Assignment 3

Due Dates: 12-6(MW) and 12-7(TTh)-2000

Assignment 3 consists of 1) Symbol table handling and 2) Generating an assembly code for the simplified version of Rat00F.

**Simplified Rat00F:** The simplified Rat00F is essentially the same except that a program has only one `<Function Definition>` with “main()”. Therefore, the grammar simplifies as follows:

```latex
<Primary> ::= <Identifier> | <Integer> | (<Expression>) | true | false
```

Therefore, `<Opt Parameter List>` is also no longer needed. Furthermore, `<Return>` and `<Expression List>` are not necessary. However, we can have multiple declarations in the “main()” function.

1) Symbol table Handling (2%):
Every identifier declared in the program should be placed in a symbol table and accessed by the symbol table handling procedures.

a) Each entry in the symbol table should hold the lexeme, and a “memory address” where an identifier is placed within the symbol table. For example, define a global integer variable called “Memory_address” and set initially 100 and increment it by one when a new identifier is declared and placed into the table.

b) You need to write a procedure that will check to see if a particular identifier is already in the table, a procedure that will insert into the table and a procedure that will printout all identifiers in the table. If an identifier is used without declaring it, then the parser should provide an error message. Also, if an identifier is already in the table and wants to declare it for the second time, then the parser should provide an error message. Also, you should check the type match.

2) Generating the assembly code (8%):
Modify your parser according to the simplified Rat00F and add code to your parser that will produce the assembly code instructions. The instructions should be kept in an array and at the end, the content of the array is printed out to produce the listing of assembly code. Your array should hold at least 300 assembly instructions.

The listing should include an array index for each entry so that it serves as label to jump to. The compiler should also produce a listing of all the identifiers.
Our target machine is a simple stack machine with following instructions

- **PI** `{Integer Value}` Pushes the `{Integer Value}` onto the Top of the Stack (TOS)
- **PM** `{ML - Memory Location}` Pushes the value stored at `{ML}` onto TOS
- **RM** `{ML}` Pops the value from the top of the stack and stores it at `{ML}`
- **RS** Pops the value from TOS and outputs it to the standard output
- **PS** Get the value from the standard input and place in onto the TOS
- **A** Pop the first two items from stack and push the sum onto the TOS
- **S** Pop the first two items from stack and push the difference onto the TOS (Second item – First item)
- **M** Pop the first two items from stack and push the product onto the TOS
- **D** Pop the first two items from stack and push the result onto the TOS (Second item / First item and ignore the remainder)
- **GT** Pops two items from the stack and pushes 1 onto TOS if second item is larger otherwise push 0
- **LS** Pops two items from the stack and pushes 1 onto TOS if the second item is smaller than first item otherwise push 0
- **EQ** Pops two items from the stack and pushes 1 onto TOS if they are equal otherwise push 0
- **BZ** `{IL – Instruction Location}` Pop the stack and if the value is 0 then jump to `{IL}`
- **B** `{IL}` Unconditionally jump to `{IL}`
- **LAB** Empty instruction; Provides the instruction location to jump to

**A Sample Source Code**

```c
function main()
    int i, max, sum;
    
    sum = 0;
    i = 1;
    
    scanf(max);
    while (i < max) {
        sum = sum + i;
        i = i + 1;
    }
    endwhile
    printf(sum+max);
```

**Assembly Code Listing** (One possible output)

1. **PI** 0
2. **RM** 102
3. **PI** 1
4. **RM** 100
5. **PS**
6     RM    101
7     LAB
8     PM    100
9     PM    101
10    LS
11    BZ    21
12    PM    102
13    PM    100
14    A
15    RM    102
16    PM    100
17    PI    1
18    A
19    RM    100
20    B     7
21    PM    102
22    PM    101
23    A
24    RS

Symbol Table

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Memory Location</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>100</td>
<td>integer</td>
</tr>
<tr>
<td>max</td>
<td>101</td>
<td>integer</td>
</tr>
<tr>
<td>sum</td>
<td>102</td>
<td>integer</td>
</tr>
</tbody>
</table>

NOTICE: DO NOT CREATE YOUR OWN ASSEMBLY INSTRUCTIONS.
USE ONLY PROVIDED INSTRUCTIONS

Turn in your document according to the instructions given in the project outline.