**Sharvan Ram Kumaran**

**CSE-C**

**185001143**

**Exercise 7:**

**Aim:**

Generate Intermediate code in the form of Three Address Code sequence for the sample input program written using declaration, conditional and assignment statements in new language **Pascal-2021**, Following is the sample input

**Code:**

Tac.l

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "y.tab.h"

%}

term ([a-zA-Z\\_][a-zA-Z\\_0-9]\*)

num ([0-9]+)

real {num}\.{num}

relop ("<"|"<="|">"|">="|"=="|"!=")

addop ("+"|"-")

mulop ("\*"|"/"|"%")

spl (";"|","|"{"|"}"|"("|")"|"="|"&"|"|"|"!"|":")

%%

"begin" {return BGN;}

"end" {return END;}

"if" {return IF;}

"then" {return THEN;}

"else" {return ELSE;}

"integer" {return INT;}

"char" {return CHAR;}

"real" {return REALVAR;}

['].['] {yylval.ch = yytext[1]; return CHCONST;}

{term} {yylval.str = strdup(yytext); return VAR;}

{real} {yylval.floatval = atof(yytext); return REAL;}

{num} {yylval.intval = atoi(yytext); return NUM;}

{relop} {yylval.str = strdup(yytext); return RELOP;}

{mulop} {yylval.str = strdup(yytext); return MULOP;}

{addop} {yylval.str = strdup(yytext); return ADDOP;}

{spl} {return \*yytext;}

[ \t\n]+ {;}

. {char errmsg[100];

strcpy(errmsg, "Invalid Character: ");

strcat(errmsg, yytext);

strcat(errmsg, "\n");

yyerror(errmsg);}

%%

Tac.y

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

int yylex(void);

int yyerror(char \*);

int yywrap();

int vars = 0, labels = 0;

struct info{

char \*var;

char \*code;

int intval;

float floatval;

char charval;

};

typedef struct info node;

node \*makeNode(){

//creating a new node to store intermediate code

node \*n = (node \*)malloc(sizeof(node));

n->intval = 0;

n->floatval = 0;

n->charval = 0;

n->var = (char \*)malloc(50 \* sizeof(char));

n->code = (char \*)malloc(5000 \* sizeof(char));

return n;

}

%}

/\*Declaration of tokens and precedence\*/

%token BGN END IF THEN ELSE INT CHAR REALVAR

%token REAL CHCONST VAR NUM RELOP ADDOP MULOP

/\*Increasing precedence\*/

%right MULOP

%left ADDOP

/\*Declaration of the types that YYSTYPE can take with the union\*/

%union{

int intval;

float floatval;

char ch;

char \*str;

struct info \*Node;

}

/\*Declaring types for the tokens\*/

%type<str> VAR RELOP ADDOP MULOP

%type<intval> NUM

%type<floatval> REAL

%type<ch> CHCONST

%type<Node> Program Structure Declarations Statements

%type<Node> Declaration Type Value Statement

%type<Node> Assignment Conditional Condition Expr

%type<Node> E T F

%%

Program : Structure{

printf("\nL%-5d - |\n%s", 0, $$->code);

}

;

Structure : Declarations BGN Statements END{

sprintf($$->code, "%s%10s\n%s", $1->code, "|", $3->code);

}

;

Declarations : Declaration Declarations{

$$ = makeNode();

sprintf($$->code, "%s%s", $1->code, $2->code);

}

| Declaration{

$$ = $1;

}

;

Declaration : VAR ':' Type ';' {

$$ = makeNode();

sprintf($$->code, "%10s %-5s := %s\n", "|", $1, $3->var);

}

| VAR ':' Type '=' Value ';'{

$$ = makeNode();

sprintf($$->code, "%10s %-5s := %s\n", "|", $1, $5->var);

}

;

Type : INT{

$$ = makeNode();

$$->intval = 0;

sprintf($$->var, "%d", 0);

sprintf($$->code, "");

