

Содержание

1. Введение	5
2. Специальная часть.....	7
2.1. Анализ исходных требований для разрабатываемой библиотеки обработки входных параметров и систематизации выходных данных	7
2.2. Разработка соглашений о вызовах функций библиотеки	11
2.2.1. Разработка соглашений о вызовах функций обработки ошибок работы библиотеки.....	11
2.2.2. Разработка соглашения о вызове функции инициализации библиотеки.....	12
2.2.3. Разработка соглашений о вызовах функций получения входных параметров программ тестирования.....	14
2.2.4. Разработка соглашений о вызовах функций обработки выходных данных программ тестирования	18
2.3. Реализация функций разрабатываемой библиотеки	21
2.4. Прототипирование среды исполнения подпрограмм библиотеки	30
3. Технологическая часть	37
3.1. Профилирование разрабатываемого программного обеспечения	37
3.2. Анализ производительности библиотеки интерфейсов.....	42
3.3. Отладка и тестирование разрабатываемой библиотеки.....	46
4. Охрана труда и окружающей среды. Разработка мероприятий по обеспечению благоприятных санитарно-гигиенических условий труда инженера.....	58
Введение.....	58
4.1. Анализ условий труда инженера-программиста.....	58
4.1.1. Характеристика условий труда инженера-программиста	58
4.1.2. Анализ освещения, микроклимата и визуальных параметров устройства отображения информации.	61
4.2. Разработка мероприятий по уменьшению отрицательного воздействия производственных факторов	71

4.2.1. Микроклимат	71
4.2.2. Визуальные параметры средств отображения информации	71
Вывод по теме	74
5. Экономическая часть. Обоснование экономической эффективности разработки библиотеки функций унификации процессов обработки входных параметров и систематизации выходных данных	76
5.1. Обоснование экономической эффективности разработки программного обеспечения “Библиотека функций унификации процессов обработки входных параметров и систематизации выходных данных в средствах тестирования и диагностики программных средств и оборудования.”	76
5.2. Определение трудоёмкости создания программного продукта	78
5.3. Календарное планирование.	80
5.4. Определение затрат на создание программного продукта	83
5.5. Оценка экономической эффективности.....	85
6. Заключение	90
Список литературы	92
Приложение 1	93
Приложение 2.....	125
Приложение 3	138

1. Введение

В ходе комплексного тестирования программных средств возникает необходимость интерпретации результатов множества тестов, написанных по различным правилам и для различных целей. Для решения задачи автоматизации запуска, сбора информации и интерпретации результатов тестирования необходимо привести интерфейсную часть всех тестирующих программ к единообразному виду, позволяющему с наименьшими затратами решать поставленную задачу. Для данных целей предлагается использовать единую библиотеку (далее именуемую библиотекой *libtio*) с небольшим прикладным программным интерфейсом (*API*), исключающую возможность административного взаимонепонимания при реализации правил для обработки входных параметров и систематизации выходных данных в средствах тестирования и диагностики.

В рамках дипломной работы будет проведен анализ требований, а также разработаны соглашения об использовании основных функций данной библиотеки. После этого будут представлены блок-схемы некоторых функций.

В технологической части с целью повышения производительности будет проведена профилировка и, по возможности, после анализа наиболее узких мест, будут внесены изменения в отдельные функции для оптимизации работы библиотеки в целом. Следующим пунктом будет проверка работоспособности функций библиотеки в различных условиях и с различными параметрами для выявления неявных ошибок. В случае если при проверке возникнут ошибки, будет проведена их локализация и устранение.

В части охраны труда и окружающей среды будет представлен анализ помещения, в котором проводилась разработка библиотеки *libtio*, а именно: анализ естественного освещения, анализ и расчет искусственного освещения, анализ микроклимата.

Экономическая часть будет отведена под обоснование экономической эффективности разработки с подсчетом годового экономического эффекта и сроков окупаемости проекта.

2. Специальная часть

2.1. Анализ исходных требований для разрабатываемой библиотеки обработки входных параметров и систематизации выходных данных

Так как в исходных требованиях к разрабатываемой библиотеке указана необходимость совместимости с архитектурами *SPARC V8*, *SPARC V9*, *i386*, *x86_64*, то следует обеспечить независимость данного программного продукта от архитектуры процессора. Это достигается путем использования при разработке языка программирования высокого уровня (Си), обеспечивающего создание кросс-платформенного приложения.

Си является стандартизированным языком программирования. Это дает гарантию того, что однажды написанная, программа может быть использована на разных архитектурах. Ответственность за адаптацию высокоуровневых конструкций языка программирования к особенностям конкретной архитектуры берет на себя компилятор с этого языка для данной конкретной архитектуры. В данной работе использовался компилятор *GNU Compiler Collection* (обычно используется сокращение *gcc*), поддерживающий большое количество архитектур, в том числе и требуемые. К тому же данный компилятор обеспечивает возможность кросс-компиляции, то есть создание исполняемого файла (в данном случае – библиотеки) для платформы отличной от той, на которой запускается компиляция.

Для осуществления кросс-компиляции в *gcc* обычно применяется команда *<архитектура>-gcc* (например: *SPARC-gcc*). Существует также и второй вариант: использование команды *gcc* с ключом *-b <архитектура>*.

Для каждой архитектуры в *gcc* имеется свой список опций. В частности для компиляции под процессоры архитектуры *SPARC V8* необходимо указать ключ *-mcpu=v8*, а для 9 версии – ключ *-mcpu=v9*.

Основное различие 8 и 9 версий архитектуры *SPARC* заключается в том, что в 9 версии добавлена поддержка 64-битной адресации. Также все целочисленные регистры *SPARC V8* были расширены из 32-битных в 64-битные. Кроме того появились новые инструкции для работы с 64-битными операндами.

Аналогично для семейств архитектур *i386* и *x86-64* у *gcc* имеется свой набор опций.

Если заранее известно на процессоре какой архитектуры будет использоваться данная библиотека, то для оптимизации её работы можно при кросс-компиляции использовать ключ *-mtune=<cpu-type>*, где *<cpu-type>* – это тип конкретной архитектуры процессора.

Основной отличительной особенностью архитектуры *x86-64* от *i386* является поддержка 64-битных регистров общего назначения, 64-битных арифметических и логических операций над целыми числами и 64-битных виртуальных адресов.

Процессоры архитектуры *x86-64* поддерживают два режима работы: *Long mode* («длинный» режим) и *Legacy mode* («наследственный», режим совместимости с 32-битным *x86*), которые можно явно задать при компиляции, используя ключи *-m64* и *-m32* соответственно. Следовательно, если нужно чтобы библиотека запускалась и на архитектуре *i386* и на архитектуре *x86-64* нужно использовать ключ *-m32*.

Важным отличием *SPARC* архитектур от архитектур *i386* и *x86-64* является порядок записи байт. *SPARC* использует *big-endian* (порядок байт от старшего к младшему), а *i386* и *x86_64* – *little-endian* (от младшего к старшему). Запись многобайтового числа из памяти компьютера в файл или передача по сети требует соблюдения соглашений о том, какой из байтов является старшим, а какой младшим, так как прямая запись ячеек памяти приводит к возможным проблемам при переносе приложения с платформы на платформу. Для разрабатываемой библиотеки был принят порядок байт от

старшего к младшему (*big-endian*), так как он является стандартным для протокола *TCP/IP* (протокола управления передачей по сети).

В исходных требованиях указано, что разрабатываемая библиотека будет использоваться в операционных системах использующих стандарты *POSIX*.

POSIX (англ. *Portable Operating System Interface for Unix* — Переносимый интерфейс операционных систем *Unix*) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой. Стандарт создан для обеспечения совместимости различных *UNIX*-подобных операционных систем и переносимости прикладных программ на уровне исходного кода.

С точки зрения программиста-разработчика следование стандарту *POSIX* заключается в использовании заголовочных файлов и системных вызовов языка Си, которые должны быть предоставлены соответствующей стандарту системой.

Библиотека должна быть написана на языке высокого уровня «Си» в соответствии со спецификацией *C99*.

C99 — современный стандарт языка программирования Си. Определен в ISO/IEC 9899:1999. Является развитием стандарта *C89*.

В *C99* было добавлено несколько новых возможностей, а также удалены лишние.

Добавленные средства:

- Встраиваемые функции (объявленные с ключевым словом *inline*)
- Место, в котором возможно объявление переменных, больше не ограничено глобальной областью видимости и началом составного оператора (блока)
- Несколько новых типов данных, включая *long long int*, дополнительные расширенные целые типы, явный логический тип данных, а также комплексный тип (*complex*) для представления комплексных чисел

- Массивы переменной длины (*variable-length arrays*)
- Поддержка однострочных комментариев, начинающихся с *//*, как в C++
- Новые библиотечные функции, как, например, *snprintf*
- Новые заголовочные файлы, такие как *stdbool.h* и *inttypes.h*
- Типовые математические функции (*tgmath.h*)
- Составные константы
- Поддержка вариативных макросов (макросов переменной аргументности)

Некоторые удаленные средства:

Самым заметным "излишеством", удаленным при создании C99, было правило "неявного *int*". В C89 во многих случаях, когда не было явного указания типа данных, подразумевался тип *int*. А в C99 такое не допускается. Также удалено неявное объявление функций. В C89, если функция перед использованием не объявлялась, то подразумевалось неявное объявление. А в C99 такое не поддерживается. Если программа должна быть совместима с C99, то из-за двух этих изменений, возможно, придется подправить код.

C99 является большей частью обратно совместимым с C90, но вместе с тем в некоторых случаях является более жестким. В частности, объявление без указания типа больше не подразумевает неявное задание типа *int*. Комитет по стандартизации языка Си решил, что для компиляторов будет более важным определять пропуск по невнимательности указания типа, чем «тихая» обработка старого кода, полагавшаяся на неявное указание *int*.

GCC и другие компиляторы языка Си поддерживают многие нововведения стандарта C99. Тем не менее, ощущается недостаточная поддержка стандарта со стороны крупных производителей средств разработки, таких как *Microsoft* и *Borland*, которые сосредоточились, в основном, на языке C++, так как C++ обеспечивает функциональность, схожую с предоставляемой нововведениями стандарта.

Согласно *Sun Microsystems*, *Sun Studio* полностью поддерживает стандарт *C99*.

Интерпретатор языка Си *Ch* поддерживает основные особенности *C99* и свободно доступен в версиях для *Windows*, *Linux*, *Mac OS X*, *Solaris*, *QNX* и *FreeBSD*.

Другие компиляторы с полной или частичной поддержкой стандарта *C99*

Digital Mars

Intel C Compiler

Oracle Solaris Studio

VPF

LCC

Pelles C

Open Watcom C

2.2.Разработка соглашений о вызовах функций библиотеки

2.2.1. Разработка соглашений о вызовах функций обработки ошибок работы библиотеки

Все функции библиотеки, которые предназначены для пользователей программистов и являющиеся экспортируемыми должны возвращать значение.

Функции, возвращающие указатель на некий тип данных, в случае ошибки должны возвращать нулевой указатель (NULL).

Функции, возвращающие числовое значение некого типа данных, при аварийном завершении возвращают максимально допустимое значение своего возвращаемого типа. Функции, возвращаемый параметр которых имеет символьный тип, также относятся к возвращающим числовое значение. Стоит отметить, что в таком случае возвращаемый параметр является беззнаковым.

Код ошибки для последней вызванной функции библиотеки можно получить, используя функцию *tioGetError()*, возвращаемым значением которой и будет код ошибки.

При возникновении ситуации, из-за которой не может продолжаться нормальная работа функций библиотеки, необходимо вызвать функцию

```
int tioDie ( int status,  
            const char* buff,  
            )
```

Вследствие её работы ресурсы памяти, занятые библиотекой, будут освобождены.

Аргументы:

status - статус завершения приложения (TOFAIL, TOTESTNOTSTART)

msg - сообщение размещаемое в потоке ошибок

Сигнал TOFAIL означает, что программа тестирования завершилась с неудовлетворительным результатом.

Если передать сигнал TOTESTNOTSTART, это будет означать, что ошибка произошла ещё на стадии инициализации библиотеки.

2.2.2. Разработка соглашения о вызове функции инициализации библиотеки

Функцией инициализации библиотеки является функция *tioInit*. До её вызова запрещается вызов любой другой функции библиотеки, за исключением функции *tioGetVersion*. В задачи *tioInit* входит выделение памяти и задание начальных значений для переменных, массивов и структур, без которых невозможно использовать другие функции разрабатываемой библиотеки. Кроме того происходит разбор входных параметров для программы тестирования. Функция принимает как "длинные", так и "короткие" параметры. Все параметры, ключи которых содержат больше одного символа, за исключением символа двоеточия на конце, являются

длинными, все прочие называются короткими. Ключ из одного символа также может быть длинным.

Прототип функции `tioInit`:

```
int tioInit ( const char* version,
              const char* help,
              const tio_param _param[],
              int argc,
              char *argv[]
            )
```

Как видно из прототипа, функция принимает 5 параметров:

1. *version* – версия теста, для которого инициализируется библиотека;
2. *help* – короткое описание назначения теста;
3. *_param[]* – список параметров, принимаемых приложением, и тех ключей для параметров, что используются в данном приложении. Признаком конца списка параметров является структура `tio_param`, у которой все поля имеют значение `NULL`. Поля структуры `tio_param` приводятся ниже;
4. *argc* – количество аргументов командной строки;
5. *argv[]* – список аргументов командной строки;

`tio_param` представляет собой структуру вида:

```
typedef struct _tio_param
{
    char *key;
    char *name;
    char* description;
} tio_param;
```

Где *key* – ключ, используемый при вызове из командной строки, *name* – имя параметра, используемое при взаимодействии приложения с библиотекой, а *description* – короткое пояснение для каких целей используется параметр.

В качестве имени параметра разрешается использовать любую последовательность символов, состоящую из букв, цифр, символа подчеркивания и знака минус, длиной до 126 символов.

В качестве ключа разрешено использовать последовательность символов, начинающуюся с буквы или с цифры. В теле последовательности могут содержаться буквы, цифры и знак минус. Также строка не должна совпадать со словами «*help*», «*version*».

Если при запуске программы тестирования используется ключ *--help*, то вместо выполнения теста в стандартный поток вывода будет представлена информация о списке ключей, доступных при вызове.

Если при запуске использовать ключ *--version*, то в стандартный поток вывода будет представлена информация о версии теста.

2.2.3. Разработка соглашений о вызовах функций получения входных параметров программ тестирования

Для того чтобы автоматизировать получение параметров командной строки, предлагается использовать семейство функций *tioGet** и *tioGetDef**, где вместо знака «*» должна быть подставлена одна из следующих букв, означающих какого типа будет возвращаемое значение:

- L – long
- D – double
- C – char
- S – char* (string)

Коды ошибок в результате работы функций приведены ниже (Таблица 2.1).

TENOPAR	Параметр не зарегистрирован при инициализации библиотеки
TEINCTYPE	Параметр не может быть приведен к запрошенному типу
TENOTSET	Параметр не передан при вызове приложения.

TENES	Размер буфера недостаточно велик для помещения параметра
TEFAILL	Отказ по непонятным причинам

Таблица 2.1

Функции `tioGetS` и `tioGetDefS`

```
int tioGetS (  const char* name,
               char* buff,
               size_t buff_len
             )
```

Функция получения параметра командной строки в форме последовательности символов. *name* – указатель на имя параметра, значение которого необходимо получить. *buff* – указатель на адрес памяти, куда функция поместит значение искомого параметра в виде последовательности символов. *buff_len* – переменная, содержащая значение максимальной длины строки.

Возвращает 0 в случае успешного выполнения. В противном случае возвращаемое значение примет вид кода ошибки из таблицы 2.1. При возникновении любой из ошибок функция *tioGetS* заносит в *buff* нулевой символ.

```
int tioGetDefS (  const char* name,
                  const char* default,
                  char* buff,
                  size_t buff_len
                )
```

Функция получения параметра командной строки в форме последовательности символов. *name* – указатель на имя параметра, значение которого необходимо получить. *default* – значение параметра, связанного с именем *name* по умолчанию. *buff* – указатель на адрес памяти, куда функция поместит значение искомого параметра в виде последовательности символов. *buff_len* – переменная, содержащая значение максимальной длины строки.

В случае если значение, связанное с именем *name* получить не удалось, то в буфер *buff* заносится значение параметра *default*.

Возвращает 0 в случае успешного выполнения. В противном случае возвращаемое значение примет вид кода ошибки из таблицы 2.1. При возникновении любой из ошибок функция *tioGetDefS* заносит в *buff* нулевой символ.

Функции *tioGetL* и *tioGetDefL*

```
long tioGetL ( const char* name )
```

Функция возвращает значение параметра командной строки, связанного с именем *name*. Значение должно быть расположено в промежутке от минимально допустимого для типа *long* до предшествующего максимально допустимому значению для типа *long* (от `LONG_MIN` до `LONG_MAX-1`). В случае, если такого параметра нет, или значения параметра не находятся в указанном промежутке или не могут быть приведены к типу данных *long*, возвращается максимально допустимое значение для типа *long*. Код ошибки в этом случае может быть получен с помощью функции *tioGetError()*.

Возможные ошибки: `TENOTSET` и `TEINCTYPE`.

```
long tioGetDefL ( const char* name,  
                  const long default  
                  )
```

Функция возвращает значение параметра командной строки, связанного с именем *name*. Значение должно быть расположено в промежутке от минимально допустимого для типа *long* до предшествующего максимально допустимому значению для типа *long* (от `LONG_MIN` до `LONG_MAX-1`). В случае, если такого параметра нет, или значения параметра не находятся в указанном промежутке или не могут быть приведены к типу данных *long*, возвращается значение по умолчанию, присвоенное при вызове функции параметру *default*. Код ошибки в этом случае может быть получен с помощью функции *tioGetError()*.

Возможные ошибки: `TENOTSET` и `TEINCTYPE`.

Функции *tioGetC* и *tioGetDefC*

```
unsigned char tioGetC ( const char* name )
```

Функция возвращает значение символа, переданного из командной строки и связанного с именем *name*. В случае, если возвращаемое значение не может быть приведено к типу *unsigned char*, возвращаемое значение будет иметь вид максимально допустимого числа для этого типа данных.

Код ошибки может быть получен при помощи вызова *tioGetError*.
Возможные ошибки: TENOTSET и TEINCTYPE.

```
unsigned char tioGetDefC ( const char* name,  
                          const unsigned char default  
                          )
```

В случае успешного завершения функции, возвращаемое значение будет равно значению, переданному из командной строки и связанному с именем *name*. В случае, если получить значение, связанное с именем *name* не удалось, то возвращаемое значение будет взято из параметра *default*.

Код ошибки может быть получен при помощи вызова *tioGetError*.
Возможные ошибки: TENOTSET и TEINCTYPE.

Функции *tioGetD* И *tioGetDefD*

```
double tioGetD ( const char* name )
```

Функция возвращает число с плавающей запятой, переданное в программу с параметром *name*. Значение числа может быть любым допустимым для переменной в формате *double*, за исключением значения максимально допустимого для данного типа данных. В случае неуспешного выполнения, возвращаемое значение принимает вид максимально возможного значения для типа *double*.

Код ошибки может быть получен при помощи вызова *tioGetError*.
Возможные ошибки: TENOTSET и TEINCTYPE.

```
double tioGetDefD ( const char* name,  
                   const double default  
                   )
```

Функция получения параметра, связанного с именем *name*, в форме числа с плавающей точкой, со значением по умолчанию. Значение числа может быть любым допустимым для переменной в формате *double*, за исключением значения максимально допустимого для данного типа данных. В случае, если

по каким либо причинам получить значение параметра по его имени не удалось, функция возвращает значение по умолчанию, определенное в параметре *default*.

Код ошибки может быть получен при помощи вызова *tioGetError*.
Возможные ошибки: TENOTSET и TEINCTYPE.

2.2.4. Разработка соглашений о вызовах функций обработки выходных данных программ тестирования

Функции вывода делятся на два типа: функции строчного вывода и функции табличного вывода.

Функции табличного вывода.

Для предоставления данных в табличной форме определено следующее семейство функций:

```
void* tioTableBegin ( const char* format, ... );  
void* tioTableRecord ( void *td, ... );  
int tioTableEnd( void *td ).
```

Первая функция предназначена для инициализации таблицы, а также для задания количества столбцов и их заголовков. В том числе, в функции *tioTableBegin* происходит определение для каждого столбца типа данных, которые он будет содержать в себе.

Параметр *format* содержит строку символов, которая содержит в себе список имен столбцов таблицы, разделенных знаком амперсанд (&). В случае если знак амперсанд является частью имени столбца, необходимо использовать последовательность символов, состоящих из двух амперсандов подряд. Далее в прототипе функции идет переменный список параметров, количество параметров которого зависит от количества столбцов таблицы. Значения этих параметров определяют типы значений соответствующих столбцов. В случае успеха возвращаемое значение является указателем на таблицу.

Функция *tioTableRecord* предназначена для добавления новой строки в таблицу, передаваемую с параметром *td*. Далее идет переменный список параметров, в каждом из которых содержатся значение соответствующей ячейки таблицы. В случае успеха, возвращаемым значением, также как и в предыдущей функции, является указатель на таблицу.

Функция *tioTableEnd* является функцией, которая выводит в виде таблицы сформированные данные, полученные от вызовов предыдущих функций семейства *tioTable*. В том случае, если какие либо значения не могут быть представлены в одной строке ячейки таблицы, функция добавляет столько строк в таблицу, сколько нужно для полного представления данного значения.

Между вызовами функций *tioTable** разрешен вызов любых других функций библиотеки.

Функции строчного вывода

Все функции для форматирования строки вывода используют формат, широко применяемый в системных функциях. Формат задаётся последовательностью символов, начинающихся с символа % и продолжающихся символами из приведенной далее таблицы:

Символ	Описание типа
c	Символ (char)
d i	Целое число в десятичной форма (long)
e	Число с мантиссой для чисел с плавающей запятой (double)
f	Число с плавающей точкой (double)
o	Целое число в восьмеричном представлении (long)
s	Строка завешающаяся нулем (char*)
x	Беззнаковое шеснадцатиричное представления (long)
X	Беззнаковое шеснадцатеричное представления с буквами в верхнем регистре (long)

Таблица 2.2

Для вывода символа % используется последовательность %%.

```
int tioPrint(const char * message)
```

Выводит префикс «(I):» и строку с, на которую указывает параметр message, в стандартный поток вывода.

```
int tioPrintf(const char* template, ... )
```

Выводит префикс «(I):» и форматируемую строку в стандартный поток вывода.

Префикс «(I):» говорит о том, что строка имеет информационный характер. Обычно сообщения с таким префиксом выводятся для пояснения чего-либо при работе программы.

```
int tioWarning( const char * message)
```

Выводит префикс «(WW):» и строку, на которую указывает параметр message в поток ошибок.

```
int tioWarningF(const char* template, ... )
```

Выводит префикс «(WW):» и форматируемую строку в поток ошибок.

Префикс «(WW):» говорит о том, что строка является предупреждением. Обычно сообщения с таким префиксом выводятся, чтобы предупредить о каком либо событии, которое может повлечь за собой ошибки.

```
int tioError( const char * message)
```

Выводит префикс «(EE):» и строку, на которую указывает параметр message в поток ошибок.

```
int tioErrorF(const char* template, ... )
```

Выводит префикс «(EE):» и форматируемую строку в поток ошибок.

Префикс «(EE):» говорит о том, что строка является сообщением об ошибке. Обычно сообщения с таким префиксом выводятся для того, чтобы сообщить какого рода произошла ошибка, с целью облегчения поиска места ее возникновения.

Следующие две функции выводят сообщения, только если программа, использующая библиотеку, была запущена с ключом `--tio-debug`

```
int tioDebug( const char * message)
```

Выводит префикс «(DD):» и строку, на которую указывает параметр `message`, в стандартный поток вывода.

```
int tioDebugF(const char* template, ... )
```

Выводит префикс «(DD):» и форматируемую строку в стандартный поток вывода.

Префикс «(DD):» говорит о том, что строка является отладочной информацией. Обычно сообщения с таким префиксом выводятся на этапе реализации кода программы.

2.3. Реализация функций разрабатываемой библиотеки

Набор кода функций библиотеки *libtio* проводился при помощи средств текстового редактора *Vim*.

Vim — это свободный режимный текстовый редактор. Одна из главных особенностей редактора — применение двух основных, вручную переключаемых, режимов ввода: командного (позволяет использовать клавиши клавиатуры не для печатанья, а для различных команд) и текстового (режим непосредственного редактирования текста, аналогичный большинству «обычных» редакторов).

Эффективная работа с редактором требует предварительного обучения, так как интерфейс этого редактора нельзя назвать интуитивно понятным.

Vim обладает возможностью, позволяющей разбивать рабочую поверхность редактора на множество окон как по вертикали, так и по горизонтали. В нем присутствует: поддержка *Unicode* символов, неограниченная глубина отмены (*undo*) и возврата (*redo*) действий, режим

сравнения двух файлов, подсветка синтаксиса, автоматическое определение величины отступа для каждой строки в зависимости от языка программирования, автоматическое продолжение команд, слов, строк целиком и имён файлов, сворачивание (*folding*) текста для лучшего обзора. поддержка цикла разработки «редактирование — компиляция — исправление» программ.

При реализации функций библиотеки *libtio* использовалась распределённая система управления версиями файлов *Git*.

Все настройки *Git* хранятся в текстовых файлах конфигурации. Такая реализация делает эту систему легко портируемой на любую платформу и даёт возможность легко интегрировать *Git* в другие системы (в частности, создавать графические *git*-клиенты с любым желаемым интерфейсом). Репозиторий *Git* представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Следует отметить, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «*git*» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий *git*, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети.

При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда `commit`).

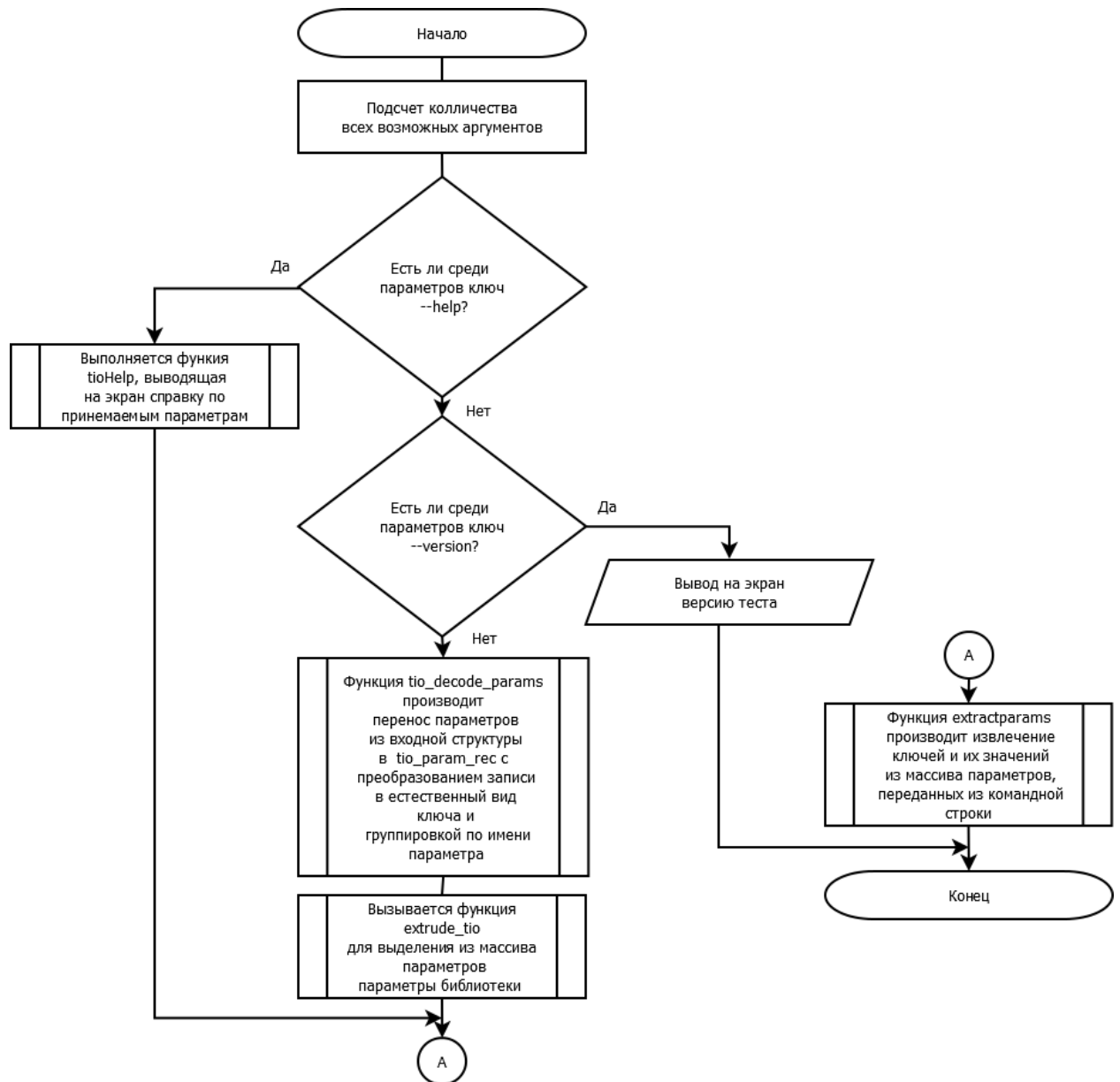
Git изначально идеологически ориентирована на работу с изменениями, а не с файлами, «единицей обработки» для нее является набор изменений, или патч.

