# 《初级软件设计实作》
# 软件开发设计文档

题目名称：Mini C 语言编译器

姓　　名：陈立基

学　　号：20192131070

专　　业：计算机科学与技术

班　　级：19 级 2 班

# 一、软件课题说明

该项目主要是设计一个 Mini C 编程语言的简单有用的编译器，输入是一段 Mini C 代码，然后程序自动对其进行扫描再执行用户所要求执行的操作。

项目实现对代码单词的扫描、语法分析、语义分析、生成中间代码、实现代码指令解释执行。这些功能整合到一个软件中，并且能按照用户需要分阶段地执行，因此分别设计分界面和执行按钮来进行操作和显示。

# 二、开发规划

| |
|---|
| 开发语言：C++ |
| 开发软件：QT 6.0.4 |
| 编译器：MinGW64 bit |
| 运行环境：Microsoft Windows10 |

# 三、需求分析

| | |
|---|---|
| 扫描： | 输入：Mini C 代码，用户首先将代码内容输入 |

| | |
|---|---|
| | 到计算机的 txt 文件中保存 |
| | 输出：在界面中显示扫描到的代码内容，供用户检查 |
| 语法分析： | 输入：在扫描完毕后，直接使用前面读取的信息，并在开头插入对 input/output 函数的定义 |
| | 输出：对代码信息进行语法分析，如有错误对用户进行提示，顺带上出错的位置；如果没有错误则建立语法树，打印在界面上，用户可选择保存在计算机上 |
| 语义分析： | 输入：在语法分析生成语法树完毕后直接使用该语法树 |
| | 输出：各级符号表，并进行类型检查，如有错误则提示 |
| 生成中间代码： | 输入：在语法分析生成语法树完毕后直接使用该语法树 |
| | 输出：根据语法树生成中间代码，结果以文件形式保存在计算机上，并且在界面上显示以供用户检查，用户可将其在计算机上以文件形式另存 |
| 解释执行： | 输入：生成中间代码时保存的中间代码文件 |
| | 输出：生成一个新的解释器界面，根据提示 |

| | |
|---|---|
| | 用户进行解释运行，且在必要时弹出对话框<br>体现用户输入数值 |

# 四、系统设计

## 概要分析

当前已知 Mini C 语法的惯用词法、语言标记，有了语言的每个语句构造的 BNF 描述和相关语言的描述。它支持数据类型（整形，浮点型）、函数、数组。关键字包含 if int return 等，专用符号+ * == 等，本质上是 C 语言的一个子集。

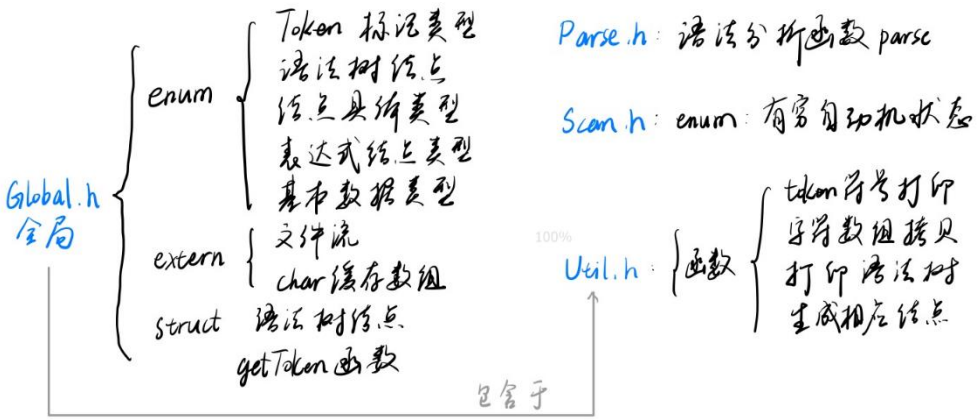
扫描方面只需要直接使用文件流。语法分析方面，可以先根据 Mini C 的语法规则，将其消除歧义性，即消除左公因子和左递归，根据每条规则写一个构造语法树结点的函数。读取的时候采用自顶向下方法，利用有穷自动机每次读取到一个 token 符号，最终构造语法树并显示出来。之后根据语法树就可以继续建立符号表、生成执行中间代码。前面的功能除了执行中间代码可以设计一个界面，执行中间代码时可能有额外的输入，因此为方便使用额外设计一个界面。然后可以为每种功能分别设计头文件、功能函数，提供函数接口以供主程序调用。

用到的几个头文件：

| | |
|---|---|
| Global.h | 提供一些全局的要用到的变量、枚举的标记类 |

| | |
|---|---|
| | 型，以及获取 token 符号的函数等 |
| Util.h | 提供一些功能函数 |
| Scan.h | 枚举获取 token 的有穷自动机的状态 |
| Parse.h | 语法分析函数 |

部分内容：

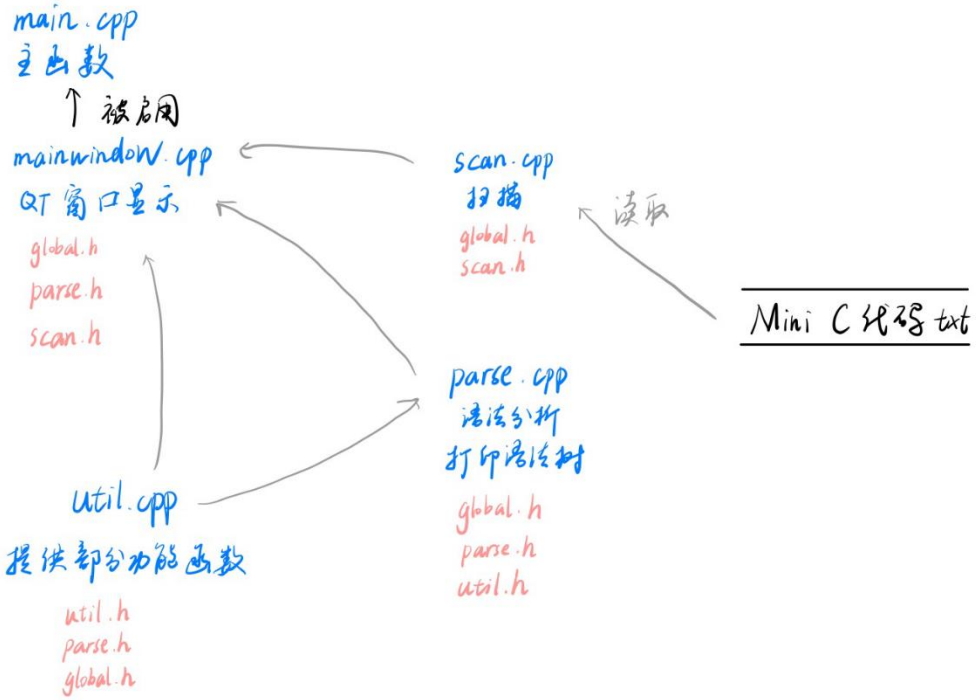



源程序：

| 程序名 | 部分数据结构 | | 说明 |
|---|---|---|---|
| Main.cpp | 函数 | 调用 mainwindow.cpp | 直接启用 mainwindow |
| Mainwindow.cpp | 函数 | 显示界面，界面构造函数提供按钮与功能调用的接口 | 调出程序界面，并根据用户操作调用相应功能并显示 |
| Scan.cpp | 变 | 1 保留字结构体数组 | 读取用户选取 |

| | | | |
|---|---|---|---|
| | 量 | 2 缓存 char 数组<br>3 各标记读取到的行、位置等的 int 记数变量 | 的文件并扫描 |
| | 函数 | 1 读取下一字符<br>2 回溯一个字符<br>3 保留字检查<br>4 获取下一 token（getToken） | |
| Util.cpp | 函数 | 1 token 打印<br>2 生成特定表达式或语句结点<br>3 拷贝字符数组<br>4 返回结点对应字符串<br>5 打印语法树的子递归函数<br>7 打印语法树 | 提供部分功能函数 |
| Parse.cpp | 变量 | 1 暂存打印语法树字符的字符串 | 进行语法分析并显示语法树 |
| | 函数 | 1 生成语法树的一系列递归子程序函数<br>2 extern 在界面输出显示语法树 | |

