额外功能

日志功能

实现了包含4个日志等级（`debug,info,warning,error`），并可以**调整等级和输出流**的日志接口。`message`输出接口则用于正常的输出内容，如`Print`节点的输出。

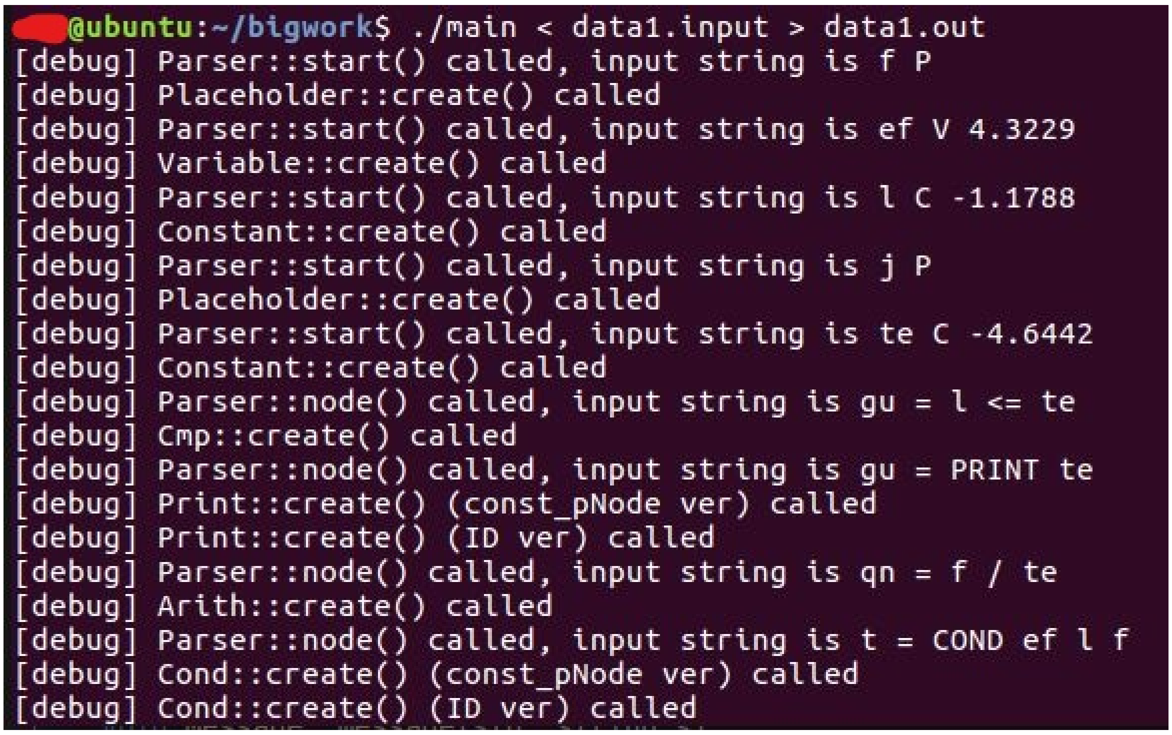
日志输出默认关联`std::cerr`,`message`输出默认关联`std::cout`。

- debug : 当 `log_level == 1` 时开启, 提供非常详尽的计算过程供开发者纠错
- info : 当 `log_level <= 2` 时开启, 提供建图重要过程的提示
- warning : 当`log_level <= 3` 时开启, 提示使用者潜在危险
- error : 当`log_level <= 4` 时开启, 指示使用者严重错误
- 当 `log_level == 5` 时, 即关闭日志功能。

示例--调整等级

(注:此次测试环境为Ubuntu系统, 但在Windows上也可顺利进行)

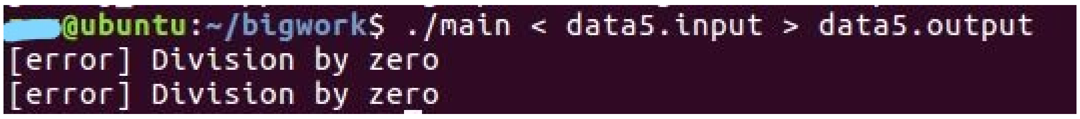
假设作为开发者, 想要确保计算图能正确建立进行调试 。 在测试代码中，调用本库之前使用`Message::set_log_level`接口将日志等级置为`1`, 报出所有信息方便调试。 我们利用OJ上的样例`example_all/data1.input` 作为调试输入。 `make`编译后执行命令： `./main < data1.input > data1.out`



```
[redacted]@ubuntu:~/bigwork$ ./main < data1.input > data1.out
[debug] Parser::start() called, input string is f P
[debug] Placeholder::create() called
[debug] Parser::start() called, input string is ef V 4.3229
[debug] Variable::create() called
[debug] Parser::start() called, input string is l C -1.1788
[debug] Constant::create() called
[debug] Parser::start() called, input string is j P
[debug] Placeholder::create() called
[debug] Parser::start() called, input string is te C -4.6442
[debug] Constant::create() called
[debug] Parser::node() called, input string is gu = l <= te
[debug] Cmp::create() called
[debug] Parser::node() called, input string is gu = PRINT te
[debug] Print::create() (const_pNode ver) called
[debug] Print::create() (ID ver) called
[debug] Parser::node() called, input string is qn = f / te
[debug] Arith::create() called
[debug] Parser::node() called, input string is t = COND ef l f
[debug] Cond::create() (const_pNode ver) called
[debug] Cond::create() (ID ver) called
```

