Tajbir Recursive Descent Parsing

```
// S  if C then S
// | while C do S
// | id = num ;
// | id ++ ;
// C  id == num | id != num

#include "stdio.h"
#include "stdlib.h"
#include "ctype.h"

const char *input;
char lookahead;

void match(char expected) {
    if (lookahead == expected) {
        lookahead = *++input;
    } else {
        printf("Syntax error: expected %c, found %c", expected, lookahead);
        exit(1);
    }
}

void match_v2(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        match(s[i]);
    }
}

void syntax_error() {
    printf("Sytax Error\n");
    exit(1);
}

void absorb_whitespace() {
    while (lookahead == ' ') lookahead = *++input;
}
```

```
void C() {
    if (isalnum(lookahead)) {
        printf("C -> id");
        match(lookahead);
        absorb_whitespace();
        if (lookahead == '=') {
            printf("==");
            match_v2("==");
            absorb_whitespace();
        } else if (lookahead == "!") {
            printf("!=");
            match_v2("!=");
            absorb_whitespace();
        } else syntax_error();

        if (isalnum(lookahead)){
            printf("num\n");
            match(lookahead);
            absorb_whitespace();
        }
        else syntax_error();
    } else syntax_error();
}
```

```
void S() {
    if (lookahead == 'i') {
        printf ("S -> if C then S\n");
        match_v2("if");
        absorb_whitespace();
        C();
        absorb_whitespace();
        match_v2("then");
        absorb_whitespace();
        S();
        absorb_whitespace();

    } else if (lookahead == 'w') {
        printf("S -> while C do S\n");
        match_v2("while");
        absorb_whitespace();
        C();
        absorb_whitespace();
        match_v2("do");
        absorb_whitespace();
        S();
        absorb_whitespace();
    } else if (isalnum(lookahead)) {
        printf("S -> id");
        match(lookahead);
        absorb_whitespace();
        if (lookahead == '=') {
            printf("=");
            absorb_whitespace();
            match('=');
            absorb_whitespace();
            if (isalnum(lookahead)) {
                printf("num");
                match(lookahead);
                absorb_whitespace();
            } else {
                syntax_error();
            }
        } else if (lookahead == '+') {
            printf("++");
            match_v2("++");
            absorb_whitespace();
        } else {
            syntax_error();
        }

        if (lookahead == ';') {
            printf(":\n");
            match(lookahead);
            absorb_whitespace();
        }
        else syntax_error();
    }
}
```

```
int main() {
    input = (char *)malloc(100 * sizeof(char));

    input = "if 2 == 3 then 3 = 2;";

    // while (scanf("%s", input) != EOF) {
    lookahead = *input;
    printf("Parsing input: %s\n", input);
    S();
    if (lookahead == '\0') {
        printf("Parsing successful!\n");
    } else {
        syntax_error();
    }
    // }
}
```

Yakin - recur

```
/*
Recursive Descent Parsing Functions for the following CFG:
E -> T E'
E' -> + T E' |
T -> F T'
T' -> * F T' |
F -> (E) | id
*/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

void E();
void E_prime();
void T();
void T_prime();
void F();

const char* input;
char lookahead;

void match(char expected)
{
    if(lookahead == expected)
    {
        lookahead = *++input;
    }
    else
    {
        printf("Expected: %c, however, Lookahead: %c\n",expected,lookahead);
        exit(1);
    }
}

//!E -> T E'
void E()
{
    printf("E -> T E'\n");
    T();
    E_prime();
}
```

```
//!E' -> + T E' |
void E_prime()
{
    if(lookahead == '+')
    {
        printf("E' -> + T E'\n");
        match('+');
        T();
        E_prime();
    }
    else
    {
      printf("E' -> \n");
    }
}

//!T -> F T'

void T()
{
    printf("T -> F T'\n");
    F();
    T_prime();
}

//!T' -> * F T' |

void T_prime()
{
    if(lookahead == '*'){
    printf("T' -> * F T'\n");
    F();
    T_prime();
    }
    else
    {
        printf("T' -> \n");
    }

}

//F -> (E) | id

void F()
{
    if(lookahead=='(')
    {
        printf("F -> (E)\n");
        match('(');
        E();
        match(')');
    }
    else if(isalnum(lookahead))
    {
        printf("F -> id\n");
        match(lookahead);
    }
}
```

```
int main()
{
    input = (char *) malloc(100* sizeof(char));
    printf("Input your expression: \n");
    while(scanf("%s",input)!=EOF)
    {
        lookahead = *input;
        E();
        if(lookahead == '\0')
        {
            printf("Parsing is successful.\n");
        }
        else
        {
            printf("Syntax error: parsing is not successful.\n");
        }
        printf("Enter your expression: \n");
    }
    return 0;
}
```

