CSCI 3210 Project 3

Due: see class calendar

Goal: To know how to construct a recursive descent parser.

Description:
In this assignment, you need to implement a recursive descent parser in C++ for the following CFG:

```
1. exps      --> exp | exp NEWLINE exps
2. exp       --> term {addop term}
3. addop     --> + | -
4. term      --> factor {mulop factor}
5. mulop     --> * | /
6. factor    --> ( exp ) | INT
```

The 1st production defines exps as an individual expression, or a sequence expressions separated by NEWLINE token.

The 2nd production describes an expression as a term followed by addop term zero, one, or more times, i.e. exp can be: term, or term addop term, or term addop term addop term, or term addop term addop term addop term, …

The 4th production describes the definition of term, which is pretty much similar to 2nd production.

The 6th production defines a factor either as an expression enclosed by a pair of parentheses or an integer.

In recursive descent parsing, a function is defined for each non-terminal, i.e. exps, exp, term, and factor in our case. The non-terminals addop and mulop are too simple so that we will process them directly in functions of exp and term respectively.

For you convenience, a C sample solution is provided for the CFG. This sample is only for your reference.

Tips:

1. Use provided tokname function to get the name of a given token.
2. Use yylval.ival to get the integer value of the INT token.

How to compile the project?

1. You don't need to build scanner.ll. Just build MainDriver project. .
2. Then, you should be able to run the program. Remember, the program read expressions from test0.txt, which can be found at FlexBison Tools\Resource Files.

What to do in this project?
You only need to work on main.cpp file by providing implementation of three functions: exp, term, and factor. Each function returns an integer as the value of sub-expression that has been parsed/evaluated so far.

If there is a syntax error detected, please throw a runtime exception so that function exps can skip the rest of the expressions and continue the parsing of next expressions. The syntax of throwing a runtime exception is given below:

```
throw runtime_error( a string of error information );
```

In this CFG, you typically can detect errors in factor function.

How to submit?

- Copy the rubric3.doc and your work to a working copy of your repository. Edit the file rubric3.doc to put your name.
- Commit the whole working copy to the server.
- **Any commit of the project after the deadline is considered as cheating. If this happens, the latest version before the deadline will be graded, and you may receive up to 50 points deduction.**
- **No hard copy needed**

Sample C solution for the CFG. Your solution to the problem will be different, especially factor function.

```
<exp> -> <term> { <addop> <term> }
<addop> -> + | -
<term> -> <factor> { <mulop> <factor> }
<mulop> -> *
<factor> -> ( <exp> ) | Number
```

```
int exp(void)
{ int temp = term();
  while ((token=='+')||(token=='-'))
    switch (token) {
    case '+': match('+');
              temp+=term();
              break;
    case '-': match('-');
              temp-=term();
              break;
    }
  return temp;
}
```

```
int factor(void)
{ int temp;
  if (token=='(') {
    match('(');
    temp = exp();
    match(')');
  }
  else if (isdigit(token)) {
    ungetc(token,stdin);
    scanf("%d",&temp);
    token = getchar();
  }
  else error();
  return temp;
}
```

```
void error(void)
{ fprintf(stderr,"Error\n");
  exit(1);
}

void match( char expectedToken)
{ if (token==expectedToken) token = getchar();
  else error();
}
```

```
int term(void)
{ int temp = factor();
  while (token=='*') {
    match('*');
    temp*=factor();
  }
  return temp;
}
```