实验三：C语言实现直观算符优先分析法

一、定义

任二个相继出现的终结符a与b(可能中间有V_N),可能有以下优先关系:

1. $a \dot{>} b$， 表示a的算符优先级比b高，应该优先于b进行运算
2. $a \dot{=} b$， 表示a的优先级和b一样高，应该一起进行规约计算
3. $a \dot{<} b$， 表示a的算符优先级比b低，规约的时候应该先规约b

二、优先级表的构造

我们 首先从算术表达式的优先级特点来进行设计。

1. 括号里的表达式优先计算，再算乘方运算，接着算乘除运算，最后才算加减。
2. 当算术表达式中只有1中提到的在四则运算规定中优先级相等的符号中，左边的算符优先级高于右边的。$e.g.\ 3-6+5$减法的优先级比加法高，如果先算加法会导致运算结果不对。
3. 多重指数运算优先计算右边的指数
4. 本次输入映入了终结符进行判断。最后的终结符方便程序的终止，并且如果将终结符的算符优先级设置为最小，也可以将前面没有算完的所有表达式全部算完。开头再加一个终结符能够让程序在符号栈中不含元素的时候不需要额外进行讨论是否为空栈。

设计结果如下（第一列的符号表示当前栈顶的符号，第一行的符号表示当前读到的符号）：

	+	-	*	/	^	(	)	#
+	$\dot{>}$	$\dot{>}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{>}$	$\dot{>}$
-	$\dot{>}$	$\dot{>}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{>}$	$\dot{>}$
*	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{<}$	$\dot{<}$	$\dot{>}$	$\dot{>}$
/	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{<}$	$\dot{<}$	$\dot{>}$	$\dot{>}$
^	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{<}$	$\dot{<}$	$\dot{>}$	$\dot{>}$
(	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{=}$	错误语法
)	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{>}$	$\dot{>}$	无用	$\dot{>}$	$\dot{>}$
#	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	$\dot{<}$	错误语法	$\dot{=}$

优先级表的用法如下：

1. 初始化：一个符号栈和一个数字栈，全为空。
2. 如果读入的使数字，就将数字压入数字栈；如果读入的是符号就先对比当前栈顶符号和读入符号的优先级。如果读入优先级高就直接压入，如果读入的优先级低，就弹出符号栈中的符号和前两个进行运算，把运算的结果再压入数字栈。直到符号栈顶的符号优先级比输入的小即可。如果优先级一样就一起规约。
3. 当栈顶为#且输入的符号为#就一起规约并结束程序。

三、代码实现

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
int float_cur = 0;
int char_cur = 0;
float *nums_expand_float(float *nums, int length)
{
    float *newnums;
    newnums = (float *)malloc(sizeof(float) * length * 2);
    if (newnums == NULL)
    {
        // 处理内存分配失败的情况
        return NULL;
    }
    memcpy(newnums, nums, sizeof(float) * length); // 使用 memcpy 复制数据
    free(nums);
    return newnums;
}
float mypow(float a, int b)
{
    float res = 1;
    while (b >= 1)
    {
        res *= a;
        b--;
    }
    return res;
}

char *symbol_sexpand_char(char *nums, int length)
{
    char *newnums;
    newnums = (char *)malloc(sizeof(char) * length * 2);
    if (newnums == NULL)
    {
        // 处理内存分配失败的情况
        return NULL;
    }
    memcpy(newnums, nums, sizeof(char) * length); // 使用 memcpy 复制数据
    free(nums);
    return newnums;
}

int main()
{
    char *symbol;
    int CHARMAXLENTH = 50;
    int FLOATMAXLENTH = 50;
    symbol = (char *)malloc(sizeof(char) * CHARMAXLENTH); // 空
    float *nums;
    nums = (float *)malloc(sizeof(float) * FLOATMAXLENTH);
    // printf("%f\n", mypow(2, 3));
```

```
    char c = getchar();
    while (1)
    {
        if (char_cur == 0 && c != '#' && float_cur == 0)
        {
            symbol[char_cur] = '#';
            char_cur++;
        }
        if ('0' <= c && '9' >= c)
        {
            float tmp = 0;
            tmp = tmp + c - '0';
            c = getchar();
            while ('0' <= c && '9' >= c)
            {
                tmp *= 10;
                tmp = tmp + c - '0';
                c = getchar();
            }
            if ('.' == c)
            {
                float i = 10;
                float flo_tmp = 0;
                c = getchar();
                while ('0' <= c && '9' >= c)
                {
                    flo_tmp = flo_tmp + (float)(c - '0') / i;
                    i *= 10;
                    c = getchar();
                }
                tmp += flo_tmp;
            }
            if (float_cur == FLOATMAXLENTH)
            {
                nums = nums_expand_float(nums, FLOATMAXLENTH);
                FLOATMAXLENTH *= 2;
            }
            nums[float_cur] = tmp;
            // printf("%f %f\n", nums[float_cur], tmp);
            float_cur++;
            printf("shift: %f\n", tmp);
            // printf("%c\n", c);
        }
        if ('#' == c || c == EOF)
        {
            if (0 == char_cur)
            {
                symbol[char_cur] = c;
                char_cur++;
            }
            else
            {
                while ('#' != symbol[char_cur - 1] && char_cur != 0)
                {
                    if ('+' == symbol[char_cur - 1])
```

