CSPC62 -Compiler Design
Lab Assignment 4
Basic Blocks, Flow Graphs, DAG, Code Optimization

Team Members & Contributions

Avinash (106120019) – DAG , Code Optimization

Hariesh (106120041) – ICG modification for loops, Basic Blocks, Flow Graphs

Gopi (106120075) – ICG modification, Basic Blocks

Vishal (106120145) – Fixing the 3-Address Codes, Code optimization

Submitted on: 3/5/2023

LEX Code

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "y.tab.h"
    int countn=0;
%}
%%
"ip"                        {strcpy(yylval.nd_obj.name, yytext); printf("ip"); return INPUT;}
"op"                        {strcpy(yylval.nd_obj.name, yytext); printf("op"); return OUTPUT;}

"int"                        {strcpy(yylval.nd_obj.name, yytext); printf("dt_integer");; return
DT_INT;}
"float"                       {strcpy(yylval.nd_obj.name, yytext); printf("dt_float"); return
DT_FLOAT;}
"char"                       {strcpy(yylval.nd_obj.name, yytext); printf("dt_character"); return
DT_CHAR;}
"str"                        {strcpy(yylval.nd_obj.name, yytext); printf("dt_string"); return DT_STR;}
"bool"                       {strcpy(yylval.nd_obj.name, yytext); printf("dt_boolean"); return
DT_BOOL;}

"if"                        {strcpy(yylval.nd_obj.name, yytext); printf("if"); return IF;}
"else"                       {strcpy(yylval.nd_obj.name, yytext); printf("else"); return ELSE;}

"as"                        {strcpy(yylval.nd_obj.name, yytext); printf("as"); return AS;}

"true"                       {strcpy(yylval.nd_obj.name, yytext); printf("true"); return TRUE;}
"false"                       {strcpy(yylval.nd_obj.name, yytext); printf("false"); return FALSE;}

"+"                        {strcpy(yylval.nd_obj.name, yytext); printf("add"); return ADD;}
"-"                        {strcpy(yylval.nd_obj.name, yytext); printf("sub"); return SUBTRACT;}
"*"                        {strcpy(yylval.nd_obj.name, yytext); printf("mult"); return MULTIPLY;}
"/"                        {strcpy(yylval.nd_obj.name, yytext); printf("divide"); return DIVIDE;}
"%"                        {strcpy(yylval.nd_obj.name, yytext); printf("mod"); return MODULO;}
"**"                        {strcpy(yylval.nd_obj.name, yytext); printf("pow"); return RAISE_TO;}
"++"                        {strcpy(yylval.nd_obj.name, yytext); printf("incr"); return INCREMENT;}
"--"                        {strcpy(yylval.nd_obj.name, yytext); printf("decr"); return DECREMENT;}
```

```
"<"                         {strcpy(yylval.nd_obj.name, yytext); printf("lt"); return LESS_THAN;}
">"                         {strcpy(yylval.nd_obj.name, yytext); printf("gt"); return GREATER_THAN;}
"<="                         {strcpy(yylval.nd_obj.name, yytext); printf("lte"); return LESS_OR_EQ;}
">="                         {strcpy(yylval.nd_obj.name, yytext); printf("gte"); return GREATER_OR_EQ;}
"="                         {strcpy(yylval.nd_obj.name, yytext); printf("assign");; return ASSIGN;}
"=="                         {strcpy(yylval.nd_obj.name, yytext); printf("eq"); return EQUAL;}
"!="                         {strcpy(yylval.nd_obj.name, yytext); printf("neq"); return NOT_EQ;}
"&&"                         {strcpy(yylval.nd_obj.name, yytext); printf("and"); return AND;}
"||"                         {strcpy(yylval.nd_obj.name, yytext); printf("or"); return OR;}


" "                         {printf(" ");}


"("                         {strcpy(yylval.nd_obj.name, yytext); printf("("); return RO;}
")"                         {strcpy(yylval.nd_obj.name, yytext); printf(")"); return RC;}
"{"                         {strcpy(yylval.nd_obj.name, yytext); printf("{"); return CO;}
"}"                         {strcpy(yylval.nd_obj.name, yytext); printf("}"); return CC;}


([0][A-Za-z]([A-Za-z]|[0-9])*|[1-9][0-9]*[A-Za-z]([A-Za-z]|[0-9])*) {printf("wrong_id"); return
ERROR;}


