Assignment 5 – Implementation of code optimization techniques

Date: 05/05/2023

Aim:

To apply the following techniques to optimize the code.

1. Constant folding
2. Algebraic identities
3. Strength reduction
4. Dead code elimination

Code:

//ex5.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror(char* s);
extern YYSTYPE yylval;
int line = 0;
%}

identifier [a-zA-Z_][a-zA-Z0-9_]*

doubConst (\+|\-)?[0-9][0-9]*\.[0-9]+
intConst (\+|\-)?[0-9][0-9]*
charConst \'.\'
stringConst \"(.)*\"

op ("+"|"-"|"*"|"/"|"%")
relop "<"|">"|"<="|">="|"=="|"!="
boolop ("!"|"&&"|"||")

%%
[ \t]          {}
\n             {}

"void"         {printf(" %25s | %-25s \n", yytext, "type void"); return VOID;}
```

```
"char"          {printf(" %25s | %-25s \n", yytext, "type char"); return CHAR;}
"int"          {printf(" %25s | %-25s \n", yytext, "type int"); return INT;}
"float"          {printf(" %25s | %-25s \n", yytext, "type float"); return FLOAT;}
"double"          {printf(" %25s | %-25s \n", yytext, "type double"); return DOUBLE;}
"String"        {printf(" %25s | %-25s \n", yytext, "type string"); return STR;}

"{"           {printf(" %25s | %-25s \n", yytext, "left curly"); return LEFTCURLY;}
"}"           {printf(" %25s | %-25s \n", yytext, "right curly"); return RIGHTCURLY;}
"["           {printf(" %25s | %-25s \n", yytext, "left square"); return LEFTSQUARE;}
"]"           {printf(" %25s | %-25s \n", yytext, "right square"); return RIGHTSQUARE;}
"("           {printf(" %25s | %-25s \n", yytext, "left parantheses"); return LEFT;}
")"           {printf(" %25s | %-25s \n", yytext, "right parantheses"); return RIGHT;}

"="           {printf(" %25s | %-25s \n", yytext, "equal to op"); return ASSIGN_OP;}

"+"           {printf(" %25s | %-25s \n", yytext, "plus op"); return PLUS_OP;}
"-"           {printf(" %25s | %-25s \n", yytext, "minus op"); return MINUS_OP;}
"*"           {printf(" %25s | %-25s \n", yytext, "mul op"); return MUL_OP;}
"/"           {printf(" %25s | %-25s \n", yytext, "div op"); return DIV_OP;}
"%"           {printf(" %25s | %-25s \n", yytext, "mod op"); return MOD_OP;}

"!"           {printf(" %25s | %-25s \n", yytext, "not op"); return NOT_OP;}
"&&"          {printf(" %25s | %-25s \n", yytext, "and op"); return AND_OP;}
"||"          {printf(" %25s | %-25s \n", yytext, "or op"); return OR_OP;}

{relop}         {printf(" %25s | %-25s \n", yytext, "rel op"); return REL_OP;}

{intConst}|{doubConst}|{charConst}      {printf(" %25s | %-25s \n", yytext, "numeric constant");
yylval.num=atoi(strdup(yytext)); return CONSTANT;}
{stringConst}   {printf(" %25s | %-25s \n", yytext, "string constant"); yylval.string=strdup(yytext);
return STRING;}

{identifier}    {printf(" %25s | %-25s \n", yytext, "identifier"); yylval.string=strdup(yytext); return
ID;}

","           {printf(" %25s | %-25s \n", yytext, "comma"); return COMMA;}
";"           {printf(" %25s | %-25s \n", yytext, "semi-colon"); return EOS;}
.

%%
```

```
//ex5.y

%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
int yyerror();
extern FILE* yyin;
extern int line;
int error = 0;
int tempCount = 1;
char temp[10];
%}

%union {
    char* string;
    int num;
    };

%type <string> expr boolop
%type <num> numexpr

%token <string> VOID INT FLOAT DOUBLE CHAR STR
%token <string> LEFTCURLY RIGHTCURLY LEFTSQUARE RIGHTSQUARE LEFT RIGHT
COMMA EOS
%token <string> ASSIGN_OP PLUS_OP MINUS_OP MUL_OP DIV_OP MOD_OP REL_OP
NOT_OP AND_OP OR_OP
%token <string> ID STRING
%token <num> CONSTANT

%%
program:
statement program {
    return 0;
    }
| '\n'
|
;

statement:
```

```
declnstatement EOS
| assignment EOS
;

declnstatement:
type var
;

type:
INT
| FLOAT
| DOUBLE
| CHAR
| STR
| type LEFTSQUARE RIGHTSQUARE
;

var:
var COMMA var
| ID
| assignment
;

assignment:
ID ASSIGN_OP expr {
    printf("%s = %s\n", $1, $3);
}
| ID ASSIGN_OP numexpr {
    printf("%s = %d\n", $1, $3);
}
;

expr:
ID {
    /*sprintf(temp, "temp%d", tempCount++);
    printf("%s = %s\n", temp, $1);
    $$ = strdup(temp);*/
    $$ = $1;
}
| expr PLUS_OP expr {
    if (strcmp($1, "0") && strcmp($3, "0")) {
        sprintf(temp, "temp%d", tempCount++);
        printf("%s = %s %s %s\n", temp, $1, "+", $3);
```

