# Lab #2. C- Interpreter

Prof. Jaeseung Choi
Dept. of Computer Science and Engineering
Sogang University

# General Information

- **Check the *Assignment* tab of *Cyber Campus***
  - Skeleton code (`Lab2.tgz`) is attached together with this slide
  - Submission will be accepted in the same post, too
- **Deadline: 4/14 Sunday 23:59**
  - Late submission deadline: **4/16 Tuesday 23:59** **(-20% penalty)**
  - Delay penalty is applied uniformly **(not problem by problem)**
- **Please read the instructions in this slide carefully**
  - This slide is a step-by-step tutorial for the lab
  - It also contains important submission guidelines
    - If you do not follow the guidelines, **you will get penalty**

# Skeleton Code Structure

- **Copy `Lab2.tgz` into CSPRO server and decompress it**
  - This course will use [cspro**2**.sogang.ac.kr](cspro2.sogang.ac.kr) (**don't miss the 2**)
  - Don't decompress-and-copy; copy-and-decompress
- **`CMinus`: Directory for the first version of `C-` language**
- **`CMinusPtr`: Directory for the extended version of `C-`**
- **`check.py`: Script for self-grading (explained later)**
- **`config`: Used by the grading script (you may ignore)**

```
jschoi@cspro2:~$ tar -xzf Lab2.tgz
jschoi@cspro2:~$ cd Lab2/
jschoi@cspro2:~/Lab2$ ls
CMinus  CMinusPtr check.py config
```

# Directory Structure of `CMinus`

- **Skeleton code of interpreter is provided under `src/`**
  - `AST.fs`: Syntax definition of the `C-` language
  - `CMinus.fs`: You have to **implement the semantics** here
  - `Types.fs`: Type definitions needed for semantics
  - `Main.fs`: Main driver code of the interpreter
  - `Lexer.fsl`, `Parser.fsy`: Parser (you don't have to care)
- **Do NOT fix any source files other than `CMinus.fs`**

```
jschoi@cspro2:~/Lab2$ cd CMinus/
jschoi@cspro2:~/Lab2/CMinus$ ls
CMinus.fsproj  src  testcase
jschoi@cspro2:~/Lab2/CMinus$ ls src
AST.fs  CMinus.fs  Lexer.fsl  Main.fs  Parser.fsy  Types.fs
```

# C- Language Syntax

- **Mostly similar to the `C-` in the lecture note**
  - A program is a statement
  - $\boldsymbol{n}$ means an integer ($n \in \boldsymbol{Z}$), $\boldsymbol{x}$ means a variable ($\boldsymbol{x} \in \boldsymbol{Var}$)

$$
\begin{aligned}
E \rightarrow\ & n \\
& \mid \textbf{true} \\
& \mid \textbf{false} \\
& \mid x \\
& \mid E + E \\
& \mid E - E \\
& \mid E < E \\
& \mid E > E \\
& \mid E == E \\
& \mid E\ != E
\end{aligned}
$$

**Expression**

$$
\begin{aligned}
S \rightarrow\ & \textbf{NOP} \\
& \mid x = E \\
& \mid S; S \\
& \mid \textbf{if}\ (E)\ \{\ S\ \}\ \textbf{else}\ \{\ S\ \} \\
& \mid \textbf{while}\ (E)\ \{\ S\ \}
\end{aligned}
$$

**Statement**

# What is NOP for?

- **NOP is a statement that does nothing (*no-operation*)**
- **It is introduced for the convenience of the syntax**
  - Ex) If you mistakenly put `;` after the last statement
  - Ex) If you leave the body of `if-else` empty ( `{ }` )

If you write like this...

```
x = 1; y = 2; z = 3;
```

```
if(x < 5) {
  y = 1;
} else { }
```

Parser will recognize as follow

```
x = 1; y = 2; z = 3; NOP
```

```
if(x < 5) {
  y = 1;
} else { NOP }
```

