MiniC-CompilingPrinciplesCourse

MiniC-Compiling 是一个在TINY编译程序基础上实现的Mini C语言编译程序.

目录

作品简介

Mini C是一种适合编译器设计方案的语言它比TINY语言更复杂，包括函数和数组. 本质上它是C的一个子集，但省去了一些重要的部分，因此得名.

本Mini C编译器实现功能包括：`Mini C扫描器`（词法分析器），`Mini C语法树生成`（语法分析器、语义分析器），`Mini C代码指令生成`（代码产生器）等功能.

软件需求

1. 根据给出的词法规则实现一个Mini C扫描器（词法分析器）.
2. 根据给出的文法规则设计及实现一个Mini C语法分析器，分析器要产生合适的语法树.
3. 实现Mini C的语义分析器. 分析器的主要要求是，除了在符号表中收集信息外，在使用变量和函数时完成类型检查. 类型检查需要处理的类型是空类型、整型、数组和函数.
4. 实现Mini C的代码产生器，其代码指令与参考资料中的虚拟机一致，代码产生结果在屏幕上显示或以文件的形式保存.
5. 配套修改参考资料中虚拟机程序以实现代码指令的解释执行，并执行得出相应的结果.

Mini C词法规则

```
1. 关键字：else if int return void while
2. 专用符号： + - * / < <= > >= == != = ; , ( ) [ ] { } /* */
3. 其他标记是ID和NUM，正则定义如下：
    ID = letter letter*
    NUM = digit digit*
    letter = a | .. | z | A | .. |Z
    digit = 0 | .. | 9
注：区分大小写
4. 空格由空白、换行符和制表符组成。
5. 注释用C语言符号/*...*/围起来，注释可以凡在任何空白出现的位置（不能放在标记内），可超过一行。注释不能嵌套。
```

Mini C语法规则

```
1. program -> declaration-list
2. declaration-list -> declaration-list declaration | declaration
3. declaration -> var-declaration | fun-declaration
4. var-declaration -> type-specifier ID; | type-specifier ID[NUM];
5. type-specifier -> int | void
6. fun-declaration -> type-specifier ID(params) | compound-stmt
7. params -> param-list | void
8. param-list -> param-list, param | param
9. param -> type-specifier ID | type-specifier ID[ ]
10. compound-stmt -> { local-declarations statement-list }
11. local-declarations -> local-declarations var-declaration | empty
12. statement-list -> statement-list statement | empty
13. statement -> expression-stmt | compound-stmt | selection-stmt | iteration-stmt
| return-stmt
14. expression-stmt -> expression ; |  ;
15. selection-stmt -> if(expression) statement | if(expression) statement else
statement
16. iterstions-stmt -> while(expression) statement
17. reutrn-stmt -> return ; |reutrn expression ;
18. expression -> var=expresion | simple-expression
19. var -> ID | ID[expression]
20. simple-expression -> additive-expression relop addtive-expression | additive-
expression
21. relop -> <= | < | > | >= | == | !=
22. addtive-expression -> additive-expression addop term | term
23. addop -> + | -
24. term -> term mulop factor | factor
25. mulop -> * | /
26. factor -> (expression) | var | call | NUM
27. call -> ID(args)
28. args -> arg-list | empty
29. arg-list -> arg-list, expression | expression
```

功能规划

- 词法分析

  词法分析阶段是编译过程的第一个阶段，是编译的基础。词法分析模块是本实验项目第一个功能模块，核心任务是对用户给出的MiniC源程序转换为字符串形式，根据MiniC语言的此法规则将源程序中的字符扫描、识别出具有独立意义的单词，输出与源程序等价的token流。本模块根据MiniC语言的构词规则，定义了关键字、标识符、常数、运算符及界符等单词的识别。

- 语法分析

  语法分析阶段是编译过程的第二个阶段，也是继词法分析模块之后的功能模块，核心任务是根据已进行文法规则，对词法分析模块中的输出项判断结构上是否符合文法规则，符合时组合成各类语法短语，并以语法树的形式返回，否则返回错误信息。本功能模块使用自顶向下分析，进行语法分析前，将文法规则进行左递归消除和合并左公因子。考虑到MiniC语法复杂性，使用LL(2)文法。

- 输出语法树

  通过编译过程的第二阶段，得到代码的语法树结构。输出语法树，则需根据语法树节点的类型属性及对应的值输出显示。对输出格式有较高要求。在本项目中，节点类型共有两种，分别为StmtK 和 ExpK类型。StmtK类型的节点，用于存储如If、While、FuncK、CompK等类型的结构，ExpK类型节点则存储如Const、Id、Assign、Call等类型的结构。 在对语法树输出中调用了UTILS.C中的printTree(TreeNode * tree)函数，使用DFS遍历树，并输出整颗树的内容。

