RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSC310A
Course Name	COMPILERS
Programme	B.TECH
Department	CSE
Faculty	FET

Name of the Student	SUDHANSHU SHEKHAR
Reg. No	17ETCS002213
Semester/Year	6SEM/2017
Course Leader/s	MS SUVIDHA

<table>
<tr><th colspan="4">Declaration Sheet</th></tr>
<tr><td>Student Name</td><td colspan="3">SUDHANSHU SHEKHAR</td></tr>
<tr><td>Reg. No</td><td colspan="3">17ETCS002213</td></tr>
<tr><td>Programme</td><td>B.TECH</td><td>Semester/Year</td><td>6</td></tr>
<tr><td>Course Code</td><td colspan="3">CSC310A</td></tr>
<tr><td>Course Title</td><td colspan="3">COMPILERS</td></tr>
<tr><td>Course Date</td><td></td><td>to</td><td></td></tr>
<tr><td>Course Leader</td><td colspan="3">MS SUVIDHA</td></tr>
</table>

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

<table>
<tr><td>Signature of the Student</td><td></td><td>Date</td><td></td></tr>
<tr><td>Submission date stamp
(by Examination & Assessment Section)</td><td colspan="3"></td></tr>
</table>

Signature of the Course Leader and date	Signature of the Reviewer and date

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech in Computer Science and Engineering
Semester/Batch	06th /2017		
Course Code	CSC310A	Course Title	Compilers
Course Leader	Mr. Hari Krishna S. M. & Ms. Suvidha		

Assignment					
Register No.	**17ETCS002213**	**Name of the student**	Sudhanshu shekhar		
Section		**Marking Scheme**	**Marks**		
			Max Marks	**First Examiner Marks**	**Moderator**
Part A					
	A 1.1	Identification and grouping of Tokens	04		
	A 1.2	Implementation in *Lex*	02		
	A 1.3	Design of Context Free Grammar	04		
	A 1.4	Implementation in *Yacc*	06		
	A 1.5	Results and Comments	04		
		Part-A 1 Max Marks	**20**		
		Total Assignment Marks	**20**		

Course Marks Tabulation				
Component- CET B Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
A.1				
Marks (out of 20)				

PART – A

A 1.1 Identification and grouping of Tokens

Tokens are sequences of characters that have a certain meaning or represent a particular group. Lexemes are a sequence of characters in the source code that is matched by the pattern for a token.
In the given problem, the set of tokens are:

- Reserved words:

```
Enter the exp: int main () {
int a;
int b;
scanf("%d",&b);
int c[100];
int s=0;
if(a>b)
s=a+b;
else
s=a-b;
int i;
for(i=0;i<3;i++)
s=s+i;
printf("s = ",s);
}
Input accepted.
```

LEXEMES	TOKENS
Int	Data Type
main	Reserved Keywords
(	Separators
)	Separators
{	Separators
Int	Data Type
a	Identifier
;	Separators
Int	Data Type
b	Identifier
;	Separators
scanf	I/O function
(	Separators
“%d”	Statement
,	Separators
&b	Statement
)	Separators
;	Separators
Int	Data Type
c	Identifier
[	Separators
100	Operator
]	Separators
;	Separators
Int	Data Type
s	Identifier
=	Operator
0	Identifier
;	Separators

If	Keyword
(	Separators
a	Identifier
>	Operator
b	Identifier
)	Separators
s	Identifier
a	Identifier
+	Operator
b	Identifier
;	Separators
Else	Keyword
s	Identifier
=	Operator
a	Identifier
-	Operator
b	Identifier
;	Separators
Int	Data Type
I	Identifier
;	Separators
For	Keyword
(	Separators
I	Identifier
=	Operator
0	Identifier
;	Separators
I	Identifier
<	Operator
3	Identifier
;	Separators
s	Identifier
=	Operator
s	Identifier
+	Operator
i	Identifier
;	Separators
Printf	I/O function
(	Separators
s	Identifier

=	Operator
,	Separators
s	Identifier
)	Separators
;	Separators
}	Separators

A1.2 Implementation in Lex

Lex is a computer program that generates lexical analyzers. Lex reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language. It returns tokes which act as input to the yacc program which is responsible for the parsing