Преимуществами *git* перед другими системами контроля версиями:

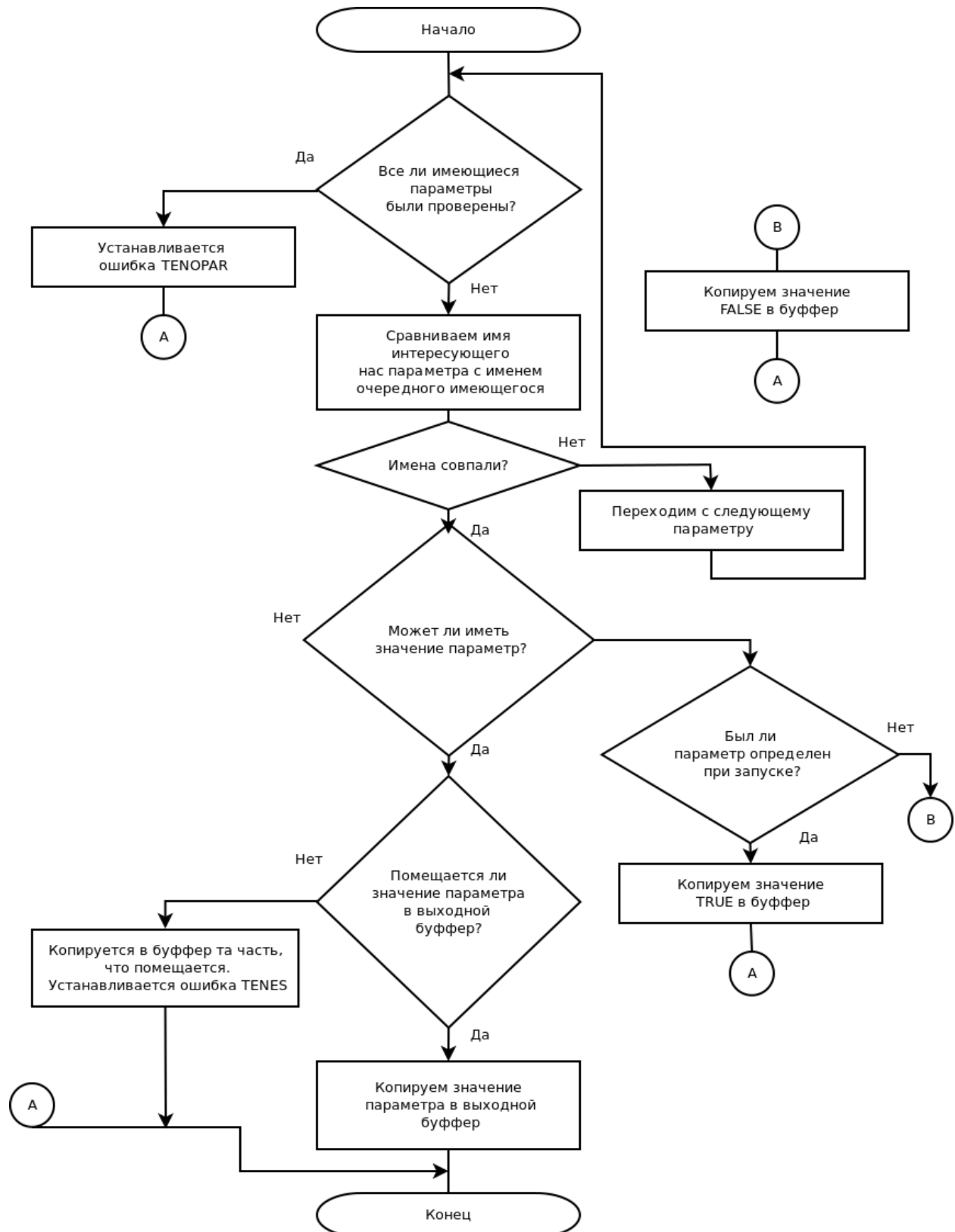
- Высокая производительность.
- Продуманная система команд, позволяющая удобно встраивать *git* в скрипты.
- Репозитории *git* могут распространяться и обновляться общесистемными файловыми утилитами архивации и обновления благодаря тому, что фиксации изменений и синхронизации не меняют существующие файлы с данными, а только добавляют новые (за исключением некоторых служебных файлов, которые могут быть автоматически обновлены с помощью имеющихся в составе системы утилит). Для раздачи репозитория по сети достаточно любого веб-сервера.

Блок-схемы реализаций некоторых функций.

Функция tioInit().



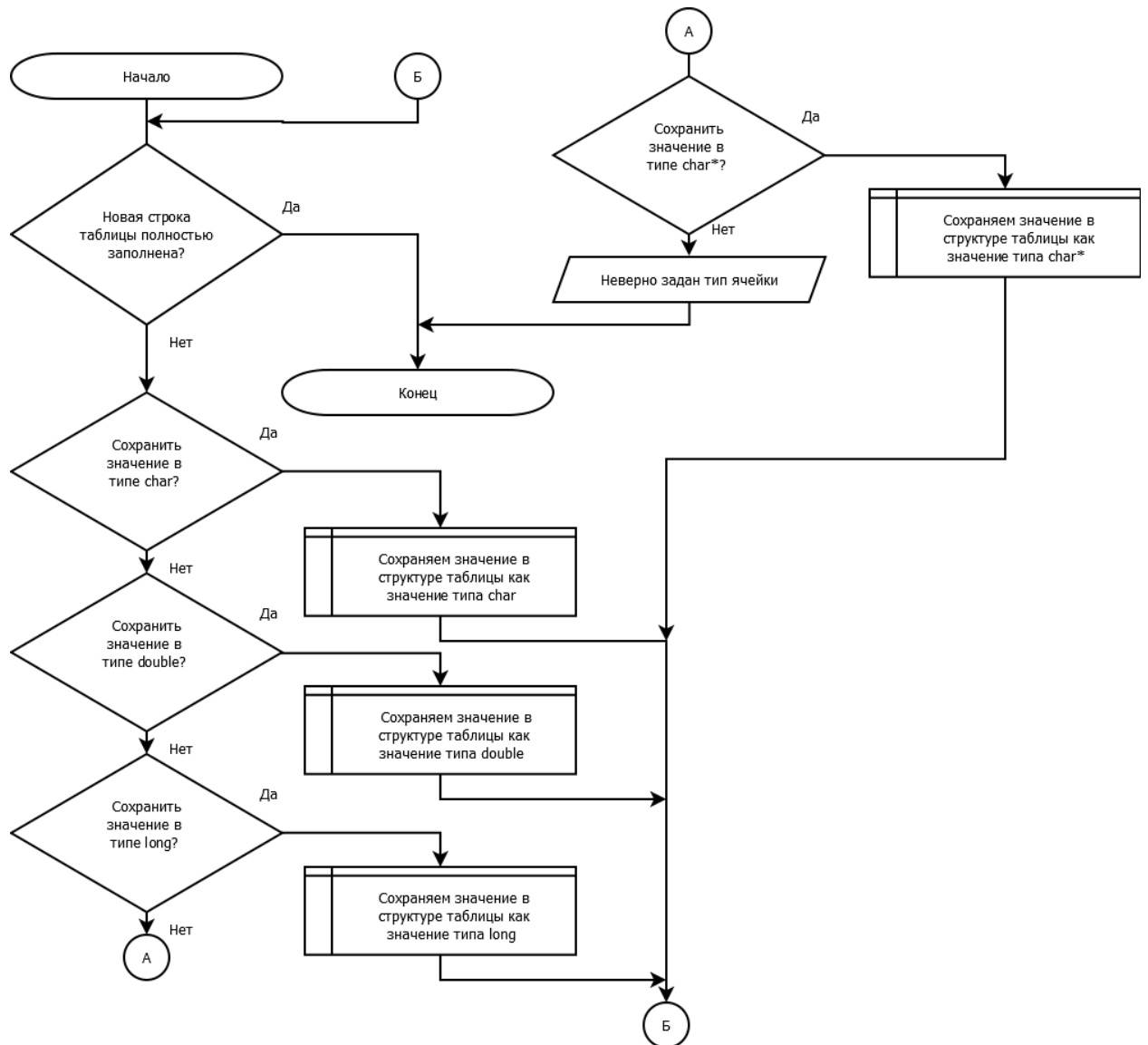
Функция tioGetS()



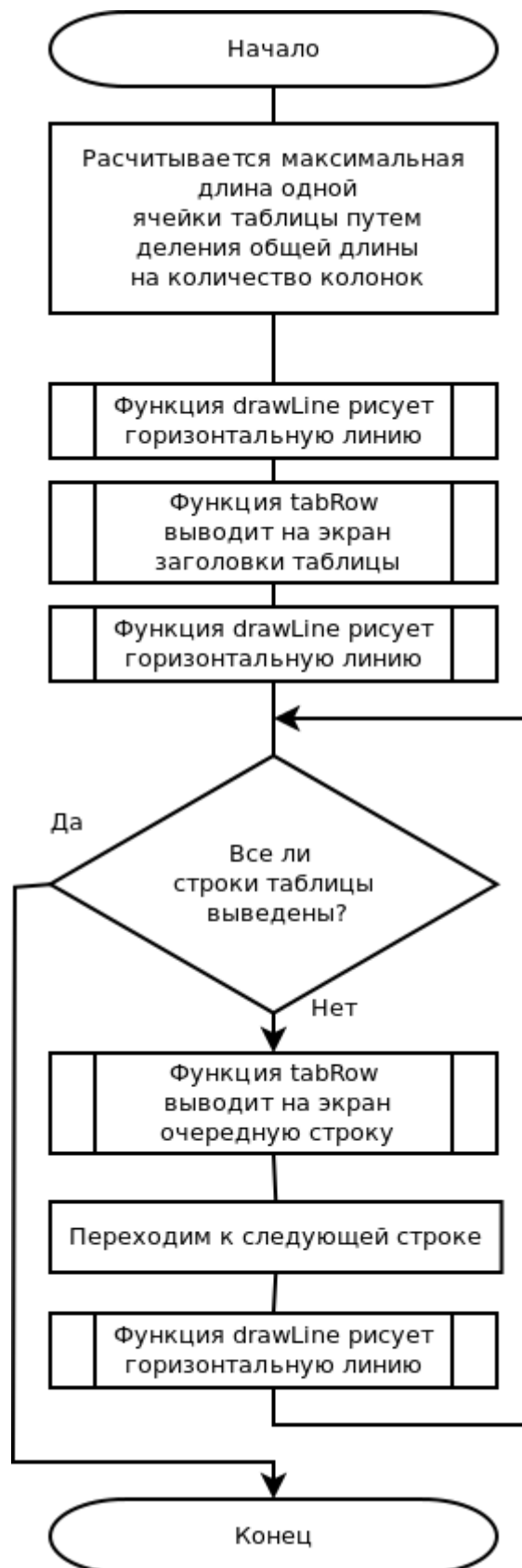
Функция tioTableBegin()



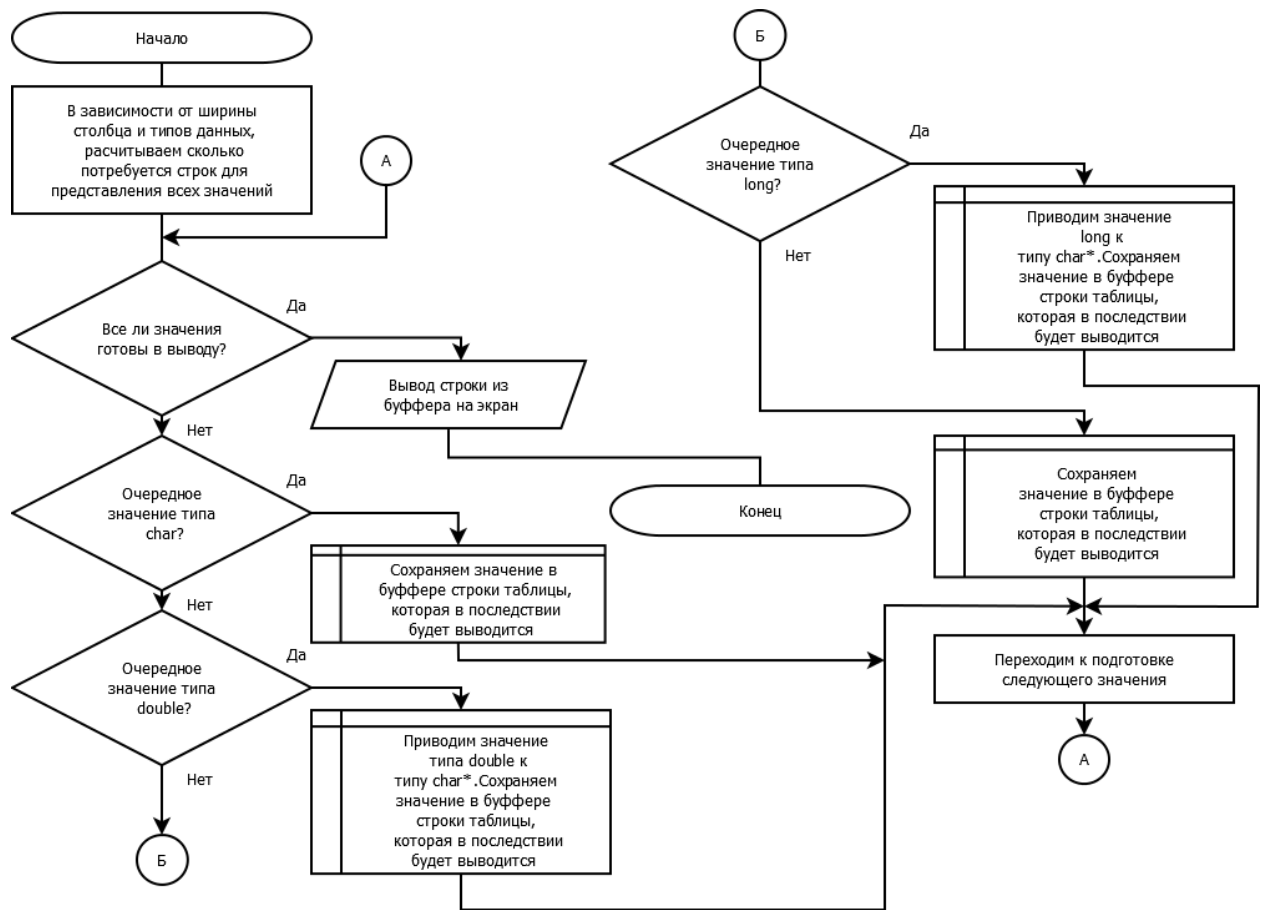
Функция tioTableRecord()



Функция tioTableEnd()



Функция tabRow()



2.4.Прототипирование среды исполнения подпрограмм библиотеки

Базовые возможности библиотеки рассмотрим на примере программы, которая тестирует функцию подсчета корней квадратного уравнения.

Есть функция, решающая квадратное уравнение.

Задача: протестировать являются ли корни, полученные на выходе функции, верными для уравнения заданного вида.

Для тестирования возьмем уравнение вида $x^2 + 2x - 3 = 0$. Значит параметр «a» равен 1, параметр «b» равен 2 и параметр «c» равен -3. Известно, что корнями данного уравнения являются 1 и -3. Для того чтобы убедиться в корректности работы функции решения квадратных уравнений, напомним тест, использующий функции разработанной библиотеки.

Параметры a, b, c, первый эталонный корень и второй эталонный корень передаются при вызове теста (см. Приложение 2) как параметры командной строки. Согласно данному уравнению, строка, запускаящая тест, должна выглядеть так:

```
./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2= -3
```

Программа теста считывает входные параметры, запускает тестируемую функцию с параметрами a, b, c. Получившиеся результаты работы функции решения квадратного уравнения выводит на экран вместе с эталонными значениями *root1* и *root2*.

Выполнение теста показано на рис. 2.1.

```
nwcfang@nf-lrti:~/current-task/prototypes/quadratic-equation$ ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
Тест написал: Гусев Михаил
Короткое описание теста:
Тестирование функции решения квадратного уравнения.
[RUN]: Запуск ./quadratic-equation
```

Name	Value
Argument A	1
Argument B	2
Argument C	-3

Сравнение эталонных и возвращаемых функцией корней

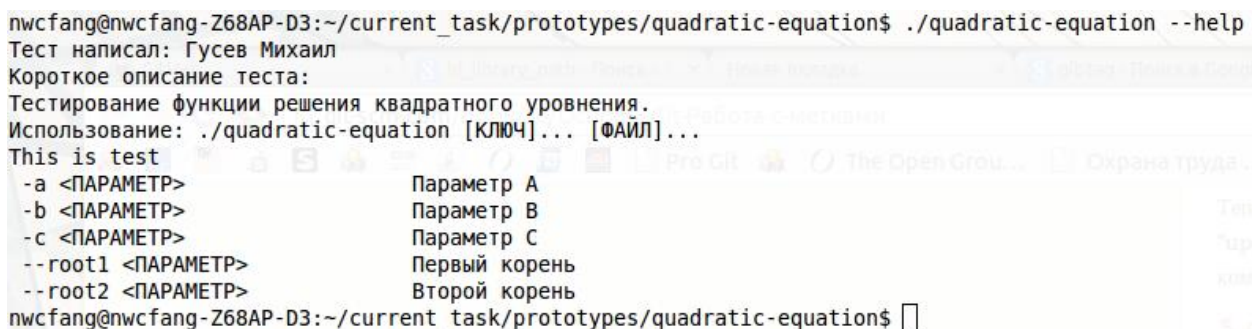
Roots etalon	Roots from function
1	1
-3	-3

Рис. 2.1

Для разбора параметров командной строки, а также инициализации разработанной библиотеки использовалась функция *tioInit*. После вызова этой функции можно использовать функции библиотеки семейства *tioGet* для доступа к интересующим нас параметрам командной строки.

Наглядное представление выходных данных обеспечивается функциями семейства *tioTable*, позволяющих рисовать динамическую таблицу, в которой можно изменять заголовки и количество колонок, а также количество строк и тип данных в каждой ячейке строки.

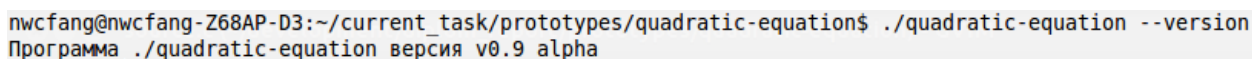
С помощью ключа *--help*, переданного при вызове тестирующей программы, использующей библиотеку *libtio*, вместо выполнения теста на экран выводится список аргументов, которые можно передать из командной строки (рис. 2.2).



```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation --help
Тест написал: Гусев Михаил
Короткое описание теста:
Тестирование функции решения квадратного уравнения.
Использование: ./quadratic-equation [КЛЮЧ]... [ФАЙЛ]...
This is test
-a <ПАРАМЕТР>          Параметр А
-b <ПАРАМЕТР>          Параметр В
-c <ПАРАМЕТР>          Параметр С
--root1 <ПАРАМЕТР>     Первый корень
--root2 <ПАРАМЕТР>     Второй корень
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 2.2

При использовании ключа *--version* после команды, запускающей исполняемый файл программы тестирования, будет выведена справка о версии запускаемого теста (рис. 2.3).



```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation --version
Программа ./quadratic-equation версия v0.9 alpha
```

Рис. 2.3

Теперь рассмотрим программу, тестирующую работоспособность COM-порта.

Программа может работать в трех режимах:

- режим «Клиент»;
- режим «Сервер»;
- режим «Клиент/Сервер».

Если выбран режим «Клиент», то программа работает по следующему алгоритму:

В течении двадцати секунд ожидает сообщение от программы «Сервер» о готовности к передаче данных. Если по истечению данного периода сообщение не получено, то тест завершается провалом. В случае если сообщение о готовности «Сервера» пришло, отправляется сообщение о готовности принимать данные. После чего принимаем пакеты.

Если выбран режим «Сервер», то программа работает так:

Вначале отправляется сообщение, что «Сервер» готов к передаче данных. Получив, ответ от «Клиента», что он готов к передаче, «Сервер» начинает передавать пакеты.

Режим «Клиент/Сервер» отличается от предыдущих тем, что создается процесс потомок, который берет на себя роль «Сервера», а родитель будет работать как «Клиент».

В качестве входных параметров для тестирующей программы принимаются следующие ключи:

- «-D» – ключ имеет числовое значение. Продолжительность передачи пакетов;
- «-m» - ключ имеет числовое значение. Скорость передачи пакетов;
- «-s» - ключ имеет числовое значение. Размер передаваемого пакета;
- «-d» - программа будет работать в режиме Сервера, то есть отправлять пакеты Клиенту;

- «-l» - программа будет работать в режиме Клиента, то есть принимать пакеты от Сервера;
- «-L» - программа работает в режиме Клиент/Сервер, то есть пакеты будут отправляться и приниматься на одной и той же ЭВМ.

Эта программа была написана без использования библиотеки *libtio*. Метод обработки входных параметров описывался в отдельном файле и выглядел так:

```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

#include "config.h"

Configuration config = {
    115200, // speed
    -1,     // device fd
    "/dev/ttyUSB0", // device path
    1000,    // minimum transferred data count
    CLIENTMODE, // mode of test
    0, // duration (nowtime is unused)
    1 // work mode
};

static int
calculate_configuration(Configuration *cfg)
{
    if (!cfg)
        return EINVAL;

    if (cfg->duration)
    {
        cfg->sendPacksLength = (cfg->duration * cfg-
>portSpeed / 8);
        cfg->duration = 0;
    }
    return 0;
}
```

```
}
```

```
int
write_configuration(Configuration *cfg, char **argv, int
argc)
{
    int opt;
    int already_typed = 0;

    if (!cfg || !argv ||  argc < 1)
        return EINVAL;

    while (-1 != (opt = getopt(argc, argv, "D:m:s:dlLh")))
    {
        switch(opt)
        {
            case 'D':
                cfg->duration = atol(optarg);
                if (cfg->duration <= 0)
                    return EAGAIN;
                break;
            case 'm':
                cfg->portSpeed = atol(optarg);
                if (cfg->portSpeed <= 0)
                    return EAGAIN;
                break;
            case 's':
                cfg->sendPacksLength = atol(optarg);
                if (cfg->sendPacksLength <= 0)
                    return EAGAIN;
                break;
            case 'd':
                if (already_typed)
                    return EAGAIN;
                cfg->serverClientMode = SERVERMODE;
                already_typed = 1;
                break;
            case 'l':
                if (already_typed)
                    return EAGAIN;
                cfg->serverClientMode = CLIENTMODE;
                already_typed = 1;
                break;
            case 'L':
                if (already_typed)
                    return EAGAIN;
```



```

        cfg->serverClientMode = CLIENTSERVERMODE;
        already_typed = 1;
        break;
    default:
        return EAGAIN;
    }
}
calculate_configuration(cfg);

if (optind < argc)
    strcpy(config.DeviceName, argv[optind]);

return 0;
}

```

Использование библиотеки `libtio`, а в частности функции `tioInit` позволяет сократить данный файл до вида:

```

#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

#include <tio.h>

#include "config.h"

Configuration config = {
    -1,          //device fd
    1000,        //minimum transferred data count
    1            //work mode
};
static int
calculate_configuration(Configuration *cfg)
{
    if (!cfg)
        return EINVAL;

    if( tioGetDefL( "DURATION", 0 ) )
        cfg->sendPacksLength = ( tioGetL( "DURATION" ),
tioGetDefL( "PORTSPEED", 115200 ) / 8);
    return 0;
}
int

```

```

write_configuration(Configuration *cfg )
{
    cfg->sendPacksLength = tioGetDefL( "SENDPACKSLENGTH",
1000 );
    calculate_configuration(cfg);
    return 0;
}

```

Причем, чем больше ассортимент параметров командной строки, тем больше преимущество по времени у программиста, использующего библиотеку *libtio*, пред программистом, пишущим код разбора входных параметров самостоятельно.

Стандартный поток вывода после выполнения тестирующей программы в режиме «Клиент/Сервер» показан на рис. 2.4

```

nwcfang@nwcfang-Z68AP-D3:~/development/rti/rs232test$ sudo ./_test_COM -L /dev/ttyS0
[sudo] password for nwcfang:
[RUN]: Заняв ./_test_COM
(DD)[client.c@228]: Starting server wait
(DD)[client.c@53]: Current 1354726090: stat at 1354726090: elapsed: 0
(DD)[server.c@162]: Starting server process
(DD)[client.c@77]{readBuffer}->{Found receive buffer: 1FFF9AA9}
(DD)[client.c@236]: Starting transfere
(DD)[client.c@157]: Ready to read status wrote
(DD)[client.c@167]: Message decoded: left space 924
(DD)[client.c@167]: Message decoded: left space 848
(DD)[client.c@167]: Message decoded: left space 772
(DD)[client.c@167]: Message decoded: left space 696
(DD)[client.c@167]: Message decoded: left space 620
(DD)[client.c@167]: Message decoded: left space 544
(DD)[client.c@167]: Message decoded: left space 468
(DD)[client.c@167]: Message decoded: left space 392
(DD)[client.c@167]: Message decoded: left space 316
(DD)[client.c@167]: Message decoded: left space 240
(DD)[client.c@167]: Message decoded: left space 164
(DD)[client.c@167]: Message decoded: left space 88
(DD)[client.c@167]: Message decoded: left space 12
(DD)[client.c@167]: Message decoded: left space -64
(DD)[client.c@211]: Client decode finished
client process - OK
[PASS]: ./_test_COM : Тест пройден успешно

```

Рис. 2.4

3. Технологическая часть

3.1. Профилирование разрабатываемого программного обеспечения

Профилирование – это сбор характеристик программного обеспечения, таких как данные о продолжительности и частоте выполнения каждой из функций программы, поиск утечек памяти, и прочих ошибок, связанных с неправильной работой с областями памяти – чтением или записью за пределами выделенных регионов и тому подобное.

В качестве профилировщика используется *Valgrind*, предоставляющий множество инструментов для поиска узких мест в коде. Инструмент по умолчанию, а также наиболее используемый – *Memcheck*.

Memcheck вставляет дополнительный код в программное обеспечение, который отслеживает любые манипуляции и перемещения данных в памяти. Более того, *Memcheck* заменяет стандартное выделение памяти языка Си собственной реализацией, которая помимо прочего включает в себя защиту памяти (*memory guards*) вокруг всех выделенных блоков. Данная возможность позволяет *Memcheck* обнаруживать ошибки несоответствия (*off-by-one errors*), при которых программа считывает или записывает вне выделенного блока памяти. Проблемы, которые может обнаруживать *Memcheck* и предупреждать о них, включают в себя:

- чтение или запись по неправильным адресам памяти — за границами выделенных блоков памяти и т.п.;
- использование не инициализированных значений, в том числе и для переменных, выделяемых в стеке;
- ошибки освобождения памяти, например, когда блок памяти уже был освобожден в другом месте;
- передача некорректных параметров системным вызовам, например указание неправильных указателей для операций чтения из буфера, указанного пользователем;

- пересечение границ блоков памяти при использовании операций копирования/перемещения данных между двумя блоками памяти.

Однако, использование *Memcheck* способствует снижению производительности в 5-12 раз, а также использованию большего объёма памяти, из чего следует, что использование *Memcheck*-реализации выделения памяти не приветствуется и должно использоваться только при профилировании программного обеспечения.

На рис. 3.1 показан вывод профилировщика *Valgrind*, использовавшего инструмент *Memcheck*. В качестве объекта для исследования используется программа тестирования функции решения квадратного уравнения (см. п. 2.4).

