

**课 程 实验 报 告**

**题目： mini-c编译器的实现**

**课程名称： 编译技术实验**

**专业班级： 软件2001**

**学 号： U202012098**

**姓 名： 余锦琪**

**指导教师： 祝建华，胡雯蔷**

**报告日期： 2023-1-20**

**软件学院**

**目录**

目录

[1概述 1](#_Toc125151330)

[2系统描述 2](#_Toc125151331)

[2.1 自定义语言概述 2](#_Toc125151332)

[2.2单词文法与语言文法 2](#_Toc125151333)

[2.2.1 单词文法 2](#_Toc125151334)

[2.2.1 语言文法 3](#_Toc125151335)

[2.3 符号表结构定义 4](#_Toc125151336)

[2.4 错误类型码定义 4](#_Toc125151337)

[2.5中间代码结构定义 5](#_Toc125151338)

[2.6目标代码指令集选择 5](#_Toc125151339)

[3系统设计与实现 6](#_Toc125151340)

[词法分析器 6](#_Toc125151341)

[语法分析器 7](#_Toc125151342)

[符号表管理 9](#_Toc125151343)

[语义检查 10](#_Toc125151344)

[中间代码生成 11](#_Toc125151345)

[汇编代码生成 13](#_Toc125151346)

[4系统测试与评价 14](#_Toc125151347)

[正确性测试 14](#_Toc125151348)

[报错功能测试 19](#_Toc125151349)

[系统的优点 23](#_Toc125151350)

[系统的缺点 24](#_Toc125151351)

[5实验小结或体会 25](#_Toc125151352)

[**参考文献** 26](#_Toc125151353)

# 1概述

本次实验是构造一个高级语言的子集的编译器，目标代码是汇编语言。按照任务书，实现的方案可以有很多种选择。

可以根据自己对编程语言的喜好选择实现。建议大家选用decaf语言或C语言的简单集合SC语言。

实验的任务主要是通过对简单编译器的完整实现，加深课程中关键算法的理解，提高学生系统软件研发技术。

# 2系统描述

## 2.1 自定义语言概述

在mini-c中，程序由外部列表组成，外部列表是一系列定义语句，可以定义变量，但是不能给变量赋值，同时可以定义一系列函数，其中必须要有一个main函数，作为程序运行的入口。函数体是一个复合语句，内部包含了多给声明和其他的语句。需要注意的是，函数内部的变量声明必须写在其他语句之前。其他语句包含：复合语句、for语句、while语句、if-then语句、if-else-then语句等等。还有后续添加的break语句和continue语句。在mini-c中，我主要花了大量时间实现数组。

## 2.2单词文法与语言文法

## 2.2.1 单词文法

单词的文法定义在lex.l文件中，它能识别整形常量、浮点数常量、字符常量、语言内置标识符、自定义标识符id、运算符和c语言中大部分的运算符号。

语言内置标识符例如：

"return"    {return RETURN;}

"if"        {return IF;}

"else"      {return ELSE;}

"while"     {return WHILE;}

"for"       {return FOR;}

"continue"  {return CONTINUE;}

运算符号例如：

">=" {strcpy(yylval.type\_id, yytext);return GE;}

"&&"    {return AND;}

">="    {strcpy(yylval.type\_id, yytext);return GE;}

"("     {return LP;}

"{"     {return LC;}

## 2.2.1 语言文法

语言文法为

G[program]:

program → ExtDefList

ExtDefList→ExtDef ExtDefList | ε

ExtDef→Specifier ExtDecList ; |Specifier FunDec CompSt

Specifier→int | float

ExtDecList→VarDec | VarDec , ExtDecList

VarDec→ID

FucDec→ID ( VarList ) | ID ( )

VarList→ParamDec , VarList | ParamDec

ParamDec→Specifier VarDec

CompSt→{ DefList StmList }

StmList→Stmt StmList | ε

Stmt→Exp ; | CompSt | return Exp ;

| if ( Exp ) Stmt | if ( Exp ) Stmt else Stmt | while ( Exp ) Stmt |continue ; | break;

DefList→Def DefList | ε

Def→Specifier DecList ;

