四元式序列:

“Common.h”里面申明了ConstantList<Quaternary> quaterList;

使用方法:文件最后有个例子

**quaterList**

API:

|  |  |
| --- | --- |
| set(int position, Quaternary value) | 设置值 |
| get(int position) | 返回值 |
| insert(Quaternary value) | 插入value并返回position |
| size() | 返回表大小 |
| find(Quaternary value) | 查找表中是否存在value,存在则返回offset,否则返回-1 |

**四元式格式Quater**

|  |  |  |  |
| --- | --- | --- | --- |
| 操作option | 第一操作数first | 第二操作数second | 目标target |

四元式里面操作数和目标均为token信息

option: std::string {+,-,\*,/, =, ==, >, <, >=, <=, if, ie, else, do…}

API:

|  |  |
| --- | --- |
| setOption(std::string option) | 设置option的值 |
| setFirst(Token token) | ... |
| setSecond(Token token) | ... |
| setTarget(Token token) | ... |
| getOption() | 返回option的值 |
| getFirst() | ... |
| getSecond() | ... |
| getTarget() | ... |

**Token结构**

|  |  |  |
| --- | --- | --- |
| Category:union Cat(编码) | Offset:int (偏移量) | Activity:bool(活跃信息) |

API:

|  |  |
| --- | --- |
| setCategory(Cat category) | 设置Category值 |
| setOffset(int offset) | 设置Offset值 |
| setActivity(bool activity) | 设置活跃信息(默认不设置初始化为true) |
| getCategory() | 返回Category值 |
| getOffset() | 返回Offset值 |
| getActivity() | 返回活跃信息 |

**符号表SymbolList**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name: string | Type: int(类型表偏移量) | Category: enum Cat(编码) | Offset: int(根据不同cat表示各类表偏移量) | Activity:bool(变量活跃信息,只有变量有意义,默认为true) |

API:

|  |  |
| --- | --- |
| setName(int position, string name) | 设置Name项 |
| setType(int position, int type) | 设置Type项 |
| setCategory(int position, Cat category) | 设置Category项 |
| setOffset(int position, int address) | 设置Offset项 |
| setActivity(int position, bool activity) | 设置Activity项 |
| find(string name) | 查找是否存在name，不存在返回-1，否则返回符号表中位置 |
| insert(string name) | 插入name新项并返回新项在符号表中的位置 |
| getName(int position) | 返回position位置的name,不存在则返回空串 |
| getType(int position) | 返回position位置的Type值 |
| getCategory(int position) | 返回position位置的Category值 |
| getActivity(int position) | 返回position位置的Activity值 |

活跃信息activity在token里面,符号表里面也有活跃信息

**例子:**

#include “Common.h”

Quaternary quat = quaterList.get(3);//得到第三个四元式放在quat变量里

quaterList.set(4, quat);//将第四个四元式修改为第三个四元式

std::string op = quat.getOption();//获得第三个四元式的操作符

Token fst = quat.getFirst();//获得第一个操作数

...

建议先写文档,之后再动手写代码

四元式优化类要求:

只提供一个优化的public方法,把其他方法和变量设为私有成员变量,对外屏蔽(像我scanner那样),不要在头文件里定义变量,不要在头文件里使用using namespace std;可以在cpp文件里使用

优化后结果仍然存入quaterList中,替换掉原来的序列

四元式翻译类:

和四元式优化一样,包含Common.h头文件,对quaterList内的四元式进行翻译,注意事项和四元式优化一样