ReadMe

1. `None`,`True`, `False`

I use number `-1`, `1`, `0` to represent `None`, `True` and `False` seperately in WASM.

And in order to print the boolean result, I will check the parameter type in call function, if the type is `bool`, I will call `print_bool` function. If the parameter result is `True`, then `True` will be printed, otherwise `False` will be printed which is shown below.

```
case "call":
  const valStmts = expr.args.map(e => codeGenExpr(e, locals)).flat();
  let toCall = expr.name;
  if(expr.name === "print") {
    switch(expr.args[0].a) {
      case "bool": toCall = "print_bool"; break;
      case "int": toCall = "print_num"; break;
      case "none": toCall = "print_none"; break;
    }
  }
  valStmts.push(`(call ${toCall})`);
  return valStmts;
```

```
var importObject = {
  imports: {
    print_num: (arg : any) => {
      console.log("Logging from WASM: ", arg);
      display(String(arg));
      return arg;
    },
    print_bool: (arg : any) => {
      if(arg === 0) { display("False"); }
      else { display("True"); }
      return arg;
    },
    print_none: (arg: any) => {
      display("None");
      return arg;
    },
```

2. exmaple

I will use a simple example here

```
x : int = 1
def f(x: int)->int:
    y: int = 3
    return x + y

f(x)
```

In this example, I define one global variable, one variable in the function and the function has one parameter. In the type check program, it will generate global variable map first, just like the picture shown below

```
export function tcProgram(p : Stmt<any>[]) : Stmt<Type>[] {
  const functions = new Map<string, [Type[], Type]>();
  p.forEach(s => {
    if(s.tag === "define") {
      functions.set(s.name, [s.params.map(p => p.typ), s.ret]);
    }
  });

  const globals = new Map<string, Type>();
  return p.map(s => {
    if(s.tag === "def") {
      const rhs = tcExpr(s.value, functions, globals);
      if(globals.has(s.name) && globals.get(s.name) !== rhs.a) {
        throw new Error(`Cannot assign ${rhs} to ${globals.get(s.name)}`);
      }
      if (rhs.a != s.a){
        throw new Error(`TypeError: The type ${s.a} of ${s.name} doesn't match with
      }
      globals.set(s.name, rhs.a);
      return { ...s, value: rhs };
    }
    else {
      const res = tcStmt(s, functions, globals, "none");
      return res;
    }
  });
}
```

When it comes across a function definition, it will add to functions map which will records its parameters types and return type. And in the "call" statement, the parameter will be loaded as local variables as well as the variables defined in the function.

3. infinite loop

I use Safari as my web browser, and it did't respond and the print in the result doesn't show anything.

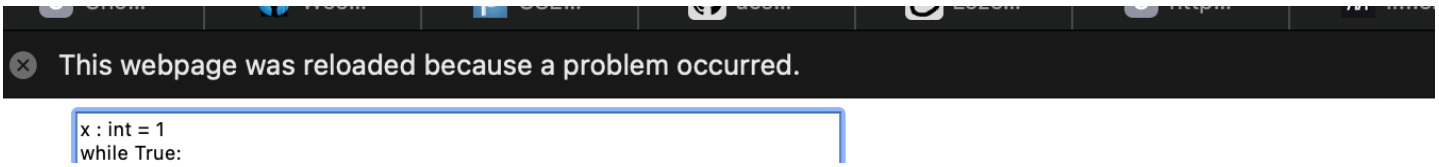
Run!

```
x : int = 1
while True:
    print(x)
```

```
    (module
      (func $print_num (import "imports" "print_num") (param i32) (result i32))
      (func $print_bool (import "imports" "print_bool") (param i32) (result i32))
      (func $print_none (import "imports" "print_none") (param i32) (result i32))
      (func $abs (import "imports" "abs") (param i32) (result i32))
      (func $max (import "imports" "max") (param i32 i32) (result i32))
      (func $min (import "imports" "min") (param i32 i32) (result i32))
      (func $pow (import "imports" "pow") (param i32 i32) (result i32))
      (global $x (mut i32) (i32.const 0))

      (func (export "_start")
        (local $scratch i32)
(i32.const 1)
(global.set $x)
(loop $while_loop_1649918448000
        (global.get $x)
(call $print_num)
(local.set $scratch)
        (i32.const 1)
        (br_if $while_loop_1649918448000))

      )
    )
```



And for a long the Safari give me a feedback.

