## Part B - Compiler Design

| Sl. No. | Questions | CO | PO |
|---|---|---|---|
| 1. | Write a C / C++ program to accept a C program and perform error detection& correction for the following:<br>a) Check for un-terminated string constant and single character constant in the input C program. i.e A string constant begins with double quotes and extends to more than one line.<br>b) Report the error line numbers and the corrective actions to user. | CO1 | 1,2,3 & 4 |
| 2. | Write a C / C++ program to accept a C program and perform error detection & correction, indicate the user for the following:<br>a) Check whether the multi-line comment statement is terminated correctly or not.<br>b) Check whether the single line comment statement is existing in your C program and report the line numbers to the user. | CO1 | 1,2,3 & 4 |
| 3. | Write a Lex program to accept a C program and perform error detection& correction for the following:<br>a) Check for valid arithmetic and relational expressions in the input C program<br>b) Recognize increment and decrement operations also.<br>c) Report the errors in the statements' to user. | CO1 | 1,2,3 & 4 |
| 4. | Write a Lex program to accept a C program and perform the following error detection & correction:<br>a) Check the validity of "*structure*" declarative statements in your program.<br>b) Indicate the invalid statements along with their line numbers to users. | CO1 | 1,2,3 & 4 |
| 5. | Write a Lex program to accept a C program and perform the following error detection & correction:<br>a) Check for the valid "*if ....else if...else*" statement in the input C program.<br>b) Report the errors to users. | CO1 | 1,2,3 & 4 |
| 6. | Write Yacc and Lex programs to accept an arithmetic expression and perform the following error detection:<br>a) Check the validity of the "*arithmetic expressions*" in the input C statement.<br>b) Report the errors in the statements to user.<br>c) Evaluate the arithmetic expression.<br>d) Recognize increment and decrement operators involved in the expressions. | CO2& 3 | 1,2,3 & 4 |
| 7. | Write Yacc and Lex programs to accept a declarative statement and perform the following error detection:<br>a) Check the validity of the "*declarative*" statement.<br>b) Recognize array declarations of any dimension.<br>c) Report the errors to users. | CO2& 3 | 1,2,3 & 4 |
| 8. | Write Yacc and Lex programs to accept a relational expression and perform the following error detection:<br>a) Check the validity of the "*relational*" expression and evaluate the expression.<br>Note: Relational expression can have arithmetic expressions embedded in it. | CO2& 3 | 1,2,3 & 4 |

|  |  |  |  |
|---|---|---|---|
| 9. | Write Yacc and Lex programs to accept a logical expression and perform the following error detection:<br>a) Check for the validity of the logical expression and evaluate it.<br>Note: Logical expression can have relational and arithmetic expressions with in it. | CO2& 3 | 1,2,3 & 4 |
| 10. | Write Yacc and Lex programs for the following grammar:<br>a) Test the executable code of Yacc program by giving valid and invalid strings as input.<br>*Grammar :*<br>S SS+ \| SS*\| (S) \|a | CO2& 3 | 1,2,3 & 4 |
| 11. | Write Yacc and Lex programs for the following grammar:<br>a) Test the executable code of Yacc program by giving valid and invalid strings as input.<br>*Grammar :*<br>S L=R \| R<br>L *R \| id \|num<br>R L | CO2& 3 | 1,2,3 & 4 |
| 12. | Write Yacc and Lex programs for the following grammar:<br>a)Test the executable code of Yacc program by giving valid and invalid strings as input.<br>*Grammar :*<br>D TL<br>T int \| float \| long int \| double \| static int \| register int<br>L L,id \| id | CO2& 3 | 1,2,3 & 4 |

