**Анализатор исходных файлов С++**

Для решения этой задачи был создан класс Analyzer и вспомогательная структура FileInfo:

struct FileInfo

{

string relativePath; // путь к файлу указанный в директиве #include

fs::path path; // полный путь к файлу

bool exist; // существует ли файл

// сравнение объектов по полному пути, если возможно (если оба файла не существуют, fs::equivalent кидает исключение)

bool operator==(const FileInfo &includeInfo) const {

try {

return fs::equivalent(includeInfo.path, path);

} catch (fs::filesystem\_error &) {

return includeInfo.path == path;

}

}

};

// вычисление хеш значения для FileInfo, чтобы хранить эту информацию в unordered\_map для быстрого поиска

template <>

struct hash<FileInfo>

{

size\_t operator()(const FileInfo &fInfo) const {

std::hash<string> hasher;

return hasher(fInfo.path.string());

}

};

class Analyzer

{

vector<fs::path> externalIncludes; // пути для поиска #include <>

string sourcesPath; // путь к каталогу с исходниками, задается в командной строке

// отображение: Файл -> информафия о файлах, указанный в #include файла-ключа

unordered\_map<FileInfo, vector<FileInfo>> dependencies;

// Подсчет кол-ва включений фалов. Эта информация сортируется после добавления всех элементов,

// т.к. на момент добавления не известно окончательное значение для элемента

unordered\_map<FileInfo, int> counters;

// все найденные cpp файлы

vector<FileInfo> cppFiles;

// Рекурсивный обход всех включений заданного файла. Если файл уже просматривался ранее, он будет проигнорирован.

// В задаче не говорилось об эффективности решения, поэтому для извлечения инклюдов применяются регулярные выражения.

void extractIncludesFromFile(struct FileInfo &);

public:

// внешние каталоги поиска заголовочных файлов (параметры -I командой строки)

void setExternalIncludes(const vector<string> &);

// каталог исходников

void setSourcesPath(string path) { sourcesPath = path; }

// основной метод построения дерева включений и подсчета их частоты

void buildTreeAndFrequence();

// вывод в консоль дерева включений

void printTree() const;

// вывод в консоль частоты включений

void printFrequence() const;

// очистка предыдущего построения

void reset();

};

**Пример запуска:**

D:\>Analyzer.exe -I C:\test21 C:\test12

c.cpp

..hh1.h

....h1.h

......h2.h

........hhh1.h

..........g.h

............gg.h (!)

............ggg.h (!)

......h3.h

..h3.h

main.cpp

..h1.h

....h2.h

......hhh1.h

........g.h

..........gg.h (!)

..........ggg.h (!)

....h3.h

..h3.h

h3.h 3

h1.h 2

g.h 1

gg.h 1

ggg.h 1

h2.h 1

hh1.h 1

hhh1.h 1

Для следующего набора данных:

D:\>dir /S /B C:\test12

C:\test12\dir1

C:\test12\h1.h

C:\test12\h2.h

C:\test12\h3.h

C:\test12\main.cpp

C:\test12\dir1\c.cpp

C:\test12\dir1\hh1.h

D:\>dir /S /B C:\test21

g.h

hhh1.h

D:\>type C:\test12\\*

C:\test12\h1.h

#include "h2.h"

#include "h3.h"

C:\test12\h2.h

test

//#include "h3.h"

#include <hhh1.h>

C:\test12\h3.h

//#include "h1.h"

C:\test12\main.cpp

#include "h1.h"

/\*

#include "h2.h"

\*/

int main()

{}

#include "h3.h"

D:\>type C:\test12\dir1\\*

C:\test12\dir1\c.cpp

#include "hh1.h"

#include "../h3.h"

C:\test12\dir1\hh1.h

#include "../h1.h"

D:\>type C:\test21\\*

C:\test21\g.h

// #include "hhhhhh.h"

/\* #include "uuu.h" \*/

/\* //#include "uuuyyy.h" \*/

#include "gg.h"

/\*

\* // #include "ga.h" \*/

#include "ggg.h"

C:\test21\hhh1.h

#include "g.h"