  本项目的树节点结构及相关的类型定义如下：

```
typedef enum { StmtK, ExpK } NodeKind;

typedef enum { IfK, ReadK, WriteK , WhileK, VarDclK, FunDclK, CompndK,
ReturnK , ParamK} StmtKind;

typedef enum { OpK, ConstK, IdK, AssignK, CallK , ArgsK } ExpKind;

typedef enum { Void, Integer, IntList, VoidList, Boolean } ExpType;

typedef struct treeNode
{
    struct treeNode * child[MAXCHILDREN];
    struct treeNode * sibling;
    int lineno;
    NodeKind nodekind;
    union { StmtKind stmt; ExpKind exp; } kind;
    union {
        TypeToken op;
        TypeToken type;
        int val;
        char * name;
        ArrayAttr arr;
        struct ScopeRec * scope;
    } attr;
    ExpType type;
} TreeNode;
```

- 代码指令产生

  中间代码（Intermediate Representation）是复杂性介于源程序语言和机器语言的一种表示形式。 编译程序中使用的中间代码有多种形式，常见的有逆波兰记号，三元式，四元式，和树形表示。四元式是一种普遍采用的中间代码形式，很类似于三地址指令，有时把这类中间表示称为“三地址代码”，这种表示可以看作是一种虚拟三地址机的通用汇编码，每条“指令”包含操作符和三个地址，两个是为运算对象的，一个是为结果的。 在本项目中，中间代码使用三元组表示，以下为使用到的指令：

```
typedef enum {
 /* RR instructions */
 opHALT,     /* RR     halt, operands are ignored */
 opIN,       /* RR     read into reg(r); s and t are ignored */
 opOUT,      /* RR     write from reg(r), s and t are ignored */
 opADD,     /* RR     reg(r) = reg(s)+reg(t) */
 opSUB,     /* RR     reg(r) = reg(s)-reg(t) */
 opMUL,     /* RR     reg(r) = reg(s)*reg(t) */
 opDIV,     /* RR     reg(r) = reg(s)/reg(t) */
 opRRLim,   /* limit of RR opcodes */

 /* RM instructions */
 opLD,       /* RM     reg(r) = mem(d+reg(s)) */
 opST,       /* RM     mem(d+reg(s)) = reg(r) */
 opRMLim,   /* Limit of RM opcodes */

 /* RA instructions */
 opLDA,      /* RA     reg(r) = d+reg(s) */
 opLDC,      /* RA     reg(r) = d ; reg(s) is ignored */
 opJLT,      /* RA     if reg(r)<0 then reg(7) = d+reg(s) */
 opJLE,      /* RA     if reg(r)<=0 then reg(7) = d+reg(s) */
 opJGT,      /* RA     if reg(r)>0 then reg(7) = d+reg(s) */
 opJGE,      /* RA     if reg(r)>=0 then reg(7) = d+reg(s) */
 opJEQ,      /* RA     if reg(r)==0 then reg(7) = d+reg(s) */
 opJNE,      /* RA     if reg(r)!=0 then reg(7) = d+reg(s) */
 opRALim     /* Limit of RA opcodes */
 } OPCODE;
```

  生成中间代码前，需获得程序的符号表，通过符号表可以获得变量的信息，如变量名、变量类型、变量相对内存的偏移地址等信息，在生成中间代码过程中通过查询符号表，对语法树进行翻译。

- 使用TINY虚拟机执行代码指令

  程序源文件为.minic文件，经过中间代码生成后，获得的三元组表达式的.cm文件，使用TINY虚拟机执行该.cm文件，可获得程序的运行结果。

项目进度

- ☑ 1~2周 项目介绍及分组
- ☑ 3~9周 开发平台及编译工具选择，完成词法分析和语法分析的设计及测试

- ☑ 10~15周 完成语义分析、代码生成功能及测试，完成虚拟机解释功能的修改及测试，完成实验报告的书写和自评，完成系统使用说明书的书写
- ☑ 整理源程序、测试用例、执行程序、使用说明书及项目设计报告书

项目目录树

```
MiniCTest
├─ MiniCTest.pro          //QT项目配置文件
├─ head                   //头文件
│  ├─ ANALYZE.H           //语法分析，生成符号表和函数栈
│  ├─ CGEN.H              //生成代码文件
│  ├─ CODE.H              //生成代码文件
│  ├─ GLOBAL.H            //存放全局变量、函数声明
│  ├─ mainwindows.h       //窗口头文件
│  ├─ PARSE.H             //语法分析函数声明
│  ├─ SCAN.H              //词法分析函数声明
│  ├─ SYMTAB.H            //符号表结构体定义
│  └─ UTIL.H              //全局变量初始化、输出语法树等工具函数声明
├─ source
│  ├─ ANALYZE.C
│  ├─ CGEN.C
│  ├─ CODE.C
│  ├─ GLOBAL.C
│  ├─ mainwindow.c
│  ├─ PARSE.C
│  ├─ SCAN.C
│  ├─ SYMTAB.C
│  └─ UTIL.C
├─ ui
└─ └─ mainwindow.ui       //ui文件
```