([0]|[1-9][0-9]*)           {strcpy(yylval.nd_obj.name, yytext); yylval.nd_obj.value = atoi(yytext);
printf("num"); return NUMBER;}
(([0-9]+)\.([0-9]+))        {strcpy(yylval.nd_obj.name, yytext); printf("float"); return FLOAT_NUM;}
(["].*["])                  {strcpy(yylval.nd_obj.name, yytext); printf("string"); return STRING;}
(['].['])                   {strcpy(yylval.nd_obj.name, yytext); printf("character"); return CHAR;}


([A-Za-z]([A-Za-z]|[0-9])*) {strcpy(yylval.nd_obj.name, yytext); printf("id (%s)", yytext);
yylval.nd_obj.idVal = yytext[0] - 'a'; return ID;}


"\n" { printf("\n"); countn++;}
"\r" { printf("\r"); }


. {printf("error - %d", *yytext); return *yytext;}


%%


int yywrap() {
   return 1;
}
```

YACC Code

```
%{
   #include<stdio.h>
   #include<string.h>
   #include<stdlib.h>
   #include<ctype.h>
   void yyerror(const char *s);
   int yylex();
   int yywrap();
   int sym[26];

   // intermediate code generation related
   char buffer[100];
   char icg[200][200];
   int label = 0;
   int ic_idx = 0;
   int temp_no = 0;

   struct Node {
      char label[32];
      struct Node* children[10];
      int num_children;
      int value;
   };

   struct Node * createEntity(char name[32], int val);
   void add_child(struct Node *parent, struct Node *child);

   struct Node *root = NULL;

   int x = 15;
%}

%union {
   struct var_name {
      char name[32];
```

```
      int value;
      int idVal;
      struct Node* entity;
      char lexname[32];
   } nd_obj;

   struct ifelse {
      char name[32];
      char if_body[5];
      char else_body[5];
      struct Node* entity;
      char lexname[32];
   } nd_obj_2;
}

%token <nd_obj> INPUT OUTPUT DT_INT DT_FLOAT DT_CHAR DT_STR DT_BOOL IF ELSE AS TRUE FALSE ADD SUBTRACT
MULTIPLY DIVIDE MODULO RAISE_TO INCREMENT DECREMENT LESS_THAN GREATER_THAN LESS_OR_EQ GREATER_OR_EQ
ASSIGN EQUAL NOT_EQ AND OR RO RC CO CC NUMBER FLOAT_NUM STRING CHAR ID ERROR
%type <nd_obj> start body block else statement value datatype relop expression term factor
%type <nd_obj_2> condition whileCondition

%%

start: body {
   $$.entity = createEntity("start", 0);
   add_child($$.entity, $1.entity);
   root = $$.entity;
}
;

body: block {
   $$.entity = createEntity("body", 0);
   add_child($$.entity, $1.entity);
}
| block body {
   $$.entity = createEntity("body", 0);
   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
```

```
}

;


block: AS RO whileAddOns whileCondition RC CO body CC addOn { printf("\n parser : while loop"); }

| IF RO condition RC { sprintf(icg[ic_idx++], "\nNEW BLOCK \nLABEL %s:\n", $3.if_body); } CO body CC {
sprintf(icg[ic_idx++], "\nBLOCK ENDS\n"); sprintf(icg[ic_idx++], "\nNEW BLOCK \nLABEL %s:\n",
$3.else_body); } else {

   //sprintf(icg[ic_idx++],"GOTO next\n");

   sprintf(icg[ic_idx++], "\nBLOCK ENDS\n");


   $$.entity = createEntity("block", 0);

   $1.entity = createEntity("IF", 0);

   $2.entity = createEntity("RO", 0);

   $4.entity = createEntity("RC", 0);

   $6.entity = createEntity("CO", 0);

   $8.entity = createEntity("CC", 0);


   add_child($$.entity, $1.entity);

   add_child($$.entity, $2.entity);

   add_child($$.entity, $3.entity);

   add_child($$.entity, $4.entity);

   add_child($$.entity, $6.entity);

   add_child($$.entity, $7.entity);

   add_child($$.entity, $8.entity);

   add_child($$.entity, $10.entity);

}

| statement {

   $$.entity = createEntity("block", 0);


   add_child($$.entity, $1.entity);

