Final

Contents

Chapter 1

继承关系索引

1.1 类继承关系

此继承关系列表按字典顺序粗略的排序:

Chapter 2

类索引

2.1 类列表

这里列出了所有类、结构、联合以及接口定义等，并附带简要说明:

Chapter 3

文件索引

3.1 文件列表

这里列出了所有文件，并附带简要说明:

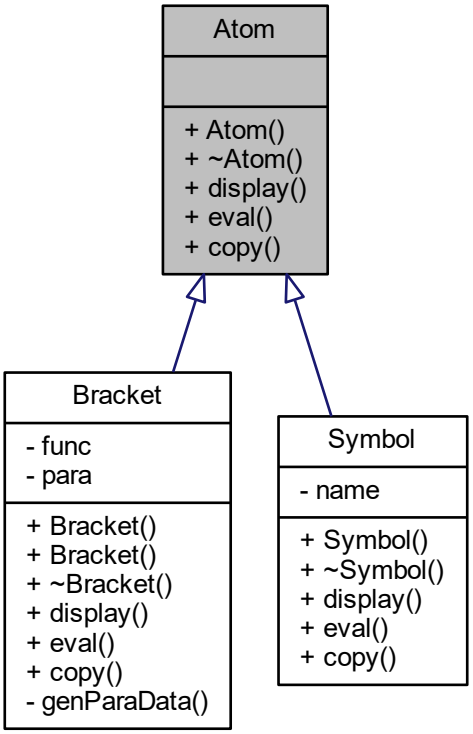
Chapter 4

类说明

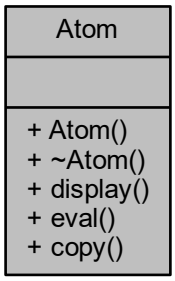
4.1 Atom 类参考

`#include <Atom.h>`

类 Atom 继承关系图:



Atom 的协作图:



Public 成员函数

- Atom ()
- virtual ~Atom ()
- virtual void display (int indent)=0
- virtual Data * eval (Runtime *runtime)=0
- virtual Atom * copy ()=0

4.1.1 详细描述

用于存储语法结构的抽象类

4.1.2 构造及析构函数说明

4.1.2.1 Atom()

```
Atom::Atom ( )  [inline]
```

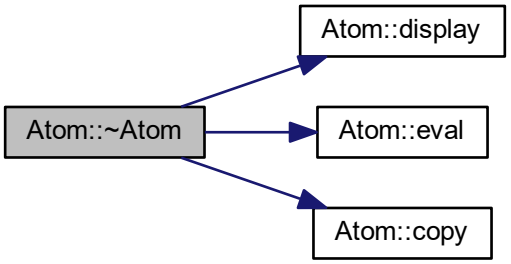
空构造函数

返回

4.1.2.2 ~Atom()

```
virtual Atom::~Atom ( )  [inline], [virtual]
```

虚析构函数函数调用图:



4.1.3 成员函数说明

4.1.3.1 copy()

```
virtual Atom* Atom::copy ( )  [pure virtual]
```

复制一个Atom 并返回

返回

在 Symbol , 以及 Bracket 内被实现.

这是这个函数的调用关系图:

4.1.3.2 display()

```
virtual void Atom::display (
            int indent )  [pure virtual]
```

显示语法树，并且使用 indent 级的缩进

参数

indent	

在 Symbol , 以及 Bracket 内被实现.

这是这个函数的调用关系图:

4.1.3.3 eval()

```
virtual Data* Atom::eval (
            Runtime * runtime )  [pure virtual]
```

根据 runtime 来求Atom 对应的值

参数

runtime	用于获取变量对应的数据、进行赋值运算等

返回

当前Atom 的值，比如如果当前Atom 是 (add 1 2) 则返回一个值为 3 的数据

在 Symbol , 以及 Bracket 内被实现.

这是这个函数的调用关系图:

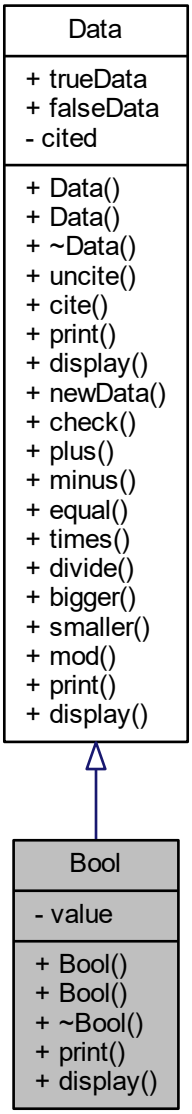
该类的文档由以下文件生成:

- Atom.h

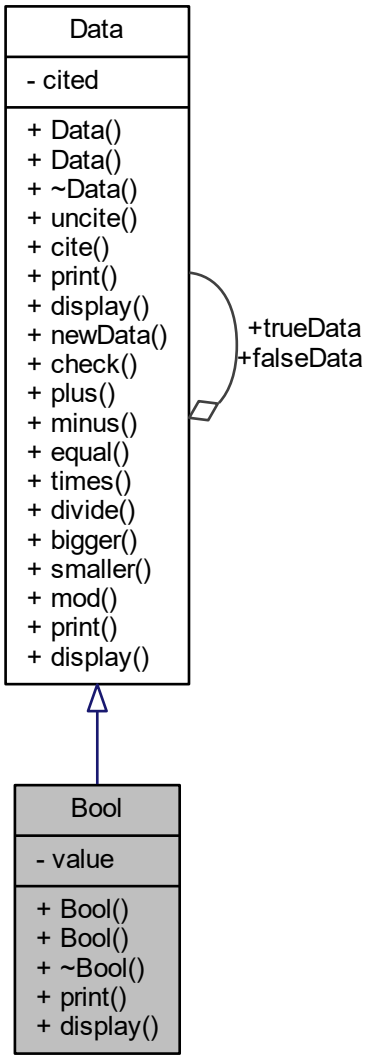
4.2 Bool 类参考

```
#include <Data.h>
```

类 Bool 继承关系图:



Bool 的协作图:



Public 成员函数

- Bool (bool value)
- Bool (bool value, int)
- virtual ~Bool ()
- virtual void print ()
- virtual void display ()

Private 属性

- bool value

友元

- class Data
- class Bracket
- class Runtime

额外继承的成员函数

4.2.1 详细描述

表示一个Bool 数据

4.2.2 构造及析构函数说明

4.2.2.1 Bool() [1/2]

```
Bool::Bool (
            bool value )
```

根据 value 为 true 或 false 来初始化

参数

value	用于初始化 value 属性

返回

4.2.2.2 Bool() [2/2]

```
Bool::Bool (
            bool value,
            int  )
```

用于初始化全局使用的 true 和 false

参数

value	

返回

4.2.2.3 ~Bool()

```
Bool::~Bool ( )  [virtual]
```

析构函数

4.2.3 成员函数说明

4.2.3.1 display()

```
void Bool::display ( )  [virtual]
```

打印当前对象的详细信息

实现了 Data.

函数调用图:

4.2.3.2 print()

```
void Bool::print ( )  [virtual]
```

打印当前对象的值

实现了 Data.

这是这个函数的调用关系图:

4.2.4 友元及相关函数文档

4.2.4.1 Bracket

```
friend class Bracket  [friend]
```

4.2.4.2 Data

```
friend class Data  [friend]
```

4.2.4.3 Runtime

```
friend class Runtime  [friend]
```

4.2.5 类成员变量说明

4.2.5.1 value

```
bool Bool::value  [private]
```

存储该对象的值

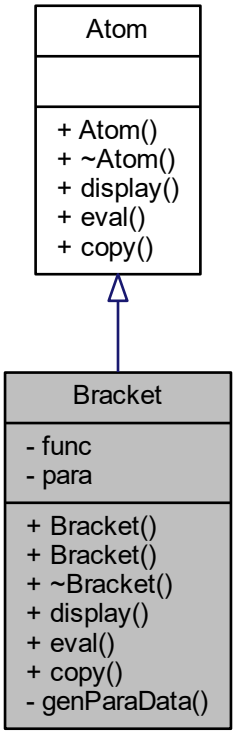
该类的文档由以下文件生成:

- Data.h
- Bool.cpp

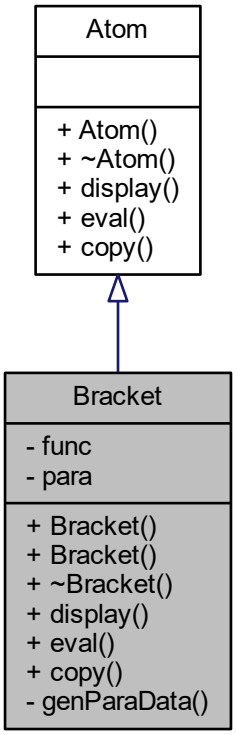
4.3 Bracket 类参考

```
#include <Atom.h>
```

类 Bracket 继承关系图:



Bracket 的协作图:



Public 成员函数

- Bracket (const std::string &str, unsigned long &pos)
- Bracket ()
- virtual ~Bracket ()
- virtual void display (int indent)
- virtual Data * eval (Runtime *runtime)
- virtual Atom * copy ()

Private 成员函数

- std::vector< Data * > * genParaData (Runtime *runtime)

Private 属性

- std::string func
- std::list< Atom * > para

友元

- class Runtime
- class Function

4.3.1 详细描述

用于存储一个括号的语法结构的类

4.3.2 构造及析构函数说明

4.3.2.1 Bracket() [1/2]

```
Bracket::Bracket (
            const std::string & str,
            unsigned long & pos )
```

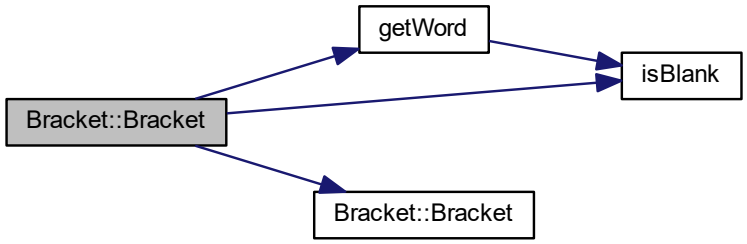
从 pos 位置开始，从 str 字符串中取出一个完整的括号表达式，并生成该表达式对应的语法树存储在新建的Bracket 对象里

参数

str	源代码
pos	用于作为“光标”所在位置，函数会修改 pos 到获取的完整的括号表达式最后一个字符的位置加一

返回

函数调用图:



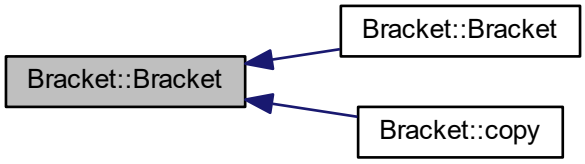
4.3.2.2 Bracket() [2/2]

```
Bracket::Bracket ( )
```

默认的空构造函数

返回

这是这个函数的调用关系图:



4.3.2.3 ~Bracket()

```
Bracket::~Bracket ( )  [virtual]
```

析构函数

4.3.3 成员函数说明

4.3.3.1 copy()

```
Atom * Bracket::copy ( )  [virtual]
```

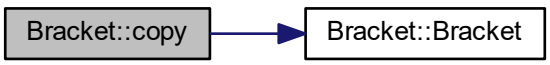
复制一份当前对象并返回

返回

复制得到的对象

实现了 Atom.

函数调用图:



4.3.3.2 display()

```
void Bracket::display (
            int indent )  [virtual]
```

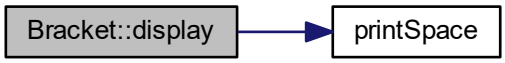
打印Bracket 对象对应的语句，并使用 indent 级的缩进

参数

indent	打印时使用的缩进深度

实现了 Atom.

