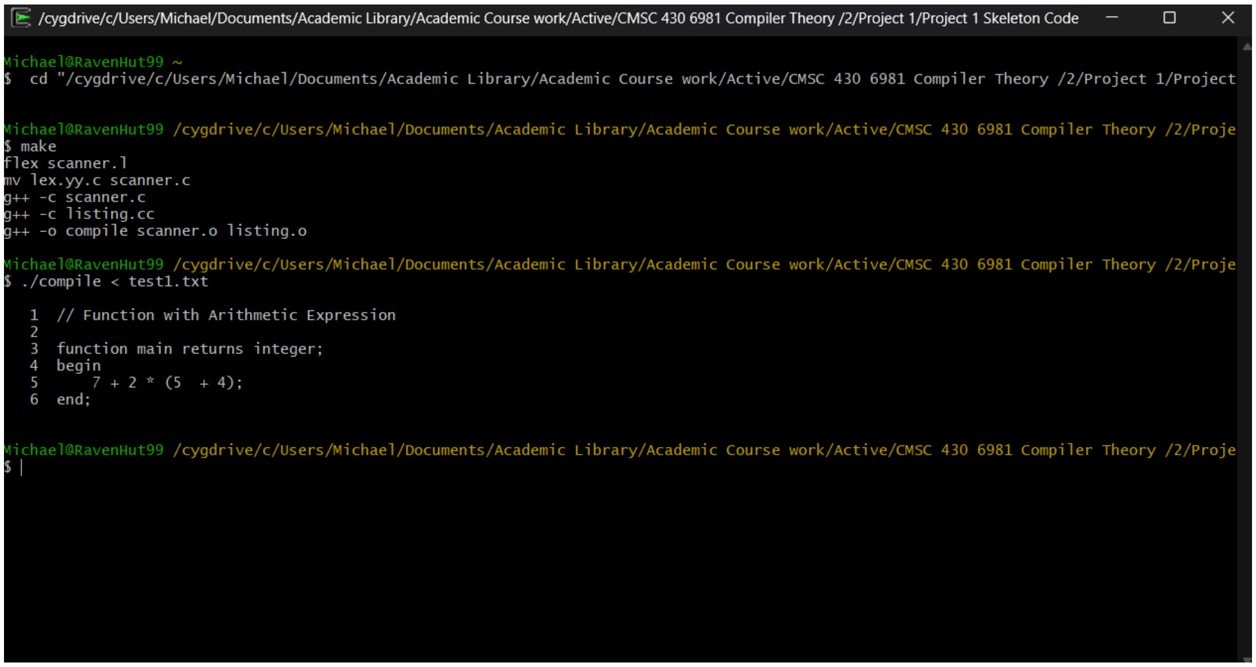
Project 1 - Melbourne Anderson

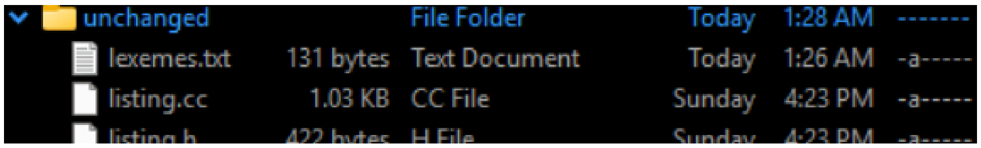
Section - 1

Build and Running the Skeleton



```
/cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Project 1/Project 1 Skeleton Code

Michael@RavenHut99 ~
$  cd "/cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Project 1/Project

Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Proje
$ make
flex scanner.l
mv lex.yy.c scanner.c
g++ -c scanner.c
g++ -c listing.cc
g++ -o compile scanner.o listing.o

Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Proje
$ ./compile < test1.txt

   1  // Function with Arithmetic Expression
   2
   3  function main returns integer;
   4  begin
   5      7 + 2 * (5  + 4);
   6  end;


Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Proje
$ |
```

