Practical No: 9

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

AIM: To implement Assembly code generator.

File 1:practical9.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "practical9.h"

/* prototypes */
nodeType *opr(int oper, int nops, ...);
nodeType *id(int i);
nodeType *con(int value);
void freeNode(nodeType *p);
int ex(nodeType *p);
int yylex(void);

void yyerror(char *s);
int sym[26];                    /* symbol table */
%}

%union {
    int iValue;                 /* integer value */
    char sIndex;                /* symbol table index */
    nodeType *nPtr;             /* node pointer */
```

```
};

%token <iValue> INTEGER
%token <sIndex> VARIABLE
%token WHILE IF PRINT
%nonassoc IFX
%nonassoc ELSE

%left GE LE EQ NE '>' '<'
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS

%type <nPtr> stmt expr stmt_list

%%

program:
        function                { exit(0); }
        ;

function:
          function stmt         { ex($2); freeNode($2); }
        | /* NULL */
        ;

stmt:
          ';'                           { $$ = opr(';', 2, NULL, NULL);
}
        | expr ';'                      { $$ = $1; }
        | PRINT expr ';'                { $$ = opr(PRINT, 1, $2); }
        | VARIABLE '=' expr ';'         { $$ = opr('=', 2, id($1), $3);
}
        | WHILE '(' expr ')' stmt       { $$ = opr(WHILE, 2, $3, $5); }
        | IF '(' expr ')' stmt %prec IFX { $$ = opr(IF, 2, $3, $5); }
        | IF '(' expr ')' stmt ELSE stmt { $$ = opr(IF, 3, $3, $5, $7);
}
        | '{' stmt_list '}'             { $$ = $2; }
        ;
```

```
stmt_list:
          stmt                  { $$ = $1; }
        | stmt_list stmt        { $$ = opr(';', 2, $1, $2); }
        ;

expr:
          INTEGER               { $$ = con($1); }
        | VARIABLE              { $$ = id($1); }
        | '-' expr %prec UMINUS { $$ = opr(UMINUS, 1, $2); }
        | expr '+' expr         { $$ = opr('+', 2, $1, $3); }
        | expr '-' expr         { $$ = opr('-', 2, $1, $3); }
        | expr '*' expr         { $$ = opr('*', 2, $1, $3); }
        | expr '/' expr         { $$ = opr('/', 2, $1, $3); }
        | expr '<' expr         { $$ = opr('<', 2, $1, $3); }
        | expr '>' expr         { $$ = opr('>', 2, $1, $3); }
        | expr GE expr          { $$ = opr(GE, 2, $1, $3); }
        | expr LE expr          { $$ = opr(LE, 2, $1, $3); }
        | expr NE expr          { $$ = opr(NE, 2, $1, $3); }
        | expr EQ expr          { $$ = opr(EQ, 2, $1, $3); }
        | '(' expr ')'          { $$ = $2; }
        ;

%%

nodeType *con(int value) {
    nodeType *p;

    /* allocate node */
    if ((p = malloc(sizeof(nodeType))) == NULL)
        yyerror("out of memory");

    /* copy information */
    p->type = typeCon;
    p->con.value = value;

    return p;
}
```

```c
nodeType *id(int i) {
    nodeType *p;

    /* allocate node */
    if ((p = malloc(sizeof(nodeType))) == NULL)
        yyerror("out of memory");

    /* copy information */
    p->type = typeId;
    p->id.i = i;

    return p;
}

nodeType *opr(int oper, int nops, ...) {
    va_list ap;
    nodeType *p;
    int i;

    /* allocate node, extending op array */
    if ((p = malloc(sizeof(nodeType) + (nops-1) * sizeof(nodeType *)))
== NULL)
        yyerror("out of memory");

    /* copy information */
    p->type = typeOpr;
    p->opr.oper = oper;
    p->opr.nops = nops;
    va_start(ap, nops);
    for (i = 0; i < nops; i++)
        p->opr.op[i] = va_arg(ap, nodeType*);
    va_end(ap);
    return p;
}

void freeNode(nodeType *p) {
    int i;

    if (!p) return;
```

```
    if (p->type == typeOpr) {
        for (i = 0; i < p->opr.nops; i++)
            freeNode(p->opr.op[i]);
    }
    free (p);
}

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

int main(void) {
    yyparse();
    return 0;
}
```

File 2: practical9.l

```
%{
#include <stdlib.h>
#include "practical9.h"
#include "y.tab.h"
void yyerror(char *);
%}

%%

[a-z]       {
                yylval.sIndex = *yytext - 'a';
                return VARIABLE;
            }

0           {
                yylval.iValue = atoi(yytext);
                return INTEGER;
            }
```

```
[1-9][0-9]* {
                yylval.iValue = atoi(yytext);
                return INTEGER;
            }

[-()<>=+*/;{}.] {
                return *yytext;
             }

">="            return GE;
"<="            return LE;
"=="            return EQ;
"!="            return NE;
"while"         return WHILE;
"if"            return IF;
"else"          return ELSE;
"print"         return PRINT;

[ \t\n]+        ;       /* ignore whitespace */

.               yyerror("Unknown character");
%%
int yywrap(void) {
    return 1;
}
```

Execution Sequence:

```
E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_9>bison -y -d practical9.y

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_9>flex practical9.l

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_9>gcc -c y.tab.c lex.yy.c

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_9>gcc y.tab.o lex.yy.o practical9.c -o practical9.exe

E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_9>practical9.exe
```

Output:

```
E:\Semester 7\CC\Lab\19BCE519_ 2CS701_Practical_9>practical9.exe
a=b+c*d/e-f*g;
        push    b
        push    c
        push    d
        mul
        push    e
        div
        add
        push    f
        push    g
        mul
        sub
        pop     a
```

Conclusion:

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