1. **Write a C / C++ program to accept a C program and perform error detection& correction for the following:**

a) **Check for un-terminated string constant and single character constant in the input C program. i.e A string constant begins with double quotes and extends to more than one line.**

b) **Report the error line numbers and the corrective actions to user.**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
        int lineno=0,open=0,close=0,string=0,strcheck,i;
        char line[100];
        FILE *fp;
        fp = fopen("file1.txt","r");
        while(fgets(line,sizeof(line),fp)!=NULL)
        {
                lineno++;
                open=0;
                close=strcheck=string=0;
        for(i = 0;i<strlen(line);i++)
        {
            if(line[i]=='"')
            {
                string = 1;
                if(open==1&&close==0)
                    close=1;
                else if(open==0&&close==0)
                    open=1;
                else if(open==1&&close==1)
                    close=0;
            }
        }
        if(open==1&&close==0)
        {
            printf("Unterminated string at line = %d\n",lineno);
            strcheck=1;
        }
        else if(string==1&&strcheck==0)
            printf("String usage validated at line = %d\n",lineno);
    }
    return 0;
}
```

**2. Write a C / C++ program to accept a C program and perform error detection & correction, indicate the user for the following:**

**c) Check whether the multi-line comment statement is terminated correctly or not.**

**d) Check whether the single line comment statement is existing in your C program and report the line numbers to the user.**

```
#include<stdio.h>
#include<string.h>
int main()
{
    int i,open,close,comment,commentcheck,lineno=0,openline=0,closeline=0;
    char line[100];
    FILE *fp;
    fp = fopen("file1.txt","r");
    while(fgets(line,sizeof(line),fp)!=NULL)
    {
        lineno++;
        comment=commentcheck=0;
        if(open==1&&close==0)
            printf("%s\n",line);
        if(strstr(line,"/*")&&open==0)
        {
            open =1;
            close=0;
            comment=1;
            openline=lineno;
            printf("Comment started at line = %d\n",lineno);
        }
        if(strstr(line,”//”))
            printf(“Single line comment identified at line no: %d\n”,lineno);
        if(strstr(line,"*/")&&open==1&&close==0)
        {
            if(open==1&&close==0)
            {
                close=1;
                open = 0;
                closeline=lineno;
                printf("Comment closed at line = %d\n",closeline);
            }
        }
    }
    if(open==1&&close==0)
    {
        printf("Comment not ended\nOpened at line = %d\n",openline);
        commentcheck=1;
    }
    else if(comment==1&&commentcheck==0)
    {
        printf("Closed\n");
    }
    return 0;
}
```

**3. Write a Lex program to accept a C program and perform error detection& correction for the following:**

- **e) Check for valid arithmetic and relational expressions in the input C program**
- **f) Recognize increment and decrement operations also.**
- **g) Report the errors in the statements’ to user.**

```
%{
    #include <stdio.h>
    int c=0;
    FILE *fp;
%}

id [a-zA-Z][a-zA-Z0-9]*
ar [/*+-]
num [0-9]+
rel [<>=!]=?
```

```
inc "++"
dec "--"

%%
\n {c++;}
{id}"="({id}|{num})({ar}({id}|{num}))+ {printf("\nValid arithmetic operation in line %d
",c+1);ECHO;}
{id}"="({id}|{num}){ar} {printf("\nInvalid operation ! No right operand for the arithmetic operation %d
",c+1);ECHO;}
{id}"="{ar}({id}|{num}) {printf("\nInvalid operation ! No Left operand for the arithmetic operation %d
",c+1);ECHO;}
{id}"="({id}|{num}){rel}({id}|{num}) {printf("\nValid relational operation in line %d ",c+1);ECHO;}
{id}"="{id}({inc}|{dec}) {printf("\nValid Unary operation %d ",c+1);ECHO;}
.|\n ;
%%

int main()
{
yyin=fopen("sample.c","r");
yylex();
fclose(yyin);
}
```

sample.c

```
#include<sdtio.h>
void main(){
        a=s+t;
        b=+6
        f=g+
        a=a<b;
        a=b++;
}
```

**4. Write a Lex program to accept a C program and perform the following error detection & correction:**

- **h) Check the validity of “*structure*” declarative statements in your program.**
- **i) Indicate the invalid statements along with their line numbers to users.**