主要调用关系：




## 详细设计

### 词法分析器

采用有穷状态机来实现，可以区分运算符，Number，ID 以及注释，并且对 ID 进行关键字检查，如果同名则识别为关键字保存。

### 语法分析器

采用一种自顶向下的分析方法——递归子程序法实现句型的匹配，实现了顺序、分支、复合、循环的语句结构，函数定义和变量定义，以及表达式的识别。不同语句结构的识别，只需要捕捉其对应的

关键字（如 if，do，while）或者运算符（如 { }）即可，而函数定义则只需匹配“（parameters）”结构即可。

表达式的形式有算术表达式和赋值表达式，算术表达式中涉及到变量、整数、浮点数、数组项以及函数调用的识别，在识别数组项时只需判断 ID 后面有无 “[ index ]”的结构即可，这一点与识别数组变量定义时相同。而对于赋值表达式，必须捕捉到“ID = expression”的形式，对此，我们设置了 is_vari——判断某个 factor 是否为 ID 类型，以及 3 个 first 标记——判断该 factor 是否存在于句首的位置，具体实现见 PARSE.cpp 文件 expression 函数。

另外，我们在递归子程序中加入了语法树结点生成的功能，以便保存相关信息用于后续工作。结点类的定义如下：

```
struct TreeNode {
    TreeNode* children[MAXCHILDREN];
    TreeNode* sibling; /* 兄弟结点*/
    NodeKind nodeKind;
    int lineno;
    union { StmtKind stmt; ExpKind exp; } kind;
    union {TokenType op;
        int _int;
        float _float;
        char* name;} attr;
    ExpType type;
};
```

## 语义分析器

这一模块的主要工作是类型检查和生成中间代码，在生成中间代

码之前，我们不得不先构建符号表，而考虑到 Mini C 程序结构的特殊性（即完全由定义语句构成），我们将在构建符号表的过程中完成类型检查。

类型检查的主要内容有：

1. 操作数类型与运算符的种类是否兼容；

2. 非 void 返回值类型函数是否缺失 return 语句，以及 return 的表达式的值类型是否与定义的返回值类型相同；

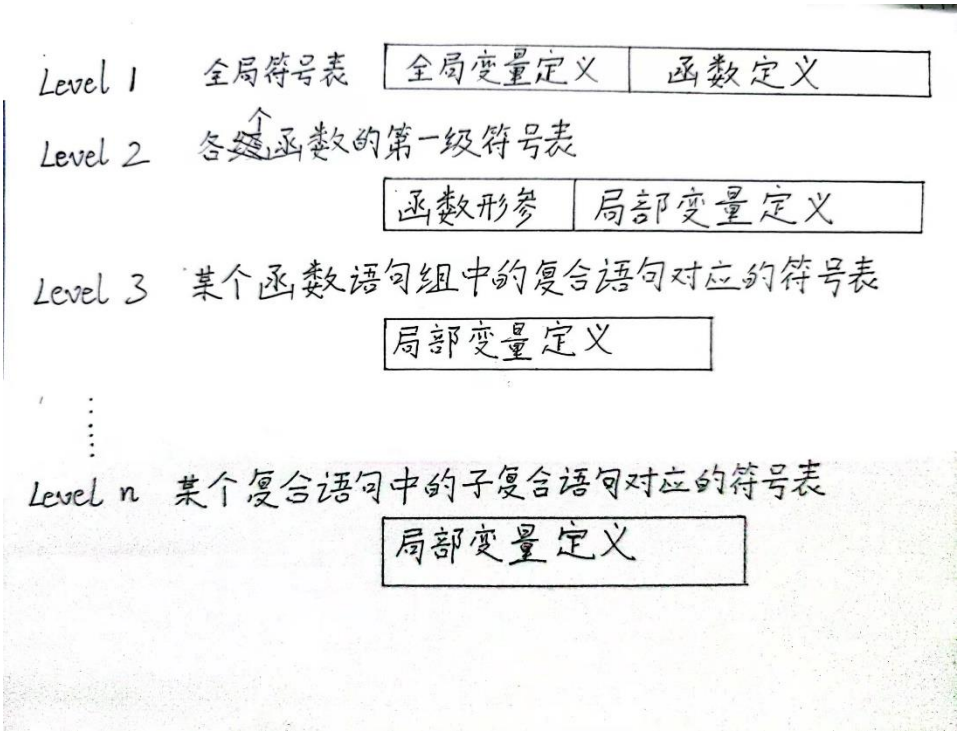
3. 条件表达式的值类型是否为 void；

4. 是否使用了未定义的变量名或者调用了未定义的函数；

5. 在使用数组型变量时是否提供了索引，数组索引的值类型是否为整型；

6. 函数调用时，形参列表和实参列表是否符合。

在 Mini C 程序中，我们需要对全局变量和局部变量作区分。我们允许同名的全局变量和局部变量同时存在，而且在使用同名的变量时，优先使用局部变量，也就是说将全局变量暂时“屏蔽”掉。为了实现这一功能，我们使用了多级符号表的结构，各级符号表如下：



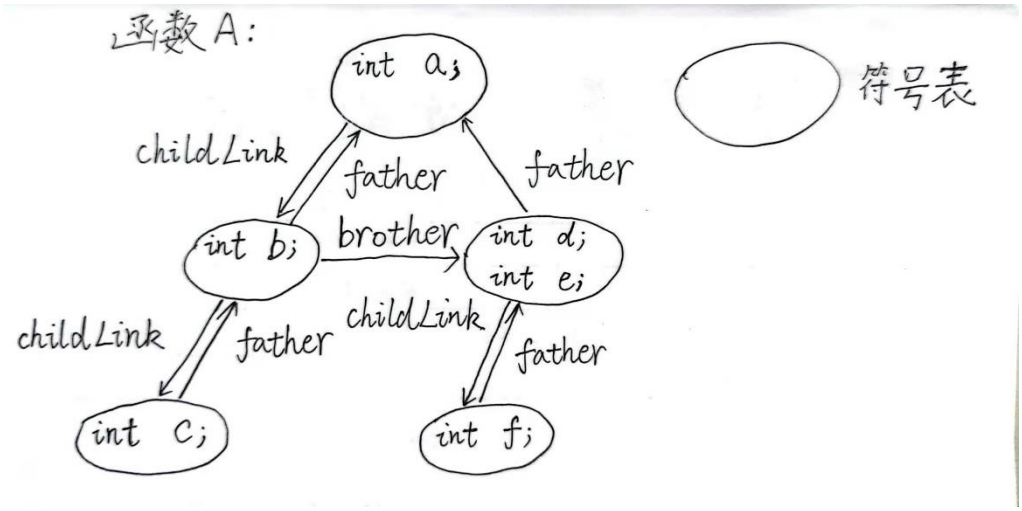

在对某个函数的语句组进行检查时，每遇到一个复合语句，倘若存在局部变量定义，我们就为其构建一个符号表，为了提高存取的效率，我们采用哈希表的数据结构进行保存，通过变量名计算哈希值来获取唯一索引，当出现哈希冲突时，我们在对应的索引位置延伸出一个链表，将哈希值相同的变量的信息依次插入链表中保存。关于一个函数内出现的多级符号表的保存，考虑到他们的生成次序，其实是和访问次序相同的，我们只需要把各级符号表按生成的次序将它们的入口地址进行保存即可，这样后面需要访问变量的时候可以准确的定位它的作用域。保存符号表入口地址的数据结构定义如下：

```
/* 用于构建符号表入口链表的表项*/
typedef struct Enter {
    char* funcName;
    TokenTable table;
    Enter* father;
    Enter* childLink;
    Enter* brother;
}*EnterList;
```

