《编译技术》课程设计

文档

学号：\_\_\_\_\_16182682\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_安彪\_\_\_\_\_\_\_\_\_

2019年 1月 4日

## 一．需求说明

### 1．文法说明

＜加法运算符＞ ::= +｜-  
＜乘法运算符＞  ::= \*｜/  
＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  
＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z  
＜数字＞   ::= ０｜＜非零数字＞  
＜非零数字＞  ::= １｜．．．｜９  
＜字符＞    ::=   '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'  
＜字符串＞   ::=  "｛十进制编码为32,33,35-126的ASCII字符｝"                           
＜程序＞    ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞  
＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  
＜常量定义＞   ::=   int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}  
                             | float＜标识符＞＝＜实数＞{,＜标识符＞＝＜实数＞}  
                             | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}  
＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝｜０  
＜整数＞        ::= ［＋｜－］＜无符号整数＞  
＜小数部分＞    ::= ＜数字＞｛＜数字＞｝  
＜实数＞        ::= ＜整数＞.＜小数部分＞  
＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝  
＜声明头部＞   ::=  int＜标识符＞ |float ＜标识符＞|char＜标识符＞  
＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}  
＜变量定义＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']' )}  //<无符号整数>表示数组元素个数，其值需大于0  
＜可枚举常量＞  ::=  ＜整数＞| ＜字符＞  
＜类型标识符＞      ::=  int | float | char  
＜有返回值函数定义＞  ::=  ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'  
＜无返回值函数定义＞  ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'  
＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  
＜参数表＞    ::=  ＜参数＞{,＜参数＞}|＜空＞  
＜参数＞    ::=＜类型标识符＞＜标识符＞  
＜主函数＞    ::= void main'('')' '{'＜复合语句＞'}'  
＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}  //[+|-]只作用于第一个<项>  
＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
＜因子＞    ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'｜＜整数＞|＜实数＞|＜字符＞｜＜有返回值函数调用语句＞|'('＜表达式＞')'           
＜语句＞    ::= ＜条件语句＞｜＜循环语句＞| '{'＜语句列＞'}'| ＜有返回值函数调用语句＞;   
                      | ＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;|＜情况语句＞｜＜返回语句＞;  
＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞  
＜条件语句＞  ::=  if '('＜条件＞')'＜语句＞［else＜语句＞］  
＜条件＞    ::=  ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真  
＜循环语句＞   ::=  while '('＜条件＞')'＜语句＞  
＜情况语句＞  ::=  switch '('＜表达式＞')' '{'＜情况表＞［＜缺省＞］'}'  
＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}  
＜情况子语句＞  ::=  case＜可枚举常量＞：＜语句＞  
＜缺省＞   ::=  default : ＜语句＞  
＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'  
＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'  
＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞  
＜语句列＞   ::= ｛＜语句＞｝  
＜读语句＞    ::=  scanf '('＜标识符＞{,＜标识符＞}')'  
＜写语句＞    ::= printf '(' ＜字符串＞,＜表达式＞ ')'|printf '('＜字符串＞ ')'|printf '('＜表达式＞')'  
＜返回语句＞   ::=  return['('＜表达式＞')']

附加说明：

（1）char类型的变量或常量，用字符的ASCII码对应的整数参加运算

（2）标识符不区分大小写字母

（3）写语句中，字符串原样输出，单个字符类型的变量或常量输出字符，其他表达式按整型或实型输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始

