**Содержание**

[1. Введение 5](#_Toc343454898)

[2. Специальная часть 7](#_Toc343454899)

[2.1. Анализ исходных требований для разрабатываемой библиотеки обработки входных параметров и систематизации выходных данных 7](#_Toc343454900)

[2.2. Разработка соглашений о вызовах функций библиотеки 11](#_Toc343454901)

[2.2.1. Разработка соглашений о вызовах функций обработки ошибок работы библиотеки 11](#_Toc343454902)

[2.2.2. Разработка соглашения о вызове функции инициализации библиотеки 12](#_Toc343454903)

[2.2.3. Разработка соглашений о вызовах функций получения входных параметров программ тестирования 14](#_Toc343454904)

[2.2.4. Разработка соглашений о вызовах функций обработки выходных данных программ тестирования 18](#_Toc343454905)

[2.3. Реализация функций разрабатываемой библиотеки 21](#_Toc343454906)

[2.4. Прототипирование среды исполнения подпрограмм библиотеки 29](#_Toc343454907)

[3. Технологическая часть 36](#_Toc343454908)

[3.1. Профилирование разрабатываемого программного обеспечения 36](#_Toc343454909)

[3.2. Анализ производительности библиотеки интерфейсов 41](#_Toc343454910)

[3.3. Отладка и тестирование разрабатываемой библиотеки 45](#_Toc343454911)

[4. Охрана труда и окружающей среды. Разработка мероприятий по обеспечению благоприятных санитарно-гигиенических условий труда инженера 57](#_Toc343454912)

[Введение 57](#_Toc343454913)

[4.1. Анализ условий труда инженера-программиста 57](#_Toc343454914)

[4.1.1. Характеристика условий труда инженера-программиста 57](#_Toc343454915)

[4.1.2. Анализ освещения, микроклимата и визуальных параметров устройства отображения информации. 60](#_Toc343454916)

[4.2. Разработка мероприятий по уменьшению отрицательного воздействия производственных факторов 70](#_Toc343454917)

[4.2.1. Микроклимат 70](#_Toc343454918)

[4.2.2. Визуальные параметры средств отображения информации 70](#_Toc343454919)

[Вывод по теме 73](#_Toc343454920)

[Список литературы 74](#_Toc343454921)

[5. Экономическая часть. Обоснование экономической эффективности разработки библиотеки функций унификации процессов обработки входных параметров и систематизации выходных данных 75](#_Toc343454922)

[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.” 75](#_Toc343454923)

[5.2. Определение трудоёмкости создания программного продукта 77](#_Toc343454924)

[5.3. Календарное планирование. 79](#_Toc343454925)

[5.4. Определение затрат на создание программного продукта 82](#_Toc343454926)

[5.5. Оценка экономической эффективности 84](#_Toc343454927)

[6. Заключение 89](#_Toc343454928)

[Приложение 1 91](#_Toc343454929)

[Приложение 2 122](#_Toc343454930)

[Приложение 3 135](#_Toc343454931)

# 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 будет 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.

# Специальная часть

* 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 арности)

Некоторые удаленные средства:

Самым заметным "излишеством", удаленным при создании *С99*, было правило "неявного *int*". В *С89* во многих случаях, когда не было явного указания типа данных, подразумевался тип *int*. А в *С99* такое не допускается. Также удалено неявное объявление функций. В *С89*, если функция перед использованием не объявлялась, то подразумевалось неявное объявление. А в *С99* такое не поддерживается. Если программа должна быть совместима с *С99*, 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 жёстким. 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 *С99*

|  |  |  |
| --- | --- | --- |
| *Digital* *Mars*  *VPF*  *Open* *Watcom* *C* | *Intel* *C* *Compiler*  *LCC* | *Oracle* *Solaris* *Studio*  *Pelles C* |

* 1. Разработка соглашений о вызовах функций библиотеки
     1. Разработка соглашений о вызовах функций обработки ошибок работы библиотеки

Все функции библиотеки, которые предназначенные 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, возвращающий 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](file:///C:\Users\mishlen\libtio\Docs\html\d6\dbb\errorout_8c_a29130a8f0f0107da5e3706f4378e89a0.html#a29130a8f0f0107da5e3706f4378e89a0) | ( | 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.

* + 1. 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 не только выделения памяти и задание начальных значений для переменных, массивов и структур, без которых невозможно использовать другие функции разрабатываемой библиотеки, но и производит 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 так же 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, символов 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. Так же 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.

* + 1. 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 ().

|  |  |
| --- | --- |
| 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 .

**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 *namе* получить не удалось, то в буфер *buff*  присваивается 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, |
|  |  | сonst 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.

* + 1. 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 так же 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 значение, 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, то 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 функция. Последовательности 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 завещающаяся нулем ( char\* ) |
| x | Беззнаковое шеснадцатиричное  представления ( long ) |
| X | Беззнаковое шестнадцатеричное  представления с буквами в верхнем  регистре ( long ) |

Таблица .

Для вывода символа % используется последовательность %%.

int tioPrint(const char \* message)

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

int tioPrintF(const char\* template, ... )

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

Префикс «(II):» 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, ... )

Ввыводит 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.

* 1. 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).