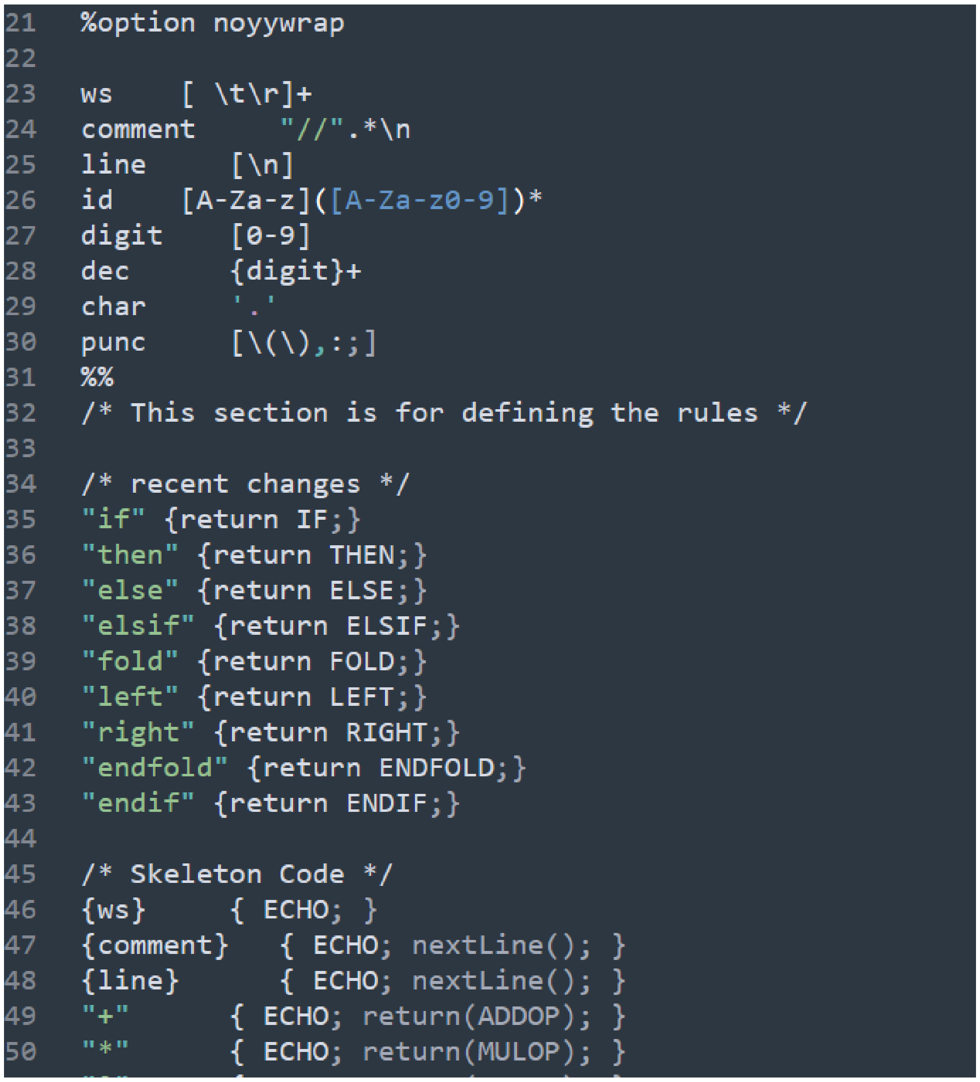
The Test results have been stored in lexemes.txt in the folder "unchanged"



```
unchanged                   File Folder      Today    1:28 AM  -------
    lexemes.txt   131 bytes Text Document    Today    1:26 AM  -a-----
    listing.cc      1.03 KB CC File          Sunday   4:23 PM  -a-----
    listing.h     422 bytes H File           Sunday   4:23 PM  -a-----
```