对此文法无扩充和改写。

### 2．目标代码说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | P-CODE | X | Y | 说明 |
| 0 | LDN | X |  | 加载数值1：将数值X存入op\_stack的栈顶 |
| 1 | PSN | X | Y | 加载数值2：将X中的数值存入op\_stack的栈顶，X为run\_stack的地址，Y为0时为静态运行栈，为1时为动态运行栈 |
| 2 | TKN | X | Y | 提取数值：将op\_stack的栈顶中的数值提取至X中，X为run\_stack的地址，Y为0时为静态运行栈，为1时为动态运行栈 |
| 3 | STA | X |  | 存入数组值：将op\_stack栈顶中的值，存入run\_stack中，地址为次栈顶，X为0时为静态运行栈，为1时为动态运行栈 |
| 4 | ADD |  |  | 加法运算：对op\_stack栈顶两个元素进行加法运算，结果存至栈顶 |
| 5 | SUB |  |  | 减法运算：op\_stack次栈顶元素减栈顶元素，结果存至栈顶 |
| 6 | MUT |  |  | 乘法运算：对op\_stack栈顶两个元素进行乘法运算，结果存至栈顶 |
| 7 | DIV |  |  | 除法运算：op\_stack次栈顶元素除以栈顶元素，结果存至栈顶 |
| 8 | BIG |  |  | 大于判断：若op\_stack次栈顶元素大于栈顶元素，退栈并栈顶置一 |
| 9 | LES |  |  | 小于判断：若op\_stack次栈顶元素小于栈顶元素，退栈并栈顶置一 |
| 10 | BOE |  |  | 大于等于：若op\_stack次栈顶元素大于等于栈顶元素，退栈并栈顶置一 |
| 11 | LOE |  |  | 小于等于：若op\_stack次栈顶元素小于等于栈顶元素，退栈并栈顶置一 |
| 12 | EQL |  |  | 等于判断：若op\_stack次栈顶元素等于栈顶元素，退栈并栈顶置一 |
| 13 | UEQ |  |  | 不等于判断：若op\_stack次栈顶元素不等于栈顶元素，退栈并栈顶置一 |
| 14 | MIN |  |  | 变负操作：将op\_stack栈顶元素变负，存入栈顶 |
| 15 | LAB |  |  | 设置标签：记录当前指令位置，存入label\_array栈顶，栈顶+1 |
| 16 | JMP | X |  | 无条件跳转：跳转至label\_array中flag\_jump和X相同的指令 |
| 17 | ZJP | X |  | 条件为零跳转：op\_stack栈顶元素为0跳转至label\_array中flag\_jump和X相同的指令 |
| 18 | SCF | X |  | 输入：格式化输入至op\_stack栈顶，X代表类型，0-char, 1-int, 2-float |
| 19 | PRS | X |  | 输出：输出string\_array[X]字符串的值 |
| 20 | PRE | X |  | 输出：输出栈顶表达式的值，X代表表达式类型，0为字符型，1为整型，2为实型 |
| 21 | RTN |  |  | 有返回值返回：返回值在op\_stack栈顶，直接用 |
| 22 | FED |  |  | 函数结束标志：返回调用函数位置，动态运行栈退一个 |
| 23 | FUS |  |  | 函数调用标志：调用函数位置 |
| 24 | MBG |  |  | 主函数开始位置： |
| 25 | JTM |  |  | 跳转至主函数开始位置： |
| 26 | ATO | X |  | 取数组元素的值：从run\_stack中取值，地址为op\_stack栈顶的值，取值后存入栈顶，X为0时为静态运行栈，为1时为动态运行栈 |
| 27 | JAM | X |  | 判断是否超过数组上届：X为数组上界，与op\_stack栈顶相比，若大于则继续执行，小于等于则停止执行。 |
| 28 | DRS |  |  | 分配动态运行栈空间以便进入函数时使用 |
| 29 | STP | X |  | 传函数参数，X为相对地址 |
| 30 | SWT |  |  | switchcase判断：和eql类似但不删除次栈顶元素 |

## 二．详细设计

### 1．程序结构

整体程序包括：词法分析，语法分析，语法制导翻译，PCODE的生成，解释执行程序，错误处理。

词法分析：获取字符-分析符号串-标识符号串类型；

语法分析：获取符号串类型-判断符号串是否为正确的语法成分，建立符号表并分配存储空间；

语法制导翻译：执行文法中的动作序列；

PCODE的生成：生成源程序的PCODE；

解释执行程序：对PCODE进行解释执行

### 2．类/方法/函数功能

//查表操作 。

int search(char name[])

//填表操作。

int insert(int n)

//进入下一层次。

void lvlup()

//退回至上一层次。

void lvlback()

//用二分法检查是否为保留字。

int isReserve()

//获取一个字符，并将字符补充到Token末尾。

void catToken()

