LP ASSIGNMENT

ROLL.NO : 411755

Name : Sai Likhith

Aim : To generate optimal code using sethi-ullmann algorithm.

Requirements : Lex,Yacc must be pre-installed to run the code.

Files : Lex.l,Parser.y

commands(to run) : (In Linux)
1. lex Lex.l
2. yacc -d Parser.y
3. cc lex.yy.c y.tab.c -ll
4. ./a.out (prompts you to enter a expression)

Lex.L

```
%{
#include "y.tab.h"
//void yyerror(char *);
%}
%%
0 {
	yylval.t= yytext;
	return INTEGER;
	}


[1-9][0-9]* {
	yylval.t= yytext;
	return INTEGER;
	}
[a-zA-Z] {
	yylval.t= yytext;
	return INTEGER; }
[-()<>=+*/;{}.] {
	return *yytext;
```

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

Parser.y

```
%{
#include<stdio.h>
#include<string.h>
typedef struct node{
    char *operator;
    struct node *loperand;
    struct node *roperand;
    int Reg_Req;
}node;
//#define YYSTYPE int
void yyerror(char *s);
node *create_node(char *op,node *left,node *right);            //to create a node in a parse tree
int Reg_required(node *root);                 //to compute no. of reg.required after
generating parse tree
void Label_leaves(node *root);              //to label the leaves....left-leaf=1,right-leaf=0
void gen_code(node *root);                  //function to generate code...
int max(int a,int b);                  //function to return the maximun of two values
int is_leaf(node *root);               //returns weather a node is leaf or not
char *decode_operator(node *root);                //decodes the oprator and returns
correspoding instruction......ex:returns "ADD" for '+'
void print_tree(node *root);                //to print the parse tree
void print2DUtil(node *root, int space);              //helper function in printing parse tree
#define COUNT 10                            //used in parse_tree printing
int n_reg=3;                           //no. of registers present
int t_reg=5;                           //no. of temporary registers
//from here these are helper fumctions for Temporary stack,Register stack
int Rstack_top;
```

```
int Rstack[3];
int Tstack_top;
int Tstack[5];
void initiate_stack();
void swap();
int Rpop();
int Tpop();
void Rpush();
void Tpush();
%}

%union {
        struct node *L;
        char *t;
        };
%start S
%type <L> E T F
%token <t> INTEGER
%left '+' '-'
%left '*' '/'

%%

        S:    E';'   {//';'remarks end of expression....
                     Label_leaves($1);
                     Reg_required($1);
                     printf("\nOptimal No. of Registers Required to compute expression : %d\n",$1->Reg_Req);
                     print_tree($1);
                     initiate_stack();
                     printf("............................Generating Optimal code using sethi-ullmann algorithm..........................................\n");
                     gen_code($1);
                    }
```

```
  ;

E:  E'+'T          {$$=create_node("+",$1,$3);}
 |  E'-'T      {$$=create_node("-",$1,$3);}
 |  T          {$$=$1;}
 ;
T:  T'*'F          {$$=create_node("*",$1,$3);}
 |  T'/'F      {$$=create_node("/",$1,$3);}
 |  F          {$$=$1;}
 ;
F:  '('E')'    {$$=$2;}
 |  INTEGER        {$$=create_node(yylval.t,NULL,NULL);
                   }
 ;

%%

node *create_node(char *op,node *op1,node *op2){

	node *temp=(node *)malloc(sizeof(node));
	char *op_temp = (char *)malloc(strlen(op)+1);
	strcpy(op_temp,op);

	temp->operator=op_temp;
	if(op1){
	temp->loperand=op1;}
	else{temp->loperand=NULL;}
	if(op2){
	temp->roperand=op2;}
	else{temp->roperand=NULL;}
	return(temp);
}
void yyerror(char *s) {
	fprintf(stdout, "%s\n", s);
```