}

| REALVAR{

$$ = makeNode();

$$->floatval = 0.0;

sprintf($$->var, "%.2f", 0.0);

sprintf($$->code, "");

}

| CHAR{

$$ = makeNode();

$$->charval = 0;

sprintf($$->var, "%s", "NULL");

sprintf($$->code, "");

}

;

Value : NUM{

$$ = makeNode();

$$->intval = $1;

sprintf($$->var, "%d", $1);

sprintf($$->code, "");

}

| REAL{

$$ = makeNode();

$$->floatval = $1;

sprintf($$->var, "%.2f", $1);

sprintf($$->code, "");

}

| CHCONST{

$$ = makeNode();

$$->charval = $1;

sprintf($$->var, "%c", $1);

sprintf($$->code, "");

}

;

Statements : Statement Statements{

$$ = makeNode();

sprintf($$->code, "%s%s", $1->code, $2->code);

}

| Statement{

$$ = $1;

}

;

Statement : Assignment {

$$ = $1;

}

| Conditional{

$$ = $1;

}

;

Assignment : VAR '=' Expr ';'{

$$ = makeNode();

char tac[100];

sprintf($$->var, "%s", $1);

sprintf(tac, "%10s %-5s := %s\n", "|", $$->var, $3->var);

sprintf($$->code, "%s%s", $3->code, tac);

}

;

Expr : E{

$$ = $1;

}

;

E : T MULOP E{

$$ = makeNode();

char tac[100];

sprintf($$->var, "x%d", ++vars);

sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->var, $1->var, $2, $3->var);

sprintf($$->code, "%s%s%s", $1->code, $3->code, tac);

}

| T{

$$ = $1;

}

| F{

$$ = $1;

}

;

T : T ADDOP F{

$$ = makeNode();

char tac[100];

sprintf($$->var, "x%d", ++vars);

sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->var, $1->var, $2, $3->var);

sprintf($$->code, "%s%s%s", $1->code, $3->code, tac);

}

| F{

$$ = $1;

}

;

F : VAR{

$$ = makeNode();

sprintf($$->var, "%s", $1);

sprintf($$->code, "");

}

| NUM{

$$ = makeNode();

$$->intval = $1;

sprintf($$->var, "%d", $1);

sprintf($$->code, "");

}

| REAL{

$$ = makeNode();

$$->floatval = $1;

sprintf($$->var, "%.2f", $1);

sprintf($$->code, "");

}

| CHCONST{

$$ = makeNode();

$$->charval = $1;

sprintf($$->var, "'%c'", $1);

sprintf($$->code, "");

}

;

Conditional : IF '(' Condition ')' THEN Statements ELSE Statements END IF{

$$ = makeNode();

int condnBlock = ++labels;

int endBlock = ++labels;

sprintf($$->code, "%s%10s if %s then goto L%d\n%s%10s goto L%d\n%10s\nL%-5d - |\n%s%10s\nL%-5d - |\n", $3->code, "|", $3->var, condnBlock, $8->code, "|", endBlock, "|", condnBlock, $6->code, "|", endBlock);

}

;

Condition : Expr RELOP Expr{

$$ = makeNode();

char tac[100];

sprintf($$->var, "%s%s%s", $1->var, $2, $3->var);

sprintf($$->code, "%s%s", $1->code, $3->code);

}

;

%%

int yyerror(char\* str){

printf("\n%s", str);

return 0;

}

int yywrap(){

return 1;

}

int main(){

printf("\n\t\tIntermediate Code Generation\n");

printf("\n\nThree Address Code:\n");

yyparse();

return 0;

}

Input.txt

i: integer=5;

a: real=4.2;

b: char='c';

c: integer=7;

x: integer;

begin

if (i>2) then

x=a-b/c;

else

x=a+b\*c;

end if

end

**Output:**

**Text

Description automatically generated**

**Learning Objective:**

* Learned about Three Address Code and it’s generation
* Learned about union of datatypes

**Result:**

Intermediate code generation using Lex and Yacc has been implemented