```
%{
        #include <stdio.h>
        int c=0;
        FILE *fp;
%}
id [a-zA-Z][a-zA-Z0-9]*
num [0-9]+\
types "int"|"float"|"char"
dec {types}" "{id}(","{id})*
%%
\n {c++;}
"struct"" "{id}"{"({dec}";"|(\n))*"}"({id}(","{id})*)?";" {printf("TESTING");ECHO;}
.|\n ;
%%
int main()
{
	yyin=fopen("sample.c","r");
	yylex();
	fclose(yyin);
}
```

(or)

```
%{
	#include <stdio.h>
	int c=0;
	FILE *fp;
%}
id [a-zA-Z][a-zA-Z0-9]*
num [0-9]+\
types "int"|"float"|"char"
dec {types}" "{id}(","{id})*
%%
\n {c++;}
"struct"" "{id}"{"({dec}";"|(\n))*"}"({id}(","{id})*)?";" {printf("Valid declaration at lin no:
%d\n",c);ECHO;printf("\n");}
"struct"" "{id}" "({dec}";"|(\n))*"}"({id})?";" {printf("Opening braces of structure missing at line no:
%d\n",c);ECHO;printf("\n");}
"struct"" "{id}"{"({dec}";"|(\n))*({id})?";" {printf("Closing braces of structure missing at line no:
%d\n",c);ECHO;printf("\n");}
.|\n ;
%%
int main()
{
	yyin=fopen("sample.c","r");
	yylex();
	fclose(yyin);
}
```

sample.c

```
#include<sdtio.h>
void main(){
	struct s{int c;}d;
```

```
        struct t int c;}f;
        struct g{int k;h;
        }
```

**5. Write a Lex program to accept a C program and perform the following error detection & correction:**

- **j) Check for the valid “*if ….else if…else*” statement in the input C program.**
- **k) Report the errors to users.**

```
%{
        #include <stdio.h>
        int c=0;
        FILE *fp;
%}

id [a-zA-Z][a-zA-Z0-9]*
num [0-9]+
rel [<>=!]=?


%%
\n {c++;}
"if""("({id}|{num})({rel}({id}|{num}))*")""{".*"}"(\n)*("else
if""("({id}|{num})({rel}({id}|{num}))*")""{".*"}")*(\n)*("else""{".*"}")? {printf("Found an if
statement !  ");ECHO;}
("else if""("({id}|{num})({rel}({id}|{num}))*")""{".*"}")*(\n)*("else""{".*"}")? {printf("No preceding
if statement before if else!  ");ECHO;}
.|\n ;
%%

int main()
{
        yyin=fopen("sample.c","r");
        yylex();
        fclose(yyin);
}
```

sample.c

```
#include<sdtio.h>
void main(){
        if(a<b){printf("a\n");}else if(b<c){printf("b\n");}else{printf("c\n");}
        else if(c!=d){printf("d\n");}
        if(v==1){printf("v\n");}
}
```

**6. Write Yacc and Lex programs to accept an arithmetic expression and perform the following error detection:**
**a) Check the validity of the “*arithmetic expressions”* in the input C statement.**
**b) Report the errors in the statements to user.**
**c) Evaluate the arithmetic expression.**
**d) Recognize increment and decrement operators involved in the expressions.**

YACC