```

nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ valgrind --tool=memcheck --leak-check=full
--show-reachable=yes --track-origins=yes ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
==3409== Memcheck, a memory error detector
==3409== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==3409== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==3409== Command: ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
==3409== Parent PID: 2454
==3409==
==3409== HEAP SUMMARY:
==3409==     in use at exit: 1,712 bytes in 16 blocks
==3409==   total heap usage: 234 allocs, 218 frees, 14,102 bytes allocated
==3409==
==3409== 16 bytes in 1 blocks are still reachable in loss record 1 of 6
==3409==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3409==    by 0x4009C6: main (source.c:24)
==3409==
==3409== 16 bytes in 1 blocks are still reachable in loss record 2 of 6
==3409==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3409==    by 0x4009D4: main (source.c:25)
==3409==
==3409== 240 bytes in 2 blocks are definitely lost in loss record 3 of 6
==3409==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3409==    by 0x4E39581: tabRow (tioTableBegin.c:371)
==3409==    by 0x4E39253: tioTableEnd (tioTableBegin.c:277)
==3409==    by 0x400B72: main (source.c:66)
==3409==
==3409== 240 bytes in 2 blocks are definitely lost in loss record 4 of 6
==3409==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3409==    by 0x4E39581: tabRow (tioTableBegin.c:371)
==3409==    by 0x4E39253: tioTableEnd (tioTableBegin.c:277)
==3409==    by 0x400AE8: main (source.c:60)
==3409==
==3409== 480 bytes in 4 blocks are definitely lost in loss record 5 of 6
==3409==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3409==    by 0x4E39581: tabRow (tioTableBegin.c:371)
==3409==    by 0x4E39284: tioTableEnd (tioTableBegin.c:286)
==3409==    by 0x400B72: main (source.c:66)
==3409==
==3409== 720 bytes in 6 blocks are definitely lost in loss record 6 of 6
==3409==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3409==    by 0x4E39581: tabRow (tioTableBegin.c:371)
==3409==    by 0x4E39284: tioTableEnd (tioTableBegin.c:286)
==3409==    by 0x400AE8: main (source.c:60)
==3409==
==3409== LEAK SUMMARY:
==3409==    definitely lost: 1,680 bytes in 14 blocks
==3409==    indirectly lost: 0 bytes in 0 blocks
==3409==    possibly lost: 0 bytes in 0 blocks
==3409==    still reachable: 32 bytes in 2 blocks
==3409==    suppressed: 0 bytes in 0 blocks
==3409==
==3409== For counts of detected and suppressed errors, rerun with: -v
==3409== ERROR SUMMARY: 18 errors from 6 contexts (suppressed: 2 from 2)

```

Рис. 3.1

В данной проверке поведение *Memcheck* настраивается следующими ключами:

--leak-check=full – функция обнаружения утечек памяти, будет выводить не только сводную информацию, но и информацию о месте, в котором происходит утечка памяти.

--show-reachable=yes – будет показана информация о блоках не освобожденной памяти, указатель на которые все ещё не потерян.

--track-origins=yes – при задании этого ключа будет выводиться информация о неинициализированных переменных и об их происхождении.

Следующий инструмент, который использовался в дипломной работе для профилирования основных функций библиотеки, - *Callgrind*.

Callgrind анализирует вызовы функций. На основе полученных данных при использовании этого инструмента можно построить дерево вызовов функций, и соответственно, проанализировать узкие места в работе программы. По умолчанию он собирает данные о количестве выполненных инструкций, зависимостях между вызывающей и вызываемой функциями и количество вызовов конкретных функций.

Для визуализации данных, полученных в результате работы *Callgrind*, использовалась программа *Kcachegrind*.

На рис. 3.2 показана схема вызовов функций для программы тестирования функции решения квадратного уравнения (см. п. 2.4). Данная схема не содержит в себе все вызываемые функции, так как нет смысла анализировать функции, выполнение которых занимает незначительную часть процессорного времени.

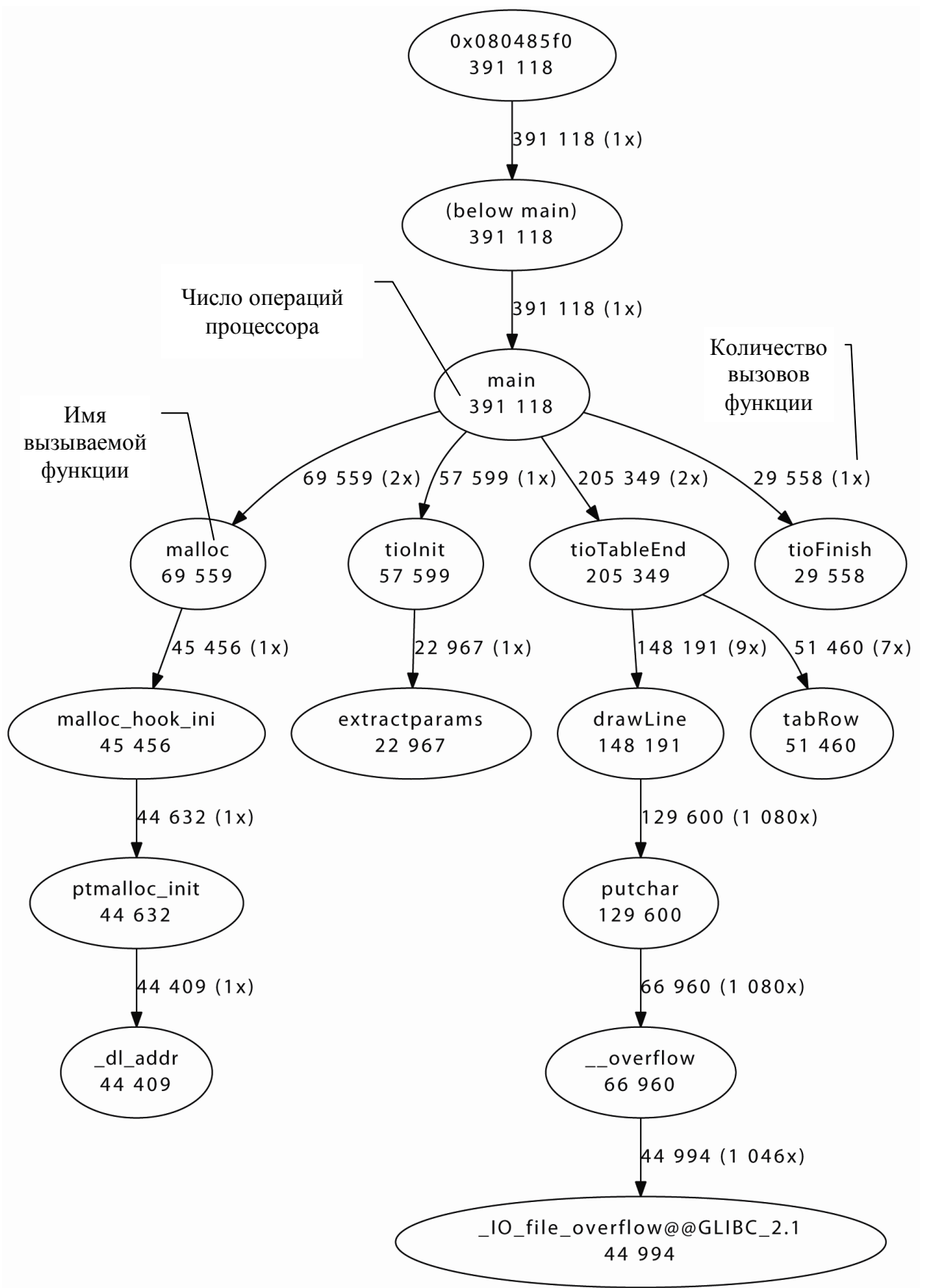


Рис. 3.2

3.2. Анализ производительности библиотеки интерфейсов

По результатам работы профилировщика *Valgrind* (инструмент профилировки *Memcheck*) из рис. 3.1 видно, что приложением выделяется 234 блока памяти, весящие 14,102 байта, а освобождается только 218. Следовательно, 16 блоков весом 1,712 байт не освобождены должным образом. Произошла утечка памяти. Так же определено, что на 14 блоков не указывает не один указатель, что говорит о том, что управление над этими блоками памяти потеряно. Оставшиеся 2 блока, на момент выхода из программы были все ещё достигаемыми, то есть существовали указатели, хранившие в себе адреса этих блоков. Так же из рис. 3.1 видны конкретные строки кода в которых выделяется память для этих блоков, что существенно упрощает поиск тех самых блоков памяти.

Устранение утечки памяти, связанной с двумя блоками памяти, которые на момент выхода из программы были все ещё достигаемыми:

```
diff --git a/prototypes/quadratic-equation/source.c
b/prototypes/quadratic-equation/source.c
index 14b83a7..a0aa898 100644
--- a/prototypes/quadratic-equation/source.c
+++ b/prototypes/quadratic-equation/source.c
@@ -22,7 +22,7 @@ int quad( long a, long b, long c, SRoots*
Roots ) {

    int main( int argc, const char* argv[] ) {
        SRoots *Roots = malloc( sizeof(SRoots) );

-       SRoots *RootsEtalon = malloc( sizeof(SRoots) );

        //int myargc = 6;

@@ -65,8 +65,8 @@ int main( int argc, const char* argv[] ) {
    tioTableRecord( td, tioGetL( "ROOT2" ), Roots->root2 );
    tioTableEnd( td );

-   tioFinish( 0 );
    free(Roots);

+   tioFinish( 0 );
```



```

    return 0;

}

```

Изменения в исходном коде функции отображения строки таблицы, помогающие исправить утечку памяти (14 потерянных блоков).

```

diff --git a/src/tioTableBegin.c b/src/tioTableBegin.c
index 030406d..422cf06 100644
--- a/src/tioTableBegin.c
+++ b/src/tioTableBegin.c
@@ +459,11 @@ int tabRow( void **strings, int *bufType, int
countColum, int lenColCon )
    }
    /*Insert spaces*/
    for(extraCounter = colStr[i] + 1; extraCounter <
(max + 1); ++ extraCounter )
+       {
+           for( offset = 0; offset < (lenColCon - 1);
++ offset )
+               data[i][extraCounter][offset] = ' ';
+           data[i][extraCounter][offset] = '\0';
+       }
    break;
    default:
        printf("ERROR!");
@@ -481,7 +484,7 @@ int tabRow( void **strings, int *bufType,
int countColum, int lenColCon )
    /*FREE */
    for( i = 0; i < countColum; ++ i )
    {
-       for( j = 0; j < colStr[i]; ++ j )
+       for( j = 0; j <= colStr[i]; ++ j )
    {
        free(data[i][j]);
    }
}

```

В результате внесенных изменений *Valgrind*, запущенный с теми же ключами и для той же программы, что и в п. 3.1 выводит сообщение показанное на рис. 3.5.

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ valgrind --tool=memcheck --leak-check=full
--show-reachable=yes --track-origins=yes ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
==4116== Memcheck, a memory error detector
==4116== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==4116== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==4116== Command: ./quadratic-equation
==4116== Parent PID: 1716
==4116==
==4116== HEAP SUMMARY:
==4116==   in use at exit: 0 bytes in 0 blocks
==4116==   total heap usage: 212 allocs, 212 frees, 13,845 bytes allocated
==4116==
==4116== All heap blocks were freed -- no leaks are possible
```

Рис. 3.3

Теперь проанализируем данные о числе вызовов функций, полученные с помощью инструмента *Callgrind*. Из рис. 3.2 видно, что выполнение функции *drawLine* занимает примерно 25% процессорного времени, затрачиваемого на работу всей программы.

Для оптимизации работы данной функции были сделаны следующие изменения:

```
diff --git a/src/tioTableBegin.c b/src/tioTableBegin.c
index 030406d..60bdcd9 100644
--- a/src/tioTableBegin.c
+++ b/src/tioTableBegin.c
int drawLine( int lenColCon )
{
+
+   char *pLine = malloc( WIDTH * sizeof( char ) );

   int i;
   for( i = 0; i < WIDTH; ++ i )
   {
       if((i % lenColCon) == 0)

-           printf("+");
+           pLine[i] = '+';

       else

-           printf("-");
+           pLine[i] = '-';

   }
}
```

```
-    printf( "+\n" );
+    pLine[i] = '+';
+    pLine[++i] = '\n';
+    fputs( pLine, stdout );
+    free(pLine);
+    pLint = NULL;

    return 0;
}
```

Данные изменения помогли уменьшить процессорное время необходимое для выполнения функции *drawLine* (рис. 3.6). Теперь оно составляет около 5% от общего времени выполнения программы.

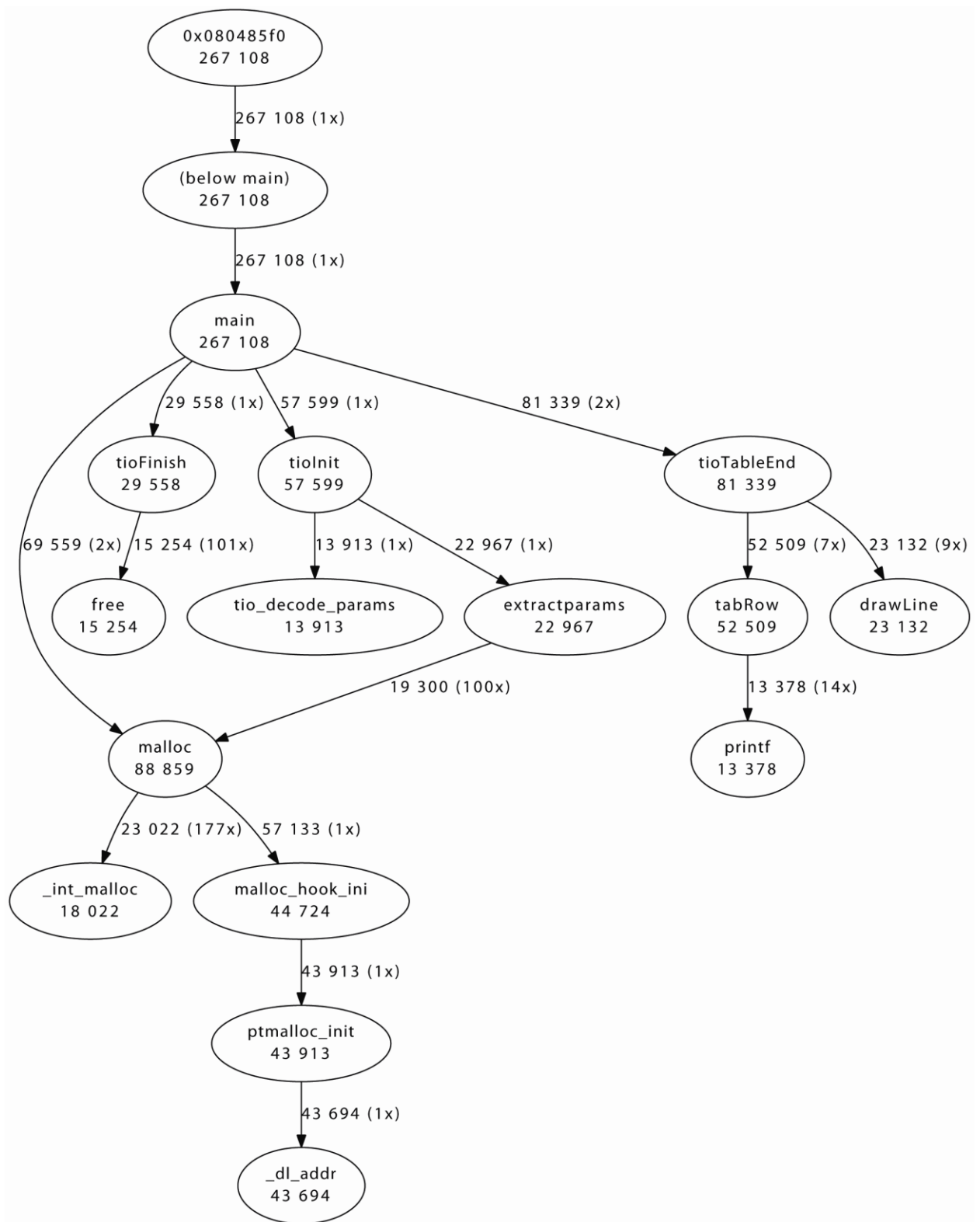


Рис. 3.4

3.3.Отладка и тестирование разрабатываемой библиотеки

Отладка разработанной библиотеки производилась с помощью программы *GDB* (*GNU Debugger*), первоначально написанной Ричардом

Столлмэном в 1988 году и являющейся свободным программным обеспечением.

GDB работает на многих *UNIX*-подобных системах и умеет производить отладку многих языков программирования, включая Си, *C++*, *Free Pascal*, *FreeBASIC*, *Ada* и Фортран.

Отладчик имеет средства для слежения и контроля за выполнением компьютерных программ. Пользователь может изменять внутренние переменные программ и вызывать функции независимо от обычного поведения программы.

С версии 7.0 добавлена поддержка «обратимой отладки», позволяющей отмотать назад процесс выполнения, чтобы посмотреть, что произошло.

При разработке библиотеки *libtio* после добавления новой функции, проводилось автоматическое модульное тестирование по принципу черного ящика, что позволяло быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчало обнаружение и устранение таких ошибок.

Поток вывода после запуска модульных тестов изображен на рис. 3.5 – рис. 3.8.

```

bin/runtests.sh bin
I have color
(TS): Run test: bin/test_all_out
[RUN]: Занык ./bin/test_all_out
(II): Hello my darling
(II): This program is 1-st test for me
(WW): I think It's rather pretty
(WW): I program it well be useful
(EE): But i think it will be great
(EE): It's over 2328
[PASS]: ./bin/test_all_out : Тест пройден успешно
(TS): Test bin/test_all_out [PASS]
(TS): Run test: bin/test_basetioGetC
[RUN]: Занык test_basetioGetC.c
[PASS]: test_basetioGetC.c : Тест пройден успешно
(TS): Test bin/test_basetioGetC [PASS]
(TS): Run test: bin/test_basetioGetD
[RUN]: Занык test_basetioGetD.c
[PASS]: test_basetioGetD.c : Тест пройден успешно
(TS): Test bin/test_basetioGetD [PASS]
(TS): Run test: bin/test_basetioGetDefC
[RUN]: Занык test_basetioGetDefC.c
[PASS]: test_basetioGetDefC.c : Тест пройден успешно
(TS): Test bin/test_basetioGetDefC [PASS]
(TS): Run test: bin/test_basetioGetDefD
[RUN]: Занык test_basetioGetDefD.c
[PASS]: test_basetioGetDefD.c : Тест пройден успешно
(TS): Test bin/test_basetioGetDefD [PASS]
(TS): Run test: bin/test_basetioGetDefL
[RUN]: Занык test_basetioGetDefL.c
[PASS]: test_basetioGetDefL.c : Тест пройден успешно
(TS): Test bin/test_basetioGetDefL [PASS]
(TS): Run test: bin/test_basetioGetDefS
[RUN]: Занык self
TRUE
[PASS]: self : Тест пройден успешно
(TS): Test bin/test_basetioGetDefS [PASS]
(TS): Run test: bin/test_basetioGetL
[RUN]: Занык self
[PASS]: self : Тест пройден успешно
(TS): Test bin/test_basetioGetL [PASS]
(TS): Run test: bin/test_basetioGetS
[RUN]: Занык self
TRUE
[PASS]: self : Тест пройден успешно
(TS): Test bin/test_basetioGetS [PASS]
(TS): Run test: bin/test_error
Attempt to set error before error initialization or after error free
(TS): Test bin/test_error [PASS]

```

Рис. 3.5

```

(TS): Run test: bin/test_error2
(TS): Test bin/test_error2 [PASS]
(TS): Run test: bin/test_ErrorF
(E):char = X long = 123 le = 1.154340e+01 float = 11.543400. 100 in oct = 144. This is string! and h = 7b, H = 7B. Happy% end!(TS): Test bin/test_ErrorF [PASS]

(TS): Run test: bin/test_help
Использование: Test [КЛЮЧ]... [ФАЙЛ]...
Проверка описания программы.
-k, -d, --list, --ls, --lst      List information about the FILES (the current dire
                                ctory by default). Sort entries alphabetically if
                                none of -cftuvSUX nor --sort. Mandatory arguments
                                to long options are mandatory for short options t
                                oo.

--ip, --adress <ПАРАМЕТР>       show / manipulate routing, devices, policy routing
                                and tunnels

-s, -S, --sort <ПАРАМЕТР>       sort lines of text files
--file, --fl <ПАРАМЕТР>         determine file type
-t <ПАРАМЕТР>                   table view
(TS): Test bin/test_help [PASS]
(TS): Run test: bin/test_longto
(TS): Test bin/test_longto [PASS]
(TS): Run test: bin/test_output
(E):26767683687684376876348768638768372686387687638763874687364876384768764387jddhkjhfkjdjhkjhew;jhl;kjelskjlkelkjlkefl;s;lkej;lkj;lej;
ojlkhjlkewyp9ub oiuro;icnj8p2[oicnpoi2poi2p3ipoi2309878yrhjewishouyfipdhlu3rpo8u7093kpojkd09iupjrepwu09ufeoi09fueoi0ifeu980j32oiy87y9jddh
lkjlkjesflkjlke
(E):Символ = Y число 128 с плавающей: 1.154340e+01, ещё раз: 11.543400, 100 в окт: 144, строка: 8048700 в хексе: 7b, в ХЕКСЕ: 7B %The end.
(E):Символ = Z число 129 с плавающей: 1.154340e+01, ещё раз: 11.543400, 100 в окт: 144, строка: 8048700 в хексе: 7b, в ХЕКСЕ: 7B %The end.
(TS): Test bin/test_output [PASS]
(TS): Run test: bin/test_strcmp
(TS): Test bin/test_strcmp [PASS]

```

Рис. 3.6

```
(TS): Run test: bin/test_table
```

```
Cap string
```

```
Call "tioTableBegin".
```

```
Call "tioTableRecord".
```

```
Call "tioTableEnd".
```

char	string	double	string
r	An advantage of COM+	23.70	Animated by Ryan Woodward
e	An advantage of COM+	43.90	The essence of COM is a language-neutral way of implementing objects that can be used in environments

```
(TS): Test bin/test_table [PASS]
```

```
(TS): Run test: bin/test_tioInit
```

```
Test start
```

```
[RUN]: Запуск self
```

```
[PASS]: self : Тест пройден успешно
```

```
(TS): Test bin/test_tioInit [PASS]
```

Рис. 3.7


```

(TS): Run test: bin/test_true
This test is allwayse good:)
(TS): Test bin/test_true [PASS]
(TS): Run test: bin/test_utf
Амбивалентный
мбивалентный
бивалентный
ивалентный
валентный
алентный
лентный
ентный
нтный
тный
ный
ый
й
(TS): Test bin/test_utf [PASS]
(TS): Run test: bin/test_version
Version 0.4.8, revision 209
(TS): Test bin/test_version [PASS]
(TS): Run test: bin/fail_test2
test2
(TS): Test bin/fail_test2 [PASS]
(TS): Run test: bin/fail_true
(TS): Test bin/fail_true [PASS]
(TS): Run test: bin/fail_true2
(TS): Test bin/fail_true2 [PASS]
(TS): All tests PASS

```

Рис. 3.8

Сценарий командной оболочки, позволяющий автоматизировать процесс запуска тестов, представлен в Приложение 3.

Вначале сценария приписываем в переменную окружения LD_LIBRARY_PATH папку *./lib*. Это делается для того, чтобы модульные тесты искали скомпилированную библиотеку *libtio* в директории *lib*, находящуюся в корневой директории проекта. Далее проводим настройку цветов некоторых элементов вывода, таких как *TS*, *Test*, *Run test*, *PASS*, *FAIL* и т. д. Далее все исполняемые файлы, начинающиеся с *test_*, поочередно запускаются и если завершаются без ошибок, то в поток вывода печатается сообщение о том, что тест пройден. В противном случае — печатается сообщение о том, что тест провален.

Так же среди прочих тестов, имеются тесты, успешным выполнением которых является их завершение с определенным кодом ошибки. Название таких тестов начинается с *fail_*. Такие тесты считаются успешно завершенными, если они возвращают значение равное значению из одноименного файла с типом *.result*.

В конце сценария переменная `LD_LIBRARY_PATH` возвращается в первоначальное значение.

Если все модульные тесты завершились успехом, то выводится сообщение, символизирующее отсутствие ошибок при автоматическом тестировании. Если хотя бы один тест провалился, то по сценарию выводится сообщение, что модульное тестирование не прошло успешно.

В ходе функционального тестирования было выявлено, что функция *tioTableEnd* отображает таблицу некорректно, если в полях таблицы используются символы кириллицы (рис. 3.9).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
[ RUN ]: Заныск ./quadratic-equation
+-----+-----+-----+
|Имя параметра|Значение|
+-----+-----+-----+
|Аргумент А   |1       |
+-----+-----+-----+
|Аргумент В   |2       |
+-----+-----+-----+
|Аргумент С   |-3      |
+-----+-----+-----+
Сравнение эталонных и возвращаемых функцией корней
+-----+-----+-----+
|Эталонные корни|Корни, посчитанные функцией|
+-----+-----+-----+
|1              |1                            |
+-----+-----+-----+
|-3             |-3                           |
+-----+-----+-----+
[PASS]: ./quadratic-equation : Тест пройден успешно
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ gvim quadratic-equation
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ gvim source.c
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 3.9

При отладке было установлено, что при работе с символьными данными (строками) по умолчанию используется кодировка *UTF-8*, а это значит, что для представления латинских символов требуется 1 байт, тогда как для представления символов кириллицы необходимо отводить под каждую букву 2 байта. В *tioTableEnd* это не было учтено.

```
@@ -345,7 +345,7 @@ int tabRow( void **strings, int *bufType,
int countColum, int lenColCon )
```

```

        /*Calculation number of extra lines of the array*/
        for( i = 0; i < countColum; ++ i )

-       {
+       {
            colStr[i] = strlen( (char *)strings[i] ) /
lenColCon;
            if(max < colStr[i])
                max = colStr[i];
@@ -368,7 +368,7 @@ int tabRow( void **strings, int *bufType,
int countColum, int lenColCon )
        {
            for (j = 0; j < (max + 1); ++ j)
            {

-               if((data[i][j] = (char *) malloc (lenColCon
* sizeof(char))) == NULL)
+               if((data[i][j] = (char *) malloc ( 2 *
lenColCon * sizeof(char))) == NULL)
            {
                printf("ERROR!\n");
                exit(EXIT_FAILURE);
@@ -419,14 +419,42 @@ int tabRow( void **strings, int
*bufType, int countColum, int lenColCon )
                case 4:
                    for( extraCounter = 0; extraCounter <=
colStr[i]; ++ extraCounter )
                    {

+               int index = 0;
+               j = extraCounter * (lenColCon - 1);

-               for( offset = 0;
+               offset = 0;
+               while( ( ( lenColCon - 1 ) != index ) && ( (
(char*)strings[i])[j] != '\0' ) )
+               {
+                   if( ( (char*)strings[i])[j] & 0x80 )
+                   {
+                       data[i][extraCounter][offset] =
((char *)strings[i])[j];
+                       ++ offset;
+                       ++ j;

```

```

+             data[i][extraCounter][offset] =
+             ((char *)strings[i])[j];
+             ++ j;
+             ++ index;
+             ++ offset;
+         }
+         else
+         {
+             data[i][extraCounter][offset] =
+             ((char *)strings[i])[j];
+             ++ j;
+             ++ offset;
+             ++ index;
+         }
+     }
+
-         for( offset = 0;
-             ((offset != (lenColCon - 1)) &&
+             (((char *)strings[i])[j] != '\0' )); ++ offset, ++ j)
-             {
-                 data[i][extraCounter][offset] = ((char
+                 *)strings[i])[j];
-             }
+
+             /*Insert spaces*/
+
-             for( offset; offset < (lenColCon - 1); ++
+             offset)
+             for( index; index < (lenColCon - 1); ++
+             index, ++ offset)
+
+                 data[i][extraCounter][offset] = ' ';
+         }
+         /*Insert spaces*/

```

После исправлений поля с символами кириллицы стали отображаться правильно (рис. 3.10).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation -a 1 -b 2 -c -3 --root1=1 --root2=-3
[RUN]: Запуск ./quadratic-equation
+-----+-----+
|Имя параметра|Значение|
+-----+-----+
|Аргумент А|1|
+-----+-----+
|Аргумент В|2|
+-----+-----+
|Аргумент С|-3|
+-----+-----+
Сравнение эталонных и возвращаемых функцией корней
+-----+-----+
|Эталонные корни|Корни, посчитанные функцией|
+-----+-----+
|1|1|
+-----+-----+
|-3|-3|
+-----+-----+
[PASS]: ./quadratic-equation : Тест пройден успешно
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 3.10

Также было выявлено, что функция *tioHelp* не выводит (см. рис. 3.11) название длинных ключей, если у них не существует аналогичных коротких.

```
nwcfang@nf-lrti:~/current-task/prototypes/quadratic-equation$ ./quadratic-equation --help
Использование: ./quadratic-equation [КЛЮЧ]... [ФАЙЛ]...
This is test
-a <ПАРАМЕТР>          Параметр А
-b <ПАРАМЕТР>          Параметр В
-c <ПАРАМЕТР>          Параметр С
<ПАРАМЕТР>            Первый корень
<ПАРАМЕТР>            Второй корень
nwcfang@nf-lrti:~/current-task/prototypes/quadratic-equation$
```

Рис. 3.11

Исправление, которое помогло решить эту проблему.

```
diff --git a/src/help.c b/src/help.c
index 9e7123a..f630ccd 100644
--- a/src/help.c
+++ b/src/help.c
@@ -29,6 +29,7 @@ int tioHelp( const char* help_msg, const
char* progName,
int counter;
for( idx = 0 ; idx < sz ; ++idx )
{
+    nolong = 0;
    counter = MAX_TAB;
    // Вывод короткого ключа
    if( par[idx].skeys )
```

Теперь ключи отображаются правильно (рис. 3.12).

```
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$ ./quadratic-equation --help
Использование: ./quadratic-equation [КЛЮЧ]... [ФАЙЛ]...
This is test
-a <ПАРАМЕТР>          Параметр А
-b <ПАРАМЕТР>          Параметр В
-c <ПАРАМЕТР>          Параметр С
--root1 <ПАРАМЕТР>     Первый корень
--root2 <ПАРАМЕТР>     Второй корень
nwcfang@nwcfang-Z68AP-D3:~/current_task/prototypes/quadratic-equation$
```

Рис. 3.12

Еще одна ошибка в логике была выявлена в функции *tioInit*. В случае, когда в программу вместе с командной строкой передавались неименованные параметры, они неправильным образом сохранялись в памяти кучи, что приводило к потерям данных (рис. 3.13).

```
(TS): Test bin/test_table [PASS]
(TS): Run test: bin/test_tioInit
Test start
[RUN]: Запуск self
unnamedKey is (null)
bin/runtests.sh: строка 41: 11171 Ошибка сегментирования
(TS): Test bin/test_tioInit [FAIL]
```

Рис. 3.13

После исправлений исходный код функций выглядит так:

```
diff --git a/src/finish.c b/src/finish.c
index f4fb5ed..beeb71c 100644
--- a/src/finish.c
+++ b/src/finish.c
@@ -18,7 +18,7 @@
#include <tioinit.h>

#include <finish_msg.h>

-
+#define MAXARGS 100

extern char *selfname;

@@ -30,6 +30,8 @@ void tioFinish(size_t num)
{
    num = finish_count;
}

+    for(int i = 0; i < MAXARGS; ++i)
```

```

+         free(tio_argv[i]);

        tioFree();
        fprintf(stdout, finish_messages[num], selfname);
        free(selfname);

diff --git a/src/init.c b/src/init.c
index bc16c10..da4fed0 100644
--- a/src/init.c
+++ b/src/init.c
@@ -573,6 +573,9 @@ static int extractparams(int start, int
argc, char** argv)
    tio_simple_chain *pt = NULL;
    tio_key_string *p;

+    for( int nfi = 0; nfi < MAXARGS; ++ nfi)
+        tio_argv[nfi] = malloc( sizeof(char) * 100);
+
    for (i=start; i < argc; i++)
    {
        if (argv[i][0]=='-')
@@ -681,7 +684,9 @@ static int extractparams(int start, int
argc, char** argv)
    }
    for (i = cnt; i>0;)
    {
-        tio_argv[--i]=pt->val;
+
+        strcpy( tio_argv[--i], pt->val );
+        /*tio_argv[--i]=pt->val;*/

        pt=pt->next;
        free(ptr);
        ptr=pt;

```

Корректная работа функции *tioInit* показана на рис. 3.14.

```

(TS): Run test: bin/test_tioInit
Test start
[RUN]: Занято self
unnamedKey is unnamedKey
[PASS]: self : Тест пройден успешно
(TS): Test bin/test_tioInit [PASS]

```

Рис. 3.14

4. Охрана труда и окружающей среды. Разработка мероприятий по обеспечению благоприятных санитарно-гигиенических условий труда инженера

Введение

Содержание дипломной работы заключается в создании библиотеки функций унификации процессов обработки входных параметров и систематизации выходных данных в средствах тестирования и диагностики программных средств и оборудования. Целью данного раздела является анализ и оценка соответствия требованиям безопасности освещенности рабочего помещения, микроклимата и визуальных параметров монитора и выработка необходимых мероприятий по обеспечению благоприятных санитарно-гигиенических условий труда.

