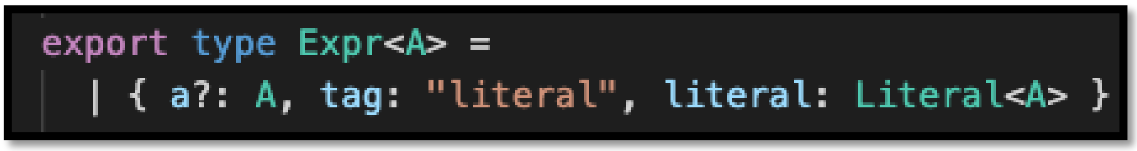
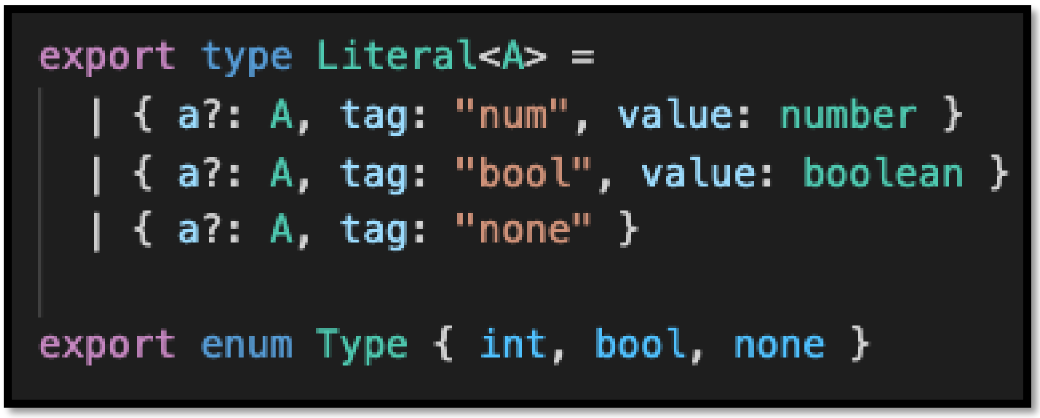
Notice:

For this project, I have finished most functions of the parser, the compiler, and the runner. However, the function definition part and type checker are still in progress, so I wouldn't be able to match some of the requirements of this assignment, but I will also show some features that are not included in the bullet points. In addition, my program supports "npm run build-web" and "npm test" but not "npm run build-cli". Since there are some config errors, and the professor told us not to spend time on those errors, I decided to use those two commands and my debugstart.ts to debug and test. I believe these methods are enough for you to test.

1. A description of the representation of values (integers, booleans, and None) in your implementation. Show how the representation is used to print True/False for Boolean results rather than numeric output.

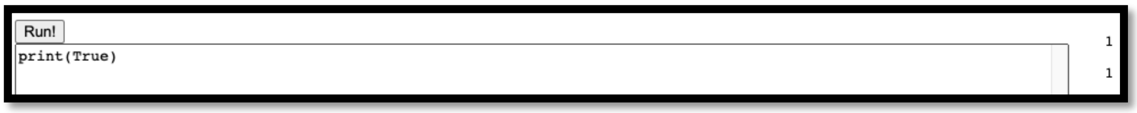


```
export type Expr<A> =
    | { a?: A, tag: "literal", literal: Literal<A> }
```



```
export type Literal<A> =
    | { a?: A, tag: "num", value: number }
    | { a?: A, tag: "bool", value: boolean }
    | { a?: A, tag: "none" }

export enum Type { int, bool, none }
```

