科研开源软件创意大赛

算法精灵软件项目分析设计书

西安德新软件

2019年7月16日

目录

[一．引言 3](#_Toc14156987)

[1.1背景 3](#_Toc14156988)

[1.2定义 4](#_Toc14156989)

[1.3参考资料 4](#_Toc14156990)

[二．系统设计 4](#_Toc14156991)

[2.1同类软件国内外现状及对比分析 4](#_Toc14156992)

[2.2系统构思 4](#_Toc14156993)

[2.3关键技术及创新点 5](#_Toc14156994)

[2.4关键数据结构 5](#_Toc14156995)

[2.5系统结构设计 9](#_Toc14156996)

[三．系统功能描述 10](#_Toc14156997)

[3.1基本处理流程 10](#_Toc14156998)

[3.1.1系统流程图 10](#_Toc14156999)

[3.1.2 数据流程图 11](#_Toc14157000)

[3.2尚未解决的问题 11](#_Toc14157001)

[3.3接口设计 12](#_Toc14157002)

[3.3.1外部接口 12](#_Toc14157003)

[3.3.2内部接口 12](#_Toc14157004)

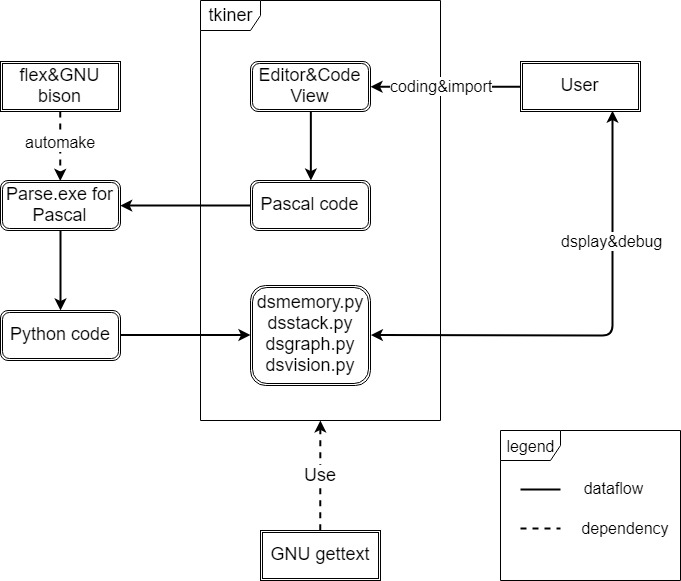
[四．相关依赖软件的说明 12](#_Toc14157005)

# 一．引言

## 1.1背景

算法精灵是一个数据结构辅助教学的工具，同时又是一个帮助学习和理解算法的工具。面向广大学生和教师。本项目的设计者和开发者是西安德新软件。

算法精灵基于Python2.7构建，支持用户使用Pascal语言编写算法进行执行和演示，我们基于flex和GNU bison设计了Pascal-python的解析器将Pascal程序转换为Python源代码并执行演示，如下图。



算法精灵可以帮助你制作算法课件，也可以帮助你设计和调试算法。

算法精灵能够把算法的执行过程能够使用动画的方式把内存数据、堆栈数据等的变化直观的显示出来，可以帮助你理解算法执行过程和算法的实现原理。而这些动画的过程可以被提取出来作为设计课件的素材。

算法精灵还允许你输入自己设计的算法，然后运行该算法，通过图形化的方式来展示算法执行过程，从而发现算法设计中的问题和更好的改进算法，还可以在算法中增加代码断点和数据断点。

算法精灵内置了大部分的经典算法。每一个算法都可以作为一个游戏来启动。在游戏过程中，你需要正确的按照算法目前的状态正确的设定图片的属性，或者填写正确的数据值，才可以继续游戏或者得到积分（游戏功能在当前版本尚未实现）。

算法精灵使用pyinstaller进行打包，可运行在window和Linux等平台上。

## 1.2定义

Flex: 基于C和C++的快速词法分析器的生成器。

GNU bison: 属于GNU项目的一个语法分析器生成器。

Pascal: 由瑞士Niklaus Wirth教授于六十年代末设计并创立的一门结构化编程语言，常常被用作学习数据结构与算法的教学语言。

GNU gettext: 一种多国语言解决方案

## 1.3参考资料

Flex: <https://github.com/westes/flex>

GNU bison: <https://www.gnu.org/software/bison/manual/>

Python: <https://docs.python.org/2/>

Tcl/Tk: <https://core.tcl-lang.org/index.html>

# 二．系统设计

## 2.1同类软件国内外现状及对比分析

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | 开发者 | 动态演示 | 堆栈展示 | 变量展示 | 具有editor | 步进调试 | 控制台 |
| [Data Structure Visualizations](https://www.cs.usfca.edu/~galles/visualization/Algorithms.html) | 美国旧金山大学 | ***√*** |  | ***√*** |  | ***√*** |  |
| VisuAlgo | Steven Halim | ***√*** |  | ***√*** |  | ***√*** |  |
| 算法精灵 |  | ***√*** | ***√*** | ***√*** | ***√*** | ***√*** | ***√*** |