如上图 (仅列出部分), 可在命令行中逐行看到日志输出，在此示例中均为`debug` 级别。 因为样例中输入皆正确 , 故没有报出`info` , `warning` , 及 `error` 。 而 `data1.out` 也与OJ上要求的输出一致

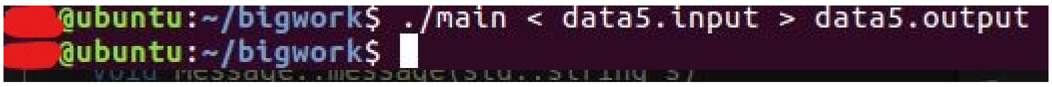
设定日志等级为`3`, 则只会报出`warning` 以及 `error`。 再利用`example_all / data5.input` 作为样例输入。



```
[redacted]@ubuntu:~/bigwork$ ./main < data5.input > data5.output
[error] Division by zero
[error] Division by zero
```

如上图, 因为在过程中计算图进行除法运算时遇到除数为零, 故会报出`error`。 而因为`log_level`已调至3, 故不会报出`debug`和`info` (本样例无`warning`)。

类似地, 要关闭日志功能, 只要将`log_level`调整至5, 便不会有报错提示(如下图)。



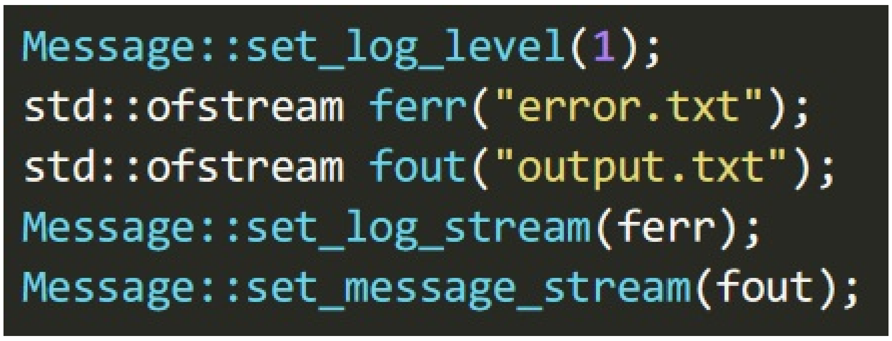
```
[redacted]@ubuntu:~/bigwork$ ./main < data5.input > data5.output
[redacted]@ubuntu:~/bigwork$
```

示例--调整输出流

(备注:此次调试环境为Ubuntu系统, 在Windows上也可顺利进行)

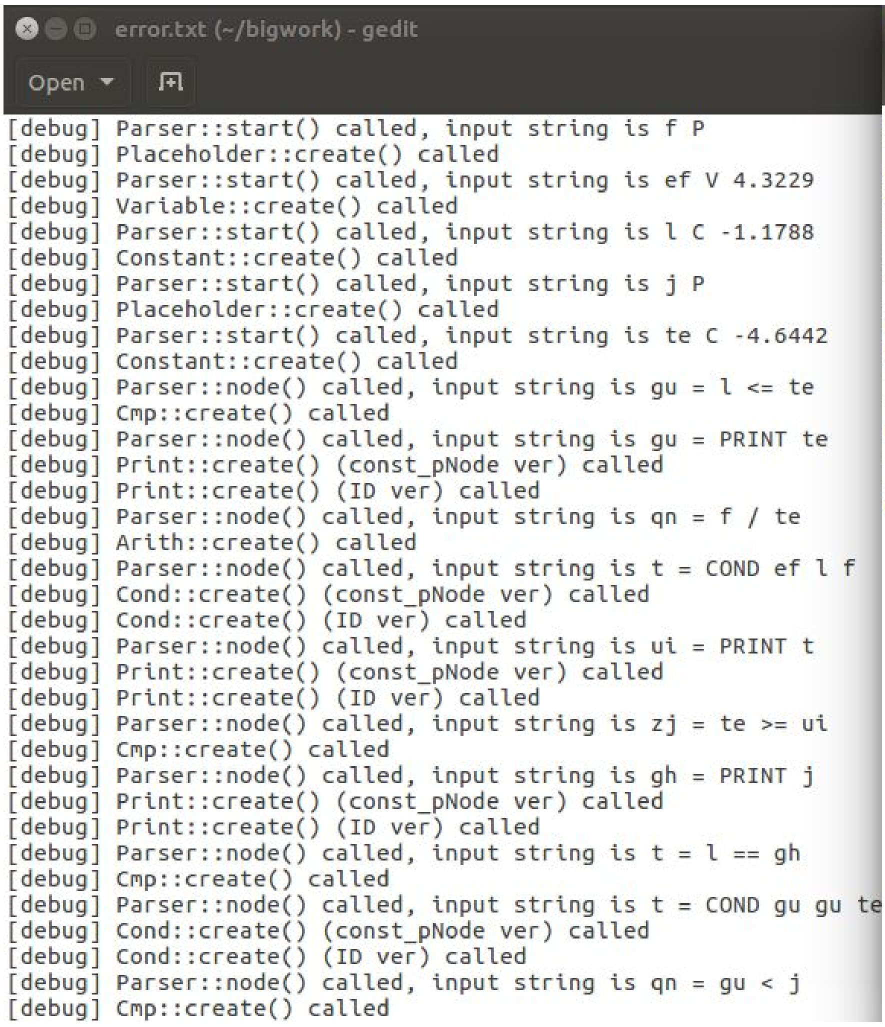
有时后, 我们并不一定想直接把结果输出到屏幕上、或者不想让一长串的报错信息直接输出到命令行。利用`Message::set_log_stream`与`Message::set_message_stream`可以分别为日志和`message`指定输出流。

