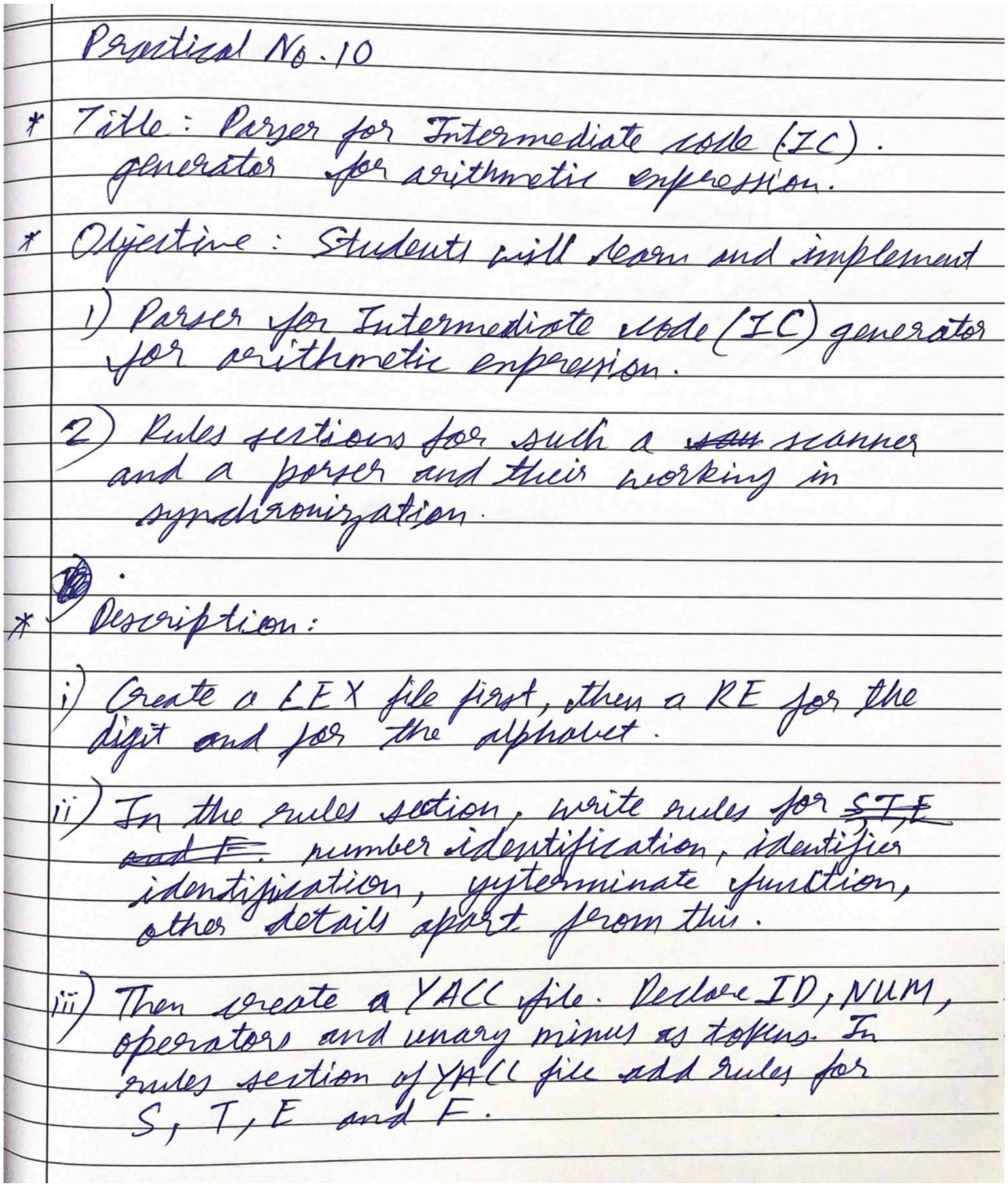


Practical No. 10

* Title: Parser for Intermediate code (IC). generator for arithmetic expression.

* Objective: Students will learn and implement

1) Parser for Intermediate code (IC) generator for arithmetic expression.