## 2.2系统构思

算法的初学者往往是编程能力欠缺，对计算思维的理解不深刻，难以把握程序执行的过程，对算法执行过程的动态展示是自解释的，初学者能够非常清楚的看到变量值和堆栈的变化过程，降低了实现的难度，让更加专注与对算法本身的理解。

同时我们实现了一个简化版的IDE，包括代码高亮、代码补全、设置断点、步进调试、自动缩进，能够让初学者自由的实现自己的算法，具有很强的代入感，初学者能够从传统IDE到算法精灵毫无难度的过渡过来。

内置对各种经典算法，用户可以直接执行演示。同时可以保存源文件便于回放提升用户体验。

## 2.3关键技术及创新点

本系统的关键技术是对用户编写的Pascal语言等源程序进行词法分析和语法分析，翻译成python代码、内存堆栈的图形化、Pascal语言标准库的实现等

本系统的创新点是能使用户可以自由的编写算法并调试、提供比传统IDE更加友好界面，降低了初学者学习算法的难度。

## 2.4关键数据结构

# DATAPOOL：

数据池，用于可视化编辑数据池数据，或者将数据池数据转换成为一个数据字典。

数据池实体数据定义：

数据池是一个字典，经过 repr 转换成为字符串；

每一个数据项格式：'name' : { 数据属性 }

数据属性定义：

type,

value,

根据类型的不同，值的格式也不一样：

Integer: int

Boolean: bool

Real: float

String: str

Char: str

Pointer: list, 包括一个字符串元素，字符串可以为空

Array: {} 其中键值为整数

Record: {} 其中键值为字符串

Tree: {} 固定键值，lchild, rchild, value,

在数据池存储的 value 要经过 repr 转换成为字符串；

如果数据池中 value 为空字符串，表示对应的变量为 None；

Array 和 Record 字典的值有两种情况：

如果是字符串，则是 int, bool, float, str 的 repr 结果；

如果是字典，则是一个数据项定义；

Pointer 对应的值必定是包含一个字符串的列表，表示其指向的变量名称；

Tree 的 lchild, rchild 可以为空，或者字典，指向子树数据项定义；

value 的含义和 Array 和 Record 中的字典值相同；

# DRIVER

算法驱动实例，由算法演示程序调用，提供操作视图和实体的功能。

用于演示类的功能

pause

step

abort

resume

内部函数

\_\_check

断点实体：

[ 是否可用，函数名称，行号，表达式，命中次数，表达式的当前值 ]