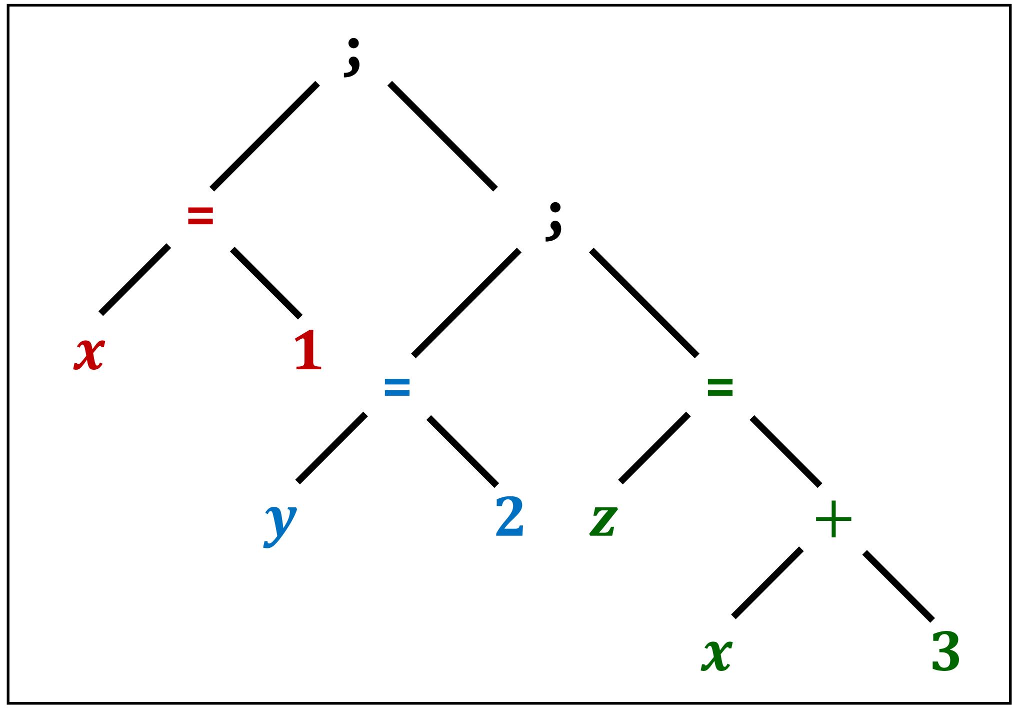
# AST in Skeleton Code

- **`AST.fs` contains the `F#` type to represent the program**
  - AST (abstract syntax tree) represents a program written in **`C-`**

```
x = 1;
y = 2;
z = x + 3
```

Program String

*Parsing*




Abstract Syntax Tree

# AST in Skeleton Code

- **`AST.fs` contains the `F#` type to represent the program**
  - AST (abstract syntax tree) represents a program written in **`C-`**
  - **Parser is already implemented** in the skeleton code

```
x = 1;
y = 2;
z = x + 3
```

Program String

*Parsing*

```
let s1 = Assign ("x", Num 1)
let s2 = Assign ("y", Num 2)
let exp = Add (Var "x", Num 3)
let s3 = Assign ("z", exp)
let prog = Seq (s1, (Seq (s2, s3)))
```

Abstract Syntax Tree in **`F#`** Code

# C- Language Semantics

- **Relation $M \vdash e \Downarrow v$ defines the evaluation of expression**
  - Similarly to our lecture note, assume $\boldsymbol{n} \in \boldsymbol{Z}$ (integers), $\boldsymbol{b} \in \boldsymbol{B}$ (Booleans), $\boldsymbol{v} \in \boldsymbol{Val} = \boldsymbol{Z} + \boldsymbol{B}$ (Value), $\boldsymbol{M} \in \boldsymbol{Mem} = \boldsymbol{Var} \to \boldsymbol{Val}$

$$\frac{}{M \vdash n \Downarrow n} \qquad \frac{}{M \vdash \textbf{true} \Downarrow true} \qquad \frac{}{M \vdash \textbf{false} \Downarrow false} \qquad \frac{}{M \vdash x \Downarrow M(x)}$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 + e_2 \Downarrow n_1 + n_2} \qquad \frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 - e_2 \Downarrow n_1 - n_2}$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 < e_2 \Downarrow true}\ n_1 < n_2 \qquad \frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 < e_2 \Downarrow false}\ n_1 \geq n_2$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 > e_2 \Downarrow true}\ n_1 > n_2 \qquad \frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 > e_2 \Downarrow false}\ n_1 \leq n_2$$

