|  |  |
| --- | --- |
| Программа для построения пути согласования документа в компании на основе иерархии подразделений | |
| Внутренняя спецификация | |
| Студент | Николаенко С. Д. |
| Преподаватель | преп. каф. ПОАС Матюшечкин Д. С. |
| Сдано |  |
| Лабораторная работа №2 |  |

1. Общие сведения

Наименование программы – «DocumentApprovalPath».

Для функционирования программы необходима операционная система Windows 8 или выше.

Программа написана на языке C++.

1. Описание логической структуры

Разработка программы осуществляется на основании задания на курсовой проект по дисциплине «Качество и надежность программного обеспечения», выданного доцентом кафедры ПОАС Сычевым О. А. 10 февраля 2022 года.

* 1. Алгоритм программы

Считать входные данные из входных файлов.

Проверить корректность входных данных.

Построить путь согласования документа на основе иерархии сотрудников.

Вывод обратного списка руководителей подразделений, которые находятся в иерархии компании выше, чем сотрудник, от имени которого просматривается список.

Записать выходные данные в выходной файл.

2.2 Декомпозиция программы

Выделенные подпрограммы (функции) описаны в приложении А.

Основные типы и структуры данных программы описаны в приложении Б.

Иерархия вызовов подпрограммы представлена в приложении В.

Диаграмма потоков данных представлена в приложении Г.

Приложение А

Описание функций

Функция: int main(const int argc, char\*\* argv)

Обеспечить считывание из файла, вывод в консоль ошибок, если они есть, запись ответа в выходной файл, вызов главной функции, решающей задачу.

Алгоритм работы функции (псевдокод):

Выдать ошибку, если входной xml-файл и txt-файл невозможно открыть

Считать данные из файлов

Проверить корректность входных данных

Если есть ошибки

{

Распечатать их в консоль ошибок

Завершить работу программы

}

Составить список всех начальников

Вывести обратный список в выходной файл...

Приложение Б

Описание структур данных

struct Person {

string Name;//ФИО сотрудника

int id = 0;//id сотрудника

};

vector <int> headers; //вектор для хранения начальников

vector <bool> habsent; //вектор для хранения отсутствующих

vector <Person> person; //вектор для хранения сотрудников

/\*!Функция сопоставления id и ФИО

\*

\* \param[in] id-сотрудника(текущее)

\* return значение ФИО соответствующее его id

\*/

string MatchingIdAndName(int id\_for\_name)

{

try{

Для всех сотрудников

{

Если текущее id соответсвует id\_имени человека

{

Возвращаем значение его ФИО

}

}

throw 1;

}

catch (int e)

{

if (e == 1) {

Вывод сообщения "Invalid employee id. Perhaps it doesn't exist."

Возвращаем пустую строку

}

}

}

/\*Функция Сохранение результатов в файл\*/

bool SavingResultsToAFile()

{

Создаем объект класса ofstream

Открываем файл для добавления данных

Для всех сотрудников(в обратном порядке)

{

Если сотрудник присутствует на рабочем месте, записываем его в выходной файл

{

Записываем сотрудника в выходной файл

Возвращаем true

}

Иначе {Возвращаем false}

}

Закрываем файл

}

/\*!Функция поиска начальников искомого сотрудника

\* \param [in] node - указатель на элемент в XML-файле

\* \param [in] findid - id сотрудника

\*/

void SearchSuperiorsOfTheDesiredEmployee(XMLElement\* node, int findid)

{

Перебираем все элементы с именем Department

{

Преобразование значения элемента из char в string

Если название элемента есть "Department"

{

Объявление переменной id

Получение значения "head" у "Department"(начальник отдела)

добавление id

подтверждение добавления

}

Если название элемента есть "Person"

{

Объявление переменной id, absent findAbsent = 1

Создание объекта Person

Получение значения "id"

Присвоение значений id и Name

Добавление полученного сотрудника

Получение значение "absent"

Если текущего сотрудника нет на рабочем месте, переход к следующему сотруднику

{

Объявление i=0

Пока не прошли всех начальников

{

Если соответсвие id произошло

{

Присвоение значения false текущему элементу вектору habsent

}

Игнорирование данного сотрудника и переход к следующему

}

}

Если соответсвие id из txt-файла найдено

{

Если сотрудник отсутствует

{

Вывод сообщения об ошибке "The person is absent"

}

Иначе

{

Если id=1, следовательно-это глава фирмы,

{Вывод сообщения "No solution"; }

Сохранение результатов в файл(вызов функции SavingResultsToAFile())

}

}

}

Рекурсивный вызов: для поиска искомого сотрудника в дочерних элементах

Если значение элемента Department

{

Удаление значений начальников и их значений

}

Переход к следующему элементу

}

}

/\*Проверка данных из txt-файла

\* \param[in] – строка

\* return true/false

\*/

bool isdigit(string s)

