**Name: Sparsh Karna**

**Reg Number: 23BDS1172**

**Lab Excercise: 6**

**Aim:**

To validate a given arithmetic expression based on specific syntactic rules.

**Procedure:**

1. To check the validity of the expression, analyze each character sequentially to ensure the expression follows the syntax rules.
2. Start by ignoring leading whitespaces.
3. Verify that the first non-space character is not an invalid operator (only unary minus is allowed at the start).
4. Initialize counters to keep track of parentheses and operators.
5. Iterate through each character:
   * Skip spaces.
   * Increment or decrement parentheses counters on encountering '(' or ')'.
   * Check for balanced parentheses (no closing parenthesis without an opening one).
   * Detect operators and verify they are placed correctly (no consecutive operators except unary minus after '(').
   * Allow only alphanumeric characters, operators, and parentheses.
   * If any invalid character or invalid placement is found, mark expression invalid.
6. Ensure the expression does not end with an operator and parentheses are balanced.

**Algorithm:**

1. Read the expression input from the user.
2. Skip all leading whitespace characters.
3. If the first non-space character is an operator other than '-', return invalid.
4. Initialize variables:
   1. paren\_count to zero for tracking parentheses
   2. last\_was\_operator and last\_was\_operand flags
5. Loop through each character of the expression:
   1. If whitespace, continue.
   2. If '(', increment paren\_count, set last\_was\_operator to true.
   3. If ')', decrement paren\_count and check if negative (invalid), set last\_was\_operand true.
   4. If operator:
      1. If '=' operator is first or follows another operator, invalid.
      2. If consecutive operators, only allow '-' after '('.
      3. Set last\_was\_operator true.
   5. If alphanumeric character:
      1. Set last\_was\_operand true and last\_was\_operator false.
   6. Otherwise, invalid character found.
6. After loop ends, check:
   1. paren\_count must be zero
   2. Expression must not end with an operator
7. Return valid if all conditions are met.

**Program:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int is\_operator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '=');

}

int validate\_expression(const char \*expr) {

int i = 0;

int len = strlen(expr);

int paren\_count = 0;

int last\_was\_operator = 0;

int last\_was\_operand = 0;

while (i < len && isspace(expr[i])) i++;

if (is\_operator(expr[i]) && expr[i] != '-') {

return 0;

}

for (; i < len; i++) {

char c = expr[i];

if (isspace(c)) {

continue;

}

else if (c == '(') {

paren\_count++;

last\_was\_operator = 1;

last\_was\_operand = 0;

}

else if (c == ')') {

paren\_count--;

if (paren\_count < 0) return 0;

last\_was\_operator = 0;

last\_was\_operand = 1;

}

else if (is\_operator(c)) {

if (c == '=') {

if (last\_was\_operator || i == 0) return 0;

}

if (last\_was\_operator) {

if (!(c == '-' && expr[i-1] == '(')) {

return 0;

}

}

last\_was\_operator = 1;

last\_was\_operand = 0;

}

else if (isalnum(c)) {

last\_was\_operator = 0;

last\_was\_operand = 1;

}

else {

return 0;

}

}

if (last\_was\_operator) return 0;

if (paren\_count != 0) return 0;

return 1;

}

int main() {

char expr[256];

printf("Enter expression: ");

fgets(expr, sizeof(expr), stdin);

expr[strcspn(expr, "\n")] = 0;

if (validate\_expression(expr)) {

printf("Expression is valid.\n");

} else {

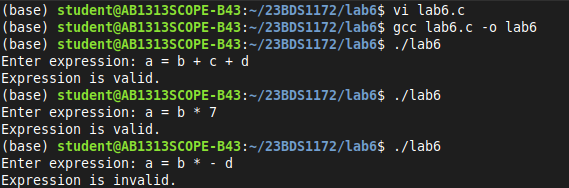
printf("Expression is invalid.\n");

}

return 0;

}

**Output:**

****

**Result:**

The program successfully validates arithmetic expressions by checking operator placement, balanced parentheses, and valid characters. It reliably identifies syntactically correct and incorrect expressions.