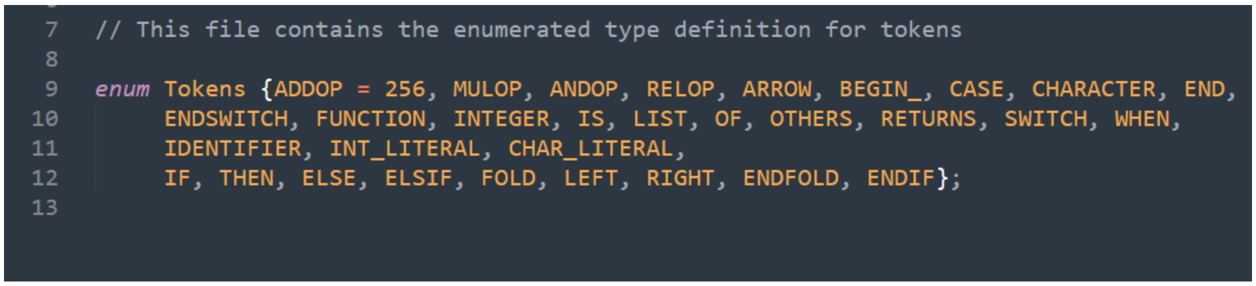
Section 2 — Adding new Reserved Words

Updating the Scanner.l



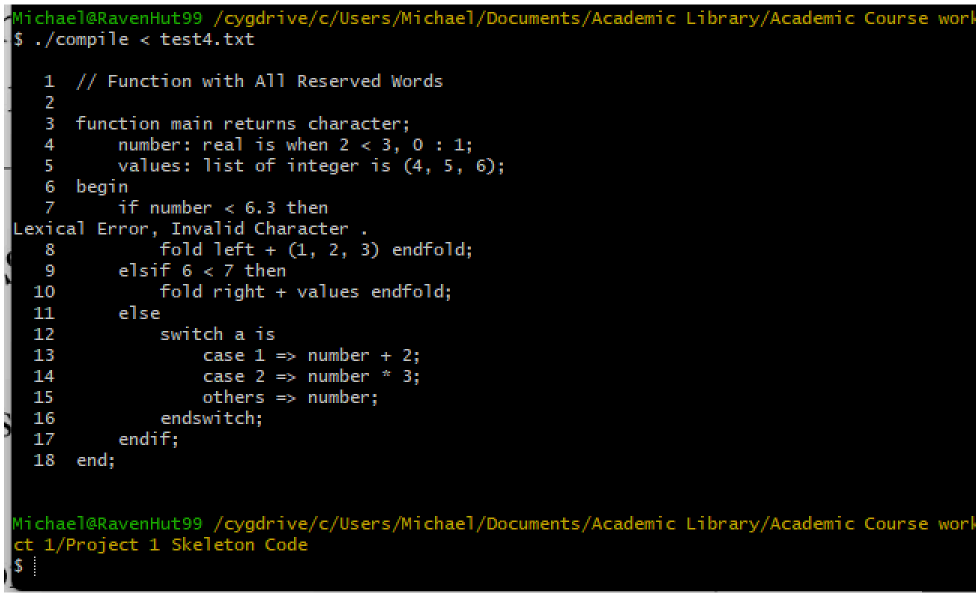
```
%option noyywrap

ws    [ \t\r]+
comment     "//".*\n
line     [\n]
id    [A-Za-z]([A-Za-z0-9])*
digit    [0-9]
dec      {digit}+
char     '.'
punc     [\(\),:;]
%%
/* This section is for defining the rules */

/* recent changes */
"if" {return IF;}
"then" {return THEN;}
"else" {return ELSE;}
"elsif" {return ELSIF;}
"fold" {return FOLD;}
"left" {return LEFT;}
"right" {return RIGHT;}
"endfold" {return ENDFOLD;}
"endif" {return ENDIF;}

/* Skeleton Code */
{ws}     { ECHO; }
{comment}   { ECHO; nextLine(); }
{line}      { ECHO; nextLine(); }
"+"      { ECHO; return(ADDOP); }
"*"      { ECHO; return(MULOP); }
```

Updating enum tokens in tokens.h



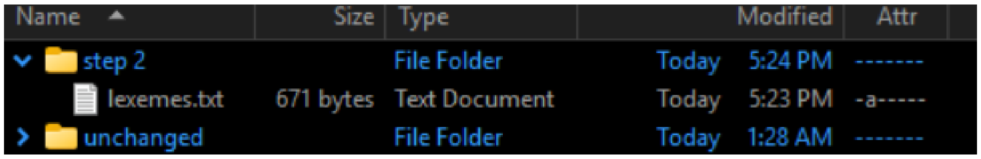
```
// This file contains the enumerated type definition for tokens

enum Tokens {ADDOP = 256, MULOP, ANDOP, RELOP, ARROW, BEGIN_, CASE, CHARACTER, END,
    ENDSWITCH, FUNCTION, INTEGER, IS, LIST, OF, OTHERS, RETURNS, SWITCH, WHEN,
    IDENTIFIER, INT_LITERAL, CHAR_LITERAL,
    IF, THEN, ELSE, ELSIF, FOLD, LEFT, RIGHT, ENDFOLD, ENDIF};