The code for the lex is as follows:

```
%{
#include "TEMP.tab.h"
%}

alpha [a-zA-Z]
digit [0-9]
%%
"main"                  {return MAIN;}
"("                     {return LB;};
")"                     {return RB;}
"{"                     {return LCB;}
"}"                     {return RCB;}
","                     {return comma;}
```

```
"%"                         {return percent;}
"&"                         {return and;}
"void"                      {return VOID;}
\"[^\"\n]*\"                        {return DQ;}
\"[^\'\n]*\"                        {return SQ;}
"printf"                    {return PRINT;}
"scanf"                     {return INPUT;}
"+"                         {return PLUS;}
"-"                         {return MINUS;}
"*"                         {return STAR;}
"/"                         {return DIVIDE;}
"int"                       { return INT;}
"char"                      { return CHAR;}
";"                         { return semicolon;}
"="                         { return EQ;}
" "                         { return space;}
{digit}+                            {return num;}
{alpha}({alpha}|{digit})*           {return var_id;}
[!@#$%^&*<>,?a-zA-Z0-9]      {return character;}
[\n]                        { return NL;}
"if"                        {return IF;}
"else"                      {return ELSE;}
"for"                       {return FOR;}
"while"                     {return WHILE;}
"++"                        {return INC;}
"--"                        {return DNC;}
">="                        {return GE;}
"<="                        {return LE;}
"=="                        {return EEQ;}
"&&"                        {return AND;}
">"                         {return GT;}
">"                         {return LT;}
"||"                        {return OR;}
"!"                         {return NOT;}
"!="                        {return NE;}
"break"                     {return BREAK;}
"["                         {return SRB;}
"]"                         {return SLB;}
[!@#$%^&*<>,?a-zA-Z0-9]*      {return print_char;}
"return(0);"                {return RETURN;}
.                           {ECHO; yyerror("unexpected character");}
%%
```

Fig 1.1: The above image shows the screenshot of a lex file with the extension -. l

A1.3 Design of Context Free Grammar for each function

The context free gramma for each function:

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

Minimum two data types:

The data types that are chosen is int and char. The grammar should be able declare variables of the above-mentioned data types, it has to allow variable assignment and allow expressions involving the variables.

```
declarations:   declarations dec
                | dec
                ;

dec:    INT space var_id semicolon NL
        | CHAR space var_id semicolon NL
        |INT space var_id space EQ space num semicolon NL
        | CHAR space var_id space SQ character SQ semicolon NL
        ;
```

Minimum two control statements:

The control statements that are accepted in the program are if-else, if and break statement.

```
control_stmt: IF RB condition LB NL RCB NL exp NL LCB NL
                | BREAK semicolon NL
                ;

condition: var_id LE var_id
          | var_id GE var_id
          | var_id EEQ var_id
          | var_id AND var_id
          | var_id NE var_id
          | var_id OR var_id
          | NOT var_id
          ;

exp: var_id space EQ space var_id space PLUS space var_id semicolon
        |var_id space EQ space var_id space MINUS space var_id semicolon
        |var_id space EQ space var_id space STAR space var_id semicolon
        |var_id space EQ space var_id space DIVIDE space var_id semicolon
        ;
```

Minimum two looping statements:

The loops used here are for loop and while loop.

```
loop_stmt: FOR RB var_id EQ num semicolon cond semicolon indc LB NL RCB NL exp NL LCB NL print_stmt
          | WHILE RB cond LB RCB NL exp NL indc NL LCB NL print_stmt
          ;

cond: var_id LE var_id
        | var_id GE var_id
        | var_id NE var_id
        ;

indc: var_id INC
   | var_id INC semicolon NL
   | var_id DNC
   | var_id DNC semicolon NL
   ;
```

exp: var_id space EQ space var_id space PLUS space var_id semicolon
|var_id space EQ space var_id space MINUS space var_id semicolon
|var_id space EQ space var_id space STAR space var_id semicolon
|var_id space EQ space var_id space DIVIDE space var_id semicolon
;

Input-output functions:
The standard input and output statements like printf and scanf are being implemented.
input_stmt: INPUT RB DQ percent var_id DQ comma and var_id LB semicolon NL
;
print_stmt: PRINT RB DQ print_char DQ semicolon LB NL
| PRINT RB DQ print_char DQ comma var_id LB semicolon NL
;

Compound statements and two dimentional arrays:
Any statement inside curly braces are called compound statements. Here all the above statements are compound statements.

One dimentional array_declaration:
dec: INT space var_id SRB num SLB semicolon NL
|CHAR space var_id SRB num SLB semicolon NL
;
Two dimentional array declaration:
dec: INT space var_id SRB num SLB SRB num SLB semicolon NL
|CHAR space var_id SRB num SLB SRB num SLB semicolon NL
;

A1.4 Implementation in YACC

The implementation of yacc is as follows:

```
%{
void yyerror (char *s);
#include <stdio.h>
#include <stdlib.h>
%}

%start S
%token NL INT CHAR character comma RCB LCB space RB LB MAIN VOID semicolon var_id EQ PLUS MINUS STAR DIVIDE RETURN GT LT
%token SQ PRINT INPUT and percent DQ num IF ELSE SRB SLB INC DNC LE EEQ AND OR NOT RE NE GE FOR WHILE BREAK print_char
%right EQ
%left  AND OR
%left  LE GE EQ NE PLUS MINUS STAR DIVIDE GT LT

%%
S:      prototype
        ;

prototype: VOID space MAIN RB LB NL RCB NL body RETURN NL LCB
           |INT space MAIN RB LB NL RCB NL body RETURN NL LCB
           ;

body:   declarations statements
        ;

declarations:   declarations dec
                | dec
                ;

dec:    INT space var_id semicolon NL
        |CHAR space var_id semicolon NL
        |INT space var_id space EQ space num semicolon NL
        |CHAR space var_id space SQ character SQ semicolon NL
        |INT space var_id SRB num SLB  semicolon NL
        |CHAR space var_id SRB num SLB  semicolon NL
        |INT space var_id SRB num SLB SRB num SLB semicolon NL
        |CHAR space var_id SRB num SLB SRB num SLB  semicolon NL
        ;
```