测试用例

- sample1

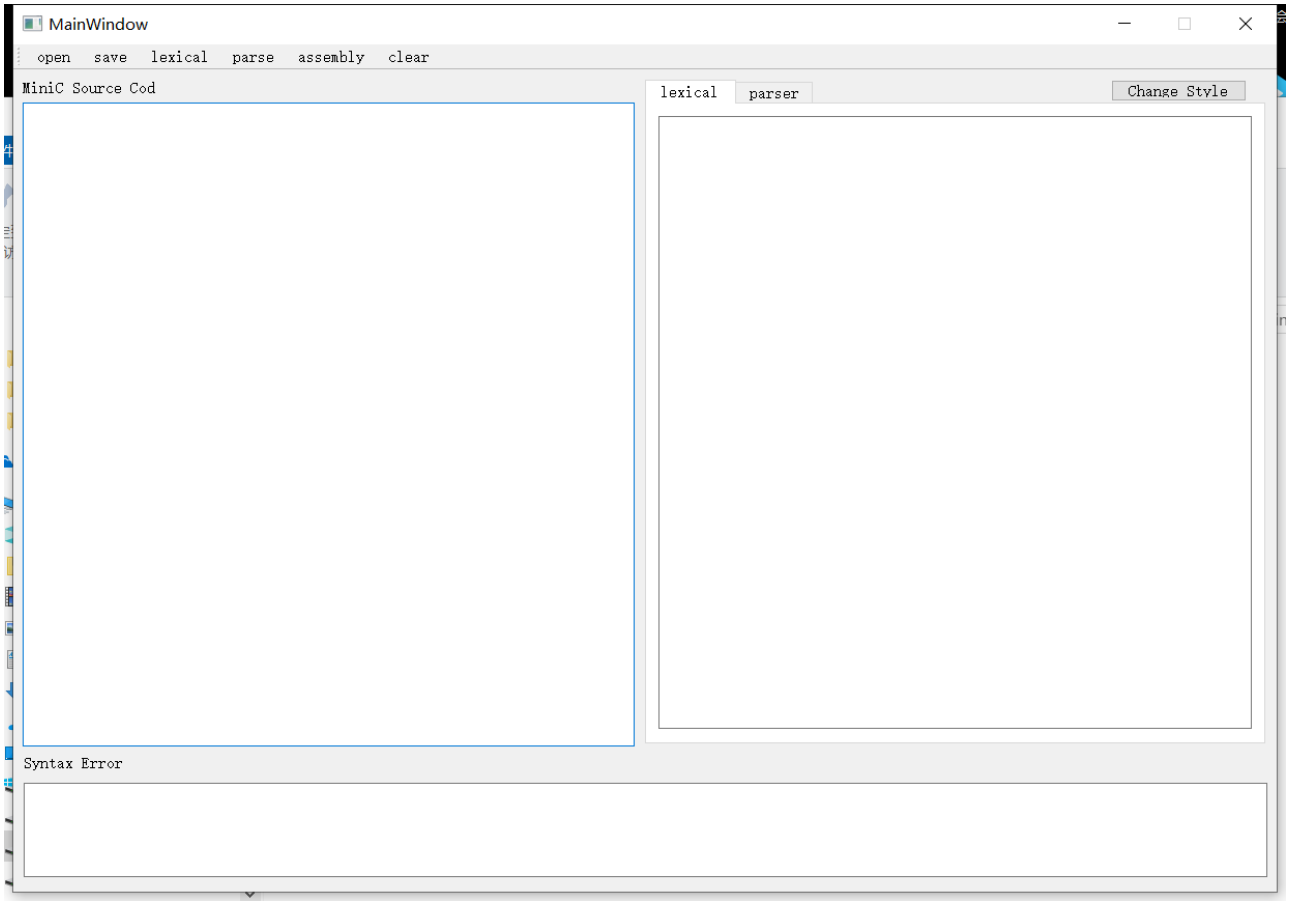
```
/* A program to perform Euclid's
Algorithm to compute gcd.  */
int gcd (int u, int v)
{ if (v == 0)return u;
    else return gcd(v,u-u/v*v);
    /*  u-u/v*v == u mod v */
}

void main(void)
{   int x, int y;
    x=input();
    y=input();
    output(gcd(x,y));
}
```