DecList→Dec | Dec , DecList

Dec→VarDec | VarDec = Exp

Exp →Exp =Exp | Exp && Exp | Exp || Exp | Exp < Exp | Exp <= Exp

| Exp == Exp | Exp != Exp | Exp > Exp | Exp >= Exp

| Exp + Exp | Exp - Exp | Exp \* Exp | Exp / Exp | ID | INT | FLOAT

| ( Exp ) | - Exp | ! Exp | ID ( Args ) | ID ( ) |ArrVar

ArrVar→ID [Exp] | ArrVar[Exp]

Args→Exp , Args | Exp

其中大部分为实验教程中已经定义好的文法，我自己添加的部分有：break语句，continue语句，数组变量（相当于数组元素）等等。在上述文法中，我用表格简要说明各个符号的含义：

表2-1 语法符号说明

|  |  |
| --- | --- |
| program | 程序 |
| ExtDefList | 外部定义列表 |
| ExtDef | 外部定义 |
| Specifier | 标识符 |
| ExtDecList | 外部声明列表 |
| VarDec | 变量声明 |
| FucDec | 函数声明 |
| VarList | 变量列表 |
| ParamDec | 参数声明（形参） |
| CompSt | 复合语句 |
| StmList | 语句列表 |
| Stmt | 语句 |
| DefList | 定义列表 |
| Dec | 声明 |
| Exp | 表达式 |
| ArrVar | 数组变量 |
| Args | 实参 |

## 2.3 符号表结构定义

表2-2 符号表结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 符号名 | 别名 | 类型 | 种类 | 其他信息 |
|  |  |  |  |  |

在其他信息中有指向下一层的指针，函数形参数，函数和数组所占空间大小、数组的维度等信息。

## 2.4 错误类型码定义

定义了许多错误，但是没有定义错误类型码：

（1）使用未定义的变量；

（2）调用未定义或未声明的函数；

（3）在同一作用域，名称的重复定义（如变量名、函数名、结构类型名以及结构体成员名等）。为更清楚说明语义错误，这里也可以拆分成几种类型的错误，如变量重复定义、函数重复定义等；