4.1. Анализ условий труда инженера-программиста

4.1.1. Характеристика условий труда инженера-программиста

4.1.1.1. Характеристика труда

Специфика труда разработчика программного обеспечения включает следующие этапы работы:

- анализ и поиск решения задачи,
- программирование,
- отладку и тестирование программных компонент,
- выпуск документации.

Большую часть рабочего времени программист проводит за компьютером. Такая работа характеризуется длительным сидячим положением, что не подразумевает значительных физических нагрузок. Продолжительность рабочего дня – 8 часов, с получасовым перерывом на обед.

4.1.1.2. Характеристика технических средств

Используемое для работы оборудование:

- персональный компьютер;
 - системный блок;
 - клавиатура;
 - мышь;
 - монитор
- принтер.

Нормальная и безопасная для здоровья работа инженера-программиста во многом зависит от того, в какой мере параметры монитора соответствуют требованиям безопасности. Поэтому рассмотрим данные параметры более подробно.

На рабочем месте программиста установлен монитор Samsung SyncMaster S27A550H:

Тип	ЖК-монитор
Диагональ	27"
Яркость	300 кд/м ²
Контрастность	1000:1
Потребляемая мощность	при работе: 40 Вт, в спящем режиме: 0.50 Вт
Частота обновления экрана	60Гц

4.1.1.3. Количество работающих людей

В помещении работают 6 человек.

4.1.1.4. Характеристика помещения

Работа ведется в офисном помещении длиной $L = 6,6\text{м}$, шириной $W = 5,4\text{м}$ и высотой $H = 4\text{м}$. Соответственно, площадь помещения составляет $S = L \cdot W = 6,6 \cdot 5,4 = 35,64 \text{ м}^2$, а его объём $V = S \cdot H = 35,64 \cdot 4 = 142,56 \text{ м}^3$.

Естественное освещение – боковое; его обеспечивают два оконных проёма, каждый проём длиной 2,3 метра и высотой 2. метра.

Потолок окрашен в белый цвет.

Для искусственного освещения используются 8 светильников ООО «Завод «Световые технологии», имеющих следующие характеристики:

Артикул	Мощность, Вт	Размеры, мм×мм
ARS/R 418 (595)	4×18	595×595

Светильники оснащены люминесцентными лампами Т8 фирмы OSRAM L 18W/640 25X1 (световой поток $\Phi_{\text{л}} = 1350$ лм) и имеют сертификат соответствия ГОСТ:

**СИСТЕМА СЕРТИФИКАЦИИ ГОСТ Р
ГОССТАНДАРТ РОССИИ**

СЕРТИФИКАТ СООТВЕТСТВИЯ

№ РОСС RU.МЕ64.В09202
Срок действия с 24.12.2009 по 24.12.2012
8582698

ОРГАН ПО СЕРТИФИКАЦИИ РОСС RU.0001.11МЕ64
ОРГАН ПО СЕРТИФИКАЦИИ СВЕТОТЕХНИЧЕСКИХ ИЗДЕЛИЙ
И ЭЛЕКТРОУСТАНОВОЧНЫХ УСТРОЙСТВ АНО "СветоС"
129626, г. Москва, проспект Мира, 106
Телефон/факс: 788-65-96, 682-39-92

ПРОДУКЦИЯ
Светильники встраиваемые
(см. приложения №№ 2298703, 2298704, 2298705,
2298706, 2298707)
ТУ 3461-002-44919750-07
Серийный выпуск

СООТВЕТСТВУЕТ ТРЕБОВАНИЯМ НОРМАТИВНЫХ ДОКУМЕНТОВ

безопасности ГОСТ Р МЭК 60598-1-2003,
ГОСТ Р МЭК 60598-2-2-99
и ЭМС ГОСТ Р 51318.15-99, ГОСТ Р 51514-99,
ГОСТ Р 51317.3.2-2006, ГОСТ Р 51317.3.3-2006

ИЗГОТОВИТЕЛЬ
ООО "Завод "Световые технологии"
390010, г.Рязань, ул.Магистральная, д.11-А

СЕРТИФИКАТ ВЫДАН
ООО "Завод "Световые технологии"
390010, г.Рязань, ул.Магистральная, д.11-А
Телефон: 8(4912)24-11-75 Факс: 8(4912)24-11-78 ИНН 6229028102

НА ОСНОВАНИИ
протоколов сертификационных испытаний №№ С-363с, С-364с, С-365с,
С-366с от 24.12.2009 г., проведенных в испытательном центре
светотехнических изделий и электроустановочных устройств
АНО "СветоС" (ИЦ СииЭУ АНО "СветоС") РОСС RU.0001.21МЕ24
протоколов испытаний №№ 271L11Z-09, 272L11Z-09, 273L11Z-09, 274L11Z-09
от 30.11.2009 г., проведенных в испытательной лаборатории
электротехнической продукции ЭМС (ИЛ "ЭП ЭМС") РОСС RU.0001.21МЭ48

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ
Маркировка продукции производится знаком соответствия по
ГОСТ Р 50460-92 на маркировке рядом с товарным знаком изготовителя
и на товарно-сопроводительной документации

Руководитель органа *С.Н.Пасынкова*
Эксперт *М.И.Мызина*

С.Н.Пасынкова
инженер, специалист
М.И.Мызина
инженер, специалист

Сертификат имеет юридическую силу на всей территории Российской Федерации

Рис. 4.1

4.1.2. Анализ освещения, микроклимата и визуальных параметров устройства отображения информации.

4.1.2.1. Освещение

а) Искусственное освещение

1) Цель расчёта:

Определить фактическую освещенность в двух точках помещения от данной осветительной установки, используя точечный метод расчета освещенности.

Выбираем следующие две точки помещения с координатами, исходя из следующих условий:

- 1) Данные точки находятся на условной поверхности, на расстоянии 0,8м от пола.
- 2) Одна точка находится посередине помещения, другая – у конца светящей линии (на **Ошибка! Источник ссылки не найден.** это точки а и б).

Длина ряда светильников равна 4,2 м, а высота осветительной установки

$$h_{0y} = H - h_c - h_{p.п.} = 4 - 0 - 0,8 = 3,2 \text{ м}$$

где:

- H – высота потолка;
- h_c – высота свеса светильника;
- $h_{p.п.}$ – высота рабочей поверхности.

Так как длина ряда светильников превышает 0,5 высоты осветительной установки ($4,2 > 0,5 \cdot 3,2$), то такой ряд можно рассматривать как светящую линию.

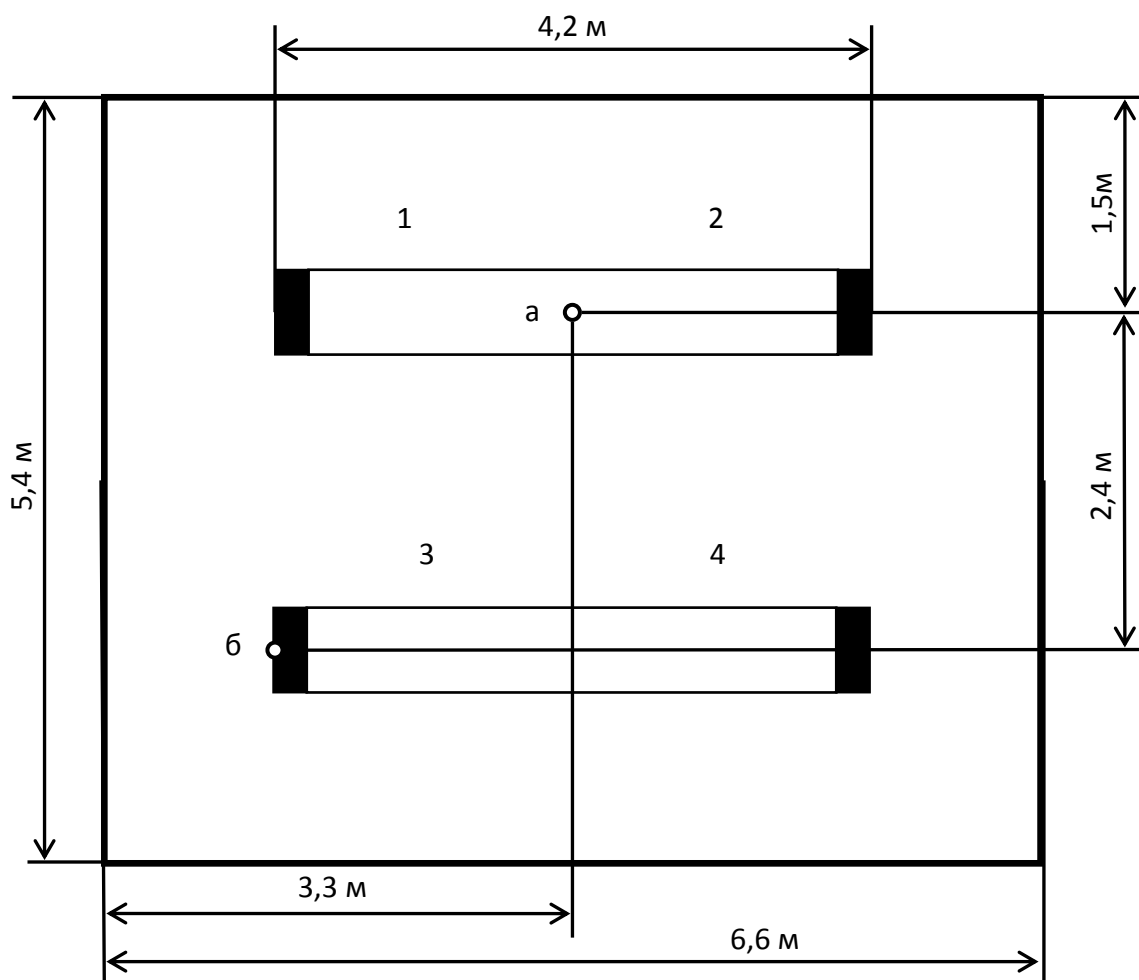


Рис. 4.2. Осветительная установка.

2) Формула расчёта фактической освещённости точек а и б.

Фактическая освещённость определяется по формуле:

$$E_{\Phi} = \frac{\Phi}{1000} \frac{\mu \Sigma \varepsilon}{k_3 h L},$$

где

- Φ – суммарный световой поток всех источников, лм;
- $\mu = 1,1 \dots 1,2$ – коэффициент, учитывающий отражённую составляющую света и действий удалённых светильников ($\mu = 1,1$);
- $\Sigma \varepsilon$ – сумма относительных освещённостей от нескольких светящих линий;
- k_3 – коэффициент запаса, учитывающий запыление светильников и износ источников света в процессе эксплуатации;

Для помещений, освещаемых люминесцентными лампами, и при условии чистки светильников не реже двух раз в год, коэффициент запаса равен 1,4 ... 1,5 ($k_3 = 1,5$);

- h – высота подвеса светильников над рабочей поверхностью;
- L – общая длина светящихся линий, м

3) Определение суммарного светового потока от всех источников, Φ .

$$\Phi = \Phi_{\text{л}} m N n,$$

где

- $\Phi_{\text{л}}$ – световой поток лампы, лм.

Световой поток от 1 лампы (см. п.4.1.1.4) $\Phi_{\text{л}} = 1350$ лм.

- $m = 4$ – количество ламп в одном светильнике;
- $N = 4$ – количество светильников в одном ряду;
- $n = 2$ – количество рядов светильников.

Следовательно:

$$\Phi = 1350 \cdot 4 \cdot 4 \cdot 2 = 43200 \text{ лм}$$

4) L – общая длина светящихся линий.

$$L = n \cdot l_{\text{св}} \cdot N$$

Где:

- h – высота подвеса светильников над рабочей поверхностью, м.

Высота рабочей поверхности над полом равна 0,8м.

Отсюда:

$$h = H - 0,8 = 4 - 0,8 = 3,2 \text{ м}$$

- $l_{\text{св}} = 0,595$ м – длина светильника;

Таким образом:

$$L = 2 \cdot 0,595 \cdot 4 = 4,76 \text{ м}$$

5) Определение суммы относительных освещенностей от нескольких светящихся линий ($\Sigma \epsilon$).

Относительная освещенность ϵ , лк, – это освещенность при удельном световом потоке

$$\Phi_0^1 = 1000 \text{ лм/м} \text{ и } h = 1 \text{ м,}$$

Относительная освещенность определяется с помощью расчетных графиков линейных изолюкс (см. рис. 4.3). Графики построены для различных типов светильников, образующих светящие линии, в координатной системе (p' , L'):

$$p' = \frac{p}{h} \text{ и } L' = \frac{L}{h} \text{ — приведенные размеры}$$

где h — высота подвеса светильников над рабочей поверхностью, м.

Для определения относительной освещенности ε , лк, находим:

- а) Для каждой точки (а или б) определяем полуряды или ряды светильников (линий), которые освещают данную точку.

Для точки *а* это полуряды 1, 2, 3, 4, а для точки *б* — ряды 1-2 и 3-4.

- б) Определяем p , L , p' , L' для каждой точки.

Точка а:

$$L = 2,1 \text{ м для всех полурядов} \Rightarrow L' = \frac{2,1}{3,2} = 0,66;$$

$$p = 0 \text{ для полурядов 1, 2 и } p = 2,4 \text{ для полурядов 3, 4 } (p' = \frac{0}{3,2} = 0 \text{ и}$$

$$p' = \frac{2,4}{3,2} = 0,75 \text{ соответственно}).$$

Точка б:

$$L = 4,2 \text{ м для всех рядов} \Rightarrow L' = \frac{4,2}{3,2} = 1,31;$$

$$p = 2,4 \text{ для ряда 1-2 и } p = 0 \text{ для ряда 3-4 } (p' = \frac{2,4}{3,2} = 0,75 \text{ и } p' =$$

$$\frac{0}{3,2} = 0 \text{ соответственно}).$$

- в) По графику линейных изолюкс (рис. 4.3) по p' , L' определяем относительную освещенность ε для каждого полуряда и ряда светильников, которые освещают точку (сначала точку *а*, а потом — точку *б*).

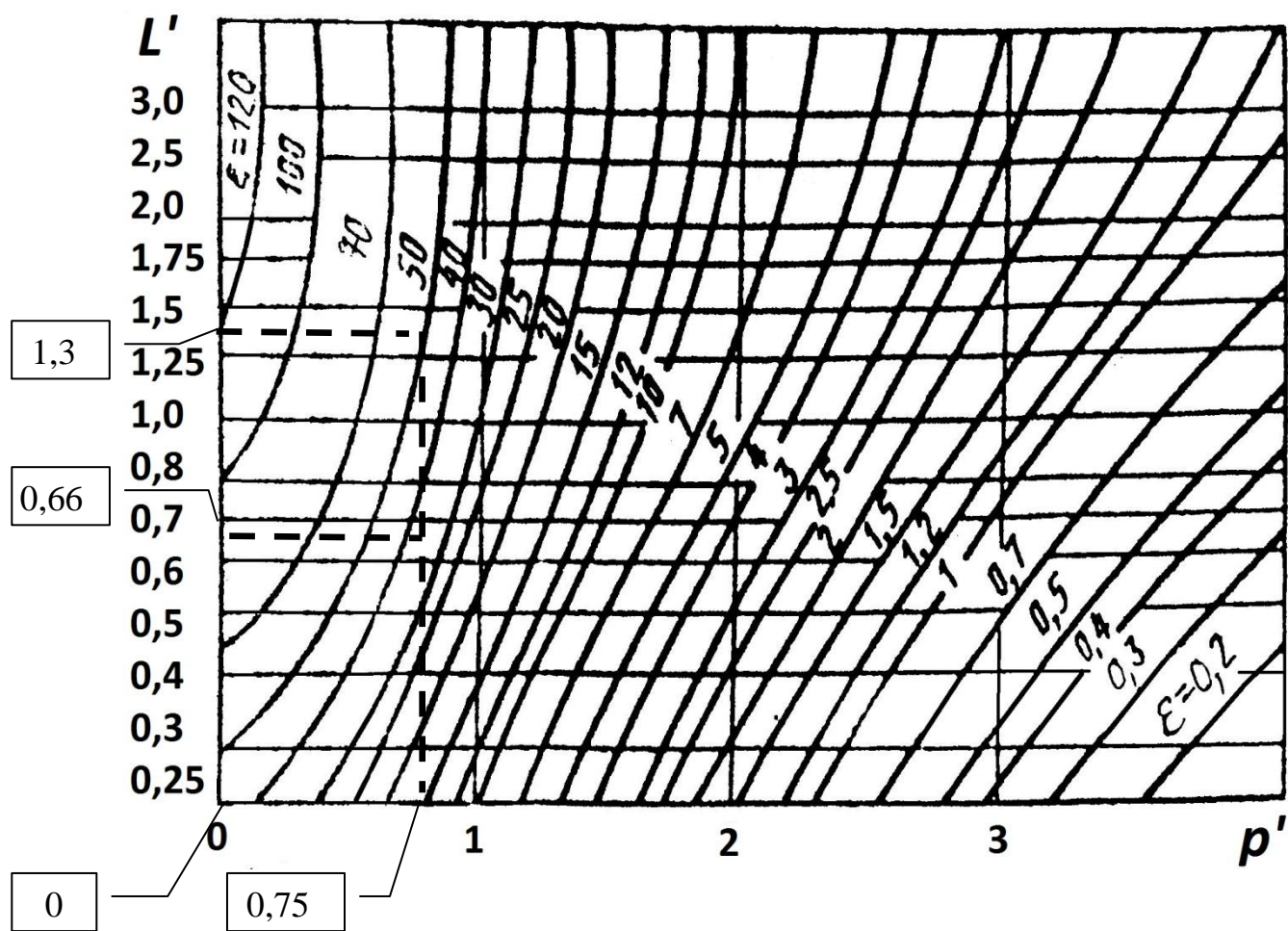


Рис. 4.3. Линейные изолюксы светильников

Итоговая таблица расчёта суммы относительных освещённостей от нескольких светящих линий для точек а и б.

Точка	Полуряд или ряд	p	L	p'	L'	Относительная освещенность ε , лк
а	1, 2	0	2,1	0	0,66	$2 \times 85 = 170$
	3, 4	2,4	2,1	0,75	0,66	$2 \times 38 = 76$
						$\Sigma \varepsilon = 246$
б	1-2	2,4	4,2	0,75	1,31	$1 \times 50 = 130$
	3-4	0	4,2	0	1,31	$1 \times 120 = 120$
						$\Sigma \varepsilon = 170$

б) Окончательный расчёт фактической освещённости $E_{\Phi} = \frac{\Phi}{1000} \frac{\mu \Sigma \varepsilon}{k_3 h L}$

Учитывая, что $\Phi = 43\,200$ лм, $L = 4,76$ м, имеем:

$$E_a = \frac{43200 \cdot 1,1 \cdot 246}{1000 \cdot 1,5 \cdot 3,2 \cdot 4,76} = 511,6 \text{ лк}$$

$$E_6 = \frac{43200 \cdot 1,1 \cdot 170}{1000 \cdot 1,5 \cdot 3,2 \cdot 4,76} = 353,57 \text{ лк}$$

7) Сравнение полученных результатов с нормативными значениями

Работа за пультами ЭВМ, дисплеев относится к III разряду зрительных работ (подразряд 2) с наименьшим эквивалентным размер объекта различения равным 0,3-0,5 мм. По таблице 1 СНиП 23-05-95 определяем нормируемую освещённость, которая равна 200 лк при общем освещении.

Характеристика зрительной работы	Разряд и подразряд	Контраст объекта с фоном	Характеристика фона	Искусственное освещение, лк	
				При комбинированном освещении	При общем
Высокой точности 0,3-0,5	III г	большой	светлый	400	200

Таким образом, можно сделать вывод, что имеющаяся система общего освещения удовлетворяет требованиям, устанавливаемым СНиП 23-05-95.

б) Естественное освещение

1) Коэффициент естественного освещения

В соответствие с характеристикой помещения, приведённой в п.4.1.1.4, площадь офисного помещения, в котором работает инженер-программист, равна $A_n = 35,64 \text{ м}^2$, а его глубина (расстояние от стены, где расположены оконные проёмы, до противоположенной стены) – $d_n = 6,6 \text{ м}$.

Суммарная площадь двух оконных проёмов высотой 2,8 м и шириной 2,3 м:

$$A_{c.o.} = 2 \cdot (2,8 \cdot 2,3) = 12,88 \text{ м}^2$$

Исходя из того, что условная рабочая поверхность находится на высоте 0,8 м от пола, высота подоконника составляет 1 м и высота оконного проёма 2,8 м, определим высоту верхней грани световых проемов над уровнем условной рабочей поверхности:

$$h_{01} = (1 + 2,8) - 0,8 = 3 \text{ м}$$

Таким образом:

$$\frac{A_{\text{с.о.}}}{A_{\text{п}}} \cdot 100\% = \frac{12,88}{35,64} \cdot 100\% = 36\%$$

$$\frac{d_{\text{п}}}{h_{01}} = \frac{6,6}{3} = 2,2$$

Для определения КЕО воспользуемся графиком, представленным на рис. 4.4.

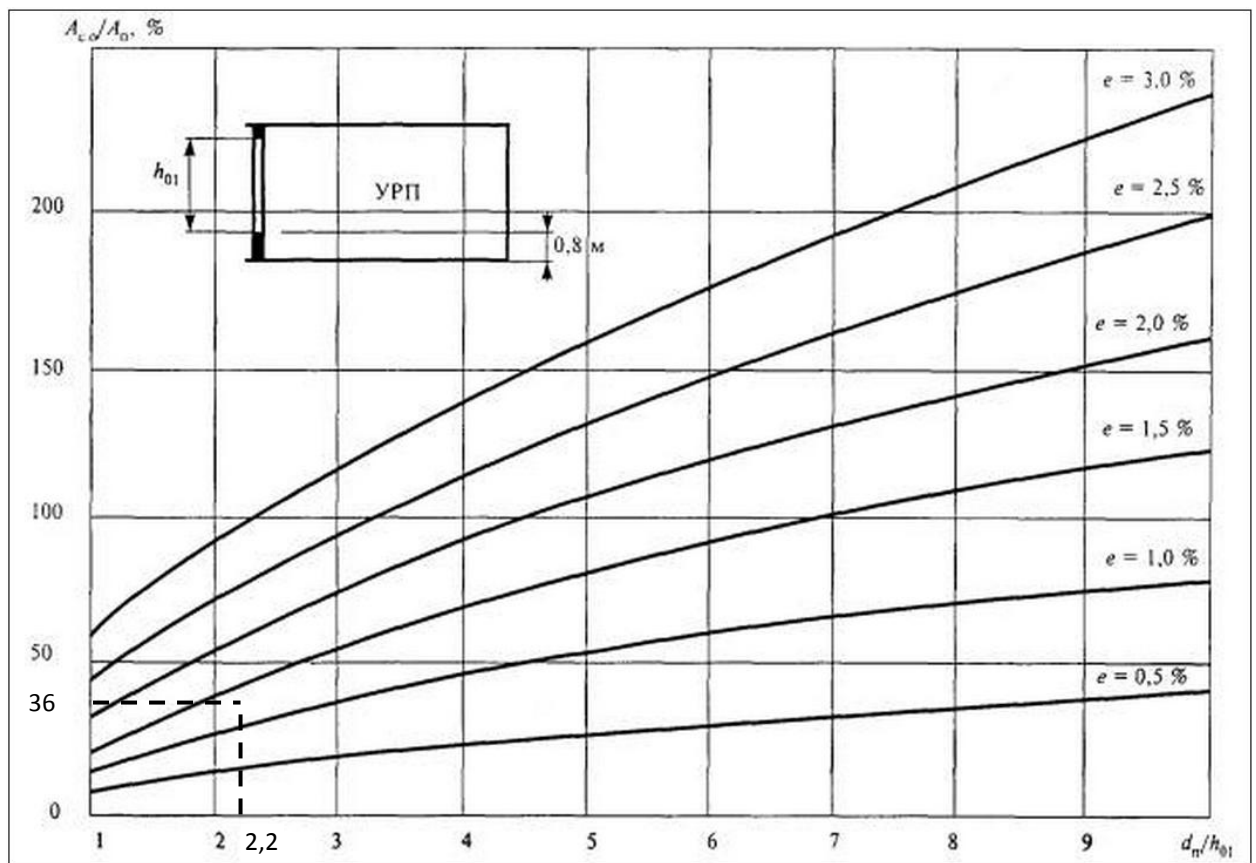


Рис. 4.4. График для определения относительной площади световых проемов $A_{\text{с.о.}}/A_{\text{п}}$ при боковом освещении помещений общественных зданий.

По графику находим, что $e = 1,3\%$.

Нормированное значение КЕО, e_N , определяем по формуле

$$e_N = e_H \cdot m_N$$

где

- N – номер группы обеспеченности естественным светом ($N = 1$ для Москвы согласно Приложению Ж, СНиП 23-05-95);
- m_N – коэффициент светового климата по табл. 4 ($m_N = 1$);
- e_H – значение КЕО по табл. 1.

Характеристика зрительной работы	Разряд и подразряд	Естественное освещение, КЕО, e_H , %	
		При верхнем или комбинированном освещении	При боковом освещении
Высокой точности 0,3-0,5	III г	3	1,2

Окончательно, $e_N = 1,2 \cdot 1 = 1,2$ %. Поэтому можно сделать вывод, что естественное освещение помещения удовлетворяет требованиям СНиП 23-05-95.

Естественное и искусственное освещение офисного помещения удовлетворяют требованиям безопасности. Это создаёт необходимые условия зрительной работы, снижает утомляемость, способствует повышению производительности труда, благотворно влияет на производственную среду, оказывая положительное психологическое воздействие, повышает безопасность труда и снижает травматизм.

4.1.2.2. Микроклимат

Общие санитарно-гигиенические требования к показателям микроклимата устанавливают ГОСТ 12.1.005-88 «Общие санитарно-гигиенические требования к воздуху рабочей зоны», СанПиН 2.2.4.548–96 «Физические факторы производственной среды гигиенические требования к микроклимату производственных помещений».

Показателями, характеризующими микроклимат в помещении, являются:

- 1) температура воздуха;
- 2) относительная влажность воздуха;
- 3) скорость движения воздуха;
- 4) интенсивность теплового излучения.

Работа за компьютером относится к категории легких физических работ (категория Ia) – виды деятельности с расходом энергии до 120 ккал/час (139 Вт), т.е. работы, производимые сидя и сопровождающиеся незначительным физическим напряжением [Приложение 1 к ГОСТ 12.1.005-88].

Оптимальные и допустимые нормы температуры, относительной влажности и скорости движения воздуха в рабочей зоне

производственных помещений

[Таблица 1, СанПиН 2.2.4.548–96]

Период года	Категория работ	Температура, °С			Относительная влажность, %		Скорость движения, м/с	
		оптим.	допустимая на рабочих местах		оптим.	допуст.	оптим. не более	допуст.
			пост.	непост.				
Холодный	Легкая - Ia	22-24	21- 25	18-26	40-60	75	0,1	0,1
Теплый	Легкая - Ia	23-25	22- 28	20-30	40-60	55	0,1	0,1-0,2

Относительная влажность воздуха в производственном помещении в холодный и теплый период года от 50-60%, что попадает в границы нормированной оптимальной температуры.

Скорость движения воздуха в холодный период и теплый период года составляет не более 0,1 м/с, что соответствует санитарным нормам.

Температура в помещении в холодный период года составляет 24-27 °С; а в теплый период – (25-30°С). То есть выходит за границы нормированной оптимальной температуры.

То, что параметры микроклимата выходят за границы оптимальных значений, может привести к возникновению общих и локальных ощущений

теплового дискомфорта, напряжению механизмов терморегуляции, ухудшению самочувствия и понижению работоспособности.

