信息与软件工程学院

项目报告

课程名称：程序设计项目实践（PBLF）

学　　期：2024-2025 第 1 学期

项目名称：计算器的设计与实现

授课教师：张学

序号	学号	姓名
1（组长）	2024091202009	王比乐
2	2024091202006	陈奕霖
3	2024091202007	孟响
4	2024091202008	韩侨
5	2024091202010	杨寓杰
6		

目录

1 项目简介

1.1 考核方式

总成绩 = 项目和项目文档成绩(40%) + 汇报幻灯片成绩(20%)
+ 表达能力(20%) + 团队合作(20%)

1.2 项目题目及内容简介

计算器的设计与实现

该项目利用 C 语言来实现一个简单的计算器，依托于一个独立的栈文件（包括一个泛型栈和通过独立链表实现的链式栈，二者可以随意切换），可进行四则运算、乘方运算，并且拥有完整的词法分析、支持各种函数计算，可以直接运行多次计算、也可进行批量计算即从文件中输入大量计算式再输出到目标文件中，同时程序是健壮的、能够处理几乎所有错误。

1.3 项目组成员与分工

陈奕霖：负责编写独立的泛型栈、链表、用链表实现的栈，并写出测试文件，确保链表和栈是正确的。

孟响：依托于栈实现计算器的基本功能。

韩侨：实现两个主函数，做到能够直接运行多次计算和批量计算。

杨寓杰：对已经写好的程序添加完整的找错机制。

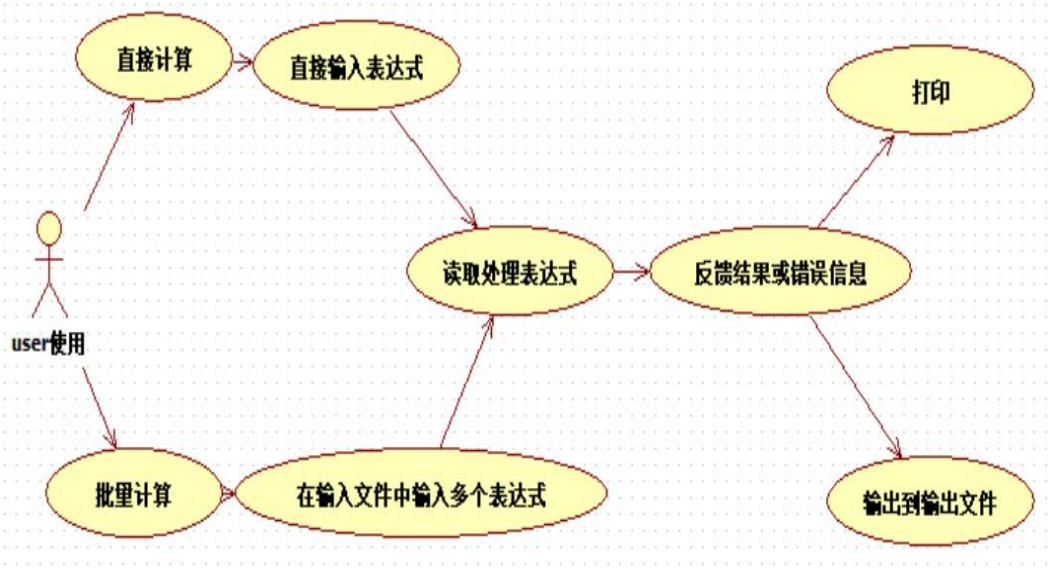
王比乐：将每位组员的成果集合为一个完整的项目，检查代码的规范性，优化代码，调试程序。