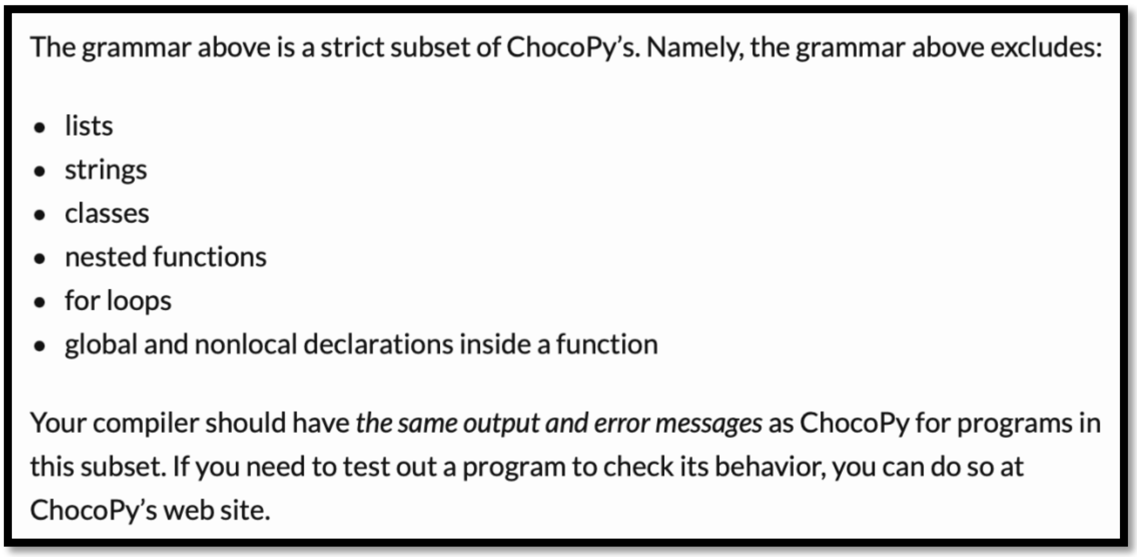
I defined Literal as a number, a Boolean, or none with tags and values, and a literal is also a kind of an expression. Since I haven't finished the function part, I still use 1 to represent True and 0 to represent False when the print function is called. I will define a print_bool function for Boolean values after the function part is done, and True and False will be displayed.



```
print(True)
```

```
1
1
```

2. Give an example of a program that uses
   (a) At least one global variable



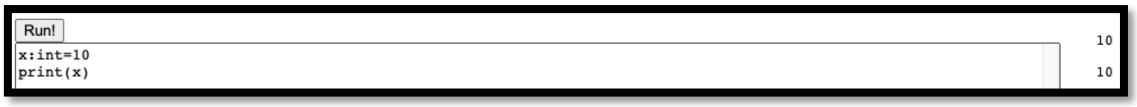
The grammar above is a strict subset of ChocoPy's. Namely, the grammar above excludes:

- lists
- strings
- classes
- nested functions
- for loops
- global and nonlocal declarations inside a function

Your compiler should have *the same output and error messages* as ChocoPy for programs in this subset. If you need to test out a program to check its behavior, you can do so at ChocoPy's web site.

Since the SPEC indicates that global declaration is not included in the subset we should implement, I exclude this feature. I will implement it if the conflict is resolved, and we are sure that this feature is needed.

(b) At least one function with a parameter

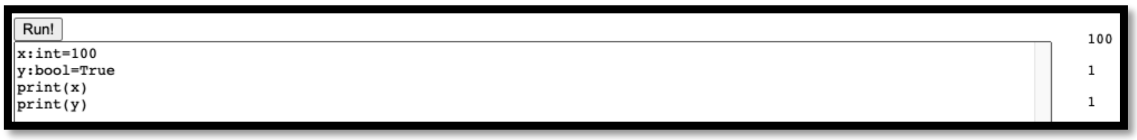


```
x:int=10
print(x)
```

```
10
10
```

The functions from the last project are all supported in this project.