4.1.2.3. Визуальные параметры устройств отображения информации

Требования к эксплуатации импортных ПЭВМ, используемых на производстве, определяют санитарно-эпидемиологические правила и нормативы СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы». Допустимые визуальные параметры устройств отображения информации представлены в Приложении 1 (таблица 4) к данному СанПиН:

N	Параметры	Допустимые значения
1	Яркость белого поля	Не менее 35 кд/кв.м
2	Неравномерность яркости рабочего поля	Не более $\pm 20\%$
3	Контрастность (для монохромного режима)	Не менее 3:1
4	Временная нестабильность изображения (непреднамеренное изменение во времени яркости изображения на экране дисплея)	Не должна фиксироваться
5	Пространственная нестабильность изображения (непреднамеренные изменения положения фрагментов изображения на экране)	Не более $2 \times 10(-4L)$, где L – проектное расстояние наблюдения, мм

Также существуют другие визуальные параметры монитора, значение которых не нормируется в данном СанПиН, но которые, тем не менее, могут оказывать вредное влияние на пользователя ПЭВМ. Например, несоответствие излучения экрана дисплея спектру естественного света (особенно в сине-фиолетовом диапазоне длин волн).

Вредное воздействие устройств отображения информации заключается в повышении зрительной и психофизической нагрузки, что приводит к

снижению работоспособности инженера-программиста. Длительное воздействие вредных факторов может привести к ухудшению зрения.

Сравнивая нормативные значения с параметрами дисплея, приведёнными в п. 4.1.1.2, можно сделать вывод, что монитор соответствует требованиям СанПиН 2.2.2/2.4.1340-03.

4.2.Разработка мероприятий по уменьшению отрицательного воздействия производственных факторов

4.2.1. Микроклимат

Исходя из анализа микроклимата в офисном помещении, проведенного в п. 4.1.2.2, видно, что температура воздуха в холодный и тёплый периоды года выходит за границы оптимальных значений, устанавливаемых ГОСТ 12.1.005-88. Поэтому необходимо принять меры к улучшению системы регулирования температуры в помещении. Такой мерой может быть установка кондиционера. Предлагается установить настенную сплит-систему от Electrolux серии Crystal Style, модель EACS-12HC (рис. 4.5).



Рис. 4.5.

Выбор обусловлен тем, что данная модель рассчитана на помещение до 39 м², что соответствует размерам рабочего помещения (≈ 36 м²). Кроме того эта сплит-система имеет функции обогрева и охлаждения, многоступенчатую систему фильтрации воздуха и низкий уровень шума (благодаря эффективной аэродинамике).

4.2.2. Визуальные параметры средств отображения информации

Несмотря на то, что параметры устройства отображения информации удовлетворяют требованиям СанПиН 2.2.2/2.4.1340-03, дисплей все равно

оказывает наиболее вредное воздействие при работе инженера-программиста. Для профилактики зрительного утомления и его снижения при выполнении напряженной зрительной работы полезны упражнения, способствующие улучшению кровоснабжения в глазах и уменьшению усталости.

Такие упражнения можно выполнять на рабочем месте, сидя на стуле.

Так же в качестве защиты глаз от ультрафиолетовой части спектра излучения монитора можно использовать специальные очки, которые должны иметь сертификат соответствия.

В качестве примера рассмотрим очки фирмы «Лорнет-М», имеющие сертификат соответствия Госстандарта РФ, сертификат на очки как на средство индивидуальной защиты (сертификат ВНИИ Сертификации 2010 года), удостоверение о включении в Реестр РФ изделий медицинского назначения за № ФС 012a1663/0921-04, а также для которых представлен спектр светопропускания линз.

Согласно исследованию, проведенному центром «Росмедком», спектральные характеристики данных очков полностью соответствуют рекомендациям Минздравсоцразвития РФ: максимально возможно «вырезают» сине-фиолетовую часть видимого спектра излучения монитора до 0 % (при 380-400 нм) до 50 % (при 440-450 нм). Это позволяет значительно уменьшить хроматическую аберрацию, повысить четкость и контрастность изображения на сетчатке глаза. Линзы обеспечивают светопропускание в диапазоне 500-600 нм, что значительно увеличивает цветоразличительные функции органа зрения. Фильтр полностью блокирует ультрафиолет.

СИСТЕМА СЕРТИФИКАЦИИ ГОСТ Р
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

СЕРТИФИКАТ СООТВЕТСТВИЯ

№ РОСС RU.СМ05.905077
Срок действия с 05.04.2013 по 05.04.2018

0260775

ОРГАН ПО СЕРТИФИКАЦИИ: РОСС RU.0001.11СМ
ОРГАН ПО СЕРТИФИКАЦИИ СРЕДСТВ ИНДИВИДУАЛЬНОЙ ЗАЩИТЫ (СИЗ) "СЕРТОСИЗ")
Открытого акционерного общества "ВСЕРОССИЙСКИЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ СЕРТИФИКАЦИИ" (ВНИИ МЕТРОЛОГИИ)
Телефон/факс (495) 253 00 66; 253 05 00
123557 Москва, Звездный переулок, д. 1, стр. 1

ПРОДУКЦИЯ: ОЧКИ ОТКРЫТЫЕ ЗАЩИТНЫЕ С ПЕРИМЕТРИЧЕСКИМИ СЕТКАМИ
СО СПЕКТРАЛЬНЫМ ФИЛЬТРОМ ЛС-"ЛОРИНЕТ"
"О-ЛС-ЖЗ-8 Лорнет-М", "О-ЛС-ОСОН Лорнет-М", "О-ЛС-ОСОН Лорнет-М"
ТУ 9442-003-17768917-03

Серийный выпуск

СООТВЕТСТВУЕТ ТРЕБОВАНИЯМ НОРМ ДОКУМЕНТОВ

ГОСТ Р 12.4.230.1-2007 (ЕН 1836) 5.2.1.1, 5.2.2, 5.2.3.3, 5.2.3.9, 5.2.6.1, 5.2.7.2, 5.2.8, 5.2.9, 5.2.10, 5.2.11, 5.2.12, 5.2.13, 5.2.14, 5.2.15, 5.2.16, 5.2.17, 5.2.18, 5.2.19, 5.2.20, 5.2.21, 5.2.22, 5.2.23, 5.2.24, 5.2.25, 5.2.26, 5.2.27, 5.2.28, 5.2.29, 5.2.30, 5.2.31, 5.2.32, 5.2.33, 5.2.34, 5.2.35, 5.2.36, 5.2.37, 5.2.38, 5.2.39, 5.2.40, 5.2.41, 5.2.42, 5.2.43, 5.2.44, 5.2.45, 5.2.46, 5.2.47, 5.2.48, 5.2.49, 5.2.50, 5.2.51, 5.2.52, 5.2.53, 5.2.54, 5.2.55, 5.2.56, 5.2.57, 5.2.58, 5.2.59, 5.2.60, 5.2.61, 5.2.62, 5.2.63, 5.2.64, 5.2.65, 5.2.66, 5.2.67, 5.2.68, 5.2.69, 5.2.70, 5.2.71, 5.2.72, 5.2.73, 5.2.74, 5.2.75, 5.2.76, 5.2.77, 5.2.78, 5.2.79, 5.2.80, 5.2.81, 5.2.82, 5.2.83, 5.2.84, 5.2.85, 5.2.86, 5.2.87, 5.2.88, 5.2.89, 5.2.90, 5.2.91, 5.2.92, 5.2.93, 5.2.94, 5.2.95, 5.2.96, 5.2.97, 5.2.98, 5.2.99, 5.2.100

ИЗГОТОВИТЕЛЬ
ЗАО "Лорнет-М"
109472, Москва, ул. Таганская, д. 24, корп. 1.

СЕРТИФИКАТ ВЫДАН
ЗАО "Лорнет-М"
109472, Москва, ул. Таганская, д. 24, корп. 1.
Телефон/факс (495) 7721015244

НА ОСНОВАНИИ
Протокол № 10/10 от 10.04.2013 г., выдан ИД СИЗ ФБУЛ "ВНИИ Метрологии", Д.И.Менделеевский район, аккредитация № РОСС RU.0001.21СМ14.
Санитарно-эпидемиологическое заключение № 33.ВЛ.02.944.П.000459.05.08 от 21.05.2008 г., выдано Управлением Роспотребнадзора по Московской области.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ
Наличие знака соответствия по ГОСТ Р 50460-92 - на основании дополнительной документации.
Сделан в соответствии с инспекционным контролем.

Подпись: _____ И.Т.Томашевский
Засверено: _____ О.В.Иванов

Сертификат имеет юридическую силу на всей территории Российской Федерации

Рис. 4.6. Сертификат подтверждает, что очки компании ЗАО Лорнет-М являются Средством Индивидуальной Защиты (СИЗ) и соответствуют ГОСТ Р 12.4.230.1-2007.



Рис. 4.7. Сертификат на линзы очковые со спектральными фильтрами ЛС-”Лорнет-М”. Констатирует факт соответствия линз ГОСТ Р 51044-97, ГОСТ Р 51854-2001 и МС ИСО 8980-1-96.

Вывод по теме

В данном разделе были проанализированы условия труда инженера-программиста и факторы, оказывающий вредное влияние на его работу и здоровье: освещение (искусственное и естественное), микроклимат в рабочем помещении и визуальные параметры устройства отображения информации. Для оценки освещенности был проведен расчет искусственного освещения в двух точках помещения, а также расчета коэффициента естественного освещения. В результате чего было установлено, что освещение помещения соответствует требованиям безопасности. Проведенный анализ микроклимата показал, что его параметры не являются оптимальными, поэтому было предложено установить кондиционер в качестве меры по улучшению микроклимата в помещении. Нормированные визуальные

параметры устройства отображения информации соответствуют требованиям безопасности.

5. Экономическая часть. Обоснование экономической эффективности разработки библиотеки функций унификации процессов обработки входных параметров и систематизации выходных данных

5.1.Обоснование экономической эффективности разработки программного обеспечения “Библиотека функций унификации процессов обработки входных параметров и систематизации выходных данных в средствах тестирования и диагностики программных средств и оборудования.”

Перечень характеристик аналога и разрабатываемой программы согласно стандарту ISO 9126: 1991 можно увидеть в таблице 5.1.

Характеристики Качества ПП	Единица измерения	Значения характеристик качества ПП		Значимость характеристик
		Аналог	Разрабатываемый продукт	
1. Пригодность для применения	Балл	5	4	0,1
2. Понятность	Балл	3	5	0,2
3. Временная экономичность	Балл	5	6	0,4
4. Удобство для анализа	Балл	4	5	0,1
5. Адаптируемость	Балл	4	5	0,2

Таблица 5.1

Расчет интегрального показателя качества разрабатываемых алгоритмов и программных продуктов определяются по формуле:

$$J_{\text{ТУ}} = J'_{\text{ТУ}}(K_{\text{в}} + 1)$$

Индекс технического уровня проектируемого изделия:

$$J'_{\text{ту}} = \sum_{i=1}^n \frac{X_{\text{Hi}}}{X_{\text{Bi}}} \cdot \mu_i$$

где $X_{\text{Hi}}, X_{\text{Bi}}$ - уровень i -й функционально-технической характеристики соответственно базового и нового программного продукта;

μ_i - значимость i -й функционально-технической характеристики проектируемого программного продукта;

n - количество рассматриваемых функционально-технических характеристик программного продукта. Значимость i -й функционально-технической характеристики определяется экспертным путем, при этом

$$\sum_{i=1}^n \mu_i = 1$$

Значение коэффициента влияния $K_{\text{в}}$ выбирается из таблицы 5.2

Наименование проектируемой техники	$K_{\text{в}}$
Аппаратура специального назначения	0,25
Техника, улучшающая характеристики системы управления	0,25
Навигационная аппаратура	0,2
Связная аппаратура	0,15
Прочая комплектующая техника	0,15

Таблица 5.2

Разработанное устройство относится к пункту “Прочая комплектующая техника”, следовательно $K_{\text{в}}=0,15$

Подставив коэффициент в формулу для технического уровня получим:

$$J'_{\text{ту}} = \frac{4}{5} * 0,1 + \frac{5}{3} * 0,2 + \frac{6}{5} * 0,4 + \frac{5}{4} * 0,1 + \frac{5}{4} * 0,2 = 1,27$$

$$J_{\text{ту}} = J'_{\text{ту}}(K_{\text{в}} + 1) = 1,27 * (0,15 + 1) = 1,49$$

Вывод: Коэффициент больше 1, что означает целесообразность разработки данной библиотеки.

5.2.Определение трудоёмкости создания программного продукта

Определяется трудоемкость по каждой стадии работ и суммарная трудоемкость. Расчеты сведены в таблице 5.3

№ п/п	Наименование стадии (этапа) работ	Доля работ на стадии(этапе)в общем объёме работ, %
1	Анализ предметной области	3
2	Изучение средств разработки	1
3	Изучение программируемой задачи	2
4	Анализ методов решения задачи	5
5	Составление структурной схемы алгоритмов	3
6	Технико-экономическое обоснование выбранного варианта алгоритма	2
7	Уточнение и доработка выбранного варианта алгоритма	10
8	Составление программы	30
9	Отладка программы	25
10	Составление документации	10
11	Анализ работы ПП	4
12	Испытание ПП в реальных условиях	5
	ИТОГО:	100

Таблица 5.3

При традиционном программировании, когда каждый ПП содержит все этапы решения задач или комплексов задач, начиная с ввода исходных данных, и заканчивая выводом результатов, затраты труда ($t_{\text{ПР Т}}$) в чел.-час. Определяются следующим образом:

$$t_{\text{ПП}} = t_{\text{И}} + t_{\text{А}} + t_{\text{К}} + t_{\text{от}} + t_{\text{Д}}$$

Где:

$t_{\text{и}}$ - затраты труда на изучение и постановку задачи;

$$t_{\text{и}} = \frac{Q * B}{75K}$$

$t_{\text{А}}$ – затраты труда на разработку алгоритма решения задачи;

$$t_{\text{А}} = \frac{Q}{20K}$$

$t_{\text{к}}$ – затраты труда на программирование по блок-схеме;

$$t_{\text{к}} = \frac{Q}{10K}$$

$t_{\text{от}}$ – затраты труда на отладку программы;

$$t_{\text{от}} = \frac{Q}{5K}$$

$t_{\text{д}}$ – затраты труда на подготовку документации по ПП;

$$t_{\text{д}} = \frac{1,75Q}{15K}$$

Для расчетов необходимо знать:

- q - количество этапов и элементарных процедур преобразования информации; ($q=120$)
- $K_{\text{с}}$ – коэффициент сложности программы $K_{\text{с}}=1,25; \dots; 2,0$; ($K_{\text{с}}=1,4$)
- $K_{\text{к}}$ – коэффициент коррекции, при разработке $K_{\text{к}} = 0,05 \dots; 0,1$; ($K_{\text{к}}=0,065$)
- n - количество коррекций; ($n=60$)
- K – коэффициент квалификации разработчика, программиста; ($K=0,8$ стаж до 2х лет)
- $B = 1,2 \dots; 3,0$;- увеличение затрат на изучение и постановку задачи вследствие ее сложности и новизны. ($B=2$)

Таким образом, получаем условное количество операторов (строк) в машинной программе:

$$\begin{aligned} Q &= q * K_{\text{с}} * (1 + K_{k1} + \dots K_{kn}) = \\ &= 120 * 1,4 * (1 + 60 * 0,065) = 823 \end{aligned}$$

Теперь подставим значение Q в формулы для определения затрат труда:

$$t_{\text{И}} = \frac{Q * B}{75K} = \frac{823 * 2}{75 * 0,8} = 27 \text{ (чел. -час)}$$

$$t_{\text{А}} = \frac{Q}{20K} = \frac{823}{20 * 0,8} = 51 \text{ (чел. -час)}$$

$$t_{\text{К}} = \frac{Q}{10K} = \frac{823}{10 * 0,8} = 103 \text{ (чел. -час)}$$

$$t_{\text{от}} = \frac{Q}{5K} = \frac{823}{5 * 0,8} = 206 \text{ (чел. -час)}$$

$$t_{\text{д}} = \frac{1,75Q}{15K} = \frac{1,75 * 823}{15 * 0,8} = 120 \text{ (чел. -час)}$$

Таким образом, суммарная трудоемкость работы составляет:

$$\begin{aligned} t_{\text{ПП}} &= t_{\text{И}} + t_{\text{А}} + t_{\text{К}} + t_{\text{от}} + t_{\text{д}} = 27 + 51 + 103 + 206 + 120 = \\ &= 507 \text{ (чел.-час.)} \end{aligned}$$

5.3. Календарное планирование.

Календарное планирование работ по созданию программного продукта осуществляется согласно директивному графику. Разработка календарного плана производится на основе данных о трудоемкости работ, связанных с выполнением дипломного проекта. Окончательно структуру трудоемкости отдельных этапов определяют, используя данные о видах работ, подлежащих выполнению.

Производственный цикл каждого этапа:

$$T_{\text{ци}} = \frac{T_{\text{эи}}}{t_{\text{р\delta}} * q},$$

где $T_{\text{эи}}$ – трудоемкость этапа, чел.-ч.;

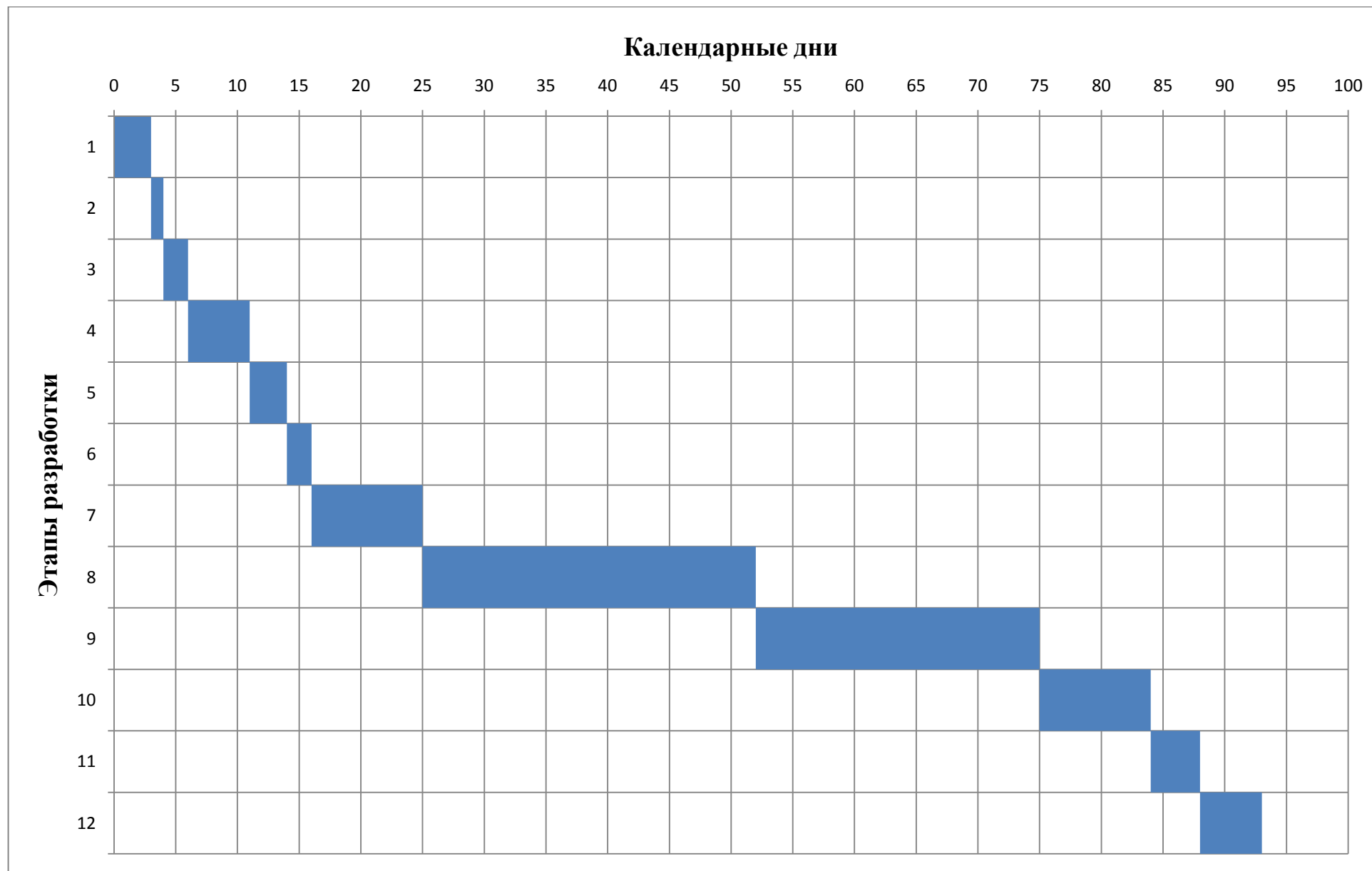
$t_{\text{р\delta}}$ – продолжительность рабочего дня (8 часов);

q - количество работников, одновременно участвующих в выполнении работ, чел.

Результаты сведены в таблицу 5.4

№ п/п	Наименование стадии (этапа) работ	Удельный вес, %	Трудоемкость, чел.-час.	Производственный цикл, календарные дни	Длительность этапа, календарные дни
1.	Анализ предметной области	3	15	1,875	3
2.	Изучение средств разработки	1	5	0,625	1
3.	Изучение программируемой задачи	2	10	1,25	2
4.	Анализ методов решения задачи	5	26	3,25	5
5.	Составление структурной схемы алгоритмов	3	15	1,875	3
6.	Технико-экономическое обоснование выбранного варианта алгоритма	2	10	1,25	2
7.	Уточнение и доработка выбранного варианта алгоритма	10	50	6,25	9
8.	Составление программы	30	152	19	27
9.	Отладка программы	25	127	15,875	23
10.	Составление документации	10	51	6,375	9
11.	Анализ работы ПП	4	20	2,5	4
12.	Испытание ПП в реальных условиях	5	26	3,25	5
	Итого:	100	507	63,375	93

Таблица 5.4



5.4.Определение затрат на создание программного продукта

Зарплата персонала по стадиям работ рассчитывается по формуле:

$$З_{зпj} = T_j \tau_j$$

Где: T_j - трудоемкость j-ой стадии работы в чел.-час;

τ_o - средняя дневная ставка оплаты работ j-ой стадии работы.

Результаты расчетов затрат на оплату труда сведены в таблицу 5.5:

№ стадии	Трудоемкость стадии, чел.- час	Исполнители		Часовая ставка, руб.	Средняя часовая ставка, руб.	Заработная плата, руб.
		Должность	Численность, чел.			
1	15	Инженер- программист	1	245	245	3675
2	5	Инженер- программист	1	245	245	1225
3	10	Инженер- программист	1	245	245	2450
4	26	Инженер- программист	1	245	245	6370
5	15	Инженер- программист	1	245	245	3675
6	10	Инженер- программист	1	245	245	2450
7	50	Инженер- программист	1	245	245	12250
8	152	Инженер- программист	1	245	245	37240
9	127	Инженер- программист	1	245	245	31115
Итого:						124215

Таблица 5.5

Состав затрат на создание программного продукта приведен в таблице

5.6:

№ п/п	Наименование статей затрат	Затраты, руб	Удельный вес, %
1	Заработная плата основных исполнителей	149058	39
2	Отчисления на соц. нужды	39053,196	10
3	Накладные расходы	193775,4	51
ИТОГО:		$З_{ПП}=381886,596$	100

Таблица 5.6

Величина заработной платы основных исполнителей разработки ПП является итогом таблицы 5.5, увеличенным на процент премиальных выплат (20%)

Норматив отчислений на социальные нужды составляет 26,2% от заработной платы основных исполнителей с учетом премий.

Величина накладных расходов определяются по отношению к заработной плате основных исполнителей:

$$P_{\text{накл}} = Z_{\text{ЗП}} K_{\text{накл}}$$

Где $K_{\text{накл}}$ - коэффициент накладных расходов; для ПП принимается на уровне: $1 \leq K_{\text{накл}} \leq 2,0$

Таким образом

$$P_{\text{накл}} = 149058 * 1,3 = 193775,4 \text{ руб.}$$

Цена первоначальной продажи разработанного программного продукта определяются с учетом рентабельности разработки по формуле:

$$C_{\text{ПП}} = Z_{\text{ПП}} + ЗП_{\text{ПП}} * \rho_{\text{ЗП}}/100$$

где $Z_{\text{ПП}}$ – текущие затраты на создание ПП;

$ЗП_{\text{ПП}}$ – оплата труда персонала в общих текущих затратах на создание ПО;

$\rho_{\text{ЗП}}$ – уровень рентабельности (прибыли по отношению к оплате труда персонала), обеспечивающий безубыточность деятельности ($\rho_{\text{ЗП}} = 200-400\%$).

Примем $\rho_{\text{зп}} = 200\%$, тогда цена разрабатываемого программного продукта будет равна:

$$C_{\text{пп}} = 381887 + 149058 * 200/100 = 680000 \text{ руб.}$$

5.5. Оценка экономической эффективности

Разработка дипломных проектов любого типа может быть связана с разработкой программного продукта с различным целевым назначением. ПП может быть использован для:

- проведения исследований при выполнении каких-либо разработок прикладного характера;
- для разработки интерфейса с применением интегральных схем на основе ПП, как части измерительного комплекса или комплексов по приему, преобразованию, передаче информации;
- для отладки и настройки РЭС;
- при эксплуатации – как реализация алгоритмов ПП для различных расчетов при управлении какими-либо объектами, при анализе и оценке информации в процессе реализации процедур по диагностике, процедур приема, преобразования, передачи информации и др.

В любом случае для оценки экономической эффективности алгоритмов и ПП требуется выполнение соответствующих расчетов. В процессе выполнения этих расчетов необходимо провести оценку целесообразности проведения разработки ПП, оценку капитальных и текущих затрат, определить уровень эффективности и срок окупаемости вложений в ПП.

Для оценки экономической эффективности создаваемых алгоритмов и ПП необходимо выяснить механизм их действия на экономические показатели в сферах применения ПП. В связи с различными направлениями использования ПП имеет место разнообразие методических подходов к оценке показателя годового экономического эффекта $\mathcal{E}_{\text{пп г}}$.

Т.к. разрабатываемый ПП будет использован для унификации процессов обработки входных параметров и систематизации выходных данных в средствах тестирования и диагностики программных средств и оборудования и позволит повысить качество этой диагностики, то $\mathcal{E}_{\text{ПП Г}}$ определяется укрупнено на основе анализа динамики эксплуатационных затрат по отношению к повышению качества диагностики:

$$\mathcal{E}_{\text{ПП Г}} = \mathcal{Z}_{\text{ЭКС Д Б}}(K_{\text{ОП ЭКС Д}} - 1)N_{\text{Д}}$$

Где:

$\mathcal{Z}_{\text{ЭКС Д Б}}$, $\mathcal{Z}_{\text{ЭКС Д Н}}$ – эксплуатационные затраты на выполнение диагностических процедур для одного объекта, до применения ПП и в новом варианте:

$$\mathcal{Z}_{\text{ЭКС Д}} = \mathcal{Z}_{\text{ПЭ}} + A_{\text{ВТ Г}} + \mathcal{Z}_{\text{ЭЛ}} + \mathcal{Z}_{\text{ПР}}$$

Где:

$\mathcal{Z}_{\text{ПЭ}}$ - затраты на оплату труда персонала, осуществляющего диагностику

$A_{\text{ВТ Г}}$ - годовые амортизационные отчисления по вычислительной технике

$\mathcal{Z}_{\text{ЭЛ}}$ - затраты на электроэнергию по вычислительной технике

$\mathcal{Z}_{\text{ПР}}$ - прочие затраты

$$0,15 * \mathcal{Z}_{\text{ПЭ}} \leq \mathcal{Z}_{\text{ПР}} \leq 0,2 * \mathcal{Z}_{\text{ПЭ}}$$

$N_{\text{Д}}$ - количество объектов, диагностируемых за год

$K_{\text{ОП ЭКС Д}}$ - коэффициент опережения повышения качества диагностики по сравнению с ростом эксплуатационных затрат

$$K_{\text{ОП ЭКС Д}} = \frac{J_{\text{Д}}}{J_{\text{ЭКС}}}$$

Где:

$J_{\text{ЭКС}}$ - индекс изменения эксплуатационных затрат

$$J_{\text{ЭКС}} = \frac{\mathcal{Z}_{\text{ЭКС Д Н}}}{\mathcal{Z}_{\text{ЭКС Д Б}}}$$

$J_{\text{Д}}$ - уровень качества диагностики

Выполним расчеты:

- количество объектов, диагностируемых за год

$$N_{\text{д}} = 300$$

- годовые амортизационные отчисления по вычислительной технике

$$A_{\text{ВТГ}} = C_{\text{ВТ}} \frac{H_{\text{АВТ}}}{100} d_{\text{ис}}$$

Где:

- $C_{\text{ВТ}}$ – стоимость вычислительной техники
- $C_{\text{ВТ}} = 6 * 12000 = 72000$ руб.
- $H_{\text{АВТ}}$ – годовая норма амортизационных отчислений (25 %)
- $d_{\text{ис}}$ – коэффициент использования мощности информационной системы для решения данной задачи

$$d_{\text{ис}} = \frac{T_{\text{М.Г.}}}{F_{\text{эфф.ВТ}}} = \frac{43 \text{ час.}}{2085 \text{ час.}} = 0,0206$$

$T_{\text{М.Г.}}$ – машинное время, используемое в течение года для реализации данного ПП, час.;

$F_{\text{эфф.ВТ}}$ – годовой эффективный фонд времени работы вычислительной техники, час.