这里举一个例子：

```
int A() {
    int a;
    a = 1;
    if (a == 1) {
        int b;
        b = 15;
        while (b > 10) {
            int c;
            c = b;
            b = b - 1;
        }
    }
    else {
        int d;
        int e;
        d = 20;
        while (d < 40) {
            int f;
            f = 3;
            d = d - f;
        }
    }
    return a;
}
```

上述函数对应的数据结构如下：




在对函数的语句组进行检查的过程中，我们模拟出一个按级存放符号表入口地址的栈 tabList，用整型变量 level 指示栈顶的位置，栈底位置的符号表就是全局符号表，而函数对应的各级符号表，则按照

访问次序依次入栈，当某个作用域检查完毕时，其对应的符号表也从栈中弹出。

另外，为了实现局部变量在虚拟机内存中的灵活存储，我们采用相对偏移量的方式表示每个函数内局部变量的相对位置，并且设计一个链表结构来存放每个函数所占用的内存空间的大小，该数据结构定义如下：

```
/* 存放函数最大内存偏移量的链表项*/
typedef struct FuncOffset{
    char* funcName;
    int offs; //偏移量
    FuncOffset* next;
} * FuncOffList;
```

存放变量定义以及函数定义相关信息的数据结构如下：

```
/* 用于存放符号表的每一个变量的有关信息*/
typedef struct BucketListRec {
    char* name;
    int memloc;  //变量在内存中的相对当前位置的偏移量
    ExpType type;  //变量的数据类型或者函数的返回值类型
    bool isParm;  //标识是否为函数参数
    bool isLocal;  //是否是局部变量
    bool isFunc; //是否为函数
    struct TreeNode* params;  //函数需要的参数列表
    struct TreeNode* compK; // 函数的复合语句结点
    struct BucketListRec* next;
} *BucketList;
```

类型检查之后即可开始生成中间代码。由于我们生成的中间代码是逐步执行的，因此必须先从 main 函数的语句组开始生成。为了快速确定某个函数调用所涉及到的局部变量的内存起点，在开始遍历 main 函数之前，我们把全局变量的最大偏移量先读取到 cp 寄存器（current pointer）中，这样就将内存最低位置的一块内存划为全局变量的存储区域，而后 main 函数中的局部变量在内存中的绝对地址

即为 (cp) + memloc。当在 main 函数中调用其他函数时，我们把 main 函数的最大偏移量也加到 cp 上，这样一来，内存中(cp) ~ (cp) + max_main 的这一块区域就被划分成 main 函数的变量存储区域，其他嵌套调用的情况以此类推。

这里举一个例子：

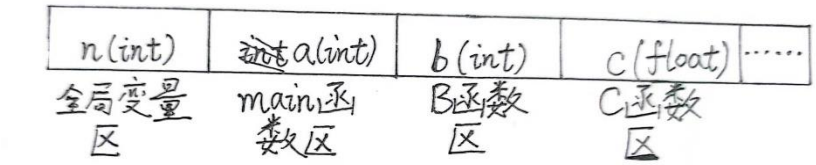
```
int n;

void C() {
    float c;
}

void B() {
    int b;
    C();
}

int main() {
    int a;
    B();
}
```

上述程序对应的内存占用情况如下：




为了方便的表示函数之间的调用关系，在遍历的过程中，我们设计一个栈来存储，每调用一个函数，就把这个函数的名字压入栈中，并且开始遍历这个函数的语句组。当函数调用结束时，将栈顶的函数名弹出，并且将该函数在内存中占用的区域回退释放，这一步通过将

cp 减去其上一层函数的最大偏移量即可完成。

关于 input 和 output 函数需要提一下，这两者都是作为 Mini C 内置的函数进行使用的，因此我们在读取源文件之后，往文件的头部插入这两个函数的定义（见下图），以便其他函数对其进行调用。在这里我们添加了两个新的关键字 INPUT 和 OUTPUT，分别表示从键盘获取一个整数，以及向屏幕打印一个整数的操作。

```
int input(void) {
    int n;
    INPUT n;
    return n;
}

void output(int x) {
    OUTPUT x;
}
```

关于函数调用还有一个很重要的步骤，那就是函数传参。因为在 Mini C 中存在数组类型变量，因此传参形式包括值传递和址传递两种。对于值传递，我们只需要将实参变量在内存中对应单元的值复制到形参变量的对应单元即可。而对于址传递（也就是数组传参），我们则是将实参数组的首地址复制到形参数组的对应的那一个单元中，这样一来，当我们访问形参数组时，就需要进行两次寻址操作来读取到真正的目标单元。

## 虚拟机

到了中间代码的解释阶段，由于虚拟机中的内存和寄存器堆必须

能够支持 int 和 float 类型的值，我们为内存和寄存器中的单元设计了一个可以兼容两者的结构，定义如下：

```
/* 声明数据单元*/
typedef struct {
    union {
        int _int;
        float _flo;
    } val;
    int isFlo;  /* 标记是否存放浮点数*/
} UNIT;

static UNIT dMem[DADDR_SIZE];
static UNIT reg[NO_REGS];
```

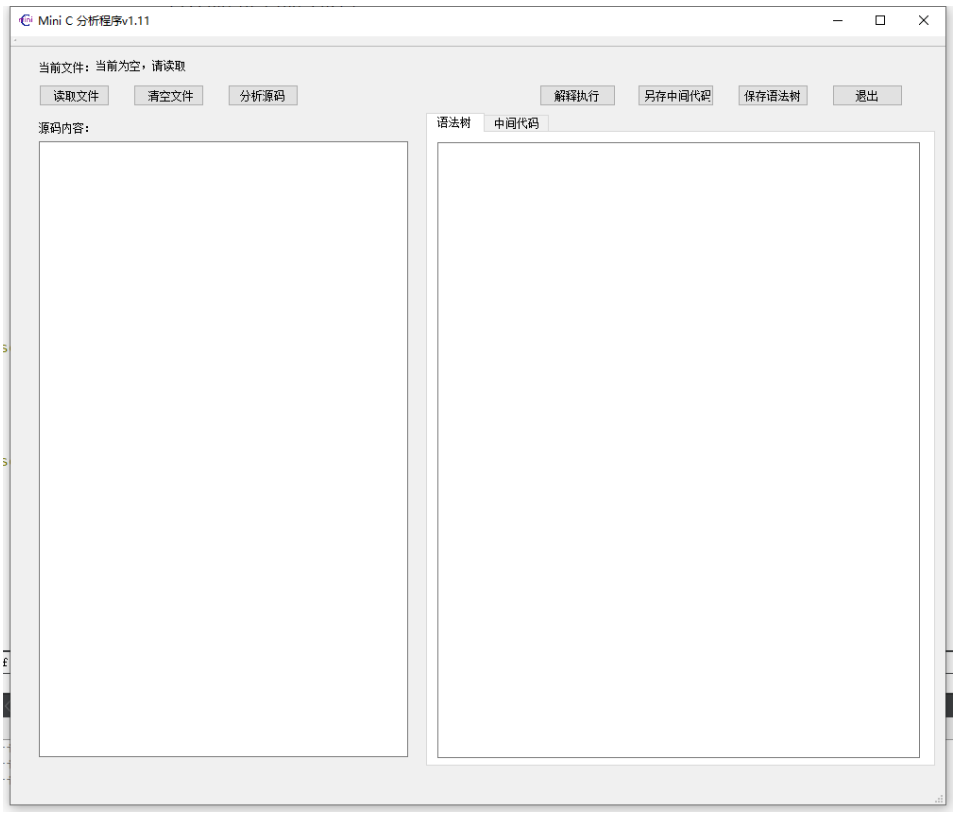
最后，在虚拟机将寄存器中的值保存到内存中时，需要判断寄存器中的值将以何种类型保存（int 还是 float）。针对这一要求，我们对虚拟机中的 ST 操作码进行了扩展，改成了 ST_INT 和 ST_FLO，用以对两种数据类型做区分。

