A report of the c-based interpreter

——the addition of the calculator

Mainly develop by Zheng Yusheng

Part1：简介：

前几周花了两三个小时，把 ds 的作业稍微整合了一下，写了一个前缀表达式转中缀表达式再进行计算的小程序。这周正好不是很忙，再加上听其他同班同学提起过写一个简单解释器的想法（大概也就是把计算器多加点东西嘛…），于是稍微读了下《编译原理》的前两章，加上自己脑补，花了四五个晚上，搓了这个简单的解释器。

这是一个语法制导的简单解释器，主要实现了词法分析、语法分析、逐语句解释三大部分，语言风格为动态类型，支持的数据类型有字符串、数组、布尔型、双精度浮点型；其中默认数据类型为浮点型。支持系统自带的数学函数运算、选择结构、循环结构，自定义函数运算并能支持递归；同时也有简单的错误抛出和异常处理机制。以及，正好看到 brainfuck 语言，感觉好玩，我也把它的解释器给加进来了。这也算是我一个人写过的最大的项目了…不过由于前期设计欠缺，（大概是边验证功能边写）总体开发效果并不好，也花了很多时间在重构和封装上面。由于时间有点紧，测试可能不是那么充分，因此可能还存在一些 bug；以及为方便观察，中间结果的打印输出我也没有把它去掉。

Part2：程序组成和结构：

整个项目由 12 个文件组成，除了 brainfuck 是单独的该语言解释器部分，process 和 calculator 共用 calculator.h 头文件，每个源文件都有相应的头文件。

以下三个部分是主要代码逻辑，其中主函数和词法分析、语法分析实现在 process 和 calculator 两个文件中：

1. process 文件处理和计算表达式，同时对于在表达式中的函数和变量完成相应的寻址工作。有如下公有函数：

   ```
   double expression(char* input, int start, int end);//对于表达式的计算
   ```

   该函数在处理过程中，调用了如下三个函数：

   ```
   int alpha(char* input, char ch, int p, char* mid, int i);//变量、函数的寻址入口

   double todigit(char a, int flag, char* mid, int i);//从中缀表达式中获得数字

   double caculate(char* mid);//计算中缀表达式
   ```

2. calculator 文件负责语句块的分片、每个语句的处理；在语句处理中完成选择循环赋值三种工作，同时提供函数定义功能；主函数的入口也在其中，去调用语句块处理函数。主要有如下函数以及主函数：

   ```
   //处理一个语句；返回值用来判断语句状态，以及如果其中存在函数的 return，使用其返回；
   double sentence(char* input, int start, int end);

   //用来处理布尔值判断，true 返回 1，false 返回 0；ERROR 返回代表不是布尔值；
   int bool(char* input, int start, int end);

   //处理一个语句块：返回值用来判断语句状态，以及如果其中存在函数的 return，使用其返回；
   double block(char* input, int start, int end);
   ```

3. functions 文件实现了相应的 c 语言函数调用接口，系统自带命令函数，以及自定义函数的处理过程；自定义函数的处理在一个 func_函数中完成，使用真实函数来模拟；有如下函数以及定义的系统自带函数：

   ```
   //自定义函数的实现过程，使用 c 语言函数模拟；
   double fun_(char* input, int p,ElementType *pele);

   //初始化变量作用域和系统函数；
   void ini_functions();

   //用来在变量作用域栈中寻找变量；
   ElementType *find_Elem(char* name);
   ```

其中系统函数和自定义函数的实现通过函数指针模拟的函数重载：

```
typedef double (*afunc)(char* input, int p); /*内置函数*/、
typedef double(*self_fun)(char* input, int p,ElementType* pele);//自己定义的函数
```

自定义函数的参数传递不做形参与实参匹配检测，只是在函数的变量作用域中创建实参；

接下来的是链表、哈希表、栈三种数据结构；

1. 栈的使用是用来完成语句块和语句处理中的括号匹配，以及前缀表达式转中缀表达式和计算；这一部分是从计算器继承下来的没有改动；
2. 哈希表用来实现变量作用域，在运行过程中哈希表由链表串联，实现了变量作用域栈，并在函数初始化和结束时生成和释放；
3. 链表主要有两个功能，在运行过程中实现变量的自动分配内存空间、释放空间和存储，同时串联哈希表；

Part3：数据结构：

本程序主要使用了链表、栈、哈希表三种数据结构。

栈部分已在计算器中描述，此处不再重复：

链表主要有两个功能，在运行过程中实现变量的自动分配内存空间、释放空间和存储，同时串联哈希表实现哈希表栈模拟变量作用域；其中链表的结构声明如下：

```
/* the defination of the list structure*/
//变量保存链表的结构声明
typedef struct LNode *PtrToLNode;
struct LNode {
ElementType Data;
PtrToLNode Next;
};
typedef PtrToLNode List;
//变量作用域链表的结构声明；
typedef struct HashNode *PtrToHNode;
```

```
11. struct HashNode {
12. HashTable Data;
13. PtrToHNode Next;
14. };
15. typedef PtrToHNode Tables;
```

链表支持如下操作:

```
List L;//在运行过程中自动分配的哈希表变量单元会被实际存放在这个链表里面
Tables T;//由哈希表组成的栈（链表实现），模拟在函数运行过程中的变量作用域栈
List Insert_list( List L, ElementType X );//用来插入新的变量节点；
List Delete_list( List L, ElementType * X );//用来删除被更新的变量节点；
Tables Insert_tables(Tables L, HashTable H);//在创建函数的时候进行变量作用域表的插入；
Tables RmFirst_tables(Tables T);//在函数结束时弹出变量作用域表；
```