是否可用为 0 或者 1, 标志当前断点是否可用；

函数名称，可以为空；

行号，当前行号；可以为 0， 表示匹配任何行号；

条件表达式，合法的 Python 表达式，数据环境为：

断点所在的执行框架中的局部变量，全局变量，以及特殊变量：

hits, 表示当前断点的命中次数，例如 hits == 1

可以使得当前断点命中一次之后永不在停下来

var 是一个字典，列出当前读写的变量,

var = {'r' : (变量名称列表,...),

'w' : (变量名称列表,...),

}

例如表达式 'x' in var['r']，表示如果读变量 x

的时候断点成立

命中次数，到目前为止断点命中的次数；

表达式的当前值，适用于监控表达式，可以为 None；

驱动实体属性说明

options, 全部的选项字典

player, 演示实体，指向各个视图；

data\_item, 数据池数据字典

algorithm\_name, 算法名称

m\_algorithm，算法类实例

m\_datapool, 算法中使用的数据池数据，已经转换成为 aftype 的类型

co\_name, 算法当前执行的代码块名称

co\_lineno, 算法执行的当前行

驱动实体行为说明：

创建算法实体，

在可见视图上创建全局堆框架，用来存放全局变量和使用 new 产生的变量

根据数据池数据创建算法使用的数据实体

算法入口模拟，

在可见视图上创建顶层算法框架

初始化算法数据实体

初始化堆栈列表

初始化当前执行的代码块名称和代码行

过程调用模拟，

在可见视图上增加一个过程框架

当前代码行和代码块名称推入堆栈，

设置当前执行的代码块名称和代码行

初始化变参、值参和局部变量

过程返回模拟，

在可见视图上删除当前过程框架

从调用堆栈中弹出调用者的代码行和名称

将函数返回值传递给调用者

语句执行模拟，

设定当前执行的代码行

变量声明模拟，

在可见视图的当前堆栈中添加一个变量，变量是一个 aftype 的类型

变量删除模拟，

在可见视图上删除一个变量，变量是一个 aftype 的类型

变量修改模拟，

简单变量修改，传入参数：var，value

前者是一个 aftype，后者是一个具体的值，如果为

None 则不显示简单变量的内容，但是所占区域不变；

结构变量修改，传入参数：var，size

结构体内部元素的修改等价于简单变量的修改，这里

的修改主要是指数组的大小发生变化等；

指针变量修改，传入参数：var，value

前者是一个 aftype.Pointer，后者是一个 aftype 的

变量，如果为空指针，则显示在过程框架的第一行；

树型变量修改，传入参数：var，index，value

树变量结点内容的修改等价于简单变量的修改，这里

的修改指的是对孩子的修改；

var 是一个 aftype.Tree 的变量，index 是孩子的位

置，从 0 开始； value 也是一个 aftype.Tree 的变

量，可以为 None，说明孩子为空。

变量显示模拟，

用来显示隐藏的变量；

变量隐藏模拟，

隐藏一个变量，可以是过程框架隐藏变量，也可以是结构、

树隐藏其子结点；也可以是指针变量隐藏；

变量高亮模拟，

高亮显示一个变量，取消其他变量的高亮显示；

变量可以是简单、结构、树或者指针，甚至是过程框架；

# VOPTIONS

可见数据选项，控制各种类型的结点和各种数据类型的显示选项

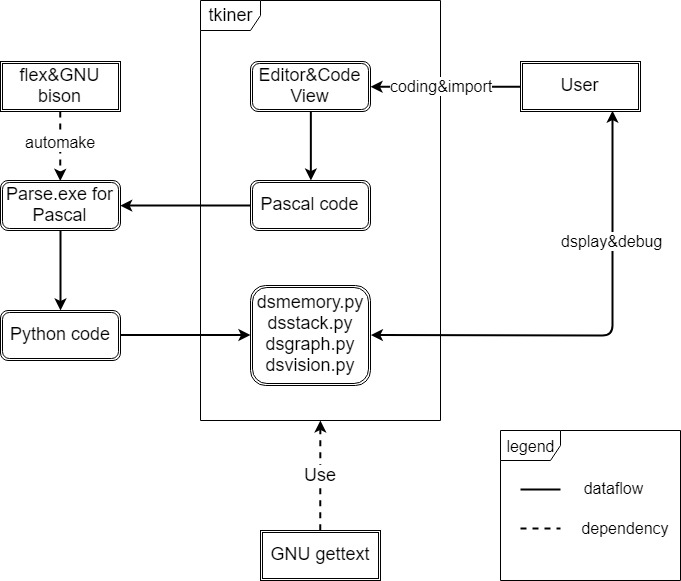
Single: 单步执行选项

Struct： 结构数据显示选项

Pointer： 指针数据结构的可视化选项

Tree： 内置树结构对象的可视化选项

## 2.5系统结构设计

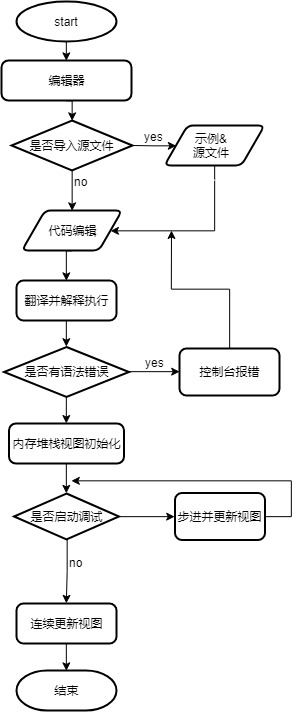


本系统基于python2.7构建，使用tkiner框架开发GUI，实现了具有代码高亮和代码补全功能的editor，使用GNU bison和flex做词法分析和语法分析实现Pascal到python代码的翻译，使用GNU gettext实现多国语言的国际化。

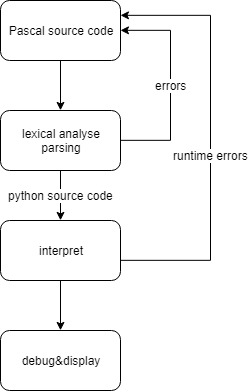
# 三．系统功能描述

## 3.1基本处理流程

### 3.1.1系统流程图



### 3.1.2 数据流程图



## 3.2尚未解决的问题

1. Editor的静态语法检查
2. 树，图的输入参数可以使用图形化。
3. 算法执行过程可以任意回退。
4. 动画过程可以保存为文件。
5. 可以定制可视视图的形状和风格。

## 3.3接口设计

### 3.3.1外部接口

*（无）*

### 3.3.2内部接口

push(name) 增加执行框架，并设置为当前执行框架

pop() 弹出当前执行框架，并设置调用者为当前框架

add\_variable(var) 在当前框架内增加一个变量

remove\_variable(var) 在当前框架内删除一个变量

update\_variable(var, value) 修改 var 的显示, value 可

能为 None，表示隐藏对应的变量

select\_variable(var) 选中一个变量，设置输入焦点

show\_variable(var) 显示隐藏的变量

hide\_variable(var) 隐藏一个变量

active\_variable(var) 激活一个变量

deactive\_variable(var) 取消激活变量

refresh() 刷新数据视图

# 四．相关依赖软件的说明

1. Flex: [https://github.com/westes/flex](https://github.com/westes/flexB) BSD许可协议

2. GNU bison: <https://www.gnu.org/software/bison/manual/> GPL许可协议