（4）对非函数名采用函数调用形式；

（5）对函数名采用非函数调用形式访问；

（6）函数调用时参数个数不匹配，如实参表达式个数太多、或实参表达式个数太少；

（7）函数调用时实参和形参类型不匹配；

（8）对非数组变量采用下标变量的形式访问；

（9）数组变量的下标不是整型表达式；

（10）赋值号左边不是左值表达式；

（11）对非左值表达式进行自增、自减运算；

（12）类型不匹配。强类型，不同类型的变量在一起运算就会类型不匹配。

（13）函数返回值类型与函数定义的返回值类型不匹配；

（14）break语句不在循环语句或switch语句中；

（15）continue语句不在循环语句中；

## 2.5中间代码结构定义

中间代码是一个四元数：（操作符，操作数1，操作数2，结果）；

这样的结果与MIPS指令的结构十分类似，有利于转化为MIPS指令。

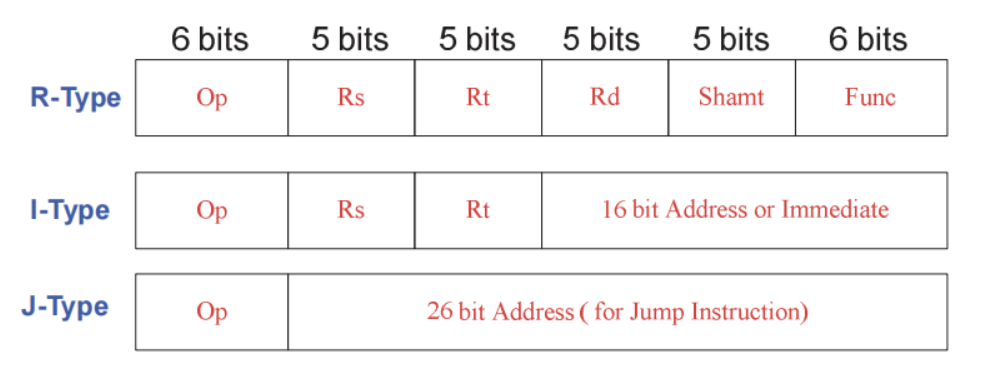


图2-1 MIPS指令格式

## 2.6目标代码指令集选择

MIPS指令集。

# 3系统设计与实现

（写清楚自己的实现步骤，详细介绍完成该步骤的目标及思路，以及在程序开发时出现错误时的调整过程）

### 词法分析器

通过flex实现，先在parser.ypp中声明一系列符号token，在flex中，遇到相应的字符串，应该返回定义的符号，例如：

"return"    {return RETURN;}

"if"        {return IF;}

"else"      {return ELSE;}

">=" {strcpy(yylval.type\_id, yytext);return GE;}

"&&"    {return AND;}

">="    {strcpy(yylval.type\_id, yytext);return GE;}

这一部分指导教程中给的代码功能已经非常全面了，由我自己完成的部分是char类型和char常量的词法定义，break，continue的词法识别。

在实现char常量的词法定义时我遇到了一个困扰许久的问题：

1. 首先我在flex文件中的YYLVAL中添加一个变量type\_char,它的类型为char，类似于int和float

typedef struct {

int type\_int;

int type\_float;

char type\_id[32];

char type\_char;

} YYLVAL;

1. 然后我在我的bison文件的YYLVAL中也添加这么一个变量

typedef struct YYLVAL

 {

    int         type\_int;

float       type\_float;

char        type\_char;

    char        type\_id[32];

1. 最后我就惊讶地发现我几乎所有的类型都不匹配了

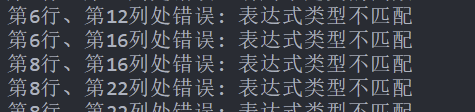


图3-1 类型不匹配

（4）经过我让程序打印各个变量的类型，然后苦苦debug，我才发现原来是上面两个YYLVAL里变量的顺序出问题了，程序中这两个结构体指向的是同一个地址，然后我这里变量的顺序不同，因此存进去的信息也发生了错乱。只需要纠正顺序，程序便可正常运行。

### 语法分析器

同样，语法分析器中的内容大部分也已经由指导教程给出，我这里只是做了部分完善，比如for语句，数组变量，数组的常量表达式，continue，break语句。

这里主要展示一下数组的部分：

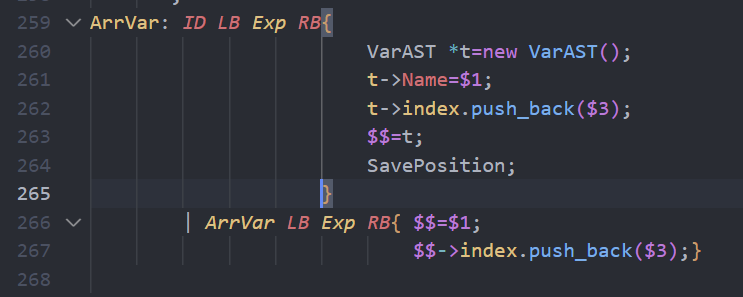


图3-2 数组变量

我认为数组变量仍然是VarAST类型的所以不需要继续添加其他的类。数组变量可以出现多个下标，例如a[b][c][c+1]。如265行代码所示，数组变量后面每多一个下标，就把这个下标表达式加到VarAST::index里面。

接下来是数组常量：

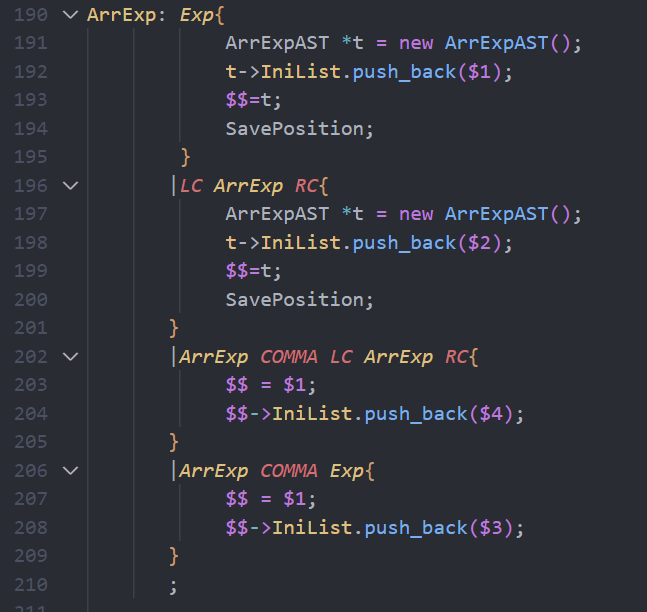


图3-3 数组常量表达式

数组常量表达式比较复杂，这里添加了一个ArrExpAST类，它继承自ExpAST,这个类里面有一个vector<ExpAST\* > IniList。由于数组常量表达式里面会出现大量的表达式，这些表达式甚至还有可能是数组，这样就形成了多维数组，例如{{1，2}，{3，4}}，这个时候数组的IniList内有两个表达式，而且这两个表达式还是数组常量表达式，然后这两个数组常量表达式的IniList也都有两个表达式，但这时表达式树的类型为常量。由于时间紧迫，到这里数组常量表达式就已经结束了，我没有继续往后面做，但是如果往后面实现，我还可以实现数组的直接初始化，而不是一个元素一个元素地写入数组。