# C- Language Semantics

- **Relation $\boldsymbol{M \vdash e \Downarrow v}$ defines the evaluation of expression**
  - Similarly to our lecture note, assume $\boldsymbol{n} \in \boldsymbol{Z}$ (integers), $\boldsymbol{b} \in \boldsymbol{B}$ (Booleans), $\boldsymbol{v} \in \boldsymbol{Val} = \boldsymbol{Z} + \boldsymbol{B}$ (Value), $\boldsymbol{M} \in \boldsymbol{Mem} = \boldsymbol{Var} \rightarrow \boldsymbol{Val}$
  - Note the **subtle difference** from the semantics in lecture note

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 == e_2 \Downarrow true} \; (v_1 = v_2 = n) \lor (v_1 = v_2 = b)$$

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 == e_2 \Downarrow false} \; (v_1 = n_1 \neq n_2 = v_2) \lor (v_1 = b_1 \neq b_2 = v_2)$$

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 \; != e_2 \Downarrow false} \; (v_1 = v_2 = n) \lor (v_1 = v_2 = b)$$

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 \; != e_2 \Downarrow true} \; (v_1 = n_1 \neq n_2 = v_2) \lor (v_1 = b_1 \neq b_2 = v_2)$$

# C- Language Semantics

- **Relation $\langle \boldsymbol{M}, \boldsymbol{s} \rangle \Rightarrow \boldsymbol{M'}$ defines the execution of statement**
  - Given program $\boldsymbol{p}$, its semantics is defined (= can be executed successfully) if we can derive $\langle \boldsymbol{\phi}, \boldsymbol{p} \rangle \Rightarrow \boldsymbol{M}$ for some $\boldsymbol{M}$

$$\frac{}{\langle M, \mathbf{NOP} \rangle \Rightarrow M} \qquad \frac{M \vdash e \Downarrow v}{\langle M, x = e \rangle \Rightarrow M[x \mapsto v]} \qquad \frac{\langle M, s_1 \rangle \Rightarrow M_1 \quad \langle M_1, s_2 \rangle \Rightarrow M_2}{\langle M, s_1; s_2 \rangle \Rightarrow M_2}$$

$$\frac{M \vdash e \Downarrow true \quad \langle M, s_1 \rangle \Rightarrow M'}{\langle M, \mathbf{if}\,(e)\,\{\, s_1 \}\;\mathbf{else}\,\{\, s_2 \} \rangle \Rightarrow M'} \qquad \frac{M \vdash e \Downarrow false \quad \langle M, s_2 \rangle \Rightarrow M'}{\langle M, \mathbf{if}\,(e)\,\{\, s_1 \}\;\mathbf{else}\,\{\, s_2 \} \rangle \Rightarrow M'}$$

$$\frac{M \vdash e \Downarrow false}{\langle M, \mathbf{while}\,(e)\,\{\, s \,\} \rangle \Rightarrow M} \qquad \frac{M \vdash e \Downarrow true \quad \langle M, s \rangle \Rightarrow M_1 \quad \langle M_1, \text{while}\,(e)\,\{\, s \,\} \rangle \Rightarrow M_2}{\langle M, \mathbf{while}\,(e)\,\{\, s \,\} \rangle \Rightarrow M_2}$$

# Implementing Semantics

- **To complete the interpreter of `C-`, you must implement the semantics of C- language in `CMinus.fs` file**
  - You have to implement two functions: **`evalExp()`** and **`exec()`**
  - Type definition of **`Mem`** and **`Val`** are provided in **`Types.fs`**
  - If the semantics of program is not defined, your interpreter must raise **`UndefinedSemantics`** exception defined in **`Types.fs`**

```
let rec evalExp (exp: Exp) (mem: Mem) : Val =
  ...

let rec exec (stmt: Stmt) (mem: Mem) : Mem =
  ...
```

# Building and Testing

- **In `testcase` directory, `tc-*` and `ans-*` files are provided**
  - After compiling the interpreter with the **`dotnet build -o out`** command, you can run program written in **`C-`** language
  - The interpreter will **print the content of final output memory**

```
jschoi@cspro2:~/Lab2/CMinus$ cat testcase/tc-1
x = 1;
y = x + 2
jschoi@cspro2:~/Lab2/CMinus$ dotnet build -o out
...
jschoi@cspro2:~/Lab2/CMinus$ ./out/CMinus testcase/tc-1
{
  x -> 1
  y -> 3
}
```

