Course Project
Phase #1. Type Checking

Prof. Jaeseung Choi

Dept. of Computer Science and Engineering

Sogang University

About the Course Project

- **Our source language will be a simple C-like language**
  - Let’s call this language **`Mini-C`**
  - So the input of our compiler will be a program written in **`Mini-C`**
- **We will pose many constraints to our `Mini-C` language**
  - Ex) A program consists of only one function
  - Ex) Support simple types like **`int`** and **`bool`** only
- **Despite such simplifications, we can still experience many important challenges in engineering a compiler**

Course Project Schedule

- **Phase #1: Type checking (5%): 11/13 - 11/22**
  - You must check the input AST and find type errors
  - It will be relatively easy (warm-up for getting familiar to the code)
- **Phase #2: IR generation (10%): 11/22 – 12/2**
  - You must convert the input AST into IR code
- **Phase #3: IR optimization (★ 20%): 12/2 – 12/22**
  - You must optimize IR code (= reduce the number of statements)
  - **Dependency:** You will work on top of your **Phase #2** code

Giving up Phase #1 ≈ Giving up the whole project

(Remind) Using ChatGPT

- **You can get the help from ChatGPT in this course**
  - But first try to write the code by yourself
  - Ask ChatGPT only after that, when you don’t have a clue
  - If you don’t practice programming at this stage, you will have a big trouble in the later phases
- **Starting from this project, you must submit a report if you used ChatGPT**
  - Write the exact **prompt you entered** to ChatGPT
  - Also, include the **code you obtained** from that prompt and **how/why you modified** that code
  - The main role of the report is to **justify that you did not copy** the solutions of other students

General Information

- **Check the "Project #1" post in *Cyber Campus***
  - Skeleton code (**`Prj1.tgz`**) is attached in the post
  - Deadline: **11/22** Friday 23:59
  - Submission will be accepted in that post, too
  - Late submission deadline: **11/24** Sunday 23:59 **(-20% penalty)**
- **Please read the instructions in this slide carefully**
  - The specification of the project is quite complex
  - The slide also contains important submission guidelines
    - If you do not follow the guidelines, **you will get penalty**

Skeleton Code

- **Copy `Prj1.tgz` into CSPRO server and decompress it**
  - You **must connect to** `csproN.sogang.ac.kr` **(N = 2, 3, or 7)**
  - Don’t decompress-and-copy; copy-and-decompress
- **`TypeCheck.fsproj`: F# project file (you may ignore)**
- **`src/`: Source files you have to work with**
- **`testcase/`: Sample test cases and their answers**
- **`check.py`: Script for self-grading with test cases**
- **`config`: Used by the grading script (you may ignore)**

```
jschoi@cspro2:~$ tar -xzf Prj1.tgz
jschoi@cspro2:~$ ls Prj1/
TypeCheck.fsproj  check.py  config  src  testcase
```

Structure of `src` Directory

- **`AST.fs` : Definition of the AST for program**
  - First, **you must read this file** to understand how the program AST is defined in F# code
- **`Lexer.fsl` & `Parser.fsy`: Input files for `F# Flex/Bison`**
  - This front-end is already implemented for you (you may ignore)
- **`Main.fs` : The main driver code**
  - It supports two modes that you can run (explained later)
- **`TypeCheck.fs` : Type checking (semantic analysis) logic**
  - This the **file that you must fill in** to implement the type checker

```
jschoi@cspro2:~/Prj1$ cd src/
jschoi@cspro2:~/Prj1/src$ ls
AST.fs  Lexer.fsl  Main.fs  Parser.fsy  TypeCheck.fs
```

Source Language: `Mini-C`

- **In this language, a program consists of one function**
  - Its name can be anything (doesn’t have to be **`main`**)
- **In this phase, we will only support `int`, `bool`, `int*`, and `bool*` types in our language**
- **The language has basic statements like assignment, `if-else`, and `while` loop**
  - But it will not have **`for`**, **`switch`**, etc. (front-end will raise error)
  - Read **`AST.fs`** file for more details

```
int f(int x, bool y) {
  if (x > 10) {
    while (...) { ... }
  }
}
```

Subtle Difference from Real C

- **Comment (`//` or `/*  */`) is not supported**
- **Explicit or implicit type conversion is not allowed**
- **Cannot declare multiple variables at once**
  - Ex) `"int x, y, z;"` **: not allowed in our language**
- **Pointer usage is limited**
  - Only single-level pointer is supported (no double pointer)
  - You can use `*` (dereference) only in front of a variable
  - Ex) `*(*p1) = *(p2 + 1);"` **: not allowed**
- **Cannot omit parentheses in `if`, `else`, or `while`**
  - Ex) `"if (b) x = 1;"` **: not allowed**
- **Many of these will be reported as error in the front-end**

Errors to Detect

- **Use of undeclared identifiers (variables)**
- **Type mismatches in expression**
  - Arithmetic operations (`+`, `-`, `*`, `/`) are allowed only for `int` types, and the outputs are also `int`
    - Note that pointer arithmetic is not allowed
  - Comparison operations (`>`, `>=`, `<=`, `<`) are allowed only `int` types, and the outputs are `bool`
  - Equality check operations (`==`, `!=`) are allowed between the same types, and the outputs are `bool`
    - Same pointer types (`int*` vs. `int*`) are also allowed
  - Logical operations (`&&`, `||`, `!`) are allowed only for `bool` types, and the outputs are `bool`