(c) At least one variable defined inside a function

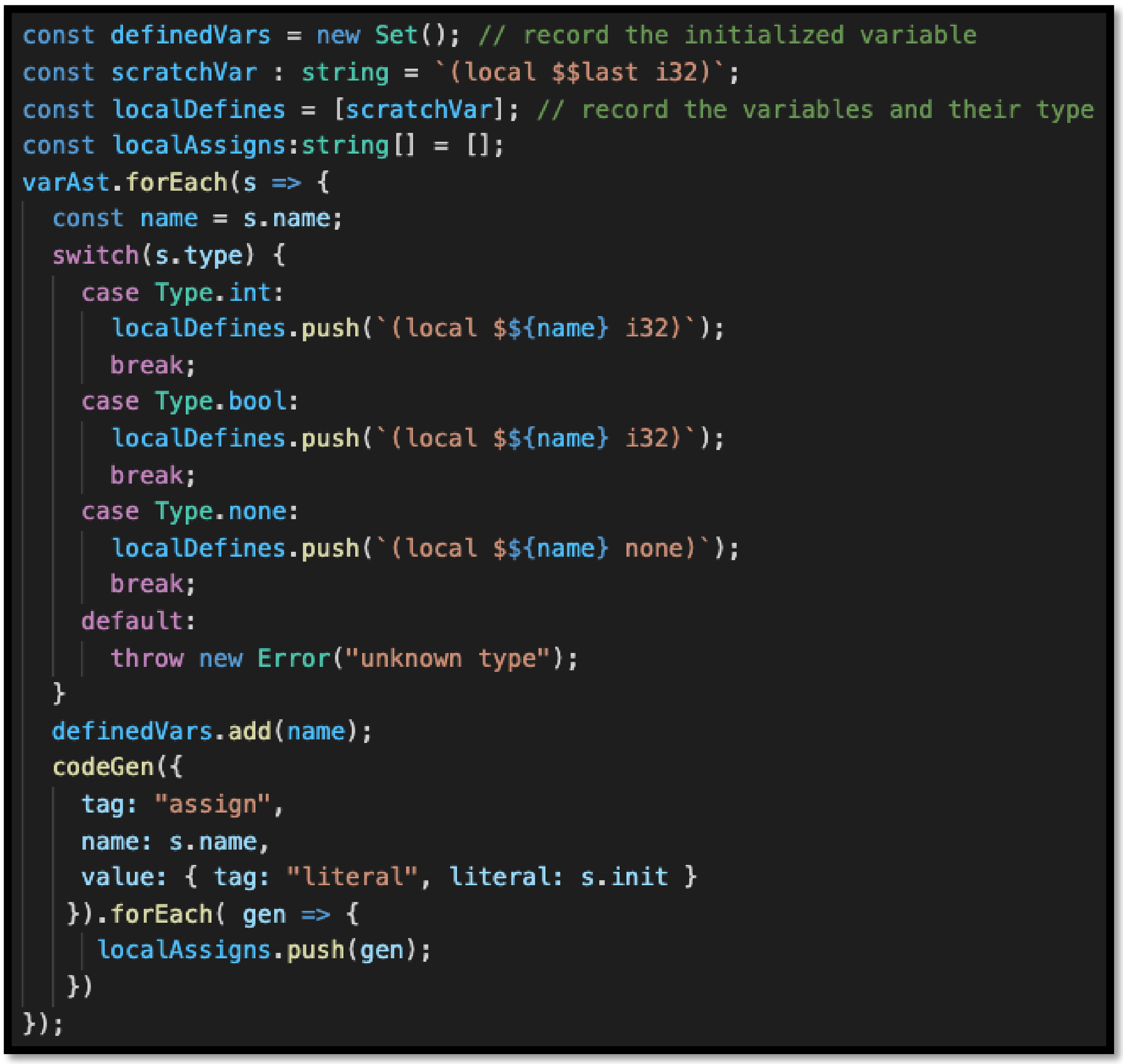


```
x:int=100
y:bool=True
print(x)
print(y)
```

```
100
1
1
```

Since I haven't done the function part, I would just show you that I can define both integers and Booleans instead.

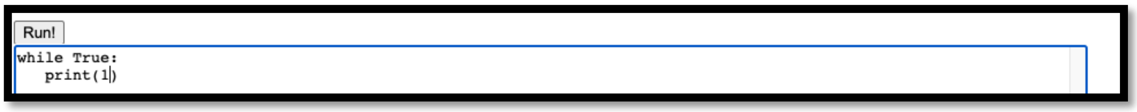
(d) By linking to specific definitions and code in your implementation, describe where and how those three variables are stored and represented throughout compilation.



```
const definedVars = new Set(); // record the initialized variable
const scratchVar : string = `(local $$last i32)`;
const localDefines = [scratchVar]; // record the variables and their type
const localAssigns:string[] = [];
varAst.forEach(s => {
  const name = s.name;
  switch(s.type) {
    case Type.int:
      localDefines.push(`(local $${name} i32)`);
      break;
    case Type.bool:
      localDefines.push(`(local $${name} i32)`);
      break;
    case Type.none:
      localDefines.push(`(local $${name} none)`);
      break;
    default:
      throw new Error("unknown type");
  }
  definedVars.add(name);
  codeGen({
    tag: "assign",
    name: s.name,
    value: { tag: "literal", literal: s.init }
   }).forEach( gen => {
     localAssigns.push(gen);
   })
});
```

Since I haven't defined global variables and functions, all my variables are stored locally. I will probably use a similar structure to store my global variables. However, for function variables, I think I will need a stack to store those structs so that we can get correct variables and values under different environments.

3. Write a ChocoPy program that goes into an infinite loop. What happens when you run it on the web page using your compiler?

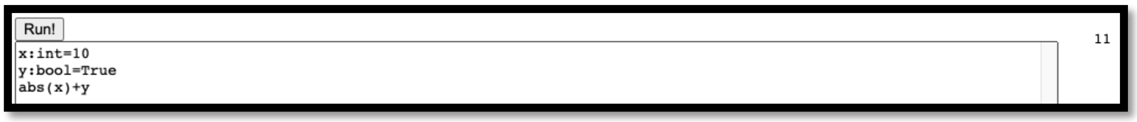


```
while True:
    print(1)
```

Since the program goes into an infinite loop, the web page would keep trying to collect all values that should be displayed. However, there are infinite values, and the web page would eventually shutdown. Ideally, we should be able to identify the infinite loop in the code and warn the user before the code is ran.