```

Validating the behavior of changes:



```
Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work
$ ./compile < test4.txt

   1  // Function with All Reserved Words
   2
   3  function main returns character;
   4      number: real is when 2 < 3, 0 : 1;
   5      values: list of integer is (4, 5, 6);
   6  begin
   7      if number < 6.3 then
Lexical Error, Invalid Character .
   8          fold left + (1, 2, 3) endfold;
   9      elsif 6 < 7 then
  10          fold right + values endfold;
  11      else
  12          switch a is
  13              case 1 => number + 2;
  14              case 2 => number * 3;
  15              others => number;
  16          endswitch;
  17      endif;
  18  end;


Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work
ct 1/Project 1 Skeleton Code
$
```

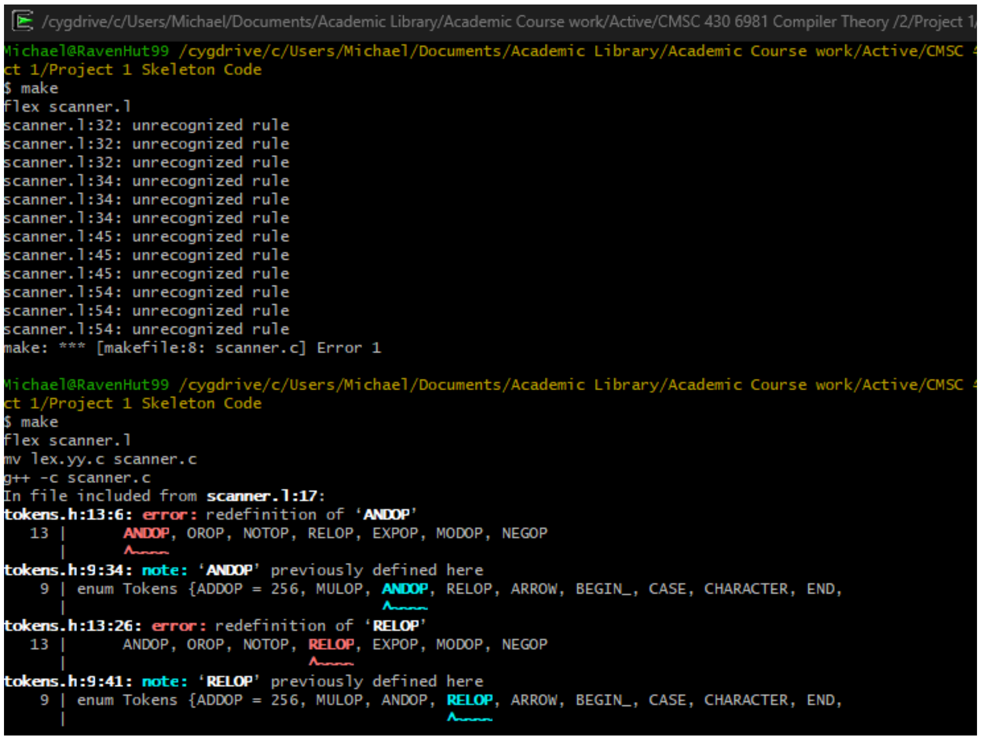
The test results have been stored in folder "Step 2"



Name	Size	Type	Modified		Attr
step 2		File Folder	Today	5:24 PM	-------
lexemes.txt	671 bytes	Text Document	Today	5:23 PM	-a-----
unchanged		File Folder	Today	1:28 AM	-------

