**Детали реализации**

Программа реализована на языке С в соответствии с ТЗ. В качестве компилятора использовался MinGW\_3.20 . Проект собирался в операционной системе Windows 7 x64. Для сборки проекта используется файл “build.cmd”. Исходный код разделён на 2 папки: h\ и src\.

Проект подключает внешние файлы входящие в стандартную библиотеку С99, а также библиотеку <windows.h> для отладочного использования консоли(при сборке релиза <windows.h> не подключается).

В файлы “config.h” и “config.c” вынесены основные настройки проекта. Среди них :

* Режим сборки проекта Debug/Release
* Все программные сообщения, указанные в ТЗ
* Константы: длина входной строки, длинна имени файла

Большая часть ТЗ реализована с помощью Автоматов состояний (look img. "State Machine of Prg1"). Аргументы в пользу решения использовать автоматы состояний:

1. В соответствии с ТЗ языком реализации должен был быть "С". Вполне очевидно, что ТЗ нужно было разбивать на модули, так как других способов сопровождать проекты в предложенном языке нет.
2. Запутанное ТЗ было преобразовано в диаграмму состояний, благодаря чему удалось найти немало ошибок.
3. Полученная архитектура получилась довольно гибкой. Для изменения поведения программы необходимо всего то изменить изолированное состояние и связанные с ним переходы.

Как собрать проект для Windows:

1. Скачать компилятор GCC <http://gcc.gnu.org/>
2. Добавить новый системный путь C:\MinGW\bin для возможности писать команду gcc в любой папке
3. Заматипить диск S: файлом disk.cmd
4. Запусти build.cmd
5. Исполняемый файл готов

**Подробнее о реализации автоматов состояний**

В любой момент программа находится в одном из возможных состояний

## enum StCodes

## {

## ST\_GET\_ARGS ,

## ST\_CHOOSE\_STRING ,

## . . .

## };

Каждому состоянию отвечает одна функция ( с таким же названием ).

## int get\_args (...);

## int choose\_string (...);

## . . .

Соответствие между функциями и состояниями задаётся в массиве actionCell.

## struct ActionCell actionCell[] =

## {

## { ST\_GET\_ARGS , get\_args },

## { ST\_CHOOSE\_STRING , choose\_string },

## . . .

## };

Таким образом зная текущее состояние машины мы всегда знаем какую функцию нужно вызвать. Кроме того все функции, отвечающие автомату возвращают RetCodes

## enum RetCodes

## {

## RET\_FILE ,

## RET\_TRUE ,

## RET\_FALSE ,

## . . .

## };

А программе определена таблица связей между старым и новым состояниями через код возврата - стрелочки на диаграмме Вот её вид :

## struct Transition state\_transitions[] =

## {

## { ST\_GET\_ARGS , RET\_CONSOLE , ST\_CHOOSE\_STRING },

## { ST\_GET\_ARGS , RET\_FILE , ST\_CHECK\_ARGS },

## . . .

## }

Таким образом зная текущее состояние **ST\_GET\_ARGS** и получив код возврата **RET\_FALS** после вызова функции **get\_args(...)** по таблице связей **state\_transitions** можно легко определить новое состояние программы.

Для стартового и конечного состояния есть синонимы:

## #define ENTRY\_STATE ST\_GET\_ARGS

## #define EXIT\_STATE ST\_EXIT\_CODE