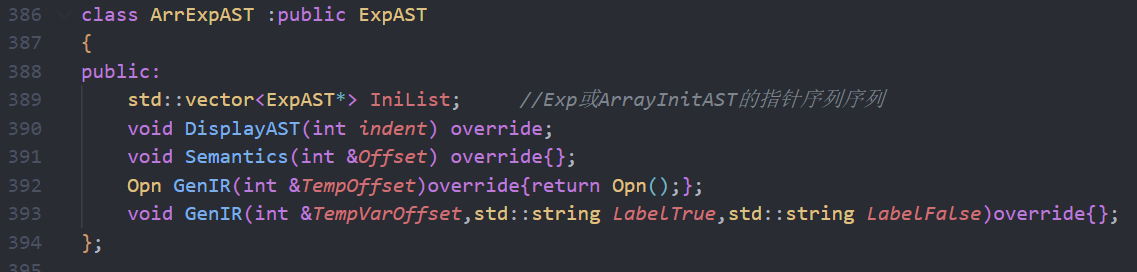


图3-4 ArrExpAST类

这里For语句、break、continue等语句的实现都非常简单，但是这里要添加三个类ForStmAST、BreakStmAST、ContinueStmAST。ForStmAST和While语句类似，这里不再赘述，直接给代码贴图。

