Project 1

Jiawei Xiong　　SID：12011022

Introduction

This program is a calculator for big floating-point numbers, utilizing the GMP (GNU Multiple Precision) library for handling large numbers. It supports basic arithmetic operations such as addition, subtraction, multiplication, and division, and can process floating-point number inputs.

Design Decisions

- The program uses the GMP library for its high-precision arithmetic capabilities.
- The default precision is set to 1024 bits to ensure sufficient precision for large numbers.
- Output of decimal digits uses the `%.*g` format to dynamically adapt to the desired number of decimal places.

Code Details

1.import libraries and define 'DBL_DIG 15' .

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmp.h>
#include <float.h>
#define DBL_DIG         15
```

`#define DBL_DIG 15` is hardcoded in order to ensure portability of the program across different compilation environments. The value of this macro is used to format the number of decimal places in the output to display the result adaptively.

The value of `DBL_DIG` is usually 15, indicating that the maximum precision that can be guaranteed for a double-precision floating-point number is 15 bits. The significance of this value is that for double-precision floating-point numbers, at least 15 significant digits are guaranteed.

2.Declare function

```
void add(mpf_t result, const mpf_t a, const mpf_t b);
void subtract(mpf_t result, const mpf_t a, const mpf_t b);
void multiply(mpf_t result, const mpf_t a, const mpf_t b);
void divide(mpf_t result, const mpf_t a, const mpf_t b);
```

- `mpf_t` is a data type in the GNU Multiple Precision (GMP) library used to represent large floating-point numbers.
- It is a structure that contains data members for storing the floating-point number.

- In the program, `mpf_t result, operand, value;` declares three variables of type `mpf_t` named `result`, `operand`, and `value`.

3.Read-in

```
int main() {
    mpf_set_default_prec(1024);
    mpf_t result, operand, value;
    mpf_inits(result, operand, value, NULL);

    printf("Calculator - type 'quit' to exit\n");

    char input[100];

    while (1) {
        printf("> ");
        if (fgets(input, sizeof(input), stdin) == NULL) {
            break;
        }
        // Remove newline character
        input[strcspn(input, "\n")] = '\0';

        // Check if it's the quit command
        if (strcmp(input, "quit") == 0) {
            break;
        }

        char op;
        double d_operand, d_value;
        if (sscanf(input, "%lf %c %lf", &d_operand, &op, &d_value) == 3) {
            mpf_set_d(operand, d_operand);
            mpf_set_d(value, d_value);
        }
    }
    // Rest of the code...
}
```

- **Initialization:**
  - `mpf_set_default_prec(1024);` sets the default precision for the GMP library to 1024 bits.
  - `mpf_t result, operand, value;` declares three variables (`result`, `operand`, and `value`) of type `mpf_t` for large floating-point numbers and initializes them using `mpf_inits`.
- **Input Loop:**
  - The program enters a loop (`while (1)`) to repeatedly prompt the user for input.
  - `printf("> ");` displays the prompt.
  - `fgets(input, sizeof(input), stdin)` reads the user's input, and `input[strcspn(input, "\n")] = '\0';` removes the newline character from the input.
- **Quit Command Check:**

- The program checks if the input is the quit command (`strcmp(input, "quit") == 0`). If so, it breaks out of the loop and exits the program.

- **Parsing Input:**
  - `sscanf(input, "%lf %c %lf", &d_operand, &op, &d_value) == 3` attempts to parse the input as a double (`%lf`) followed by a character (`%c`) followed by another double. If successful, it sets the values of `d_operand`, `op`, and `d_value`.
- **Conversion to GMP Types:**
  - `mpf_set_d(operand, d_operand);` and `mpf_set_d(value, d_value);` convert the parsed double values to GMP's `mpf_t` type.

4. Calculation Options