Yakin - Symbol Table

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

#define TABLE_SIZE 100

typedef struct Symbol
{
    char name[50];
    char type[20];
    int scope;
    struct Symbol* next;
}Symbol;

Symbol* symbolTable[TABLE_SIZE];

unsigned int hash(char* name)
{
    unsigned int hashValue = 0;
    for(int i=0;name[i]!='\0';i++)
    {
        hashValue = 31 * hashValue + name[i];
    }
    return hashValue%TABLE_SIZE;
}

void insert(char* name,char* type,int scope)
{
    unsigned int index = hash(name);
    Symbol* newSymbol = (Symbol*) malloc(sizeof(Symbol));
    strcpy(newSymbol->name,name);
    strcpy(newSymbol->type,type);
    newSymbol->scope = scope;
    newSymbol->next = symbolTable[index];
    symbolTable[index] = newSymbol;
}
```

```
Symbol* lookup(char* name)
{
    unsigned int index = hash(name);
    Symbol* current = symbolTable[index];
    if(current!=NULL)
    {
        if(strcmp(current->name,name)==0)
        {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

int main()
{
    insert("x","int",0);
    Symbol* s = lookup("x");
    if(s!=NULL)
    {
        printf("%s of type %s is found in scope %d\n",s->name,s->type,s->scope);
    }
    else
    {
        printf("Symbol is not found.\n");
    }
    return 0;
}
```

Scanner.l

```
/* recognize tokens for the calculator and print them out */
%{
   #include "parser.tab.h"
   #include<stdlib.h>
   extern int yylval;
%}
%%
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"|-" { return ABS; }
"(" {return OP;}
")" {return CP;}
"//".* { return EOL;}
"&" {return AND;}
"|" {return OR;}
0x[a-f0-9]+ {yylval = strtol(yytext,'\0',16); return NUMBER;}
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n { return EOL; }
[ \t] { /* ignore whitespace */ }
. { printf("Mystery character %c\n", *yytext); }
%%
```

parser.y

```
/* simplest version of calculator */
%{
#include<stdio.h>
int yylex();
void yyerror(const char *s);
%}

/* declare tokens */
%token NUMBER
%token ADD SUB MUL DIV ABS
%token EOL
%token OP CP
%token AND OR

%%
calclist:
    | calclist exp EOL {printf("\nResult = %ld\n",$2);}
    | calclist EOL
    ;

exp: factor
    | exp ADD factor {$$ = $1 + $3;}
    | exp SUB factor {$$ = $1 - $3;}
    | exp OR factor {$$ = $1 | $3;}
    ;
factor: term
    | factor MUL term {$$ = $1 * $3;}
    | factor DIV term {$$ = $1 / $3;}
    | factor AND term {$$ = $1 & $3;}

    ;
term: NUMBER
    | ABS term {$$ = $2 >=0 ? $2 : -$2;}
    | OP exp CP {$$ = $2;}
```

```
%%

int main(int argc,char **argv)
{
    yyparse();
}
void yyerror(const char *s)
{
    fprintf(stderr,"error:
%s\n",s);
}
int yywrap(){
    return 1;
}
```

Makefile

```
Runner: scanner.l parser.y
 bison -d parser.y
 flex scanner.l
 gcc lex.yy.c parser.tab.c -o test
```

Only Flex

```
/* recognize tokens for the calculator and print them out */
%{
    #include "example1_5.tab.h"
    #include<stdlib.h>
    enum yytokentype{
        NUMBER = 258,
        ADD = 259,
        SUB = 260,
        MUL = 261,
        DIV = 262,
        ABS = 263,
        EOL = 264
    };
    extern int yylval;
%}
%%
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"|" { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n { return EOL; }
[ \t] { /* ignore whitespace */ }
. { printf("Mystery character %c\n", *yytext); }
%%
int main(int argc, char **argv)
{
    int tok;
    while(tok = yylex()) {
    printf("%d", tok);
    if(tok == NUMBER) printf(" = %d\n", yylval);
    else printf("\n");
    }
    return 0;
}
int yywrap()
{
    return 1;
}
```

Counting words, characters, and lines