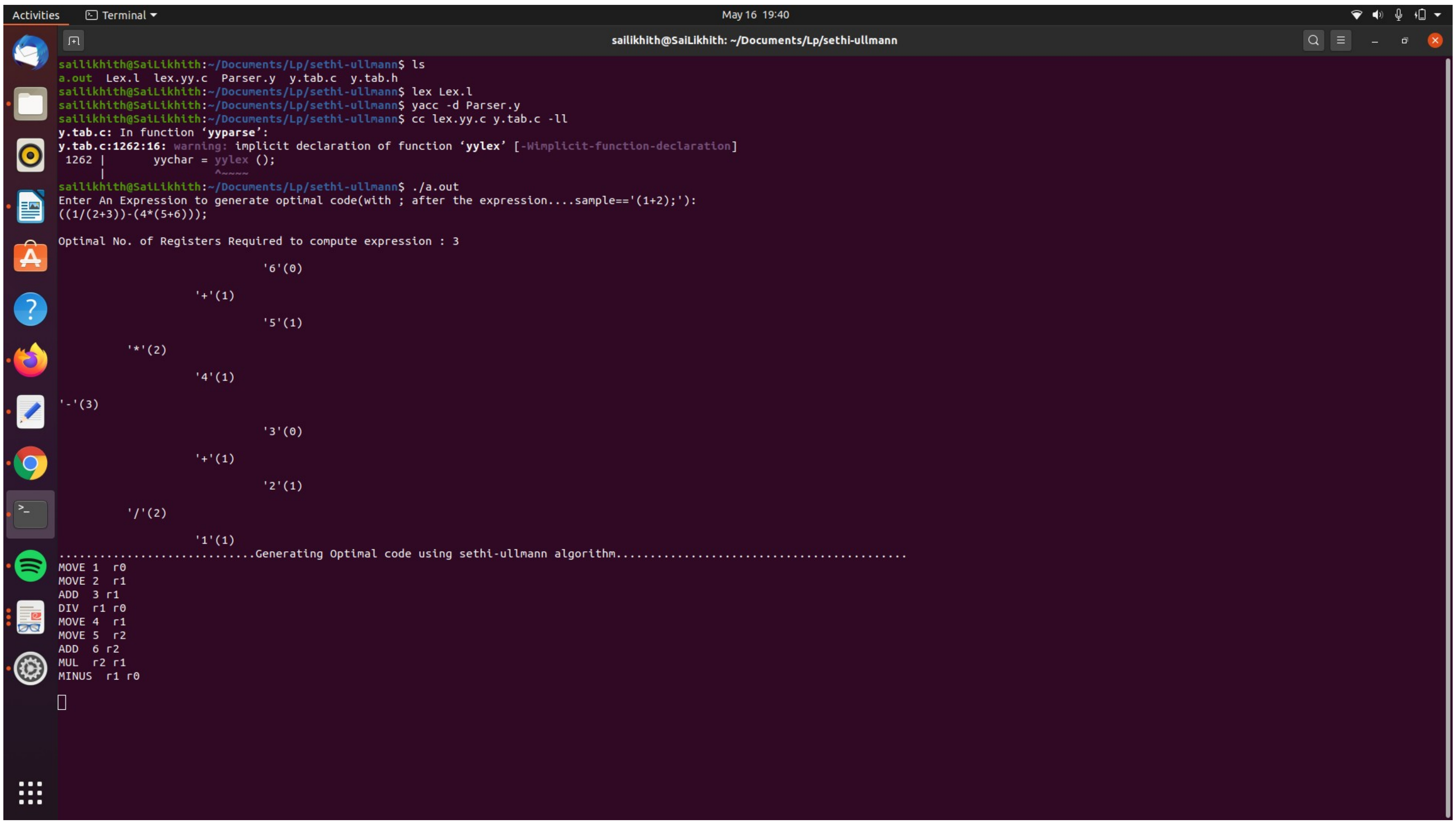
```
	}

int is_leaf(node *root){
	if(!(root->loperand) && !(root->roperand)){
	return 1;}
	else{
	return 0;}
}

int Reg_required(node *root){

	if(!(is_leaf(root))){
		int k=Reg_required(root->loperand);
		int l=Reg_required(root->roperand);
		if (k!=l){
			root->Reg_Req=max(k,l);
			}
		else{
			root->Reg_Req=k+1;
			}
		return root->Reg_Req;
	}
	else{
		return root->Reg_Req;
	}

}
void Label_leaves(node *root){
	if(root){
	if(root->loperand && is_leaf(root->loperand)){
		root->loperand->Reg_Req=1;
	}
	if(root->roperand && is_leaf(root->roperand)){
		root->roperand->Reg_Req=0;
```

```
	}
	Label_leaves(root->loperand);
	Label_leaves(root->roperand);
	}
	else{
		return ;
	}
}
void gen_code(node *root){
	if(is_leaf(root)&&(root->Reg_Req==1)){
		printf("MOVE %s  r%d  \n",root->operator,Rstack[Rstack_top]);
		return ;
	}
	else if(is_leaf(root->roperand)&& root->roperand->Reg_Req==0){

		gen_code(root->loperand);
		printf("%s  %s r%d  \n",decode_operator(root),root->roperand-
>operator,Rstack[Rstack_top]);
		return ;
	}
	else if(root->loperand->Reg_Req < root->roperand->Reg_Req && root->roperand->Reg_Req
< n_reg){

		swap();
		gen_code(root->roperand);
		int R=Rpop();
		gen_code(root->loperand);
		printf("%s  r%d r%d  \n",decode_operator(root),Rstack[Rstack_top],R);
		Rpush();
		swap();
		return ;
	}
	else if(root->loperand->Reg_Req >=root->roperand->Reg_Req && root->loperand->Reg_Req
< n_reg){
		gen_code(root->loperand);
```

