Programming Assignment 5
(Based on Material By: R. Heckendorn)

Due: Monday, Mar. 26 @ 11:59pm

The problem
The objective of this assignment is to make all errors and warnings have the same format and have syntax errors not stop the syntax analysis phase. There are several parts to this assignment:

- Nice yyerror and error count
- Insert the error tokens into your grammar
- Insert the yyerrok macros into your grammar

nicer errors
yyerror catches all the message that come back from the parser. Here are some examples of syntax error messages that might come into yyerror from the parser if YYERROR_VERBOSE is set:

```
syntax error, unexpected '+'
syntax error, unexpected ELSE
syntax error, unexpected '=', expecting '('
syntax error, unexpected '=', expecting WHILE
syntax error, unexpected ID, expecting '('
syntax error, unexpected ID, expecting WHILE
syntax error, unexpected '+', expecting ',' or ';'
syntax error, unexpected '/', expecting BOOL or INT or VOID
syntax error, unexpected ID, expecting $end or BOOL or INT or VOID
```

We will put in a line number, format the message like in the semantic analysis section, and add some extra info when appropriate. Below is the code that will translate these messages. Use this code or improve on it if you want as long as it doesn't affect the output. Note that the code assumes my names for the tokens. You will need to adjust this for your token names!!

You may use the code in the two posted files `yyerror.cpp` and `yyerror.h` to print out your messages so they conform to a standard style. It is free code! To integrate this code, you must read and understand `yyerror.h`. Yes, I mean it. Your variables may have different names. NOTE: this software includes a tiny sort routine that sorts the expecting tokens to print out a message that can be compared despite the order in which you declare the tokens in your `.y` file.

- line is the line number where the parser is working at the time of the error.
- msg is the extended error message that comes from Bison when the YYERROR_VERBOSE macro is set as in using:
  `#define YYERROR_VERBOSE`
  Use this definition in your code.
- This code makes the brash assumption that `yytext` is relevant for constants.

error count

There will be two global variables counting the number of errors and number of warnings. Warnings will not stop the successful compilation of the program. Keep count of both kinds and report at the end of the compile exactly in the order and format as in this example:

```
Number of warnings: 0
Number of errors: 666
```

other modifications by phase of the compiler
Our compiler now was three phases: lexical analysis, syntax analysis, and semantic analysis. Each phase can produce its own errors. Here is the description of what you need to do:

Lexical analysis: In this phase, each input character that is not part of a legal token is reported as a warning and the character ignored. That is `yylex` does not return a token for these "bad characters" and the lexical analysis continues to run. Example warnings for C-:

```
WARNING(3): Invalid input character: '^'.  Character ignored.
```

These errors will increase the global warning count.

This works easiest in the flex file. Finally, do NOT return a token. This way no token is passed back.

IMPORTANT: if a character is not single character or an escaped single character in single quotes then the single quote will be considered an invalid input character. For example:

```
char z:'dogs and cats';
z = 'oobleck;
```

will generate two invalid input character errors for the first line and one for the second line.

Syntax analysis:
The syntax analysis errors will continue to work as before (except as noted below) but these errors will also increase the global error count.

We will be adding error tokens to our bison grammar to allow the parser to match an error creating legal input and keep running.
We will also add `yyerrok` as needed.
A list of possible edits is provided below.

Semantic analysis:
Only if there are no errors in the syntax analysis then you proceed to semantic analysis.
The errors in semantic analysis will increase the global error count.

At end of compile:
Print the total number of errors and warnings from all phases. For example:

```
Number of warnings: 28
Number of errors: 496
```

or

```
Number of warnings: 0
Number of errors: 0
```

further information on inserting error tokens
We want to add error tokens so syntactic analysis continues past errors. We do this to help the user get as much useful information about their program as we can in one compile.

The file `yyerroredits.txt` (posted with this assignments) contains the places to make edits to your grammar. The lhs symbol is provided as a road map for where the edits go. Some are just there to show there are no edits, for example: factor. term is an example where two rhs productions need to be added. In all cases only the beginning of the actions are shown. Sometimes, the only thing that is added is the addition of the word YYERROK which indicates that yyerrok macro should be added somewhere in your actions. Your token names, of course, may be different. Even some of your productions may be slightly different.

The result of adding these error tokens will be that your compile will now have many shift/reduce conflicts, and reduce/reduce conflicts.

You may have to adjust your grammar a little to get these to fit. On the final grading assignment, you will be graded on the actual error messages you generate, not where you put your error tokens. Try to come as close as you can to my error messages. You may miss 10% of the messages or have some extra and it won't count against you. That is OK. Get the text of the messages as close as possible and as many of my errors as you can.

Example input and output

```
bool a;
int b[100];

int @ max(int x)
{
    int z, zz;
    char c;

    if x>y) z=x 666;
    else z=y;

    while (666>) {
       break;

    z = & zz;

} 12
```

```
WARNING(4): Invalid input character: '@'.  Character ignored.
ERROR(9): Syntax error, unexpected identifier 'x', expecting '('.
ERROR(9): Syntax error, unexpected ')', expecting ';'.
ERROR(9): Syntax error, unexpected identifier 'z', expecting ';'.
ERROR(9): Syntax error, unexpected numeric constant '666', expecting ';'.
ERROR(10): Syntax error, unexpected else.
ERROR(12): Syntax error, unexpected ')'.
ERROR(12): Syntax error, unexpected '{', expecting ')' or or.
WARNING(15): Invalid input character: '&'.  Character ignored.
Number of warnings: 2
Number of errors: 7
```

Submission

You will submit a single uncompressed tar file through Canvas. You can submit as many times as you like. The LAST file you submit BEFORE the deadline will be the one graded.

We will use extensive tests, so thoroughly test your program with inputs of your own.

If you have tests you really think are important or just cool please send them to me and I will consider adding them to the test suite.