Section 3 - Operator Support

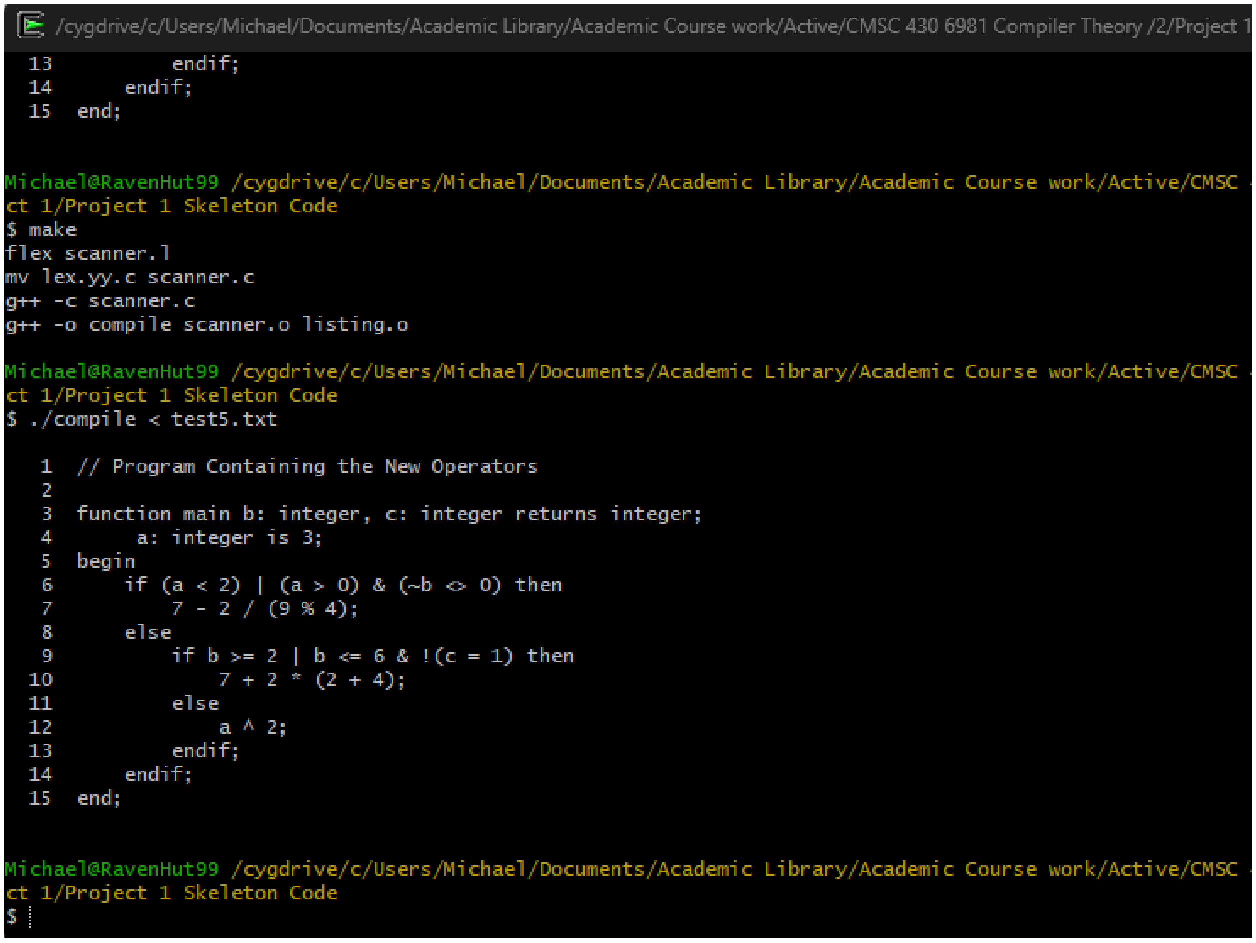
In this section, I found that I was getting Lexical Error. I recompiled the file and identified errors in several areas.



```
/cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Project 1/
Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 4
ct 1/Project 1 Skeleton Code
$ make
flex scanner.l
scanner.l:32: unrecognized rule
scanner.l:32: unrecognized rule
scanner.l:32: unrecognized rule
scanner.l:34: unrecognized rule
scanner.l:34: unrecognized rule
scanner.l:34: unrecognized rule
scanner.l:45: unrecognized rule
scanner.l:45: unrecognized rule
scanner.l:45: unrecognized rule
scanner.l:54: unrecognized rule
scanner.l:54: unrecognized rule
scanner.l:54: unrecognized rule
make: *** [makefile:8: scanner.c] Error 1

Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 4
ct 1/Project 1 Skeleton Code
$ make
flex scanner.l
mv lex.yy.c scanner.c
g++ -c scanner.c
In file included from scanner.l:17:
tokens.h:13:6: error: redefinition of 'ANDOP'
   13 |      ANDOP, OROP, NOTOP, RELOP, EXPOP, MODOP, NEGOP
      |      ^~~~~
tokens.h:9:34: note: 'ANDOP' previously defined here
    9 | enum Tokens {ADDOP = 256, MULOP, ANDOP, RELOP, ARROW, BEGIN_, CASE, CHARACTER, END,
      |                                  ^~~~~
tokens.h:13:26: error: redefinition of 'RELOP'
   13 |      ANDOP, OROP, NOTOP, RELOP, EXPOP, MODOP, NEGOP
      |                          ^~~~~
tokens.h:9:41: note: 'RELOP' previously defined here
    9 | enum Tokens {ADDOP = 256, MULOP, ANDOP, RELOP, ARROW, BEGIN_, CASE, CHARACTER, END,
      |                                         ^~~~~
```

