San José State University
Department of Computer Engineering

CMPE 152
Compiler Design
Spring 2018
Instructor: Ron Mak

Assignment #3

Assigned: Tuesday, September 11
Due: Friday, September 28 at 11:59 pm
Team assignment, 100 points max

WHEN statement

The purpose of this assignment is to give your project team experience adding a **parser** in the front end for a new Pascal control statement, designing a new **parse tree** in the intermediate tier for this statement, and then adding an **executor** for the new parse tree in the back end.

You should start with code from Chapter 8.

Here is an example of the new **WHEN** statement:

```
WHEN
    i = 1 => f := 10;
    i = 2 => f := 20;
    i = 3 => f := 30;
    i = 4 => f := 40;
    OTHERWISE => f := -1
END
```

Syntax

A **WHEN** statement consists of a sequence of *when-branches* separated by semicolons. The last when-branch must be an **OTHERWISE** branch.

A when-branch consists of a selector and a statement to execute. The selector is an *arbitrary boolean expression* followed by the *arrow special symbol* => and then a *single branch statement* that corresponds to the selector. This single statement can be a compound statement. The **OTHERWISE** branch must come last, and it has the new reserved word **OTHERWISE** instead of a boolean expression.

Semantics

A **WHEN** statement is another way to write cascading **IF-ELSE** statements. The above example is equivalent at run time to

```
IF      i = 1 THEN f := 10
ELSE IF i = 2 THEN f := 20
ELSE IF i = 3 THEN t := 30
ELSE IF i = 4 THEN f := 40
ELSE               f := -1;
```

Starting with the first when-branch, evaluate each boolean expression in turn. If a boolean expression evaluates to true, execute the corresponding branch statement, and then immediately leave the **WHEN** statement. Therefore, you can execute at most one branch statement, even if more than one boolean expression evaluates to true. However, if none of the boolean expressions evaluates to true, execute the **OTHERWISE** branch statement.

Input files

Test your modified compiler with the following input file **when.txt**:

```
BEGIN
    i := 3;

    WHEN
        i = 1 => f := 10;
        i = 2 => f := 20;
        i = 3 => f := 30;
        i = 4 => f := 40;
        OTHERWISE => f := -1
    END;

    range := 5.7;

    WHEN
        (1.0 <= range) AND (range < 3.0) => level := 1;
        (4.5 <= range) AND (range < 7.5) => BEGIN
                                                level := 2;
                                                alpha := range;
                                            END;
        (8.0 <= range) AND (range < 9.9) => level := 3;
        OTHERWISE => level := -1
    END;
END.
```

Test your modified compiler's syntax error handling with the following file **whenerrors.txt**: Your modified compiler should parse the entire file and flag each error without crashing.

```
BEGIN
    WHEN
        i := 3 => k > 5;
        m = n  :  m := 2*n;
    END;
END.
```

Output

Create text files of the output of the interpreter. Input file **when.txt** should generate a program listing, a cross-reference table, a parse tree, and runtime messages from executing the assignment statements. Input file **whenerrors.txt** should generate a program listing with syntax error messages.

Source files to modify or add

Start with the C++ source files from Chapter 8. Do the work in two stages:

- Stage 1: Modify the frontend parser to parse the **`WHEN`** statement and build an appropriate parse tree.
    - **`PascalToken.h`**
    - **`PascalToken.cpp`**
    - **`PascalSpecialSymbolToken.cpp`**
    - **`ICodeNodeImpl.h`**
    - **`ICodeNodeImpl.cpp`**
    - **`WhenStatementParser.h`**
    - **`WhenStatementParser.cpp`**
    - **`StatementParser.cpp`**
- Stage 2: Modify the backend executor to execute the parse tree.
    - **`WhenExecutor.h`**
    - **`WhenExecutor.cpp`**
    - **`StatementExecutor.cpp`**

Not all the source files may be listed above.

TIP: If your new **`WHEN`** parser generates a parse tree that is equivalent to cascading **`IF-ELSE`** statements, then Stage 2 won't be necessary.

What to turn in

Create a zip file that contains:

- All the source files of your modified compiler. The ISA must be able to compile and run your compiler.
- A syntax diagram (hand-drawn is OK) of the new **`WHEN`** statement.
- An example parse tree (hand-drawn is OK) that your parser generates for an example **`WHEN`** statement.
- Text files that contain output from running your modified compiler on source files **when.txt** and **whenerrors.txt**.

Submit into Canvas: **Assignment #3. WHEN statement**

Rubric

Your program will be graded according to these criteria:

Criteria	Maximum points
Design • **`WHEN`** syntax diagram • **`WHEN`** parse tree example	**20** • 10 • 10
Modified compiler • Frontend parsers • Backend executors	**50** • 30 • 20
Output • Generated from **when.txt** • Generated from **whenerrors.txt**	**30** • 15 • 15