Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Department of Computer Science and Engineering

Continuous Assessment Test - 4

Question Paper

SET I

Degree & Branch		BE (CSE)		**Semester**	VI
Subject Code & Name		UCS1602 Compiler Design		**Regulation: 2018**	
Section		C	**Academic Year**	**2020-2021**	
Reg. No:		18 5001 196	**Name**	S. Vishakan	
Date:	30.04.2021	Batch: III	**Time: 1.15 pm – 3.15 pm**	**Max. marks : 50**	

Code for Optimized three address code generation

Lex Code:

```
%{

	#include <stdio.h>
	#include <stdlib.h>
	#include <string.h>
	#include <math.h>

	#include "y.tab.h"
%}

id ([a-zA-Z\_][a-zA-Z0-9]*)
digit ([0-9]+)

%%

{id}		{yylval.str = strdup(yytext); return ID;}
{digit}		{yylval.intval = atoi(yytext); return INT;}
{digit}[.]{digit}	{yylval.fltval = atof(yytext); return FLOAT;}
```

```
[\n]            {return *yytext;}
[ \t]           {;}
.               {return *yytext;}

%%
```

Yacc Code:

```
%{

        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        #include <math.h>

        //Redefine the inbuilt yyfunctions
        int yyerror(char *);
        int yylex(void);
        int yywrap();

        //Keep track of counts
        int labelCounter(){
                static int labels = 0;
                return labels++;
        }

        int varCounter(){
                static int variables = 0;
                return variables++;
        }

        //Node to keep track of ICG, Optimized Code and ASM
        typedef struct Node{
                int intval;
                float fltval;
                char *code;
                char *optcode;
                char *tac;
                char *opttac;
                char *gen;
        }Node;

Node *makeNode(){
```

```
Node *n = (Node *)malloc(sizeof(Node));
n->intval = 0;
n->fltval = 0.0;
n->code = (char *)malloc(sizeof(char) * 1024);
n->optcode = (char *)malloc(sizeof(char) * 1024);
n->tac = (char *)malloc(sizeof(char) * 1024);
n->opttac = (char *)malloc(sizeof(char) * 1024);
n->gen = (char *)malloc(sizeof(char) * 1024);
return n;
}
%}

%union{
int intval;
float fltval;
char *str;
struct Node* node;
}

/*
Precedence ^ is lesser priority than + , -, *, /
Associativity + and - left , * and / right
*/

/* %right '^'
%right '*' '/'
%left '+' '-' */

%token <str> ID
%token <intval> INT
%token <fltval> FLOAT
%type <node> Code Block Stmts Stmt Assign Expr E T F

%%
Code : Block{
printf("\n---INPUT CODE---\n");
system("cat code.txt");
printf("\n---SYNTAX CHECK---\n");
printf("\nSyntactically Correct.\n");
printf("\n---TAC CODE---\n");
printf("\n%s\n", $1->tac);
printf("\n---OPTIMIZED CODE---\n");
printf("\n%s\n", $1->opttac);
printf("\n---MACHINE CODE---\n");
printf("\n%s\n", $1->gen);
```

```
}
;

Block : Stmts{
$$ = $1;
}
;

Stmts : Stmt Stmts{
$$ = makeNode();
sprintf($$->tac, "%s%s", $1->tac, $2->tac);
sprintf($$->opttac, "%s%s", $1->opttac, $2->opttac);
sprintf($$->gen, "%s%s", $1->gen, $2->gen);
}
| Stmt{
$$ = $1;
}
;

Stmt : Assign '\n'{
$$ = $1;
}
;

Assign : ID '=' Expr{
$$ = makeNode();
sprintf($$->code, "%s", $1);
char temp[100], asmcode[100];
sprintf(temp, "%s := %s\n", $$->code, $3->code);
sprintf(asmcode, "MOV %s, R0\nMOV R0, %s\n", $3->code, $$->code);

sprintf($$->tac, "%s%s", $3->tac, temp);
sprintf($$->opttac, "%s%s", $3->opttac, temp);
sprintf($$->gen, "%s%s", $3->gen, asmcode);
}
;

Expr : E{
$$ = $1;
}
;

E : E '+' T{
```

```
$$ = makeNode();
int vc = varCounter();
sprintf($$->code, "T%d", vc);
char temp[100], asmcode[100];
sprintf(temp, "%s = %s + %s\n", $$->code, $1->code, $3->code);

float v1, v2;

if (v1 = strtof($1->code, NULL)){
if(v2 = strtof($3->code, NULL)){
v1 = v1 + v2;
sprintf($$->opttac, "%s = %.2f\n", $$->code, v1);
sprintf(asmcode, "MOV #%2.f, R0\n", v1);
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nADD %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nADD %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf($$->gen, "%s%s%s", $1->gen, $3->gen, asmcode);
}
| E '-' T{
$$ = makeNode();
int vc = varCounter();
sprintf($$->code, "T%d", vc);
char temp[100], asmcode[100];
sprintf(temp, "%s = %s - %s\n", $$->code, $1->code, $3->code);

float v1, v2;

if (v1 = strtof($1->code, NULL)){
if(v2 = strtof($3->code, NULL)){
v1 = v1 - v2;
sprintf($$->opttac, "%s = %.2f\n", $$->code, v1);
sprintf(asmcode, "MOV #%2.f, R0\n", v1);
}
else{
```

```
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nSUB %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nSUB %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf($$->gen, "%s%s%s", $1->gen, $3->gen, asmcode);
}
| T{
$$ = $1;
}
;

T : F '*' T{
$$ = makeNode();
int vc = varCounter();
sprintf($$->code, "T%d", vc);
char temp[100], asmcode[100];
sprintf(temp, "%s = %s * %s\n", $$->code, $1->code, $3->code);

float v1, v2;

if (v1 = strtof($1->code, NULL)){
if(v2 = strtof($3->code, NULL)){
v1 = v1 * v2;
sprintf($$->opttac, "%s = %.2f\n", $$->code, v1);
sprintf(asmcode, "MOV #%2.f, R0\n", v1);
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nMUL %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nMUL %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}
```

```
sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf($$->gen, "%s%s%s", $1->gen, $3->gen, asmcode);
}
| F '/' T{
$$ = makeNode();
int vc = varCounter();
sprintf($$->code, "T%d", vc);
char temp[100], asmcode[100];
sprintf(temp, "%s = %s / %s\n", $$->code, $1->code, $3->code);

float v1, v2;

if (v1 = strtof($1->code, NULL)){
if(v2 = strtof($3->code, NULL)){
v1 = v1 / v2;
sprintf($$->opttac, "%s = %.2f\n", $$->code, v1);
sprintf(asmcode, "MOV #%2.f, R0\n", v1);
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nDIV %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nDIV %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf($$->gen, "%s%s%s", $1->gen, $3->gen, asmcode);
}
| F '^' T{
$$ = makeNode();
int vc = varCounter();
sprintf($$->code, "T%d", vc);
char temp[100], asmcode[100];
sprintf(temp, "%s = %s ^ %s\n", $$->code, $1->code, $3->code);

float v1, v2;

if (v1 = strtof($1->code, NULL)){
if(v2 = strtof($3->code, NULL)){
```