2 需求分析

2.1 选题的依据

题目是由老师指定的。

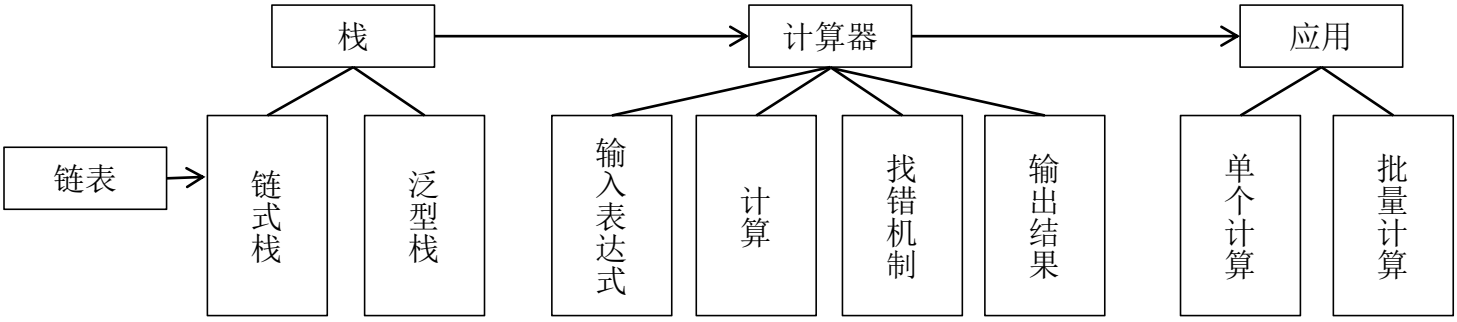
2.2 功能需求



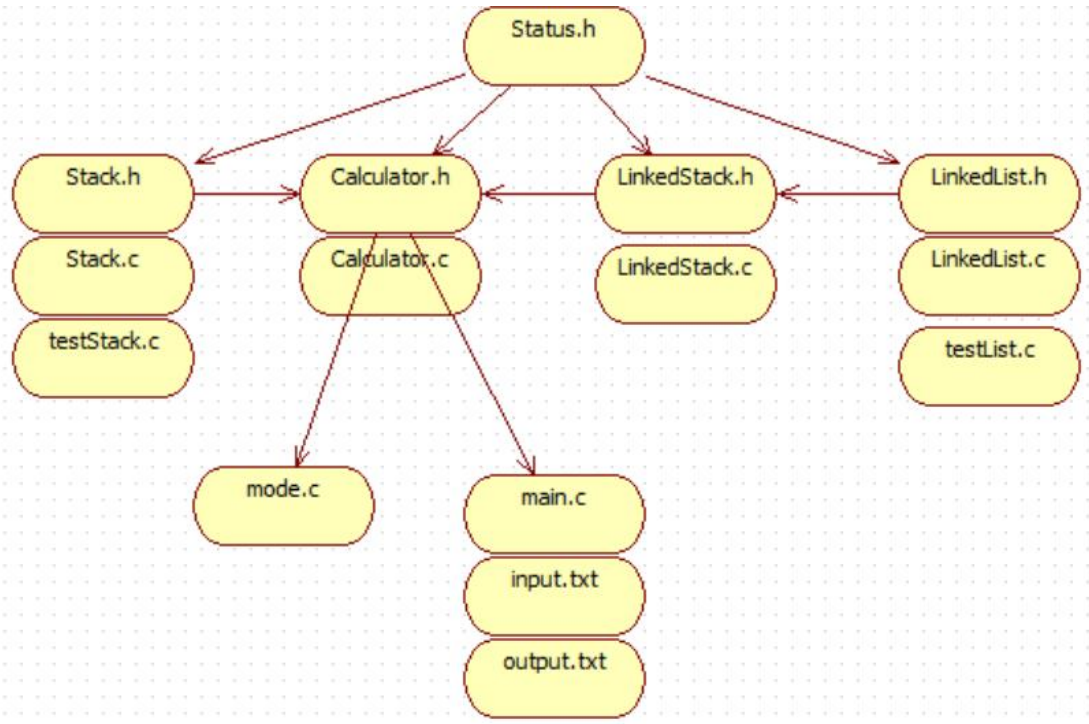
满足用户单个计算式计算和批量计算的需求。

3 系统设计

3.1 总体设计



3.2 模块设计



多个文件之间的关系。

Status.h 是整个项目公用的头文件。

mode.c 中包含实现单个计算的主函数。

main.c 中包含实现批量计算的主函数（input.txt 中输入，output.txt 中输出）。

Calculator.h 和 Calculator.c 是依托于栈实现的计算器主体。

以下给出了 Stack.h 和 Stack.c（泛型栈）中函数的声明和实现。

```c
#ifndef STACK_H
#define STACK_H

#include<stdbool.h>
#include"Status.h"

#define STACK_SIZE 100

typedef struct Stack{
    void* pBase;
    void* pTop;
    int elementSize;
    int stackSize;
}Stack;
typedef Status (*FreeData)(void*);

Stack* StackConstruct(int sizeOfElement);
Status StackDestruct(Stack* pStack,FreeData freeData);
bool IsStackEmpty(Stack* pStack);
int StackLength(Stack* pStack);
Status StackPush(Stack* pStack,void* pElem);
Status StackPop(Stack* pStack,void* pElem,FreeData freeData);
Status StackGetTop(Stack* pStack,void* pElem);
Status StackTraverse(Stack* pStack,Status (*visit)(void*));
Status StackClear(Stack* pStack,FreeData freeData);

#endif
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include"Stack.h"

#define DEBUG 0

Stack* StackConstruct(int sizeOfElement){
    Stack *pStack;
    pStack=(Stack*)malloc(sizeof(Stack));
    if(pStack==NULL){
        return NULL;
    }

    pStack->pBase=malloc(STACK_SIZE*sizeOfElement);
    if(pStack->pBase==NULL){
        free(pStack);
        return NULL;
    }
    pStack->pTop=pStack->pBase;
    pStack->elementSize=sizeOfElement;
    pStack->stackSize=STACK_SIZE;

    return pStack;
}

Status StackDestruct(Stack* pStack,FreeData freeData){
    if(pStack==NULL){
        return OK;
    }
    StackTraverse(pStack,freeData);
    free(pStack->pBase);
    free(pStack);
    return OK;
}
```