4. For each of the following scenarios, show a screenshot of your compiler running the scenario in the browser:

(a) A program that reports a type error for adding a number and a boolean where one operand is a call expression and the other operand is a variable (for example, f(1) + x)

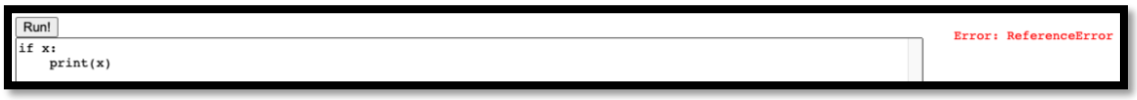


```
x:int=10
y:bool=True
abs(x)+y
```

```
11
```

Since I haven't done a type checker, the type error would not be reported, and the value would be 11 because True is considered as 1 in my program. This is a temporary feature, and it can be modified easily once the type checker is done.

(b) A program that has a type error in a conditional position (the condition part of an if or while), where that position is a non-global identifier (function parameter or local variable)

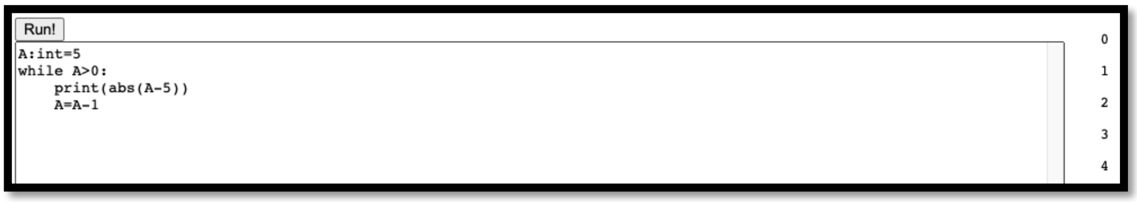


```
if x:
    print(x)
```

```
Error: ReferenceError
```

Since I haven't finished my type checker, there would be no type error, but I can show that a reference error would pop out if a variable that is not in scope is used as a condition.

(c) A program that with a loop that has multiple iterations, and calls a function on each iteration (it can be the same function)

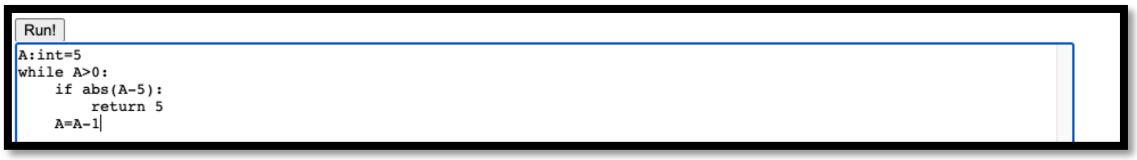


```
A:int=5
while A>0:
    print(abs(A-5))
    A=A-1
```

```
0
1
2
3
4
```

There are 5 iterations in this program, and the print function and the abs function are called on each iteration.

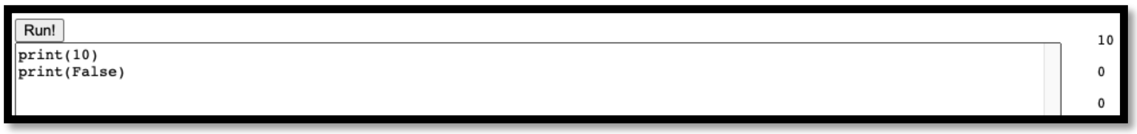
(d) A program that returns from the body of a loop, and not on the first iteration of the loop



```
A:int=5
while A>0:
    if abs(A-5):
        return 5
    A=A-1
```

This program would return on the second iteration, and I can use a variable to get the return value once the function part is done.

(e) Printing an integer and a Boolean



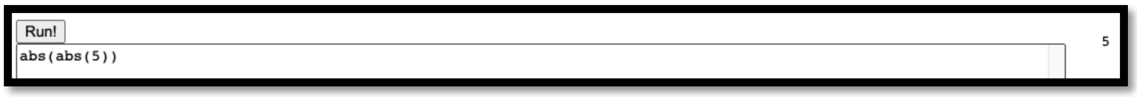
Run!

```
print(10)
print(False)
```

```
10
0
0
```

After the print_bool function is done, the output would be more ideal, but the output 10 and 0 are still reasonable now.

(f) A recursive function that terminates (e.g. no stack overflow)



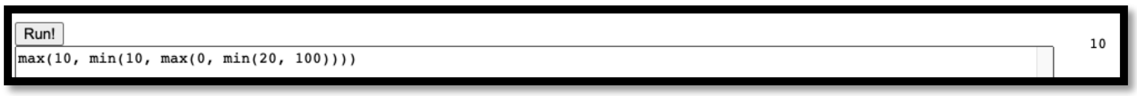
Run!

```
abs(abs(5))
```

```
5
```

The function part is not done, but I can show that some of my functions can be called recursively.