Errors to Detect (Cont')

- **Type mismatches in statements**
  - In assign statement, left and right side must properly match
    - Ex) "`int i = true;`" **: Error**
    - Ex) "`bool b; int *p = &b;`" **: Error**
  - For conditions, any type can come (`bool`, `int`, pointer)
    - Ex) "`if (true) {...}`", "`if (1) {...}`" **: Both OK**
    - Ex) "`int *p = ...; if (ptr) {...}`" **: Also OK**
  - Operand of `return` must match with the function type
    - Ex) "`int f(bool x) { return true; }`" **: Error**
  - Consider `NULL` as compatible with any pointer type
    - Ex) "`int *p = NULL;`" **: OK**
    - Ex) "`int *p = &i; if (p == NULL) {...}`" **: OK**

Errors to Ignore

- **Following cases are obviously semantic errors, but they are out the scope of the compiler’s type checking**
  - **Missing `return`** (in a function that must return something)
  - **Division by zero**
  - **NULL dereference**
  - **Use of uninitialized variable**
- **In general setting, these are hard to detect correctly**
  - Even real-world compilers do not catch these errors

```
int f(int n) {
  int x = n / 0; // Div-by-0
} // Didn’t return anything
```

```
int g(int n) {
  int a; int b;
  int *p = NULL;
  *p = n; // NULL dereference
  a = b; // Uninitialized var
}
```

Errors to Ignore (Cont')

- **Redeclaration of a variable is also an obvious error, and it can be actually detected by compiler's type checking**
  - But catching this can be a little bit tricky for you at this point
  - As I said, I'm trying to make **Phase #1** as easy as possible
- **For simplicity, let's assume our inputs (test cases) do not contain this kind of error**
  - Thus, your type checker **does not have to** care about this

```
// I will not use a test case like this
int f(int x) {
  int x = 1; // Error
  int y = 2;
  bool y = 3; // Error
}
```

Declaration within Block

- **Be careful: declaring a variable with the same name in an inner block is a valid behavior!**
  - Also, note that the a variable declared inside a block persists only within that block (cf. *scope* of a variable)
  - Therefore, the example code below is a valid program

```
// Following test case can be used in the grading
int f(bool b) {
  int x = 1;
  if (x == 1) {
     bool x = true; // This is not an error
  }
  int y = x; // After exiting "if", x becomes int again
}
```

You Mission: `TypeCheck.run`

- **In `src/TypeCheck.fs` file, you have to implement the following function (its type must not change)**
  - `let run (prog: Program) : LineNo list = ...`
  - This function must return the **list of line numbers** that contain semantic errors you found
  - **`LineNo`** is defined as `int` in **`AST.fs`**
- **I already provided some code as a guideline**
  - Key functions are declared with appropriate type and comment
  - You may define more functions if you want
  - Moreover, you may even choose to delete all the provided code and write everything from scratch
  - FYI, my reference solution is about 150 lines

Mode 1: Printing Program AST

- **After building the project with `dotnet build -o out`, you can print the AST of input programs as follow**
  - With this, you can see how the program is parsed into AST

```
jschoi@cspro2:~/Prj1$ dotnet build -o out
...
jschoi@cspro2:~/Prj1$ ./out/TypeCheck print-ast testcase/prog-1
int f(bool b, int i) [
  int x = i,
  int y = b,
  if ((i > 5)) [
    i = (i - (b * 2))
  ] else [

  ],
  return 0
]
```

Explicitly shows you the precedence of operators

Mode 2: Running Type Checker

- **Next, you can run the type checking as follow**
  - It prints out the list of error line numbers returned by your type checker (i.e., **`TypeCheck.run`** function)
  - Read **`Main.fs`** for more details about each mode
- **Once you complete, the output for `testcase/prog-N` must match with the content of `testcase/ans-N`**

```
jschoi@cspro2:~/Prj1$ dotnet build -o out
...
jschoi@cspro2:~/Prj1$ ./out/TypeCheck check-error testcase/prog-1
3
6
jschoi@cspro2:~/Prj1$ $ cat testcase/ans-1
3
6
```

Self-Grading

- **If you think you have solved all the problems, you can run `check.py` as a final check**
  - `'O'`: Correct, `'X'`: Incorrect, `'E'`: exception, `'C'`: Compile error, `'T'`: Timeout (maybe infinite recursion)
- **Recall that I use different test cases in the real grading**
- **You can assume the followings for the test cases**
  - Test cases with lexing or parsing error will not be used
  - Test cases will not contain the errors that we promised to ignore in the previous page

```
jschoi@cspro2:~/Prj1$ ./check.py
[*] Result : OOXX
```

Submission Guideline

- **You should submit one F# source code file and report**
  - **`TypeCheck.fs`**
  - **`report.pdf`** (Please use the PDF format)
- **The whole project must properly compile when I copy the `TypeCheck.fs` file you submitted**
  - If the skeleton code does not build, cannot give you any point
- **Also, don’t forget the report this time**
- **Submission format**
  - Upload these files directly to *Cyber Campus* (**do not zip them**)
  - **Do not change the file name** (e.g., adding any prefix or suffix)
  - If your submission format is wrong, you will get **-20% penalty**