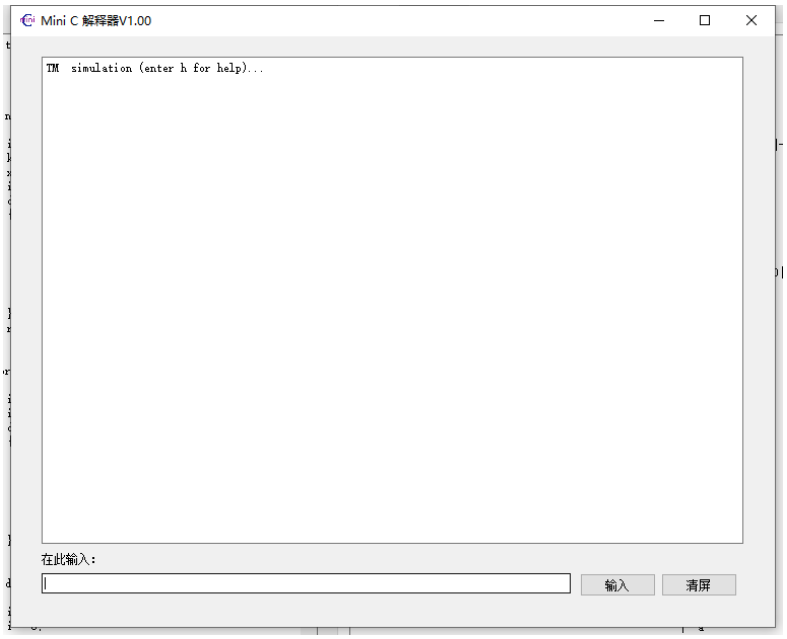
# 五、开发实现

详细代码见于附带的源程序中，在相应版本的 QT 中可直接读取运行。

# 六、测试

界面：

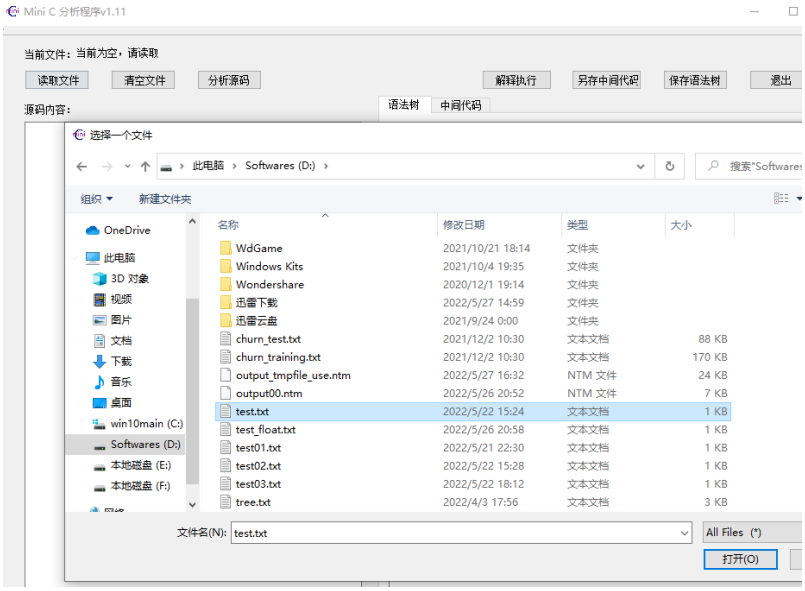


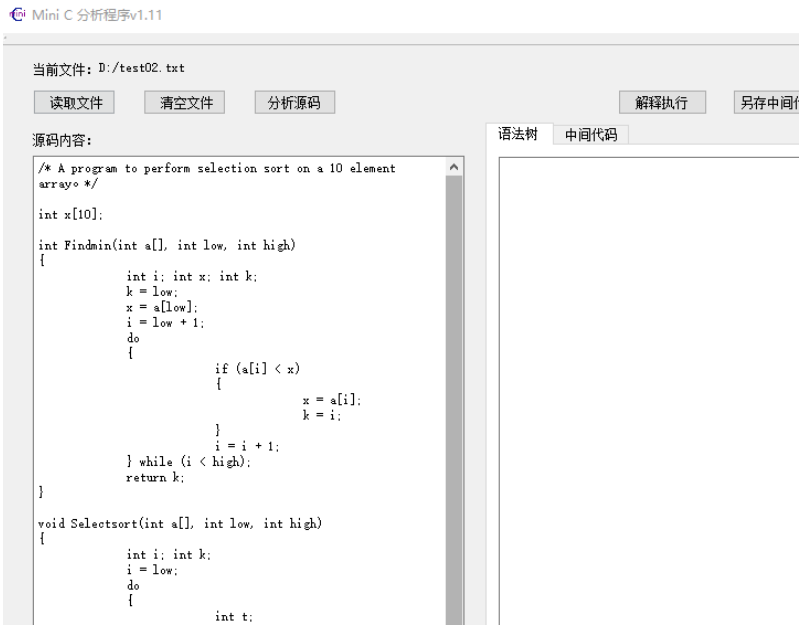




首先进行简单的功能测试，先测试输入 10 个整数进行排序的程序(测试用例文件中的 test00.txt)为例，顺便测试关键字 void int do while if return，符号 /* */ ; [ ] ( ) { } = < + - 以及注释、数组、变量与函数定义(参数函数、空参数函数)、复合语句、表达式赋值、main 函数、

