Func Compiler - Group project middle-term report

Group: Zhezhen Cao Fan Club

Leader: Kuang Liang 12111012

Language Design

- *Defining functions*
  - Define a function named with a LID (`[a-z][a-zA-Z]`) with an expression.

    ```
    defn f x = {x + 1}
    ```

- *Declaring data types*
  - Define a data type named with a UID (`[A-Z][a-zA-Z]`) with multiple constructors.
  - New feature: Define a data type with a placeholder.

    ```
    data Shape = {Circle Int, Rectangle Int Int}
    data Pair a = {MkPair a a}  # new
    ```

- *Applying fuctions*

  ```
  f 10
  ```

- *Basic data types*
  - Int, Float, Bool (`True` , `False`), Char (`'a'` , `'\n'` , ...).
  - List: `[0, 1, 2,]` (Remeber to end with a comma+csquare: `,]`)
  - String (List of Char): `"012"` -> `['0', '1', '2',]`
- *Arithmetic*
  - All C-featured operations (five-rule operation, boolean operations, bitwise operations, comparing operations and indexing)
  - Connect two lists with `++`
- *Pattern matching* (case of)

  ```
  defn area shape = {
      case shape of {
          Circle r -> {r * r * 3}
          Rectangle w h -> {w * h}
      }
  }
  ```

- *Do block*
  - To specify a process.

    ```
    defn aPlusB = do {
      defn a <- {readInt}
      defn b <- {readInt}
      defn c <- return {a + b}
      {print c}
    ```

```
  return {c}
}
```

Language Feature

- Language design:
  - Advantages: Concise, elegant and safe for logic-based process.
  - Disadvantages: Able but not suitable for building/manipulating data structures.
- Lazy evaluation:
  - Advantages: Comptutation efficiency & improved abstraction
  - Disadvantages: Memory overhead & unpredictability

Progress & Plan

Tools: Flex, Bison, LLVM (in progress).

We have finished lexical checking, parsing and type checking. Later, we plan to compile the code and run it on LLVM.

Testcases

All testcases `.func` files are under `\res\test-case`. Here is the explanation:

- `\basic`: showing the basic ability of lexer and parser.
  - `basic.func`: all basic types and operations of our language.
  - `data.func`: correctly defined data types of out language.
  - `syc-err.func`: all-in-one lexer/parser error trigger, showing how our lexer and parser detect and recover from errors.
- `\typ-chk`: type-checking testcases.
  - `\err`: all kinds of errors, like:
    - `1.func`: wrong parameter type.
    - `2.func`: wrong operand type.
    - `3.func` and `4.func`: using non-function type as a function.
    - `6.func`: of-type is not the same with pattern-type in pattern matching (case-of).
    - `7.func`: case-types are not the same.
    - `8.func` and `9.func`: mixing IO type and non-IO type. (Hint: the correct one is `type-checking-5.func`)
    - `10.func`: unknown LID.
    - `5.func`: our compiler ends with a Segmentation fault in this testcase.
  - `type-checking-[1-8].func`: correct cases (the 1st and 8th are recommended for understanding the advantage of our language).

Score: 20/21

How to Run

1. Skip this if you do not want. I have included `func-highlighter-0.0.1.vsix` in our folder. This is our VsCode extension which can highlight a `.func` file. It can help if you want to self-define some testcase to test our code. Codes are on GitHub.
2. Skip this if you do not want. Run `.\build.sh` to generate our compiler. The version I uploaded on BlackBoard is compiled.
3. Run `clear && .\test.sh` to test all testcases in the folder `\res\test-case`. You can add more examples to `\res\test-case\z` (so that they will appear in the bottom).

Contribution Ratio

1:1:1:1