函数调用图:



4.3.3.3 eval()

```
Data * Bracket::eval (
            Runtime * runtime )  [virtual]
```

根据 runtime 来求本对象对应的表达式的值

参数

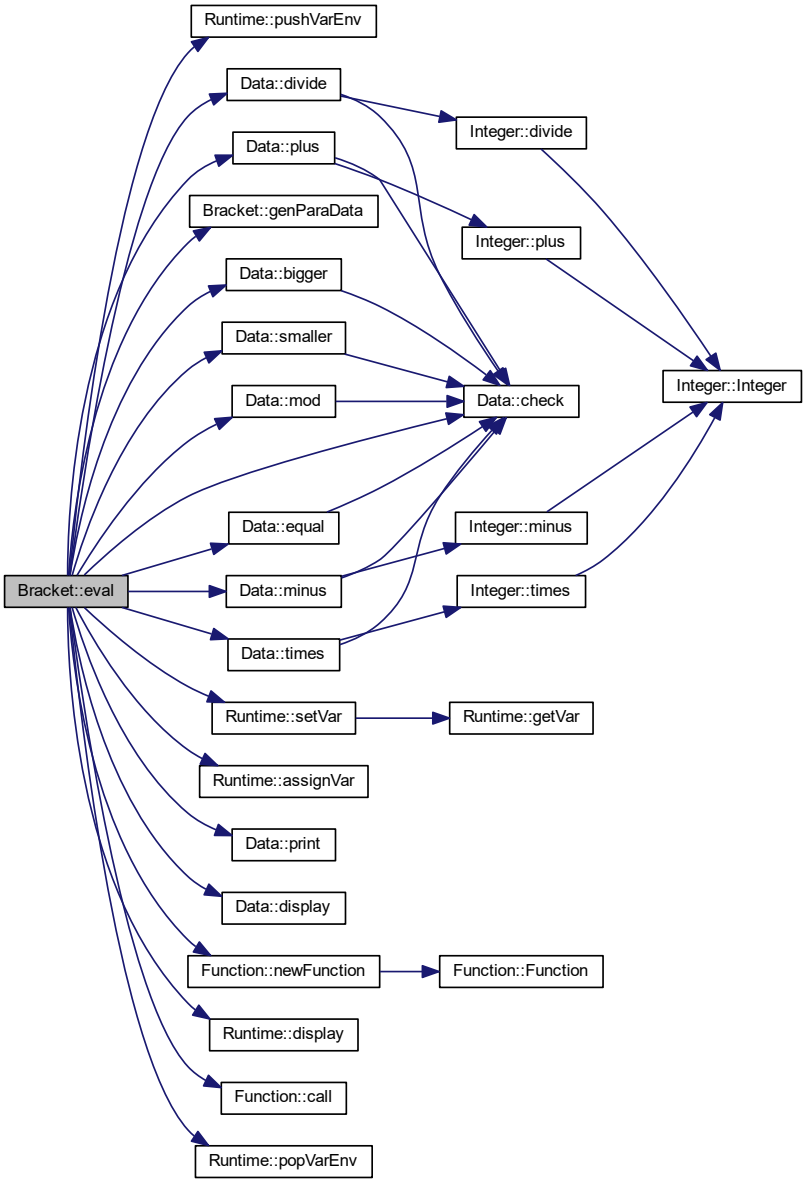
runtime	用于获取求值时需要的变量的值以及进行赋值操作等

返回

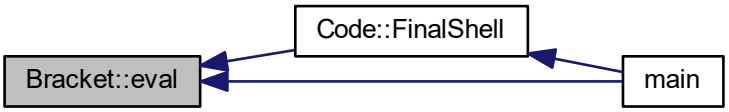
本对象对应表达式的值

实现了 Atom.

函数调用图:



这是这个函数的调用关系图:



4.3.3.4 genParaData()

```
std::vector< Data * > * Bracket::genParaData (
            Runtime * runtime )  [private]
```

根据 para 属性生成参数的具体数据

参数

runtime	求参数的数据时，从 runtime 中获取变量名对应的数据以及进行赋值操作等

返回

返回一个根据参数生成的数据的 vector

这是这个函数的调用关系图:

4.3.4 友元及相关函数文档

4.3.4.1 Function

```
friend class Function  [friend]
```

4.3.4.2 Runtime

```
friend class Runtime  [friend]
```

4.3.5 类成员变量说明

4.3.5.1 func

```
std::string Bracket::func  [private]
```

用于存储该表达式对应的函数名称，比如 (test 1 2) 对应的表达式的函数名称就是 test

4.3.5.2 para

```
std::list<Atom *> Bracket::para  [private]
```

用于存储参数对应的语法结构

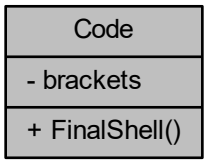
该类的文档由以下文件生成:

- Atom.h
- Atom.cpp

4.4 Code 类参考

```
#include <Code.h>
```

Code 的协作图:



静态 Public 成员函数

- static void FinalShell ()

Private 属性

- std::list< std::string * > brackets

4.4.1 详细描述

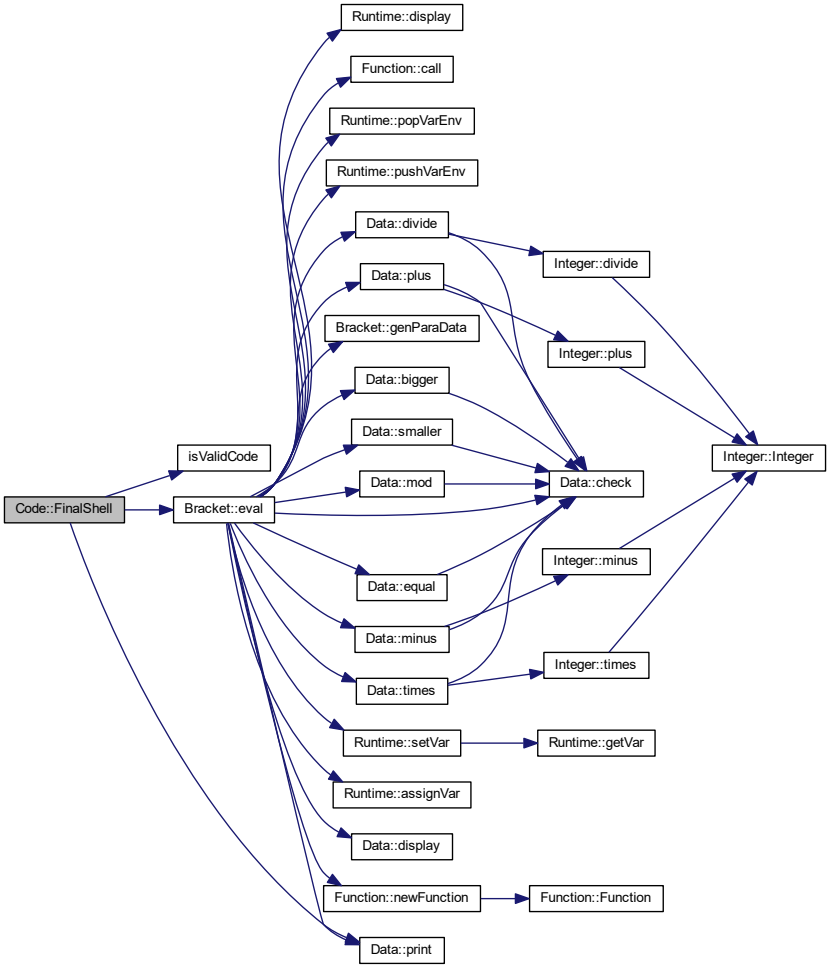
用于提供一个交互运行代码的Shell

4.4.2 成员函数说明

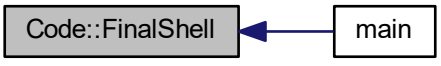
4.4.2.1 FinalShell()

```
void Code::FinalShell ( )  [static]
```

用于提供一个交互运行代码的Shell 函数调用图:



这是这个函数的调用关系图:



4.4.3 类成员变量说明

4.4.3.1 brackets

```
std::list<std::string *> Code::brackets  [private]
```

用于存储用户输入的代码

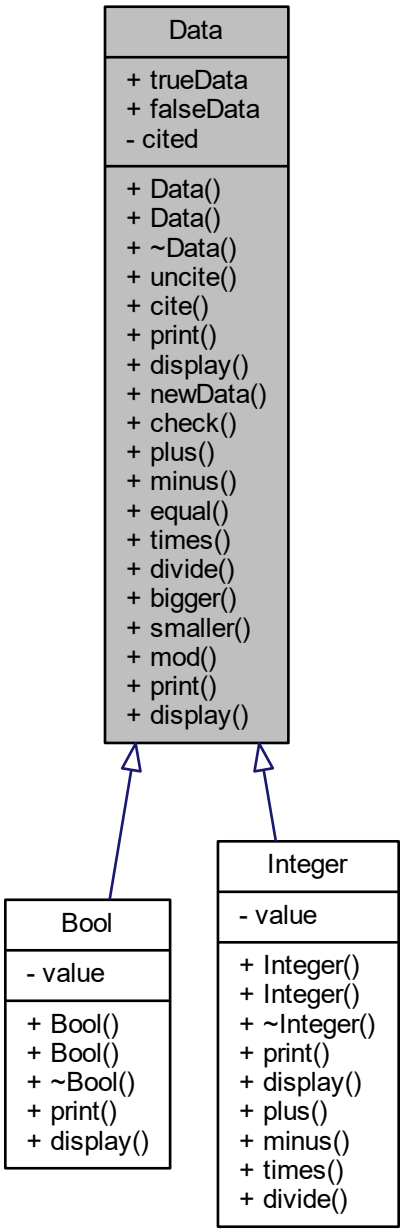
该类的文档由以下文件生成:

- Code.h
- Code.cpp

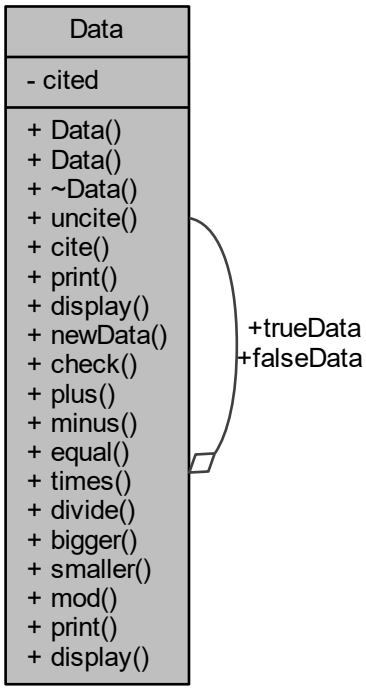
4.5 Data 类参考

```
#include <Data.h>
```

类 Data 继承关系图:


Data
+ trueData
+ falseData
- cited
+ Data()
+ Data()
+ ~Data()
+ uncite()
+ cite()
+ print()
+ display()
+ newData()
+ check()
+ plus()
+ minus()
+ equal()
+ times()
+ divide()
+ bigger()
+ smaller()
+ mod()
+ print()
+ display()
Bool
- value
+ Bool()
+ Bool()
+ ~Bool()
+ print()
+ display()
Integer
- value
+ Integer()
+ Integer()
+ ~Integer()
+ print()
+ display()
+ plus()
+ minus()
+ times()
+ divide()

Data 的协作图:



Public 成员函数

- Data ()
- Data (int)
- virtual ~Data ()
- void uncite ()
- void cite ()
- virtual void print ()=0
- virtual void display ()=0