Fig 1.1: Source code

```
statements: statements input_stmt control_stmt NL
          ;

input_stmt: INPUT RB DQ percent var_id DQ comma and var_id LB semicolon NL
          ;

control_stmt: IF RB condition LB NL RCB NL exp NL LCB NL loop_stmt
                |IF RB condition space logical_op space condition LB NL RCB NL exp NL LCB NL loop_stmt
                | BREAK semicolon NL loop_stmt
                ;

condition: var_id LE var_id
           | var_id GE var_id
           | var_id EEQ var_id
           | var_id NE var_id

logical_op:     | AND
                | OR
                | NOT
                ;

exp: var_id space EQ space var_id space PLUS space var_id semicolon
        |var_id space EQ space var_id space MINUS space var_id semicolon
        |var_id space EQ space var_id space STAR space var_id semicolon
        |var_id space EQ space var_id space DIVIDE space var_id semicolon
        ;

loop_stmt: FOR RB var_id EQ num semicolon cond semicolon indc LB NL RCB NL exp NL LCB NL print_stmt
           | WHILE RB cond LB RCB NL exp NL indc NL LCB NL print_stmt
           ;

cond: var_id LE var_id
        | var_id GE var_id
        | var_id NE var_id
        ;
```

Fig 1.2: Source code

```
indc: var_id INC
     | var_id INC semicolon NL
     | var_id DNC
     | var_id DNC semicolon NL
     ;

print_stmt: PRINT RB DQ print_char DQ semicolon LB NL
          | PRINT RB DQ print_char DQ comma var_id LB semicolon NL
          ;

%%

int main(){
printf("Enter the exp: ");
if (yyparse() == 0)
printf("accepted");
else{
printf("NOT accepted");
return(0);
}
}

int yywrap(){
return 1;
}

void yyerror (char *s) {fprintf (stderr, "%s\n", s);}
```

Fig 1.3: Source code

A1.5 Results and comments

Execution of the above program is as follows:

```
C:\Users\sudhanshu shekhar>cd C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>lex compiler.l

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>yacc -d compiler.y

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>gcc lex.yy.c compiler.tab.c
compiler.tab.c: In function 'yyparse':
compiler.tab.c:1602:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'?
-declaration]
 1602 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
compiler.y: At top level:
compiler.y:149:1: warning: return type defaults to 'int' [-Wimplicit-int]
  149 | main()
      | ^~~~

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>a.exe
```

```
Enter the exp: int main () {
int a;
int b;
scanf("%d",&b);
int c[100];
int s=0;
if(a>b)
s=a+b;
else
s=a-b;
int i;
for(i=0;i<3;i++)
s=s+i;
printf("s = ",s);
}
Input accepted.
```

Fig : Results, the code typed in the above figure does not have any syntax error or it is according to the grammar defined, so the input is accepted


```
C:\Users\sudhanshu shekhar>cd C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>lex compiler.l

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>yacc -d compiler.y

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>gcc lex.yy.c compiler.tab.c
compiler.tab.c: In function 'yyparse':
compiler.tab.c:1602:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'?
-declaration]
 1602 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
compiler.y: At top level:
compiler.y:149:1: warning: return type defaults to 'int' [-Wimplicit-int]
  149 | main()
      | ^~~~

C:\Users\sudhanshu shekhar\OneDrive\Desktop\compiler>a.exe
```

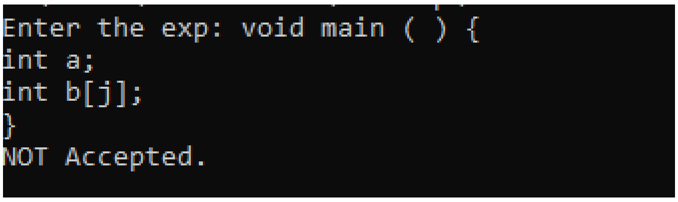

Fig : Results, the code typed in the above figure does have a syntax error in the third line, so the input is not accepted

Comments:

Yacc reads the grammar descriptions in yacc file and generates a syntax analyzer (parser), that includes function yyparse, in file y.tab.c. Included in file lab8.y are token declarations. The –d option causes yacc to generate definitions for tokens and place them in file y.tab.h. Lex reads the pattern descriptions in the lex file, includes file y.tab.h, and generates a lexical analyzer, that includes function yylex, in file lex.yy.c. Finally, the lexer and parser are compiled and linked together to create executable a.exe. From main we call yyparse to run the compiler. Function yyparse automatically calls yylex to obtain each token.

Limitations:

The above program works only for simple statements and does not work for nested statements. Nested if statements and nested looping statements are not supported by the program. The program should be in a particular order as per the grammar