input/output 等功能。

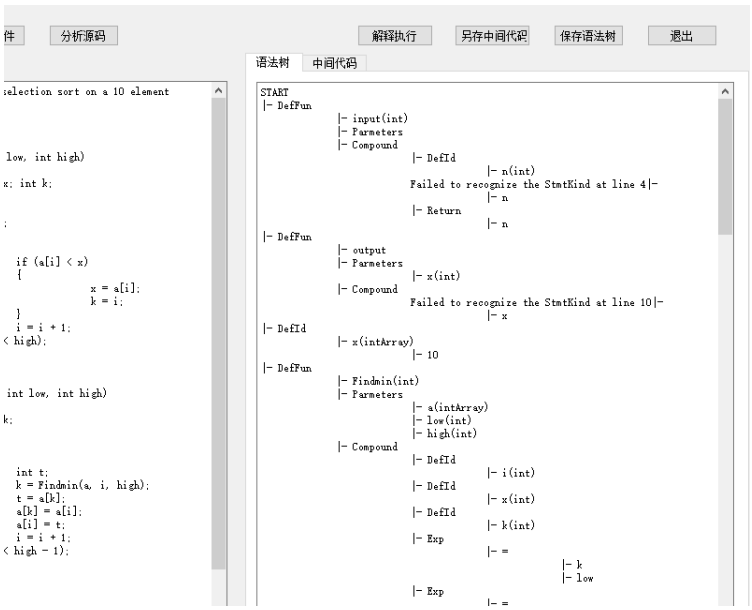
首先读取选择文件，在左边界面上显示，检查

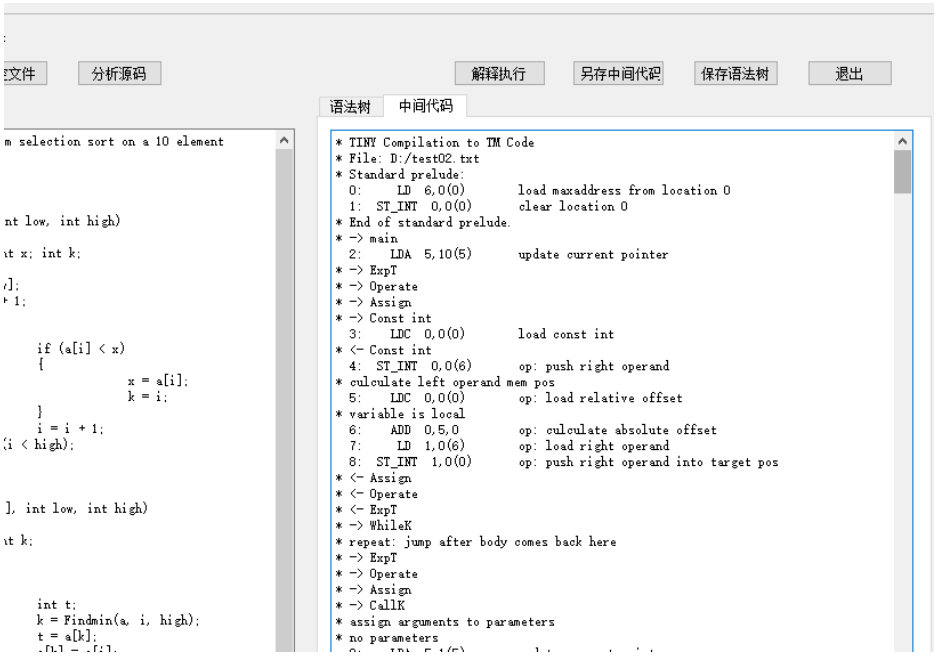


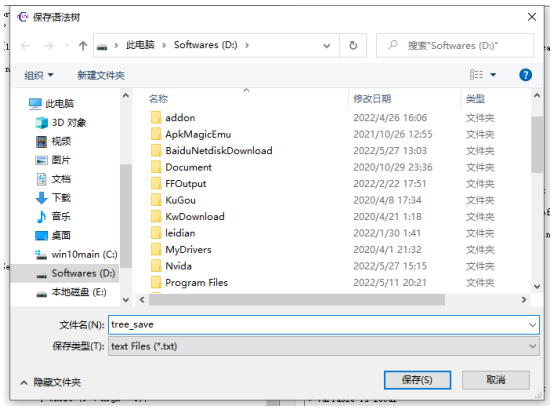



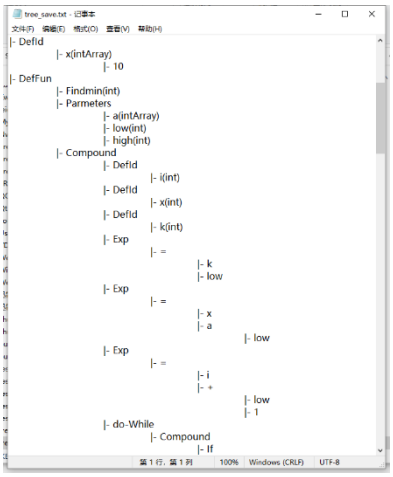

正确读取显示，且清空文件功能正常

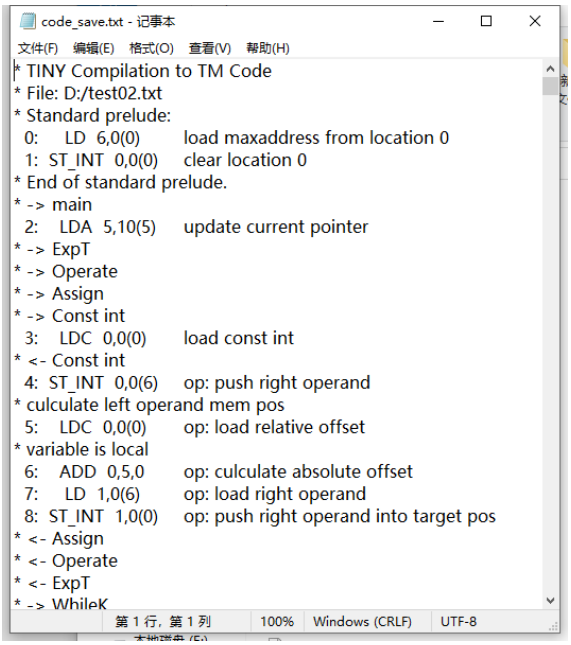
语法分析生成语法树和中间代码显示，并测试保存语法树和中间代码功能


件
分析源码
解释执行
另存中间代码
保存语法树
退出
语法树
中间代码
selection sort on a 10 element
low, int high)
x; int k;
if (a[i] < x)
{
x = a[i];
k = i;
}
i = i + 1;
< high);
int low, int high)
k;
int t;
k = Findmin(a, i, high);
t = a[k];
a[k] = a[i];
a[i] = t;
i = i + 1;
< high - 1);
START
|- DefFun
|- input(int)
|- Parmeters
|- Compound
|- DefId
|- n(int)
Failed to recognize the StmtKind at line 4|-
|- n
|- Return
|- n
|- DefFun
|- output
|- Parmeters
|- x(int)
|- Compound
Failed to recognize the StmtKind at line 10|-
|- x
|- DefId
|- x(intArray)
|- 10
|- DefFun
|- Findmin(int)
|- Parmeters
|- a(intArray)
|- low(int)
|- high(int)
|- Compound
|- DefId
|- i(int)
|- DefId
|- x(int)
|- DefId
|- k(int)
|- Exp
|- =
|- k
|- low
|- Exp
|- =

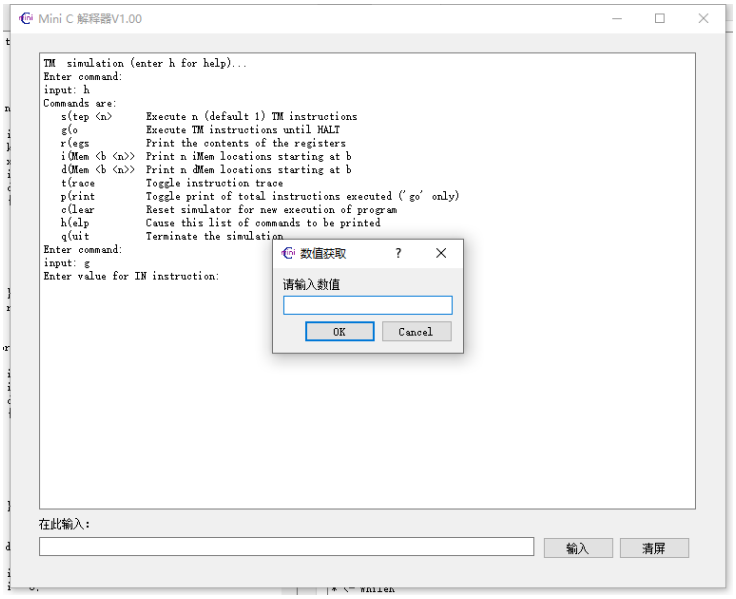

文件
分析源码
解释执行
另存中间代码
保存语法树
退出
语法树
中间代码
m selection sort on a 10 element
nt low, int high)
it x; int k;
i];
+ 1;
if (a[i] < x)
{
x = a[i];
k = i;
}
i = i + 1;
i < high);
], int low, int high)
it k;
int t;
k = Findmin(a, i, high);
t = a[k];
* TINY Compilation to TM Code
* File: D:/test02.txt
* Standard prelude:
0: LD 6,0(0) load maxaddress from location 0
1: ST_INT 0,0(0) clear location 0
* End of standard prelude.
* -> main
2: LDA 5,10(5) update current pointer
* -> ExpT
* -> Operate
* -> Assign
* -> Const int
3: LDC 0,0(0) load const int
* <- Const int
4: ST_INT 0,0(6) op: push right operand
* culculate left operand mem pos
5: LDC 0,0(0) op: load relative offset
* variable is local
6: ADD 0,5,0 op: culculate absolute offset
7: LD 1,0(6) op: load right operand
8: ST_INT 1,0(0) op: push right operand into target pos
* <- Assign
* <- Operate
* <- ExpT
* -> WhileK
* repeat: jump after body comes back here
* -> ExpT
* -> Operate
* -> Assign
* -> CallK
* assign arguments to parameters
* no parameters