- sample2

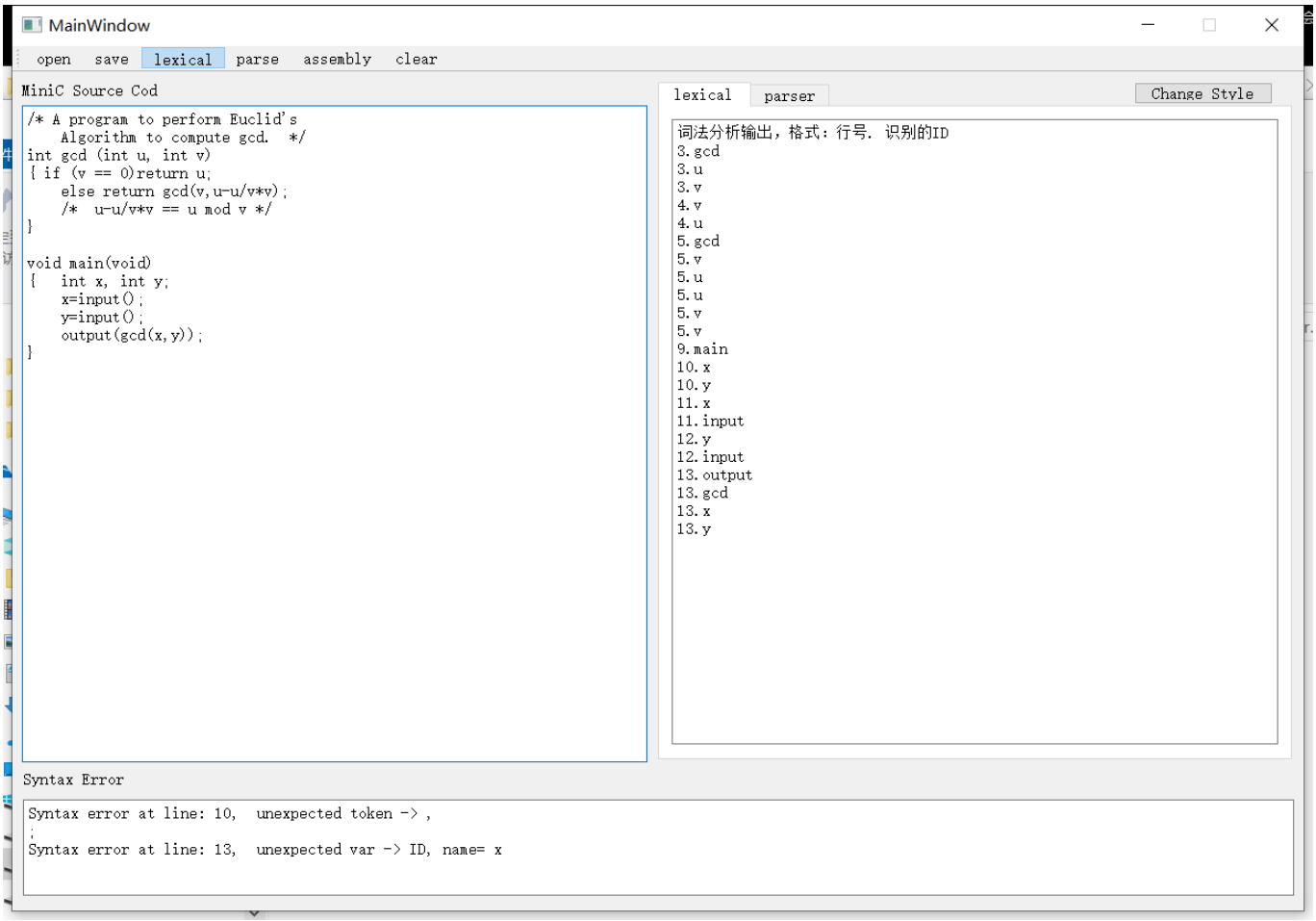
```
/* A program to perform selection sort on a 10
element array。 */
int x[10];
int minloc(int a[], int low, int high)
{ int i; int x; int k;
k=low;
x=a[low];
i=low+1;
while(i<high)
    { if(a[i]< x)
        { x =a[i];
        k=i;
        }
        i=i+1;
    }
    return k;
}

void sort( int a[], int low, int high)
{ int i; int k;
i=low;
while(i<high-1)
    {  int t;
        k=minloc(a,i,high);
        t=a[k];
        a[k]= a[i];
        a[i]=t;
        i=i+1;
    }
}

void main(void)
{   int i;
    i=0;
    while(i<10)
    { x[i]=input();
        i=i+1;
        sort(x,0,10);
        i=0;
        while(i<10)
        { output(x[i]);
        i=i+1;
        }
    }

}
```