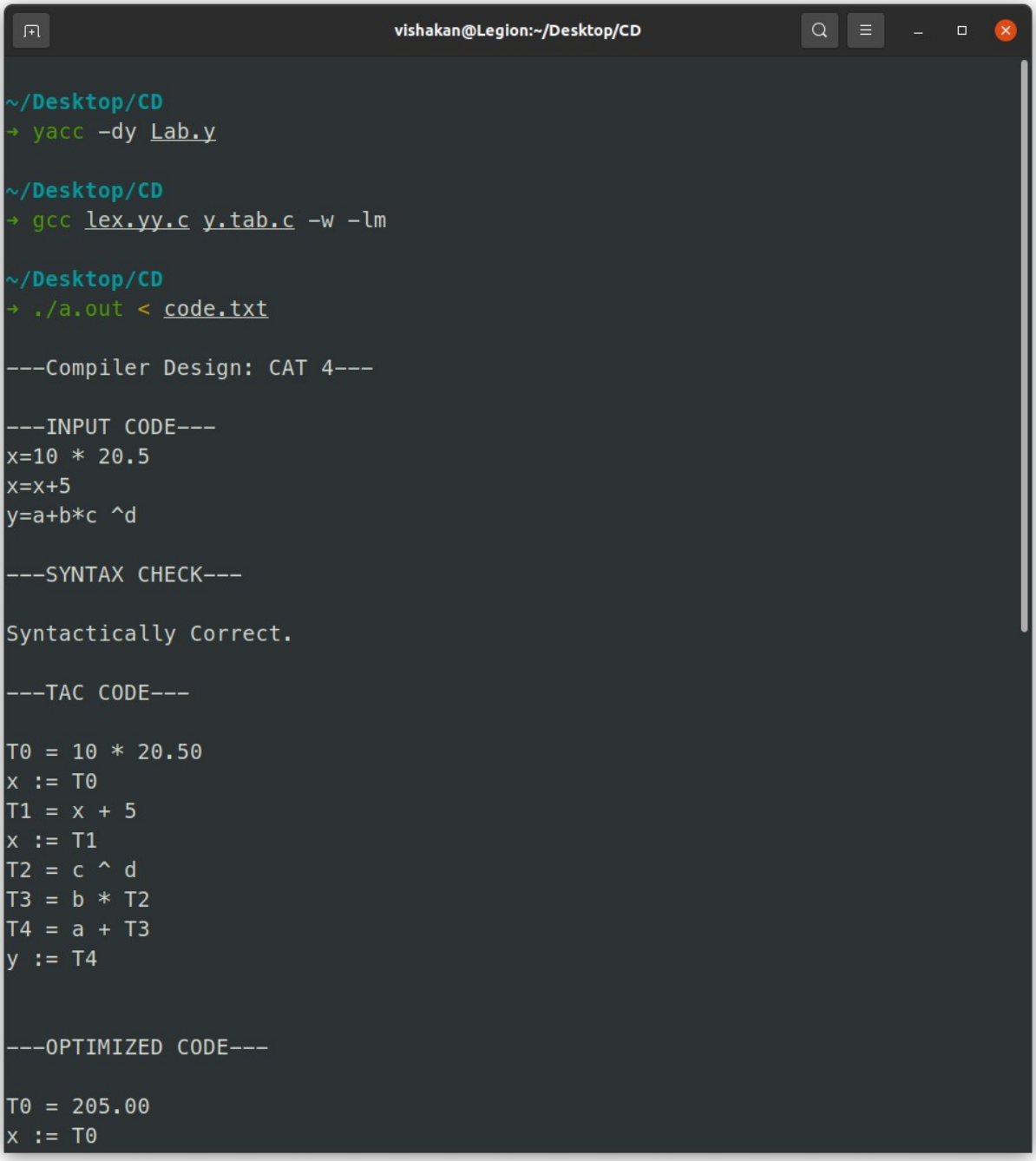
```
v1 = pow(v1, v2);
sprintf($$->opttac, "%s = %.2f\n", $$->code, v1);
sprintf(asmcode, "MOV #%2.f, R0\n", v1);
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nPOW %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}
}
else{
sprintf($$->opttac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf(asmcode, "MOV %s, R0\nPOW %s, R0\nMOV R0, %s\n", $1->code, $3-
>code, $$->code);
}

sprintf($$->tac, "%s%s%s", $1->tac, $3->tac, temp);
sprintf($$->gen, "%s%s%s", $1->gen, $3->gen, asmcode);
}
| F{
$$ = $1;
}
;

F : INT{
$$ = makeNode();
sprintf($$->code, "%d", $1);
sprintf($$->tac, "");
sprintf($$->opttac, "");
sprintf($$->gen, "");
}
| FLOAT{
$$ = makeNode();
sprintf($$->code, "%.2f", $1);
sprintf($$->tac, "");
sprintf($$->opttac, "");
sprintf($$->gen, "");
}
| ID{
$$ = makeNode();
sprintf($$->code, "%s", $1);
sprintf($$->tac, "");
sprintf($$->opttac, "");
sprintf($$->gen, "");
}
```