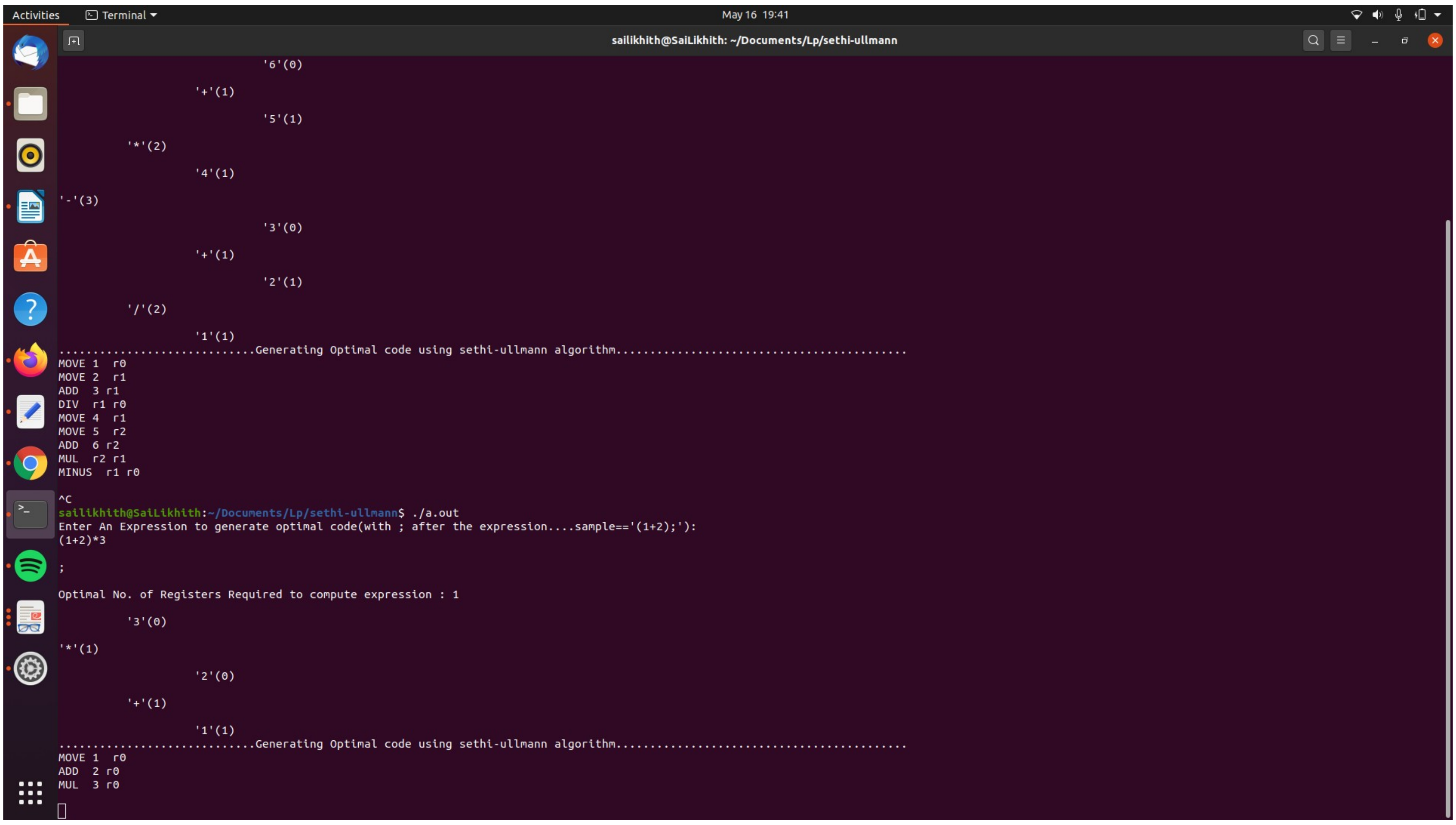
```
        int R=Rpop();
        gen_code(root->roperand);
        printf("%s  r%d r%d  \n",decode_operator(root),Rstack[Rstack_top],R);
        Rpush();
        return ;
    }
    else if(root->loperand->Reg_Req >= n_reg && root->roperand->Reg_Req >=n_reg){
        gen_code(root->roperand);
        int T=Tpop();
        gen_code(root->loperand);
        Tpush();
        printf("%s <- r%d t%d  \n",decode_operator(root),Rstack[Rstack_top],T);
        return ;
    }
    else{
        return ;
    }
}
void swap(){
    int temp=Rstack[Rstack_top];
    Rstack[Rstack_top]=Rstack[Rstack_top-1];
    Rstack[Rstack_top-1]=temp;
}
int Rpop(){
    int temp=Rstack[Rstack_top];
    Rstack_top=Rstack_top-1;
    return temp;
}
int Tpop(){
    int temp=Tstack[Tstack_top];
    Tstack_top=Tstack_top-1;
    return temp;
}
char *decode_operator(node *root){
```

```
        if(strcmp(root->operator,"+")==0){return "ADD";}
        else if(strcmp(root->operator,"-")==0){return "MINUS";}
        else if(strcmp(root->operator,"*")==0){return "MUL";}
        else {return "DIV";}
}
void initiate_stack(){
        for(int i=0;i<n_reg;i++){Rstack[i]=n_reg-i-1;}
        Rstack_top=n_reg-1;
        for(int i=0;i<t_reg;i++){Tstack[i]=t_reg-i-1;}
        Tstack_top=t_reg-1;
}
void Rpush() {
//no need to check whther stack is full or not as sethi-ullman algo.takes care of it
      Rstack_top = Rstack_top + 1;
}
void Tpush() {
//no need to check whther stack is full or not as sethi-ullman algo.takes care of it
      Tstack_top = Tstack_top + 1;
}
void print_tree(node *root){
        print2DUtil(root, 0);
        }
void print2DUtil(node *root, int space)
{
    // Base case
    if (root == NULL)
        return;

    // Increase distance between levels
    space += COUNT;

    // Process right child first
    print2DUtil(root->roperand, space);
```

```
    // Print current node after space
    // count
    printf("\n");
    for (int i = COUNT; i < space; i++)
        printf(" ");
    printf("'%s'(%d)\n", root->operator,root->Reg_Req);

    // Process left child
    print2DUtil(root->loperand, space);
}
int max(int a,int b){
        if(a>b){
                return a;
                }
                else{
                return b;}
        }
int main (void) {
        printf("Enter An Expression to generate optimal code(with ; after the