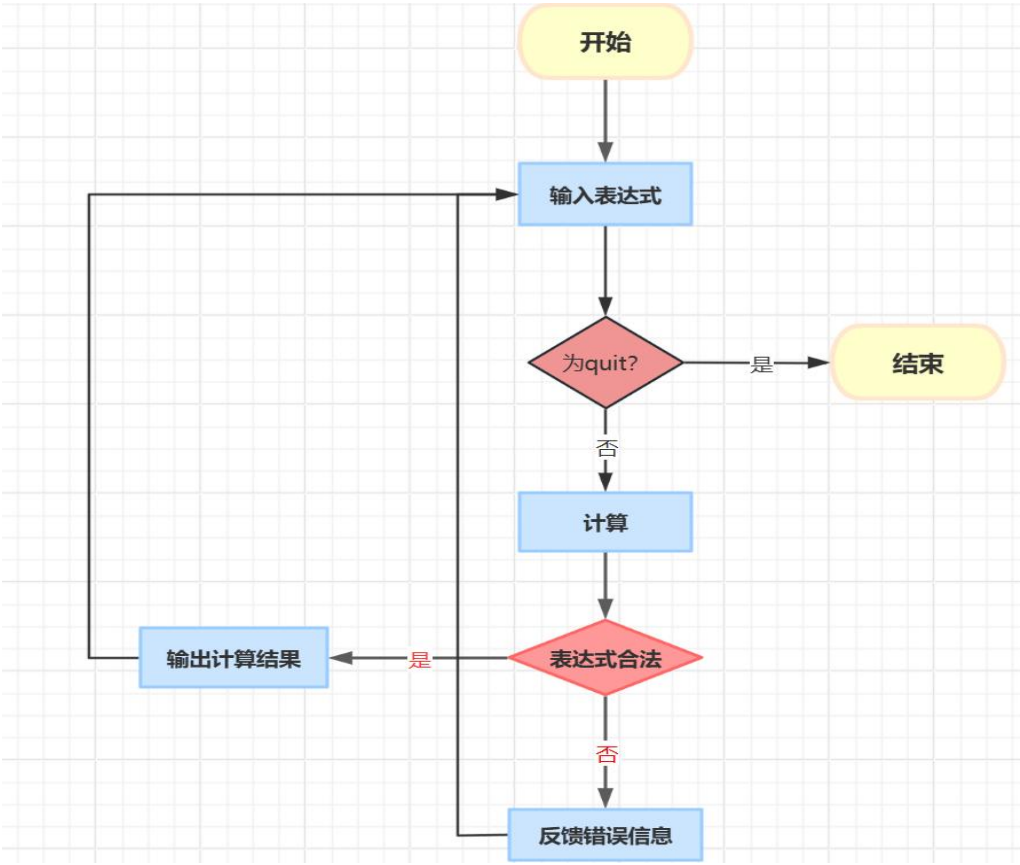
```c
bool IsStackEmpty(Stack* pStack){
    if(pStack->pBase==pStack->pTop){
        return true;
    }else{
        return false;
    }
}

int StackLength(Stack* pStack){
    return ((pStack->pTop)-(pStack->pBase))/pStack->elementSize;
}

Status StackPush(Stack* pStack,void* pElem){
    if(StackLength(pStack)>=pStack->stackSize){
        (pStack->stackSize)+=STACK_SIZE;
        pStack->pBase=realloc(pStack->pBase,(pStack->stackSize)*(pStack->elementSize));
        if(pStack->pBase==NULL){
            return ERROR;
        }
        pStack->pTop=pStack->pBase+(pStack->stackSize-STACK_SIZE)*(pStack->elementSize);
    }

    if(pStack->pBase==NULL){
        return ERROR;
    }

    memcpy(pStack->pTop,pElem,pStack->elementSize);
    pStack->pTop+=pStack->elementSize;

    #if DEBUG
        printf("%d ",StackLength(pStack));
    #endif

    return OK;
}
```

```c
Status StackPop(Stack* pStack,void* pElem,FreeData freeData){
    if(IsStackEmpty(pStack)){
        return ERROR;
    }else{
        pStack->pTop-=pStack->elementSize;
        memcpy(pElem,pStack->pTop,pStack->elementSize);
        freeData(pStack->pTop);
        return OK;
    }
}

Status StackGetTop(Stack* pStack,void* pElem){
    if(IsStackEmpty(pStack)){
        return ERROR;
    }
    memcpy(pElem,pStack->pTop-pStack->elementSize,pStack->elementSize);
    return OK;
}

Status StackTraverse(Stack* pStack,Status (*visit)(void*)){
    int i;
    void* pTemp=pStack->pBase;
    for(i=0;i<StackLength(pStack);i++){
        if(visit(pTemp)==ERROR){
            return ERROR;
        }
        pTemp+=pStack->elementSize;
    }
    return OK;
}

Status StackClear(Stack* pStack,FreeData freeData){
    if(StackTraverse(pStack,freeData)==ERROR){
        return ERROR;
    }
    pStack->pTop=pStack->pBase;
    return OK;
}
```