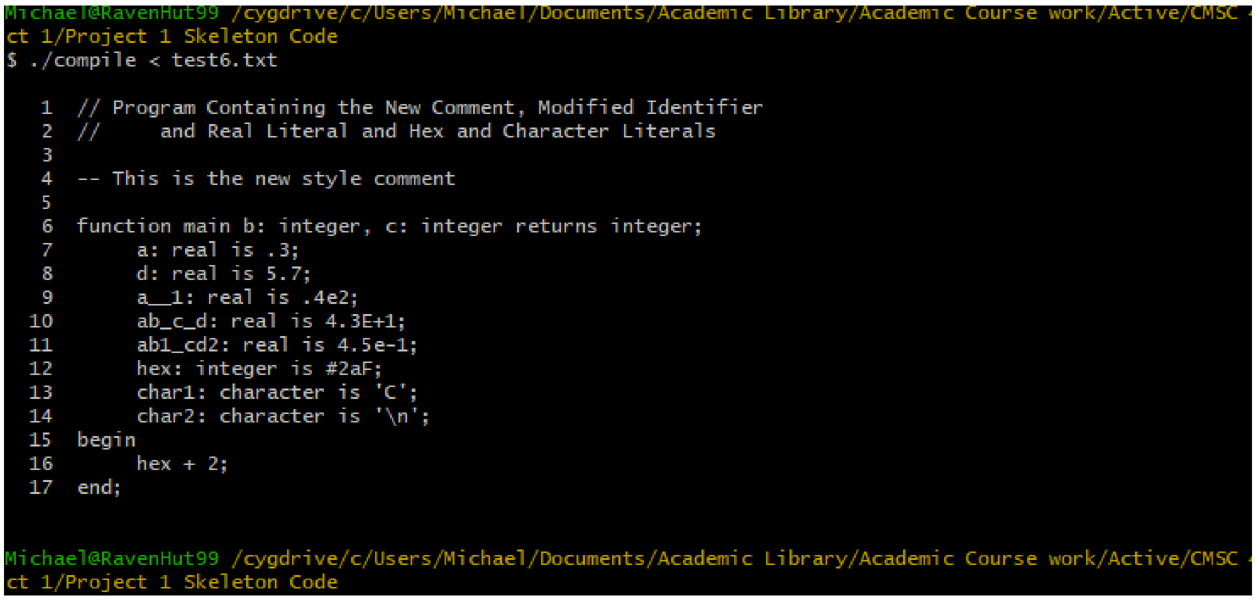
I did the following to :

1. Resolving syntax comment error.
2. Resolve duplicates in tokens.h error.



```
/cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC 430 6981 Compiler Theory /2/Project 1
  13         endif;
  14     endif;
  15  end;


Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC
ct 1/Project 1 Skeleton Code
$ make
flex scanner.l
mv lex.yy.c scanner.c
g++ -c scanner.c
g++ -o compile scanner.o listing.o

Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC
ct 1/Project 1 Skeleton Code
$ ./compile < test5.txt

   1  // Program Containing the New Operators
   2
   3  function main b: integer, c: integer returns integer;
   4       a: integer is 3;
   5  begin
   6      if (a < 2) | (a > 0) & (~b <> 0) then
   7          7 - 2 / (9 % 4);
   8      else
   9          if b >= 2 | b <= 6 & !(c = 1) then
  10              7 + 2 * (2 + 4);
  11          else
  12              a ^ 2;
  13          endif;
  14      endif;
  15  end;


Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC
ct 1/Project 1 Skeleton Code
$
```