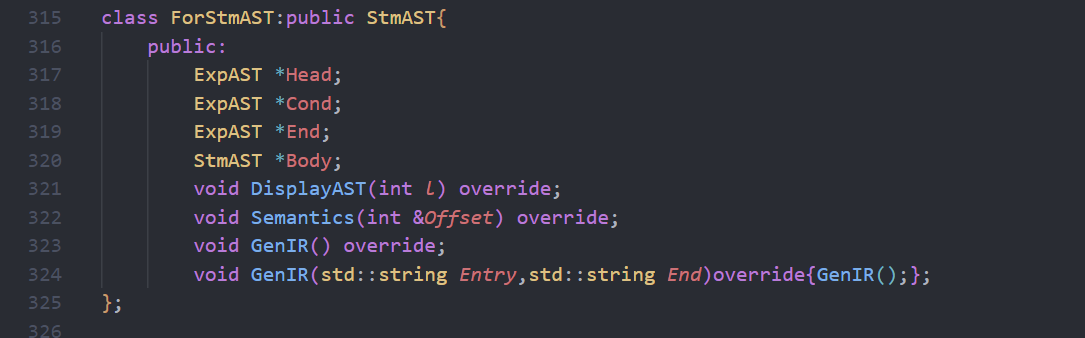


图3-5 For语句节点类

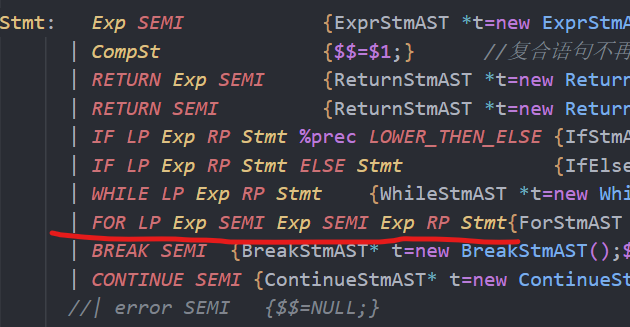


图3-6 for语句语法定义

自此，声明语法分析完成，已经生成了一颗抽象语法树，接下来的步骤就是对语法树进行遍历，得到想要的结果。

### 符号表管理

只有在遇到变量或函数的声明或者定义时，才需要向符号表添加成员，在初始化时，指导教程的代码向符号表添加了两个函数read和write，还有一个变量x。在遇到变量时，生成一个变量符号VarSymbol，生成一个别名，根据当前的OffSet，给变量符号的Offset赋值……总而言之，就是完善这个符号的信息，然后把这符号插入到当前作用域的符号表。在遇到复合语句时，会生成一个作用域（也就是一个符号表），进符号作用域栈（这里面有多个符号表），然后该复合语句内部的所有语句都是基于该作用域对于的符号表进行符号表管理，从复合语句出来时，该作用域出栈。如果是遇到函数定义，函数符号先进入符号表，然后同样生成一个作用域，函数会先把它的参数添加到这个新的作用域。在定义数组，表数组名当成变量处理，然后计算数组的维数和所需空间大小。各符号项的定义如图3-7，这里把数组和变量一样用VarSymbol表示，但是如果是数组，还需要写入ARsize（数组所需空间）、dims（数组各维度的大小）这两个信息：



图3-7 符号项的定义

### 语义检查

语义检查是和符号表管理同时进行，它更是基于符号表进行，在进行语义检查时，会一边查符号表，如果出现重复定义，类型不正确，使用未定义符号等错误时，都可以根据符号表内容查找出相应的错误。

### 中间代码生成

中间代码生成是在语义分析之后，大部分代码已经给出，我所做的工作主要是数组部分的工作，由于数组变量对应的节点是VarAST，所以工作也主要集中在VarAST。在VarAST内，我在已有代码的基础上添加了一个Opn成员ArrOff，这个成员用于记录数组变量相对于数组首地址的偏移量。



图3-8 VarAST

然后在它的GenIR成员函数中进行处理，相关代码如下：



图3-9 VarAST::GenIR部分内容

根据符号表，当遇到的变量为数组元素，且没有语法错误时，这个时候我们需要计算这个数组元素的地址相对于数组首地址的偏移，这部分都由目标代码进行，这里的工作是生成目标代码，由生成的目标代码进行详细的计算：

首先，声明一个临时变量Arr，用来存储偏移值，在276行，给偏移值初始化为0，然后对每一个索引表达式进行处理。先求生成代码，求出索引表达式运算结果的地址，如果索引表达式不是数组，那么这个地址存储的值就是该索引的值，于是把该值赋给一个临时变量Temp。有了索引的值，接下来只需要根据数组的维度计算偏移量。但是，如果索引也是数组，它必然是VarAST类，那么根据索引表达式的地址（这个地址是数组的首地址），和VarAST里的ArrOff，生成中间代码IRCode（READARRAY，数组起始地址，数组元素偏移量，临时变量），这个代码根据数组的起始地址和数组元素的偏移量，取出数组元素的值，然后把该值赋给Temp，这个值就是索引的值。最终的数组变量的偏移量存储在Arr中，把Arr赋值给VarAST::ArrOff。我写的这些代码只负责生成中间代码和目标代码，实际的数组元素的偏移量计算都由目标代码完成。

如果遇到给数组变量赋值，我们看到赋值语句这部分的代码，用这部分代码作为示例，展示如何对数组进行读写：



图3-10 赋值语句的中间代码生成

如果赋值表达式的右操作数是数组元素，如353-355行，读出数组元素到一个临时变量；如果不是数组元素，那么直接把右表达式的值赋给临时变量。由这个临时变量进行接下来的处理。

读出右操作数后，需要把右操作数写入到左操作数中，如果左操作数不是数组，那么可以直接把临时变量赋值给左操作数；反之，要生成中间代码IRCode(WRITEARRAY,临时变量,数组起始地址,数组元素的偏移量)。这样就可以把临时变量写入到数组元素所在的地址空间。