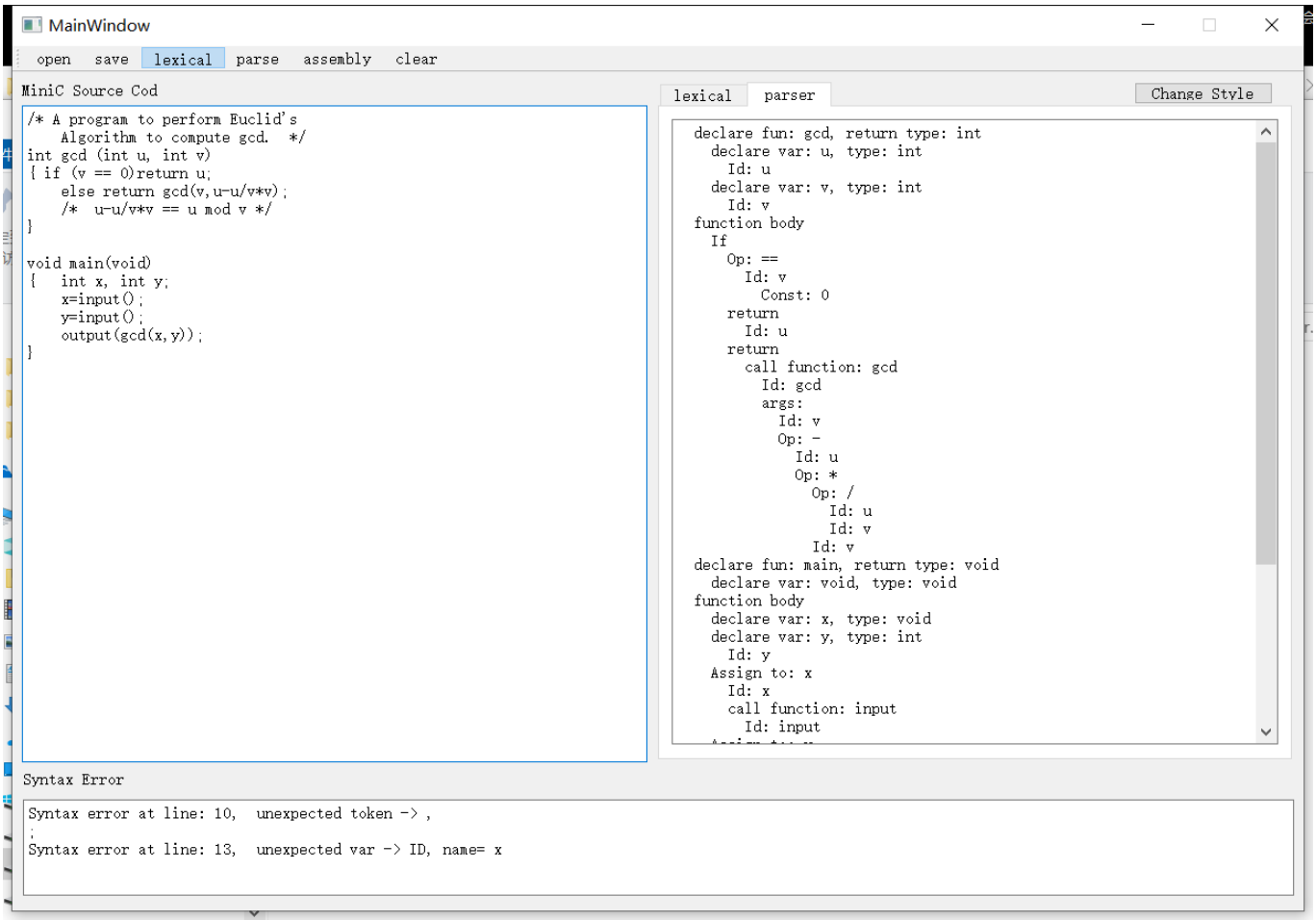
运行效果

- 主界面


MainWindow
open
save
lexical
parse
assembly
clear
MiniC Source Cod
lexical
parser
Change Style
Syntax Error