(g) Two mutually-recursive functions that terminate (e.g. no stack overflow)



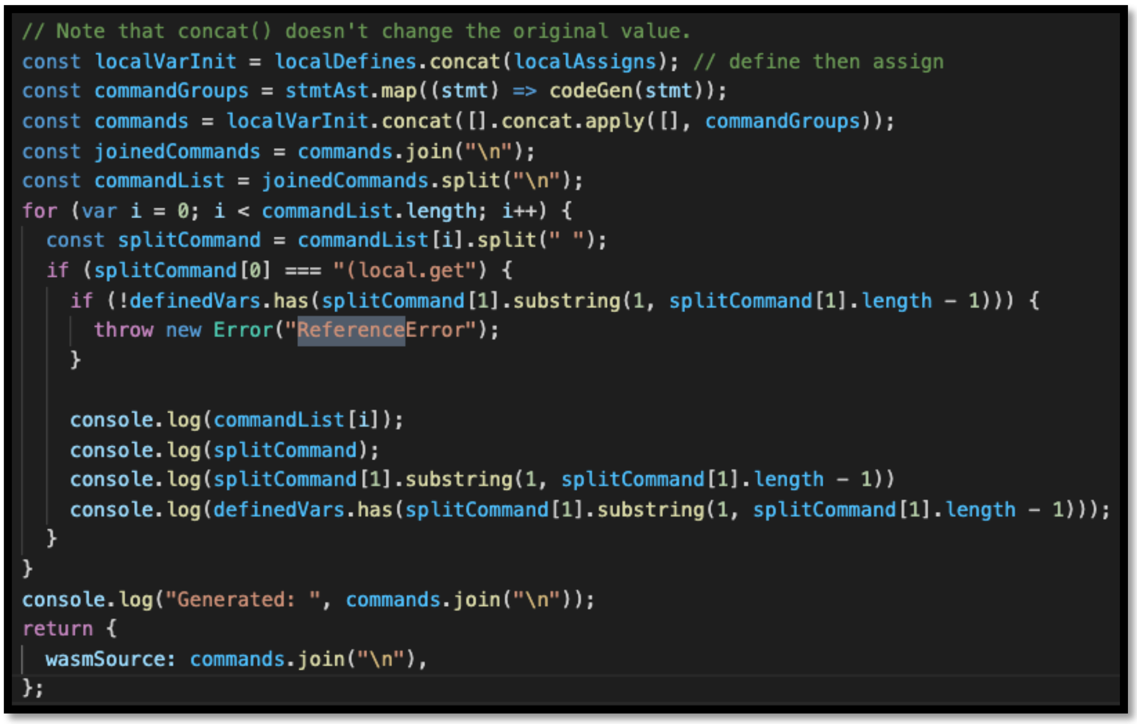
Run!

```
max(10, min(10, max(0, min(20, 100))))
```

```
10
```

The function part is not finished, but I can show that some of my functions can be called in a mutually-recursive way.

