详细设计说明书

# 引言

## 1目的

该文档是本小组为开发mini-C语言解释器而编写的详细设计说明书，为解释器的设计工作做出指导并提供可靠的依据。

## 2文档综述

详细设计说明书包含以下多个部分：引言、体系结构设计、接口设计、数据结构设计、算法设计、技术难点及解决方案和附录。

1. 引言：明确详细设计说明书的书写目的，确定软件范围，提供书写参考资料。
2. 体系结构设计：详细设计中，接口的设计需要体系结构设计的指导，我们将体系结构设计放在详细设计说明书中，方便对照查阅。
3. 接口设计：各个软件模块的接口设计，是详细设计中的重点，明确了接口之间的输入数据流及流向和输出数据流及流向，对接口的开发有指导意义。
4. 数据结构设计：软件开发用于存储数据和实现算法的数据结构设计，明确了各种数据结构的作用、使用的方式，对数据结构的开发有指导作用。
5. 算法设计：软件开发用于实现主要功能时的算法设计，算法指导软件的运行流程，对软件的运行作用有指导作用。
6. 技术难点：项目开发中设计时的技术难点和解决方案。
7. 附录：撰写本文档的其他参考资料。

# 体系结构设计

通过在需求分析中确定的数据流图，我们通过变换分析法分析得体系结构如下图。



（软件体系结构设计图）

## 1.1输入文本

   用户将自己需要解释的程序以文件或命令行形式导入至解释器中

## 1.2词法分析

   从左到右逐个字符的读入源程序，即对构成源程序的字符流进行扫描然后根据词规则识别单词（或单词符号或符号）。

## 1.3 产生保留字集

   确定mini-C语言中的关键字，建立词法集。

   根据现有的词法集，针对读入文本产生保留字。

   将所有的保留字归并到一个集合中。

## 2.1 确定代码块并解析代码块

**构造语法树**：对代码块中的表达式进行构造语法树。

**遍历形成后缀表达式**：遍历语法树，将表达式构变成后缀表达式。

**解析表达式：**计算后缀表达式。

**形成代码块：**将表达式的值构成代码块。

**控制语句解析：**将代码块中的控制语句解析形成语法树。

## 2.2 函数分析

**确定局部变量生命周期：**确定函数中的局部变量，生命周期，方便申请内存。

**确定返回值类型：**确定函数的返回值，同时记录函数的返回值类型 。

**确定形参：**确定函数的输入形参，记录函数的形参个数以及类型。

**确定代码块并解析代码：**将函数内的代码块执行2.1.2.1的操作。

## 3执行中间代码

在分析语法，解析语法的同时一边执行运算，同时生成解释信息。

## 4输出解释信息

根据每部分不同的功能确定出错信息，建立异常库。在嵌套的形式中，每一层向上一层抛出异常信息，最后最外层输出。

其中，错误信息类型如下：

1）解释成功信息：无致命错误，整个程序正常解释成功。

2）输入错误信息输入文件不存在或格式错误。

3）词法错误信息：词法分析错误，关键字拼写错误。

4）内存分配错误信息：内存分配出错，无法分配或尝试给常量分配

5）表达式错误信息：表达式计算出错，表达式格式出错。

6）语法错误信息：关键字不匹配或无语句结束符“；”。

### 4.1判断有无错误

模块接收之前所有模块传递过来的错误信息，如果全部错误信息的种类为“0”，即无错误信息，那么即准备输出成功信息。

### 4.2统计整合解释信息

如果只有成功信息，那么向用户显示整个解释程序成功信息。反之，按照顺序和重要程度显示错误信息。

# 接口设计

## 1.1读入文本

功能说明： 用户将自己需要解释的程序以文件或命令行形式导入至解释器中

输入数据流：Mini-C程序文本。

输入数据来源：用户

输出数据流：词，错误信息。

输出数据走向：词法分析，输出解释信息

接口操作： 输入文本。

限制说明：输入文本为系统可识别信息。

## 1.2词法分析

功能说明：从左到右逐个字符的读入源程序，即对构成源程序的字符流进行扫描然后根据词规则识别单词（或单词符号或符号）。

输入数据流：词，保留字。

输入数据来源：读入文本，产生保留字。

输出数据流：有标签的词，词法分析错误信息。

输出数据走向：识别，输出解释信息

接口操作： 拆分字符集。

## 1.3产生保留字集

功能说明：确定mini-C语言中的关键字，建立词法集。根据现有的词法集，针对读入文本产生保留字。将所有的保留字归并到一个集合中。

输入数据流：无。

输入数据来源：无。

输出数据流：保留字。

输出数据走向：词法分析

接口操作：产生保留字的集合。

## 2.1 确定代码块并解析代码块

功能说明：将词法分析后形成的单词（token）构成代码块，并且解析执行代码块。