对于其他的一些需要用到数组的地方，以同样的方式先判断是否为数组，然后再读写数组。

### 汇编代码生成

同样的，汇编代码大部分也已经给出，我这里只描述我完善的部分，代码如图3-11所示：

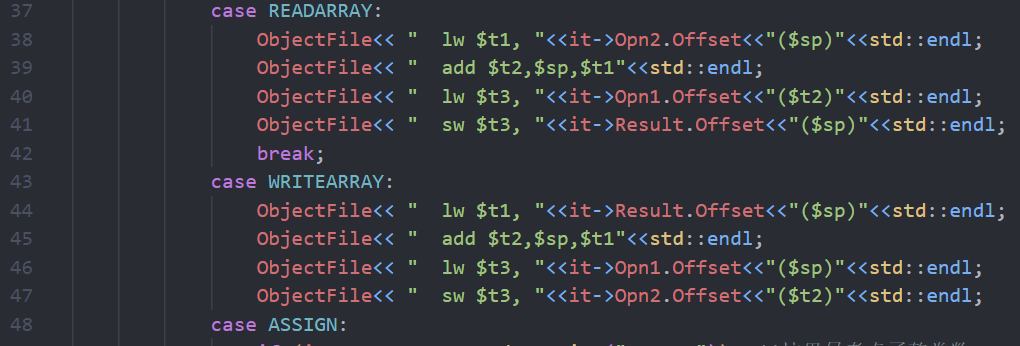


图3-11 汇编代码生成

我完成的部分主要是READARRAY和WRITEARRAY。在中间代码的部分，这两个操作符都有对应的数组起始地址，数组元素偏移量，临时变量。READARRAY的目标代码为，取数组元素偏移量到$t1,$t2=($t1)+($sp)……总之就是从数组元素所在的地址取值，然后把值写到临时变量所在的地址。写操作与之类似。