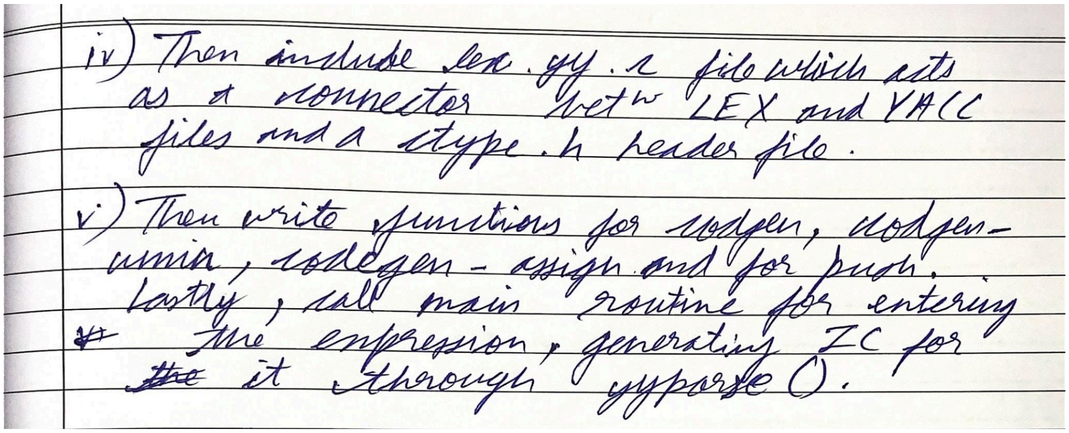
2) Rules sections for such a ~~scan~~ scanner and a parser and their working in synchronization.

* Description:

i) Create a LEX file first, then a RE for the digit and for the alphabet.

ii) In the rules section, write rules for ~~S,T,E and F.~~ number identification, identifier identification, yyterminate function, other details apart from this.

iii) Then create a YACC file. Declare ID, NUM, operators and unary minus as tokens. In rules section of YACC file add rules for S, T, E and F.



iv) Then include lex.yy.c file which acts as a connector betw LEX and YACC files and a ctype.h header file.

v) Then write functions for codgen, codgen-umin, codegen-assign and for push. Lastly, call main routine for entering the expression, generating IC for ~~the~~ it through yyparse().

Code for LEX:

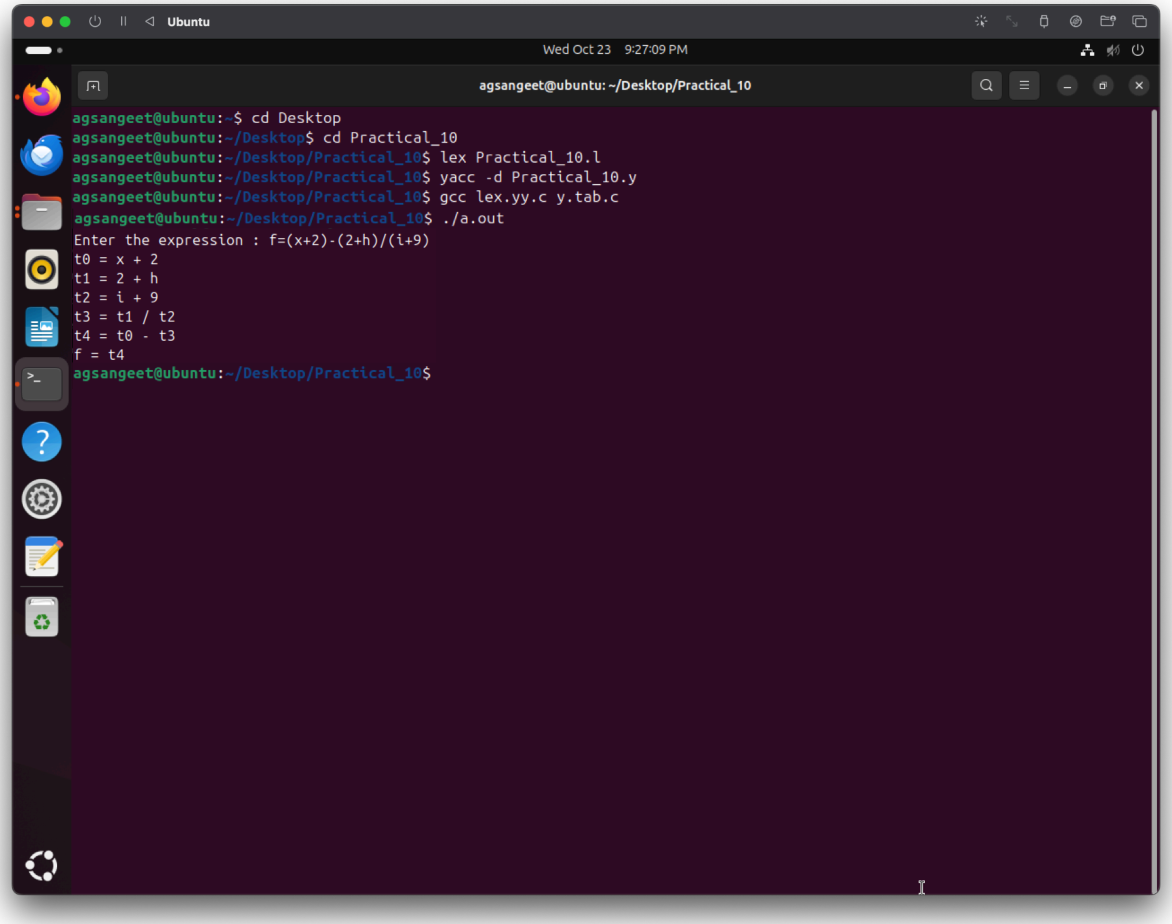
```
ALPHA [A-Za-z]
DIGIT [0-9]

%%
" "                             ;
{ALPHA}({ALPHA}|{DIGIT})*       return ID;
{DIGIT}+                        {yylval = atoi(yytext); return NUM;}
[\n\t]                          yyterminate();
.                               return yytext[0];
%%
```