静态 Public 成员函数

- static Data * newData (const std::string &str)
- static void check (Data *data)
- static Data * plus (std::vector< Data *> *dataPara)
- static Data * minus (std::vector< Data *> *dataPara)
- static Data * equal (std::vector< Data *> *dataPara)
- static Data * times (std::vector< Data *> *dataPara)
- static Data * divide (std::vector< Data *> *dataPara)
- static Data * bigger (std::vector< Data *> *dataPara)
- static Data * smaller (std::vector< Data *> *dataPara)
- static Data * mod (std::vector< Data *> *dataPara)
- static Data * print (std::vector< Data *> *dataPara)
- static Data * display (std::vector< Data *> *dataPara)

静态 Public 属性

- static Data * trueData = new Bool(true, 1)
- static Data * falseData = new Bool(false, 1)

Private 属性

- unsigned int cited

友元

- class Integer
- class Bool
- class VarEnv

4.5.1 详细描述

表示数据的抽象类，数据的类型与值由派生类存储与管理，Data 类只负责对数据进行引用计数

4.5.2 构造及析构函数说明

4.5.2.1 Data() [1/2]

```
Data::Data ( )
```

默认构造函数，将引用次数设为 0

返回

4.5.2.2 Data() [2/2]

```
Data::Data (
            int  )
```

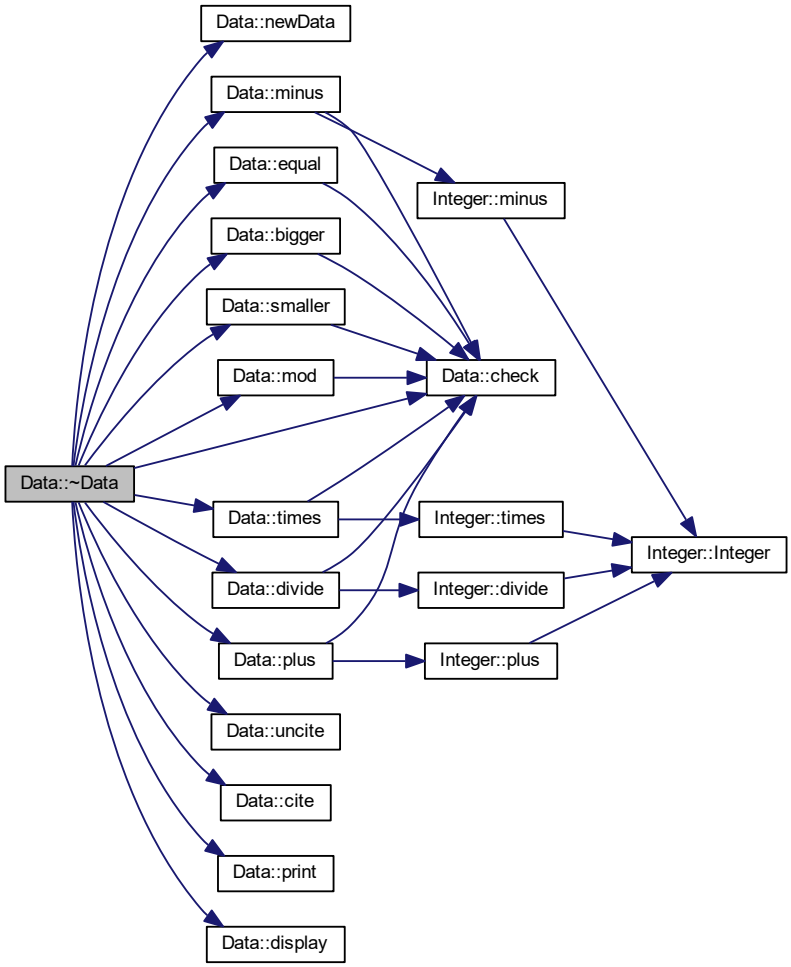
用于生成公用的 true false 数据对象，为了避免这两个对象被析构，在函数体中会将引用次数初始化为 1

返回

4.5.2.3 ~Data()

```
virtual Data::~Data ( )  [inline], [virtual]
```

虚析构函数函数调用图:



4.5.3 成员函数说明

4.5.3.1 bigger()

```
Data * Data::bigger (
            std::vector< Data *> * dataPara )  [static]
```

比较两个数的大小

参数

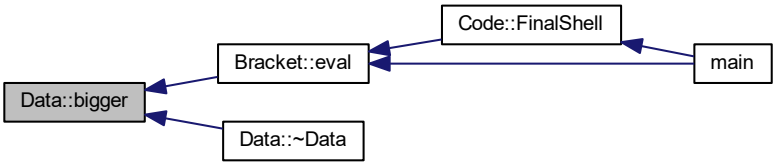
dataPara	被判断的是 dataPara 的前两个数据，更多的数据将会被忽略，被比较的数据只可以是Integer 类

返回

一个Bool 类型的数据，如果 dataPara[0] > dataPara[1]则值为 true，否则为 false

函数调用图:

这是这个函数的调用关系图:



4.5.3.2 check()

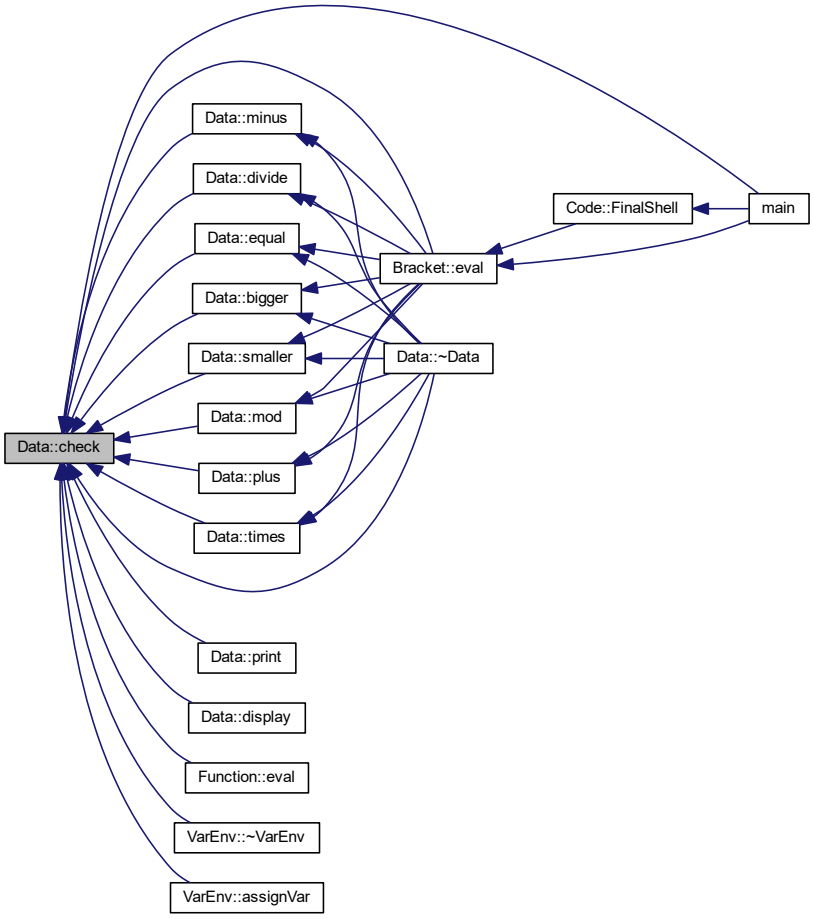
```
void Data::check (
            Data * data )  [static]
```

检查 data 对象的引用次数是否为 0，如果为 0，则析构这个对象

参数

data	指向被检查的对象的指针

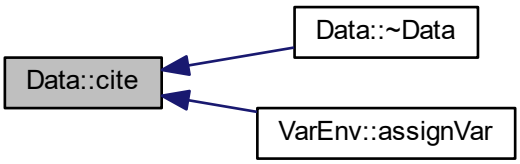
这是这个函数的调用关系图:



4.5.3.3 cite()

```
void Data::cite ( )
```

将对象的引用次数增加 1 这是这个函数的调用关系图:



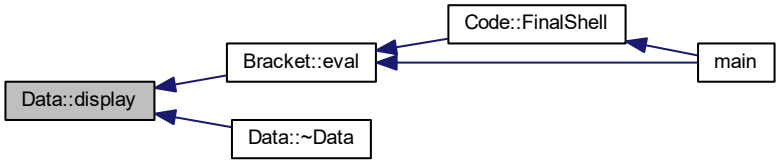
4.5.3.4 display() [1/2]

```
virtual void Data::display ( )  [pure virtual]
```

打印对象的详细信息，包括被引用次数等

在 Bool , 以及 Integer 内被实现.

这是这个函数的调用关系图:



4.5.3.5 display() [2/2]

```
Data * Data::display (
            std::vector< Data *> * dataPara )  [static]
```

逐个打印一组数据的详细信息

参数

dataPara	被打印的数据

返回

一个值为 true 的Bool 类型数据

函数调用图:

4.5.3.6 divide()

```
Data * Data::divide (
            std::vector< Data *> * dataPara )  [static]
```

两个整数的除法

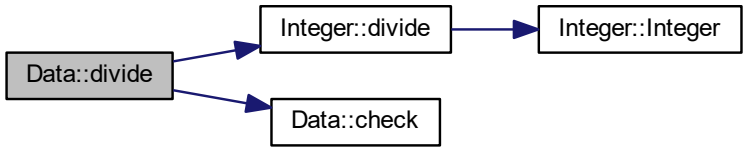
参数

dataPara	dataPara[0]是被除数，dataPara[1]是除数（不能为 0），更多的参数会被忽略

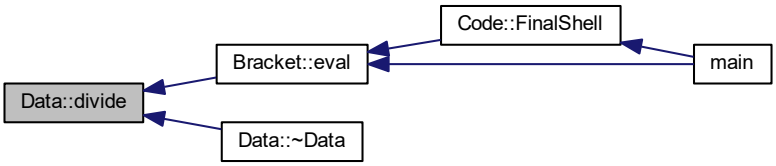
返回

除法的结果

函数调用图:



这是这个函数的调用关系图:



4.5.3.7 equal()

```
Data * Data::equal (
            std::vector< Data *> * dataPara )  [static]
```

判断两个数据是否相等

参数

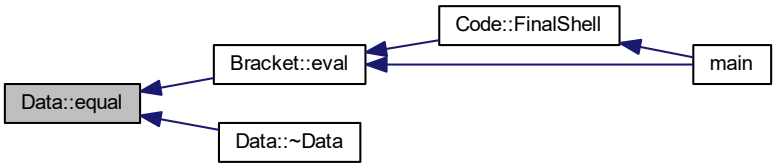
dataPara	被判断的是 dataPara 的前两个数据，更多的数据将会被忽略，被比较的数据可以是Bool 类的对象，也可以是Integer 类的对象

返回

一个Bool 类型的数据

函数调用图:

这是这个函数的调用关系图:



4.5.3.8 minus()

```
Data * Data::minus (
            std::vector< Data *> * dataPara )  [static]
```

两个整数的减法

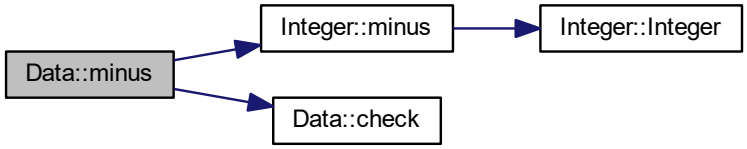
参数

dataPara	dataPara[0]是被减数，dataPara[1]是减数，更多的参数会被忽略

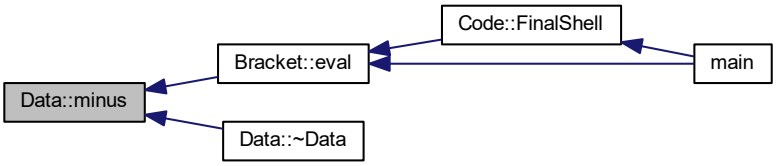
返回

减法的结果

函数调用图:



这是这个函数的调用关系图:



4.5.3.9 mod()

```
Data * Data::mod (
            std::vector< Data *> * dataPara )  [static]
```

求两个数的余数

参数

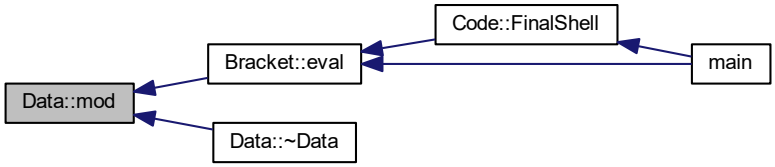
dataPara	dataPara[0]是被除数，dataPara[1]是除数（不能为 0），更多的参数会被忽略

返回

dataPara[0]除以 dataPara[1]得到的余数

函数调用图:

这是这个函数的调用关系图:



4.5.3.10 newData()

```
Data * Data::newData (
            const std::string & str )  [static]
```

根据 str 的内容构造一个数据

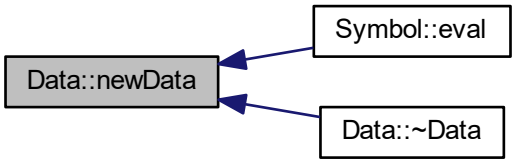
参数

str	作为构造数据时的依据

返回

str 对应的数据，比如 str=="123" 则返回一个指向值为 123 的整数类对象的指针

这是这个函数的调用关系图:



4.5.3.11 plus()

```
Data * Data::plus (
            std::vector< Data *> * dataPara )  [static]
```

任意数量的整数的加法

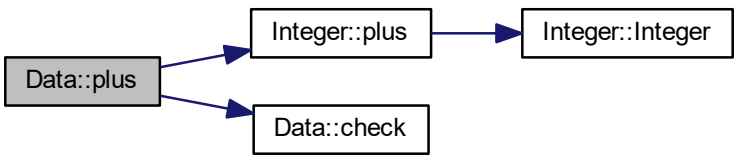
参数

dataPara	被加数

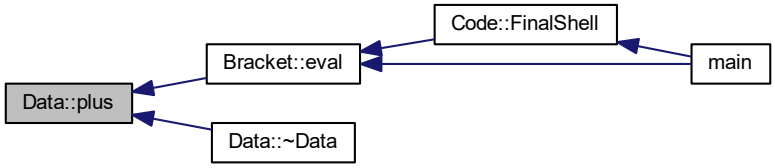
返回

得到的加法的结果

函数调用图:



这是这个函数的调用关系图:



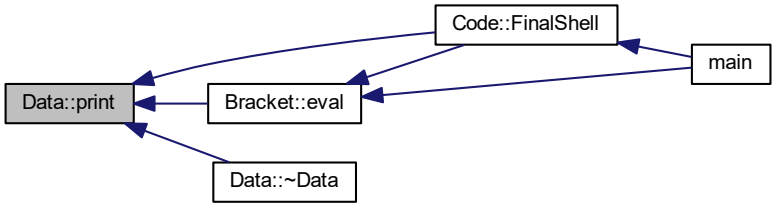
4.5.3.12 print() [1/2]

```
virtual void Data::print ( )  [pure virtual]
```

打印对象

在 Bool , 以及 Integer 内被实现.

这是这个函数的调用关系图:



4.5.3.13 print() [2/2]

```
Data * Data::print (
            std::vector< Data *> * dataPara )  [static]
```

逐个打印一组数据

参数

dataPara	被打印的数据

返回

一个值为 true 的Bool 类型数据

函数调用图:

4.5.3.14 smaller()

```
Data * Data::smaller (
            std::vector< Data *> * dataPara )  [static]
```

比较两个数的大小

参数

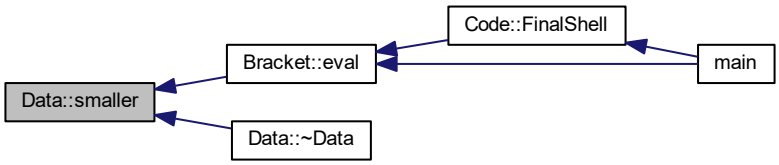
dataPara	被判断的是 dataPara 的前两个数据，更多的数据将会被忽略，被比较的数据只可以是Integer 类

返回

一个Bool 类型的数据，如果 dataPara[0] < dataPara[1]则值为 true，否则为 false

函数调用图:

这是这个函数的调用关系图:



4.5.3.15 times()

```
Data * Data::times (
            std::vector< Data *> * dataPara )  [static]
```

任意数量的整数的乘法

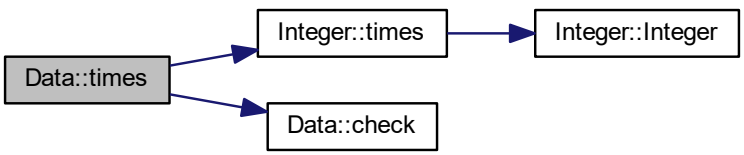
参数

dataPara	被乘数

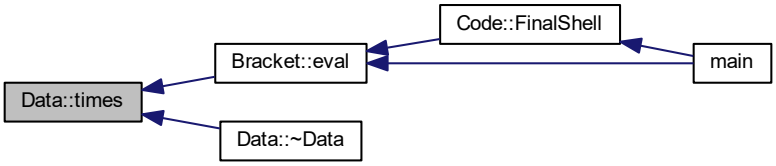
返回

得到的乘法的结果

函数调用图:



这是这个函数的调用关系图:



4.5.3.16 uncite()

```
void Data::uncite ( )
```

将对象的引用次数减少 1 这是这个函数的调用关系图:

4.5.4 友元及相关函数文档

4.5.4.1 Bool

```
friend class Bool  [friend]
```

4.5.4.2 Integer

```
friend class Integer  [friend]
```

4.5.4.3 VarEnv

```
friend class VarEnv  [friend]
```

4.5.5 类成员变量说明

4.5.5.1 cited

```
unsigned int Data::cited  [private]
```

记录当前对象被引用的次数

4.5.5.2 falseData

```
Data * Data::falseData = new Bool(false, 1)  [static]
```

用于作为全局唯一的值为 false 的Bool 类对象使用

4.5.5.3 trueData

```
Data * Data::trueData = new Bool(true, 1)  [static]
```

用于作为全局唯一的值为 true 的Bool 类对象使用

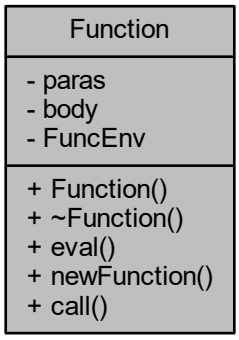
该类的文档由以下文件生成:

- Data.h
- Data.cpp

4.6 Function 类参考

```
#include <Function.h>
```

Function 的协作图:



Public 成员函数

- Function ()
- ~Function ()
- Data * eval (std::vector< Data *> *parasData)

静态 Public 成员函数

- static void newFunction (std::list< Atom *> *source)
- static Data * call (std::string &name, std::vector< Data *> *parasData)

Private 属性

- std::list< std::string > paras
- std::list< Atom * > body

静态 Private 属性

- static std::map< std::string, Function * > FuncEnv

4.6.1 详细描述

用户自定义的函数

4.6.2 构造及析构函数说明

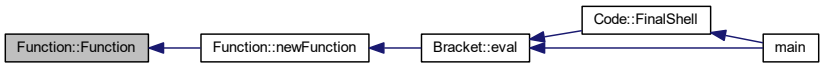
4.6.2.1 Function()

```
Function::Function ( )
```

默认构造函数

返回

这是这个函数的调用关系图:



4.6.2.2 ~Function()

```
Function::~Function ( )
```

析构函数

4.6.3 成员函数说明

4.6.3.1 call()

```
Data * Function::call (
            std::string & name,
            std::vector< Data *> * parasData )  [static]
```

调用名称为 name 的函数，并将相关参数传递过去

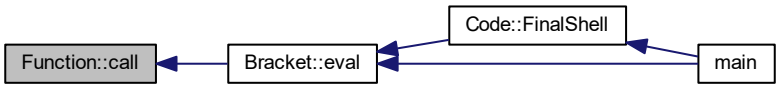
参数

name	被调用函数的名称
parasData	调用函数时使用的参数

返回

指向函数返回的数据的指针

这是这个函数的调用关系图:



4.6.3.2 eval()

```
Data * Function::eval (
            std::vector< Data *> * parasData )
```

根据参数值求函数的返回值

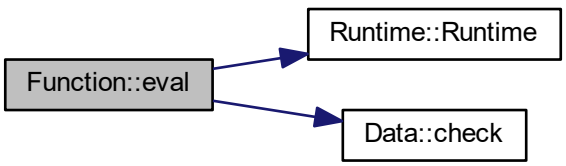
参数

parasData	参数值

返回

指向函数返回的数据的指针

函数调用图:



4.6.3.3 newFunction()

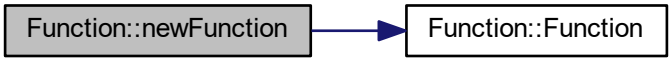
```
void Function::newFunction (
            std::list< Atom *> * source )  [static]
```

根据传入的语法树构造一个新的函数，并录入FuncEnv 属性中

参数

source	包括构造Function 对象所需要的全部信息，比如：(function (test a b) (assign a (+ a b)) (echo a)) 传入参数的内容就是 (test a b) (assign a (+ a b)) (echo a) 对应的语法树

函数调用图:



这是这个函数的调用关系图:

4.6.4 类成员变量说明

4.6.4.1 body

```
std::list<Atom *> Function::body  [private]
```

存储函数体

4.6.4.2 FuncEnv

```
std::map< std::string, Function * > Function::FuncEnv  [static], [private]
```

存储函数名及其对应的Function 对象

4.6.4.3 paras

```
std::list<std::string> Function::paras  [private]
```

用于存储函数的参数名称

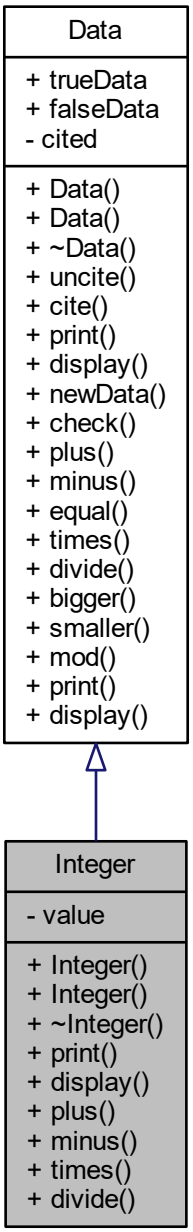
该类的文档由以下文件生成:

- Function.h
- Function.cpp

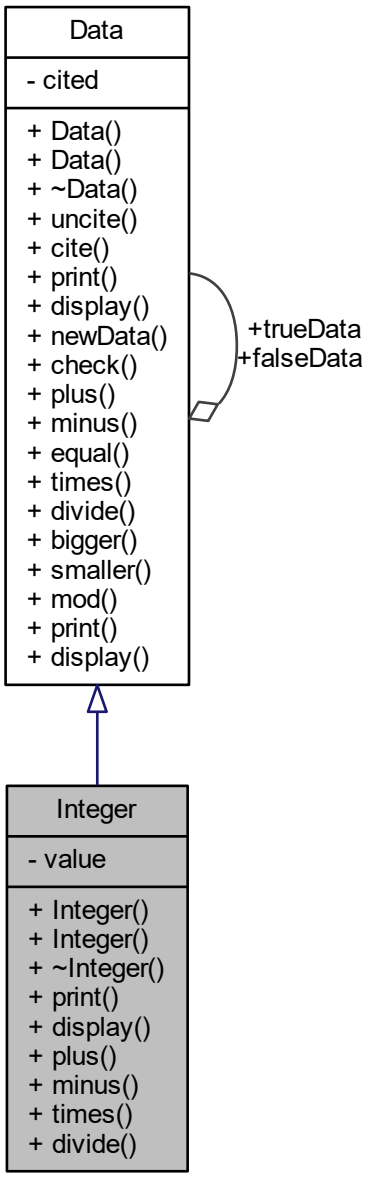
4.7 Integer 类参考

```
#include <Data.h>
```