```
        $$ = strdup(temp);
    }
    else {
        sprintf(temp, "temp%d", tempCount-1);
        $$ = strdup($1);
    }
}
| expr MINUS_OP expr {
    if (strcmp($1, "0") && strcmp($3, "0")) {
        sprintf(temp, "temp%d", tempCount++);
        printf("%s = %s %s %s\n", temp, $1, "-", $3);
        $$ = strdup(temp);
    }
    else {
        sprintf(temp, "temp%d", tempCount-1);
        $$ = strdup($1);
    }
}
| expr MUL_OP expr {
    if (strcmp($1, "1") && strcmp($3, "1")) {
        sprintf(temp, "temp%d", tempCount++);
        printf("%s = %s %s %s\n", temp, $1, "*", $3);
        $$ = strdup(temp);
    }
    else {
        sprintf(temp, "temp%d", tempCount-1);
        $$ = strdup($1);
    }
}
| expr DIV_OP expr {
    if (strcmp($3, "1")) {
        sprintf(temp, "temp%d", tempCount++);
        printf("%s = %s %s %s\n", temp, $1, "/", $3);
        $$ = strdup(temp);
    }
    else {
        sprintf(temp, "temp%d", tempCount-1);
        $$ = strdup($1);
    }
}
| expr REL_OP expr
| expr boolop expr
| NOT_OP expr
```

```
| numexpr {
    char number[10];
    sprintf(number, "%d", $1);
    $$ = strdup(number);
}
;

numexpr:
CONSTANT {
    $$ = $1;
}
| numexpr PLUS_OP numexpr {
    $$ = $1 + $3;
}
| numexpr MINUS_OP numexpr {
    $$ = $1 - $3;
}
| numexpr MUL_OP numexpr {
    $$ = $1 * $3;
}
| numexpr DIV_OP numexpr {
    $$ = $1 / $3;
}
;

boolop:
AND_OP {
    $$ = strdup($1);
}
| OR_OP {
    $$ = strdup($1);
}
;

%%

int yywrap(){
    return 1;
}

int yyerror() {
    fprintf(stderr, "\n\nSyntax is NOT valid! Error at line %d\n", line);
    error = 1;
```

```
    return 0;
}

int main(int argc, char *argv[])
{
    printf("Syntax Analyser: \n");
    if (argc != 2) {
        printf("Please enter the sample file as the second argument!\n");
        exit(0);
    }

    yyin = fopen(argv[1], "r");
    if (!yyin) {
        printf("File not found!\n");
        exit(0);
    }

    yyparse();
    if(!error){
        printf("\n\nValid syntax!\n");
    }

    return 0;
}
```

Output:

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex5$ ./a.out test1.txt
Syntax Analyser:
                        c | identifier
                        = | equal to op
                        a | identifier
                        + | plus op
                        b | identifier
                        * | mul op
                       20 | numeric constant
                        ; | semi-colon
temp1 = b * 20
temp2 = a + temp1
c = temp2
                        d | identifier
                        = | equal to op
                        a | identifier
                        + | plus op
                        c | identifier
                        ; | semi-colon
temp3 = a + c
d = temp3


Valid syntax!
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex5$ cat test1.txt
c = a + b * 20;
d = a + c;
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex5$
```

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex5$ ./a.out test2.txt
Syntax Analyser:
                        c | identifier
                        = | equal to op
                        2 | numeric constant
                        + | plus op
                        3 | numeric constant
                        ; | semi-colon
c = 5
                        a | identifier
                        = | equal to op
                        c | identifier
                        * | mul op
                        4 | numeric constant
                        ; | semi-colon
temp1 = c * 4
a = temp1


Valid syntax!
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex5$ cat test2.txt
c = 2 + 3;
a = c * 4;
```

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex5$ ./a.out test3.txt
Syntax Analyser:
                        y | identifier
                        = | equal to op
                        a | identifier
                        * | mul op
                        b | identifier
                        + | plus op
                        c | identifier
                        ; | semi-colon
temp1 = b + c
temp2 = a * temp1
y = temp2
                        x | identifier
                        = | equal to op
                        c | identifier
                        * | mul op
                        1 | numeric constant
                        + | plus op
                        0 | numeric constant
                        ; | semi-colon
x = c


Valid syntax!
```

Learning outcomes:

- The internal working of a compiler was analysed and understood.
- The concept of tokens and parsing for tokens in Java was understood and implemented.
- A syntax analyser was implemented for a Java program using the lex and yacc tools.
- Intermediate code was generated for the given sample code using the lex and yacc tools.
- Intermediate code was optimised for the given sample code.