假定今天我们为调试者想记录调试过程到文件中。如下图，指定两个`std::ofstream`对象。

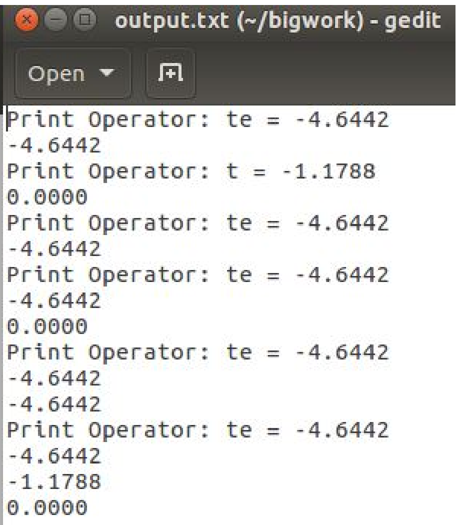


```
Message::set_log_level(1);
std::ofstream ferr("error.txt");
std::ofstream fout("output.txt");
Message::set_log_stream(ferr);
Message::set_message_stream(fout);
```

我们再利用`data1.input`进行调试：`./main < data1.input`。



```
[debug] Parser::start() called, input string is f P
[debug] Placeholder::create() called
[debug] Parser::start() called, input string is ef V 4.3229
[debug] Variable::create() called
[debug] Parser::start() called, input string is l C -1.1788
[debug] Constant::create() called
[debug] Parser::start() called, input string is j P
[debug] Placeholder::create() called
[debug] Parser::start() called, input string is te C -4.6442
[debug] Constant::create() called
[debug] Parser::node() called, input string is gu = l <= te
[debug] Cmp::create() called
[debug] Parser::node() called, input string is gu = PRINT te
[debug] Print::create() (const_pNode ver) called
[debug] Print::create() (ID ver) called
[debug] Parser::node() called, input string is qn = f / te
[debug] Arith::create() called
[debug] Parser::node() called, input string is t = COND ef l f
[debug] Cond::create() (const_pNode ver) called
[debug] Cond::create() (ID ver) called
[debug] Parser::node() called, input string is ui = PRINT t
[debug] Print::create() (const_pNode ver) called
[debug] Print::create() (ID ver) called
[debug] Parser::node() called, input string is zj = te >= ui
[debug] Cmp::create() called
[debug] Parser::node() called, input string is gh = PRINT j
[debug] Print::create() (const_pNode ver) called
[debug] Print::create() (ID ver) called
[debug] Parser::node() called, input string is t = l == gh
[debug] Cmp::create() called
[debug] Parser::node() called, input string is t = COND gu gu te
[debug] Cond::create() (const_pNode ver) called
[debug] Cond::create() (ID ver) called
[debug] Parser::node() called, input string is qn = gu < j
[debug] Cmp::create() called
```



```
Print Operator: te = -4.6442
-4.6442
Print Operator: t = -1.1788
0.0000
Print Operator: te = -4.6442
-4.6442
Print Operator: te = -4.6442
-4.6442
0.0000
Print Operator: te = -4.6442
-4.6442
-4.6442
Print Operator: te = -4.6442
-4.6442
-1.1788
0.0000
```

可发现原目录下已经多了`error.txt`以及`output.txt`，且正确输出。

Graph和Session机制

可以分离不同的计算图和会话。一个会话管理着一个计算图中的一组变量取值状态。

字符串Parser

虽然没有要求，但我们提供了一个额外的工具，用于解析OJ测试时描述计算图操作的语言。