```
%{
#include <stdio.h>
int res=0;
%}

%token id num

%%
stmt:expr {res=$$;};
expr: expr '+"+' {printf("\n++ sign detected");exit(0);}
        |expr '-"-' {printf("\n-- sign detected");exit(0);}
        |expr '+' expr {$$=$1+$3;printf("\n+ sign detected");}
        |expr '-' expr {$$=$1-$3;printf("\n- sign detected");}
        |expr '*' expr {$$=$1*$3;printf("\n* sign detected");}
        |expr '/' expr {$$=$1/$3;printf("\n/ sign detected");}
        |expr '*' {printf("\nError no right operand!");exit(0);}
        |'(' expr ')' {$$=$2;printf("\nbrackets detected");}
        |id
        |num
        ;
%%

void main()
{
        printf("\nEnter the expression : ");
        yyparse();
        printf("\nThe result is %d",res);
        exit(0);
}

void yyerror()
{
        printf("Invalid\n");
        exit(0);
}
```

Lex

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "y.tab.h"
    extern int yylval;
    int val = 0;
%}

%%
[a-zA-Z][a-zA-Z0-9]* {printf(“Enter the value of: %s\n”.yytext);scanf(“%d”,&val);yylval=val;return
id;}
[0-9]+ {yylval=atoi(yytext);return num;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**7. Write Yacc and Lex programs to accept a declarative statement and perform the following error detection:**

**a) Check the validity of the “*declarative”* statement.**

**b) Recognize array declarations of any dimension.**

**c) Report the errors to users.**

YACC

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    int res;
%}

%token id num type

%%
stmt:expr {res=1;}
    ;
expr:type ' ' id ex2
    |type ' ' id arr
    ;
ex2:',' id ex2
    |';'
    ;
arr:'[' num ']' arr
    |'[' num {printf("Array bracket not closed!");exit(0);}
    |';'
```

```
        ;
%%

int main()
{
        printf("Enter a declaration statement : ");
        yyparse();
        printf("Success!");
        return 0;
}

void yyerror()
{
        printf("Error!!");
        exit(0);
}
```

Lex

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        #include "y.tab.h"
        extern yylval;
%}

%%
("int"|"float"|"char"|"double") {return type;}
[a-zA-Z][a-zA-Z0-9]* {return id;}
[0-9]+ {yylval=atoi(yytext);return num;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**8. Write Yacc and Lex programs to accept a relational expression and perform the following error detection:**
**a) Check the validity of the “*relational”* expression and evaluate the expression.**
**Note: Relational expression can have arithmetic expressions embedded in it.**

YACC

```
%{
        #include <stdio.h>
```

```
    #include <stdlib.h>
    int res;
%}

%token id num

%%
stmt:expr{res=$$;};
expr:expr '+' expr {$$=$1+$3;}
    |expr '-' expr {$$=$1-$3;}
    |expr '*' expr {$$=$1*$3;}
    |expr '/' expr {$$=$1/$3;}
    |expr '<' expr {$$=($1<$3);}
    |expr '>' expr {$$=($1>$3);}
    |expr '<' '=' expr {$$=($1<=$4);}
    |expr '>' '=' expr {$$=($1>=$4);}
    |expr '=' '=' expr {$$=($1==$4);}
    |'(' expr ')'{$$=$2;}
    |id
    |num
    ;
%%

int main()
{
    printf("Enter an expression : ");
    yyparse();
    printf("\nThe result is : %d",res);
    return 0;
}

int yyerror()
{
    printf("Error!");
    exit(0);
}
```

Lex

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "y.tab.h"
    extern yylval;
    int val=0;
%}
```

```
%%
[a-zA-Z][a-zA-Z0-9]* {printf(“Enter the value of: %s\n”,yytext);scanf(“%d”,&val);yylval=val;return
id;}
[0-9]+ {yylval=atoi(yytext);return num;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**9. Write Yacc and Lex programs to accept a logical expression and perform the following error detection:**
**a) Check for the validity of the logical expression and evaluate it.**
**Note: Logical expression can have relational and arithmetic expressions with in it.**

