Calculator Program Report

Introduction

This calculator program is a versatile and robust application designed to perform various mathematical operations. It is capable of reading input directly from the command line or through arguments, providing flexibility to the user.

Functionality

The calculator supports five primary operations:

1. Addition (`+`)
2. Subtraction (`-`)
3. Multiplication (`*`)
4. Division (`/`)
5. Exponentiation (`^`)

These operations can be performed on any two numbers provided as input.

Features

1. **Operator Precedence**: The calculator respects the standard order of operations (PEMDAS), ensuring that expressions are evaluated correctly.

   ```
   >> 5*3+7/2+3*2
   24.5
   ```

   Relevant code:

   ```
   do
   {
       if (prevIsOp && !regexec(&numRegex, input + offset, 1, pmatch, 0) &&
           pmatch[0].rm_so == 0)
       {
           DEBUG_PRINT("Match: %.*s\n", pmatch[0].rm_eo - pmatch[0].rm_so,
                       input + offset + pmatch[0].rm_so);
           elements[index].type = NUMBER;
           elements[index].number = atof(input + offset + pmatch[0].rm_so);
           offset += pmatch[0].rm_eo;
           prevIsOp = false;
       }
       else if (!prevIsOp && !regexec(&opRegex, input + offset, 1, pmatch,
   0) &&
                pmatch[0].rm_so == 0)
       {
           DEBUG_PRINT("Match: %.*s\n", pmatch[0].rm_eo - pmatch[0].rm_so,
                       input + offset + pmatch[0].rm_so);
           elements[index].type = OPERATOR;
           elements[index].op = input[offset];
           elements[index].opPriority = opPriority(input[offset]);
           offset += pmatch[0].rm_eo;
   ```

```
        prevIsOp = true;
    }
    else if (input[offset] == '(' && prevIsOp)
    {
        DEBUG_PRINT("Match: (\n");
        elements[index].type = LEFT_BRACKET;
        offset++;
        bracketCount++;
    }
    else if (input[offset] == ')' && !prevIsOp)
    {
        DEBUG_PRINT("Match: )\n");
        elements[index].type = RIGHT_BRACKET;
        offset++;
        bracketCount--;
    }
    else
    {
        printf("The input cannot be interpret as an expression.\n");
        return 1;
    }
    index++;
} while (input[offset] != '\0');
if (bracketCount != 0 || prevIsOp)
{
    printf("The input cannot be interpret as an expression.\n");
    return 1;
}
```

2. **Bracket Support**: The calculator can handle expressions with brackets, allowing for more complex calculations. For example, `(3 + 2) * 4` yields `20`, not `14`.

```
>> (11^(2+12/4)-54+6*3)+4-(4+2)*3
161001
```

3. **Input Validation**: The program is equipped with input validation mechanisms. It checks the validity of the input before performing any operations. This feature ensures that the program doesn't crash due to invalid input and provides a user-friendly experience.

```
>> 6*((5+6)*4-4
The input cannot be interpret as an expression.
>> 24+(4++9)*2
The input cannot be interpret as an expression.
>> 4*5+-2
18
```

Relevant code:

```
    while (elements[index].type != 0)
{
    if (elements[index].type == NUMBER)
    {
        push(&numStack, &elements[index]);
    }
    else if (elements[index].type == OPERATOR)
```

```
    {
        if (isEmpty(&opStack) ||
            elements[index].opPriority > peek(&opStack)->opPriority ||
            peek(&opStack)->type == LEFT_BRACKET)
        {
            push(&opStack, &elements[index]);
        }
        else
        {
            while (!isEmpty(&opStack) && peek(&opStack)->type == OPERATOR &&
                   elements[index].opPriority <= peek(&opStack)->opPriority)
            {
                if (performOperation(&numStack, &opStack))
                {
                    return;
                }
            }
            push(&opStack, &elements[index]);
        }
    }
    else if (elements[index].type == LEFT_BRACKET)
    {
        push(&opStack, &elements[index]);
    }
    else if (elements[index].type == RIGHT_BRACKET)
    {
        while (peek(&opStack)->type != LEFT_BRACKET)
        {
            if (performOperation(&numStack, &opStack))
            {
                return;
            }
        }
        pop(&opStack);
    }
    index++;
}
```

4. **Big Number Calculation**: The calculator supports large number calculations, thanks to the GNU Multiple Precision Arithmetic Library (GMP) and the GNU MPFR Library. This feature allows the program to handle and perform operations on numbers of any size with acceptable precision loss. You can enable this feature by uncommenting the `#define GMP` line in the `calculator.c` file.

```
>> 3e40-4e24
0.29999999999999996e41
>> 123456789*54321
0.6706296235269e13
```

