struct ITokCollection

{

virtual bool get()=0;

//获取下一个Token集合，如果已经到了文件尾，返回false

virtual int length()=0;

//实现的类中会包含tokens的集合，该方法将返回集合中token的数量

virtual std::string& operator[](int n)=0;

//返回指定索引的token值

virtual int find(const std::string& tok)=0;

//查找指定的token，返回在集合中的第一个出现位置的索引，若不存在则返回

//token集合的大小

virtual void push\_back(const std::string& tok)=0;

//向集合中装入token（一般是实现类自己使用）

virtual bool remove(const std::string& tok)=0;

//从集合中删除第一次出现的指定的token，若token不存在则返回false

virtual void toLower()=0;

//将集合中所有的token都变成小写形式（基本不会使用到）

virtual void trimFront()=0;

//删除token中在最前面的换行符

virtual void clear()=0;

//清除集合中的tokens（一般是实现类自己使用，目前不曾被调用过）

virtual std::string show()=0;

//将tokens组合成字符串返回（忽略其中所以换行符token）

virtual ~ITokCollection() {};

//析构函数（没有什么好说的）

};

//灰色背景的方法会删除或修改集合中的token，即会影响到实际读入的token信息因此要小心使用

#ifndef SEMIEXPRESSION\_H

#define SEMIEXPRESSION\_H

Module Operations:

==================

This module defines a SemiExp class. Its instances build a sequence

of tokens obtained from the Tokenizer module for analysis of code.

SemiExpressions are lists of tokens that terminate on one of the tokens:

'{', '}', ';', or '\n' when its line starts with '#'

Public Interface:

=================

Toker t; // create tokenizer instance

SemiExp se(&t); // create instance and attach

if(se.get()) // collect a semiExpression

std::cout << se.showSemiExp().c\_str(); // show it

int n = se.length(); // number of tokens in se

std::string tok = se[2] = "a change"; // indexed read and write

if(se.find("aTok") < se.length()) // search for a token

std::cout << "found aTok";

se.clear(); // remove all tokens

#include <string>

#include <iostream>

#include <fstream>

#include <vector>

#include "ITokCollection.h"

#include "tokenizer.h"

class SemiExp : public ITokCollection

{

public:

SemiExp(Toker\* pTokr);

~SemiExp();

bool get();

int length();

std::string& operator[](int n);

int find(const std::string& tok);

void push\_back(const std::string& tok);

bool remove(const std::string& tok);

void toLower();

void trimFront();

void clear();

std::string show();

void verbose(bool v = true);

void makeCommentSemiExp(bool commentIsSE = true);

bool isComment(const std::string& tok);

private:

Toker\* pToker;

std::vector<std::string> toks;

bool isTerminator(std::string tok);

bool Verbose;

bool commentIsSemiExp;

};

inline int SemiExp::length() { return (int)toks.size(); }

inline void SemiExp::verbose(bool v) { Verbose = v; }

inline void SemiExp::makeCommentSemiExp(bool commentIsSE)

{

commentIsSemiExp = commentIsSE;

}

inline void SemiExp::clear() { toks.clear(); }

inline void SemiExp::push\_back(const std::string& tok)

{

toks.push\_back(tok);

}

#endif

#include <algorithm>

#include "SemiExpression.h"

//----< construct SemiExp instance >---------------------------

SemiExp::SemiExp(Toker\* pTokr) : pToker(pTokr), Verbose(false),

commentIsSemiExp(false)

{

}

//----< destructor >-------------------------------------------

SemiExp::~SemiExp()

{

}

//----< is this a terminating token? >-------------------------

bool SemiExp::isTerminator(std::string tok)

{

if(tok.length() == 0)

return false;

if(tok == "{" || tok == "}" || tok == ";")

return true;

if(tok == "\n")

{

int n = find("#");

if(n < length())

return true;

}

if(commentIsSemiExp && isComment(tok))

return true;

return false;

}

//\_

//----< collect a semi-expression >----------------------------

bool SemiExp::get()

{

toks.clear();

std::string tok;

do

{

if(pToker->isFileEnd())

{

if(length() > 0)

return true;

return false;

}

tok = pToker->getTok();

if(Verbose)

{

if(tok != "\n")

std::cout << "\n--tok=" << tok << std::endl;

else

std::cout << "\n--tok=newline\n";

}

toks.push\_back(tok);

} while(!isTerminator(tok));

return true;

}

//----< is this token a comment? >--------­­­­---------------------

bool SemiExp::isComment(const std::string& tok)

{

if(tok.length() < 2) return false;

if(tok[0] != '/') return false;

if(tok[1] == '/' || tok[1] == '\*') return true;

return false;

}

//----< index operator >---------------------------------------

std::string& SemiExp::operator[](int n)

{

if(n < 0 || toks.size() <= (size\_t)n)

throw std::exception("SemiExp index out of range");

return toks[n];

}

//----< collect semi-expression as space-seperated string >----

std::string SemiExp::show()

{

if(toks.size() == 0)

return "";

std::string temp(" ");

for(size\_t i=0; i<toks.size(); ++i)

if(toks[i] != "\n")

temp.append(" ").append(toks[i]);

return temp;

}

//\_

//----< is tok found in semi-expression? >---------------------

int SemiExp::find(const std::string& tok)

{

for(int i=0; i<length(); ++i)

if(tok == toks[i])

return i;

return length();

}

//----< remove tok if found in semi-expression >---------------

bool SemiExp::remove(const std::string& tok)

{

std::vector<std::string>::iterator it;

it = std::find(toks.begin(),toks.end(),tok);

if(it != toks.end())

{

toks.erase(it);

return true;

}

return false;

}

//----< remove leading newlines >------------------------------

void SemiExp::trimFront()

{

while(toks.size() > 0 && toks[0] == "\n")

remove("\n");

}

//----< make tokens lowercase >--------------------------------

void SemiExp::toLower()

{

for(int i=0; i<length(); ++i)

{

for(size\_t j=0; j<(\*this)[i].length(); ++j)

{

(\*this)[i][j] = tolower((\*this)[i][j]);

}

}

}