}

| OUTPUT RO value RC {

   $$.entity = createEntity("block", 0);

   $1.entity = createEntity("OUTPUT", 0);

   $2.entity = createEntity("RO", 0);

   $4.entity = createEntity("RC", 0);


   add_child($$.entity, $1.entity);
```

```
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);
   add_child($$.entity, $4.entity);

   sprintf(icg[ic_idx++], "\nOUTPUT %s", $3.name);
}
| datatype ID ASSIGN INPUT RO STRING RC
;

else: ELSE CO body CC {
   $$.entity = createEntity("else", 0);
   $1.entity = createEntity("ELSE", 0);
   $2.entity = createEntity("CO", 0);
   $4.entity = createEntity("CC", 0);

   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);
   add_child($$.entity, $4.entity);
}
|
;

whileCondition: value relop value {
   sprintf(icg[ic_idx++], "\nif !(%s %s %s) GOTO LOOP_EXIT\n", $1.name, $2.name, $3.name);

   $$.entity = createEntity("condition", 0);

   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);
}

addOn: {sprintf(icg[ic_idx++], "GOTO LOOP\n\nNEW BLOCK \nLOOP_EXIT: \n");}

whileAddOns : {sprintf(icg[ic_idx++], "NEW BLOCK \nLOOP:");}
|
```

```
statement: datatype ID ASSIGN expression {
   sprintf(icg[ic_idx++], "%s = %s\n", $2.name, $4.name);

   $$.entity = createEntity("statement", 0);
   $2.entity = createEntity("ID", 0);
   $3.entity = createEntity("ASSIGN", 0);
   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);
   add_child($$.entity, $4.entity);

   sym[$2.idVal] = ($4.entity)->value;
}
| ID ASSIGN expression {

   sprintf(icg[ic_idx++], "%s = %s\n", $1.name, $3.name); sym[$1.idVal] = $3.value;

   $$.entity = createEntity("statement", 0);
   $1.entity = createEntity("ID", ($3.entity)->value);
   $2.entity = createEntity("ASSIGN", 0);
   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);

   sym[$1.idVal] = ($3.entity)->value;
}
;

value: NUMBER {
  // sprintf(icg[ic_idx++], "value = %s\n", $1.name);
   strcpy($$.name, $1.name);

   $$.entity = createEntity("value", 0);
   $1.entity = createEntity("NUMBER", $1.value);

   add_child($$.entity, $1.entity);
```

```
   ($$.entity)->value = $1.value;
   ($1.entity)->value = $1.value;
}
| FLOAT_NUM {
   $$.entity = createEntity("value", 0);
   $1.entity = createEntity("FLOAT_NUM", $1.value);

   add_child($$.entity, $1.entity);

   ($$.entity)->value = $1.value;
   ($1.entity)->value = $1.value;
}
| STRING
| CHAR
| ID {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("value", 0);
   $1.entity = createEntity("ID", sym[$1.idVal]);

   add_child($$.entity, $1.entity);

   ($$.entity)->value = sym[$1.idVal];
   ($1.entity)->value = sym[$1.idVal];
}
;

datatype: DT_INT {
   $$.entity = createEntity("datatype", 0);
   $1.entity = createEntity("DT_INT", 0);

   add_child($$.entity, $1.entity);
}
| DT_FLOAT {
   $$.entity = createEntity("datatype", 0);
   $1.entity = createEntity("DT_FLOAT", 0);
```

```
   add_child($$.entity, $1.entity);
}
| DT_CHAR
| DT_STR
| DT_BOOL {
   $$.entity = createEntity("datatype", 0);
   $1.entity = createEntity("DT_BOOL", 0);

   add_child($$.entity, $1.entity);
}
;

condition: value relop value {
   sprintf(icg[ic_idx++], "\nif (%s %s %s) GOTO L%d else GOTO L%d\n", $1.name, $2.name, $3.name,
label, label+1);

   sprintf($$.if_body, "L%d",label++);
   sprintf($$.else_body,"L%d",label++);

   $$.entity = createEntity("condition", 0);

