Assignment no:8

Write a program using LEX and YACC to generate Intermediate code in the form of Three addresss and Quadruple form for assignment statement

Pmcd45.l

```
%{

#include"y.tab.h"

#include"stdio.h"

#include"string.h"

int lineno=1;

%}


number [0-9]+|([0-9]*\.[0-9]+)

identifier [a-zA-Z][a-zA-Z0-9]*

%%


{identifier} {strcpy(yylval.var,yytext);

             return VAR;}


{number} {strcpy(yylval.var,yytext);

        return NUM;}


\n lineno++;


[\t ] ;
```

```
. {return yytext[0];}
%%
```

pmcd45.y

```
%{
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int i=0,index1=0,tindex=0;
void addqruple(char op[5],char arg1[10],char arg2[10],char result[10]);
int yylex();
int yyerror();
struct q
{
    char op[5];
    char arg1[10];
    char arg2[10];
    char result[10];
}q[30];

%}

%union
{
  char var[10];
}


%token <var>NUM VAR

%type <var>EXPR ASSIGNMENT

%left'-''+'

%left'*''/'

%nonassoc UMINUS

%left '('')'


%%


ASSIGNMENT:VAR'='EXPR {

          strcpy(q[index1].op,"=");

          strcpy(q[index1].arg1,$3);

          strcpy(q[index1].arg2,"");

          strcpy(q[index1].result,$1);

          strcpy($$,q[index1++].result);

                    }

          ;

EXPR:EXPR'+'EXPR {addqruple("+",$1,$3,$$);}

    |EXPR'-'EXPR {addqruple("-",$1,$3,$$);}

    |EXPR'*'EXPR {addqruple("*",$1,$3,$$);}

    |EXPR'/'EXPR {addqruple("/",$1,$3,$$);}

    |'('EXPR')'EXPR {strcpy($$,$2);}
    |'-'EXPR {addqruple("uminus",$2,"",$$);}
    |VAR
    |NUM
    ;

%%

int main()
{


yyparse();
printf("\n\nthree address code");
for(i=0;i<index1;i++)
{
 printf("\n %s\t %c\t %s\t %s\t %s\n",q[i].result, '=', q[i].arg1,q[i].op,q[i].arg2);
}
printf("\n\nINDEX\t OP\t ARG1\t ARG2\t RESULT");
for(i=0;i<index1;i++)
{
 printf("\n%d\t %s\t %s\t %s\t %s\n",i,q[i].op,q[i].arg1,q[i].arg2,q[i].result);
}
return 0;
}
void addqruple(char op[5],char arg1[10],char arg2[10],char result[10])
```

```
 {
 strcpy(q[index1].op,op);
 strcpy(q[index1].arg1,arg1);
 strcpy(q[index1].arg2,arg2);
 sprintf(q[index1].result,"t%d",tindex++);
 strcpy(result,q[index1++].result);
 }

int yyerror()
{
printf("syntax error");
}
int yywrap()
{
return 1;
}
```

```
Output: lex pmcd45.l
yacc -d pmcd45.y
cc lex.yy.c y.tab.c
./a.out
```

Explanation: ### Lex File (`Pmcd45.l`)

- **Includes:** Includes header files and declares a global variable `lineno` for line numbers.

- **Patterns and Actions:**

  - **Identifiers (`identifier`):** Copies the text to `yylval.var` and returns `VAR`.

  - **Numbers (`number`):** Copies the text to `yylval.var` and returns `NUM`.

  - **Newlines (`\n`):** Increments line number.

  - **Whitespace:** Ignores tabs and spaces.

  - **Others:** Returns the character itself.

Yacc File (`pmcd45.y`)

- **Includes and Declarations:**

  - Includes necessary headers and declares functions and variables.

  - Defines a structure `q` for quadruples.

- **Union and Tokens:** Defines a union for variables and tokens for numbers and variables.

- **Operator Precedence:** Sets precedence and associativity for operators.

- **Grammar Rules:**

  - **ASSIGNMENT:** Parses assignment statements and generates quadruple code.

  - **EXPR:** Parses expressions, handling arithmetic operations and generating intermediate code.

- **Functions:**

  - **`main`:** Parses input and prints the three-address and quadruple codes.

  - **`addqruple`:** Adds a quadruple to the list.

  - **`yyerror`:** Handles syntax errors.

  - **`yywrap`:** Indicates end of input.

Execution Flow

1. **Lexical Analysis:** Tokenizes input into numbers and variables.

2. **Parsing:**

   - **Assignment:** Matches `VAR = EXPR`.

   - **Expression:** Handles arithmetic and nested expressions.

3. **Intermediate Code Generation:**

   - **Three-Address Code:** Stores operations in `q` structure.

   - **Quadruple Form:** Prints each operation with its arguments and result.

Output

- **Three-Address Code:** Lists operations as `result = arg1 op arg2`.

- **Quadruple Form:** Displays the operation, arguments, and results in a structured format.

Sure, let's break it down:

Intermediate Code: It's a simplified version of the source code, used by compilers during translation. It's easier to work with and optimize compared to the original code.

Three-Address Code: This is a specific type of intermediate code that represents instructions with at most three operands. For example, `x = y + z` would be represented as `x = y + z`, where `x`, `y`, and `z` are operands, and `+` is an operator.

Quadruple Form: This is another type of intermediate representation where each instruction is represented by four parts: an operator, two operands, and a result. For instance, `x = y + z` might be represented as `(+, y, z, x)`.

In essence, these forms help simplify and organize the translation process, making it easier to generate machine code from high-level source code.