保存语法树
此电脑 › Softwares (D:) ›
搜索"Softwares (D:)"
组织
新建文件夹
此电脑
3D 对象
视频
图片
文档
下载
音乐
桌面
win10main (C:)
Softwares (D:)
本地磁盘 (E:)
名称
修改日期
类型
addon 2022/4/26 16:06 文件夹
ApkMagicEmu 2021/10/26 12:55 文件夹
BaiduNetdiskDownload 2022/5/27 13:03 文件夹
Document 2020/10/29 23:36 文件夹
FFOutput 2022/2/22 17:51 文件夹
KuGou 2020/4/8 17:34 文件夹
KwDownload 2020/4/21 1:18 文件夹
leidian 2022/1/30 1:41 文件夹
MyDrivers 2020/4/1 21:32 文件夹
Nvida 2022/5/27 15:15 文件夹
Program Files 2022/5/11 20:21 文件夹
文件名(N): tree_save
保存类型(T): text Files (*.txt)
隐藏文件夹
保存(S)
取消


tree_save.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
|- DefId
|- x(intArray)
|- 10
|- DefFun
|- Findmin(int)
|- Parmeters
|- a(intArray)
|- low(int)
|- high(int)
|- Compound
|- DefId
|- i(int)
|- DefId
|- x(int)
|- DefId
|- k(int)
|- Exp
|- =
|- k
|- low
|- Exp
|- =
|- x
|- a
|- low
|- Exp
|- =
|- i
|- +
|- low
|- 1
|- do-While
|- Compound
|- If
第 1 行，第 1 列
100%
Windows (CRLF)
UTF-8