YACC

```
%{
	#include <stdio.h>
	#include <stdlib.h>
	int res;
%}

%token id num

%%
stmt:expr{res=$$;};
expr:expr '+' expr {$$=$1+$3;}
	|expr '-' expr {$$=$1-$3;}
	|expr '*' expr {$$=$1*$3;}
	|expr '/' expr {$$=$1/$3;}
	|expr '<' expr {$$=($1<$3);}
	|expr '>' expr {$$=($1>$3);}
	|expr '<' '=' expr {$$=($1<=$4);}
	|expr '>' '=' expr {$$=($1>=$4);}
	|expr '=' '=' expr {$$=($1==$4);}
	|expr '&' '&' expr {$$=($1&&$4);}
	|expr '|' '|' expr {$$=($1||$4);}
	|'(' expr ')'{$$=$2;}
	|'!' expr {$$=!$2;}
	|id
	|num
	;
%%

int main()
{
        printf("Enter an expression : ");
        yyparse();
        printf("\nThe result is : %d",res);
        return 0;
}

int yyerror()
{
        printf("Error!");
        exit(0);
}
```

Lex

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        #include "y.tab.h"
        extern yylval;
        int val=0;
%}

%%
[a-zA-Z][a-zA-Z0-9]* {printf("Enter the value of %s\n",yytext);scanf("%d",&val);yylval=val;return
id;}
[0-9]+ {yylval=atoi(yytext);return num;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**10. Write Yacc and Lex programs for the following grammar:**
**a) Test the executable code of Yacc program by giving valid and invalid strings as input.**
***Grammar :***
**S   SS+ | SS*| (S) |a**

YACC

```
%{
        #include <stdio.h>
        #include <stdlib.h>
%}

%token id
```

```
%%
stmt:expr {printf("\nValid!\n");}
        ;
expr:expr expr '+'
        |expr expr '*'
        |'(' expr ')'
        |id
        ;
%%

int main()
{
        printf("Enter a string as input : ");
        yyparse();
        return 0;
}

void yyerror()
{
        printf("\nInvalid");
        exit(0);
}
```

Lex

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        #include "y.tab.h"

%}

%%
[a] {return id;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**11. Write Yacc and Lex programs for the following grammar:**
**a) Test the executable code of Yacc program by giving valid and invalid strings as input.**
***Grammar :***
**S   L=R | R**
**L    *R | id  |num**
**R   L**

YACC

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        int res=0;
%}

%token id num

%%
stmt:S {res=1;}
        ;
S:L '=' R
 |R
 ;
L:'*' R
 |id
 |num
 ;
R:L;
%%

int main()
{
        printf("\nENTER STRING  : ");
        yyparse();
        if(res==1)
        {
        printf("\nValid\n");
        }
        return 0;
}

void yyerror()
{
        printf("\nInvalid!\n");
        exit(0);
}
```

Lex

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        #include "y.tab.h"
        extern yylval;
%}

%%
[a-zA-Z][a-zA-Z0-9]* {return id;}
[0-9]+ {yylval=atoi(yytext);return num;}
```

```
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```

**12. Write Yacc and Lex programs for the following grammar:**
**a)Test the executable code of Yacc program by giving valid and invalid strings as input.**
***Grammar :***
**D   TL**
**T   int | float | long int | double | static int | register int**
**L   L,id | id**

YACC

```
%{
        #include <stdio.h>
        #include <stdlib.h>
        int res=0;
%}

%token id type
%%
stmt:D {res=1;}
        ;
D:T ' ' L
        ;
T:type
        ;
L:L ',' id
 |id
 ;
%%

int main()
{
        printf("\nEnter a string : ");
        yyparse();
        if(res==1)
        {
        printf("\nValid\n");
        }
        return 0;
}

void yyerror()
{
        printf("\nInvalid !!\n");
        exit(0);
}
```

Lex

```
%{
	#include <stdio.h>
	#include <stdlib.h>
	#include "y.tab.h"
%}

%%
("int"|"double"|"float"|"static int"|"long int"|"register int") {return type;}
[a-zA-Z][a-zA-Z0-9]* {return id;}
[\t] {;}
[\n] {return 0;}
. {return yytext[0];}
%%
```