Relevant code:

```
    switch (op->op)
{
case '+':
```

```
    mpf_add(num2->number, num2->number, num1->number);
    break;
case '-':
    mpf_sub(num2->number, num2->number, num1->number);
    break;
case '*':
    mpf_mul(num2->number, num2->number, num1->number);
    break;
case '/':
    if (mpf_cmp_ui(num1->number, 0) == 0)
    {
        printf("A number cannot be divied by zero.\n");
        return 1;
    }
    mpf_div(num2->number, num2->number, num1->number);
    break;
case '^':
    if (mpf_integer_p(num1->number))
    {
        mpf_pow_ui(num2->number, num2->number, mpf_get_ui(num1->number));
    }
    else
    {
        mpfr_init2(tempExp, PRECISION);
        mpfr_init2(tempBase, PRECISION);
        mpfr_set_f(tempExp, num1->number, MPFR_RNDN);
        mpfr_set_f(tempBase, num2->number, MPFR_RNDN);
        mpfr_pow(tempBase, tempBase, tempExp, MPFR_RNDN);
        mpfr_get_f(num2->number, tempBase, MPFR_RNDN);
        mpfr_clear(tempExp);
        mpfr_clear(tempBase);
    }
    break;
default:
    break;
}
```

It also implements a simple big number struct which can do basic add and sub.

```
typedef struct
{
int sign;
char *number;
} BigNumber;
void add(BigNumber *num1, BigNumber *num2, BigNumber *result);
void sub(BigNumber *num1, BigNumber *num2, BigNumber *result);
void add(BigNumber *num1, BigNumber *num2, BigNumber *result)
{
int len1 = strlen(num1->number);
int len2 = strlen(num2->number);
if (num1->sign == 1 && num2->sign == 1)
{
    result->sign = 1;
}
else if (num1->sign == -1 && num2->sign == -1)
{
```

```
    result->sign = -1;
}
else if (num1->sign == 1 && num2->sign == -1)
{
    num2->sign = 1;
    sub(num1, num2, result);
    num2->sign = -1;
    return;
}
else if (num1->sign == -1 && num2->sign == 1)
{
    num1->sign = 1;
    sub(num2, num1, result);
    num1->sign = -1;
    return;
}

int carry = 0;
int i = 0;
int j = 0;
int k = 0;
int sum = 0;
int maxLen = len1 > len2 ? len1 : len2;
char *temp = (char *)malloc(maxLen + 2);
for (i = len1 - 1, j = len2 - 1, k = maxLen; i >= 0 && j >= 0; i--, j--, k--)
{
    sum = (num1->number[i] - '0') + (num2->number[j] - '0') + carry;
    temp[k] = (sum % 10) + '0';
    carry = sum / 10;
}
while (i >= 0)
{
    sum = (num1->number[i] - '0') + carry;
    temp[k] = (sum % 10) + '0';
    carry = sum / 10;
    i--;
    k--;
}
while (j >= 0)
{
    sum = (num2->number[j] - '0') + carry;
    temp[k] = (sum % 10) + '0';
    carry = sum / 10;
    j--;
    k--;
}
if (carry > 0)
{
    temp[k] = carry + '0';
    k--;
}
result->number = (char *)malloc(maxLen + 2 - k);
strcpy(result->number, temp + k + 1);
free(temp);
}
void sub(BigNumber *num1, BigNumber *num2, BigNumber *result)
{
```

```
int len1 = strlen(num1->number);
int len2 = strlen(num2->number);
if (num1->sign != num2->sign)
{
    num2->sign = num1->sign;
    add(num1, num2, result);
    return;
}
//meke the num1 bigger one
if (len1 < len2 || (len1 == len2 && strcmp(num1->number, num2->number)
< 0))
{
    BigNumber *temp = num1;
    num1 = num2;
    num2 = temp;
    len1 = strlen(num1->number);
    len2 = strlen(num2->number);
    result->sign = -num1->sign;
}
else
{
    result->sign = num1->sign;
}
int borrow = 0;
int i = 0;
int j = 0;
int k = 0;
int diff = 0;
int maxLen = len1 > len2 ? len1 : len2;
char *temp = (char *)malloc(maxLen + 2);
for (i = len1 - 1, j = len2 - 1, k = maxLen; i >= 0 && j >= 0; i--, j-
-, k--)
{
    diff = (num1->number[i] - '0') - (num2->number[j] - '0') - borrow;
    if (diff < 0)
    {
        diff += 10;
        borrow = 1;
    }
    else
    {
        borrow = 0;
    }
    temp[k] = diff + '0';
}
while (i >= 0)
{
    diff = (num1->number[i] - '0') - borrow;
    if (diff < 0)
    {
        diff += 10;
        borrow = 1;
    }
    else
    {
        borrow = 0;
    }
    temp[k] = diff + '0';
```

```
    i--;
    k--;
}
result->number = (char *)malloc(maxLen + 2 - k);
strcpy(result->number, temp + k + 1);
free(temp);
}
```

Conclusion

In summary, this calculator program is a powerful tool for performing mathematical operations on any size of numbers. Its ability to validate input and support for big number calculations using the GMP library sets it apart from standard calculators.