Practical No: 8

Name : Patel Savankumar P.
Enroll No : 19BCE519
Subject : Compiler Construction

AIM: To implement a Type Checker.

File 1: practical8.y

```
%{

        #include <stdio.h>
        #include <string.h>
        #include <stdlib.h>

        typedef struct variable{
                char name[63];
                char type[63];
                struct variable *next;
        }variable;

        variable* symbolTable;


        int insert(char* name, char* type);
        variable* search(char* input);
        void display();
        void freeTable();
```

```
        void init();

%}

%union
{
        char *string;
        struct variable *node;
}

//Tokens
%token INT FLOAT CHAR DOUBLE EXIT
%token PLUS MINUS MUL DIV MOD EQUAL
%token OPENPAREN CLOSEPAREN
%token SEMICOLON COMMA
%token <node> NUMBER
%token <string> ID
%type <node> E Expression Assignment


%left PLUS MINUS
%left MUL DIV MOD
%left OPENPAREN CLOSEPAREN
/*Rule Section*/
%%

Statement :     StartDec SEMICOLON Statement
                | Expression SEMICOLON Statement
                | Assignment SEMICOLON Statement
                | EXIT
                ;

StartDec        : INT ID1 {display();}
                | FLOAT ID2 {display();}
                | CHAR ID3 {display();}
                | DOUBLE ID4 {display();}

ID1 :   ID1 COMMA ID{insert(strdup($3),strdup("int"));}
        |ID{insert(strdup($1),strdup("int"));}
```

```
        |ID1 COMMA ID EQUAL Expression{
                                insert(strdup($3),strdup("int"));
                        }
        |ID EQUAL Expression{
                                insert(strdup($1),strdup("int"));
                        }
        ;

ID2 :   ID2 COMMA ID{insert(strdup($3),strdup("float"));}
        |ID{insert(strdup($1),strdup("float"));}
        ;

ID3 :ID3 COMMA ID{insert(strdup($3),strdup("char"));}
        |ID{insert(strdup($1),strdup("char"));}
        ;

ID4 :ID4 COMMA ID{insert(strdup($3),strdup("double"));}
        |ID{insert(strdup($1),strdup("double"));}
;

Expression:E {printf("Type Checking Done, Type Valid\n");}
;

E:      E PLUS E {
                        if(strcmp($1->type,$3->type) == 0)
                                $$ = $1;
                        else
                                printf("Type Error %s is %s and %s is
%s\n",$1->name,$1->type,$3->name,$3->type);
                    }
        |E MINUS E {
                        if(strcmp($1->type,$3->type) == 0)
                                $$ = $1;
                        else
                                printf("Type Error %s is %s and %s is
%s\n",$1->name,$1->type,$3->name,$3->type);
                    }
         |E MUL E {
                        if(strcmp($1->type,$3->type) == 0)
```

```
                                $$ = $1;
                        else
                                printf("Type Error %s is %s and %s is
%s\n",$1->name,$1->type,$3->name,$3->type);
                    }
        |E DIV E {
                        if(strcmp($1->type,$3->type) == 0)
                                $$ = $1;
                        else
                                printf("Type Error %s is %s and %s is
%s\n",$1->name,$1->type,$3->name,$3->type);
                    }

        |E MOD E{
                        if(strcmp($1->type,$3->type) == 0)
                                $$ = $1;
                        else
                                printf("Type Error %s is %s and %s is
%s\n",$1->name,$1->type,$3->name,$3->type);
                }

        |OPENPAREN E CLOSEPAREN {
                                        $$=$2;
                                }

        |NUMBER {
                        insert(strdup("Constant"),strdup("Number"));
                        variable* result = search($1);
                        $$ = result;
                }

        |ID     {
                        variable* result = search($1);
                        if(result != NULL)
                        {
                                $$ = result;
                        }
                        else
                                printf("Variable is not Declared\n");
```

```
                }
        ;

Assignment: ID EQUAL Expression {
                variable* result = search($1);
                if(result != NULL){
                        if(strcmp(result->type,$3->type) == 0)
                                $$ = $1;
                        else
                        printf("Type Error %s is %s and %s is
%s\n",result->name,result->type,$3->name,$3->type);
                }
                else
                        printf("Variable is not Declared\n");
        }
;
%%


//Insert into Symbol Table
int insert(char* name, char* type)
{
        variable* result = search(name);
        if(result == NULL)
        {
                //Allocate Memory
                variable* new = (variable*) malloc(sizeof(variable));
                strcpy(new->name,name);
                strcpy(new->type,type);
                new->next = NULL;

                if(symbolTable == NULL)
                {
                        symbolTable = new;
                }
                else
                {
                        new->next = symbolTable;
                        symbolTable = new;
```

```
                }
                return 1;
        }
        else
        {
               return 0;
        }

}

// Search
variable* search(char* input)
{
        variable* temp = symbolTable;

        while(temp != NULL)
        {
                if(strcmp(temp->name,input) == 0)
                        return temp;
                temp = temp->next;
        }
        return NULL;
}


// Display
void display()
{
        variable* temp = symbolTable;
        printf("\n%10s %10s\n","Name","Type");
        while(temp != NULL)
        {
                printf("%10s %10s\n",temp->name,temp->type);
                temp = temp->next;
        }
}

void freeTable()
{
```

```
        variable* temp = symbolTable;
        variable* freeVar;
        while(temp != NULL)
        {
                freeVar = temp;
                temp = temp->next;
                printf("Freeing %s\n",freeVar->name);
                free(freeVar);
        }
}

void init()
{
        symbolTable =  NULL;
}

void yyerror(const char *str)
{
        printf("\nSyntax Error - Freeing Symbol Table\n");
        freeTable();
}

int yywrap()
{
        return 0;
}

main(int argc,char* argv[])
{
   init();
   yyparse();
}
```

File 2: Practical8.l

```
%{
    /* Definition Section*/
    /*Lex Definition for Variables*/
    #include "y.tab.h"
%}

identifier [a-zA-Z_][a-zA-Z0-9_]*
number [0-9]+
%%

"int" {return INT;}

"exit" {return EXIT;}

"float" {return FLOAT;}

"char" {return CHAR;}

"double" {return DOUBLE;}

"+" {return PLUS;}

"-" {return MINUS;}

"*" {return MUL;}

"/" {return DIV;}

"%" {return MOD;}

"(" {return OPENPAREN;}

")" {return CLOSEPAREN;}

"=" {return EQUAL;}

{number} {return NUMBER;}
```

```
{identifier} {yylval.string=strdup(yytext);
                return ID;}

; return SEMICOLON;

, return COMMA;



\n                      /* ignore end of line */;
\r                      /* ignore end of line */;
[ \t]+                  /* ignore whitespace */;

%%
```

Execution Sequence:

```
E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_8>flex practical8.l

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_8>bison -dy practical8.y
bison: cannot open file `practical8.y': No such file or directory

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_8>gcc lex.yy.c y.tab.c -w

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_8>a.exe
```

Output:

```
E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_8>a.exe
int a=10;
Type Checking Done, Type Valid

      Name        Type
         a         int
  Constant     Number
int b=15;
Type Checking Done, Type Valid

      Name        Type
         b         int
         a         int
  Constant     Number
c=20;
Type Checking Done, Type Valid
Variable is not Declared
```

Conclusion:

From this practical I learned how to write Yacc and Lex code to do type checking.