Таким образом:

$$A_{\text{ВТГ}} = C_{\text{ВТ}} \frac{H_{\text{АВТ}}}{100} d_{\text{ис}} = 72000 * 0,25 * 0,0206 = 371 \text{ руб.}$$

- затраты на электроэнергию

$$З_{\text{эл}} = W F_{\text{эфф.ВТ}} C_{\text{эл}} d_{\text{ис}} = 1,8 * 2085 * 2,05 * 0,0206 = 158,67 \text{ руб.}$$

Где:

W – мощность вычислительной техники, кВт·час (0,3*6)

$C_{\text{эл}}$ – стоимость одного кВт·ч электроэнергии, руб. (2,05)

- затраты на оплату труда персонала, осуществляющего диагностику одного объекта, руб.:

- Затраты в базовом варианте

$$З_{\text{Пэ}} = 14 \text{ тыс. руб.}$$

- Затраты в новом варианте

$$З_{ПЗ} = 11600 \text{ руб.}$$

- прочие затраты, руб.:

- Затраты в базовом варианте

$$З_{ПР} = З_{ПЗ} * 0,175 = 2450 \text{ руб.}$$

- Затраты в новом варианте

$$З_{ПР} = З_{ПЗ} * 0,175 = 2030 \text{ руб.}$$

$$З_{ЭКС Д Б} = 14000 + 371 + 158,67 + 2450 = 16979,67 \text{ руб.}$$

$$З_{ЭКС Д Н} = 11600 + 371 + 158,67 + 2030 = 14159,67 \text{ руб.}$$

$$J_{ЭКС} = \frac{З_{ЭКС Д Н}}{З_{ЭКС Д Б}} = \frac{14159,67}{16979,67} = 0,83$$

Признаки эксплуатационного эффекта диагностики	Единица измерения	Значения характеристик качества ПП		Значимость характеристик
		Аналог	Разрабатываемый продукт	
1. Средняя продолжительность процедуры диагностики	Балл	5	6	0,5
2. Автоматизация процесса диагностики	Балл	2	5	0,3
3 Простота использования	Балл	3	5	0,2

Таблица 5.7

$$J_D = \sum_{i=1}^m \alpha_i \left(\frac{X_{Hi}}{X_{Bi}} \right) = 0,5 * \frac{6}{5} + 0,3 * \frac{5}{2} + 0,2 * \frac{5}{3} = 1,68$$

$$K_{ОП ЭКС Д} = \frac{J_D}{J_{ЭКС}} = \frac{1,68}{0,83} = 2,02$$

$$\Delta_{ПП Г} = З_{ЭКС Д Б} (K_{ОП ЭКС Д} - 1) * N_D =$$

$$= 16979,67 * (2,02 - 1) * 300 = 5195779,02 \text{ руб.}$$

Уровень экономической эффективности затрат на разработку ПП при использовании его результатов в сфере производства

$$E = \frac{\Delta_{\text{ППГ}} * \beta}{\text{Ц}}$$

Где:

β – коэффициент долевого участия разработчика. ($\beta = 0,2$)

Ц – цена нового продукта

Ц = 680000 руб. (см. п.3.4)

Тогда:

$$E = \frac{\Delta_{\text{ППГ}} * \beta}{\text{Ц}} = \frac{5195779,02 * 0,2}{680000} = 1,52$$

Рассчитаем срок окупаемости затрат:

$$T_{\text{ок}} = \frac{1}{E} = \frac{1}{1,52} = 0,66 \text{ года}$$

Выводы: Показатель годового экономического эффекта при разработке данного программного продукта равен 5,2 млн. руб. Срок окупаемости проекта составляет 0,66 года (≈ 8 месяцев).

С такими экономическими показателями разработка проекта весьма целесообразна.

6. Заключение

В ходе создания библиотеки унификации получения входных параметров и систематизации выходных данных были рассмотрены опции компилятора *Gcc* для компиляции программ под разные архитектуры процессоров. Также согласно требованиям из исходных данных были проанализированы основные отличия языка Си со спецификацией C99 от предшествующей спецификации C89.

Функции работы с ошибками библиотеки, функция инициализации, функции получения входных параметров, функции обработки выходных данных были спроектированы, реализованы и интегрированы в библиотеку. Для редактирования текста программы использовался редактор *Vim*. Также при разработке применялась система контроля версий *Git*.

Для разработанного набора средств получения входных параметров и систематизации выходных данных была проведена демонстрация основных возможностей на примере программы тестирования функции нахождения корней квадратного уравнения и программы, тестирующей работоспособность COM-порта.

Было проведено профилирование разработанной библиотеки, целью которого было не только увеличение общей производительности, но и выявление утечек памяти и ошибок, связанных с манипуляцией данными в памяти. В качестве профилировщика использовалась программа *Valgrind*.

Каждый раз при добавлении новых возможностей, для всех нетривиальных функции библиотеки проводилось автоматическое модульное тестирование по принципу черного ящика с целью обнаружения ситуаций, в которых новый функционал препятствовал работе функций, добавленных в библиотеку ранее. Все выявленные ошибки, связанные с логикой работы функций были отлажены.

В разделе по охране труда и окружающей среды был проведен анализ освещения (естественного и искусственного) и микроклимата помещения, в

котором происходила разработка программного обеспечения. А также проанализированы визуальные параметры устройства отображения информации (монитора Samsung SyncMaster S27A550H).

В разделе обоснования экономической эффективности разработки программного обеспечения был рассчитан показатель годового экономического эффекта при разработке данного программного продукта, который равен 5,2 млн. руб. Срок окупаемости проекта составляет 0,66 года (≈ 8 месяцев).

Данный вариант библиотеки libtio не является окончательным. В будущем планируется расширение функциональных возможностей этого программного продукта, а также доработка уже имеющихся функций, которая может потребоваться при дальнейшем использовании библиотеки в реальных условиях.

Список литературы

- 1) *Нейл М., Ричард С.* Основы программирования в Linux. – 2-е изд. – СПб: «БХВ-Петербург», 2009. - 881с
- 2) *Керниган Б., Ритчи Д.* Язык программирования С. - 2-е изд. –М: «Вильямс», 2009. – 292 с.
- 3) *Скотт Ч.* Pro Git. - «Apress», 2012. - 272 с.
- 4) *Сибаров Ю.Г. и др.* Охрана труда в вычислительных центрах. Учебник. – М.: Машиностроение, 1990.
- 5) *Березин В.М., Дайнов М.И.* Защита от вредных производственных факторов при работе на ПЭВМ. – М.: Изд. МАИ, 2003.
- 6) ГОСТ 12.1.005-88 «Общие санитарно-гигиенические требования к воздуху рабочей зоны».
- 7) СНиП 23-05-95 «Естественное и искусственное освещение».
- 8) СанПиН 2.2.2./2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организация работы».
- 9) СанПиН 2.2.4.548–96 «Физические факторы производственной среды гигиенические требования к микроклимату производственных помещений»

Приложение 1

Исходные файлы функций библиотеки

Файл init.c

```
#include <errno.h>
#include <string.h>
#include <stdio.h>

#include <tioinit.h>
#include <tioerror.h>
#include <tiowerror.h>
#include <tio.h>

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <init_msg.h>

#define MAXARGS 100

char *tio_argv[MAXARGS] = {0};

char *selfname = NULL;

char *True="True";
char *False="False";

tio_param      TIOPARAMNULL = { NULL, NULL, NULL };
tio_param_rec *Tio_params = NULL;
size_t         Tio_params_sz = 0;

tio_key_string *longkeys = NULL; /**< Временный массив коротких ключей
                                   * ( используется для поиска ) */
tio_key_string *shortkeys = NULL; /**< Временный массив длинных ключей
                                   * ( используется для поиска ) */

tio_lib_state tio_self_state [] = {
    {"--tio-debug", 0, TIODEBUG, NULL },
    {"--tio-file" , 0, TIOFILETAR, NULL},
    { NULL, 0, 0, NULL }
};

/// Проверяет является-ли tio_param нулевым
static inline int is_paramNULL(const tio_param* obj)
{
    if ( obj->description == NULL &&
        obj->name == NULL &&
        obj->key == NULL )
        return 1;
    return 0;
}

/// Сортировка параметров по имени
static int tio_params_sort(tio_param *obj)
{
    tio_param *i, *j, k;

    for ( i = obj; !is_paramNULL(i); ++i)
        for ( j = i+1; !is_paramNULL(j); ++j)
        {
            if ( strcmp(i->name, j->name) > 0 )
            {
                k=i;
                i=j;
                j=k;
            }
        }
}
```

```

    return 0;
}

/**
 * Создание копии строки.
 * Память из под строки заем необходимо освободить вызовом free
 * @return Указатель на новую строку или NULL в случае ошибки
 */
static char* newstring(const char* obj)
{
    size_t sz = strlen(obj);
    char *ret;
    if (!obj)
        return NULL;
    ret = malloc(sz+1);
    if(!ret)
        return NULL;
    strcpy(ret, obj);
    return ret;
}

/**
 * Преобразование ключей в полную форму и первичный разбор на длинные
 * и короткие.
 * @param obj Массив ключей - изменяется во время работы.
 * @param sz Количество ключей
 * @return 0 при успешном выполнении или код ошибки в противном случае.
 */
static int decode_keys(tio_key_string *obj, size_t sz)
{
    size_t i, ll;
    char buff[TIOMAXKEY+3];
    if (!obj)
        return EINVAL;

    for ( i = 0; i < sz; ++i)
    {
        ll = strlen(obj[i].str);
        if (!ll || ll > TIOMAXKEY)
            return EINVAL;
        if( 1 == ll )
        {
            obj[i].symb=obj[i].str[0];
            free(obj[i].str);
            obj[i].str=NULL;
            obj[i].per->has_key = 0;
        }
        else if ( 2 == ll)
        {
            if (obj[i].str[0]=='-')
            {
                buff[0] = buff[1] = '-';
                buff[2] = obj[i].str[1];
                buff[3] = '\0';
                obj[i].per->has_key = 0;
                if (!(obj[i].str = newstring(buff)))
                    return ENOMEM;
                obj[i].symb='\0';
            }
            else if (obj[i].str[1] == ':')
            {
                obj[i].symb = obj[i].str[0];
                free(obj[i].str);
                obj[i].str = NULL;
                obj[i].per->has_key = 1;
            }
            else
            {
                buff[0]=buff[1] = '-';
                buff[2] = '\0';
            }
        }
    }
}

```

```

        strcpy(buff+2, obj[i].str);
        obj[i].per->has_key=0;
        obj[i].symb='\0';
        free(obj[i].str);
        if (!(obj[i].str = newstring(buff)))
            return ENOMEM;
    }
}
else
{
    obj[i].symb='\0';
    buff[0] = buff[1] = '-';
    if (3 == 11 && obj[i].str[0]=='-' && obj[i].str[2]==':')
    {
        buff[2] = obj[i].str[1];
        buff[3] = '\0';
        obj[i].per->has_key = 1;
        free(obj[i].str);
        if (!(obj[i].str = newstring(buff)))
            return ENOMEM;
    }
    else
    {
        if (obj[i].str[11-1] == ':')
        {
            obj[i].per->has_key=1;
            obj[i].str[11-1] = '\0';
        }
        else
        {
            obj[i].per->has_key=0;
            strcpy(buff + 2, obj[i].str);
            free(obj[i].str);
            if (!(obj[i].str=newstring(buff)))
                return ENOMEM;
        }
    }
}
return 0;
}

/**
 * Сортировка длинных ключей в порядке который позволяет при прямом
 * переборе от младшего индекса к старшему исключить что более короткий
 * ключ перекроет более длинный
 *
 * @param[in,out] obj Массив подвергающийся сортировки.
 *
 * @return 0 в случае успеха и ненуль при ошибке.
 */
static int sortkeys(tio_key_string *obj)
{
    tio_key_string t, *j, *p = obj;
    for(; p->per; ++p)
        for( j = p; j->per; ++j)
        {
            if(!j->str)
                return EINVAL;
            if(strcmp(p->str, j->str)<0)
            {
                t=*j;
                *j=*p;
                *p=t;
            }
        }
    return 0;
}

/**
 * Разделение на длинные и короткие ключи
 *
 * Производит заполнение массивов shortkeys longkeys завершая их
 * записью содержащей в поле per значение NULL. Записи str в longkey

```

```

* заполняются строками размещенными в heap и должны быть высвобождены
* при помощи вызова free по окончании использования.
*
* Внимание строки из массива obj не должны быть освобождены так как
* они будут использованы при работе с shortcuts и longkeys и
* будут освобождены при завершении работы с библиотекой.
*
* @param obj Исходный массив данных
*
* @param sz Колличество элементов входного массива
*
* @return 0 в случае успеха и ненулевое значение при ошибке
*/
static int splitkeys( tio_key_string *obj, const size_t sz)
{
    size_t lng = 0;
    size_t srt = 0;
    tio_key_string *i, *j, *k;
    tio_key_string *last = (tio_key_string*)(obj + sz);

    for ( i = obj; i < last ; ++i)
    {
        if (i->symb == '\0')
            ++lng;
        else
            ++srt;
    }
    ++srt, ++lng;
    if (!(shortkeys=(tio_key_string*)malloc((srt+1)*sizeof(tio_key_string))))
        return errno;
    if (!(longkeys=(tio_key_string*)malloc((1+lng)*sizeof(tio_key_string))))
        return errno;

    for ( i = obj, j = shortkeys, k = longkeys; i < last; ++i)
    {
        if (i->symb == '\0')
        {
            *k=i;
            ++k;
        }
        else if(i->str == NULL)
        {
            *j=i;
            ++j;
        }
    }
    j->per = NULL;
    k->per = NULL;
    return 0;
}

/**
* Связывание расшифрованных ключей и параметров.
*
* Функция производит обратное связывание заполняя поля keys и skeys
*
* @param keys Сгруппированный массив структур tio_key_string в котором
* если два элемента ссылаются на родительский элемент а между ними не
* может быть элемента сылающегося на не а.
*
* @param ksz Длинна массива keys
*
* @return Код ошибки при провале или ноль в случае успеха.
*/
static int reassing_keys(tio_key_string *keys, size_t ksz)
{
    tio_key_string *i, *last = keys + ksz;
    tio_param_rec **mas, *ptr = NULL;
    size_t *shrtk;
    size_t *longk;
    size_t cnts = 0;
    size_t cntl = 0;
    size_t offset = 0;

    if (!(shrtk=malloc(sizeof(size_t)*(Tio_params_sz+1))))

```

```

    return errno;
if(!(longk=malloc(sizeof(size_t)*(Tio_params_sz+1))))
{
    int err = errno;
    free(shrtk);
    errno = err;
    return err;
}
if(!(mas = malloc(sizeof(tio_param_rec*)*Tio_params_sz)))
{
    int err = errno;
    free(shrtk);
    free(longk);
    errno = err;
    return err;
}
ptr = keys->per;
for ( i = keys; i < last; ++i )
{
    if (i->per != ptr )
    {
        shrtk[offset]=cnts;
        longk[offset]=cntl;
        mas[offset]=ptr;
        cntl=cnts=0;
        ++offset;
        ptr=i->per;
    }
    if (i->str != NULL)
        ++cntl;
    else if (i->symb != '\0')
        ++cnts;
    else
    {
        free(shrtk);
        free(longk);
        fputs(INIT_REASSING_KEYS_ERROR1, stderr);
        return -1;
    }
}
shrtk[offset]=cnts;
longk[offset]=cntl;
mas[offset]=ptr;
for ( offset = 0 ; offset < Tio_params_sz; ++offset)
{
    Tio_params[offset].val = NULL;
    if (longk[offset]==0)
        Tio_params[offset].keys=NULL;
    else
        if(!(Tio_params[offset].keys=malloc(sizeof(char*)*(longk[offset]+1))))
        {
            free(shrtk);
            free(longk);
            return ENOMEM;
        }
    if(shrtk[offset]==0)
        Tio_params[offset].skeys=NULL;
    else
        if(!(Tio_params[offset].skeys=malloc(shrtk[offset]+1)))
        {
            free(shrtk);
            free(longk);
            return ENOMEM;
        }
}
cntl = cnts = 0;
ptr = keys->per;
for ( i = keys; i < last ; i++ )
{
    if (i->per != ptr)
    {
        if (ptr->keys != NULL)
            ptr->keys[cntl]=NULL;
        if(ptr->skeys != NULL)
            ptr->skeys[cnts]='\0';
        cntl=cnts=0;
    }
}

```

```

        ptr=i->per;
    }
    if (i->str != NULL)
        ptr->keys[cntl++] = i->str;
    else
        ptr->skeys[cnts++] = i->symb;
    }
    free(mas);
    free(shrtk);
    free(longk);
    return 0;
}

/**
 * Разбор введенных параметров, перенос из входной структуры в
 * tio_param_rec с преобразованием записи в естественный вид ключа и
 * группировкой по имени параметра.
 *
 * @param params Список параметров переданных а tioInit.
 *
 * @return 0 при успешном завершении и ненулевое значение в противном
 * случае.
 */
int tio_decode_params(tio_param *params)
{
    tio_param *i;
    tio_param_rec *j;
    tio_key_string *keys, *k;
    size_t cnt = 0;
    size_t fsz = 0;

    // if ( (!params) || is_paramNULL(params) )
    if ( (!params) )
        return EINVAL;
    if (is_paramNULL(params))
        return 0;
    tio_params_sort(params);

    for( i = params; !is_paramNULL(i); ++i, ++fsz)
        if(i->description) cnt++;

    if ( !(Tio_params = malloc(sizeof(tio_param_rec)*cnt)))
        return ENOMEM;
    Tio_params_sz = cnt;
    if ( !(keys = malloc(sizeof(tio_key_string)*(fsz+1))))
    {
        free(Tio_params);
        return ENOMEM;
    }

    Tio_params[0].name=newstring(params->name);
    for( i = params, j = Tio_params, k=keys; !is_paramNULL(i); i++, ++k )
    {
        if ( j->name == NULL || strcmp(j->name, i->name))
        {
            ++j;
            j->name=newstring(i->name);
        }
        k->str = newstring(i->key);
        if (i->description)
            if ( !(j->description = newstring(i->description)))
                return ENOMEM;
        k->per = j;
    }
    // Связали ключи с именами удалив лишние имена - теперь есть два
    // массива - массив имен и массив строк
    if (decode_keys(keys, fsz))
        return ENOMEM;
    if (splitkeys(keys, fsz))
        return ENOMEM;
}

```



```

    if (reassing_keys(keys, fsz))
        return ENOMEM;
    if(sortkeys(longkeys))
        return ENOMEM;
    free(keys);
    return 0;
}

```

```

/**
 * Функция извлечения параметров библиотеки из переданных программных параметров
 *
 * @param argc Количество параметров передаваемых функции
 *
 * @param argv Массив параметров для обработки
 *
 * @param[out] cnt Количество оставшихся элементов
 *
 * @return Массив строк с параметрами не подпадающими под шаблоню Все
 * эти строки должны быть освобождены вызовом free, как и сам массив
 * строк.
 */

```

```

static char** extrude_tio(const int argc, const char** argv, int* cnt)
{
    char** ptr;
    tio_simple_chain *par=NULL;
    tio_lib_state *p = tio_self_state;
    tio_simple_chain *tp = NULL;
    int i;

    *cnt = 0;
    for (i=0; i<argc; i++)
    {
        if (!strncmp("--tio-", argv[i], 6))
        {
            for ( p = tio_self_state; p->key != NULL; ++p)
            {
                size_t len = strlen(p->key);
                if (!strncmp(p->key, argv[i], len))
                {
                    p->set=1;
                    if (argv[i][len]=='=')
                    {
                        if(argv[i][len+1]!='\0')
                            p->value = newstring(argv[i]+len+1);
                        else if (++i < argc )
                            p->value = newstring(argv[i]);
                        else
                        {
                            errno=EINVAL;
                            return NULL;
                        }
                    }
                }
                else if (i+1 < argc && argv[i+1][0]=='=')
                {
                    if (argv[++i][1]!='\0')
                    {
                        if (i+1 < argc )
                            p->value = newstring(argv[++i]);
                        else
                        {
                            errno=EINVAL;
                            return NULL;
                        }
                    }
                }
                else
                    p->value = newstring(argv[i]+1);
            }
            if (i==argc)
            {
                p->value = NULL;
            }
        }
        *cnt++;
    }
}

```

```

    }
}
else
{
    tp = NULL;
    if (!(tp = malloc(sizeof(tio_simple_chain))))

        return NULL;
    tp->val = newstring(argv[i]);
    tp->next = par;
    par = tp;
    ++(*cnt);
}
}
// for (tp=par, *cnt=0; tp!=NULL; tp = tp->next, ++(*cnt));
if (!(ptr = malloc(sizeof(char*) * *cnt)))
    return NULL;
for (i = *cnt, tp = par; i>0;)
{
    ptr[--i] = tp->val;
    par = tp;
    tp = tp->next;
    free(par);
}
errno=0;
return ptr;
}

static int extractparams(int start, int argc, char** argv)
{
    size_t i;
    size_t cnt=0;
    tio_simple_chain *ptr = NULL;
    tio_simple_chain *pt = NULL;
    tio_key_string *p;

    for( int nfi = 0; nfi < MAXARGS; ++ nfi)
        tio_argv[nfi] = malloc( sizeof(char) * 100);

    for (i=start; i < argc; i++)
    {
        if (argv[i][0]=='-')
        {
            if (argv[i][1]=='-')
            {
                for (p=longkeys; p->per != NULL; ++p)
                {
                    size_t k=strlen(p->str);
                    if (!strncmp(p->str, argv[i], k))
                    {
                        if (!(p->per->has_key) && strlen(argv[i])!=k)
                            fprintf(stderr, INIT_UNKNOWN_PARAM, argv[i]);
                        else if(p->per->has_key)
                        {
                            if (argv[i][k]=='=')
                            {
                                if (strlen(argv[i])>k+1)
                                    p->per->val=newstring(argv[i]+k+1);
                                else
                                    p->per->val=newstring(argv[++i]);
                            }
                        }
                        else if (argc - i > 1 && argv[i+1][0]!='=')
                        {
                            if (argv[++i][1]!='\0')
                                p->per->val=newstring(argv[++i]);
                            else
                                p->per->val=newstring(argv[i]+1);
                        }
                    }
                }
            }
            else
            {
                fprintf(stderr,
                    INIT_KEY_WITHOUT_PARAM,

```

```

        argv[i]);
        return -1;
    }
    }
    else
        p->per->val=True;
        break;
    }
}
if (p->per==NULL)
{
    fprintf(stderr, INIT_UNKNOWN_KEY, argv[i]);
}
}
else
{
    int j, sz = strlen(argv[i]);
    int brk = 0;
    for (j=1; !brk && j<sz; ++j)
    {
        for(p=shortkeys; p->per!=NULL; ++p)
        {
            if(p->symb == argv[i][j])
            {
                if(p->per->has_key)
                {
                    if (argv[i][j+1]!='\0')
                        p->per->val = newstring(argv[i]+j+1);
                    else
                    {
                        ++i;
                        if (i < argc)
                            p->per->val = newstring(argv[i]);
                        else
                        {
                            fprintf(stderr,
                                INIT_KEY_WITHOUT_PARAM2,
                                argv[i-1][j]);
                            return -1;
                        }
                    }
                }
                brk=1;
                break;
            }
            else
            {
                p->per->val = True;
                break;
            }
        }
        if (p->per==NULL)
            fprintf(stderr, INIT_UNKNOWN_KEY2, argv[i][j]);
    }
}
}
}
else // argv[i][0]=='-'
{
    if(!(pt=malloc(sizeof(tio_simple_chain))))
        return -1;
    pt->next = ptr;
    ptr=pt;
    pt->val = argv[i];
    cnt++;
}
}
// Обратный ход - сбор неиспользованных параметров;
if (cnt>=MAXARGS)
{
    fprintf(stderr,INIT_TO_MUCH_ARGUMENTS);
    return -1;
}
for (i = cnt; i>0;)
{

```

```

        strcpy( tio_argv[--i], pt->val );
        /*tio_argv[--i]=pt->val;*/
        pt=pt->next;
        free(ptr);
        ptr=pt;
    }
    return 0;
}

int tioInit(const char* version, const char* help,
            const tio_param param[], const int argc, const char* argv[] )
{
    char **mass;
    int cnt, i;
    int res;
    int *ptr=&cnt;
    tio_param *tmpar;

    if((res = tioErrorInit()))
    {
        fputs(INIT_INIT_FAIL, stderr);
#ifdef INITRETURN
        return res;
#else
        exit(res);
#endif
    }
    for(cnt=0;!is_paramNULL(param+cnt);++cnt);
    if(!(tmpar=malloc(sizeof(tio_param)*(++cnt))))
    {
        fputs(INIT_MEMORY_ERROR, stderr);
#ifdef INITRETURN
        return TEEPICFAIL;
#else
        exit(TEEPICFAIL);
#endif
    }
    memcpy(tmpar, param, sizeof(tio_param) * cnt);

    if (!version || !help || !argv || !(argc > 0) || tio_decode_params(tmpar))
    {
        fputs(INIT_INIT_PARAM_ERROR, stderr);
#ifdef INITRETURN
        return EINVAL;
#else
        exit(EINVAL);
#endif
    }

    for(i=1; i<argc; ++i)
    {
        if (!strcmp(argv[i], "--help"))
        {
            tioHelp(help, argv[0], Tio_params, Tio_params_sz);
            exit(0);
        }
        if(!strcmp(argv[i], "--version"))
        {
            printf(INIT_VERSION_INFO, argv[0], version);
            exit(0);
        }
    }

    selfname = newstring(argv[0]);
    fprintf(stdout, INIT_RUN_MSG, argv[0]);

    if(!(mass = extrude_tio(argc-1, argv + 1, ptr))&&errno)
    {
        puts(strerror(errno));
        fputs(INIT_MEMORY_ERROR, stderr);
#ifdef INITRETURN
        return TEFAIL;
#else
        exit(TEFAIL);
#endif
    }
}

```

```

#endif
}

    if (extractparams(0, cnt, mass))
    {
#ifdef NDEBBUG
        fputs(INIT_READ_PARAM_ERROR, stderr);
#endif
#ifdef INITRETURN
        return TEFAIL;;
#else
        exit(TEFAIL);
#endif
    }
    for (i = 0; i < cnt; ++i)
        free(mass[i]);
    free(mass);
    free(tmpar);

    return 0;
}

/**
 * Функция освобождения ресурсов выделенных при запуске библиотеки
 */
void tioFree()
{
    tio_key_string *p;
    size_t i;

    tioErrorFree();

    if(longkeys)
        for ( p = longkeys; p->per != NULL; ++p )
            free(p->str);
    for (i=0 ; i < Tio_params_sz; ++i)
    {
        if (Tio_params[i].keys)
            free(Tio_params[i].keys);
        if(Tio_params[i].skeys)
            free(Tio_params[i].skeys);
        if(Tio_params[i].has_key && Tio_params[i].val)
            free(Tio_params[i].val);
        free(Tio_params[i].name);
        free(Tio_params[i].description);
    }
    if (Tio_params)
        free(Tio_params);
    if(longkeys)
        free(longkeys);
    if(shortkeys)
        free(shortkeys);
}

```

Файл help.c

```

#include <stdio.h>
#include <string.h>
#include <tioinit.h>
#include <utf.h>

#include <help_msg.h>

#define MAX_TAB 4
#define MAX_DES 50
#define SIZE 100

int tioHelp( const char* help_msg, const char* progName,
             const tio_param_rec par[], const size_t sz )
{
    printf( HELP_USAGE, progName );
}

```

```

puts( help_msg );

int idx, i;
int nolong = 0;
size_t lskeys;
int counter;
for( idx = 0 ; idx < sz ; ++idx )
{
    nolong = 0;
    counter = MAX_TAB;
    // Вывод короткого ключа
    if( par[idx].skeys )
    {
        if( !par[idx].keys )
            nolong = 1;
        else nolong = 0;
        lskeys = strlen( par[idx].skeys );

        for( i = 0; i < lskeys-1 ; ++i )
        {
            printf( " -%c,", par[idx].skeys[i] );
        }
        // Если нет длинных ключей то ставим последний короткий
        if( nolong )
            printf( " -%c", par[idx].skeys[lskeys-1] );
        else
            printf( " -%c,", par[idx].skeys[lskeys-1] );
        counter -= lskeys / 2;
    }

    // Вывод длинных ключей
    if( !nolong )
    {
        for( i = 0; par[idx].keys[i]; ++i )
        {
            if( par[idx].keys[i+1] )
                printf( " %s,", par[idx].keys[i] );
            else
                printf( " %s", par[idx].keys[i] );
            --counter;
        }
    }

    //Если нужен для данного ключа параметр
    if( par[idx].has_key )
    {
        printf(HELP_PARAM);
        counter -= 2;
    }
    for( i = 0; i <= counter; ++i )
        printf("\t");

    // Разбор параметра описания ключей
    size_t lenDes = utf_strlen( par[idx].description ); //длина описания
    char buf[SIZE]; // буфер для текста входящего в одну строку
    char* headStr = par[idx].description;
    char* endStr;
    int size;
    if( lenDes > MAX_DES )
    {
        do {
            endStr = utf_stroffset( headStr, MAX_DES );
            size = (int)( endStr - headStr );
            strncpy( buf, headStr, size );
            buf[size] = '\0';
            if( headStr == par[idx].description ) // если начало описания
            {
                printf( "%s\n", buf );
            } else
            {
                printf( "\t\t\t\t\t%s\n", buf );
            }
            headStr = endStr;
            lenDes = utf_strlen( headStr );
        } while( lenDes > MAX_DES );
        endStr = utf_stroffset( headStr, MAX_DES );
    }
}