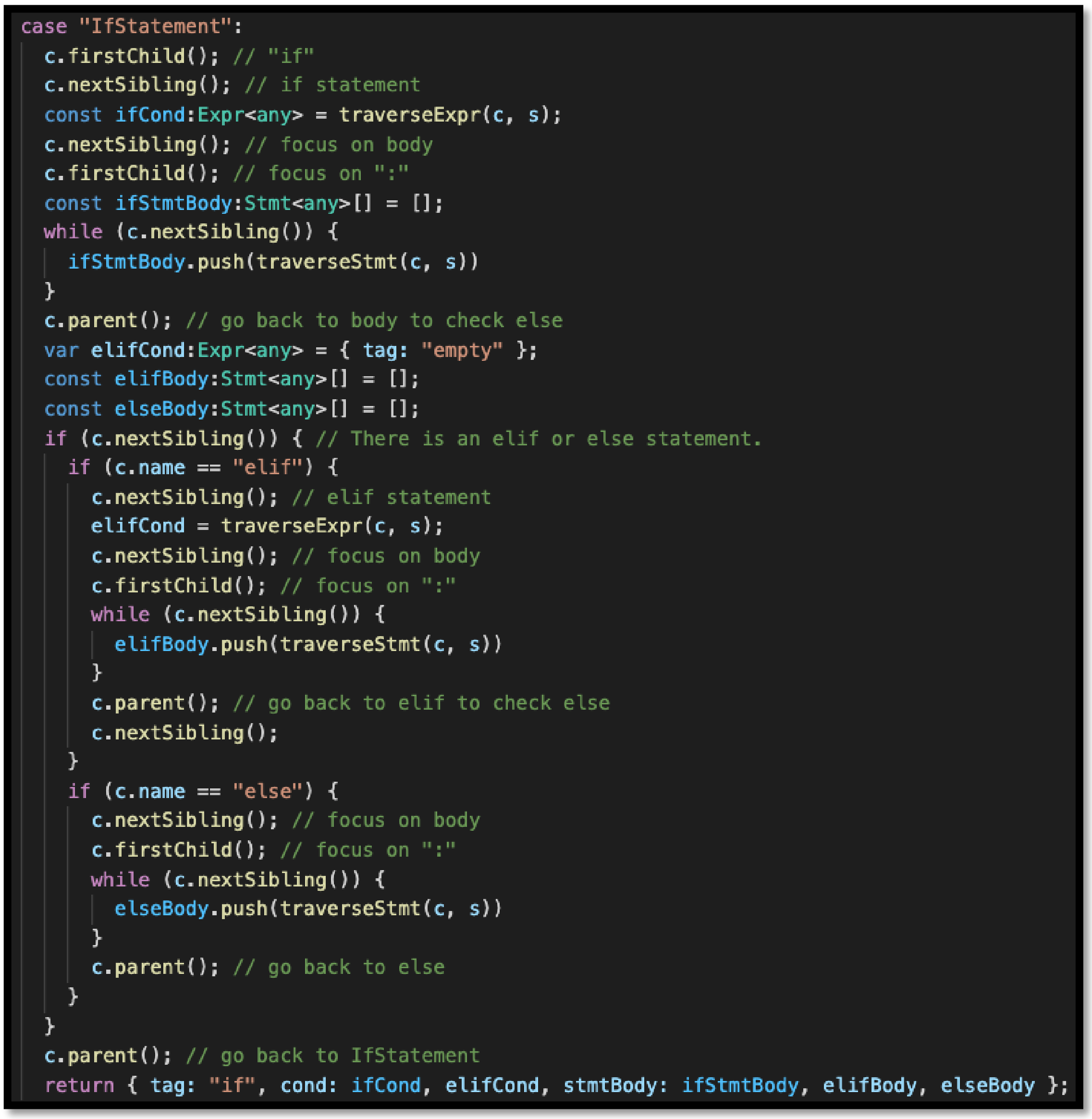
5. Choose one of example (1) or (2) above, show a few lines of code around the line that reports the error and describe the relevant parts of the type-checking environment at that point.