//文件指针回退一个字符。

void retract()

//清楚当前Token中的所有字符。

void clearToken()

//错误处理（词法）。

void error()

//错误处理（语法）。

void grammer\_error(int x)

//判断是否为字母。

int isLetter()

//判断是否为数字。

int isDigit()

//判断是否为无符号整数。

int isIntNum()

//获取字符并判断字符类型以及后续的一些列操作。

void getsym()

//表达式分析。

int express()

//项分析。

int term()

//因子分析。

int factor()

//有/无返回值函数调用。

int function\_use()

//常量说明分析。

void const\_description()

//变量说明分析。

void var\_description()

//复合语句分析。

void compound\_statement()

//if，while条件分析。

int condition()

//语句列分析。

void sentence\_list()

//语句分析。

void sentence()

//有/无返回值函数定义。

void function\_description()

//主函数分析。

void main\_function()

//判断是否为程序中的一部分(常量说明，变量说明，有/无返回值函数，主函数)。

void program()

//PCODE解释执行程序。

void execute()

### 3．调用依赖关系

main()函数调用getsym(),program(),进行对源程序的编译，生成并存储PCODE，随后调用execute()解释执行PCODE。

getsym()调用isLetter(),isDigit(),isIntNum()进行词法分析，调用error()进行报错处理。

program()调用getsym(),const\_description(),var\_description(),function\_description(),main\_function()进行对源程序的初步分析。并调用grammer\_error(int x)进行出错处理。

compound\_statement()调用sentence\_list()，进而调用sentence()，对源程序中的各种语句进行逐一分析，并生成相应指令。其中sentence()会逐步调用express(),term(),factor()，对各类表达式进行分析。

### 4．符号表管理方案

符号表采取结构链表表示：

struct Table

{

char name[100];//标识符名称。

char kind[20];//标识符种类（变量，函数，常量，数组）。

char type[20];//标识符类型（int, char, float）。

int level;//标识符层级。

int addr;//标识符地址。

int array\_max;//数组上限。

int para[100];//函数参数地址。

int para\_type[100];//函数参数类型。

int func\_begin\_addr;//函数地址。

int para\_num;//函数参数个数。

int D\_or\_S;//标识符存需要放在静态/动态运行栈的标志。

struct Table \*next;

struct Table \*pre;

}

对于每个标识符定义，调用insert()函数，向符号表中插入表项。符号表中所包含的标识符为当前函数内的标识符以及全局常量、变量，每读完一个函数时，函数内相应的标识符从符号表中退栈。在引用和定义标识符时，调用search()函数进行查询。

### 5．存储分配方案

运行时采用两个运行栈相互结合的方法进行存储管理，static\_run\_stack为静态运行栈，一维数组；dynamic\_run\_stack为动态运行栈，二维数组。静态运行栈主要存放全局常量、变量、局部常量，动态运行栈存放局部变量，以及各函数的参数。

### 6. 解释执行程序\*

【说明解释执行程序的数据结构，关键算法，及解释执行过程】

解释执行过程主要为以下几步：

1.按当前指令序号读取指令名称及编号

2.进行指令相应操作

3.进行下一条指定指令

解释执行的过程中，还需要用到一个存储临时变量的栈op\_stack，每一个计算、比较、存储、数据的转移等有关数据计算的操作，都需要用到这个栈，并且在每次使用时要格外注意栈顶元素以及栈顶的位置，绝大部分操作是对栈顶元素进行操作的，有一些也包含次栈顶元素，比如数组元素的存储。

### 7. 出错处理

出错包含语法错误，语义错误，以及执行错误。

语法错误为读取非法字符，报错并告知非法字符是什么。

语义错误为语法成分的缺失或不当，报错并告知错误发生的行数，以及缺少的成分。

执行错误主要为数组越界，出现错误报错并停止执行。

只有当执行错误的时候才会停止执行，其他错误都继续编译。

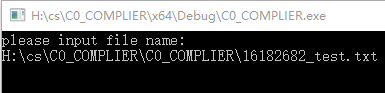
## 三．操作说明

### 1．运行环境

运行环境为vs2017，c语言环境

### 2．操作步骤

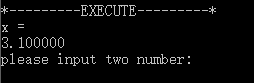
执行程序，按提示输入路径及文件名，执行



随后生成指令并显示，以” \*----------PCODE----------\*”开始



若程序有输入要求，进行输入，显示结果；若无要求，直接显示结果，以” \*---------EXECUTE---------\*”开始



## 四．测试报告

### 1．测试程序及测试结果

（1）fun\_use\_test：

输入：5

输出：

your number is:

5

fib number is:

5

4

3

2

1

2

3

2

1

5

（2）16182682\_test：

输入：1，2

输出：

x =

3.100000

please input two number:

1

2

num[0]:

-8

num[1]:

12

num[2]:

2

51

output is

72.000000

Over!

（3）function\_test：

输入：无

输出：

function1

1

1.100000

function2

（4）identity\_test：

输入：无

输出：

\*\*\*const\*\*\*

1

2

3

1.000000

1.123457

0.123000

a

+

3

123

456

\*\*\*function1\*\*\*

0

1.000000

c

1

1

1

\*\*\*function2\*\*\*

1

1

1.100000

1.100000

a

a

over!

（5）final\_test：

输入：1,1.1,a

输出：

fun\_1's result:

@1 fun\_1\_vi1:

9799

@2 fun\_1\_vi1:

-39

@3 fun\_1\_vi2:

1870

@4 fun\_1\_vf1:

107.699997

@5 fun\_1\_vf1:

4.000000

@6 fun\_1\_vc1:

b

fun\_1 is over!

fun\_2's result:

@1 fun\_2\_ti<=10:

1

@1 fun\_2\_tf<=10:

1.100000

fun\_2\_tc:

a

@2 fun\_2\_tc is 'a'.

fun\_2 is over!

program end!

（6）16182682\_test\_e：

输入：1，2

输出：

语法错误：第66行，条件语句关系比较时不能包含字符。

语法错误：第76行丢失 ' } ' 。

（7）final\_test\_e：

输入：无

输出：

cc3 is undefined!

fun\_1\_vi2 is undefined!

fun\_1\_vi2 is undefined!

（8）fun\_use\_test\_e:

输入：5

输出：over array max!

（9）function\_test\_e：

输入：无

输出：

语法错误：第2行丢失 ' ( ' 。

语法错误：第2行丢失 ' } ' 。

语法错误：第2行，出现不符合预期的符号。;

语法错误：第3行，出现不符合预期的符号。}

（10）identity\_test\_e：

输入：无

输出：

vi\_1 is redefined!

语法错误：第10行丢失 ' ] ' 。

num is undefined!

xxxx is undefined!

语法错误：第27行丢失类型标识符。

语法错误：第27行丢失 ' } ' 。

语法错误：第27行，出现不符合预期的符号。1

语法错误：第27行，出现不符合预期的符号。;

语法错误：第28行，常量声明应在程序开始处。

语法错误：第29行，常量声明应在程序开始处。

语法错误：第30行，出现不符合预期的符号。printf

语法错误：第30行，出现不符合预期的符号。(

语法错误：第30行，出现不符合预期的符号。\*\*\*function2\*\*\*

语法错误：第30行，出现不符合预期的符号。)

语法错误：第30行，出现不符合预期的符号。;

语法错误：第31行，出现不符合预期的符号。vi\_1

语法错误：第31行，出现不符合预期的符号。=

语法错误：第31行，出现不符合预期的符号。xxxx

语法错误：第31行，出现不符合预期的符号。;

语法错误：第32行，出现不符合预期的符号。vi\_2

语法错误：第32行，出现不符合预期的符号。=

语法错误：第32行，出现不符合预期的符号。xxx

语法错误：第32行，出现不符合预期的符号。;

语法错误：第33行，出现不符合预期的符号。vf\_1

语法错误：第33行，出现不符合预期的符号。=

语法错误：第33行，出现不符合预期的符号。vvv

语法错误：第33行，出现不符合预期的符号。;