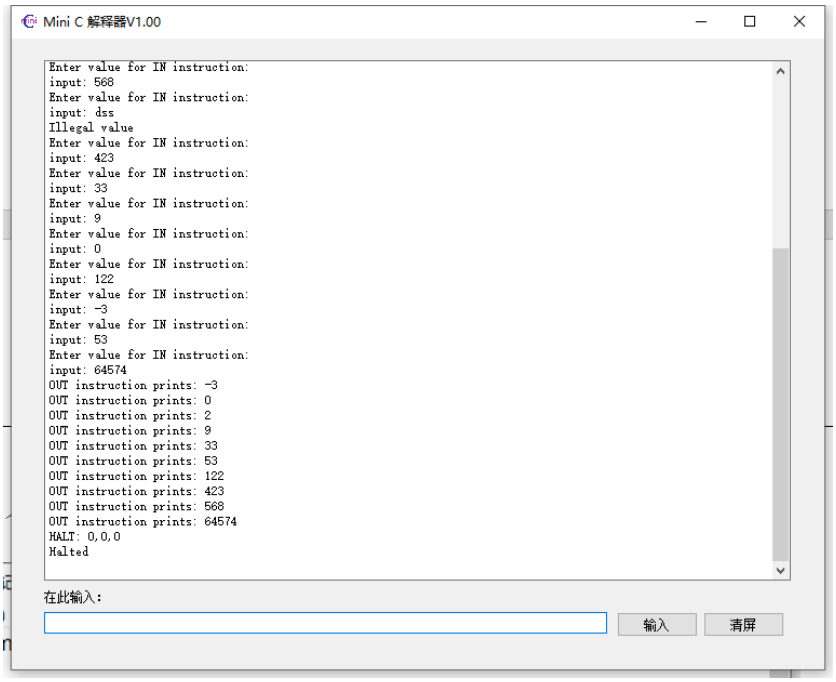



然后解释执行，预期为输入 10 个整数，非输入整数时过滤，最后排序好输出





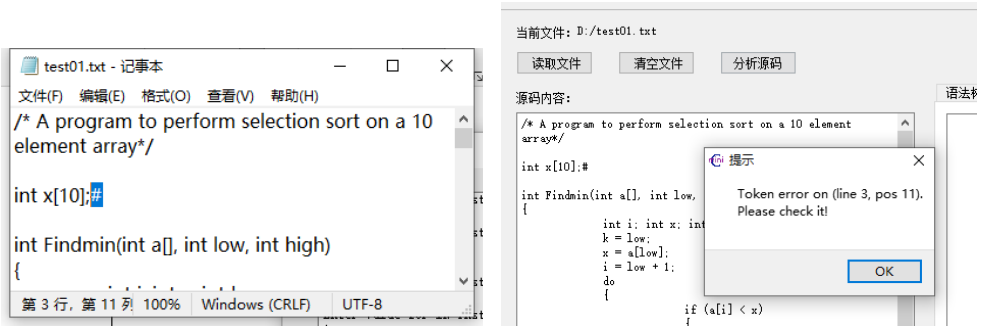
尝试输入一次字母，提示非法数值，继续输入




在接受 10 个合法数据后，成功正确排序并输出。所测试的关键字、功能均可正常识别、运行

接下来简单测试错误

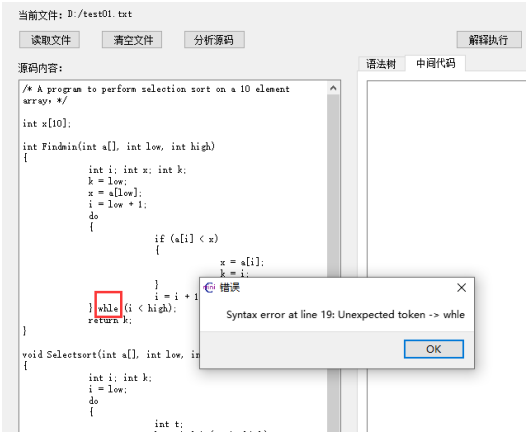
另外准备一个测试文档，在第 3 行 11 列加一个’#’，尝试错误的符号



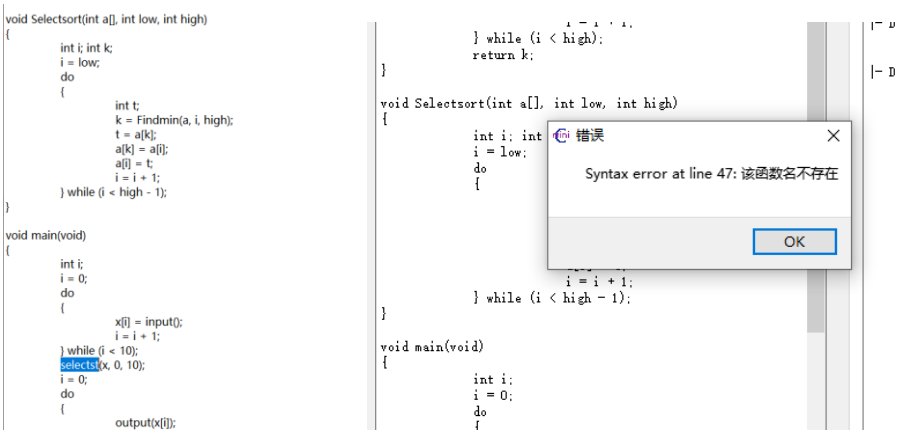

成功进行提示

再简单测试错误关键字、语法。

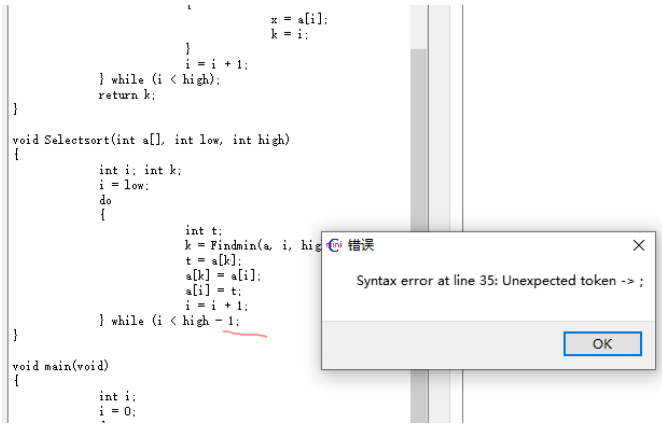
故意写错 while 测试




故意写错调用的函数测试



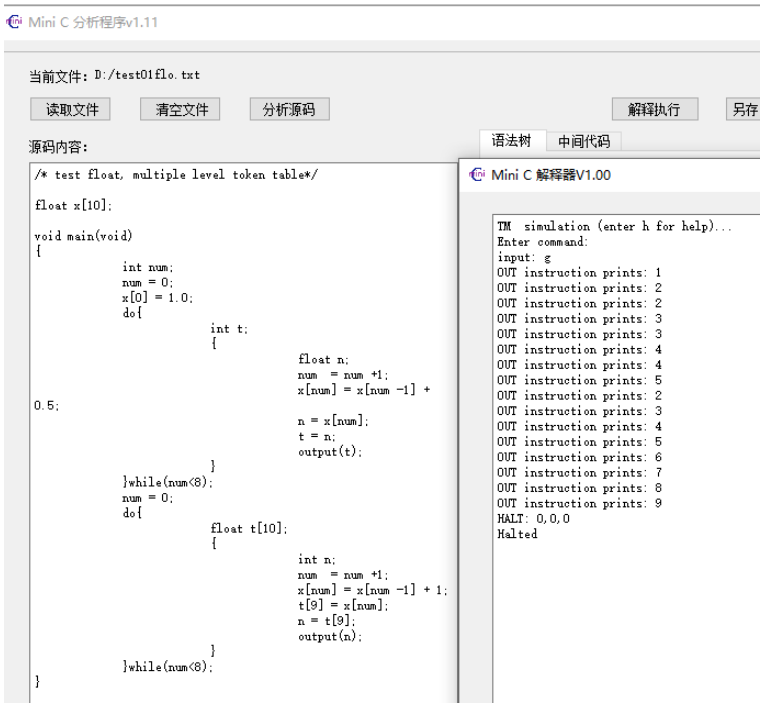

故意写漏右括号



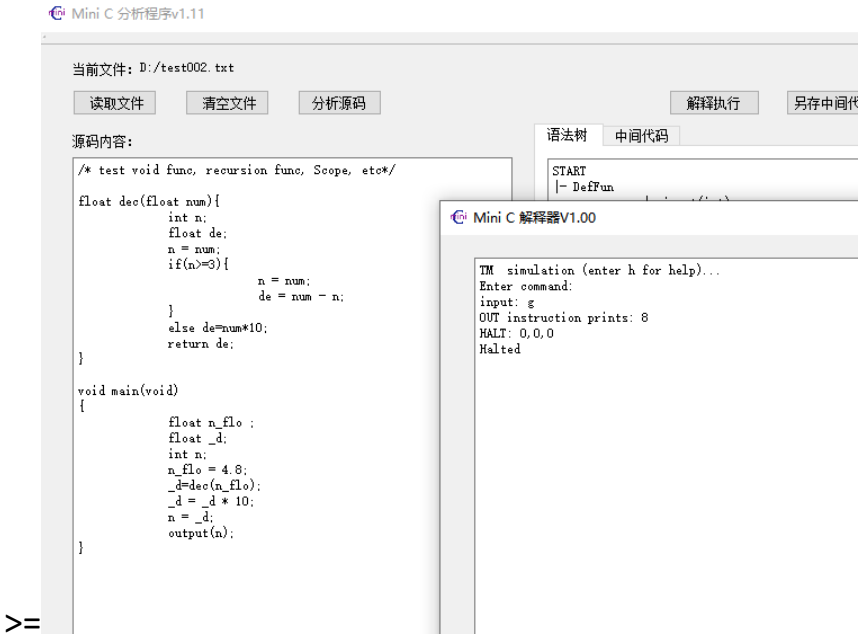

均成功进行相应错误提示

接下来测试剩下的一些关键字和功能等。

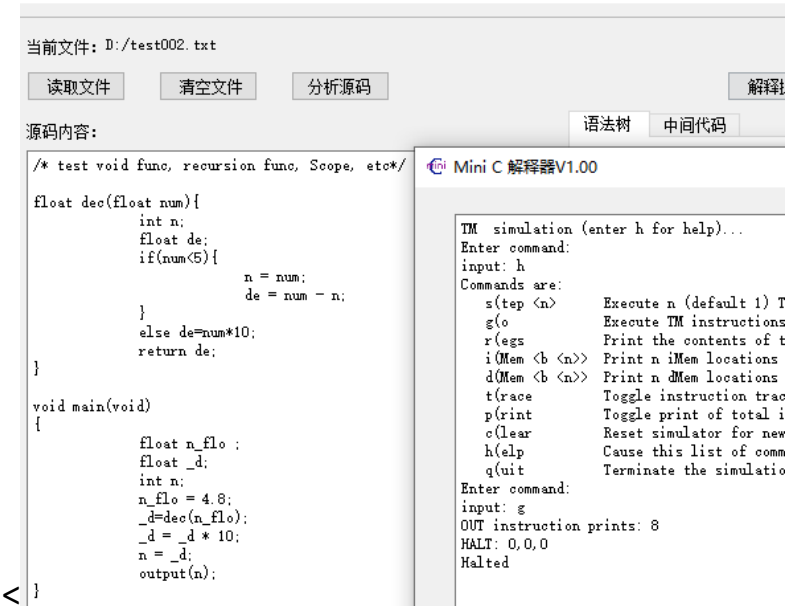
test01.txt：测试 float 、类型自动转换 符号

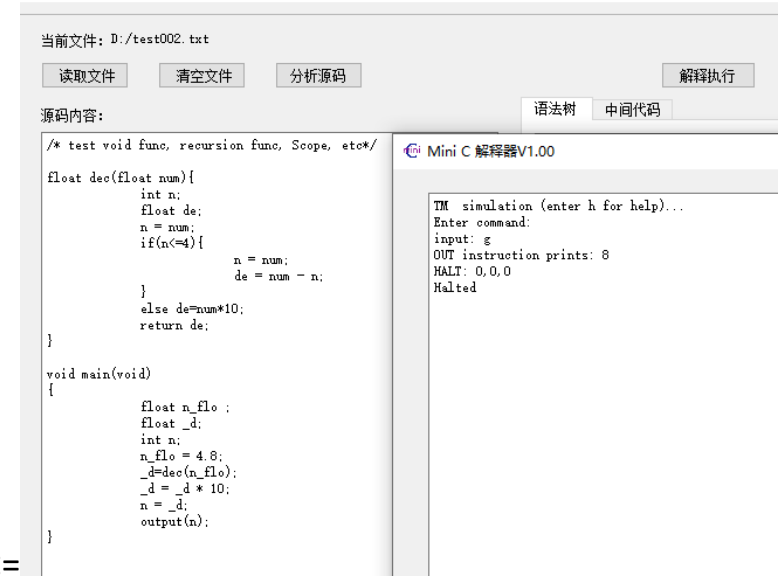


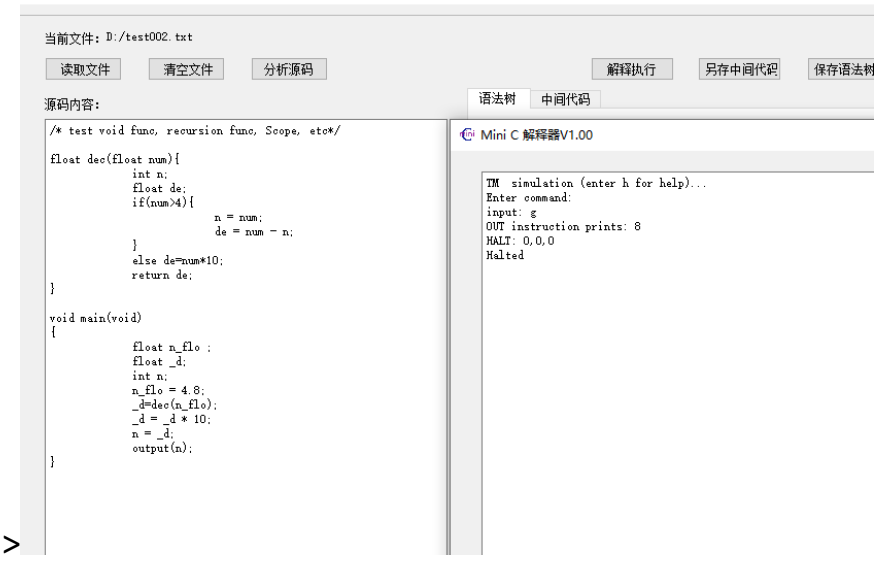

test02.txt：测试关键字 else 符号 * >= _ 以及定义变量作用域，并通过修改 dec 函数内容测试了剩下的<、 <=、 == 运算符

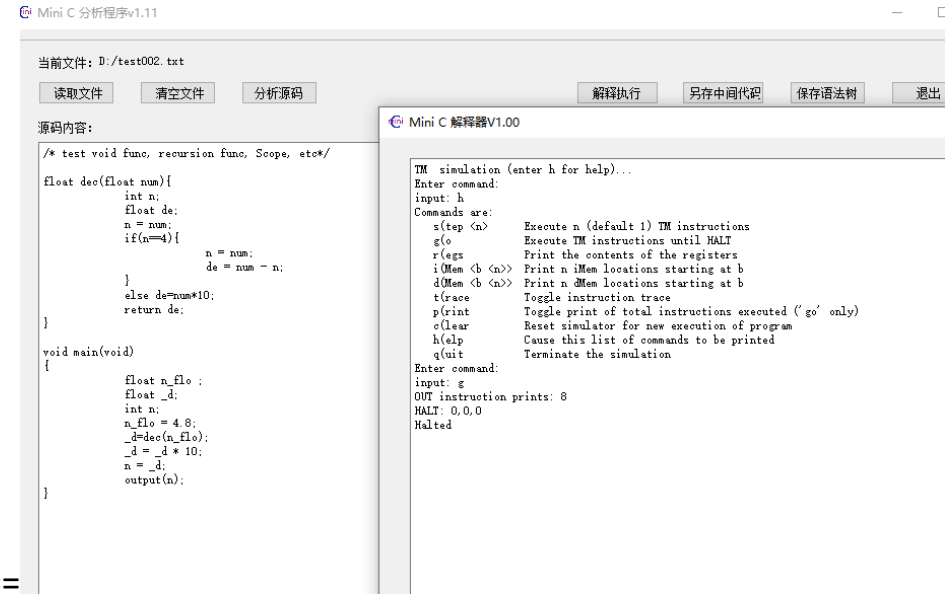



>=

<

当前文件：D:/test002.txt
读取文件
清空文件
分析源码
解释执
源码内容：
语法树
中间代码
/* test void func, recursion func, Scope, etc*/
float dec(float num){
int n;
float de;
if(num<5){
n = num;
de = num - n;
}
else de=num*10;
return de;
}
void main(void)
{
float n_flo ;
float _d;
int n;
n_flo = 4.8;
_d=dec(n_flo);
_d = _d * 10;
n = _d;
output(n);
}
Mini C 解释器V1.00
TM simulation (enter h for help)...
Enter command:
input: h
Commands are:
s(tep <n> Execute n (default 1) T
g(o Execute TM instructions
r(egs Print the contents of t
i(Mem <b <n>> Print n iMem locations
d(Mem <b <n>> Print n dMem locations
t(race Toggle instruction trac
p(rint Toggle print of total i
c(lear Reset simulator for new
h(elp Cause this list of comm
q(uit Terminate the simulatio
Enter command:
input: g
OUT instruction prints: 8
HALT: 0,0,0
Halted

<=

当前文件：D:/test002.txt