- sample1

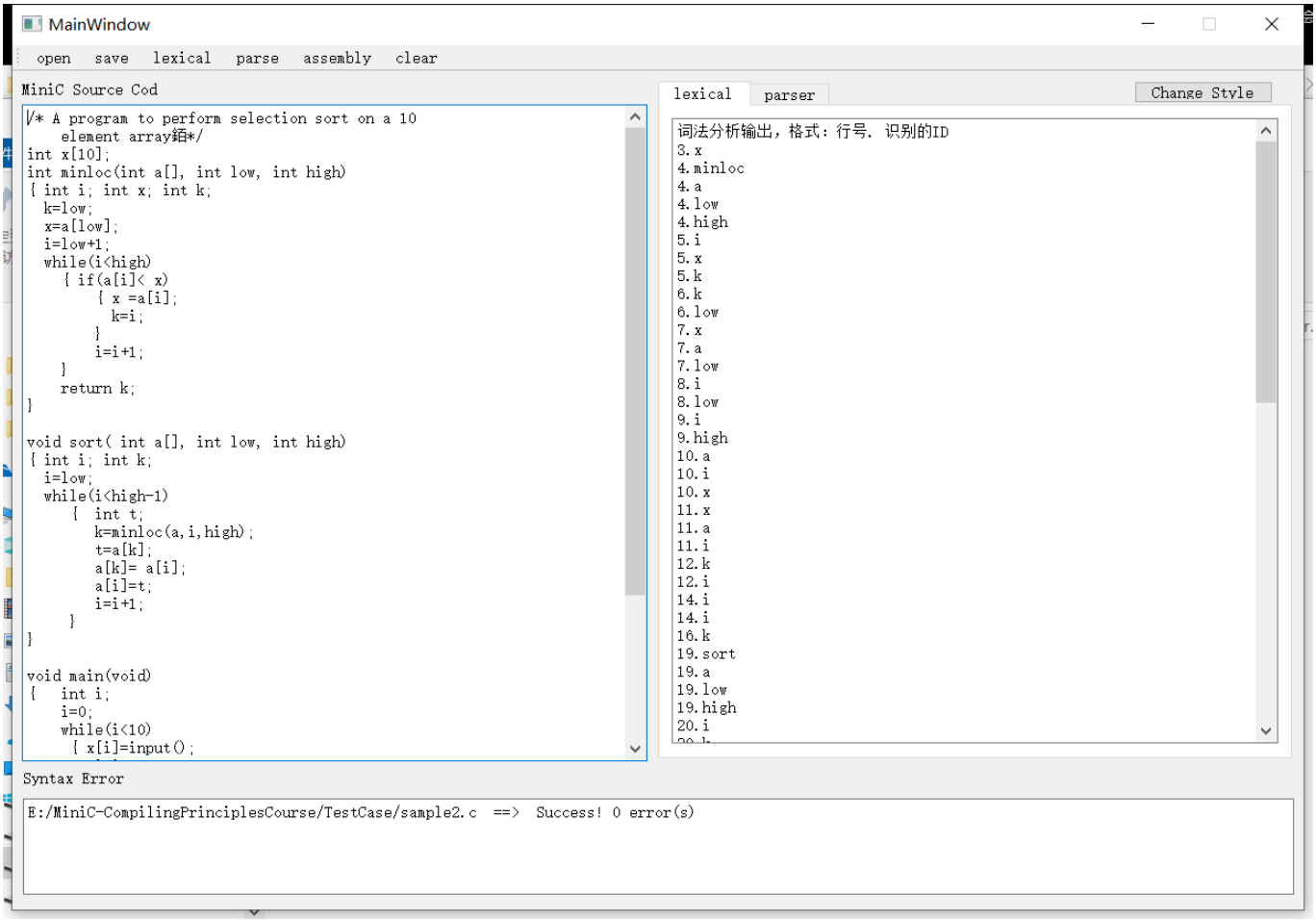

MainWindow
open save lexical parse assembly clear
MiniC Source Cod
/* A program to perform Euclid's
Algorithm to compute gcd. */
int gcd (int u, int v)
{ if (v == 0)return u;
else return gcd(v,u-u/v*v);
/* u-u/v*v == u mod v */
}
void main(void)
{ int x, int y;
x=input();
y=input();
output(gcd(x,y));
}
lexical parser
Change Style
词法分析输出，格式：行号. 识别的ID
3.gcd
3.u
3.v
4.v
4.u
5.gcd
5.v
5.u
5.u
5.v
5.v
9.main
10.x
10.y
11.x
11.input
12.y
12.input
13.output
13.gcd
13.x
13.y
Syntax Error
Syntax error at line: 10, unexpected token -> ,
;
Syntax error at line: 13, unexpected var -> ID, name= x