输入数据流：词法分析之后的单词，保留字。

输入数据来源：词法分析。

输出数据流：mini-c程序运行结果或者错误说明。

输出数据走向：用户或者错误分析模块

接口操作：确定代码块并解析执行代码块。

### 构造语法树：

功能说明：将词法分析后单词形成的表达式构成语法树

输入数据流：词法分析之后的单词。

输入数据来源：词法分析。

输出数据流：语法树。

输出数据走向：遍历语法树

接口操作：构造语法树。

### 遍历形成后缀表达式：

功能说明：遍历语法树，将表达式构变成后缀表达式。

输入数据流：语法树。

输入数据来源：构造语法树。

输出数据流：后缀表达式。

输出数据走向：解析表达式

接口操作：遍历形成后缀表达式。

### 解析表达式：

功能说明：计算后缀表达式。

输入数据流：后缀表达式。

输入数据来源：遍历形成后缀表达式。

输出数据流： 后缀表达式的值。

输出数据走向：形成代码块。

接口操作：解析表达式。

### 形成代码块：

功能说明：将表达式的值构成代码块。

输入数据流：表达式的值。

输入数据来源：解析表达式。

输出数据流： 代码块。

输出数据走向：控制语句解析。

接口操作：形成代码块。

### 控制语句解析：

功能说明：将代码块中的控制语句解析形成语法树。

输入数据流：代码块

输入数据来源：形成代码块。

输出数据流： 解析执行后的控制语句。

输出数据走向：用户。

接口操作：控制语句解析。

## 2.2 函数分析

功能说明：分析mini-c程序中的函数，并进行解析执行。

输入数据流：函数代码块

输入数据来源：形成代码块。

输出数据流： 解析执行后的mini-c函数。

输出数据走向：用户。

接口操作：函数分析。

### 确定局部变量生命周期：

功能说明：确定函数中的局部变量，生命周期，方便申请内存。

输入数据流：函数代码块

输入数据来源：形成代码块。

输出数据流： 无。

输出数据走向：无。

接口操作：确定局部变量生命周期。

### 确定返回值类型：

功能说明：确定函数的返回值，同时记录函数的返回值类型 。

输入数据流：函数代码块

输入数据来源：形成代码块。

输出数据流： 无。

输出数据走向：无。

接口操作：确定函数的返回值。

### 确定形参：

功能说明：确定函数的输入形参，记录函数的形参个数以及类型。

输入数据流：函数代码块

输入数据来源：形成代码块。

输出数据流： 无。

输出数据走向：无。

接口操作：确定形参。

### 确定代码块并解析代码：

功能说明：将词法分析后形成的单词（token）构成代码块，并且解析执行代码块。

输入数据流：词法分析之后的单词，保留字。

输入数据来源：词法分析。

输出数据流：mini-c函数运行结果或者错误说明。

输出数据走向：用户或者错误分析模块

接口操作：确定代码块并解析执行代码块。

## 3对中间代码操作

功能说明：对已经构造的语法树进行遍历，对中间代码按照统一过程标准进行执行，输出最终执行结果

输入数据流：语法树，保留字。

输入数据来源：语法识别模块

输出数据流：程序解释结果，内存分配错误信息。

输出数据走向：用户，输出解释信息

接口操作： 解释语法树。

## 4.1判断有无错误

功能说明：模块接收之前所有模块传递过来的错误信息，如果全部错误信息的种类为“0”，即无错误信息，那么即准备输出成功信息。

输入数据流：各种错误信息

输入数据来源：读入、识别、解释模块

输出数据流：出错信息或成功信息。

输出数据走向：统计整合解释信息模块

接口操作： 判断有无错误信息。

## 4.2统计整合解释信息

功能说明：如果只有成功信息，那么向用户显示整个解释程序成功信息。反之，按照顺序和重要程度显示错误信息。

输入数据流：出错信息或成功信息。

输入数据来源：判断有无错误模块

输出数据流：全部的错误信息或成功信息。

输出数据走向：用户

接口操作： 统计整合解释信息。

# 数据结构设计

## 链表

基本的数据结构，是一种线性表，每一个节点里存着下一个节点的指针

*typedef struct \_Node*

*{*

*ElemType data;*

*struct \_Node \* next;*

*}NODE,\*PNODE;*

## 内存池

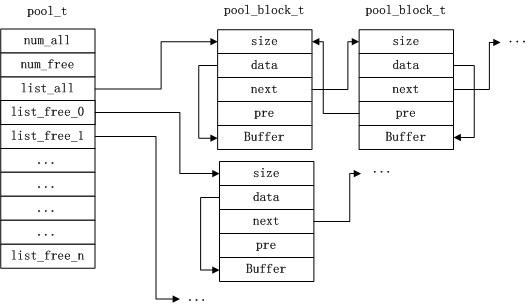
主要思想：

1 记录所有申请的内存

2 当释放内存时，记录下来以供下次申请使用

3 申请内存时，可以直接使用前面释放过的内存