```
                    {
                        printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                        nums[float_cur - 2] += nums[float_cur - 1];
                        printf("%f\n", nums[float_cur - 2]);
                        float_cur--;
                        char_cur--;
                    }
                    else if ('-' == symbol[char_cur - 1])
                    {
                        printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                        nums[float_cur - 2] -= nums[float_cur - 1];
                        printf("%f\n", nums[float_cur - 2]);
                        float_cur--;
                        char_cur--;
                    }
                    else if ('*' == symbol[char_cur - 1])
                    {
                        printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                        nums[float_cur - 2] *= nums[float_cur - 1];
                        printf("%f\n", nums[float_cur - 2]);
                        float_cur--;
                        char_cur--;
                    }
                    else if ('/' == symbol[char_cur - 1])
                    {
                        printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                        nums[float_cur - 2] /= nums[float_cur - 1];
                        printf("%f\n", nums[float_cur - 2]);
                        float_cur--;
                        char_cur--;
                    }
                    else if ('^' == symbol[char_cur - 1])
                    {
                        printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                        nums[float_cur - 2] = mypow(nums[float_cur - 2], (int)
(nums[float_cur - 1]));
                        printf("%f\n", nums[float_cur - 2]);
                        float_cur--;
                        char_cur--;
                    }
                    else if ('(' == symbol[char_cur - 1])
                    {
                        printf("expected )\n");
                        free(nums);
                        free(symbol);
                        return 1;
                    }
                }
                if (char_cur == FLOATMAXLENTH)
                {
```

```
                    symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                    CHARMAXLENTH *= 2;
                }
                if (symbol[char_cur - 1] == '#')
                {
                    if (float_cur != 1)
                    {
                        printf("need one more operation\n");
                        free(nums);
                        free(symbol);
                        return 3;
                    }
                    printf("result is %f\nfinish", nums[0]);
                    break;
                }
            }
        }
        if ('+' == c)
        {

            while ('#' != symbol[char_cur - 1] && '(' != symbol[char_cur - 1])
            {
                // printf("+");
                if ('+' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] += nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('-' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] -= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('*' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] *= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('/' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] /= nums[float_cur - 1];
```

```
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('^' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] = mypow(nums[float_cur - 2],
(int)nums[float_cur - 1]);
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
            }
            if (char_cur == FLOATMAXLENTH)
            {
                symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                CHARMAXLENTH *= 2;
            }
            printf("shift: %c\n", c);
            symbol[char_cur] = c;
            char_cur++;
        }
        if ('-' == c)
        {
            while ('#' != symbol[char_cur - 1] && '(' != symbol[char_cur - 1])
            {
                if ('+' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] += nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('-' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] -= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('*' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] *= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
```

```
                else if ('/' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] /= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('^' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] -= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('#' == symbol[char_cur - 1])
                {
                    printf("expected (\n");
                    free(nums);
                    free(symbol);
                    return 2;
                }
            }
            if (char_cur == FLOATMAXLENTH)
            {
                symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                CHARMAXLENTH *= 2;
            }
            printf("shift: %c\n", c);
            symbol[char_cur] = c;
            char_cur++;
        }
        if ('*' == c)
        {
            while ('#' != symbol[char_cur - 1] && '(' != symbol[char_cur - 1] &&
'+' != symbol[char_cur - 1] && '-' != symbol[char_cur - 1])
            {
                if ('*' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] *= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('/' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] /= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
```

```
                    float_cur--;
                    char_cur--;
                }
                else if ('^' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] -= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
            }
            if (char_cur == FLOATMAXLENTH)
            {
                symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                CHARMAXLENTH *= 2;
            }
            printf("shift: %c\n", c);
            symbol[char_cur] = c;
            char_cur++;
        }
        if ('/' == c)
        {
            while ('#' != symbol[char_cur - 1] && '(' != symbol[char_cur - 1] &&
'+' != symbol[char_cur - 1] && '-' != symbol[char_cur - 1])
            {
                if ('*' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] *= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('/' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] /= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('^' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] -= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
            }
```