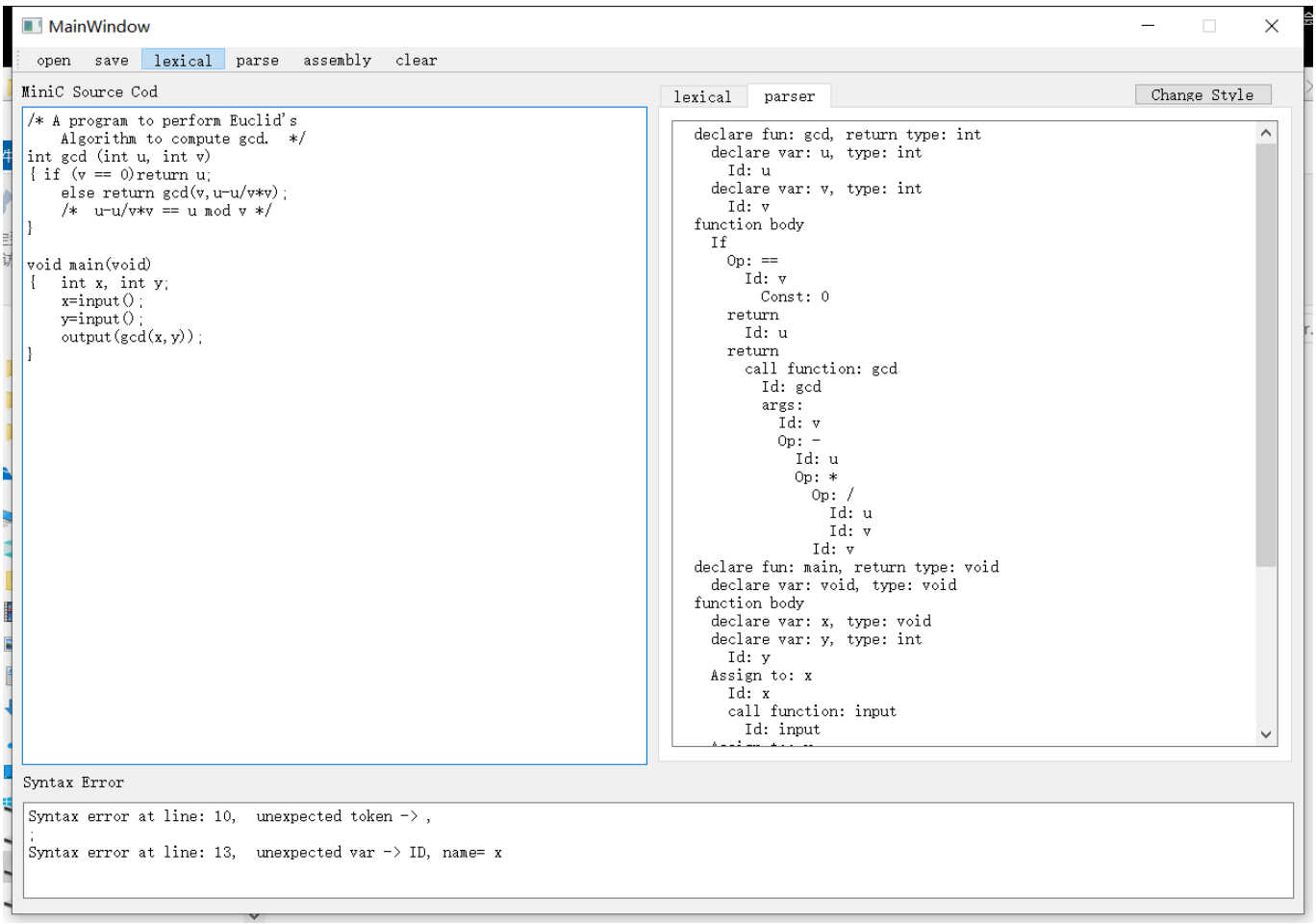

MainWindow
open save lexical parse assembly clear
MiniC Source Cod
/* A program to perform Euclid's
Algorithm to compute gcd. */
int gcd (int u, int v)
{ if (v == 0)return u;
else return gcd(v,u-u/v*v);
/* u-u/v*v == u mod v */
}
void main(void)
{ int x, int y;
x=input();
y=input();
output(gcd(x,y));
}
lexical parser
Change Style
declare fun: gcd, return type: int
declare var: u, type: int
Id: u
declare var: v, type: int
Id: v
function body
If
Op: ==
Id: v
Const: 0
return
Id: u
return
call function: gcd
Id: gcd
args:
Id: v
Op: -
Id: u
Op: *
Op: /
Id: u
Id: v
Id: v
declare fun: main, return type: void
declare var: void, type: void
function body
declare var: x, type: void
declare var: y, type: int
Id: y
Assign to: x
Id: x
call function: input
Id: input
Syntax Error
Syntax error at line: 10, unexpected token -> ,
;
Syntax error at line: 13, unexpected var -> ID, name= x

```
* MiniC Compilation to CM Code
* File: C:/Users/Pietra/Desktop/sample1.cm
* Standard prelude:
  0:     LD  6,0(0)   load gp with maxaddress
  1:    LDC  5,0(0)   copy gp to mp
  2:     ST  0,0(0)   clear location 0
* End of standard prelude.
* -> funDcl (gcd)
  4:     ST  1,2(5)   func: store the location of func. entry
* func: uncondition jump to next declaration belongs here
* func: function body starts here
  3:    LDC  1,6(0)   func: load function location
  6:     ST  0,-1(6)  func: store return address
* -> param (u)
* <- param
* -> param (v)
* <- param
* -> Compnd
* -> if
* -> Op
* -> Id (v)
  7:    LDA  1,0(6)
  8:    LDC  0,-3(0)  id: load varOffset
  9:    ADD  0,6,0  id: calculate the address
 10:     LD  0,0(0)   load id value
* <- Id
 11:     ST  0,-4(6)  op: push left
* -> Const
 12:    LDC  0,0(0)   load const
* <- Const
 13:     LD  1,-4(6)  op: load left
 14:    SUB  0,1,0  op ==
 15:    JEQ  0,2(7)   br if true
 16:    LDC  0,0(0)   false case
 17:    LDA  7,1(7)   unconditional jmp
 18:    LDC  0,1(0)   true case
* <- Op
* if: jump to else belongs here
* -> return
* -> Id (u)
 20:    LDA  1,0(6)
 21:    LDC  0,-2(0)  id: load varOffset
```

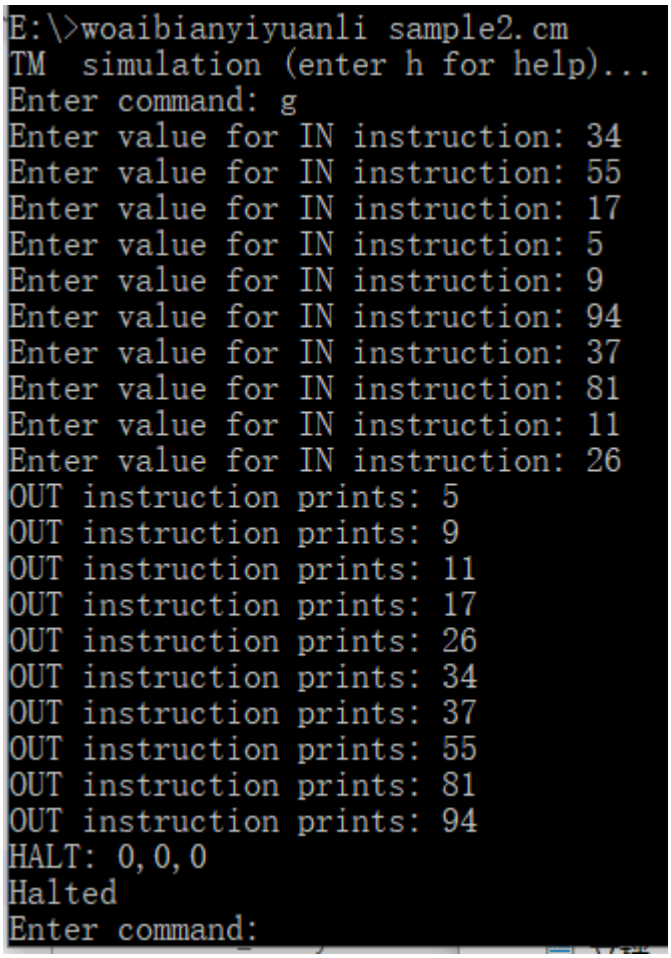
```
E:\>woaibianyiyuanli sample1.cm
TM  simulation (enter h for help)...
Enter command: g
Enter value for IN instruction: 12
Enter value for IN instruction: 8
OUT instruction prints: 4
HALT: 0,0,0
Halted
Enter command:
```