类 Integer 继承关系图:



Integer 的协作图:



Public 成员函数

- Integer ()
- Integer (const long &value)
- virtual ~Integer ()
- virtual void print ()
- virtual void display ()

静态 Public 成员函数

- static Data * plus (std::vector< Data *> *dataPara)
- static Data * minus (std::vector< Data *> *dataPara)
- static Data * times (std::vector< Data *> *dataPara)
- static Data * divide (std::vector< Data *> *dataPara)

Private 属性

- long value

友元

- class Data
- class Runtime

额外继承的成员函数

4.7.1 详细描述

表示一个整数

4.7.2 构造及析构函数说明

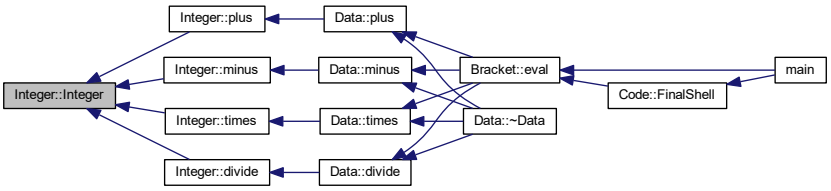
4.7.2.1 Integer() [1/2]

```
Integer::Integer ( )
```

构造函数

返回

这是这个函数的调用关系图:



4.7.2.2 Integer() [2/2]

```
Integer::Integer (
            const long & value )
```

将值设置为 value

参数

value	用于初始化 value 属性的整数

返回

4.7.2.3 ~Integer()

```
Integer::~Integer ( )  [virtual]
```

析构函数

4.7.3 成员函数说明

4.7.3.1 display()

```
void Integer::display ( )  [virtual]
```

打印当前对象的详细信息

实现了 Data.

4.7.3.2 divide()

```
Data * Integer::divide (
            std::vector< Data *> * dataPara )  [static]
```

两个整数的除法

参数

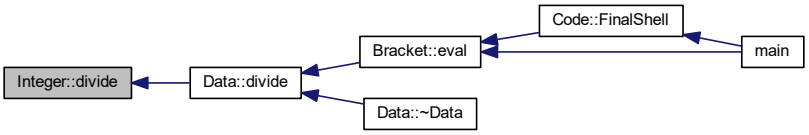
dataPara	dataPara[0]是被除数，dataPara[1]是除数（不能为 0），更多的参数会被忽略

返回

除法的结果

函数调用图:

这是这个函数的调用关系图:



4.7.3.3 minus()

```
Data * Integer::minus (
            std::vector< Data *> * dataPara )  [static]
```

两个整数的减法

参数

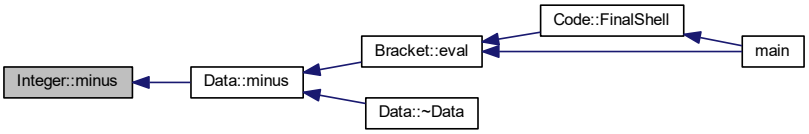
dataPara	dataPara[0]是被减数，dataPara[1]是减数，更多的参数会被忽略

返回

减法的结果

函数调用图:

这是这个函数的调用关系图:



4.7.3.4 plus()

```
Data * Integer::plus (
            std::vector< Data *> * dataPara )  [static]
```

任意数量的整数的加法

参数

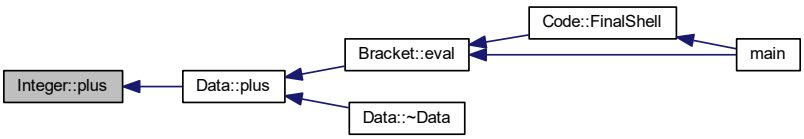
dataPara	被加数

返回

得到的加法的结果

函数调用图:

这是这个函数的调用关系图:



4.7.3.5 print()

```
void Integer::print ( )  [virtual]
```

打印当前对象的值

实现了 Data.

4.7.3.6 times()

```
Data * Integer::times (
            std::vector< Data *> * dataPara )  [static]
```

任意数量的整数的乘法

参数

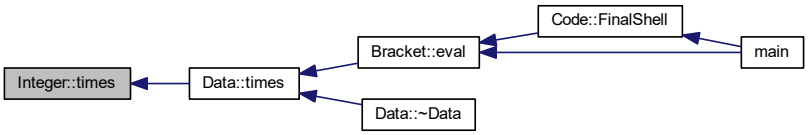
dataPara	被乘数

返回

得到的乘法的结果

函数调用图:

这是这个函数的调用关系图:



4.7.4 友元及相关函数文档

4.7.4.1 Data

```
friend class Data  [friend]
```

4.7.4.2 Runtime

```
friend class Runtime  [friend]
```

4.7.5 类成员变量说明

4.7.5.1 value

```
long Integer::value  [private]
```

存储对象的数值

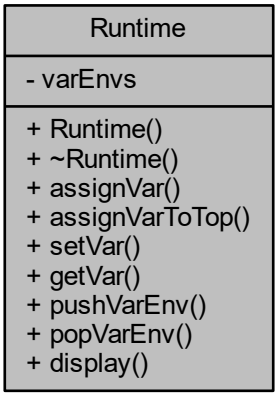
该类的文档由以下文件生成:

- Data.h
- Integer.cpp

4.8 Runtime 类参考

```
#include <Runtime.h>
```

Runtime 的协作图:



Public 成员函数

- Runtime ()
- ~Runtime ()
- void assignVar (std::string &name, Data *data)
- void assignVarToTop (std::string &name, Data *data)
- void setVar (std::string &name, Data *data)
- Data * getVar (std::string &name)
- void pushVarEnv ()
- void popVarEnv ()
- void display ()

Private 属性

- std::forward_list< VarEnv * > varEnvs

4.8.1 详细描述

运行时，运行一段代码时，会从Runtime 对象中获得变量名对应的数据，以及进行赋值操作等

4.8.2 构造及析构函数说明

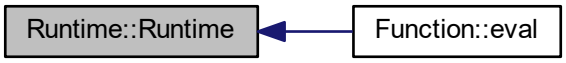
4.8.2.1 Runtime()

```
Runtime::Runtime ( )
```

构造函数

返回

这是这个函数的调用关系图:



4.8.2.2 ~Runtime()

```
Runtime::~Runtime ( )
```

析构函数

4.8.3 成员函数说明

4.8.3.1 assignVar()

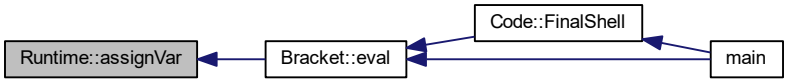
```
void Runtime::assignVar (
            std::string & name,
            Data * data )
```

将变量名 name 指向 data 数据，如果没有在Runtime 中找到 name 变量，就会新建一个名称为 name 的变量，并指向 data，如果找到了，则将 name 原来指向的数据的被引用次数减少一，在将 name 指向 data

参数

name	被赋值的变量名称
data	待赋值的数据

这是这个函数的调用关系图:



4.8.3.2 assignVarToTop()

```
void Runtime::assignVarToTop (
            std::string & name,
            Data * data )
```

在最顶层的VarEnv 中新建变量并赋值，用于实现函数调用时传递参数

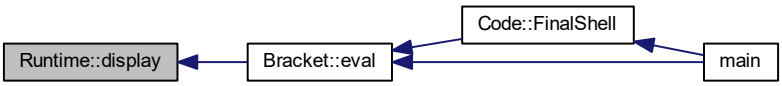
参数

name	被赋值的变量名称
data	待赋值的数据

4.8.3.3 display()

```
void Runtime::display ( )
```

打印该对象中所有的变量的详细信息这是这个函数的调用关系图:



4.8.3.4 getVar()

```
Data * Runtime::getVar (
            std::string & name )
```

获取 name 变量名对应的数据

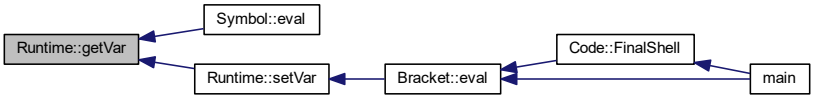
参数

name	待获取的变量的名称

返回

name 对应的数据

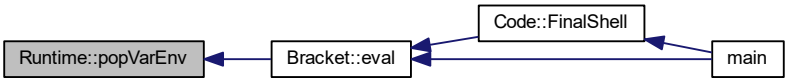
这是这个函数的调用关系图:



4.8.3.5 popVarEnv()

```
void Runtime::popVarEnv ( )
```

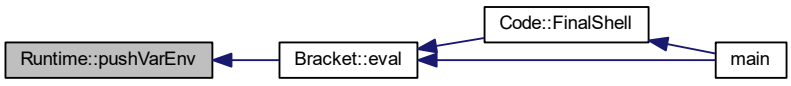
出栈一个VarEnv 并析构它这是这个函数的调用关系图:



4.8.3.6 pushVarEnv()

```
void Runtime::pushVarEnv ( )
```

构造一个VarEnv 并入栈这是这个函数的调用关系图:



4.8.3.7 setVar()

```
void Runtime::setVar (
            std::string & name,
            Data * data )
```

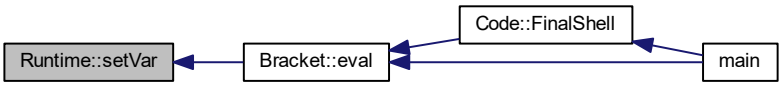
将 name 对应的整数重新设置为 data 的值，和 assign 的区别在于，set 会改变 name 指向的数据的值

参数

name	被赋值的变量名称
data	待赋值的数据

函数调用图:

这是这个函数的调用关系图:



4.8.4 类成员变量说明

4.8.4.1 varEnvs

```
std::forward_list<VarEnv *> Runtime::varEnvs  [private]
```

作为一个存储变量的栈使用

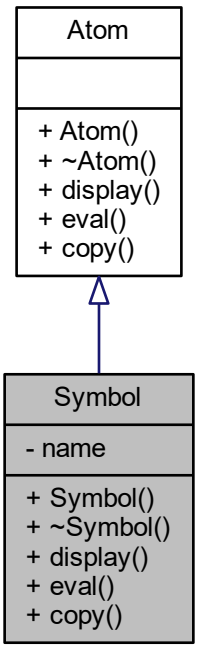
该类的文档由以下文件生成:

- Runtime.h
- Runtime.cpp

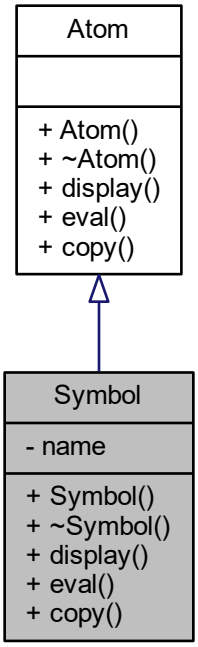
4.9 Symbol 类参考

```
#include <Atom.h>
```

类 Symbol 继承关系图:



Symbol 的协作图:



Public 成员函数

- Symbol (const std::string &str)
- virtual ~Symbol ()
- virtual void display (int indent)
- virtual Data * eval (Runtime *runtime)
- virtual Atom * copy ()

Private 属性

- std::string name

友元

- class Runtime
- class Bracket
- class Function

4.9.1 详细描述

用于存储一个符号对应的语法结构，这个符号可以是一个常量，也可以是一个变量名，比如 (test 1 2 a) 中的 1、2、a 都是一个符号

4.9.2 构造及析构函数说明

4.9.2.1 Symbol()

```
Symbol::Symbol (
            const std::string & str )
```

根据 str 来构造

参数

str	比如为"true" "false" "123" "a" 等

返回

4.9.2.2 ~Symbol()

```
Symbol::~Symbol ( )  [virtual]
```

析构函数

4.9.3 成员函数说明

4.9.3.1 copy()

```
Atom * Symbol::copy ( )  [virtual]
```

复制当前对象并返回

返回

返回复制得到的对象

实现了 Atom.

