Language Documentation:

C.O.O.L. - C Object Oriented Language

Student Names: Tommy Avetisyan, Kyle Felkel, Matthew Estes

Why this language, and why this language design:

Our team was well knowledged in Java and felt it was appropriate to start with a language we knew to not hinder our progress learning a new language. We targeted C as it is fairly low level and to see if we can translate the ideas of classes and polymorphism. Overall the language was designed to be pretty similar to Java/C with changes meant to reduce complexity of the compiler code.

Code snippets in your language highlighting features and edge cases, along with relevant explanations:

```
class TestClassA{
    int test = 50;
    constructor(){}
    void changeMax(int max){
        this.test=max;
        return;
    }
    void testMethodOddEven(){
        for(Int i = 0; i < this.test; i = i + 1;){
            if(i%2==0){
                print(i);
                println(" is an even number");
            }else{
                print(i);
                println(" is an odd number");
            }
        }
        return;
    }
```

```
}
class TestClassB : TestClassA{
    constructor(){}
}
Int main(){
    TestClassA TestClassA_Object=new TestClassA();
    TestClassA TestClassB_Object=new TestClassB();

    TestClassA_Object.testMethodOddEven();

    TestClassB_Object.changeMax(100);
    TestClassB_Object.testMethodOddEven();

    return 0;
}
```

Above is a code snippet of C.O.O.L where we show inheritance and subtyping.

ClassA sets variable “test” to 50 by default and will print out odd and even numbers from 0 to 50 in a for loop. (our language also supports while loops).

ClassB is created as a subtype of ClassA and inherits all of ClassA’s methods and variables. We now change the value of “test” to 100 which then prints out all odd and even numbers from 0 to 100.

Known limitations:

- No decimal/double values.
- No static or private modifiers for class contents.
- Each method can only have a single return statement, and it must be the last statement in the method.
- No arrays or build in list type.
- Given the lack of pointers building a dynamically sized list would be difficult.
- Compiler makes no attempt to optimize code performance.
- Compiler itself is not very optimized, long chains of if-else statements means compiling a large project could take a significant amount of time.
- Using ‘this’ is required when referring to variables declared at the top of the object and any internal method calls

Knowing what you know now, what would you do differently?

I think knowing in detail what each part of the project looked like would have helped to create a stronger foundation that would have caused less trouble in the future. A problem we kept facing was going back to the previous step of the project and redoing parts of the code that were causing trouble.

Some examples were;

- When we decided to only have a single return statement at the end of our methods and we had to go review our parser to compensate.
- Parsing through an expression and differentiate how operators, method calls, parentheses were handled
- Separating “main” into its own token would have been handled better in the typechecker.

Our development environment was fairly stable except for when that class packages and java versions would error out on updates.

Translating to our target language raised some discussion on how we handle the new features we wanted to implement to C, we went through a couple of iterations of how we did it.

Some of the features we added to our language sounded great when we were talking about it at the beginning but ended up being difficult to implement

In general more time spent doing design would have been very helpful. Starting with code snippets to build a grammar then showing how each component would handle the code snippets would have provided a lot of early clarity. Having more of a high-level plan would have allowed for more consistency in how each component implements features. This would have also made testing easier as tests could have been built around the initial code snippets, this way we could quickly ensure if a component fulfils its requirements and ensure the components will correctly interact with each other for the overall compiler.

How to compile compiler:

Project is written in Java and can be compiled using any Java compiler. A pom.xml file is included to compile the project using Maven through ‘mvn compile’.

How to run compiler: Project has a main method located in Compiler.java. Compile and run this file for prompts to enter source code file.

Formal syntax definition:

- `int` is an integer
- `string` is a string
- `Var` is a variable

- obj is a variable that is an instance of an object
- `classname` is the name of a class

type	`int | String | Bool | classname`
op	`+ | - | * | / | % | < | <= | > | >= | ==` *Arithmetic operations*
exp	`var | string | int` | *Variables, string, and integers are expressions* `exp op exp` | *Arithmetic operations* `(exp)` | *Parentheses* `obj.field | Used to access internal variables` `obj.methodname(exp*)` | *Calls a method* `this.field` | *this refers to current instance* `this.methodname(exp*)` | *this refers to current instance* `new classname(exp*)` | *Creates a new instance of a class*
vardec	`type var | type var = exp` *Variable declaration; value of var is undefined until assigned*
stmt	`vardec;` | *Variable declaration* `var = exp;` | *Assignment* `print(exp);` | *Print to the terminal* `println(exp);` | *Print to the terminal with new line* `for(stmt; exp; stmt;) {stmt*}`| *for loops* `while (exp) {stmt*}` | *while loops* `if (exp) {stmt*} else {stmt*}` | *if-else* `if (exp) {stmt*}` | *if (but no else)*
methoddef	`type methodname(vardec*) {stmt* return exp;}` | `Void methodname(vardec*) {stmt* return;}`

classdef	`class classname {`
	`  vardec*`
	`  constructor(vardec*) {stmt*}` *vardecs are comma-sep*
	`  methoddef*`
	`} |`
	`class classname: classname {` *With inheritance*
	`  vardec*`
	`  constructor(vardec*) {stmt*}` *vardecs are comma-sep*
	`  methoddef*`
	`}`

program	`classdef* methoddef` | *main methoddef is entry point*