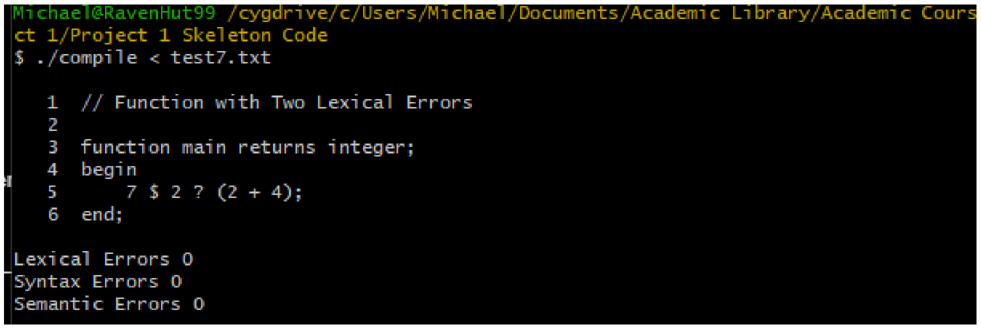
Section 4 - Literal and Identifier Enhancements



```
Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC
ct 1/Project 1 Skeleton Code
$ ./compile < test6.txt

   1  // Program Containing the New Comment, Modified Identifier
   2  //     and Real Literal and Hex and Character Literals
   3
   4  -- This is the new style comment
   5
   6  function main b: integer, c: integer returns integer;
   7       a: real is .3;
   8       d: real is 5.7;
   9       a__1: real is .4e2;
  10       ab_c_d: real is 4.3E+1;
  11       ab1_cd2: real is 4.5e-1;
  12       hex: integer is #2aF;
  13       char1: character is 'C';
  14       char2: character is '\n';
  15  begin
  16       hex + 2;
  17  end;


Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC
ct 1/Project 1 Skeleton Code
```

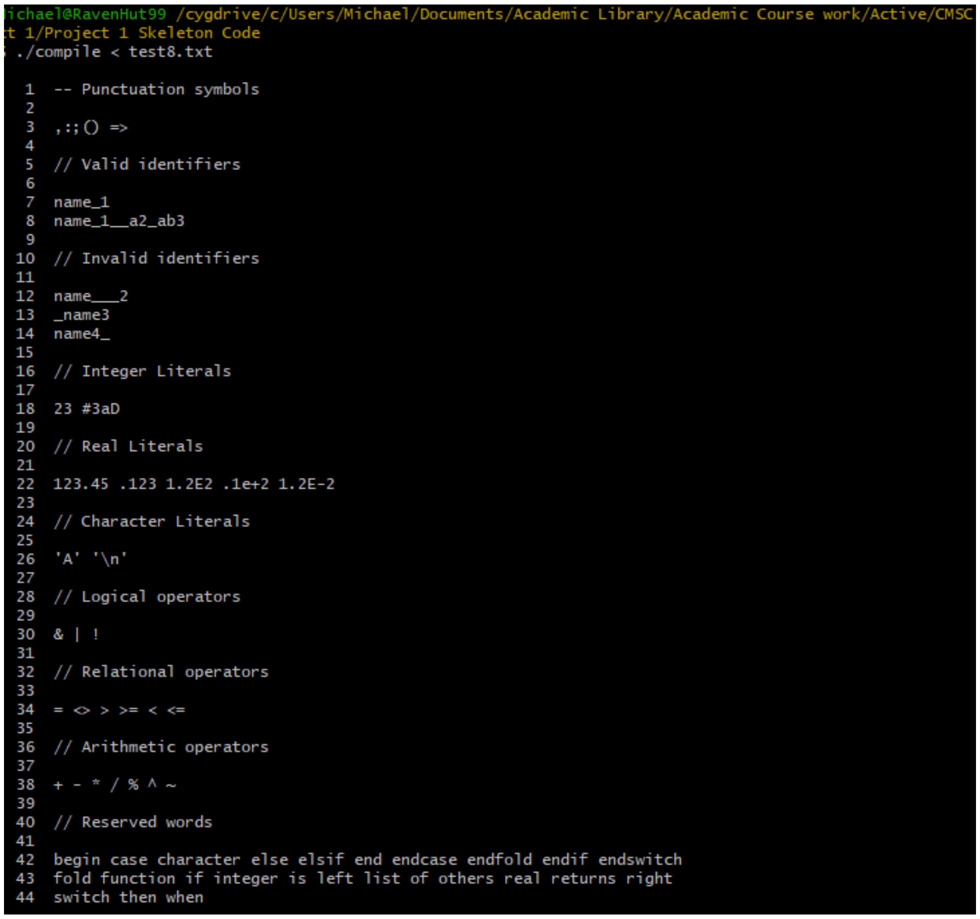
Section 5 - Error Reporting Enhancement

Test 7



```
Michael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Cours
ct 1/Project 1 Skeleton Code
$ ./compile < test7.txt

   1  // Function with Two Lexical Errors
   2
   3  function main returns integer;
   4  begin
   5      7 $ 2 ? (2 + 4);
   6  end;

Lexical Errors 0
Syntax Errors 0
Semantic Errors 0
```