This result must match with the content of **`ans-1`** (expected output)

# `C-` with Pointer

- **Now let's move on to `CMinusPtr` directory**
- **This directory contains the interpreter for `C-` language extended to have pointer (&x and *e)**
- **The structure of skeleton code is same to `CMinus`**
  - This time, you have to fill in **`CMinusPtr.fs`**
  - You can reuse parts of the code from your **`CMinus.fs`**

```
jschoi@cspro2:~/Lab2$ ls
check.py  CMinus  CMinusPtr  config
jschoi@cspro2:~/Lab2$ cd CMinusPtr/
jschoi@cspro2:~/Lab2$ ls
CMinusPtr.fsproj  src  testcase
jschoi@cspro2:~/Lab2$ ls src
AST.fs  CMinusPtr.fs  Lexer.fsl  Main.fs  Parser.fsy  Types.fs
```

# `C-` Language Syntax (Extended)

- **Mostly similar to the `C-` in the lecture note**
  - Changed parts are highlighted

$$
\begin{aligned}
E \rightarrow\ & n \\
\mid\ & \textbf{true} \\
\mid\ & \textbf{false} \\
\mid\ & \&x \\
\mid\ & LV \\
\mid\ & E + E \\
\mid\ & E - E \\
\mid\ & E < E \\
\mid\ & E > E \\
\mid\ & E == E \\
\mid\ & E \mathrel{!}= E
\end{aligned}
$$

**Expression**

$$
\begin{aligned}
LV \rightarrow\ & x \\
\mid\ & *E
\end{aligned}
$$

**L-value**

$$
\begin{aligned}
S \rightarrow\ & \textbf{NOP} \\
\mid\ & LV = E \\
\mid\ & S; S \\
\mid\ & \textbf{if}\ (E)\ \{\ S\ \}\ \textbf{else}\ \{\ S\ \} \\
\mid\ & \textbf{while}\ (E)\ \{\ S\ \}
\end{aligned}
$$

**Statement**

# C- Language Semantics (Extended)

- **Relation $M \vdash lv \downarrow l$ defines the evaluation of l-value**
  - $l \in Var$ represents a memory location (variable)
- **Relation $M \vdash e \Downarrow v$ defines the evaluation of expression**
  - $v \in Val = Z + B + Var$ represents a value
- **New parts and changed parts are highlighted**

$$\frac{}{M \vdash x \downarrow x} \qquad \frac{M \vdash e \Downarrow l}{M \vdash *e \downarrow l}$$

---

$$\frac{}{M \vdash n \Downarrow n} \qquad \frac{}{M \vdash \textbf{true} \Downarrow true} \qquad \frac{}{M \vdash \textbf{false} \Downarrow false}$$

$$\frac{}{M \vdash \&x \Downarrow x} \qquad \frac{M \vdash lv \downarrow l}{M \vdash lv \Downarrow M(l)}$$

# `C-` Language Semantics (Extended)

- **Relation $M \vdash e \Downarrow v$ defines the evaluation of expression**
  - Assume $n \in Z$, $b \in B$ (Boolean), $l \in Var$ (variable name)
  - There is no change to the semantics of **+**, **-**, **<** and **>**

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 + e_2 \Downarrow n_1 + n_2}$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 - e_2 \Downarrow n_1 - n_2}$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 < e_2 \Downarrow true}\ n_1 < n_2$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 < e_2 \Downarrow false}\ n_1 \geq n_2$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 > e_2 \Downarrow true}\ n_1 > n_2$$

$$\frac{M \vdash e_1 \Downarrow n_1 \quad M \vdash e_2 \Downarrow n_2}{M \vdash e_1 > e_2 \Downarrow false}\ n_1 \leq n_2$$

# C- Language Semantics (Extended)

- **Relation $M \vdash e \Downarrow v$ defines the evaluation of expression**
  - Assume $n \in Z$, $b \in B$ (Boolean), $l \in Var$ (variable name)
  - Note the changes to the semantics of `==` and `!=`