   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);
}
| TRUE {
   $$.entity = createEntity("condition", 1);
   $1.entity = createEntity("TRUE", 1);

   add_child($$.entity, $1.entity);
}
| FALSE {
   $$.entity = createEntity("condition", 0);
   $1.entity = createEntity("FALSE", 0);

   add_child($$.entity, $1.entity);
}
```

```
;

relop: LESS_THAN {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("LESS_THAN", 0);

   add_child($$.entity, $1.entity);
}
| GREATER_THAN {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("GREATER_THAN", 0);

   add_child($$.entity, $1.entity);
}
| LESS_OR_EQ {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("LESS_OR_EQ", 0);

   add_child($$.entity, $1.entity);
}
| GREATER_OR_EQ {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("GREATER_OR_EQ", 0);

   add_child($$.entity, $1.entity);
}
| EQUAL {
   strcpy($$.name, $1.name);
```

```
   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("EQUAL", 0);

   add_child($$.entity, $1.entity);
}
| NOT_EQ {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("NOT_EQ", 0);

   add_child($$.entity, $1.entity);
}
| AND {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("AND", 0);

   add_child($$.entity, $1.entity);
}
| OR {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("relop", 0);
   $1.entity = createEntity("OR", 0);

   add_child($$.entity, $1.entity);
}
;

expression: expression ADD term {
   $$.entity = createEntity("expression", 0);
   $2.entity = createEntity("ADD", 0);

   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
```

```
   add_child($$.entity, $3.entity);


   sprintf(icg[ic_idx++], "t%d = %s %s %s\n",  temp_no ,$1.name, $2.name, $3.name); $$.value =
$1.value + $3.value;
   char tempvar[21] = "t";
   char tNo[11];
   sprintf(tNo, "%d", temp_no);


   strcat(tempvar, tNo);
   temp_no++;
   strcpy($$.name, tempvar);


   ($$.entity)->value = ($1.entity)->value + ($3.entity)->value;
}
| expression SUBTRACT term {
   $$.entity = createEntity("expression", 0);
   $2.entity = createEntity("SUBTRACT", 0);


   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);


   sprintf(icg[ic_idx++], "t%d = %s %s %s\n",  temp_no ,$1.name, $2.name, $3.name); $$.value =
$1.value - $3.value;
   char tempvar[21] = "t";
   char tNo[11];
   sprintf(tNo, "%d", temp_no);


   strcat(tempvar, tNo);
   temp_no++;
   strcpy($$.name, tempvar);


   ($$.entity)->value = ($1.entity)->value - ($3.entity)->value;
}
| term {


   $$.entity = createEntity("term", 0);
```

```
   add_child($$.entity, $1.entity);

   ($$.entity)->value = ($1.entity)->value;
}
;

term: term MULTIPLY factor {
   $$.entity = createEntity("term", 0);
   $2.entity = createEntity("MULTIPLY", 0);

   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);

   sprintf(icg[ic_idx++], "Unoptimized: \n");
   sprintf(icg[ic_idx++], "t%d = %s %s %s\n\n",  temp_no ,$1.name, $2.name, $3.name); $$.value =
$1.value * $3.value;

   sprintf(icg[ic_idx++], "Optimized: \n");
   for(int i = 0; i < ($3.entity)->value; i++)
   {
      sprintf(icg[ic_idx++], "t%d = t%d + %s\n",  temp_no , temp_no, $1.name);
   }
   sprintf(icg[ic_idx++], "\n");

   char tempvar[21] = "t";
   char tNo[11];
   sprintf(tNo, "%d", temp_no);

   strcat(tempvar, tNo);
   temp_no++;
   strcpy($$.name, tempvar);

   ($$.entity)->value = ($1.entity)->value * ($3.entity)->value;
}
| term DIVIDE factor {
   $$.entity = createEntity("term", 0);
   $2.entity = createEntity("DIVIDE", 0);
```

```
   add_child($$.entity, $1.entity);
   add_child($$.entity, $2.entity);
   add_child($$.entity, $3.entity);

   sprintf(icg[ic_idx++], "t%d = %s %s %s\n",  temp_no ,$1.name, $2.name, $3.name); $$.value =
$1.value / $3.value;
   char tempvar[21] = "t";
   char tNo[11];
   sprintf(tNo, "%d", temp_no);

   strcat(tempvar, tNo);
   temp_no++;
   strcpy($$.name, tempvar);

   ($$.entity)->value = ($1.entity)->value / ($3.entity)->value;
}
| factor {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("term", 0);

   add_child($$.entity, $1.entity);