expression....sample=='(1+2);'):");
        yyparse ();
        return 0;
        }
```



```
sailikhith@SaiLikhith:~/Documents/Lp/sethi-ullmann$ ls
a.out  Lex.l  lex.yy.c  Parser.y  y.tab.c  y.tab.h
sailikhith@SaiLikhith:~/Documents/Lp/sethi-ullmann$ lex Lex.l
sailikhith@SaiLikhith:~/Documents/Lp/sethi-ullmann$ yacc -d Parser.y
sailikhith@SaiLikhith:~/Documents/Lp/sethi-ullmann$ cc lex.yy.c y.tab.c -ll
y.tab.c: In function 'yyparse':
y.tab.c:1262:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 1262 |       yychar = yylex ();
      |                ^~~~~
sailikhith@SaiLikhith:~/Documents/Lp/sethi-ullmann$ ./a.out
Enter An Expression to generate optimal code(with ; after the expression....sample=='(1+2);'):
((1/(2+3))-(4*(5+6)));

Optimal No. of Registers Required to compute expression : 3

                              '6'(0)

                    '+'(1)

                              '5'(1)

          '*'(2)

                    '4'(1)

'-'(3)

                              '3'(0)

                    '+'(1)

                              '2'(1)

          '/'(2)

                    '1'(1)
.............................Generating Optimal code using sethi-ullmann algorithm.........................................
MOVE 1  r0
MOVE 2  r1
ADD  3 r1
DIV  r1 r0
MOVE 4  r1
MOVE 5  r2
ADD  6 r2
MUL  r2 r1
MINUS  r1 r0
```



```
                              '6'(0)

                    '+'(1)

                              '5'(1)

          '*'(2)

                    '4'(1)

'-'(3)

                              '3'(0)

                    '+'(1)

                              '2'(1)

          '/'(2)

                    '1'(1)
.............................Generating Optimal code using sethi-ullmann algorithm.........................................
MOVE 1  r0
MOVE 2  r1
ADD  3 r1
DIV  r1 r0
MOVE 4  r1
MOVE 5  r2
ADD  6 r2
MUL  r2 r1
MINUS  r1 r0

^C
sailikhith@SaiLikhith:~/Documents/Lp/sethi-ullmann$ ./a.out
Enter An Expression to generate optimal code(with ; after the expression....sample=='(1+2);'):
(1+2)*3

;

Optimal No. of Registers Required to compute expression : 1

          '3'(0)

'*'(1)

                    '2'(0)

          '+'(1)

                    '1'(1)
.............................Generating Optimal code using sethi-ullmann algorithm.........................................
MOVE 1  r0
ADD  2 r0
MUL  3 r0
```