```
/* just like Unix wc */
%{
#include<stdio.h>
#include<ctype.h>
#include<string.h>

int chars = 0;
int words = 0;
int lines = 0;
%}
%%
[^ \t\n\r\f\v]+ { words++; chars += strlen(yytext); }
\n {lines++;}
. {chars += yyleng;}
%%
```

```
int main(int argc,char* argv[])
{
    if(argc==2)
    {
        yyin = fopen(argv[1],"r");
        if(!yyin)
        {
            perror("Error opening file");
            return 1;
        }
    }
    else{
        printf("Enter input: \n");
        yyin=stdin;
    }
    yylex();
    printf("\nNumber of characters : %d\nNumber of
words : %d\nNumber of lines :
%d\n",chars,words,lines);
    if(yyin != stdin)
        fclose(yyin);
    return 0;
}
int yywrap()
{
    return 1;
}
```

Similar to previous one - but from multiple files

```
%{
#include <stdio.h>
#include <stdlib.h>
#include<string.h>

int chars = 0, words = 0, lines = 0;
int t_chars = 0, t_words = 0, t_lines = 0;
%}
%option noyywrap
WORDS [^ \t\n\r\v]+
NEWLINE \n
%%
{WORDS}  { words++; chars += strlen
(yytext); }  /* Count words and characters */
{NEWLINE}  { lines++; chars++; }  /* Count new
lines and add newline char */
. {chars++;}
%%
```

```
int main(int argc, char** argv) {
    if (argc>=2) {
        for(int i=1;i<argc;i++)
        {
        yyin = fopen(argv[i], "r");
        if (!yyin) {
            perror(argv[1]);
            return 1;
        }
            yyrestart(yyin);
            yylex();
            t_chars += chars;
            t_words += words;
            t_lines += lines;
            chars = 0;
            words = 0;
            lines = 0;
        }
        printf("\nNumber of total characters: %d\n", t_chars);
        printf("Number of total words: %d\n", t_words);
        printf("Number of total lines: %d\n", t_lines);

    }
    else {
        printf("Enter input:\n");
        yyin = stdin;
        yylex();

        printf("\nNumber of characters: %d\n", chars);
        printf("Number of words: %d\n", words);
        printf("Number of lines: %d\n", lines);
    }

    if (yyin != stdin) fclose(yyin);
    return 0;
}

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

Raihancal.l

```
/* recognize tokens for the calculator and print them out */
%{
#include "Raihancalc.tab.h"
#include <stdio.h>

%}

%%
"("    { return OP; }
")"    { return CP; }
"+"     { return ADD; }
"-"    { return SUB; }
"*"    { return MUL; }
"/"    { return DIV; }
"%"    { return MOD; }
"sq"   { return SQUARE; }
"sqrt" { return SQROOT; }
"sin"  { return SIN; }
"cos"  { return COS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n     { return EOL; }
[ \t]  { /* ignore whitespace */ }
.      { printf("Mystery character %c\n", *yytext); }
%%
```

```
%{

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

double const PI = 3.14159265359;
int yylex();
void yyerror(char *s);
%}

%token SQUARE SQROOT SIN COS
%token ADD SUB MUL DIV MOD OP CP
%token NUMBER
%token EOL

%left ADD SUB
%left MUL DIV MOD

%%

calclist:
        | calclist exp EOL { printf("= %d\n", $2); }
        ;

exp: exp ADD exp        { $$ = $1 + $3; }
   | exp SUB exp        { $$ = $1 - $3; }
   | exp MUL exp        { $$ = $1 * $3; }
   | exp DIV exp        { $$ = $1 / $3; }
   | exp MOD exp        { $$ = $1 % $3; }
   | OP exp CP          { $$ = $2; } // ( exp )
   | SQUARE OP exp CP   { $$ = $3 * $3; } // sq ( exp )
   | SQROOT OP exp CP   { $$ = sqrt($3); } // sqrt ( exp )
   | SIN OP exp CP      { $$ = sin($3 * PI / 180); } // sin ( exp )
   | COS OP exp CP      { $$ = cos($3 * PI / 180); } // cos ( exp )
   | NUMBER
   ;
%%

int main(int argc, char *argv[]) {
    yyparse();
}

void yyerror(char *s) {
    fprintf(stderr, "error: %s\n", s);
}
int yywrap()
{
    return 1;
}
```