需要为申请内存的大小划分，可以划分为 {16,32,64,128,...}，为每一个大小的粒度建立一个双向链表。



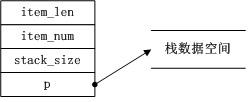
*typedef struct \_pool\_block{*  
*int size;*  
*void \* data;*  
*struct \_pool\_block \* next;*  
*struct \_pool\_block \* pre;*  
*}pool\_block\_t;*

*typedef struct \_pool{*  
*int num\_all;*  
*int num\_free;*  
*pool\_block\_t \* list\_all;*  
*pool\_block\_t \* list\_free[POOL\_ATOM\_NUM];*  
*}pool\_t;*

*int pool\_atom\_tab[POOL\_ATOM\_NUM] = {*  
*32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, -1*  
*};*

## 栈

申请一个空间，按照一个一个节点的顺序保存进去，按照保存顺序一个一个的取出来



*typedef struct \_stack{  
    int item\_len;  
    int item\_num;  
    int stack\_size;  
    char \*p;  
}stack\_t;*

## Hash表

虽然Hash表很浪费内存，但是其十分快速的查找能力使其成为必要的一个数据结构，但是用的比较少

# 算法设计

## 1读入模块：

1.1以文件形式将用户输入整体读入，解释器需先行建立自身固有的保留字集，能够做到对于任意一个保留字在O(1)时间内查询出该字符保留字的特性。整体读入用户输入后，对用户输入进行词法分析，将程序语句拆分成无数含义确定的词，使用递归方法进行拆分。

1.2读入模块将文本读入并分词处理后，将处理结果与文本统一提供给下一模块。

1.3利用标志记录是否存在错误，如果存在错误则反馈给最终收集错误的模块，向后续模块提示错误并终止程序。

## 2识别模块：

2.1利用读入模块的分词结果，递归构建语法树。

2.2利用标志记录是否存在错误，如果存在错误则反馈给最终收集错误的模块，向后续模块提示错误并终止程序。

## 3执行模块：

3.1按照后序遍历语法树，按照运算、判断等规则中间代码进行解释。

3.2利用标志记录是否存在错误，如果存在错误则反馈给最终收集错误的模块，向后续模块提示错误并终止程序，特别注意如果发生错误，不输出结果。

## 4解释信息：

按照各模块反馈的错误信息进行输出。

# 技术难点及解决方案

## 1.根据DFA构造词法分析程序

根据识别SIM程序段的DFA手工编写一个识别SIM语言单词的程序，运行该程序即可识别SIM语言程序段的所有单词。如下：

输入一个SIM语言的程序段；对SIM语言程序段进行扫描，根据程序段的字符进行单词的拼写和识别，直到识别出一个单词，给出该单词的二元式；依次识别SIM程序段的每个单词，直到程序段扫描完毕；程序结束。

## 2.建立语法树

语法树是源代码的抽象语法结构的抽象语法结构的树状表现形式，特质编程语言的源代码。要建立一个抽象语法树，必须以抽象语法树的文法为前提，文法是决定怎样将语言的元素组合起来的规则的集合，给定一个编程语言之后，我们可以根据语言的特点定义出抽象语法树的文法，基于语言的抽象语法树的文法列出语法树

抽象语法树的每一个节点是一个类的对象，代表一个类型的语句，基于语言的抽象语法树文法包含了86条文法规则，规则的左部出现的符合成为非终结符号，对应为抽象语法树的符合结点；规则中不属于非终结符号集合的符号为终结符号，对应的是树叶结点，因此在我们设计的解释器中，结点类包含了这86条规则对应的基本类。

## 3.抽象语法树的遍历

在遍历过程中，我们可以采用访问者模式。将每一个类中相关的操作封装在一个独立的对象中，并在遍历抽象语法树时，将此对象传递给当前访问的节点，当一个结点接收该访问者时，该结点向访问者发送一个包含自身类信息的请求，该请求向访问者发送一个包含自身类信息的请求，该请求同时也将该结点本身作为一个参数，然后访问者将该节点执行某一类型的操作。

# 附录

#### 参考资料：

[1] mini-C语言语法说明

[2]CSDN博客 <http://www.cnblogs.com/linxr/>

[3]两周自制脚本语言，千叶滋[日]