```
switch (op) {
                case '+':add(result, operand, value);break;
                case '-':subtract(result, operand, value);break;
                case '*':multiply(result, operand, value);break;
                case '/':
                    if (mpf_cmp_ui(value, 0) != 0) {divide(result, operand, value);}
                    else {printf("Error: Division by zero is not allowed.\n");}
                    break;
                default:
                    printf("Error: Invalid operator '%c'\n", op);
                    break;
            }
             gmp_printf("%.*g %c %.*g = %.*g\n",
                        DBL_DIG, mpf_get_d(operand), op,
                        DBL_DIG, mpf_get_d(value), DBL_DIG, mpf_get_d(result));
        } }
        else {
            printf("Error: Invalid input format.\n");
        }
```

- **Switch Statement:**
  - The `switch` statement evaluates the value of `op`, which represents the operator entered by the user (`+`, `-`, `*`, or `/`).
- **Division Check:**
  - In the case of division (`case '/'`), it checks if the divisor (`value`) is not equal to zero using `mpf_cmp_ui(value, 0) != 0`.
  - If the divisor is not zero, it calls the `divide` function to perform the division.
  - If the divisor is zero, it prints an error message stating that division by zero is not allowed.
- **Default Case:**
  - The `default` case of the `switch` statement handles the scenario where the operator entered by the user is not one of the expected operators. It prints an error message indicating that the operator is invalid.
- **Error Handling:**

- If the user's input cannot be parsed correctly (the `sscanf` condition is not met), it prints an error message stating that the input format is invalid.

5. Function Body

```
void add(mpf_t result, const mpf_t a, const mpf_t b) {
    mpf_add(result, a, b);
}
void subtract(mpf_t result, const mpf_t a, const mpf_t b) {
    mpf_sub(result, a, b);
}
void multiply(mpf_t result, const mpf_t a, const mpf_t b) {
    mpf_mul(result, a, b);
}
void divide(mpf_t result, const mpf_t a, const mpf_t b) {
    mpf_div(result, a, b);
}
```

Using the `mpf_t` type in GMP provides a larger range and higher precision representation of numbers in mathematical calculations, enabling programs to handle more complex computational problems.

Result

Compile the .C file and GMP library by `-lgmp`

```
xiong@DESKTOP-5DRLVJO:/mnt/c/Users/Lenovo/Desktop/C++/project1$ gcc calculator.c -o calculator -lgmp
```

1 Basic Arithmetic Operations: The calculator supports standard arithmetic operations, including addition, subtraction, multiplication, and division. Users can input expressions like "89+ 77" or "12 - 80" to obtain accurate results.

```
xiong@DESKTOP-5DRLVJO:/mnt/c/Users/Lenovo/Desktop/C++/project1$ ./calculator
Calculator - type 'quit' to exit
> 89+77
89 + 77 = 166
> 12-80
12 - 80 = -68
> 22*13
22 * 13 = 286
> 18/6
18 / 6 = 3
```

2.2 Continuous Calculation: The calculator enables users to perform multiple calculations consecutively without restarting the program. Users can input expressions one after another until they decide to exit by entering the command "quit."

2.3 Error Diagnosis: The calculator includes robust error handling. It detects and communicates errors such as division by zero or incorrect input format, providing informative error messages to guide users in correcting their inputs.

```
> 12/0
Error: Division by zero is not allowed.
> we12+21
Error: Invalid input format.
```

2.4 Scientific Notation: Scientific notation is supported for both input and output. Users can perform calculations with numbers expressed in scientific notation, allowing for a more versatile and convenient user experience.

2.5 Large Numeric Computations: The calculator leverages the GNU Multiple Precision Arithmetic Library (GMP) to handle large numeric values efficiently. This capability is particularly beneficial for computations involving extremely large numbers, such as "1.0e200 * 1.0e200," which may exceed the range of standard numeric types

```
> 987654321*987654321
987654321 * 987654321 = 9.75461057789971e+17
> 1.2e21*1.2e21
1.2e+21 * 1.2e+21 = 1.44e+42
> 1.0e20*1.0e200
1e+20 * 1e+200 = 1e+220
> quit
```

*** Text generation has the help of ChatGPT