```

```

        size = (int)(endStr - headStr);
        strncpy( buf, headStr, size );
        buf[size] = '\0';
        printf( "\t\t\t\t\t%s\n", buf );
    }
    else
    {
        printf( "%s\n", par[idx].description );
    }
}

return 0;
}

```

Файл version.c

```

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

// #define VERSTR #VERSIONSTR

static char *versionstr=VERSIONSTR;

long tioGetVersion(void )
{
    return VERSION;
}

char* tioGetVersionString(void )
{
    return versionstr;
}

```

Файл finish.c

```

#include <stdlib.h>
#include <stdio.h>

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <tioinit.h>

#include <finish_msg.h>
#define MAXARGS 100

extern char *selfname;

static size_t finish_count = 3;

void tioFinish(size_t num)
{
    if(num >= finish_count)
    {
        num = finish_count;
    }
    for(int i = 0; i < MAXARGS; ++i)
        free(tio_argv[i]);
    tioFree();
    fprintf(stdout, finish_messages[num], selfname);
    free(selfname);
    exit(num);
}

```

Файл error.c

```

#include <tioerror.h>

```

```

#include <tiowerror.h>

#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <errno.h>

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <compare.h>

typedef struct _errdeque
{
    pthread_t id;    //дескриптор потока
    int error;
    struct _errdeque *next;
    struct _errdeque *prev;
} errdeque;

static pthread_mutex_t errlock = PTHREAD_MUTEX_INITIALIZER;

static errdeque *starterr = NULL;
static errdeque *enderr = NULL;

int tioErrorInit(void)
{
    errdeque* obj = NULL;

    pthread_mutex_lock(&errlock);
    if (starterr || enderr)
    {
#ifdef NDEBUG
        fputs("Internal pointers for deque already initialized", stderr);
#endif
        pthread_mutex_unlock(&errlock);
        return TEINTMC;
    }
    if(!(obj=malloc(sizeof(errdeque))))
    {
#ifdef NDEBUG
        fprintf(stderr,"Memory error at %s:%d\n", __FILE__, __LINE__);
#endif
        return TEEPICFAIL;
    }
    obj->next = obj->prev = NULL;
    obj->error = 0;
    obj->id = pthread_self();
    starterr = enderr = obj;
    pthread_mutex_unlock(&errlock);
    return 0;
}

void tioErrorFree(void)
{
    errdeque *fr, *iter = starterr;

    pthread_mutex_lock(&errlock);
    while (iter)
    {
        fr = iter;
        iter = iter->next;
        free(fr);
    }
    starterr = enderr = NULL;
    pthread_mutex_unlock(&errlock);
}

//=====
int tioSetErrorNum(int num)
{

```



```

//typedef struct _errdeque
//{
//    //pthread_t id;    //дескриптор потока
//    //int error;
//    //struct _errdeque *next;
//    //struct _errdeque *prev;
//} errdeque;

errdeque *iter, *obj;
pthread_t thread;

thread = pthread_self();
pthread_mutex_lock(&errlock);
if ( !starterr || !enderr )
{
#ifdef NDEBUG
    fputs("Attempt to set error before error initialization or after error
free\n", \
        stderr);
#endif
    pthread_mutex_unlock(&errlock);
    return TEINTMC;
}

if (cmppthread_t(&(((errdeque*)enderr)->id), &thread) > 0)
{
    if (!(obj=malloc(sizeof(errdeque))))
    {
#ifdef NDEBUG
        fprintf(stderr, "Memory error at %s:%d\n", __FILE__, __LINE__);
#endif
        pthread_mutex_unlock(&errlock);
        return TEEPICFAIL;
    }
    // сохраняем код новой ошибки и id потока
    obj->prev = enderr;
    obj->next = NULL;
    enderr->next = obj;
    obj->error = num;
    obj->id = thread;
    enderr = obj;
}
for (iter=starterr; iter != NULL; iter = iter->next)
{
    long long res;
    if (!(res = cmppthread_t(&(((errdeque*)iter)->id), &thread)))
    {
        iter->error = num;
        break;
    }

    if(res < 0)
    {
        if(!(obj = malloc(sizeof(errdeque))))
        {
#ifdef NDEBUG
            fprintf(stderr, "Memory error at %s:%d\n", __FILE__, __LINE__);
#endif
            pthread_mutex_unlock(&errlock);
            return TEEPICFAIL;
        }
        if ( iter == starterr )
            starterr = obj;
        obj->next = iter;
        obj->prev = iter->prev;
        obj->id = thread;
        obj->error = num;
        if(iter->prev)
            iter->prev->next = obj;
        iter->prev = obj;
        break;
    }
}
pthread_mutex_unlock(&errlock);
return 0;

```

```

}
int tioGetError()
{
    errdeque *iter;
    pthread_t id = pthread_self(); //берет id своего потока
    int result = 0;

    pthread_mutex_lock(&errlock);
    if( !starterr || !enderr)
    {
#ifdef NDEBUG
        fputs("Attemp to use uninitialized error deque\n", stderr);
#endif
        return TEEPICFAIL;
    }
    if (cmppthread_t(&(((errdeque*)enderr)->id), &id)>0)
    {
        pthread_mutex_unlock(&errlock);
        return 0;
    }
    for (iter=starterr; iter != NULL; iter = iter->next)
    {
        long long res;
        if(!(res=cmppthread_t(&(((errdeque*)iter)->id), &id)))
        {
            result = iter->error;
            iter->error = 0;
            break;
        }
        if (res<0)
        {
            result = 0;
            break;
        }
    }
    pthread_mutex_unlock(&errlock);
    return result;
}

```

Файл getparam.c

```

#include <tio.h>
#include <tioerror.h>
#include <tioinit.h>
#include <string.h>
#include <tiowerror.h>
#include <math.h>

#ifdef HAVE_CONFIG_H
    #include <config.h>
#endif //HAVE_CONFIG_H

#define TRUE 1
#define FALSE 0

#define BASE_CONVERT 10

tio_param_rec* getnextval(tio_param_rec* val);
int write_val_in_buff(char* val, char *buff, size_t buff_len);
int write_bool_in_buff(char* Bool, char* buff, size_t buff_len);

struct _strBool
{
    char *sTrue;
    char *sFalse;
} strBool = {"TRUE", "FALSE"};

static inline long retbool(const char* str)
{
    if (str == NULL)
    {
        return (FALSE);
    }
}

```

```

    else
    {
        return (TRUE);
    }
}

int tioGetS(const char* name, char* buff, const size_t buff_len)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;

    if (buff == NULL)
    {
        tioSetErrorNum(TEINVAL);
        return(TEINVAL);
    }
    else if (!buff_len)
    {
        tioSetErrorNum(TENES);
        return (TENES);
    }
    while(cnt <= Tio_params_sz)
    {
        int result = strcmp(name,curr_tio_param->name);
        if (result == 0)
        {
            if (curr_tio_param->has_key)
            {
                if (curr_tio_param->val != NULL)
                {
                    char *curr_val = curr_tio_param->val;
                    size_t vallen = strlen(curr_val);
                    if ((vallen+1) <= buff_len)
                    {
                        strcpy(buff,curr_val);
                        tioSetErrorNum(TESUC);
                        return (TESUC);
                    }
                    else
                    {
                        strncpy(buff,curr_val,buff_len-1);
                        char *setoff = (buff + buff_len - 1);
                        *setoff = '\0';
                        tioSetErrorNum(TENES);
                        return (TENES);
                    }
                }
            }
            else
            {
                tioSetErrorNum(TENOTSET);
                return (TENOTSET);
            }
        }
        else
        {
            if (curr_tio_param->val != NULL)
            {
                size_t strval = strlen(strBool.sTrue);
                if (strval+1 <= buff_len)
                {
                    strcpy(buff,strBool.sTrue);
                    tioSetErrorNum(TESUC);
                    return (TESUC);
                }
                else
                {
                    tioSetErrorNum(TENES);
                    return (TENES);
                }
            }
            else
            {
                size_t strval = strlen(strBool.sFalse);
                if (strval+1 <= buff_len)
                {
                    strcpy(buff,strBool.sFalse);

```

```

        tioSetErrorNum(TESUC);
        return (TESUC);
    }
    else
    {
        tioSetErrorNum(TENES);
        return (TENES);
    }
}
}
}
curr_tio_param++;
cnt++;
}
tioSetErrorNum(TENOPAR);
return (TENOPAR);
//*****
}

int tioGetDefS(const char* name, const char* Default, char* buff, const size_t
buff_len)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;

    if (buff == NULL)
    {
        tioSetErrorNum(TEINVAL);
        return(TEINVAL);
    }
    else if (!buff_len)
    {
        tioSetErrorNum(TENES);
        return (TENES);
    }
    while(cnt <= Tio_params_sz)
    {
        int result = strcmp(name,curr_tio_param->name);
        if (result == 0)
        {
            if (curr_tio_param->has_key)
            {
                if(curr_tio_param->val != NULL)
                {
                    char *curr_val = curr_tio_param->val;
                    size_t vallen = strlen(curr_val);
                    if ((vallen+1) <= buff_len)
                    {
                        strcpy(buff,curr_val);
                        tioSetErrorNum(TESUC);
                        return (EXIT_SUCCESS);
                    }
                    else
                    {
                        strncpy(buff,curr_val,buff_len-1);
                        char *setoff = (buff + buff_len - 1);
                        *setoff = '\0';
                        tioSetErrorNum(TENES);
                        return (TENES);
                    }
                }
            }
            else
            {
                tioSetErrorNum(TENOTSET);
                size_t vallen = strlen(Default);
                if((vallen+1) <= buff_len)
                {
                    strcpy(buff,Default);
                }
                return (TENOTSET);
            }
        }
        else
        {
            if (curr_tio_param->val != NULL)
            {

```

```

        size_t strval = strlen(strBool.sTrue);
        if (strval+1 <= buff_len)
        {
            strcpy(buff, strBool.sTrue);
            tioSetErrorNum(TESUC);
            return (TESUC);
        }
        else
        {
            tioSetErrorNum(TENES);
            return (TENES);
        }
    }
    else
    {
        size_t strval = strlen(strBool.sFalse);
        if (strval+1 <= buff_len)
        {
            strcpy(buff, strBool.sFalse);
            tioSetErrorNum(TESUC);
            return (TESUC);
        }
        else
        {
            tioSetErrorNum(TENES);
            return (TENES);
        }
    }
}
curr_tio_param++;
cnt++;
}
size_t vallen = strlen(Default);
if((vallen+1) <= buff_len)
{
    strcpy(buff, Default);
}
tioSetErrorNum(TENOPAR);
return (TENOPAR);
}

long tioGetL(const char* name)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;
    while(cnt <= Tio_params_sz)
    {
        int rescmp = strcmp(name, curr_tio_param->name);
        if (rescmp == 0)
        {
            long result;
            if (curr_tio_param->has_key)
            {
                if (curr_tio_param->val != NULL)
                {
                    char* endptr;
                    result = strtol(curr_tio_param->val, &endptr, BASE_CONVERT);
                    if (!strcmp(endptr, ""))
                    {
                        if (result == LONG_MAX || result == LONG_MIN)
                        {
                            tioSetErrorNum(TEINCTYPE);
                            return (LONG_MAX);
                        }
                        else
                        {
                            tioSetErrorNum(TESUC);
                            return (result);
                        }
                    }
                }
            }
            else
            {
                tioSetErrorNum(TEINCTYPE);
                return (LONG_MAX);
            }
        }
    }
}

```

```

        }
    }
    else
    {
        tioSetErrorNum(TENOTSET);
        return (LONG_MAX);
    }
}
else
{
    result = retbool(curr_tio_param->val);
    tioSetErrorNum(TESUC);
    return(result);
}
}
curr_tio_param++;
cnt++;
}
tioSetErrorNum(TENOPAR);
return (LONG_MAX);
}

long tioGetDefL(const char* name, const long Default)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;

    while(cnt <= Tio_params_sz)
    {
        int rescmp = strcmp(name,curr_tio_param->name);
        if (rescmp == 0)
        {
            long result;
            if (curr_tio_param->has_key)
            {
                if(curr_tio_param->val != NULL)
                {
                    char* endptr;
                    result = strtol(curr_tio_param->val,&endptr,BASE_CONVERT);
                    if (!strcmp(endptr,""))
                    {
                        if (result == LONG_MAX || result == LONG_MIN)
                        {
                            tioSetErrorNum(TEINCTYPE);
                            return (Default);
                        }
                        else
                        {
                            tioSetErrorNum(TESUC);
                            return (result);
                        }
                    }
                }
                else
                {
                    tioSetErrorNum(TEINCTYPE);
                    return (Default);
                }
            }
            else
            {
                tioSetErrorNum(TENOTSET);
                return (Default);
            }
        }
        else
        {
            result = retbool(curr_tio_param->val);
            tioSetErrorNum(TESUC);
            return(result);
        }
    }
    curr_tio_param++;
    cnt++;
}
tioSetErrorNum(TENOPAR);
return (Default);
}

```

```

}

double tioGetD(const char* name)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;
    while(cnt <= Tio_params_sz)
    {
        int result = strcmp(name,curr_tio_param->name);
        if (result == 0)
        {
            double result;
            if (curr_tio_param->has_key)
            {
                if(curr_tio_param->val != NULL)
                {
                    char* endptr;
                    result = strtod(curr_tio_param->val,&endptr);
                    if (!strcmp(endptr,""))
                    {
                        if ((result == HUGE_VAL) || (result == -HUGE_VAL))
                        {
                            tioSetErrorNum(TEINCTYPE);
                            return (DBL_MAX);
                        }
                        else
                        {
                            tioSetErrorNum(TESUC);
                            return (result);
                        }
                    }
                }
                else
                {
                    tioSetErrorNum(TEINCTYPE);
                    return (DBL_MAX);
                }
            }
            else
            {
                tioSetErrorNum(TENOTSET);
                return (DBL_MAX);
            }
        }
        else
        {
            result = (double)retbool(curr_tio_param->val);
            tioSetErrorNum(TESUC);
            return(result);
        }
    }
    curr_tio_param++;
    cnt++;
}
tioSetErrorNum(TENOPAR);
return (DBL_MAX);
}

double tioGetDefD(const char* name, const double Default)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;
    while(cnt <= Tio_params_sz)
    {
        int result = strcmp(name,curr_tio_param->name);
        if (result == 0)
        {
            double result;
            if (curr_tio_param->has_key)
            {
                if(curr_tio_param->val != NULL)
                {
                    char* endptr;
                    result = strtod(curr_tio_param->val,&endptr);

```

```

        if (!strcmp(endptr, ""))
        {
            if ((result == HUGE_VAL) || (result == -HUGE_VAL))
            {
                tioSetErrorNum(TEINCTYPE);
                return (Default);
            }
            else
            {
                tioSetErrorNum(TESUC);
                return (result);
            }
        }
        else
        {
            tioSetErrorNum(TEINCTYPE);
            return (Default);
        }
    }
    else
    {
        tioSetErrorNum(TENOTSET);
        return (Default);
    }
}
else
{
    result = (double)retbool(curr_tio_param->val);
    tioSetErrorNum(TESUC);
    return(result);
}
}
curr_tio_param++;
cnt++;
}
tioSetErrorNum(TENOPAR);
return (Default);
}

unsigned char tioGetC(const char* name)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;

    while(cnt <= Tio_params_sz)
    {
        int rescmp = strcmp(name, curr_tio_param->name);
        if (rescmp == 0)
        {
            long result;
            if (curr_tio_param->has_key)
            {
                if(curr_tio_param->val != NULL)
                {
                    char* endptr;
                    result = strtol(curr_tio_param->val, &endptr, BASE_CONVERT);

                    if (!strcmp(endptr, ""))
                    {
                        if (result >= 0 && result < UCHAR_MAX)
                        {
                            tioSetErrorNum(TESUC);
                            return(result);
                        }
                        else
                        {
                            tioSetErrorNum(TEINCTYPE);
                            return (UCHAR_MAX);
                        }
                    }
                    else
                    {
                        tioSetErrorNum(TEINCTYPE);
                        return (UCHAR_MAX);
                    }
                }
            }
        }
        cnt++;
        curr_tio_param++;
    }
    tioSetErrorNum(TENOPAR);
    return (Default);
}

```



```

        }
        else
        {
            tioSetErrorNum(TENOTSET);
            return (UCHAR_MAX);
        }
    }
    else
    {
        result = retbool(curr_tio_param->val);
        tioSetErrorNum(TESUC);
        return(result);
    }
}
curr_tio_param++;
cnt++;
}
tioSetErrorNum(TENOPAR);
return (UCHAR_MAX);
}

unsigned char tioGetDefC(const char* name, const unsigned char Default)
{
    int cnt = 1;
    tio_param_rec *curr_tio_param = Tio_params;
    while(cnt <= Tio_params_sz)
    {
        int rescmp = strcmp(name,curr_tio_param->name);
        if (rescmp == 0)
        {
            long result;
            if (curr_tio_param->has_key)
            {
                if(curr_tio_param->val != NULL)
                {
                    char* endptr;
                    result = strtol(curr_tio_param-
>val,&endptr,BASE_CONVERT);
                    if (!strcmp(endptr,""))
                    {
                        if (result >= 0 && result < UCHAR_MAX)
                        {
                            tioSetErrorNum(TESUC);
                            return(result);
                        }
                        else
                        {
                            tioSetErrorNum(TEINCTYPE);
                            return (Default);
                        }
                    }
                    else
                    {
                        tioSetErrorNum(TEINCTYPE);
                        return (Default);
                    }
                }
                else
                {
                    tioSetErrorNum(TENOTSET);
                    return (Default);
                }
            }
            else
            {
                result = retbool(curr_tio_param->val);
                tioSetErrorNum(TESUC);
                return(result);
            }
        }
        curr_tio_param++;
        cnt++;
    }
    tioSetErrorNum(TENOPAR);
    return (Default);
}

```

```

}

tio_param_rec* getnextval(tio_param_rec* val)
{
    static int cnt_sz = 0;
    if(val == NULL || (++val) == NULL)
    {
        return(NULL);
    }
    else if(cnt_sz < Tio_params_sz)
    {
        cnt_sz++;
        return(val);
    }
    else
    {
        return(NULL);
    }
}

int write_val_in_buff(char* val, char *buff, size_t buff_len)
{
    if (buff == NULL)
    {
        tioSetErrorNum(TEINVAL);
        return(TEINVAL);
    }
    else if (!buff_len)
    {
        tioSetErrorNum(TENES);
        return(TENES);
    }

    if(strlen(val)+1 <= buff_len)
    {
        strcpy(buff,val);
        tioSetErrorNum(TESUC);
        return(TESUC);
    }
    else
    {
        strncpy(buff,val,buff_len-1);
        char *setoff = (buff + buff_len - 1);
        *setoff = '\0';
        tioSetErrorNum(TENES);
        return(TENES);
    }
}

int write_bool_in_buff(char* Bool, char* buff, size_t buff_len)
{
    if (buff == NULL)
    {
        tioSetErrorNum(TEINVAL);
        return(TEINVAL);
    }
    else if (!buff_len)
    {
        tioSetErrorNum(TENES);
        return(TENES);
    }

    if (strlen(Bool)+1 <= buff_len)
    {
        strcpy(buff, Bool);
        tioSetErrorNum(TESUC);
        return(TESUC);
    }
    else
    {
        tioSetErrorNum(TENES);
        return (TENES);
    }
}

```

Файл tioTableBegin.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>

#include <tio.h>

#define WIDTH 120

typedef struct cell      /*list*/
{
    void **s;            /*Массив указателей на неизвестный тип данных*/
    struct cell *n;      /*Указатель на следующий элемент списка*/
} cl;

struct inform
{
    char **cap;          /*Шапка*/
    int countLetter;     /*Количество символов в строке*/
    int countColum;      /*Количество столбцов*/
    int *bufType;        /*Буфер с типам данных для каждого столбца*/
    cl *head, *ptr;      /*Список данных*/
};

int tabRow( void **strings, int *bufType, int countColum, int lenColCon );
int capMap( int countColum, char **cap, int lenColCon );
int drawLine( int lenColCon );

/*Table initialization*/
void* tioTableBegin( const char* format, ... )
{
    va_list ap;
    int i;
    int jj;
    int j;
    struct inform *datTab;
    void *extPointer;

    va_start( ap, format );

    /*Allocating memory for datTab structure*/
    if( (datTab = (struct inform*) calloc( 1, sizeof(struct inform)) ) == NULL )
    {
        fprintf( stderr, "\nError \n" );
        return NULL;
    }

    /*Count the number of columns in a table and calculate the maximum length of the
column.*/
    datTab->countColum = 1;
    for( i = 0; format[i] != '\0'; ++i )
    {
        ++ datTab->countLetter;
        if( (format[i] == '&') && ((format[i+1]) != '&'))
            ++ datTab->countColum;
        else if( (format[i] == '&') && ((format[i+1]) == '&'))
            ++ i;
    }

    //printf("Count of letter %d.\nCount of colum %d.\n", datTab->countLetter,
    //      datTab->countColum );

    /*Allocating memory for cap*/
    if( (datTab->cap = (char**) calloc(datTab->countColum , sizeof(char*))) != NULL )
    {
        for( i = 0; i < datTab->countColum ; ++ i)
        {
            if( (datTab->cap[i] = (char*) calloc(datTab->countLetter + 1,
sizeof(char))) == NULL )
            {
                printf(" error datTab->cap[%d][]\n",i);
            }
        }
    }
}
```

```

        return NULL;
    }
}
else
{
    printf("\nerror\n");
    return NULL;
}

/*Allocating memory for datTab->bufType*/
if( (datTab->bufType = (int*) calloc(datTab->countColum, sizeof(int))) == NULL )
{
    printf("\nerror *datTab->bufType\n");
    return NULL;
}

/*printf("\n");*/

/*Allocating memory for pointer to the head of list*/
if( (datTab->head = (cl *) malloc( sizeof(cl))) == NULL )
{
    printf("\nError ptr\n");
    return NULL;
}

datTab->ptr = datTab->head;
datTab->ptr->n = NULL;

/*Allocating memory for ptr->s*/
if( (datTab->ptr->s = (void **) calloc(datTab->countColum, sizeof(void *))) ==
NULL )
{
    printf("\nError ptr->s\n");
    return NULL;
}

/*Writing data types to the buffer*/
//printf("Data types: ");
for( i = 0; i < datTab->countColum; ++ i )
{
    datTab->bufType[i] = va_arg(ap, int);
    /*printf("%d ", datTab->bufType[i]);*/
}

/*printf("\n");*/

/*Set colum's name*/
jj = 0;
for( i = 0; i < datTab->countColum; ++ i )
{
    j = 0;
    while(1)
    {
        if(format[jj] == '\0')
            break;
        if((format[jj] == '&') && (format[jj+1] != '&'))
            break;
        datTab->cap[i][j] = format[jj];
        if((format[jj] == '&') && (format[jj+1] == '&'))
            ++jj;
        ++ j;
        ++ jj;
    }
    ++jj;
}

/*Print colum's name*/
for( i = 0; i < datTab->countColum; ++i )
{
    /*printf("%s", datTab->cap[i]);*/
    /*printf("\n");*/
}

```

```

    }

    va_end( ap );

    extPointer = (void *) datTab;
    return extPointer;
}

/*RECORDING DATA IN TABLE*/
void *tioTableRecord( void *td, ... )
{
    va_list ap;
    struct inform *datTab;
    int i;
    char ch;
    double dbd;
    long ln;
    char* strn;
    void *extPointer;

    va_start( ap, td );

    datTab = (struct inform *) td;

    for( i = 0; i < datTab->countColum; ++ i )
    {
        switch( datTab->bufType[i] )
        {
            case 1:
                ch = (char) va_arg( ap, int);
                datTab->ptr->s[i] = calloc(1, sizeof(char));
                *(char *)datTab->ptr->s[i] = ch;
                //printf("char %c /added\n", *(char *)datTab->ptr->s[i]);
                break;
            case 2:
                dbd = va_arg( ap, double);

                datTab->ptr->s[i] = malloc(sizeof(double));
                *(double *)datTab->ptr->s[i] = dbd;
                //printf("double %f /added\n", *(double *)datTab->ptr->s[i]);

                break;
            case 3:
                ln = va_arg( ap, long);
                datTab->ptr->s[i] = calloc(1, sizeof(long));
                *(long *)datTab->ptr->s[i] = ln;
                //printf("long %ld /added\n", *(long *)datTab->ptr->s[i]);
                break;
            case 4:
                strn = va_arg( ap, char *);
                datTab->ptr->s[i]=(char *)calloc(strlen(strn) + 1, sizeof(char));
                strcpy ((char *)datTab->ptr->s[i],strn);
                //printf("string %s /added\n", (char *) datTab->ptr->s[i]);
                break;
            default:
                printf("Неправильный параметр функции tioTableRecord!\n");
                return NULL;
        }
    }

    /*ALLOCATING MEMORY FOR THE HEAD POINTER OF LIST*/
    if( (datTab->ptr->n = (cl*) malloc(sizeof(cl))) == NULL )
    {
        printf("Error\n");
        return NULL;
    }
    datTab->ptr = datTab->ptr->n;
    datTab->ptr->n = NULL;

    /*ALLOCATIONG MEMORY FOR THE POINTER OF ptr->s*/
    if( (datTab->ptr->s = (void **) calloc(datTab->countColum, sizeof(void *))) ==
    NULL )
    {
        printf("\nError ptr->s\n");
        return NULL;
    }
}

```

```

    }

    va_end(ap);

    //extPointer = (void *) datTab;
    td = ( void * ) datTab;

    return td;
}

/*Table's output*/
int tioTableEnd( void *td )
{
    struct inform *datTab;

    int i;                                /*Counter*/
    int lenColCon;                        /*Cell width*/
    void *next;
    int *masType;                        /*For type of cap*/
    datTab = (struct inform *) td;

    datTab->ptr = datTab->head;
    /*printf("string in End = %s\n", (char*)datTab->ptr->s[3]);*/

    /*Calculate the column width depending on the number of*/
    lenColCon = WIDTH / datTab->countColumn;
    /*printf("Размер колонки = %d\n", lenColCon);

    if( (masType = (int *) malloc(datTab->countColumn * sizeof(int))) == NULL)
    {
        printf("ERROR!");
        exit(EXIT_FAILURE);
    }

    /*Draw line*/
    drawLine(lenColCon);

    for( i = 0; i < datTab->countColumn; ++ i )
    {
        masType[i] = 4;
    }
    /*Display cap*/
    tabRow( (void *)datTab->cap, masType, datTab->countColumn, lenColCon);

    /*Draw line*/
    drawLine(lenColCon);

    while( datTab->ptr->n != NULL )
    {
        /*Print table row*/
        tabRow( datTab->ptr->s, datTab->bufType, datTab->countColumn, lenColCon );

        /*Jump to the new cell*/
        datTab->ptr = datTab->ptr->n;
        /*Draw line*/
        drawLine(lenColCon);
    }

    /*Free masType*/
    free(masType);
    /*Free data*/
    for( i = 0; i < datTab->countColumn; ++ i )
        free(datTab->cap[i]);
    free( datTab->cap );

    /*Free bufTab*/
    free(datTab->bufType);

    /*Free ptr->s*/
    datTab->ptr = datTab->head;
    do
    {
        next = (void *) datTab->ptr->n;

```

```

        for( i = 0; i < datTab->countColumn; ++ i )
        {
            if(datTab->ptr->n != NULL)
                free(datTab->ptr->s[i]);
        }
        free(datTab->ptr->s);
        free(datTab->ptr);
        datTab->ptr = (cl *) next;
    } while (datTab->ptr != NULL);

    /*Free datTab*/
    free(datTab);

    return 0;
}

/*LOCAL FUNCTION*/

/*Table row*/
int tabRow( void **strings, int *bufType, int countColumn, int lenColCon )
{
    char ***data;                /*Pointer of the table's data*/
    int *colStr;                 /*Array of extra lines*/
    int i;
    int j;
    int max = 0;
    int extraCounter;            /*Extra lines counter*/
    int offset;                  /*Offset counter*/
    int strCounter;

    /*Allocating memory for colStr*/
    if( (colStr = (int *) malloc(countColumn * sizeof(int))) == NULL )
    {
        printf("ERROR!");
        exit(EXIT_FAILURE);
    }

    /*Calculation number of extra lines of the array*/
    for( i = 0; i < countColumn; ++ i )
    {
        colStr[i] = strlen( (char *)strings[i] ) / lenColCon;
        if(max < colStr[i])
            max = colStr[i];
    }

    /*Allocating memory for data*/
    if((data = (char ***) malloc (countColumn * sizeof(char **))) == NULL)
    {
        printf("ERROR!\n");
        exit(EXIT_FAILURE);
    }
    for( i = 0; i < countColumn; ++ i )
    {
        if((data[i] = (char **) malloc ((max + 1) * sizeof(char *))) == NULL)
        {
            printf("ERROR!\n");
            exit(EXIT_FAILURE);
        }
        else
        {
            for (j = 0; j < (max + 1); ++ j)
            {
                if((data[i][j] = (char *) malloc ( 2 * lenColCon * sizeof(char))) ==
NULL)
                {
                    printf("ERROR!\n");
                    exit(EXIT_FAILURE);
                }
            }
        }
    }

    /*Partition line*/
    for(i = 0; i < countColumn; ++ i)
    {
        switch( bufType[i] )

```