```
            if (char_cur == FLOATMAXLENTH)
            {
                symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                CHARMAXLENTH *= 2;
            }
            printf("shift: %c\n", c);
            symbol[char_cur] = c;
            char_cur++;
        }
        if ('^' == c || '(' == c)
        {
            if (char_cur == FLOATMAXLENTH)
            {
                symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                CHARMAXLENTH *= 2;
            }
            printf("shift: %c\n", c);
            symbol[char_cur] = c;
            char_cur++;
            // printf("%c\n", symbol[char_cur - 1]);
        }
        if (')' == c)
        {
            while ('(' != symbol[char_cur - 1])
            {
                if ('+' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1]);
                    nums[float_cur - 2] += nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('-' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] -= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('*' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] *= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('/' == symbol[char_cur - 1])
                {
```

```
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] /= nums[float_cur - 1];
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if ('^' == symbol[char_cur - 1])
                {
                    printf("Reduce: %f %c %f=", nums[float_cur - 2],
symbol[char_cur - 1], nums[float_cur - 1]);
                    nums[float_cur - 2] = mypow(nums[float_cur - 2],
(int)nums[float_cur - 1]);
                    printf("%f\n", nums[float_cur - 2]);
                    float_cur--;
                    char_cur--;
                }
                else if (char_cur == 1)
                {
                    printf("expected (\n");
                    free(nums);
                    free(symbol);
                    return 4;
                }
                        }
            if (char_cur == FLOATMAXLENTH)
            {
                symbol = symbol_sexpand_char(symbol, CHARMAXLENTH);
                CHARMAXLENTH *= 2;
            }
            printf("Reduce: (%d)\n", nums[float_cur - 1]);
            char_cur--;
        }

        c = getchar();
        // printf("c=%c\n", c);
    }
    free(symbol);
    free(nums);

    return 0;
}
```

另外多加了一个扩展栈区的函数，当检测到满栈的时候，就会触发对应的扩展函数，将空间扩大到原先的两倍。

四、实验结果

```
PS F:\kingwood\kingwood\快速访问\大三下\CS15445> ./mybison
#1+2+4+5*1+2*(1-2+3^2)/7#
shift: 1.000000
shift: +
shift: 2.000000
Reduce: 1.000000 + 2.000000=3.000000
shift: +
shift: 4.000000
Reduce: 3.000000 + 4.000000=7.000000
shift: +
shift: 5.000000
shift: *
shift: 1.000000
Reduce: 5.000000 * 1.000000=5.000000
Reduce: 7.000000 + 5.000000=12.000000
shift: +
shift: 2.000000
shift: *
shift: (
shift: 1.000000
shift: -
shift: 2.000000
Reduce: 1.000000 - 2.000000=-1.000000
shift: +
shift: 3.000000
shift: ^
shift: 2.000000
Reduce: 3.000000 ^ 2.000000=9.000000
Reduce: -1.000000 + 2.000000=8.000000
Reduce: (0)
Reduce: 2.000000 * 8.000000=16.000000
shift: /
shift: 7.000000
Reduce: 16.000000 / 7.000000=2.285714
Reduce: 12.000000 + 2.285714=14.285714
result is 14.285714
finish
```

将最大长度改为10并且运行更长的输入

```
PS F:\kingwood\kingwood\快速访问\大三下\CS15445> ./mybison
#1+2+4+7+8+9+5+6+4+1+7-8-9-6-4-1*8+8/7+2^2+8*(1-3)+(6+4)/3#
shift: 1.000000
shift: +
shift: 2.000000
Reduce: 1.000000 + 2.000000=3.000000
shift: +
shift: 4.000000
Reduce: 3.000000 + 4.000000=7.000000
shift: +
shift: 7.000000
Reduce: 7.000000 + 7.000000=14.000000
shift: +
shift: 8.000000
Reduce: 14.000000 + 8.000000=22.000000
shift: +
shift: 9.000000
Reduce: 22.000000 + 9.000000=31.000000
shift: +
shift: 5.000000
Reduce: 31.000000 + 5.000000=36.000000
shift: +
shift: 6.000000
Reduce: 36.000000 + 6.000000=42.000000
shift: +
shift: 4.000000
Reduce: 42.000000 + 4.000000=46.000000
shift: +
shift: 1.000000
Reduce: 46.000000 + 1.000000=47.000000
shift: +
shift: 7.000000
Reduce: 47.000000 + 7.000000=54.000000
shift: -
shift: 8.000000
Reduce: 54.000000 - 8.000000=46.000000
```