Test 8



```
ichael@RavenHut99 /cygdrive/c/Users/Michael/Documents/Academic Library/Academic Course work/Active/CMSC
t 1/Project 1 Skeleton Code
 ./compile < test8.txt

   1  -- Punctuation symbols
   2
   3  ,:;() =>
   4
   5  // Valid identifiers
   6
   7  name_1
   8  name_1__a2_ab3
   9
  10  // Invalid identifiers
  11
  12  name___2
  13  _name3
  14  name4_
  15
  16  // Integer Literals
  17
  18  23 #3aD
  19
  20  // Real Literals
  21
  22  123.45 .123 1.2E2 .1e+2 1.2E-2
  23
  24  // Character Literals
  25
  26  'A' '\n'
  27
  28  // Logical operators
  29
  30  & | !
  31
  32  // Relational operators
  33
  34  = <> > >= < <=
  35
  36  // Arithmetic operators
  37
  38  + - * / % ^ ~
  39
  40  // Reserved words
  41
  42  begin case character else elsif end endcase endfold endif endswitch
  43  fold function if integer is left list of others real returns right
  44  switch then when
```

Section 6 - lesson learned and Approach

📘 Approach

To complete Project 1, the following structured approach was followed:

1. **Skeleton Build and Verification**
   - Compiled the provided `scanner.l` using the Makefile to establish a working baseline.
   - Ran `test1.txt` through `test3.txt` to understand how lexemes are processed and output into `lexemes.txt`.
2. **Reserved Word Expansion**

- Added new reserved words (e.g., `if`, `then`, `elsif`, `fold`, `endfold`) as literal string matches in `scanner.l`.
- Updated `tokens.h` with corresponding token enums to ensure successful compilation.

3. **Operator Support**
    - Implemented additional logical (`&`, `|`, `!`), relational (`=`, `<`, `>`, `<>`, `>=`, `<=`), and arithmetic (`^`, `%`, `~`) operators.
    - Placed multi-character operators before single-character ones to avoid early pattern matching.
4. **Advanced Lexeme Support**
    - Defined rules for:
        - Real literals (e.g., `.3`, `4.5e-1`)
        - Hexadecimal integers (`#2AF`)
        - Enhanced character literals (`'A'`, `'\n'`)
        - Valid identifiers with internal underscores and error rules for invalid identifiers.
5. **Error Reporting Enhancements**
    - Modified `listing.cc` to store multiple errors per line using `vector<string>`.
    - Tracked and displayed lexical, syntax, and semantic error counts separately.
    - Verified results using `test7.txt` and `test8.txt`.

🧠 Lessons Learned

1. **Lexical Matching Priority Matters**
    - Flex matches rules top-down, so **more specific patterns must be listed before general ones** to avoid unintended matches (e.g., `>=` vs `>`).
2. **Literal vs Pattern Matching**

- Reserved words and operators are best matched as **literal strings** (`"if"`, `"^"`) while identifiers and numbers require **regex macros** for flexibility.

3. **Macro Definitions Must Match Usage**
    - Any macro used in `{macro}` format must be **defined before** `%%`. Otherwise, Flex will throw an **"undefined definition"** error.
4. **Multiple Errors per Line Requires a Vector**
    - Using a `string` variable only allows for one error message per line. A `vector<string>` allows storage and display of all errors on a given line.
5. **C++ Naming Conflicts Can Break Compilation**
    - Naming a variable `messages` can conflict with `std::messages`. Always use unique names (e.g., `errorMessages`) to avoid ambiguity in complex systems.
6. **The Role of Each File is Distinct**
    - `scanner.l` defines what tokens are recognized.
    - `tokens.h` gives meaning to those tokens via enum values.
    - `listing.cc` manages how errors are tracked and reported.
    - Understanding how these interact is crucial to successful project completion.
7. **Testing Iteratively Prevents Late Surprises**
    - Running tests incrementally (`test4.txt` through `test8.txt`) helped isolate bugs and validate grammar rules step-by-step.