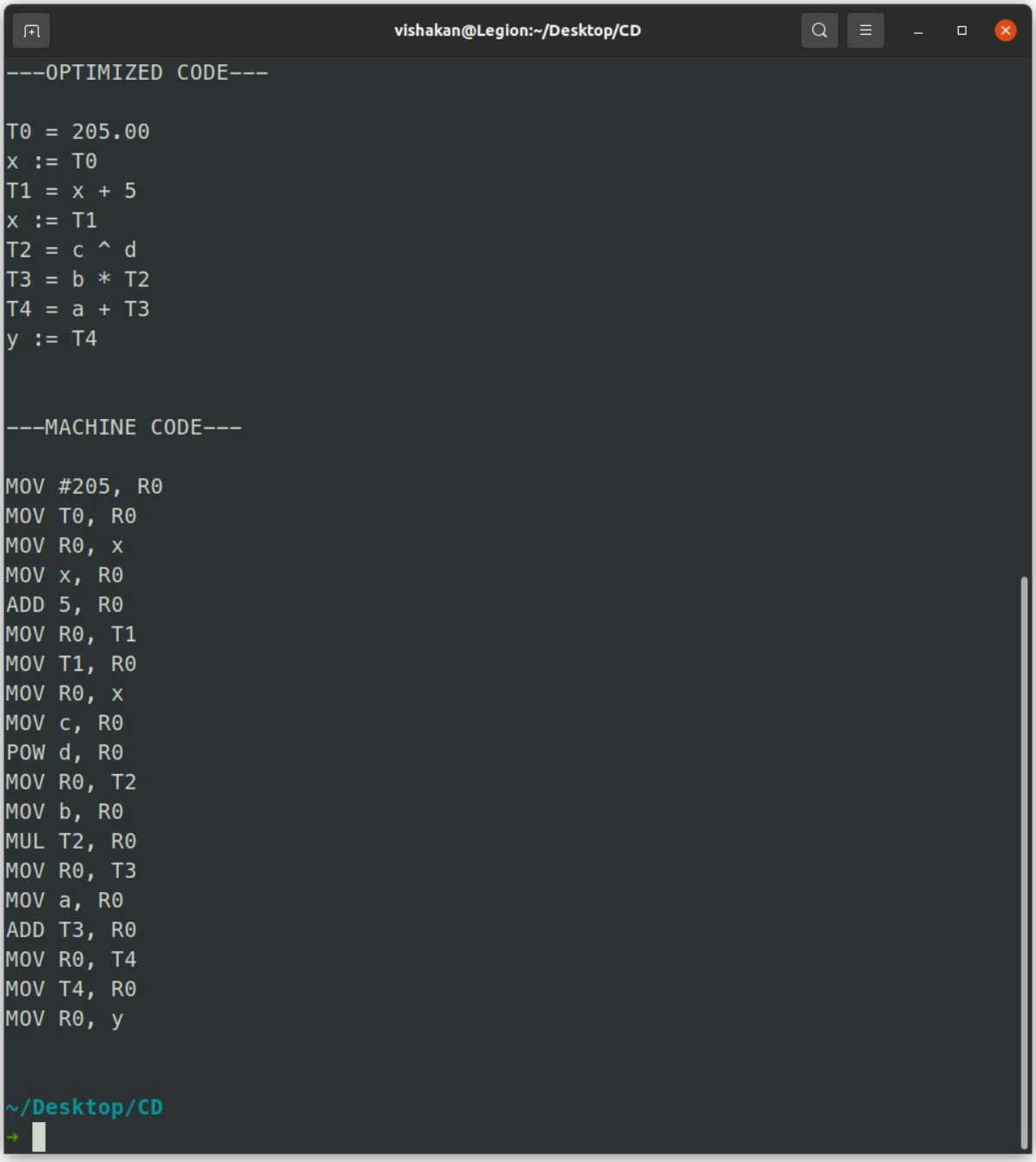
```
;

%%

int yyerror(char *error){
        fprintf(stderr, "\nError: %s\n", error);
        return 0;
}

int yywrap(){
        return 1;
}

int main(){
        printf("\n---Compiler Design: CAT 4---\n");
        yyparse();
        return 0;
}
```