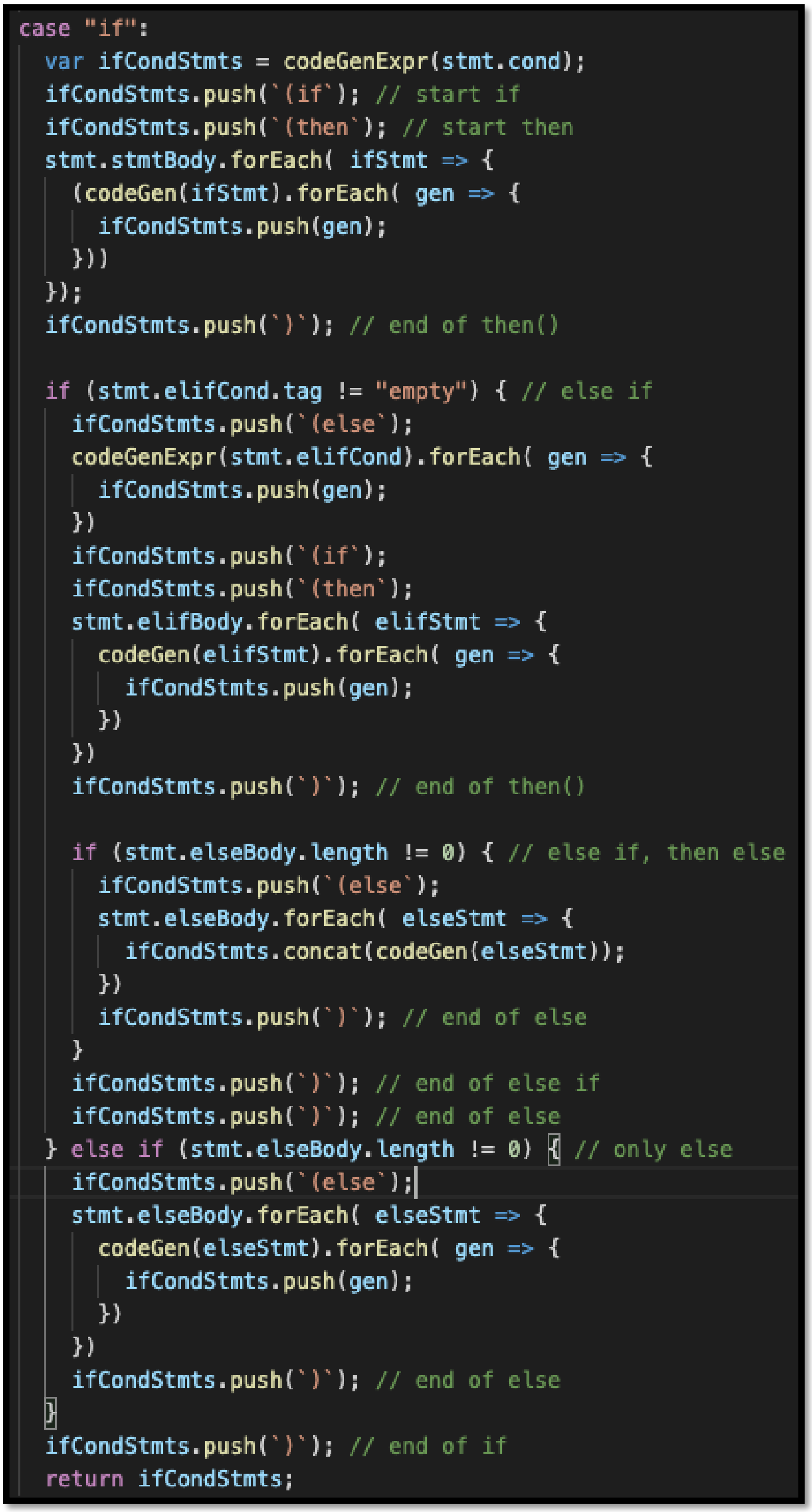
```
// Note that concat() doesn't change the original value.
const localVarInit = localDefines.concat(localAssigns); // define then assign
const commandGroups = stmtAst.map((stmt) => codeGen(stmt));
const commands = localVarInit.concat([].concat.apply([], commandGroups));
const joinedCommands = commands.join("\n");
const commandList = joinedCommands.split("\n");
for (var i = 0; i < commandList.length; i++) {
  const splitCommand = commandList[i].split(" ");
  if (splitCommand[0] === "(local.get") {
    if (!definedVars.has(splitCommand[1].substring(1, splitCommand[1].length - 1))) {
      throw new Error("ReferenceError");
    }

    console.log(commandList[i]);
    console.log(splitCommand);
    console.log(splitCommand[1].substring(1, splitCommand[1].length - 1))
    console.log(definedVars.has(splitCommand[1].substring(1, splitCommand[1].length - 1)));
  }
}
console.log("Generated: ", commands.join("\n"));
return {
  wasmSource: commands.join("\n"),
};
```

Since my program pops a reference error at example (2), I would like to show how my complier catches the error. Basically, it checks the definedVars struct which I used to store defined variables to know if a variable can be referenced.