- sample2


MainWindow
open save lexical parse assembly clear
MiniC Source Cod
/* A program to perform selection sort on a 10
element array組*/
int x[10];
int minloc(int a[], int low, int high)
{ int i; int x; int k;
k=low;
x=a[low];
i=low+1;
while(i<high)
{ if(a[i]< x)
{ x =a[i];
k=i;
}
i=i+1;
}
return k;
}
void sort(int a[], int low, int high)
{ int i; int k;
i=low;
while(i<high-1)
{ int t;
k=minloc(a,i,high);
t=a[k];
a[k]= a[i];
a[i]=t;
i=i+1;
}
}
void main(void)
{ int i;
i=0;
while(i<10)
{ x[i]=input();
lexical parser
Change Style
词法分析输出，格式：行号. 识别的ID
3.x
4.minloc
4.a
4.low
4.high
5.i
5.x
5.k
6.k
6.low
7.x
7.a
7.low
8.i
8.low
9.i
9.high
10.a
10.i
10.x
11.x
11.a
11.i
12.k
12.i
14.i
14.i
16.k
19.sort
19.a
19.low
19.high
20.i
Syntax Error
E:/MiniC-CompilingPrinciplesCourse/TestCase/sample2.c ==> Success! 0 error(s)


MainWindow
open save lexical parse assembly clear
MiniC Source Cod
/* A program to perform Euclid's
Algorithm to compute gcd. */
int gcd (int u, int v)
{ if (v == 0)return u;
else return gcd(v,u-u/v*v);
/* u-u/v*v == u mod v */
}
void main(void)
{ int x, int y;
x=input();
y=input();
output(gcd(x,y));
}
lexical parser
Change Style
declare fun: gcd, return type: int
declare var: u, type: int
Id: u
declare var: v, type: int
Id: v
function body
If
Op: ==
Id: v
Const: 0
return
Id: u
return
call function: gcd
Id: gcd
args:
Id: v
Op: -
Id: u
Op: *
Op: /
Id: u
Id: v
Id: v
declare fun: main, return type: void
declare var: void, type: void
function body
declare var: x, type: void
declare var: y, type: int
Id: y
Assign to: x
Id: x
call function: input
Id: input
Syntax Error
Syntax error at line: 10, unexpected token -> ,
;
Syntax error at line: 13, unexpected var -> ID, name= x

```
 0:     LD  6, 0(0)
 1:    LDC  5, 0(0)
 2:     ST  0, 0(0)
 4:     ST  1, 12(5)
 3:    LDC  1, 6(0)
 6:     ST  0, -1(6)
 7:    LDA  1, 0(6)
 8:    LDC  0, -7(0)
 9:    ADD  0, 1, 0
10:    LDA  0, 0(0)
11:     ST  0, -8(6)
12:    LDA  1, 0(6)
13:    LDC  0, -3(0)
14:    ADD  0, 1, 0
15:     LD  0, 0(0)
16:     LD  1, -8(6)
17:     ST  0, 0(1)
18:    LDA  1, 0(6)
19:    LDC  0, -6(0)
20:    ADD  0, 1, 0
21:    LDA  0, 0(0)
22:     ST  0, -8(6)
23:    LDA  1, 0(6)
24:    LDC  0, -2(0)
25:    ADD  0, 1, 0
26:     LD  0, 0, 0
28:    SUB  0, 5, 0
29:     ST  0, -9(6)
30:    LDA  1, 0(6)
31:    LDC  0, -3(0)
32:    ADD  0, 1, 0
33:     LD  0, 0(0)
34:     LD  1, -9(6)
35:    ADD  0, 1, 0
27:    JGT  0, 9(7)
37:     ST  0, -9(6)
```



下载

- Version 1.0

用法

```
运行MiniCTest.exe程序.
点击lexical、parse、assembly按钮生成Token序列、语法树以及中间代码.cm文件.
使用`woaibianyiyuanli.exe`运行生成的中间代码，命令为：woaibianyiyuanli sample1.cm或
woaibianyiyuanli sample2.cm.
```

- 注意：可参见系统使用说明书

开发者

- Contributors

证书

- 参见 LICENSE 文件

版本说明

- Version 1.0

联系我们

联系方式

- E-mail: 20172131134@m.scnu.edu.cn 、 13128684834@163.com