4.9.3.2 display()

```
void Symbol::display (
            int indent )  [virtual]
```

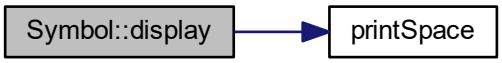
打印本对象对应的符号，并使用 indent 级的缩进

参数

indent	打印时使用的缩进级数

实现了 Atom.

函数调用图:



4.9.3.3 eval()

```
Data * Symbol::eval (
            Runtime * runtime )  [virtual]
```

获取当前对象的值，比如如果 name 属性是一个常量，比如"123"，就返回一个值为 123 的整数数据，如果是变量名就在 runtime 中查询该变量名对应的数据并且返回

参数

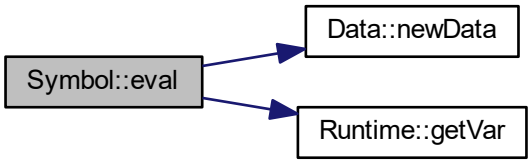
runtime	用于获取变量对应的数据

返回

如果 name 属性是一个常量，比如"123"，就返回一个值为 123 的整数数据，如果是变量名就在 runtime 中查询该变量名对应的数据并且返回

实现了 Atom.

函数调用图:



4.9.4 友元及相关函数文档

4.9.4.1 Bracket

```
friend class Bracket  [friend]
```

4.9.4.2 Function

```
friend class Function  [friend]
```

4.9.4.3 Runtime

```
friend class Runtime  [friend]
```

4.9.5 类成员变量说明

4.9.5.1 name

```
std::string Symbol::name  [private]
```

用于存储符号对应的字符串，比如"a" "123" "true" "false" 等

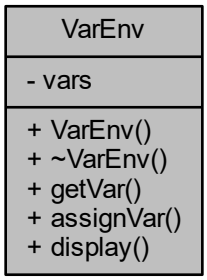
该类的文档由以下文件生成:

- Atom.h
- Atom.cpp

4.10 VarEnv 类参考

```
#include <VarEnv.h>
```

VarEnv 的协作图:



Public 成员函数

- VarEnv ()
- ~VarEnv ()
- Data * getVar (const std::string &name)
- void assignVar (const std::string &name, Data *data)
- void display ()

Private 属性

- std::map< std::string, Data * > vars

4.10.1 详细描述

变量环境

4.10.2 构造及析构函数说明

4.10.2.1 VarEnv()

```
VarEnv::VarEnv ( )
```

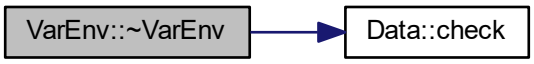
构造函数

返回

4.10.2.2 ~VarEnv()

```
VarEnv::~VarEnv ( )
```

析构函数函数调用图:



4.10.3 成员函数说明

4.10.3.1 assignVar()

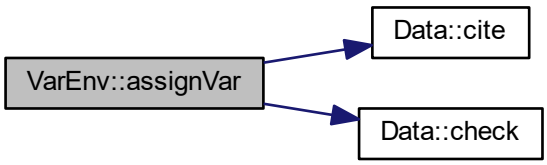
```
void VarEnv::assignVar (
            const std::string & name,
            Data * data )
```

将变量名 name 指向 data 数据，如果没有在Runtime 中找到 name 变量，就会新建一个名称为 name 的变量，并指向 data，如果找到了，则将 name 原来指向的数据的被引用次数减少一，在将 name 指向 data

参数

name	被赋值的变量名称
data	待赋值的数据

函数调用图:



4.10.3.2 display()

```
void VarEnv::display ( )
```

打印该对象中所有的变量的详细信息

4.10.3.3 getVar()

```
Data * VarEnv::getVar (
            const std::string & name )
```

获取 name 变量名对应的数据

参数

name	待获取的变量的名称

返回

name 对应的数据，如果未找到，返回的是NULL

4.10.4 类成员变量说明

4.10.4.1 vars

```
std::map<std::string, Data *> VarEnv::vars  [private]
```

用于存储变量名及其对应的数据

该类的文档由以下文件生成:

- VarEnv.h
- VarEnv.cpp

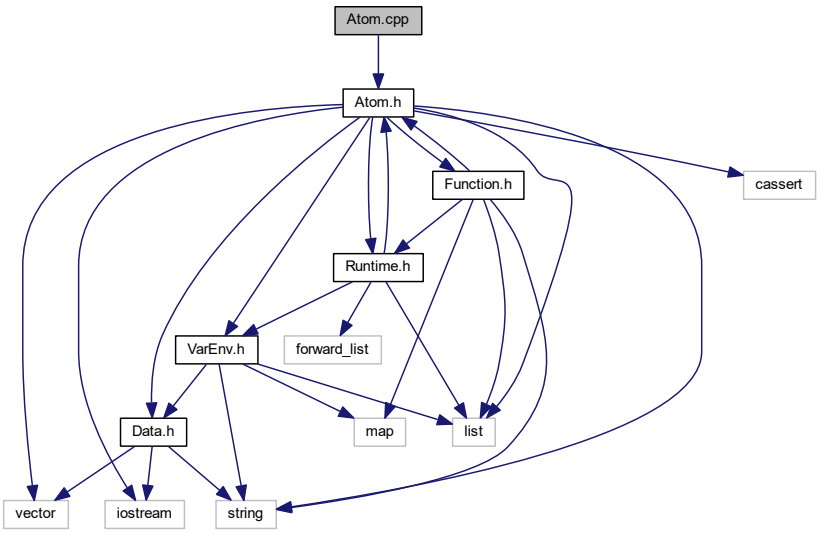
Chapter 5

文件说明

5.1 Atom.cpp 文件参考

```
#include "Atom.h"
```

Atom.cpp 的引用 (Include) 关系图:



函数

- void printSpace (int i)
- bool isBlank (const char &achar)
- std::string getWord (const std::string &str, unsigned long &pos)

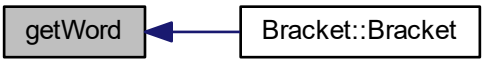
5.1.1 函数说明

5.1.1.1 getWord()

```
std::string getWord (
             const std::string & str,
             unsigned long & pos )
```

函数调用图:

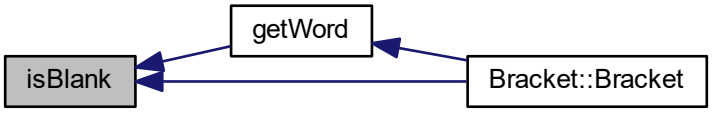
这是这个函数的调用关系图:



5.1.1.2 isBlank()

```
bool isBlank (
             const char & achar )
```

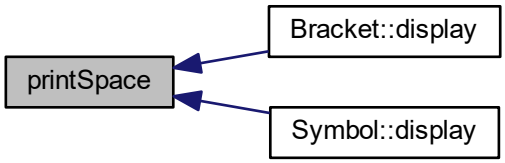
这是这个函数的调用关系图:



5.1.1.3 printSpace()

```
void printSpace (
             int i )
```

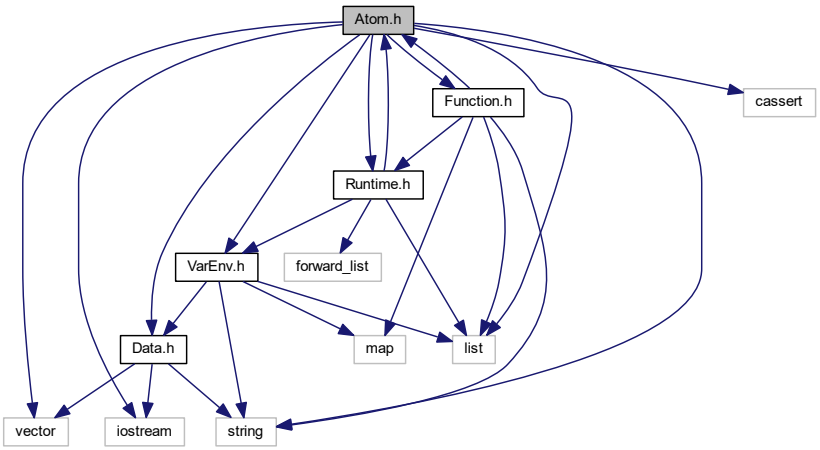
这是这个函数的调用关系图:



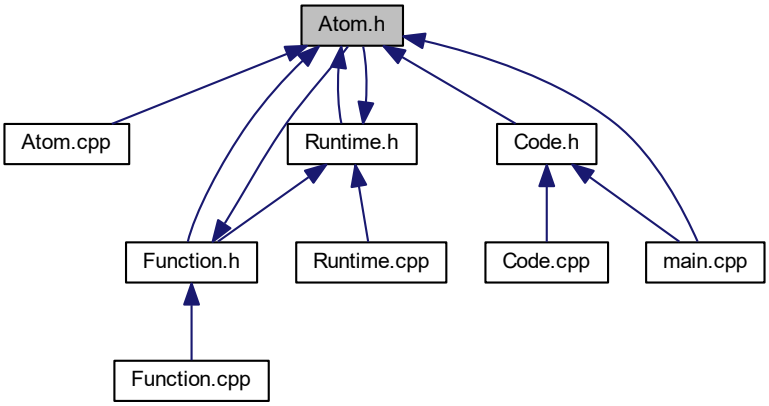
5.2 Atom.h 文件参考

```
#include <vector>
#include <iostream>
#include <string>
#include <list>
#include <cassert>
#include "Data.h"
#include "Runtime.h"
#include "Function.h"
#include "VarEnv.h"
```

Atom.h 的引用 (Include) 关系图:



此图展示该文件直接或间接的被哪些文件引用了:



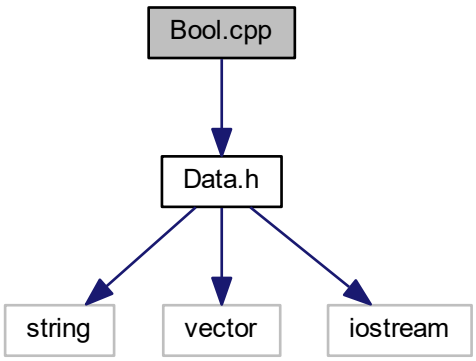
类

- class Atom
- class Bracket
- class Symbol

5.3 Bool.cpp 文件参考

```
#include "Data.h"
```

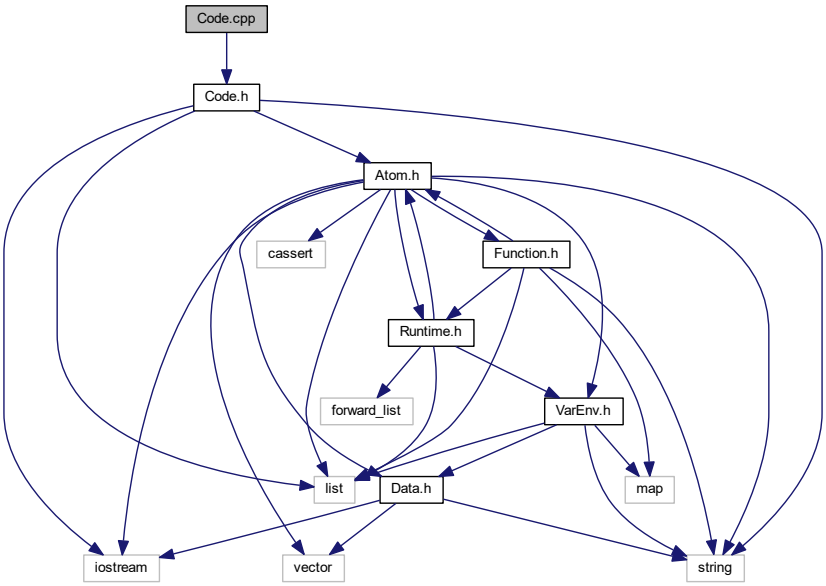
Bool.cpp 的引用 (Include) 关系图:



5.4 Code.cpp 文件参考

```
#include "Code.h"
```

Code.cpp 的引用 (Include) 关系图:



函数

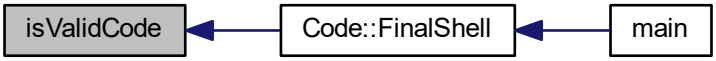
- bool isValidCode (std::string &str)

5.4.1 函数说明

5.4.1.1 isValidCode()

```
bool isValidCode (
            std::string & str )
```

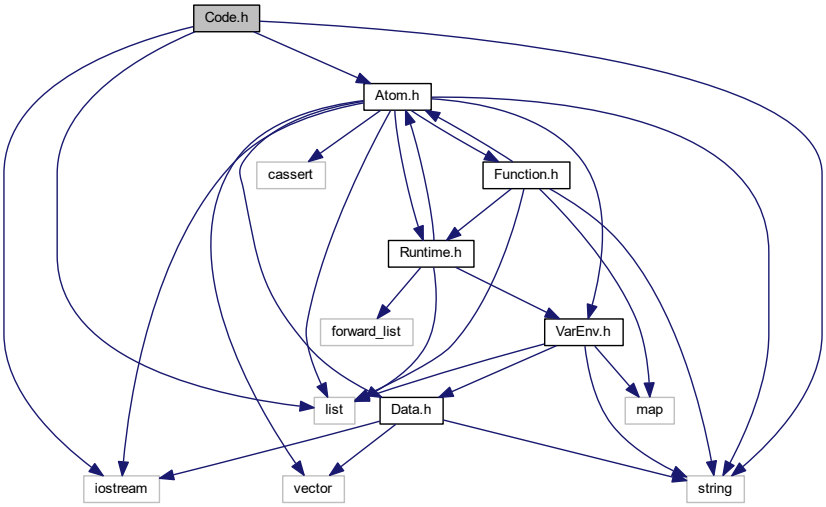
这是这个函数的调用关系图:



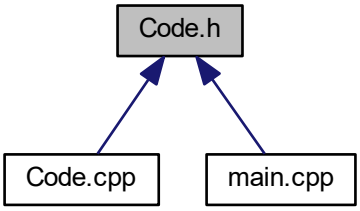
5.5 Code.h 文件参考

```
#include <string>
#include <list>
#include <iostream>
#include "Atom.h"
```

Code.h 的引用 (Include) 关系图:



此图展示该文件直接或间接的被哪些文件引用了:



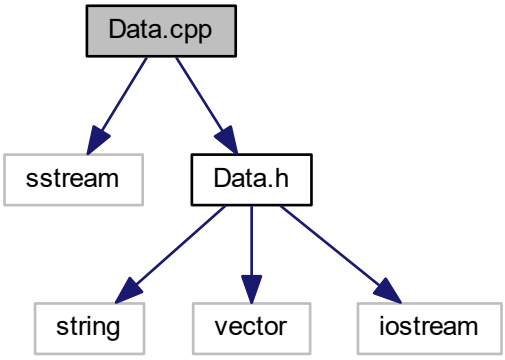
类

- class Code

5.6 Data.cpp 文件参考

```
#include <sstream>
#include "Data.h"
```

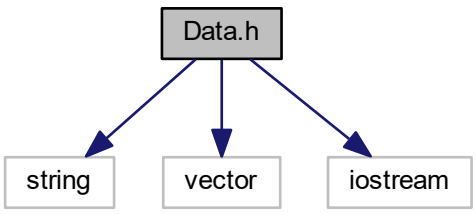
Data.cpp 的引用 (Include) 关系图:



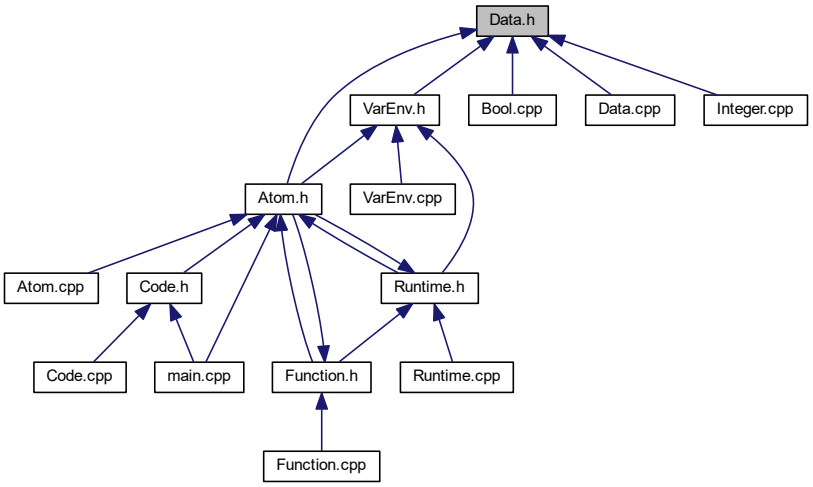
5.7 Data.h 文件参考

```
#include <string>
#include <vector>
#include <iostream>
```

Data.h 的引用 (Include) 关系图:



此图展示该文件直接或间接的被哪些文件引用了:



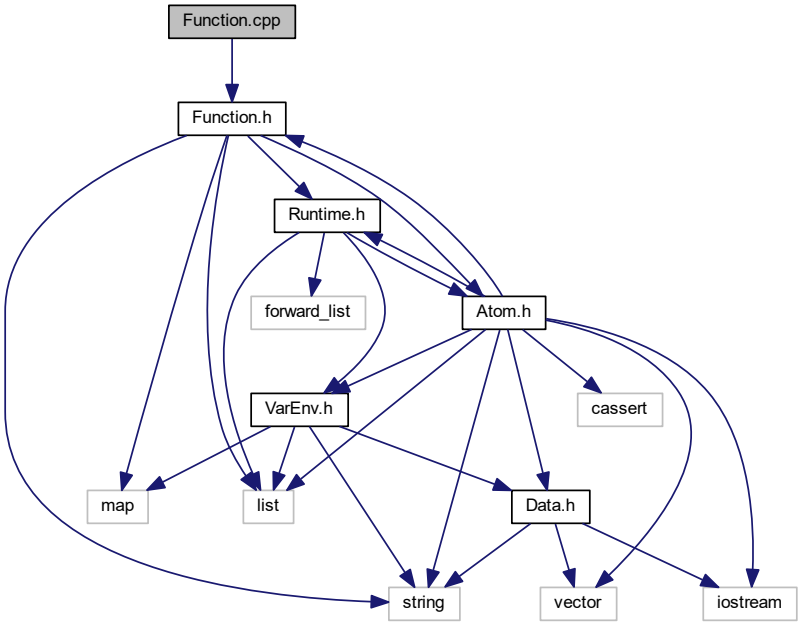
类

- class Data
- class Integer
- class Bool

5.8 Function.cpp 文件参考

```
#include "Function.h"
```

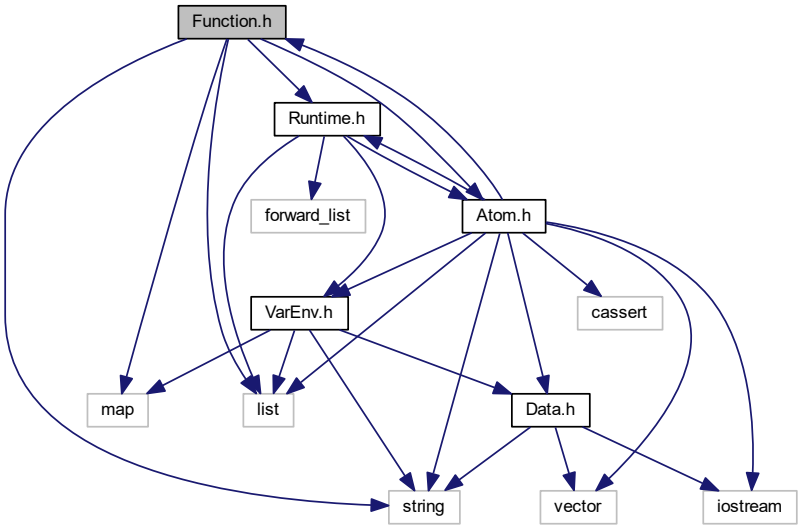
Function.cpp 的引用 (Include) 关系图:



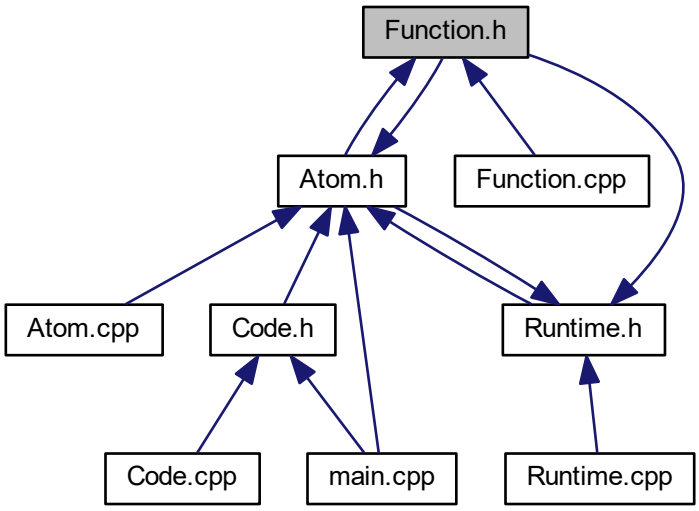
5.9 Function.h 文件参考

```
#include <list>
#include <string>
#include <map>
#include "Runtime.h"
#include "Atom.h"
```

Function.h 的引用 (Include) 关系图:



此图展示该文件直接或间接的被哪些文件引用了:



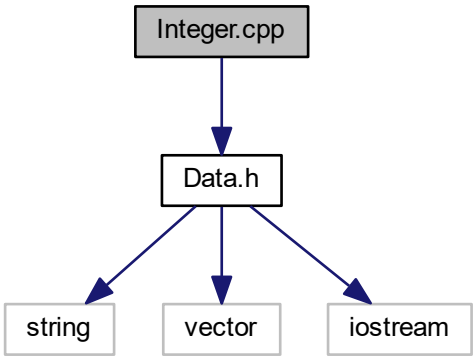
类

- class Function

5.10 Integer.cpp 文件参考

```
#include "Data.h"
```

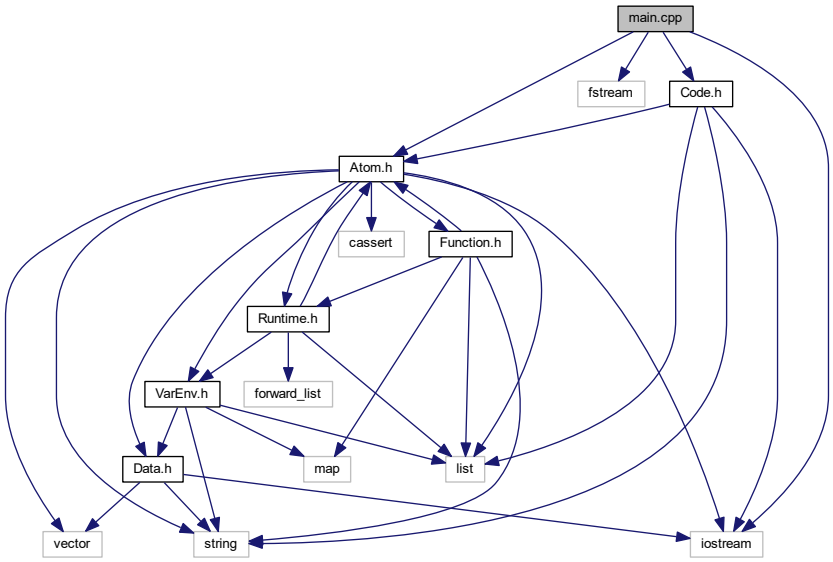
Integer.cpp 的引用 (Include) 关系图:



5.11 main.cpp 文件参考

```
#include <iostream>
#include <fstream>
#include "Code.h"
#include "Atom.h"
```

main.cpp 的引用 (Include) 关系图:



函数

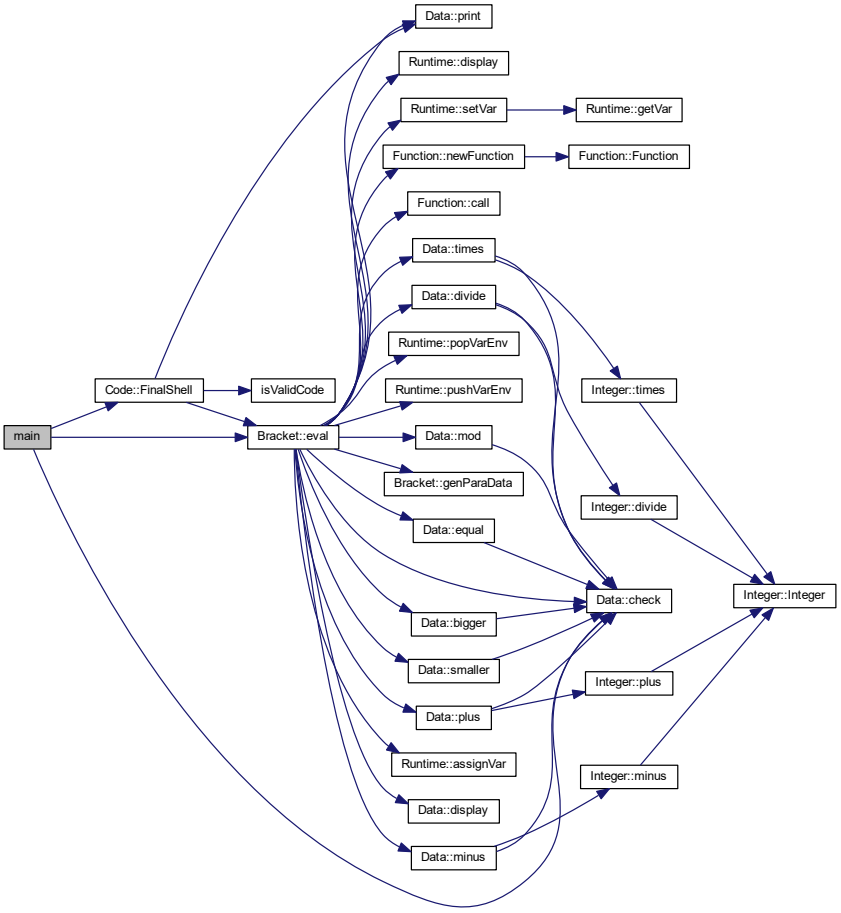
- int main (int argc, char ∗argv[ ])

5.11.1 函数说明

5.11.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

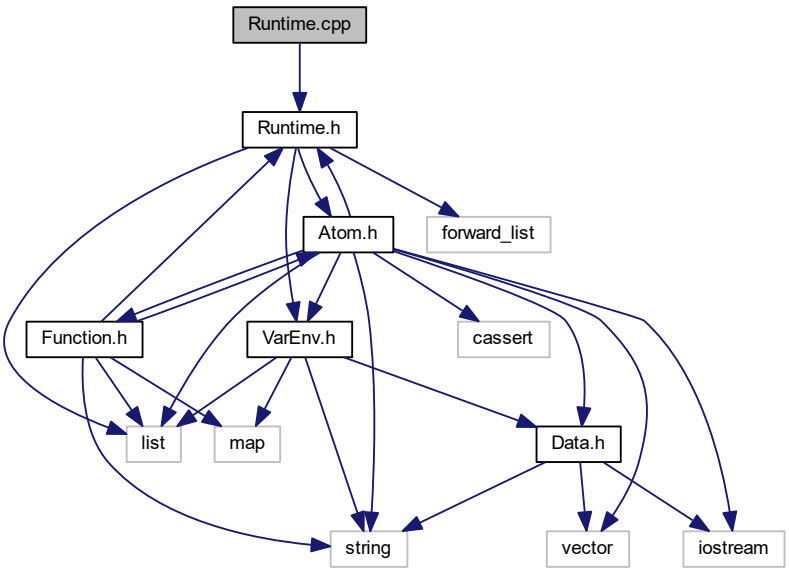
函数调用图:



5.12 Runtime.cpp 文件参考

```
#include "Runtime.h"
```

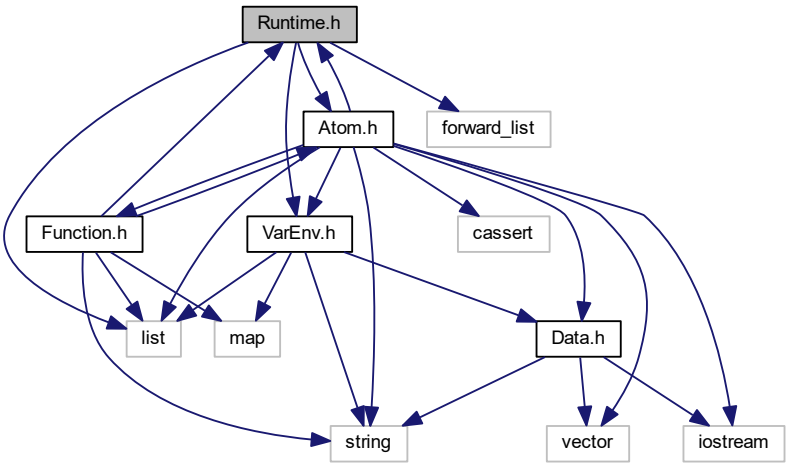
Runtime.cpp 的引用 (Include) 关系图:



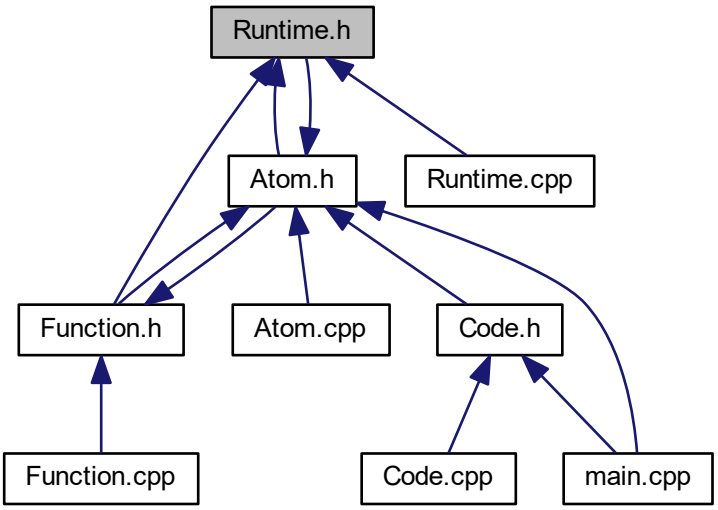
5.13 Runtime.h 文件参考

```
#include <list>
#include <forward_list>
#include "VarEnv.h"
#include "Atom.h"
```

Runtime.h 的引用 (Include) 关系图:



此图展示该文件直接或间接的被哪些文件引用了:



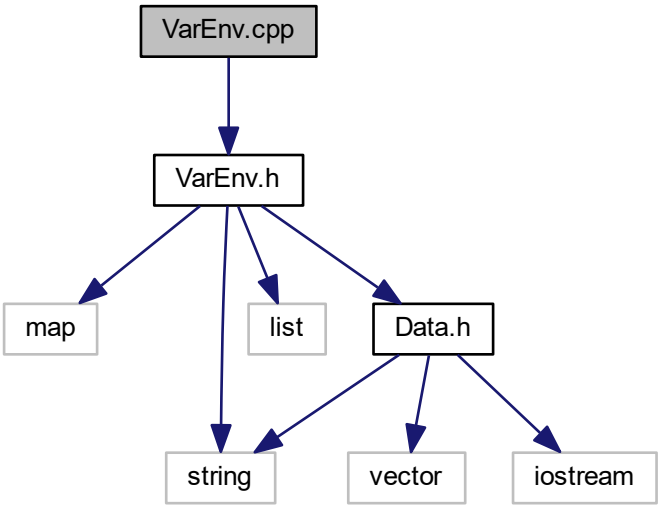
类

- class Runtime

5.14 VarEnv.cpp 文件参考

```
#include "VarEnv.h"
```

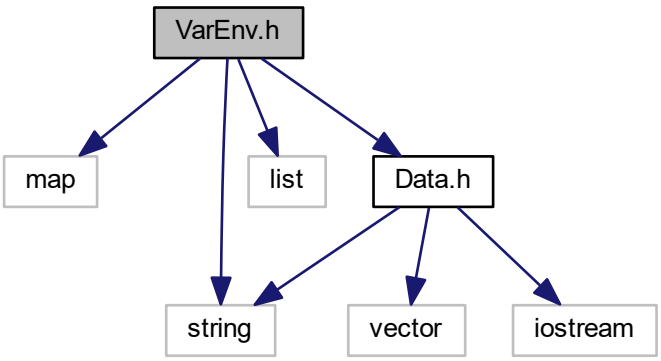
VarEnv.cpp 的引用 (Include) 关系图:



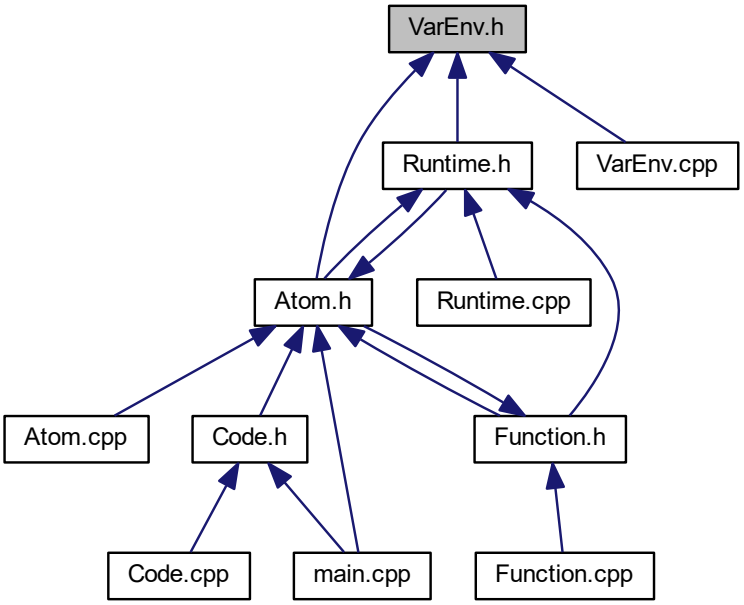
5.15 VarEnv.h 文件参考

```
#include <map>
#include <string>
#include <list>
#include "Data.h"
```

VarEnv.h 的引用 (Include) 关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- class VarEnv

Index