```
shift: -
shift: 8.000000
Reduce: 54.000000 - 8.000000=46.000000
shift: -
shift: 9.000000
Reduce: 46.000000 - 9.000000=37.000000
shift: -
shift: 6.000000
Reduce: 37.000000 - 6.000000=31.000000
shift: -
shift: 4.000000
Reduce: 31.000000 - 4.000000=27.000000
shift: -
shift: 1.000000
shift: *
shift: 8.000000
Reduce: 1.000000 * 8.000000=8.000000
Reduce: 27.000000 - 8.000000=19.000000
shift: +
shift: 8.000000
shift: /
shift: 7.000000
Reduce: 8.000000 / 7.000000=1.142857
Reduce: 19.000000 + 1.142857=20.142857
shift: +
shift: 2.000000
shift: ^
shift: 2.000000
Reduce: 2.000000 ^ 2.000000=4.000000
Reduce: 20.142857 + 4.000000=24.142857
shift: +
shift: 8.000000
shift: *
shift: (
shift: 1.000000
shift: -
shift: 3.000000
```

```
shift: +
shift: 2.000000
shift: ^
shift: 2.000000
Reduce: 2.000000 ^ 2.000000=4.000000
Reduce: 20.142857 + 4.000000=24.142857
shift: +
shift: 8.000000
shift: *
shift: (
shift: 1.000000
shift: -
shift: 3.000000
Reduce: 1.000000 - 3.000000=-2.000000
Reduce: (0)
Reduce: 8.000000 * -2.000000=-16.000000
Reduce: 24.142857 + -16.000000=8.142857
shift: +
shift: (
shift: 6.000000
shift: +
shift: 4.000000
Reduce: 6.000000 + -16.000000=10.000000
Reduce: (0)
shift: /
shift: 3.000000
Reduce: 10.000000 / 3.000000=3.333333
Reduce: 8.142857 + 3.333333=11.476190
result is 11.476190
finish
```

发现不会溢出报错。

五、扩展功能

能够识别非法算数表达式，包括（）不匹配，缺少运算符。并且能够自动补全开始的符号和结束的符号。

补全开头：

```
1+2*3+3*(1-2)#
shift: 1.000000
shift: +
shift: 2.000000
shift: *
shift: 3.000000
Reduce: 2.000000 * 3.000000=6.000000
Reduce: 1.000000 + 6.000000=7.000000
shift: +
shift: 3.000000
shift: *
shift: (
shift: 1.000000
shift: -
shift: 2.000000
Reduce: 1.000000 - 2.000000=-1.000000
Reduce: (0)
Reduce: 3.000000 * -1.000000=-3.000000
Reduce: 7.000000 + -3.000000=4.000000
result is 4.000000
finish
```

补全结尾：

```
#1+2*3+3*(1-2)
shift: 1.000000
shift: +
shift: 2.000000
shift: *
shift: 3.000000
Reduce: 2.000000 * 3.000000=6.000000
Reduce: 1.000000 + 6.000000=7.000000
shift: +
shift: 3.000000
shift: *
shift: (
shift: 1.000000
shift: -
shift: 2.000000
Reduce: 1.000000 - 2.000000=-1.000000
Reduce: (0)
^Z
Reduce: 3.000000 * -1.000000=-3.000000
Reduce: 7.000000 + -3.000000=4.000000
result is 4.000000
finish
```

识别缺少左括号

```
PS F:\kingwood\kingwood\快速访问\大三下\CS15445> ./mybison
#1+2)#
shift: 1.000000
shift: +
shift: 2.000000
Reduce: 1.000000 + 0.000000=3.000000
expected (
```

识别缺少右括号

```
PS F:\kingwood\kingwood\快速访问\大三下\CS15445> ./mybison
#1+2*(3+4))#
shift: 1.000000
shift: +
shift: 2.000000
shift: *
shift: (
shift: 3.000000
shift: +
shift: 4.000000
Reduce: 3.000000 + 0.000000=7.000000
Reduce: (0)
Reduce: 2.000000 * 7.000000=14.000000
Reduce: 1.000000 + 7.000000=15.000000
expected (
```

识别缺少运算符

```
PS F:\kingwood\kingwood\快速访问\大三下\CS15445> ./mybison
1+2(1+3)#
shift: 1.000000
shift: +
shift: 2.000000
shift: (
shift: 1.000000
shift: +
shift: 3.000000
Reduce: 1.000000 + 0.000000=4.000000
Reduce: (0)
Reduce: 2.000000 + 4.000000=6.000000
need one more operation
```

六、实验中遇到的问题和缺陷

用到指数运算的时候第二个参数要是整数，需要强制转换；第一个参数是浮点数，需要转化成浮点数。

缺陷，没有非法文法 的匹配，容易出现因为输入错误导致的死循环。