{

Для всех символов

{

Если текущий элемент меньше 48 или больше 57

{

Возвращаем false

}

}

Возвращаем true

}

/\*!Получение id из txt-файла

\*\param [in] file\_xml – название файл

\* return id(число)/сообщение об ошибке

\*/

int GettingIdFromTxtFile(const char\* file\_txt)

{

Строка для записи

Открытие файла в режиме чтения

Если открытие файла прошло корректно, то

try

{

Если файл работает

{

Цикл для чтения значений из файла; выполнение цикла прервется,

Когда достигнем конца файла, в этом случае F.eof() вернет истину.

while (!file.eof())

{

Чтение очередного значения из потока F в переменную a

Если число

{

Возвращаем id

}

Иначе

throw 2;

}

Закрываем файл

}

else

{

throw 1;

}

}

catch (int e)

{

if (e == 1) { cout << "Invalid input file specified. The file may not exist" << endl; }

if (e == 2) { cout << "Data entered incorrectly. The input string contains a set of different characters" << endl; }

}

}

/\*!Работа с xml файлом

\* \param[in] file\_xml – название файла

\* return true/false

\*/

bool WorkingWithXMLFile(const char\* file\_xml)

{

Создаем объект документа XML и передаем ему имя файла, из которого будем читать

Если файл не открыт {

Вывод сообщения "Invalid input file specified. The file may not exist"

Возвращеам false

}

Иначе

{

Объявление названия txt-файла

Получение значение id из txt-файла (вызов функции GettingIdFromTxtFile)

Находим первый элемент с именем Department

Вызов функция поиска начальников искомого сотрудника

Проверка корректности файла(логический уровень)-(вызов функции checkHeaders)

Возвращаем true

}

}

Функция применяется в модульных тестах

/\*Прооверка на существование файла\*/

int exists(const char\* fname)

{

FILE\* file;

if ((file = fopen(fname, "r")))

{

fclose(file);

return 1;

}

return 0;