Code for YACC:

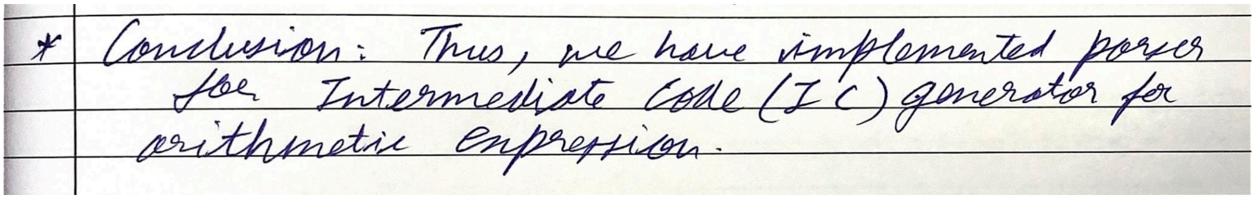
```
%{
#include <stdio.h>
void yyerror(char*);
int yylex(void);
void codegen();
void codegen_assign();
void codegen_umin();
void printnum(int);
void push();
%}

%token ID NUM
%right '='
%left '+' '-'
%left '*' '/'
%left UMINUS
%%

S    :    ID{push();} '='{push();} E{codegen_assign();}
      ;
E    :    E '+'{push();} T{codegen();}
      |    E '-'{push();} T{codegen();}
      |    T
      ;
T    :    T '*'{push();} F{codegen();}
      |    T '/'{push();} F{codegen();}
      |    F
      ;
F    :    '(' E ')'
      |    '-'{push();} F{codegen_umin();} %prec UMINUS
      |    ID{push();}
      |    NUM{push();}
      ;

%%

#include "lex.yy.c"
#include<ctype.h>
#include<string.h>
char st[100][25];
int top=0,ptr=0;
int tint=0; int tintar[200];

int main()
{
    printf("Enter the expression : ");
    yyparse();
    return 0;
}

void push()
{
  strcpy(st[++top],yytext);
  ptr++;
}

void codegen(){
    printf("t%d = %s",tint,st[top-2]);
    printnum(2);
    printf(" %s %s",st[top-1],st[top]);
    printnum(0);
    printf("\n");
    top-=2;ptr-=2;
    strcpy(st[top],"t");
    tintar[ptr]=tint;
    tint++;
}
```

```
void codegen_umin(){
    printf("t%d = -%s\n",tint,st[top]);
    printnum(0);
    top--;ptr--;
    strcpy(st[top],"t");
    tintar[ptr]=tint;
    tint++;
}

void codegen_assign(){
    printf("%s = ",st[top-2]);
    printnum(2);
    printf("%s",st[top]);
    printnum(0);
    printf("\n");
    top-=2;ptr-=2;
}

void printnum(int n){
    if( strcmp(st[top-n],"t")==0)
    {
         printf("%d",tintar[ptr-n]);
    }
}

void yyerror(char* errorText){
    printf("[ERROR] : %s",errorText);
}
```

Output:



```
agsangeet@ubuntu:~$ cd Desktop
agsangeet@ubuntu:~/Desktop$ cd Practical_10
agsangeet@ubuntu:~/Desktop/Practical_10$ lex Practical_10.l
agsangeet@ubuntu:~/Desktop/Practical_10$ yacc -d Practical_10.y
agsangeet@ubuntu:~/Desktop/Practical_10$ gcc lex.yy.c y.tab.c
agsangeet@ubuntu:~/Desktop/Practical_10$ ./a.out
Enter the expression : f=(x+2)-(2+h)/(i+9)
t0 = x + 2
t1 = 2 + h
t2 = i + 9
t3 = t1 / t2
t4 = t0 - t3
f = t4
agsangeet@ubuntu:~/Desktop/Practical_10$
```



* Conclusion: Thus, we have implemented parser for Intermediate Code (IC) generator for arithmetic expression.