4 系统实现

4.1 主函数

主函数 1：mode.c

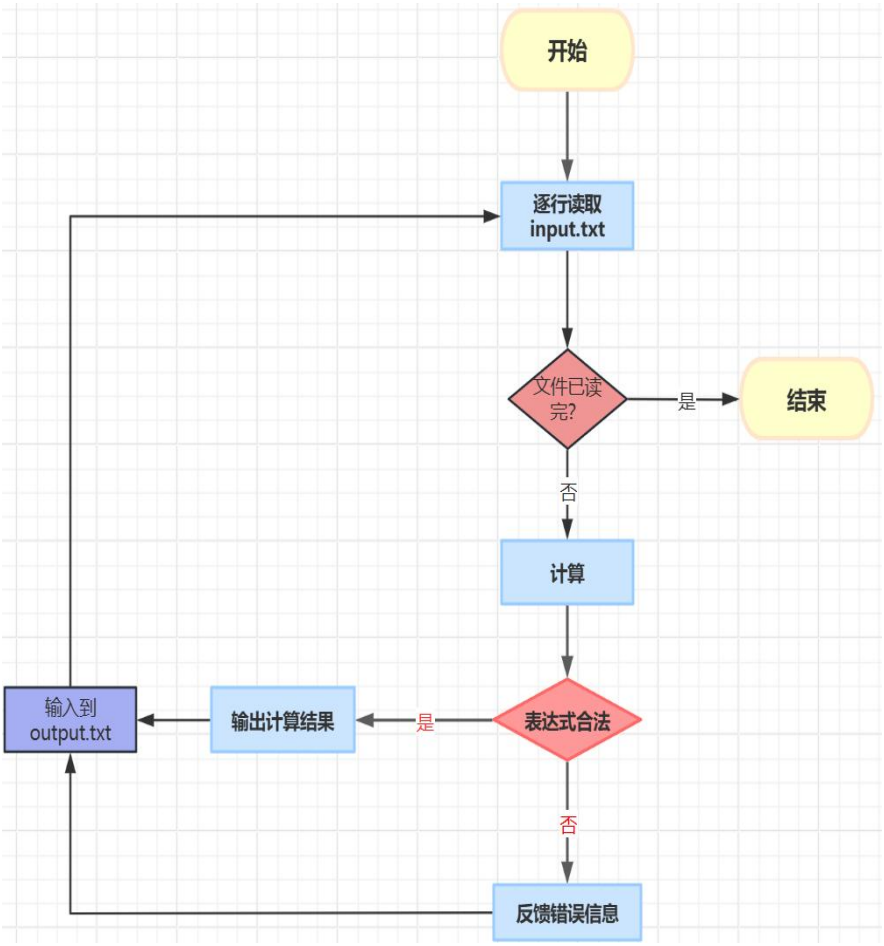


关键代码分析：

```
if((isError=Calculator(expr,&result))!=CALL_OK){
                putchar('\n');
                printf("%s",FindError(isError));
                putchar('\n');
                putchar('\n');
                continue;
}
```

调用 Calculator 函数计算，根据返回值判断表达式是否有错，如果有错，调用 FindError 函数打印错误信息，否则将计算结果存入 result 中再打印出来。

主函数 2：main.c



关键代码分析：

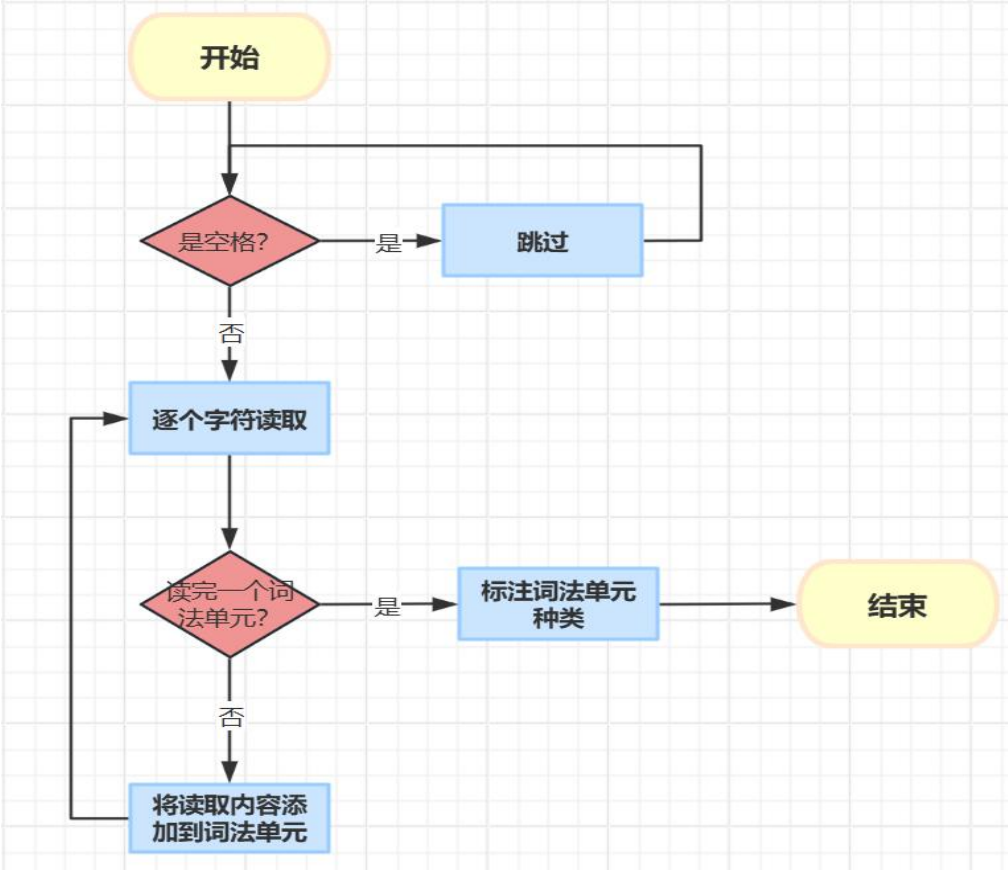
```
for(i=0;expr[i]!='\0';i++){
                if(expr[i]=='\r'){
                    expr[i]='\0';
                    break;
                }
}
```