Parsed Code:

```
x=10 * 20.5
x=x+5
y=a+b*c ^d
```

Output:


vishakan@Legion:~/Desktop/CD
~/Desktop/CD
→ yacc -dy Lab.y
~/Desktop/CD
→ gcc lex.yy.c y.tab.c -w -lm
~/Desktop/CD
→ ./a.out < code.txt
---Compiler Design: CAT 4---
---INPUT CODE---
x=10 * 20.5
x=x+5
y=a+b*c ^d
---SYNTAX CHECK---
Syntactically Correct.
---TAC CODE---
T0 = 10 * 20.50
x := T0
T1 = x + 5
x := T1
T2 = c ^ d
T3 = b * T2
T4 = a + T3
y := T4
---OPTIMIZED CODE---
T0 = 205.00
x := T0


vishakan@Legion:~/Desktop/CD
---OPTIMIZED CODE---
T0 = 205.00
x := T0
T1 = x + 5
x := T1
T2 = c ^ d
T3 = b * T2
T4 = a + T3
y := T4
---MACHINE CODE---
MOV #205, R0
MOV T0, R0
MOV R0, x
MOV x, R0
ADD 5, R0
MOV R0, T1
MOV T1, R0
MOV R0, x
MOV c, R0
POW d, R0
MOV R0, T2
MOV b, R0
MUL T2, R0
MOV R0, T3
MOV a, R0
ADD T3, R0
MOV R0, T4
MOV T4, R0
MOV R0, y
~/Desktop/CD