CS 5500 – The Structure of a Compiler
HW #4

Due by 2 p.m. on Friday, February 26, 2016

- This assignment will be worth **13%** of your course grade.
- You may work on this assignment **with at most one other person in this class**.
- You are strongly encouraged to **take a look at all of the sample input and output files** posted on the Blackboard website **before** you actually submit your assignment for grading.

Basic Instructions:

For this assignment you are to modify your code from HW #3 to make it also do **semantic analysis**.

If your *flex* file was named **mipl.l** and your ***bison*** file was named **mipl.y**, you should be able to compile and execute your program on one of the campus Linux machines (such as rc*nn*ucs213.managed.mst.edu where ***nn*** is **01-08**) using the following commands (where *inputFileName* is the name of some input file):

```
flex mipl.l
bison mipl.y
g++ mipl.tab.c -o mipl_parser
mipl_parser < inputFileName
```

As in HW #3, **no attempt should be made to recover from errors**. Every error message should **include the line number** where your compiler detected the error. Listed below are the **new errors** that you must be able to detect:

- **Procedure/variable mismatch**
- **Indexed variable must be of array type**
- **Start index must be less than or equal to end index of array**
- **Index expression must be of type integer**
- **Expression must be of type boolean**
- **Expression must be of type integer**
- **Expressions must both be int, or both char, or both boolean**
- **Expression must be of same type as variable**
- **Input variable must be of type integer or char**
- **Output expression must be of type integer or char**
- **Cannot make assignment to an array**

Sample input and output files are posted on the Blackboard website. They give examples of every type of semantic error that you need to be able to detect.

Regarding legal expression types for the various operators in MIPL, for the **binary arithmetic operators** (i.e., **+**, **-**, *****, ***div***), both operand expressions must be **integer** (and the resulting type of the expression would be **integer**). For the **binary logical operators** (i.e., ***and***, ***or***), both operand expressions must be **boolean** (and the resulting type of the expression would be **boolean**). For the **binary relational operators** (i.e., **=**, **>**, **<**, **<>**, **>=**, **<=**), both operand expressions must be of the **same type**, and must be **either char or integer or boolean** (and the resulting type of the expression would be **boolean**). For the **unary logical operator *not***, the operand expression must be **boolean** (and the resulting type of the expression would be **boolean**). For the **unary arithmetic operators +** and **-**, the operand expression must be **integer** (and the resulting type of the expression would be **integer**). Because the acceptable operand expression types are dependent upon the operator and because the MIPL grammar has arithmetic and logical operators "mixed" in <add op> and <mult op>, you are allowed to modify the grammar slightly as follows:

<add op> → <add op logical> | <add op arithmetic>
<add op logical> → **or**
<add op arithmetic> → **+ | -**
<mult op> → <mult op logical> **|** <mult op arithmetic>
<mult op logical> → **and**
<mult op arithmetic> → *** | div**

You then could do something similar to the following so that you'll know whether the operator was a logical or an arithmetic operator when <add op> or <mult op> is used on the right hand side of some production:

```
#define LOGICAL_OP      100
#define ARITHMETIC_OP 101
...
%union {
  char* text;
  TYPE_INFO typeInfo;
   int intValue;
};
...
```

```
%type<intValue> N_ADD_OP  N_MULT_OP
...

N_ADD_OP  : N_ADD_OP_LOGICAL
             {
              ...
              $$ = LOGICAL_OP;
             }
            | N_ADD_OP_ARITHMETIC
              {
              ...
              $$ = ARITHMETIC_OP;
              }
            ;

N_MULT_OP : N_ MULT_OP_LOGICAL
             {
              ...
              $$ = LOGICAL_OP;
             }
            | N_ MULT_OP_ARITHMETIC
              {
              ...
              $$ = ARITHMETIC_OP;
              }
...
```

<u>Sample Input and Output:</u>

In this part of the MIPL compiler, there are situations where there is more than one legitimate option for where exactly in the grammar to check for various types of errors. Therefore, it could be difficult to get your output to exactly match ours in terms of where error messages appear with respect to where it outputs the productions that are being applied. For this reason, you should **suppress the output of token information and productions being applied in the output when you submit your assignment for grading**. All we should see is either an error message or the “Completed parsing” message (if no errors were found) in the input file. You are advised to use some internal Boolean variable (**<u>NOT</u>** a command line argument!) to “turn off” this output; although for debugging purposes you may want to still be able to “turn on” this output.

Sample input and output files are posted on Blackboard. Note that these files were created on a Windows PC, so you should run *dos2unix* on them before using them on a Linux/Unix machine. With the exception of whitespace and capitalization, the output produced by your program **<u>MUST</u>** be **<u>IDENTICAL</u>** to what our “grader” program produces! Otherwise, the grading script says your program is incorrect and the instructor has to grade your program manually ☹

What To Submit For Grading:

You should submit via Blackboard a single **zip** or **tar** file containing your *flex* and *bison* files **as well as any .h files necessary for your symbol table**. Note that a *make* file will **NOT** be accepted (since that is not what the automated grading script is expecting); your *flex* and *bison* files must ***#include*** your .h files as necessary. Name your *flex* and *bison* files using your last name followed by your first initial (e.g., Homer Simpson would name his files **simpsonh.l** and **simpsonh.y**). Also name your **zip** or **tar** file accordingly (e.g., **simpsonh.zip**).

If you work with someone, instead name your *flex*, *bison*, and zip (or tar) files as the combination of each of your last names. For example, if Daffy Duck and Bugs Bunny worked together, their submission would be named duckbunny or bunnyduck. Also, if you work with someone, **only submit under ONE person’s username**, **NOT both! We don’t want to grade your program twice!**

Submitting from Blackboard:
1. Go to **Assignments** -> **HW #4** (although it may not appear that you can click on **HW #4**, you actually can)
2. Scroll down to **Section 2. Assignment Materials**. Under the text box, click on “Browse My Computer” to attach a file. Find and attach your *zip or tar* file.
3. Scroll to the top or the bottom, and click submit. Confirm your submission.

***WARNING*: If you fail to follow all of these instructions, the automated grading script may reject your submission, in which case it will NOT be graded!!! ☹**