以 rb 方式打开每一行以\r\n 结尾，故将\r 替换为\0 使得字符数组中存储一个完整的表达式。

4.2 其他函数

1.Situation GetToken(int *pExpr,char* expr,Token *pToken)；

进行词法分析的函数



关键代码分析：

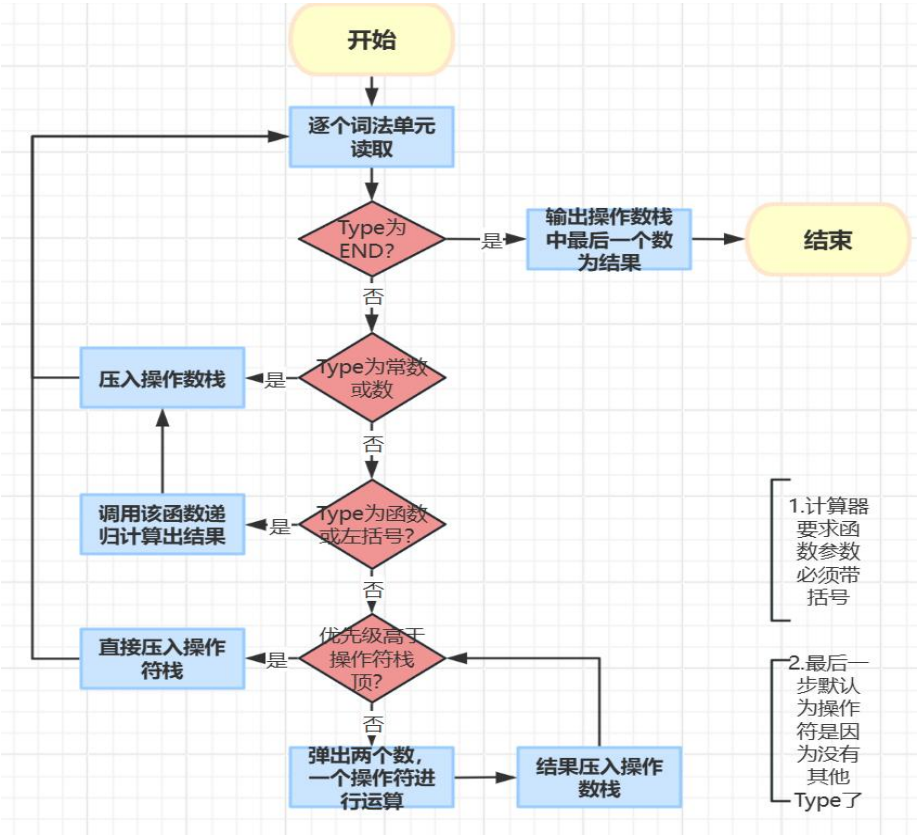
```
if(IsAlpha(expr[*pExpr])){
        int i=0;

        #if DEBUG
            printf("Sucess1.\n");
        #endif

        while(IsAlpha(expr[*pExpr])){
            if(expr[*pExpr]){
```

```
                current_token.value[i++]=expr[(*pExpr)++];
            }
        }
        #if DEBUG
            printf("Sucess2.\n");
        #endif
        current_token.value[i]='\0';
        if(strcmp(current_token.value,"PI")==0){
            current_token.type=CONSTANT;
            (*pToken)=current_token;
            return CALL_OK;
        }
        if(strcmp(current_token.value,"e")==0){
            current_token.type=CONSTANT;
            (*pToken)=current_token;
            return CALL_OK;
        }
        current_token.type=FUNCTION;
        (*pToken)=current_token;
        return CALL_OK;
}
```