# 4系统测试与评价

### 正确性测试



图4-1数组、for语句自增自减正确性测试

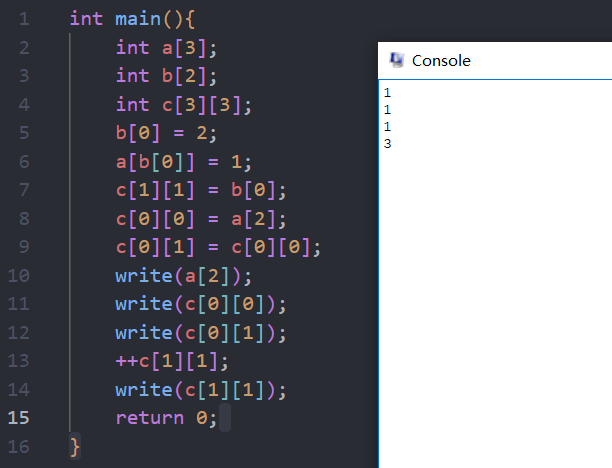


图4-2 多维数组正确性测试

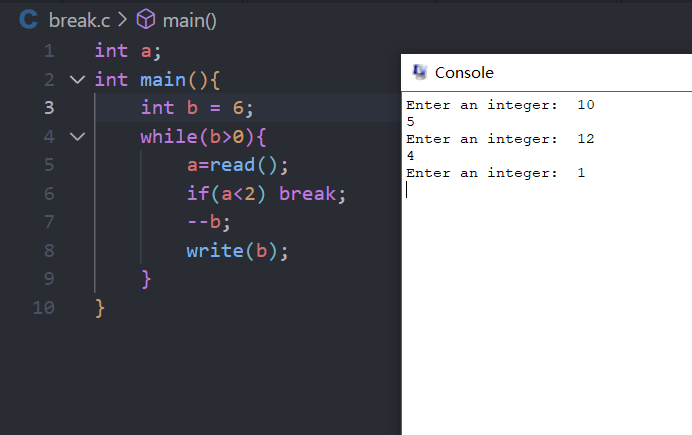


图4-3 break语句正确性测试

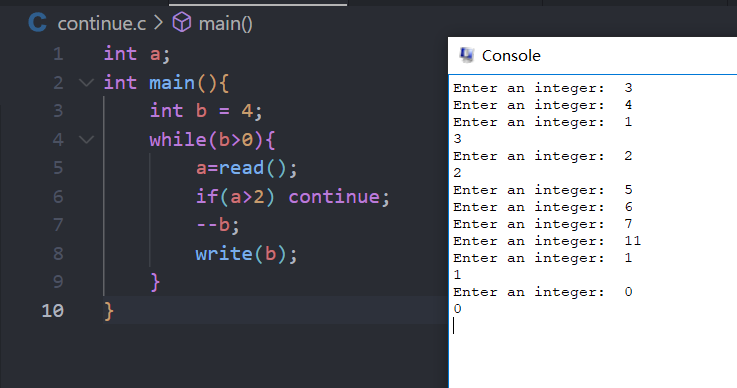


图4-4 continue语句正确性测试

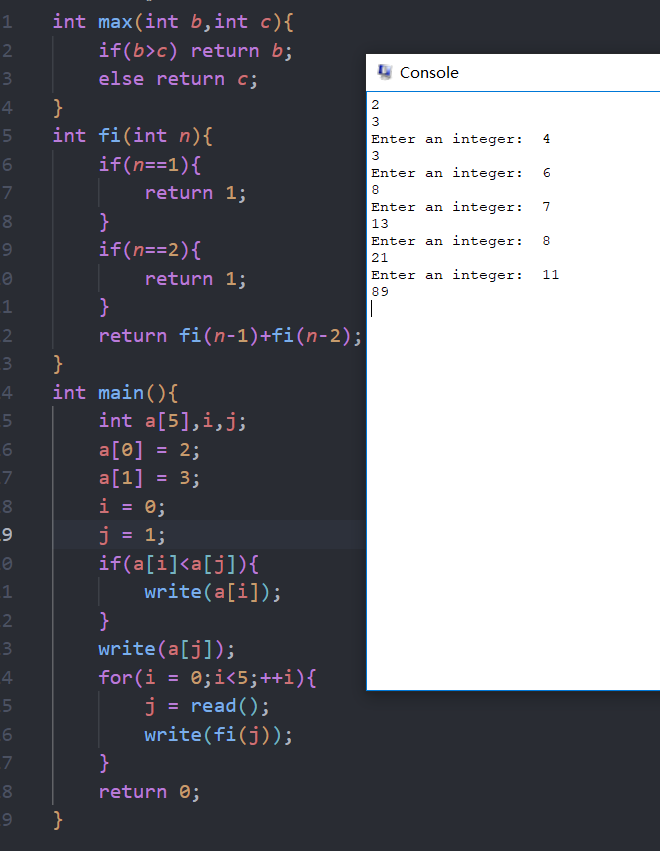


图4-5 函数正确性测试（含递归）

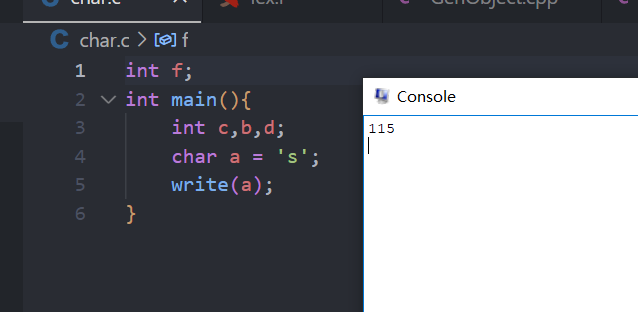


图4-6 char类型支持测试

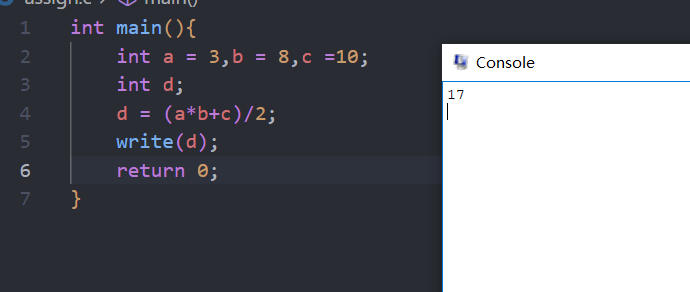


图4-7 表达式运算正确性测试

### 报错功能测试



图4-8 函数和数组相关报错测试



图4-9 运算符错误测试



图4-10 break错误测试



4-11 continue错误测试

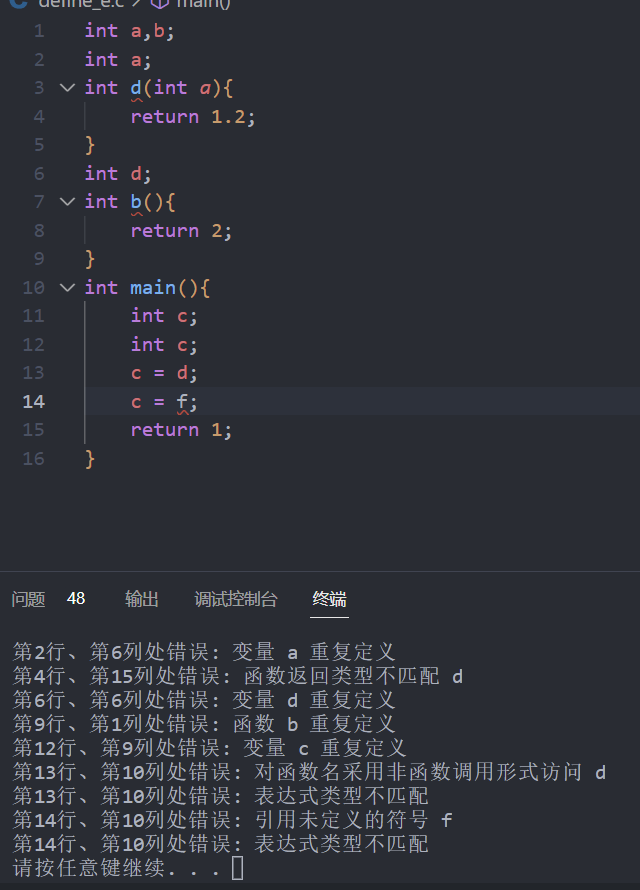


图4-12 定义相关的运算错误测试

### 系统的优点

功能上：系统实现了数组、多维数组，char、float、int三种常量，以及c语言里面的多种语句，能够识别各种语法错误，实现了编译器的基本功能，称得上一个mini-c。

结构上：采用了C++实现，面向对象的实现方法，尽管还有部分功能没有实现，但是后面重构添加功能比较简单，有利于重构。

### 系统的缺点

没有考虑中间代码的优化，数组的直接初始化没有实现，结构体没有实现，仍然有许多功能没有实现。

由于时间仓促，代码的结构非常差，可读性不强，需要后面进行重构。