}

Приложение В

Иерархия вызовов подпрограммы представлена в приложении

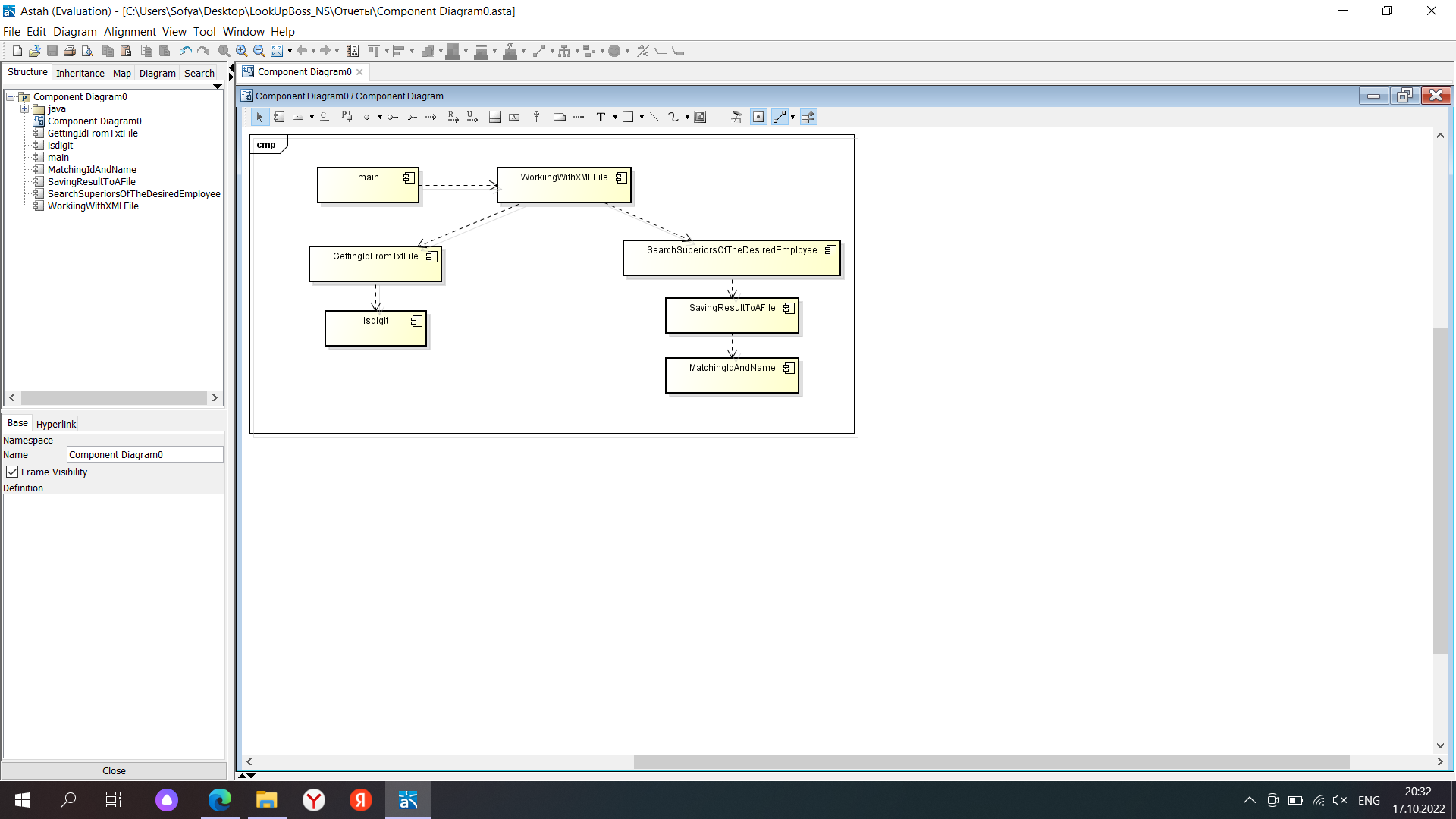


Рис. 1. Иерархия вызовов функций

Приложение Г

Диаграмма потоков данных представлена в приложении

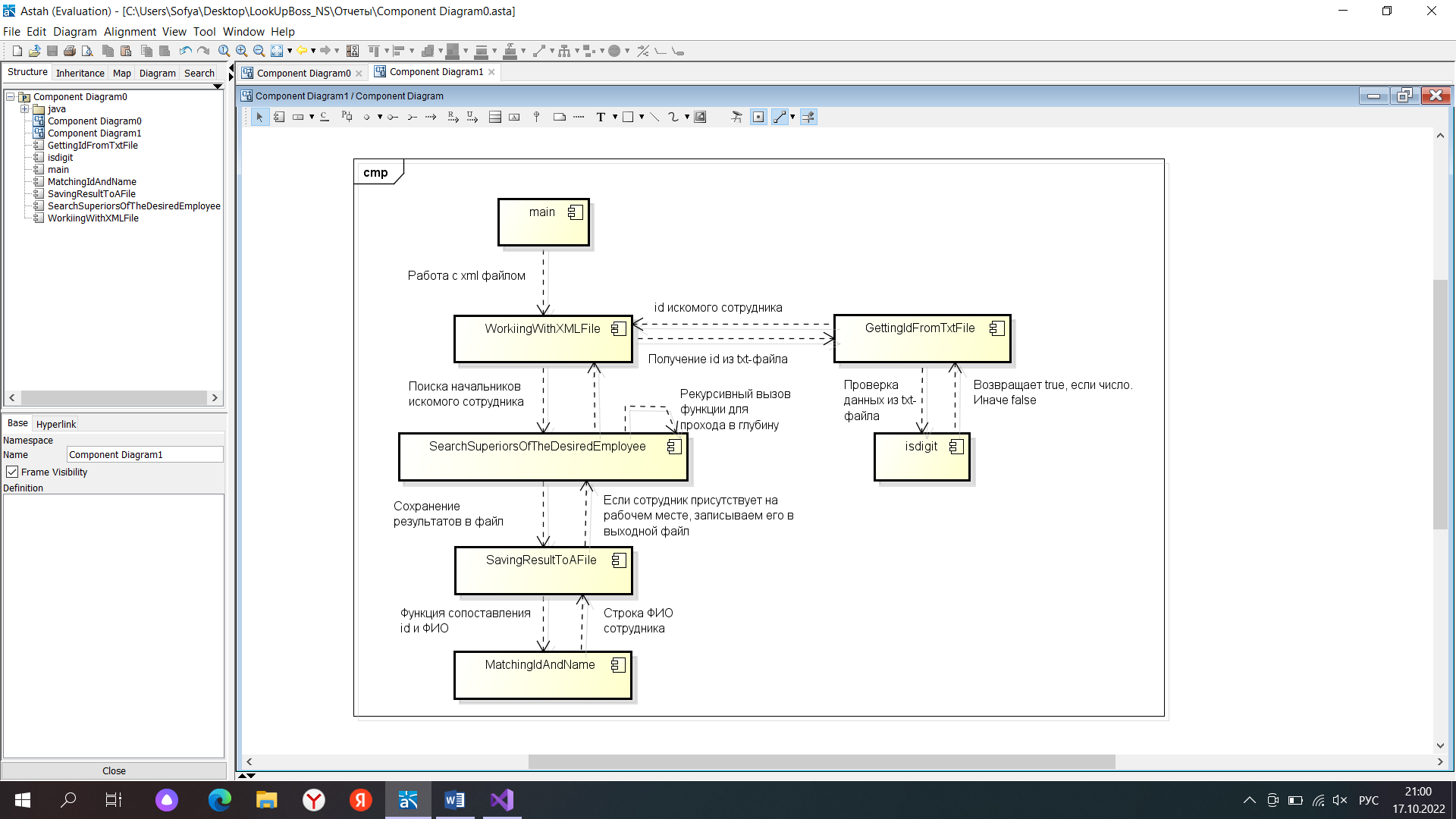


Рис. 2. Диаграмма потоков данных