遇到字母先将连续的字母全部读取并存储到词法单元中，而字母实际上有两种情况，先判断是否为系统中已经存入的常数，是则处理为常数，不是就先处理为函数，之后再判断系统中是否存在这个函数。

2. Situation Calculate(Token* p,double* result);

对词法单元数组进行计算的函数



关键代码分析：

```
if(p[loc].type==LPAREN){
                    i=1;
                    Token cal[MAX_EXPR];
                    int cal_len=0;
                    loc++;
                    while(1){
                        if(p[loc].type==LPAREN){
                            i++;
                        }
                        if(p[loc].type==RPAREN){
                            i--;
                        }
                        if(i==0){
                            cal[cal_len++].type=END;
```

```
                break;
            }
            cal[cal_len++]=p[loc++];
        }
        double calL;
        if((isError=Calculate(cal,&calL)!=CALL_OK)){
            return isError;
        }

        StackPush(operand,&calL);
        loc++;
        continue;
}
```

读取词法单元数组时读取到左括号，通过循环将括号内的表达式读取为一个新的Token（词法单元）数组（通过完整括号嵌套来判断读取完成，在之前已经有代码保证此处有完整正确的括号嵌套），递归调用该函数进行计算，将计算结果存入操作数栈中。

5 测试

```
PS D:\wang\CLanguage\Semter1Calculator\program_copy> ./mode
Enter an expression(or 'quit' to exit): 0^(-1+1)

The exponent of 0 must be greater than 0.

Enter an expression(or 'quit' to exit): sin(lg(10^(ln(e*e(PI/2-1)))))

It may be a fault after operator.

Enter an expression(or 'quit' to exit): sin(lg(10^(ln(e*e^(PI/2-1)))))
Resule: 1.

Enter an expression(or 'quit' to exit): 12.22*31-12.3/3
Resule: 374.72.

Enter an expression(or 'quit' to exit): 1.121.1+2

The inputted number is illegal.

Enter an expression(or 'quit' to exit): quit

Goodbye!
```

主函数 1：mode.c

```
Present time: 2024.12.26 21:13:33
sin(0)=0
cos(PI/2)=0
sin(PI/2)=1
cos(5*PI/2)=0
sin(PI/6)=0.5
cos(-PI/3)=0.5
cos(PI/4)=0.707107
sin(100*PI+PI*6)=0
sin(100*PI+PI/6)=0.5
    ERROR: You don't input.
 1   +1=2
1+@1    ERROR: Invalid character.
1/(-1+1)    ERROR: Division by zero.
sin1    ERROR: It's should be '(' after function.
sqrt(-1.1+1)    ERROR: Invalid argument for sqrt.
lg(-1.1+1)  ERROR: Invalid argument for lg.
ln(-1.1+1)  ERROR: Invalid argument for ln.
(-1+1)^0    ERROR: The exponent of 0 must be greater than 0.
g(1+1)  ERROR: The function doesn't exist.
1.11.1+2    ERROR: The inputted number is illegal.
(1+(1-1)    ERROR: The number of '(' isn't equal to the number of ')'.
)1( ERROR: The parentheses are nested incorrectly.
(1+)    ERROR: It should be a number after operator.
1++1    ERROR: It should be a number after operator.
()  ERROR: The expression includes empty parentheses.
sin(PI/2)+ln(e^2+(e+2e))=3.49381
    ERROR: You don't input.
ln(e)+sin(ln(e^(PI/2)))=2
ln(e)+sin(lg(100)+(-2+PI/2))=2
sin(PI/2)+ln(e^2+(e+2*e))=3.74367

Present time: 2024.12.26 21:13:49
sin(0)=0
```

主函数 2：main.c

6 总结

姓名-学号

你的总结

学到了什么？

痛点和难点

自己的贡献

如何与他人合作？

王比乐 2024091202009

学到了什么：

学习了中缀表达式直接计算的逻辑如对括号递归计算和对不同运算符的处理方法；学习了对栈这种数据结构后进先出特点的利用；学会了泛型栈和链表的编写；同时初步了解了编程规范。在此之外我通过实践编代码，提升了编写代码的能力。