语法错误：第34行，出现不符合预期的符号。vf\_2

语法错误：第34行，出现不符合预期的符号。=

语法错误：第34行，出现不符合预期的符号。vvv

语法错误：第34行，出现不符合预期的符号。;

语法错误：第35行，出现不符合预期的符号。vc\_1

语法错误：第35行，出现不符合预期的符号。=

语法错误：第35行，出现不符合预期的符号。a

语法错误：第35行，出现不符合预期的符号。;

语法错误：第36行，出现不符合预期的符号。vc\_2

语法错误：第36行，出现不符合预期的符号。=

语法错误：第36行，出现不符合预期的符号。a

语法错误：第36行，出现不符合预期的符号。;

语法错误：第37行，出现不符合预期的符号。printf

语法错误：第37行，出现不符合预期的符号。(

语法错误：第37行，出现不符合预期的符号。vi\_1

语法错误：第37行，出现不符合预期的符号。)

语法错误：第37行，出现不符合预期的符号。;

语法错误：第38行，出现不符合预期的符号。printf

语法错误：第38行，出现不符合预期的符号。(

语法错误：第38行，出现不符合预期的符号。vi\_2

语法错误：第38行，出现不符合预期的符号。)

语法错误：第38行，出现不符合预期的符号。;

语法错误：第39行，出现不符合预期的符号。printf

语法错误：第39行，出现不符合预期的符号。(

语法错误：第39行，出现不符合预期的符号。vf\_1

语法错误：第39行，出现不符合预期的符号。)

语法错误：第39行，出现不符合预期的符号。;

语法错误：第40行，出现不符合预期的符号。printf

语法错误：第40行，出现不符合预期的符号。(

语法错误：第40行，出现不符合预期的符号。vf\_2

语法错误：第40行，出现不符合预期的符号。)

语法错误：第40行，出现不符合预期的符号。;

语法错误：第41行，出现不符合预期的符号。printf

语法错误：第41行，出现不符合预期的符号。(

语法错误：第41行，出现不符合预期的符号。vc\_1

语法错误：第41行，出现不符合预期的符号。)

语法错误：第41行，出现不符合预期的符号。;

语法错误：第42行，出现不符合预期的符号。printf

语法错误：第42行，出现不符合预期的符号。(

语法错误：第42行，出现不符合预期的符号。vc\_2

语法错误：第42行，出现不符合预期的符号。)

语法错误：第42行，出现不符合预期的符号。;

语法错误：第43行，出现不符合预期的符号。}

语法错误：第44行，变量声明应在函数声明前，常量声明后。

### 2．测试结果分析

测试程序前五个为正确测试程序，从各个方面测试了程序的准确性。

16182682\_test综合测试了各方面的编译执行准确性，包含了if,while,switch等多种语句，也包含了各类计算。

Fun\_use\_test主要为斐波那契数列的测试，主要测试函数递归调用。

Final\_test主要测试了各类表达式的计算以及各类输出语句，也包含了其他的语句。

Function\_test主要测试了函数调用功能。

Identity\_test主要测试了标识符的各类定义以及赋值。

后五个测试程序为错误程序，输出结果符合预期。

16182682\_test\_e测试了比较表达式两侧的类型匹配，也测试了字符的缺失。

final\_test\_e测试了标识符未定义的出错。

fun\_use\_test\_e测试了数组越界。

function\_test\_e测试了字符缺失。

identity\_test\_e综合测试了字符缺失，标识符未定义，标识符重定义等问题。

以上测试结果均符合预期。

## 五．总结感想

经过半学期的学习，无论是理论知识还是动手实践，都让我对编译技术有了深入的了解。现在回想起来觉得，应该在学期初的时候更加用功的完成自己的编译程序，这样在期末的时候就有时间去完善去debug。在提交截止前的几个小时我才算改的没什么太大问题，搞得自己很紧迫。并且随着程序的完成，也越来越理解到理论知识与实践结合的重要性，如果之前理论知识学的很透彻的话，之后就不会走很多弯路，也不会改来改去。写到现在给我最大的感受就是在完成一个大型项目之前，一定要统筹规划好，大体安排得当之后再进行小修小补就容易很多。