```

{
case 1:
    sprintf( data[i][0], "%c",*(char *) strings[i]);
    offset = strlen(data[i][0]);
    /*Insert spaces*/
    for( offset; offset < (lenColCon - 1); ++ offset)
        data[i][0][offset] = ' ';
    /*Insert spaces*/
    for(extraCounter = 1; extraCounter < (max + 1); ++ extraCounter)
        for( offset = 0; offset < (lenColCon - 1); ++ offset )
            data[i][extraCounter][offset] = ' ';
    break;
case 2:
    sprintf( data[i][0], "%.2f",*(double *) strings[i]);
    offset = strlen(data[i][0]);
    /*Insert spaces*/
    for( offset; offset < (lenColCon - 1); ++ offset)
        data[i][0][offset] = ' ';
    /*Insert spaces*/
    for(extraCounter = 1; extraCounter < (max + 1); ++ extraCounter)
        for( offset = 0; offset < (lenColCon - 1); ++ offset )
            data[i][extraCounter][offset] = ' ';
    /*printf("90 string in End = %s\n", (char*)strings[3]);*/
    break;
case 3:
    sprintf( data[i][0], "%ld",*(long *) strings[i]);
    offset = strlen(data[i][0]);
    /*Insert spaces*/
    for( offset; offset < (lenColCon - 1); ++ offset)
        data[i][0][offset] = ' ';
    /*Insert spaces*/
    for(extraCounter = 1; extraCounter < (max + 1); ++ extraCounter)
        for( offset = 0; offset < (lenColCon - 1); ++ offset )
            data[i][extraCounter][offset] = ' ';
    break;
case 4:
    for( extraCounter = 0; extraCounter <= colStr[i]; ++ extraCounter )
    {
        int index = 0;
        j = extraCounter * (lenColCon - 1);
        offset = 0;

        while( ( ( lenColCon - 1 ) != index ) && ( ( (char*)strings[i])[j] !=
'\0' ) )
        {
            if( ( (char*)strings[i])[j] & 0x80 )
            {
                data[i][extraCounter][offset] = ((char *)strings[i])[j];
                ++ offset;
                ++ j;
                data[i][extraCounter][offset] = ((char *)strings[i])[j];
                ++ j;
                ++ index;
                ++ offset;
            }
            else
            {
                data[i][extraCounter][offset] = ((char *)strings[i])[j];
                ++ j;
                ++ offset;
                ++ index;
            }
        }

        /*for( offset = 0;
((offset != (lenColCon - 1)) && (((char *)strings[i])[j] !=
'\0' )); ++ offset, ++ j)
        {
            data[i][extraCounter][offset] = ((char *)strings[i])[j];
        }*/

        /*Insert spaces*/
        for( index; index < (lenColCon - 1); ++ index, ++ offset)

```



```

        data[i][extraCounter][offset] = ' ';
    }
    /*Insert spaces*/
    for(extraCounter = colStr[i] + 1; extraCounter < (max + 1); ++
extraCounter )
    {
        for( offset = 0; offset < (lenColCon - 1); ++ offset )
            data[i][extraCounter][offset] = ' ';
        data[i][extraCounter][offset] = '\0';
    }
    break;
default:
    printf("ERROR!");
    exit(EXIT_FAILURE);
}

}

/*Print row*/
for( strCounter = 0; strCounter < (max + 1); ++ strCounter )
{
    for( i = 0; i < countColumn; ++ i )
        printf("|%s", data[i][strCounter]);
    printf("|\\n");
}

/*FREE */
for( i = 0; i < countColumn; ++ i )
{
    for( j = 0; j <= colStr[i]; ++ j )
    {
        free(data[i][j]);
    }
    free(data[i]);
}
free(data);
free(colStr);
return 0;
}

/*Display cap*/
int capMap( int countColumn, char **cap, int lenColCon )
{
    int i;
    int j;
    int tmp;
    for(i = 0; i < countColumn; ++ i )
    {
        printf( "|%s", cap[i] );
        tmp = lenColCon - strlen(cap[i]);
        for( j = 0; j < tmp - 1; ++ j )
            printf( " " );
    }
    printf( "|\\n");
    return 0;
}

/*Draw line function*/
int drawLine( int lenColCon )
{
    char *pLine = malloc( WIDTH * sizeof( char ) );
    int i;
    for( i = 0; i < WIDTH; ++ i ) /*Need correct WIDTH*/
    {
        if((i % lenColCon) == 0)
            pLine[i] = '+';
        else
            pLine[i] = '-';
    }
}

```

```
    pLine[i] = '+';  
    pLine[++i] = '\\n';  
    fputs( pLine, stdout );  
    free( pLine );  
    return 0;  
}
```

Приложение 2

Исходные файлы тестов прототипов

Решение квадратного уравнения

Файл *source.c*

```
#include <tio.h>
#include <tioerror.h>
#include <utf.h>
#include <limits.h>
#include <stdio.h>
#include <math.h>
/*#include <stdio.h>*/

typedef struct _SRoots {
    long root1;
    long root2;
} SRoots;

/*Решение квадратного уравнения*/
int quad( long a, long b, long c, SRoots* Roots ) {
    long d = b * b - 4 * a * c;
    Roots->root1 = ( -b + sqrt( (double)d ) ) / ( 2 * a );
    Roots->root2 = ( -b - sqrt( (double)d ) ) / ( 2 * a );
    return 0;
}

int main( int argc, const char* argv[] ) {
    SRoots *Roots = malloc( sizeof(SRoots) );
    //SRoots *RootsEtalon = malloc( sizeof(SRoots) );

    //int myargc = 6;

    //const char* myargv[] = {
        //argv[0],
        //"-a",
        //"1",
        //"-b",
        //"2",
        //"-c",
        //"-3",
    //};

    tio_param mypar [] = {
        {"a:", "A", "Параметр A"},
        {"b:", "B", "Параметр B"},
        {"c:", "C", "Параметр C"},
        {"root1:", "ROOT1", "Первый корень"},
        {"root2:", "ROOT2", "Второй корень"},
        {NULL, NULL, NULL}
    };

    //puts( "Тест написал: Гусев Михаил" );
    //puts( "Короткое описание теста:\nТестирование функции решения квадратного
уровнения." );
    tioInit( "v0.9 alpha", "This is test", mypar, argc, argv );

    quad( tioGetL( "A" ), tioGetL( "B" ), tioGetL( "C" ), Roots );
    void *td = tioTableBegin( "Имя параметра&Значение", TIOSTRING, TIOLONG );

    tioTableRecord( td, "Аргумент A", tioGetL( "A" ) );
    tioTableRecord( td, "Аргумент B", tioGetL( "B" ) );
    tioTableRecord( td, "Аргумент C", tioGetL( "C" ) );
    /*tioTableRecord( td, "The equation root 1", Roots->root1 );*/
    /*tioTableRecord( td, "The equation root 2", Roots->root2 );*/
    tioTableEnd( td );

    puts( "Сравнение эталонных и возвращаемых функцией корней" );
    td = tioTableBegin( "Эталонные корни&Корни, посчитанные функцией", TIOLONG,
TIOLONG );
```

```

tioTableRecord( td, tioGetL( "ROOT1" ), Roots->root1 );
tioTableRecord( td, tioGetL( "ROOT2" ), Roots->root2 );
tioTableEnd( td );

free(Roots);
tioFinish( 0 );
return 0;
}

```

Тестирование работоспособности com-порта

Файл *main.c*

```

#define _POSIX_SOURCES 1

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <sys/wait.h>

#include <tio.h>

#include "client.h"
#include "server.h"
#include "config.h"
#include "termcontrol.h"
#include "debuging.h"

void termination_signal(int signum)
{
    int work = config.work;
    config.work = 0;
    if (config.outputDevice != -1)
    {
        close_serial_port(config.outputDevice);
        config.outputDevice = -1;
    }
    if (work)
        kill(0, SIGTERM);
}

int main(int argc, const char* argv[])
{
    pid_t server_child = 0;
    int fd;
    int res = 0, status;
    struct sigaction sigchildAction;

    tio_param sParam [] = {
        { "D:", "DURATION", "Duration" },
        { "m:", "PORTSPEED", "Prot speed" },
        { "s:", "SENDPACKSLENGTH", "Send pack length" },
        { "d", "SERVERMODE", "Server mode" },
        { "l", "CLIENTMODE", "Client mode" },
        { "L", "CLIENTSERVERMODE", "Client/Server mode" },
        { NULL, NULL, NULL }
    };

    sigchildAction.sa_handler = termination_signal;
    sigchildAction.sa_flags = SA_NOCLDSTOP;
    sigemptyset(&(sigchildAction.sa_mask));
    sigaddset(&(sigchildAction.sa_mask), SIGTERM);

    if (sigaction(SIGTERM, &sigchildAction, NULL))
    {
        perror("Signal SIGTERM registration failed");
        return -1;
    }
    if (sigaction(SIGINT, &sigchildAction, NULL))
    {

```

```

        perror("Signal SIGINT registration failed");
        return -1;
    }

    // Инициализация tio и разбор входных параметров командной строки
    tioInit( "alpha", "RS232 test", sParam, argc, argv);

    if (write_configuration(&config))
    {
        fputs("Congiguration read error\n", stderr);
        return -1;
    }

    fd = open_serial_port( tio_argv[0], tioGetDefL( "PORTSPEED", 115200 ));
    if (fd < 0)
    {
        return -1;
    }
    config.outputDevice = fd;

    if ( tioGetL( "CLIENTSERVERMODE" ) > 0 )
    {
        server_child = fork();
    }
    if ((server_child == 0) && (tioGetL( "CLIENTSERVERMODE" ) || tioGetL(
"SERVERMODE" ) ) )
    {
        if (server_process(&config))
        {
            return -1;
        }
    }
    else if ( tioGetL( "CLIENTSERVERMODE" ) || tioGetL( "CLIENTMODE" ))
        res = client_process(&config);
    else
    {
        DEBUGMSG("Undefined target action");
        return -1;
    }

    if (server_child != 0)
        waitpid(server_child, &status, WNOHANG);

    // Завершение работы библиотеки tio
    tioFinish(0);

    return (int)(res || status);
    /*return 0;*/
}

```

Файл client.c

```

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <arpa/inet.h>
#include <time.h>
#include "debuging.h"
#include "datapacker.h"
#include "delays.h"
#include "config.h"

#include <tio.h>

static int
wait(int ttyFd)
{
    uint32_t readBuffer    = 0;

    struct timeval selectDelay;

```

```

    fd_set  set, *fdSet = &set;
    int     retval;

    time_t start = time(NULL);
    while(config.work)
    {
        time_t current = time(NULL);
        selectDelay.tv_sec = 0;
        selectDelay.tv_usec = BASEDELAY;
        FD_ZERO(fdSet);
        FD_SET(ttyFd, fdSet);

        if ((current - start) > 20)
            break;
        DEBUGOUT("Current %ld: stat at %ld: elapsed: %ld", current, start, (current -
start))

        retval = select(ttyFd+1, fdSet, NULL, NULL, &selectDelay);
        if (retval == -1)
        {
            perror("Client wait polling error: ");
            kill(0, SIGTERM);
            return EIO;
        }
        usleep(usec_floor(32 * tioGetDefL( "PORTSPEED", 115200 ) * 1e-6));
        retval = read(ttyFd, &readBuffer, sizeof(readBuffer));
        if (retval == -1)
        {
            if(errno == EAGAIN)
            {
                DEBUGMSG("Port is empty - nothing read");
                continue;
            }
            perror("Client waiting read error: ");
            kill(0, SIGTERM);
            return EIO;
        }
        if(retval == sizeof(readBuffer))
        {
            DEBUGPARAMTRS("Found recive buffer: %X", readBuffer);
            if(htonl(readBuffer) == WAITMAGIC)
            {
                return 0;
            }
        }
    }
    DEBUGMSG("Waiting of server failed");
    return -1;
}

static int
read_and_compare(int ttyFd, DataPack *standartMessage)
{
    DataPack buffer, decodedBuffer;
    char*     buffPtr=(char*)&buffer;
    int       buffFill=0;

    int pass = 1;
    long iteration = 0;

    struct timeval selectDelay;
    fd_set  set, *fdSet = &set;
    int     retval;
    long lastLength = 1;

    *((uint32_t*)buffPtr) = htonl(RECIVEREADMAGIC);

    while(config.work)
    {
        retval = write(ttyFd, &buffer, sizeof(uint32_t));
        if (-1 == retval)
        {
            perror("Client error write ready status: ");

```

```

        kill(0, SIGTERM);
        return EIO;
    }
    usleep(calculate_delay_in_usec(2 * WAITBASEDELAY, WAITBASEDELAY));

    selectDelay.tv_sec = 0;
    selectDelay.tv_usec = BASEDELAY;
    FD_ZERO(&fdSet);
    FD_SET(ttyFd, &fdSet);
    select(ttyFd + 1, &fdSet, NULL, NULL, &selectDelay);

    retval = read(ttyFd, &buffer, sizeof(buffer));
    if (retval == -1)
    {
        if (errno == EAGAIN)
            continue;
        perror("Client: read data error");
        kill(0, SIGTERM);
        return EIO;
    }

    buffer.magic = ntohl(buffer.magic);
    if (buffer.magic == RECIVEREADYMAGIC)
    {
        continue;
    }
    else if (buffer.magic == SENDMAGIC)
    {
        buffPtr = (char*)&buffer;
        buffFill = retval;
        break;
    }
    else if (buffer.magic == WAITMAGIC)
    {
        DEBUGOUT("Geting wait message - again");
        DEBUGPARAMETERS("Unexpected value in buffer: %X", (uint32_t)buffer.magic);
        kill(0, SIGTERM);
    }
    else
    {
        DEBUGOUT("Unexpected value in buffer: %X", (uint32_t)buffer.magic);
        kill(0, SIGTERM);
        return EINVAL;
    }
}
DEBUGMSG("Ready to read status wrote");

while(/*config.work && */ lastLength > 0)
{
    if (buffFill == sizeof(buffer))
    {
        convert_network_string_to_datapack((char*)&buffer, &decodedBuffer);
        pass = pass && !compare_data_packages(standartMessage, &decodedBuffer);
        buffFill = 0;
        buffPtr = (char*)&buffer;
        DEBUGOUT2("Message decoded: left space %d",
decodedBuffer.dataTotalLength);
        lastLength = (int)decodedBuffer.dataTotalLength;
        memset(&buffer, 0, sizeof(buffer));
        ++iteration;
        continue;
    }

    selectDelay.tv_sec = 0;
    selectDelay.tv_usec = 20 * calculate_delay_from_speed_usec( tioGetDefL(
"PORTSPEED", 115200 ) );
    FD_ZERO(&fdSet);
    FD_SET(ttyFd, &fdSet);

    retval = select(ttyFd + 1, &fdSet, NULL, NULL, &selectDelay);
    if (retval == -1)
    {
        if ( errno == EINTR )
            continue;
        perror("Client compare failed: ");
        kill(0, SIGTERM);
    }
}

```

```

        return EIO;
    }
    else if (retval == 0)
    {
        if (lastLength < 0)
            break;
        DEBUGOUT("Too long nothing happening: Waiting more bytes %ld\n",
lastLength);
        kill(0, SIGTERM);
        break;
    }
    retval = read(ttyFd, buffPtr, sizeof(buffer) - buffFill);
    if (retval == -1)
    {
        if(errno == EAGAIN)
        {
            continue;
        }
        perror("Client read port data failed: ");
        kill(0, SIGTERM);
        return EIO;
    }
    buffFill += retval;
    buffPtr += retval;
}
DEBUGMSG("Client decode finished");
return (pass && (iteration > 0)) ? 0 : 1;
}

int
client_process(Configuration *config)
{
    DataPack standartMessage;

    if
(convert_message_to_datapack(messageString, sizeof(messageString), &standartMessage))
    {
        fputs("Client: Failed to create standart message\n", stderr);
        config->work = 0;
        kill(0, SIGTERM);
        return -1;
    }
    DEBUGMSG("Starting server wait");
    if (wait(config->outputDevice))
    {
        config->work = 0;
        puts("Test FAILED");
        kill(0, SIGTERM);
        return -1;
    }
    DEBUGMSG("Starting transfere");
    if(read_and_compare(config->outputDevice, &standartMessage))
    {
        puts("Test COM - FAILED");
        config->work = 0;
        kill(0, SIGTERM);
        return -1;
    }
    config->work = 0;
    puts("client process - OK");
    return 0;
}

```

Файл condig.c

```

#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

#include <tio.h>

```



```
#include "config.h"
```

```
Configuration config = {
    -1,          //device fd
    1000,        //minimum transferred data count
    1            //work mode
};

static int
calculate_configuration(Configuration *cfg)
{
    if (!cfg)
        return EINVAL;

    if( tioGetDefL( "DURATION", 0 ) )
        cfg->sendPacksLength = ( tioGetL( "DURATION" ), tioGetDefL( "PORTSPEED",
115200 ) / 8);
    return 0;
}

int
write_configuration(Configuration *cfg )
{
    cfg->sendPacksLength = tioGetDefL( "SENDPACKSLENGTH", 1000 );
    calculate_configuration(cfg);
    return 0;
}
```

Файл datapacker.c

```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <arpa/inet.h>

#include "datapacker.h"

/**
 * Тестовая строка представляющая собой sha256 строки
 * "Test of RS-574 of RTI"
 */

char messageString[] = {0xb8, 0xf6, 0xa8, 0x47,
                        0xfe, 0xc1, 0x58, 0x2a,
                        0xd9, 0xf8, 0xa1, 0xef,
                        0x9f, 0x39, 0xe3, 0xe2,
                        0x15, 0x3e, 0x4b, 0xd4,
                        0x94, 0xc5, 0x83, 0xd5,
                        0xd0, 0x95, 0xd8, 0x43,
                        0x06, 0x73, 0xdd, 0xb8};

int
convert_message_to_datapack(char *message, size_t messageLength, DataPack *pack)
{
    if (messageLength > MESSAGELENGTH)
        return EINVAL;
    memcpy(pack->message, message, messageLength);
    pack->magic = SENDMAGIC;
    pack->messageLength = messageLength;
    pack->dataTotalLength = 0;
    return 0;
}
```

```

int
convert_datapack_to_network_string(DataPack *pack, void* buff)
{
    DataPack internalPack;
    char *pointer = buff;

    if (!pack | !buff)
        return EINVAL;

    internalPack.magic          = htonl(pack->magic);
    internalPack.messageLength  = htonl(pack->messageLength);
    internalPack.dataTotalLength = htonl(pack->dataTotalLength);

    memcpy(pointer, &(internalPack.magic), sizeof(internalPack.magic));
    pointer += sizeof(internalPack.magic);
    memcpy(pointer, &(internalPack.messageLength),
sizeof(internalPack.messageLength));
    pointer += sizeof(internalPack.messageLength);
    memcpy(pointer, &(internalPack.dataTotalLength),
sizeof(internalPack.dataTotalLength));
    pointer += sizeof(internalPack.dataTotalLength);
    memcpy(pointer, pack->message, sizeof(pack->message));
    return 0;
}

```

```

int
convert_network_string_to_datapack(char* string, DataPack *pack)
{
    char* pointer = string;

    if (!string || !pack)
        return EINVAL;

    pack->magic = ntohl(*(uint32_t*)pointer);
    pointer += sizeof(pack->magic);
    pack->messageLength = ntohl(*(uint32_t*)pointer);
    pointer += sizeof(pack->messageLength);
    pack->dataTotalLength = ntohl(*(uint32_t*)pointer);
    pointer += sizeof(pack->dataTotalLength);
    memcpy(pack->message, pointer, sizeof(pack->message));
    return 0;
}

```

```

int
compare_data_packages(DataPack* a, DataPack* b)
{
    if (a->messageLength != b->messageLength)
    {
        return (a->messageLength > b->messageLength) ? 1 : -1;
    }

    return memcmp(a->message, b->message, a->messageLength);
}

```

Файл delays.c

```

#include <stdlib.h>
#include "delays.h"

#define positive_limited_rand(limit) \
    ((rand() * limit) / RAND_MAX)

long calculate_delay_in_usec(long baseDelay, long maxRand)
{
    return baseDelay + positive_limited_rand(maxRand);
}

```

Файл server.c

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <termios.h>
#include <arpa/inet.h>
#include <tio.h>

#include "datapacker.h"
#include "debuging.h"
#include "delays.h"
#include "config.h"

/**
 * @brief Функция ожидания запроса на передачу данных.
 *
 * Функция ожидающая получения входного сообщения с кодом для начала
 * передачи данных. Данная функция в цикле размещает в СОМ порт
 * переданный в качестве параметра сообщение о готовности передачи
 * сообщения и ожидает подтверждения о готовности приема.
 *
 * @param ttyFd Фаловый дескриптор настроенного фала последовательного
 * порта.
 *
 * @return 0 в случае успешного завершения и код ошибки в противном
 * случае.
 */
static int
wait_ready_state(int ttyFd)
{
    uint32_t waitMessage = htonl(WAITMAGIC);
    uint32_t reciver      = 0;
    struct timeval selectDelay;
    fd_set    ttyset;
    int       retval;

    FD_ZERO(&ttyset);
    FD_SET(ttyFd, &ttyset);

    while(config.work)
    {
        if (-1 == write(ttyFd, &waitMessage, sizeof(waitMessage)))
        {
            int err = errno;
            perror("Server: waitin failed on write to serial");
            return err;
        }
        usleep(calculate_delay_in_usec(WAITBASEDELAY, WAITBASEDELAY));
        selectDelay.tv_sec = 0;
        selectDelay.tv_usec = BASEDELAY;
        retval = select(ttyFd+1, &ttyset, NULL, NULL, &selectDelay);
        if (-1 == retval)
        {
            perror("Server: polling file descriptor failed");
        }
        if (retval == 0)
        {
            DEBUGMSG("Waiting for answer failed new wait circle");
            continue;
        }
        if (-1 == read(ttyFd, &reciver, sizeof(uint32_t)))
        {
            int err = errno;
            if (errno == EAGAIN)
            {
                DEBUGMSG("TTY port is empty");
                continue;
            }
        }
    }
}
```

```

        perror("Server: read port failed");
        return err;
    }
    reciver = ntohl(reciver);
    DEBUGOUT2("Getting value %X", reciver);
    switch(reciver)
    {
    case WAITMAGIC:
        continue;
    case RECIVEREADYMAGIC:
        return 0;
    default:
        DEBUGOUT("Starng man state %X recived",reciver);
        break;
    }
}
DEBUGMSG("Process aborted before somthing happened");
return -1;
}

/**
 * @brief Осуществляет передачу данных через последовательный порт.
 *
 * Функция осуществляет передачу данных в количестве dataCount
 * стандартных пакетов.
 *
 * @param ttyFd Дескриптор открытого файла порта
 * @param dataCount Количество передаваемых пакетов данных
 *
 * @return 0 при удачном завершении и код ошибки в противном случае.
 */
static int
send_data_to_client(int ttyFd, long dataCount)
{
    DataPack sendData, encodedData;
    long dataSize = dataCount;

    if ((unsigned long)dataCount > (unsigned long)UINT32_MAX)
    {
        DEBUGOUT2("Requested %ld data but can't accept more then %lu", dataCount,
            (unsigned long)UINT32_MAX);
        errno = EINVAL;
        return EINVAL;
    }

    convert_message_to_datapack(messageString, sizeof(messageString), &sendData);

    dataSize = dataCount;

    do
    {
        int res;
        dataSize -= sizeof(DataPack);
        DEBUGPARAMETRS2("%ld", dataSize);
        sendData.dataTotalLength = dataSize;
        convert_datapack_to_network_string(&sendData, &encodedData);
        res = write(ttyFd, &encodedData, sizeof(DataPack));
        if (-1 == res)
        {
            perror("Server main circle - cant write data to port");
            return EIO;
        }
        usleep(10 * calculate_delay_from_speed_usec(tioGetDefL( "PORTSPEED", 115200
    )));
    }
    while (dataSize > 0 && config.work);
    return 0;
}

int
server_process(const Configuration* config)
{
    int rc;

    DEBUGMSG("Starting server process");

```

```

if( tioGetL( "CLIENTSERVERMODE" ) )
    daemon(0,0);

if (0 != (rc = wait_ready_state(config->outputDevice)))
{
    errno = rc;
    perror("Server fatal error: stoped");
    exit(-1);
}
DEBUGMSG("Conformation recived. Starting data transfare");
if (0 != (rc = send_data_to_client(config->outputDevice, config-
>sendPacksLength)))
{
    perror("Server sending fatal error: aborted");
    exit(-1);
}
DEBUGMSG("Server data transfere finished.");
return 0;
}

```

Файл termcontrol.c

```

#define _BSD_SOURCES

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <termios.h>

#include "termcontrol.h"

/**
 * @brief Функция перевода численного значения скорости в константу
 * скорости передачи данных по COM порту.
 *
 * Функция возвращает значение константы скорости для выбранного
 * скоростного режима. В случае если для последовательного порта такой
 * скоростной режим невозможно возвращается константа B0 а значение
 * errno устанавливается равным EINVAL.
 *
 * @param speedValue - численное значение желаемой скорости
 */
static speed_t
choose_speed(long speedValue)
{
    speed_t speed;
    switch (speedValue)
    {
        case 0:
            speed = B0;
            break;
        case 50:
            speed = B50;
            break;
        case 75:
            speed = B75;
            break;
        case 110:
            speed = B110;
            break;
        case 134:
            speed = B134;
            break;
        case 150:
            speed = B150;

```

```

        break;
    case 200:
        speed = B200;
        break;
    case 300:
        speed = B300;
        break;
    case 600:
        speed = B600;
        break;
    case 1200:
        speed = B1200;
        break;
    case 1800:
        speed = B1800;
        break;
    case 2400:
        speed = B2400;
        break;
    case 4800:
        speed = B4800;
        break;
    case 9600:
        speed = B9600;
        break;
    case 19200:
        speed = B19200;
        break;
    case 38400:
        speed = B38400;
        break;
    case 57600:
        speed = B57600;
        break;
    case 115200:
        speed = B115200;
        break;
    case 230400:
        speed = B230400;
        break;
    default:
        speed = B0;
        errno = EINVAL;
    }
    return speed;
}

```

```

int
open_serial_port(const char *path, long speed)
{
    int fd;
    speed_t speedValue;
    struct termios terminalStatus;

    fd = open(path, O_RDWR | O_NOCTTY | O_NONBLOCK);
    if (-1 == fd)
    {
        perror("Open serial device failed:");
        return -1;
    }

    if (tcgetattr(fd, &terminalStatus))
    {
        perror("Get status of serial port failed:");
        close(fd);
        return -2;
    }

    cfmakeraw(&terminalStatus);
    speedValue = choose_speed(speed);
    if (speedValue == B0 && errno == EINVAL)
    {
        perror("Wrong speed parametr selected:");
        close(fd);
    }
}

```

```
        return -3;
    }
    cfsetspeed(&terminalStatus, speedValue);
    if (tcsetattr(fd, TCSANOW, &terminalStatus))
    {
        perror("Can not configure serial port");
        close(fd);
        return -4;
    }
    return fd;
}
```

Приложение 3

Файл runtests.sh

```
#!/bin/bash

state=""

LD_OLD=${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${(pwd)}/lib/

case "$TERM" in
    xterm-color|linux|xterm*)
        color_prompt="yes";
        echo "I have color";;
    *)
        color_prompt='no';
        echo "No colors :(";;
esac

if [ "${color_prompt}" = "yes" ] ; then
    TS="(\033[0;36mTS\033[0m)";
    TSPASS="\033[0;32mPASS\033[0m";
    TSFAIL="\033[0;31mFAIL\033[0m";
    TSFINFAIL="\033[0;31mTESTING FAIL\033[0m";
    TSFINPASS="\033[0;32mAll tests PASS\033[0m";
    RUNMSG="\033[1;34mRun test:\033[0m";
    TEST="\033[1;34mTest \033[0m";
else
    TS="(TS) ";
    TSPASS="PASS";
    TSFAIL="FAIL";
    TSFINFAIL="TESTING FAIL"
    TSFINPASS="All tests PASS"
    RUNMSG="Run test:";
    TEST="Test ";
fi;

if test -z $1 ; then
    echo "ERROR YOU SHULD SHOW WHERE I NEED SEARCH TEST FILES";
fi;

for i in `ls $1/test_*`; do
    echo -e "$TS: $RUNMSG $i"
    if ! ./$i ; then
        echo -e "$TS: $TEST $i [$TSFAIL]";
        state="fail";
    else
        echo -e "$TS: $TEST $i [$TSPASS]";
    fi;
done;

if ls $1/fail_* &> /dev/null; then
    for i in `ls $1/fail_* | egrep -v "\.result$"`; do
        echo -e "$TS: $RUNMSG $i"
        result="";
    done;
fi;
```



```

    if test -f "${i}.result"; then
        result=`cat ${i}.result`;
    fi
    ./${i};
    pres=$?;
    if test -n "${result}"; then
        if test ! "${result}" = "${pres}"; then
            echo -e "$TS: $TEST $i [$TSFAIL]";
            state="fail";
        else
            echo -e "$TS: $TEST $i [$TSPASS]";
        fi;
    else
        if test ${pres} -ne 0; then
            echo -e "$TS: $TEST $i [$TSPASS]";
        else
            echo -e "$TS: $TEST $i [$TSFAIL]";
            state="fail";
        fi;
    fi;
done;
fi;

export LD_LIBRARY_PATH=${LD_OLD}

if [ -z ${state} ]; then
    echo -e "$TS: $TSFINPASS";
else
    echo -e "\a$TS: $TSFINFAIL";
    exit -1;
fi;

```