痛点和难点：

代码不够简洁，常常为优化某个逻辑而思考良久（同时也受益匪浅）；为遵守编程规范而苦恼（但后来发现这样大大提高了代码的可读性，利远远大于弊）；词法分析如何实现才简洁而快速；泛型栈因为用的是通用指针而大大增加了编写难度。

自己的贡献：

将每位组员的成果集合为一个完整的项目，检查代码的规范性，优化代码，调试程序。

如何与他人合作：

明白了小组成员们应该多多沟通讨论，在项目开始前应共同制定项目计划与规范以保证项目所使用的头文件、参数、函数等命名保持一致使得项目便于衔接和推进。而组长尤其需要根据每位队员的能力合理地对项目进行拆分实现，还要按时监督小组成员，督促项目的进行，确保项目不因某个人的原因而耽误进度。

陈奕霖 2024091202006

学到了什么：

学习了对结构体栈和链表的应用并充分了解栈和链表的思想，可以以不同的方法实现栈和链表。

痛点和难点：

栈和链表需要多次使用指针甚至二重指针，还有数据类型的转换，需要清晰的思路。

自己的贡献：

为整个程序提供基础的对象，链表和栈，使得其他成员在运用函数的时候有坚实的基础。

如何与其他人合作：

分工需要明确，就像编写程序的时候需要做到高内聚松耦合，做到自己的工作完成好，这样才不会耽误整个工作的进行。

韩侨 2024091202008

学到了什么：

在本次项目中，我深入理解了程序从功能设计到实现的完整流程，熟练掌握了文件读取、写入操作以及对不同计算场景的逻辑控制。学会了如何将复杂的计算任务分解为可管理的步骤，并利用已有模块进行整合，增强了模块化编程思维，提升了代码的组织和架构能力，也进一步熟悉了 C 语言在实际应用中的各种语法细节和库函数使用技巧。

痛点和难点：

不同模块的数据传递和函数调用关系复杂，需要仔细研读其他组员的代码逻辑和接口设计，花费大量时间调试参数传递和返回值处理，以保证数据在各模块间准确流动。例如，在批量计算主函数中，从文件读取数据并转换为适合计算器处理的格式时，要兼顾数据完整性和格式正确性，同时处理可能出现的文件读取错误和数据解析异常，这对错误处理能力提出了较高要求，在复杂的错误情况交织时，定位和解决问题较为棘手。

自己的贡献：

实现两个主函数，为项目提供了核心的运行入口。在直接运行多次计算的主函数中，精心设计了用户输入循环和结果输出逻辑，确保用户能便捷地输入表达式并获取准确结果。对于批量计算主函数，实现了从指定输入文件读取大量计算式，并将计算结果准确输出到目标文件的功能，通过优化文件读取和写入流程，提高了批量计算的效率，增强了程序的实用性，使整个计算器能够满足不同场景下的计算需求，。

如何与他人合作：

积极参与小组讨论，与组员密切沟通，及时反馈遇到的问题和需求，共同商讨解决方案，确保各部分协同工作，才能保障项目的顺利完成。

杨寓杰 2024091202010

学到什么：

学习到了大型程序管理。通过编写多个文件，集中管理，完成了一个功能性强，集成度高的计算器成品。

痛点和难点：

将使用链式栈将泛型栈代替时，发现主函数的耦合度高，有些代码独立性不强，导致代码运行失败。

自己的贡献：

通过不断调试程序，发现了程序中的 bug，并调整程序使得计算器的功能更加完善。

如何与其他人合作：

对于产品的预期与实现的可能性，要多与别人讨论。达成一致后，对于代码实现的细节，也要和他人一起思考，一起讨论。

孟响 2024091202007

学到了什么：

学习了栈结构在计算器函数中的功能应用，能够准确掌握栈的不同函数实现并依据计算器不同应用场景进行调整。

痛点和难点：

栈函数的一些指针可以用更简洁的方式处理，编程不够规范，对栈的掌握熟练度仍有待提高。

自己的贡献：

完成代码最基础也最关键的栈的部分，避免程序从底层运行出现问题。

如何与他人合作：

多沟通多协调，只有与其他成员沟通过需求才能保证代码自洽，才能确保前后逻辑完整。