   ($$.entity)->value = ($1.entity)->value;
}
;

factor: value {
   strcpy($$.name, $1.name);

   $$.entity = createEntity("factor", 0);

   add_child($$.entity, $1.entity);

   ($$.entity)->value = ($1.entity)->value;
}
;
```

```
%%

struct Node * createEntity(char name[32], int val){
   struct Node *parent = (struct Node*)malloc(sizeof(struct Node));
   for(int i=0; i<32; i++){
      parent->label[i] = name[i];
   }
   parent->value = val;
   parent->num_children = 0;
   return parent;
}

void add_child(struct Node *parent, struct Node *child) {
   parent->children[parent->num_children] = child;
   parent->num_children++;
}

void st_traverse(struct Node *root){
   if(root == NULL)
      return;

   printf("\n");
   printf(root->label);
   printf(" %d", root->value);
   for(int i=0; i<root->num_children; i++){
      st_traverse(root->children[i]);
   }

   return;
}

int main() {
   printf("LEXICAL ANALYSIS\n");
   yyparse();
   printf("\n\nSYNTAX TREE PREORDER\n");
   printf("node value\n");
```

```
    st_traverse(root);
    printf("\n\nINTERMEDIATE CODE GENERATION\n\n");
    printf("NEW BLOCK (start)\n");
    for(int i=0; i<ic_idx; i++){
        printf("%s", icg[i]);
    }
}

void yyerror(const char* msg) {
    fprintf(stderr, "%s\n", msg);
}
```

Sample input

main.bs

```
1    int a = 10
2    int b = a * 2 - 5
3    int c = a / 2 + 10
4
5    as (a < 15) {
6        if(a > 12){
7            c = c * 5
8        } else {
9            b = b - 1
10       }
11       a = a - 1
12   }
13
14   op(a)
15   op(b)
16   op(c)
```

Sample output

```
LEXICAL ANALYSIS
dt_integer id (a) assign num
dt_integer id (b) assign id (a) mult num sub num
dt_integer id (c) assign id (a) divide num add num

as (id (a) lt num) {
    if(id (a) gt num){
        id (c) assign id (c) mult num
    } else {
        id (b) assign id (b) sub num
    }
    id (a) assign id (a) sub num
}
 parser : while loop

op(id (a))
op(id (b))
op(id (c))
```

```
SYNTAX TREE PREORDER
node value

start 0
body 0
block 0
statement 0
datatype 0
DT_INT 0
ID 0
ASSIGN 0
term 10
term 10
factor 10
value 10
NUMBER 10
body 0
block 0
statement 0
datatype 0
DT_INT 0
ID 0
ASSIGN 0
expression 15
term 20
term 20
term 10
factor 10
value 10
ID 10
MULTIPLY 0
factor 2
value 2
NUMBER 2
```

```
SUBTRACT 0
term 5
factor 5
value 5
NUMBER 5
body 0
block 0
statement 0
datatype 0
DT_INT 0
ID 0
ASSIGN 0
expression 15
term 5
term 5
term 10
factor 10
value 10
ID 10
DIVIDE 0
factor 2
value 2
NUMBER 2
ADD 0
term 10
factor 10
value 10
NUMBER 10
body 0
body 0
block 0
OUTPUT 0
RO 0
value 9
ID 9
RC 0
body 0
block 0
OUTPUT 0
```

```
RO 0
value 14
ID 14
RC 0
body 0
block 0
OUTPUT 0
RO 0
value 75
ID 75
RC 0
```

```
INTERMEDIATE CODE GENERATION

NEW BLOCK (start)
a = 10
Unoptimized:
t0 = a * 2

Optimized:
t0 = t0 + a
t0 = t0 + a

t1 = t0 - 5
b = t1
t2 = a / 2
t3 = t2 + 10
c = t3
NEW BLOCK
LOOP:
if !(a < 15) GOTO LOOP_EXIT

if (a > 12) GOTO L0 else GOTO L1

NEW BLOCK
LABEL L0:
Unoptimized:
t4 = c * 5
```

```
Optimized:
t4 = t4 + c
t4 = t4 + c
t4 = t4 + c
t4 = t4 + c
t4 = t4 + c

c = t4

BLOCK ENDS

NEW BLOCK
LABEL L1:
t5 = b - 1
b = t5

BLOCK ENDS
t6 = a - 1
a = t6
GOTO LOOP

NEW BLOCK
LOOP_EXIT:
```