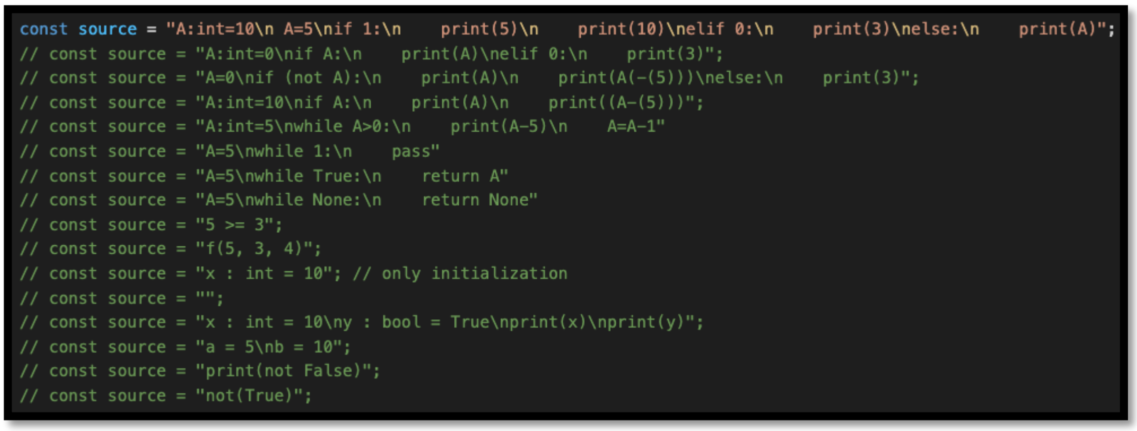
Explanation:



```
case "IfStatement":
  c.firstChild(); // "if"
  c.nextSibling(); // if statement
  const ifCond:Expr<any> = traverseExpr(c, s);
  c.nextSibling(); // focus on body
  c.firstChild(); // focus on ":"
  const ifStmtBody:Stmt<any>[] = [];
  while (c.nextSibling()) {
    ifStmtBody.push(traverseStmt(c, s))
  }
  c.parent(); // go back to body to check else
  var elifCond:Expr<any> = { tag: "empty" };
  const elifBody:Stmt<any>[] = [];
  const elseBody:Stmt<any>[] = [];
  if (c.nextSibling()) { // There is an elif or else statement.
    if (c.name == "elif") {
      c.nextSibling(); // elif statement
      elifCond = traverseExpr(c, s);
      c.nextSibling(); // focus on body
      c.firstChild(); // focus on ":"
      while (c.nextSibling()) {
        elifBody.push(traverseStmt(c, s))
      }
      c.parent(); // go back to elif to check else
      c.nextSibling();
    }
    if (c.name == "else") {
      c.nextSibling(); // focus on body
      c.firstChild(); // focus on ":"
      while (c.nextSibling()) {
        elseBody.push(traverseStmt(c, s))
      }
      c.parent(); // go back to else
    }
  }
  c.parent(); // go back to IfStatement
  return { tag: "if", cond: ifCond, elifCond, stmtBody: ifStmtBody, elifBody, elseBody };
```



```
case "if":
  var ifCondStmts = codeGenExpr(stmt.cond);
  ifCondStmts.push(`(if`); // start if
  ifCondStmts.push(`(then`); // start then
  stmt.stmtBody.forEach( ifStmt => {
    (codeGen(ifStmt).forEach( gen => {
      ifCondStmts.push(gen);
    }))
  });
  ifCondStmts.push(`)`); // end of then()

  if (stmt.elifCond.tag != "empty") { // else if
    ifCondStmts.push(`(else`);
    codeGenExpr(stmt.elifCond).forEach( gen => {
      ifCondStmts.push(gen);
    })
    ifCondStmts.push(`(if`);
    ifCondStmts.push(`(then`);
    stmt.elifBody.forEach( elifStmt => {
      codeGen(elifStmt).forEach( gen => {
        ifCondStmts.push(gen);
      })
    })
    ifCondStmts.push(`)`); // end of then()

    if (stmt.elseBody.length != 0) { // else if, then else
      ifCondStmts.push(`(else`);
      stmt.elseBody.forEach( elseStmt => {
        ifCondStmts.concat(codeGen(elseStmt));
      })
      ifCondStmts.push(`)`); // end of else
    }
    ifCondStmts.push(`)`); // end of else if
    ifCondStmts.push(`)`); // end of else
  } else if (stmt.elseBody.length != 0) { // only else
    ifCondStmts.push(`(else`);
    stmt.elseBody.forEach( elseStmt => {
      codeGen(elseStmt).forEach( gen => {
        ifCondStmts.push(gen);
      })
    })
    ifCondStmts.push(`)`); // end of else
  }
  ifCondStmts.push(`)`); // end of if
  return ifCondStmts;
```



```
const source = "A:int=10\n A=5\nif 1:\n    print(5)\n    print(10)\nelif 0:\n    print(3)\nelse:\n    print(A)";
// const source = "A:int=0\nif A:\n    print(A)\nelif 0:\n    print(3)";
// const source = "A=0\nif (not A):\n    print(A)\n    print(A(-(5)))\nelse:\n    print(3)";
// const source = "A:int=10\nif A:\n    print(A)\n    print((A-(5)))";
// const source = "A:int=5\nwhile A>0:\n    print(A-5)\n    A=A-1"
// const source = "A=5\nwhile 1:\n    pass"
// const source = "A=5\nwhile True:\n    return A"
// const source = "A=5\nwhile None:\n    return None"
// const source = "5 >= 3";
// const source = "f(5, 3, 4)";
// const source = "x : int = 10"; // only initialization
// const source = "";
// const source = "x : int = 10\ny : bool = True\nprint(x)\nprint(y)";
// const source = "a = 5\nb = 10";
// const source = "print(not False)";
// const source = "not(True)";
```

I wouldn’t say that the required features are really hard to implement, but my initial strategy was to finish the parser, then the compiler, and then the type checker. Thus, I spent a lot of time checking the logic of my program and thinking of edge cases before I move on to the next steps. It might seem that there are a few requirements that I cannot meet, but I think that I am actually pretty close. It’s mainly related to my strategy, but I also don’t think this is wrong because what I focused is also really important for a practical compiler. For example, I spent a lot of time on the logic and test cases of the if statements, and this kinds of details stops me from finishing the program completely.