哈希表用来实现变量作用域，在运行过程中哈希表由链表串联，实现了变量作用域栈，并在函数初始化和结束时生成和释放；哈希表中存放数据指向类型结构的指针，具体的类型结构体存放在链表内；数据类型结构体如下:

```
struct hashElem { //自定义元素结构，用来存储变量，总共有
    enum datatype type;//元素类型;
    double value1;//双精度浮点型，默认元素类型，在自定义函数中用来标识函数块开头
    int value2;//整型变量和布尔值，在自定义函数中用来标识函数块结尾
    void* f;//指针：可以是函数指针，也可以是数组和字符串指针
};

enum datatype { var , function , boolean, self, string, array };//元素类型标识；

struct hashElem;
typedef struct hashElem ElementType;//哈希表的入口，用来存储指向每个元素的指针；元素在运行时自动分配内存空间
```

哈希表的结构声明如下:

```
enum KindOfEntry { Legitimate, Empty, Deleted };//哈希表单元状态标识；
    struct hashElem { //自定义元素结构，用来存储变量，总共有
    enum datatype type;//元素类型;
    double value1;//双精度浮点型，默认元素类型，在自定义函数中用来标识函数块开头
    int value2;//整型变量和布尔值，在自定义函数中用来标识函数块结尾
    void* f;//指针：可以是函数指针，也可以是数组和字符串指针
};
```

```
    struct HashEntry//每个哈希表单元的入口
    {
        char key[30];
        ElementType*  pElement;
        enum KindOfEntry Info;
    };

    struct HashTbl//哈希表主体结构
    {
        int TableSize;
        int num;
        Cell *TheCells;    /* Cell *TheCells will be an array of  HashEntry cells,
allocated later*/
    };

    typedef unsigned int Index;
    typedef Index Position;
```

哈希表支持如下操作:

```
HashTable InitializeTable(Index TableSize);/*创建一个hash表并返回。 */
void DestroyTable(HashTable H); /*干掉这个哈希表 */
Position Find(KeyType Key, HashTable H);
/*输入关键字和hash表，返回在hash表中的索引，元素情况可查看.info */
int Insert(KeyType Key, ElementType* pElem, HashTable H);
/*输入hash值和元素指针以及hash表，将元素插入其中。 */
ElementType* Retrieve(Position P, HashTable H);/*获取hash表某个位置的元素的指针。 */
HashTable Rehash(HashTable H);/*将旧表扩大为2倍以上，返回新表。 */
```

Part4：测试样例集:

数据运算测试:

（单个数字）
1
answer = 1.000000

1+2*3
answer = 7.000000

（有运算优先级）
 1 - 2 / 3

answer = 0.333333

(1+3*4/(8/2))+1.9
answer = 5.900000

（结构中包含空格回车制表符等）
x=3;y=4;(1+x*y/(8/2))+1.9;
x =3; y= 4;(1+ x* y/ (8 /2))+1.9;
:
x =3;
y=
 4;
(1+ x
* y
/ (8
/2))+1.9;
:
answer = 5.900000

（包含系统函数）
sin(1)+cos(2)*sqrt(4)+pow(5,6)
sin(1)+cos(2)* sqrt(4)+pow(5 , 6)
mid[]: 0.841471 -0.416147 2.000000 * + 15625.000000 +
answer = 15625.009177

（异常：除以 0）
2/0
ERROR: dividing number can not be zero

（异常：错误表达式）
2/%43
calculate error!

逻辑结构测试

（循环结构：1-100 的和）
 x=1;sum=0;while(x<=100){sum=sum+x;x=x+1;};sum;
answer = 5050.000000

（函数测试：递归计算文波那契数）
define(ret){if(x<=2){return 1;}; x=x-1; y=ret(x); x=x-1; return y+ret(x);}; x=5; ret(x);
mid[]: 1.000000 1.000000 +
mid[]: 3.000000 2.000000 +

mid[]: 5.000000
answer = 5.000000

（函数与选择结构测试）
define(max){ if(a>b){return a;}; return b;};
a=3; b=4;answer = 4.000000max(a,b);
a=4; b=7;max(a b);answer = 7.000000;
a=3642; b=17.13;max(a b);answer = 3642.000000;

（多种格式测试选择与循环结构）
a=1;
b=0;
c=0;
while(a<=100){if(a%2==0){b=b+a;};if(a%2>0){c=c+a;};a=a+1;};
:
or type:
:
a=1;
b=0;
c=0;
while(a<=100){
if(a%2==0){
b=b+a;
};
if(a%2>0){
c=c+a;
};
a=a+1;
};
:
or:
 :
define(sum){a=1;
b=0;
c=0;
while(a<=100){
if(a%2==0){
b=b+a;
};
if(a%2>0){
c=c+a;
};
a=a+1;
};

```
a;b;c;};
:
sum();
a
answer = 101.000000
b
answer = 2550.000000
c
answer = 2500.000000

（数组定义与运算测试）
defarray(a,10);
a
a is an array:
 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000
i=0;
while(i<10){a[i]=i;i=i+1;};
a
a is an array:
 0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000
9.000000

（字符串定义与显示测试）
x="string";
x;
string
answer = 6.000000
x[2];
the char: r
answer = 114.000000
```

Part5：贡献：

本项目主体设计以及实现由郑昱笙（3180102760）完成，同时完成了测试和撰写文档；王晨旭（3180106324）提供了用于测试计算器的测试点，同时其中部分也用于测试本编译器；高嵘倩（3170105931）完成了计算器中 stack.c 和 stack.h 部分，同时也被本编译器继承。