// NAME: AHAN BANDYOPADHYAY

//ROLL No. 211210008

//Compiler Design Lab 11

1. Consider the example of simple desk calculator that performs simple operations on integer expressions with the grammar:

exp -> exp addop term | term addop -> + | -

term -> term mulop factor | factor mulop -> *

factor -> (exp) | number

number -> number digit | digit

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

You are required to write YACC specifications for this grammar so that the parser evaluates any arithmetic expressions and the output shows each grammar rule as it is applied in the parsing process. Show your parsing sequence for the input string: (2+(3*4))

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
%}

%token NUMBER
%token PLUS MINUS TIMES LPAREN RPAREN
%left PLUS MINUS
%left TIMES
%start exp

%%

exp : exp PLUS term  { printf("exp -> exp + term\n"); $$ = $1 + $3; }
   | exp MINUS term { printf("exp -> exp - term\n"); $$ = $1 - $3; }
   | term           { printf("exp -> term\n"); $$ = $1; }
   ;

term : term TIMES factor { printf("term -> term * factor\n"); $$ = $1 * $3; }
    | factor          { printf("term -> factor\n"); $$ = $1; }
    ;

factor : LPAREN exp RPAREN { printf("factor -> (exp)\n"); $$ = $2; }
      | NUMBER            { printf("factor -> number\n"); $$ = $1; }
      ;

NUMBER: DIGIT+
DIGIT: '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
PLUS: '+'
MINUS: '-'
TIMES: '*'
LPAREN: '('
RPAREN: ')'

%%
```

```
int yylex() {
    int c = getchar();
    if (isdigit(c)) {
        yylval = c - '0';
        return NUMBER;
    } else if (c == '+') {
        return PLUS;
    } else if (c == '-') {
        return MINUS;
    } else if (c == '*') {
        return TIMES;
    } else if (c == '(') {
        return LPAREN;
    } else if (c == ')') {
        return RPAREN;
    } else if (c == EOF) {
        return 0;
    } else {
        return -1; // Error
    }
}

int main() {
    yyparse();
    return 0;
}
```

OUTPUT:

```
Input: (2+(3*4))

factor -> (exp)
exp -> term
term -> term * factor
exp -> exp + term
exp -> term
term -> factor
factor -> number
factor -> number
factor -> number
```

2. Write a YACC program to evaluate the following expressions. Specify the semantic actions clearly.

Functions Meaning
(+ x1 x2 x3 ….. xn) Calculate Sum of x1, x2 upto xn
(* x1 x2 x3 ….. xn) Calculate Product of x1, x2 upto xn
(max x1 x2 x3 ….. xn) Calculate Maximum of x1, x2 upto xn
(min x1 x2 x3 ….. xn) Calculate Minimum of x1, x2 upto xn
Sample Input:
(+ 6 12 18)
Sample Output:
36

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
%}

%token NUMBER
%token PLUS MULTIPLY MAXIMUM MINIMUM LPAREN RPAREN
%start exp

%%

exp : PLUS LPAREN expr_list RPAREN { $$ = calculate_sum($3); }
   | MULTIPLY LPAREN expr_list RPAREN { $$ = calculate_product($3); }
   | MAXIMUM LPAREN expr_list RPAREN { $$ = calculate_maximum($3); }
   | MINIMUM LPAREN expr_list RPAREN { $$ = calculate_minimum($3); };

expr_list : expr { $$ = $1; }
         | expr_list expr { $$ = join_expr_list($1, $2); } ;

expr : NUMBER { $$ = $1; };
%%
int calculate_sum(int count, ...) {
   va_list args;
   va_start(args, count);
   int sum = 0;
   for (int i = 0; i < count; i++) {
      sum += va_arg(args, int);
   }
   va_end(args);
   return sum;}
int calculate_product(int count, ...) {
   va_list args;
   va_start(args, count);
   int product = 1;
   for (int i = 0; i < count; i++) {
```

```
        product *= va_arg(args, int);
    }
    va_end(args);
    return product;
}

int calculate_maximum(int count, ...) {
    va_list args;
    va_start(args, count);
    int max = va_arg(args, int);
    for (int i = 1; i < count; i++) {
        int num = va_arg(args, int);
        if (num > max) {
            max = num;
        }
    }
    va_end(args);
    return max;
}

int calculate_minimum(int count, ...) {
    va_list args;
    va_start(args, count);
    int min = va_arg(args, int);
    for (int i = 1; i < count; i++) {
        int num = va_arg(args, int);
        if (num < min) {
            min = num;
        }
    }
    va_end(args);
    return min;
}

int* join_expr_list(int* list1, int num) {
    int* result = malloc(sizeof(int) * (list1[0] + 1));
    if (!result) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    for (int i = 0; i < list1[0]; i++) {
        result[i] = list1[i];
    }
    result[list1[0]] = num;
```

```
    result[0]++;
    free(list1);
    return result;
}

int yylex() {
    int c = getchar();
    if (isdigit(c)) {
        ungetc(c, stdin);
        int num;
        scanf("%d", &num);
        return NUMBER;
    } else if (c == '+') {
        return PLUS;
    } else if (c == '*') {
        return MULTIPLY;
    } else if (c == 'm' && getchar() == 'a' && getchar() == 'x') {
        return MAXIMUM;
    } else if (c == 'm' && getchar() == 'i' && getchar() == 'n') {
        return MINIMUM;
    } else if (c == '(') {
        return LPAREN;
    } else if (c == ')') {
        return RPAREN;
    } else if (c == EOF) {
        return 0;
    } else {
        fprintf(stderr, "Invalid input\n");
        exit(1);
    }
}

int main() {
    int result = yyparse();
    if (result != 0) {
        fprintf(stderr, "Parsing error\n");
        return 1;
    }
    return 0;
}

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