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 == e_2 \Downarrow true} (v_1 = v_2 = n) \vee (v_1 = v_2 = b) \vee (v_1 = v_2 = l)$$

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 == e_2 \Downarrow false} (v_1 = n_1 \neq n_2 = v_2) \vee (v_1 = b_1 \neq b_2 = v_2) \vee (v_1 = l_1 \neq l_2 = v_2)$$

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 \ ! = e_2 \Downarrow false} (v_1 = v_2 = n) \vee (v_1 = v_2 = b) \vee (v_1 = v_2 = l)$$

$$\frac{M \vdash e_1 \Downarrow v_1 \quad M \vdash e_2 \Downarrow v_2}{M \vdash e_1 \ ! = e_2 \Downarrow true} (v_1 = n_1 \neq n_2 = v_2) \vee (v_1 = b_1 \neq b_2 = v_2) \vee (v_1 = l_1 \neq l_2 = v_2)$$

# C- Language Semantics (Extended)

- **Relation $\langle M, s\rangle \Rightarrow M'$ defines the execution of statement**
  - Assignment (`lv = e`) is the only affected statement

$$\frac{}{\langle M, \textbf{NOP}\rangle \Rightarrow M} \qquad \frac{M \vdash lv \downarrow l \qquad M \vdash e \Downarrow v}{\langle M, lv = e\rangle \Rightarrow M[l \mapsto v]} \qquad \frac{\langle M, s_1\rangle \Rightarrow M_1 \qquad \langle M_1, s_2\rangle \Rightarrow M_2}{\langle M, s_1; s_2\rangle \Rightarrow M_2}$$

$$\frac{M \vdash e \Downarrow true \qquad \langle M, s_1\rangle \Rightarrow M'}{\langle M, \textbf{if}\ (e)\ \{\ s_1\}\ \textbf{else}\ \{\ s_2\}\rangle \Rightarrow M'} \qquad \frac{M \vdash e \Downarrow false \qquad \langle M, s_2\rangle \Rightarrow M'}{\langle M, \textbf{if}\ (e)\ \{\ s_1\}\ \textbf{else}\ \{\ s_2\}\rangle \Rightarrow M'}$$

$$\frac{M \vdash e \Downarrow false}{\langle M, \textbf{while}\ (e)\ \{\ s\ \}\rangle \Rightarrow M} \qquad \frac{M \vdash e \Downarrow true \qquad \langle M, s\rangle \Rightarrow M_1 \qquad \langle M_1, \text{while}\ (e)\ \{\ s\ \}\rangle \Rightarrow M_2}{\langle M, \textbf{while}\ (e)\ \{\ s\ \}\rangle \Rightarrow M_2}$$

# Self-Grading Script

- **If you think you have solved all the problems, you can run `check.py` as a final check**
  - '`O`': Correct, '`X`': Incorrect, '`E`': Unhandled exception in your code
  - '`C`': Compile error, '`T`': Timeout (maybe infinite recursion)
- **If you correctly raise `UndefinedSemantics` exception for an invalid program, it will be graded as '`O`' (not '`E`')**
  - If you raise **`UndefinedSemantics`** for valid program, it is **'`X`'**

```
jschoi@cspro2:~/Lab2$ ls
check.py   CMinus   CMinusPtr   config
jschoi@cspro2:~/Lab2$ $ ./check.py
[*] CMinus      : OOO
[*] CMinusPtr   : OOO
```

# Problem Information

- **Two sub-problems**
  - **`CMinus`:** 60 point
  - **`CMinusPtr`:** 40 point
  - 100 point in total (but recall that each lab has different weight)
- **You will get the point for each problem based on the number of test cases that your code passes**
  - You are encouraged to run you code with your own test cases (try to think of various inputs)
  - Some students ask me to provide more test cases, but **it is important to practice this on your own**

# Submission Guideline

- **You should submit two F# source code files**
  - **`CMinus.fs` (from `Lab2/CMinus/src/CMinus.fs`)**
  - **`CMinusPtr.fs` (from `Lab2/CMinusPtr/src/CMinusPtr.fs`)**
- **If the submitted file fails to compile with skeleton code when I type "`dotnet build`", cannot give you any point**
- **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**