary operator %
  - [x] (optional) Comparison operators >= , <= , and !=
  - [x] (optional) Logical operators && , || , and !
  - [x] 1.2 (mandatory) AST
  - [x] 1.3 (mandatory) Visualization
  - [x] 1.4 (optional) Constant folding

Notes: - We supporten het declaren van meerder variables in hetzelfde statement niet, eg. `int a, b, c;`

Project 2)

- 2.1 Variables:
  - 2.1.1 Grammar:
    - (mandatory) Types
      - [x] char
      - [x] int
      - [x] float
      - [x] pointer (no pointer arithmetic)
    - (mandatory) Reserved words
      - [x] const
      - [x] int
      - [x] float
      - [x] char
    - [x] (mandatory) Variables
    - [x] (mandatory) Pointer Operations * and &
    - [x] (optional) Identifier Operations ++ and –
    - [x] (optional) Implicit Conversions (+ warnings for non-promotions)
- [x] 2.1.2 (mandatory) AST
- [x] 2.1.3 (mandatory) Visualization
- [ ] 2.1.4 (optional) Constant Propagation
- 2 Error Analysis
  - [x] 2.2.1 Syntax Errors
  - [x] 2.2.2 Semantic Errors
    - [x] undefined & uninitialised variables
    - [x] redeclared & redefined variables
    - [x] operations on incompatible types (dereferencing a non-ptr type)
    - [x] Assignment to an rvalue
    - [x] Assignment to a const variable
    - [x] Symbol table (scoped)

Project 3)

- 1 Variables
  - 1.1 Grammar
    - [x] (mandatory) Comments
    - [x] (mandatory) printf() for char, int & float (without metastring)
  - [x] 1.2 (mandatory) AST
  - [x] 1.3 (mandatory) Visualization

  - 2 (mandatory) LLVM
  - [x] (mandatory) Binary operations + , - , * , and /
  - [x] (mandatory) Binary operations > , < , and ==
  - [x] (mandatory) Unary operators + and -
  - [x] (mandatory) Printf
  - [x] (mandatory) Pointers + pointer operators
  - [x] (optional) Identifier Operations ++ and –
  - [x] (optional) Comments for each machine instruction
  - [x] (optional) Comparison operators >= , <= , and !=
  - [x] (optional) Logical operators && , || , and !
  - [x] (optional) Conversions (bool <> char <> int <> float)
  - [x] (optional) Binary operator %
  - [ ] (optional) Include comments in compiled LLVM

Project 4)

- 4.1 Grammar
  - [x] (mandatory) if
  - [x] (mandatory) else
  - [x] (mandatory) while
  - [x] (mandatory) for
  - [x] (mandatory) break
  - [x] (mandatory) continue
  - [ ] (optional) switch
  - [ ] (optional) case
  - [ ] (optional) default
  - [x] (mandatory) scopes
- [x] 4.2 (mandatory) AST
- [x] 4.3 (mandatory) Visualization
- [x] 4.4 (mandatory) Semantic Analysis
- 4.5 (mandatory) LLVM
  - [x] (mandatory) if
  - [x] (mandatory) else
  - [x] (mandatory) while
  - [x] (mandatory) for
  - [x] (mandatory) break
  - [x] (mandatory) continue
  - [ ] (optional) switch
  - [ ] (optional) case
  - [ ] (optional) default
  - [x] (mandatory) scopes ### Project 5)
- 5.1 Grammar
  - [x] (mandatory) return
  - [x] (mandatory) void
  - [x] (mandatory) scopes

    - [x] (mandatory) local variables
    - [x] (mandatory) global variables
    - [x] (mandatory) functions
- [x] 5.2 (mandatory) AST
- [x] 5.3 (mandatory) Visualization
- 5.4 (mandatory) Semantic Analysis
    - [x] (optional) check if all paths end with return
- 5.5 (mandatory) Optimizations
    - [ ] (mandatory) unreachable & dead code after return
    - [ ] (mandatory) unreachable & dead code after break/continue
    - [ ] (optional) non used variables
    - [ ] (optional) conditionals that are never true
- 5.6 (mandatory) LLVM
    - [x] (mandatory) return
    - [x] (mandatory) void
    - [x] (mandatory) scopes
    - [x] (mandatory) local variables
    - [x] (mandatory) global variables
    - [x] (mandatory) functions ### Project 6)
- 6.1 Grammar
    - [x] (mandatory) arrays
    - [x] (mandatory) import
    - [ ] (optional) arrays with variable size
    - [ ] (optional) multi-dimensional arrays
- [x] 6.2 (mandatory) AST
- [x] 6.3 (mandatory) Visualization
- [x] 6.4 (mandatory) LLVM

Notes: - We hebben arrays die kunnen gebruikt worden, maar ze kunnen wel niet aangesproken worden via een pointer. - We hebben support voor strings in de AST & Visualization, maar niet in LLVM. (wel char arrays) - Onze printf en scanf supporten het printen en lezen van char arrays niet.

Remarks + extras

- We supporten nested pointers & arrays (toch tot op zekere hoogte)
- Er is een warning indien een non-void functie mogelijk niet returned, en juist geen warning indien een void functie zeker niet returned
- Er is geen support voor compound assignment (+=, *= etc)
- Er is geen support voor multi-declaraions (int a, b=3, c;)
- Er zijn enkele problemen met o.a. scoping, arrays & semantic errors die verde zullen toegelicht worden in de video.
- De enige niet uitgevoerde verplichte opdracht is de optimisation van onberijkbare code

PART 2 (C -> MIPS)

Overview of the features

- [x] = implemented in MIPS
- [ ] = not implemented in MIPS

Mandatory features:

- [x] binary operations + , - , * , and /
- [x] binary operations > , < , and ==
- [x] unary operators + and -
- [x] char
- [x] int
- [x] float
- [x] pointers
- [x] pointer operations * and &
- [x] if
- [x] else
- [x] while
- [x] for
- [x] break
- [x] continue
- [x] global variables
- [x] functions
- [x] printf
- [x] scanf
- [x] arrays

Optional features:

- [x] binary operator %
- [x] comparison operators >= , <= , and !=
- [x] logical operators && , || , and !
- [x] identifier Operations ++ and –
- [x] implicit Conversions (+ warnings for non-promotions)
- [ ] include comments in compiled LLVM
- [ ] switch
- [ ] case
- [ ] default
- [ ] arrays with variable size
- [ ] multi-dimensional arrays

Notes:

- Arrays kunnen niet via een pointer worden aangesproken
- We supporten het inlezen en uitlezen van char arrays als strings niet
- We supporten geen functies met meer dan 4 argumenten
- Er is geen support voor multi-declaraions (int a, b=3, c;)
- Er is geen support voor compound assignment (+=, *= etc)