# 

# 5实验小结或体会

经过本次课程设计，花费了几天几夜除了吃喝睡，实现这个mini-c的过程，就好像打怪升级，从最初的词法、语法分析，到后面语义分析，中间代码生成，一步一步走来，遇到过各种各样的bug。第一次遇到这么大的工程量，写到现在，心力交瘁，虽然现在已经完成了大部分功能，但是代码还没有停止，bug也还没有停止。通过一个编译器的实现，我现在更能理解编程语言的内部工作原理，这使得我们去接触一门新的语言时更容易上手。

在这个过程中，我先实现了所有的词法语法分析，包括数组表达式的语法分析，然后先是实现了for语句，再慢慢地实现break，continue等功能，同时，补充了语义分析的代码，使得报错功能更加全面。中间不知遇到过多少错误，最后实现了数组的基本功能，包含获取数组元素的值，写入数组元素，数组元素作为数组索引，表达式作为数组索引等等。但遗憾的是，数组的直接初始化停留在了一半，或许这也是我后面继续完善这个编译器的一个动力。

本次实验后，我的工程能力得到了加强，对编译原理的理解更加透彻，但未来还会遇到许许多多的bug，希望每一次都能像这次实验这样，一步一步突破！

**错误分析：**

实验中，我最经常碰到的一个错误，如图5-1，这里是正确的示例：

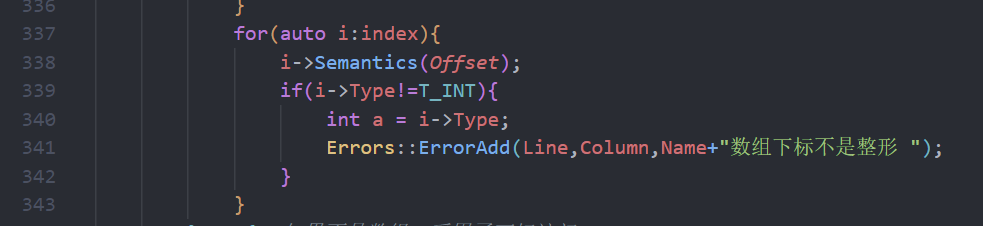


图5-1 数组下标的语义分析

在这里，我经常把第338行的代码写到if语句后面，导致语义分析时程序总是报错。原因是，i还没有语义分析，它的Type还是不确定的，默认为0，导致报错。在实验中有多处类似的代码，我都出现了同样的错误。

**参考文献**

[1] 王生元 等. 编译原理(第三版). 北京：清华大学出版社，20016

[2] 胡伦俊等. 编译原理(第二版). 北京：电子工业出版社，2005

[3] 王元珍等. 80X86汇编语言程序设计. 武汉：华中科技大学出版社,2005

[4] 王雷等. 编译原理课程设计. 北京：机械工业出版社，2005

[5] 曹计昌等. C语言程序设计. 北京：科学出版社，2008