读取文件
清空文件
分析源码
解释执行
源码内容：
语法树
中间代码
/* test void func, recursion func, Scope, etc*/
float dec(float num){
int n;
float de;
n = num;
if(n<=4){
n = num;
de = num - n;
}
else de=num*10;
return de;
}
void main(void)
{
float n_flo ;
float _d;
int n;
n_flo = 4.8;
_d=dec(n_flo);
_d = _d * 10;
n = _d;
output(n);
}
Mini C 解释器V1.00
TM simulation (enter h for help)...
Enter command:
input: g
OUT instruction prints: 8
HALT: 0,0,0
Halted

>

当前文件：D:/test002.txt
读取文件
清空文件
分析源码
解释执行
另存中间代码
保存语法树
源码内容：
语法树
中间代码
/* test void func, recursion func, Scope, etc*/
float dec(float num){
int n;
float de;
if(num>4){
n = num;
de = num - n;
}
else de=num*10;
return de;
}
void main(void)
{
float n_flo ;
float _d;
int n;
n_flo = 4.8;
_d=dec(n_flo);
_d = _d * 10;
n = _d;
output(n);
}
Mini C 解释器V1.00
TM simulation (enter h for help)...
Enter command:
input: g
OUT instruction prints: 8
HALT: 0,0,0
Halted

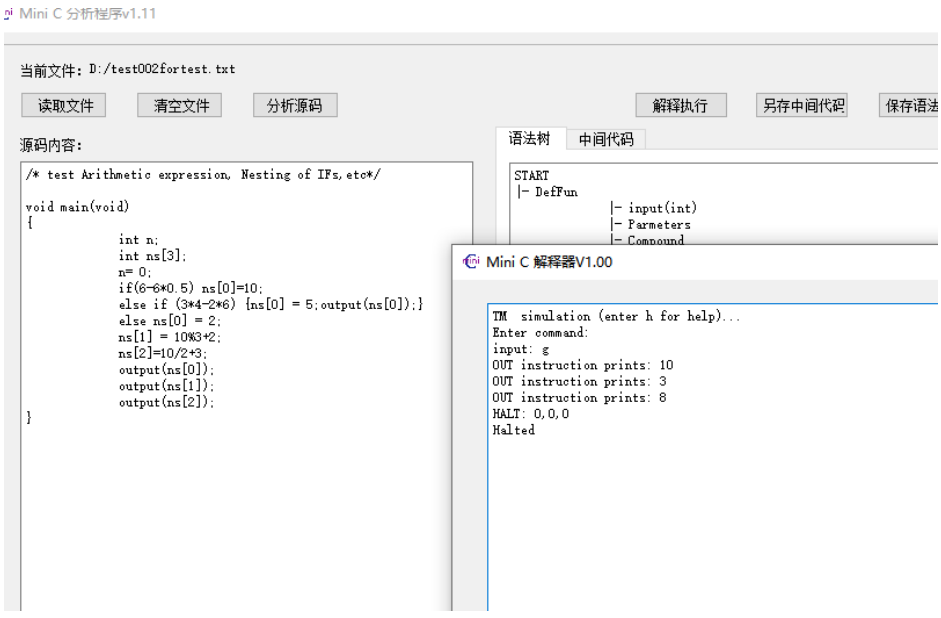



==

结果均可正确输出

test03.txt: 测试 if 嵌套，符号 % / 以及部分运算符号的优先级是否正确

正确则预期为输出 10 3 8




修改第一个 if 内为 0，再测试，若无误则输出为 2 3 8


结果显示测试均准确

# 七、安装部署

1.下载 QT6.0.4 版本。

2.使用 QT creator 新建一个 QT widget application 项目，编译器选择添加 Desktop MinGW 64bit，

主窗口类名使用默认的 mainwindow，不要修改成其他名字。

3.将源文件夹中的所有文件放到项目的文件夹下，将新建项目里默认生成的四个文件：

main.cpp

mainwindow.cpp

mainwindow.h

mainwindow.ui

全部替换成文件夹里的同名文件。

4.将文本编辑器使用的字符集设置为 UTF-8。

5.运行。

# 八、实验总结与心得

经历了两个月的项目工作，从温习编译原理网课，到阅读 Tiny 编译器的代码，到发邮件向老师请教，再到动手撰写各模块的代码，我充分感受到编译原理这一学科的魅力，也理解了为什么它会有计算机“高阶魔法”的美名。在开发过程中我对于编译器各模块的实现原理有了更深的理解，个人的编码能力和软件开发的宏观把控能力也得到了一定的锻炼，这将有利于我以后从事计算机相关的行业。

但即使经过这么长时间的尝试，我还是有没能解决的问题，例如函数的递归调用等。这当然是令人沮丧的事情，但正如梁朝伟在《一代宗师》里说过：“有缺憾，才有进步。”只有发现不足才能清楚自己的短板，明确自己前进的方向，所以也未尝不是好事。