I think maybe it is because the Safari will detect the abnormal memory consuming?

4. Different Scenario

(1) one operand is a call expression and the other operand is a variable

```
def f() ->int :
    y: int = 4
    return y

x : bool = True

f()+x
```

Run!

```
Error: TypeError: The type of bool doesn't match with int
```

(2) A program that has a type error in a conditional position

```
while x < 4:
```

```
while x < 4:
    x: int = 3
    x = x + 1
```

Run!

```
Error: TypeError: The type of int doesn't match with undefined
```

I didn't deal with particularly, but if above the while scope, the `x` is not define, then it cann't be success in binary expression.

(3) multiple iterations and call function

```
x: int = 4
while x > 0:
    x  = x -1
    print(x)
```

Run!

```
    (module
      (func $print_num (import "imports" "print_num") (param i32) (result i32))
      (func $print_bool (import "imports" "print_bool") (param i32) (result i32))
      (func $print_none (import "imports" "print_none") (param i32) (result i32))
      (func $abs (import "imports" "abs") (param i32) (result i32))
      (func $max (import "imports" "max") (param i32 i32) (result i32))
      (func $min (import "imports" "min") (param i32 i32) (result i32))
      (func $pow (import "imports" "pow") (param i32 i32) (result i32))
      (global $x (mut i32) (i32.const 0))

      (func (export "_start")
        (local $scratch i32)
(i32.const 4)
(global.set $x)
(loop $while_loop_1649920065000
        (global.get $x)
(i32.const 1)
i32.sub
(global.set $x)
(global.get $x)
(call $print_num)
(local.set $scratch)
        (global.get $x)
(i32.const 0)
i32.gt_s
        (br_if $while_loop_1649920065000))

      )
    )
```

```
3210undefined
```

(4) Actually, I didn't implemet `break` and `if` successfully, I have see the if else statment in WASM and realize I need write a resursive function, but I am not sure how to add flow control generator into the statement generator.

(5) Printing an integer and a boolean

```
print(True)
print(3)
```

Run!

```
    (module
      (func $print_num (import "imports" "print_num") (param i32) (result i32))
      (func $print_bool (import "imports" "print_bool") (param i32) (result i32))
      (func $print_none (import "imports" "print_none") (param i32) (result i32))
      (func $abs (import "imports" "abs") (param i32) (result i32))
      (func $max (import "imports" "max") (param i32 i32) (result i32))
      (func $min (import "imports" "min") (param i32 i32) (result i32))
      (func $pow (import "imports" "pow") (param i32 i32) (result i32))


      (func (export "_start") (result i32)
        (local $scratch i32)
(i32.const 1)
(call $print_bool)
(local.set $scratch)
(i32.const 3)
(call $print_num)
(local.set $scratch)
        (local.get $scratch)
      )
    )

True33
```

(6) and (7) I still cann't write, because I didn't implement the flow control. I have already completed its parse and type check.

5. Codes

In the (1) in question 4, I will store the type of call function in `a`, the annotation which is same as return type and type of variables in variables map.Then I will check the type of call fucntion expression and the type of variables. If the type doesn't match, it will throw the error.

```
export function tcExpr(e : Expr<any>, functions : FunctionsEnv, variables : BodyEnv) : Expr<Type> {
  switch(e.tag) {
    case "none": return {...e, a: "none"};
    case "number": return { ...e, a: "int" };
    case "true": return { ...e, a: "bool" };
    case "false": return { ...e, a: "bool" };
    case "binop": {
      const newRhs = tcExpr(e.rhs, functions, variables);
      const newLhs = tcExpr(e.lhs, functions, variables);
      switch(e.op) {
        case "+":
        case "-":
        case "*":
        case "//":
        case "%":
          if (newRhs.a != "int" || newLhs.a != "int"){
              throw new Error(`TypeError: The type of ${newRhs.a} doesn't match with ${newLhs.a}`)
          }
          return { tag: "binop", op: e.op , lhs:newLhs, rhs: newRhs, a: "int" };
```