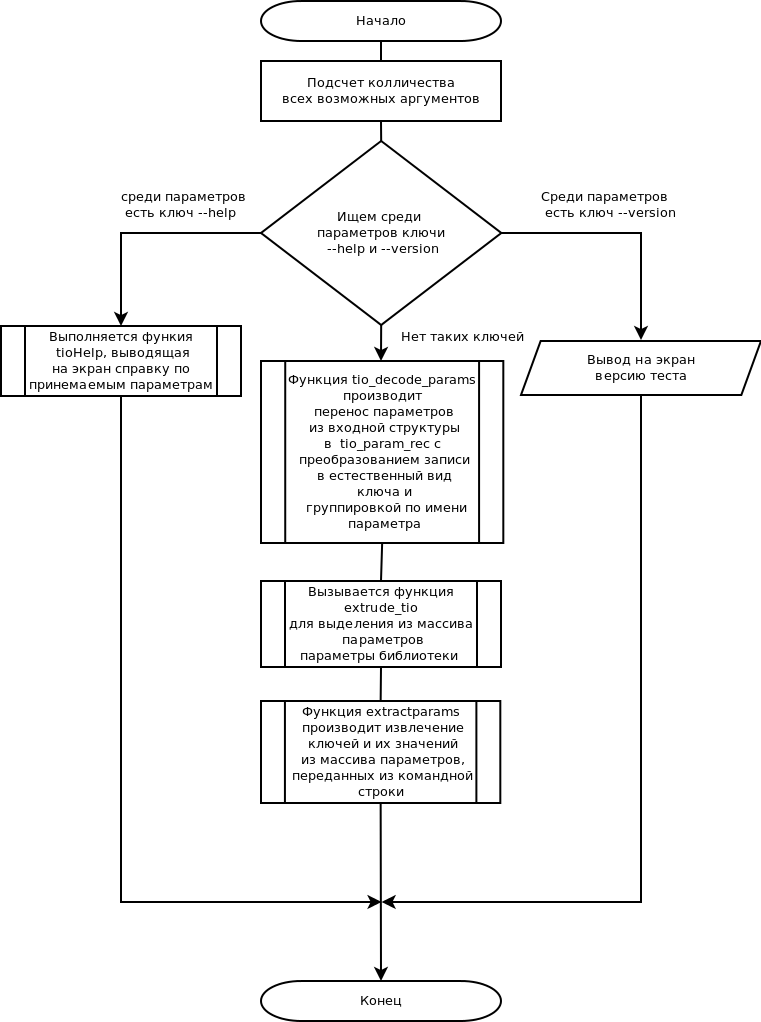
Git изначально идеологически ориентирована на работу с изменениями, а не с файлами, «единицей обработки» для нее является набор изменений, или патч.

Преимуществами *git* перед другими системами контроля версиями:

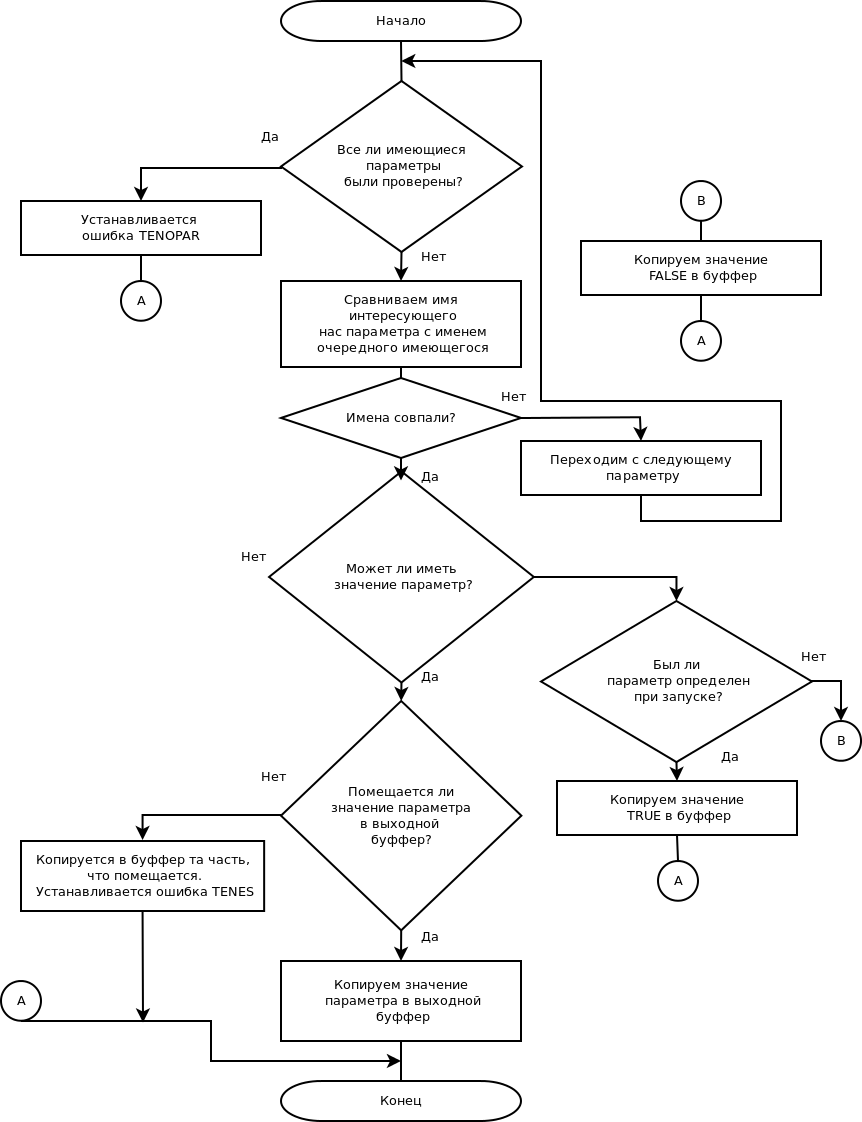
* Высокая производительность.
* Продуманная система команд, позволяющая удобно встраивать *git* в скрипты.
* Репозитории *git* могут распространяться и обновляться общесистемными файловыми утилитами архивации и обновления благодаря тому, что фиксации изменений и синхронизации не меняют существующие файлы 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().



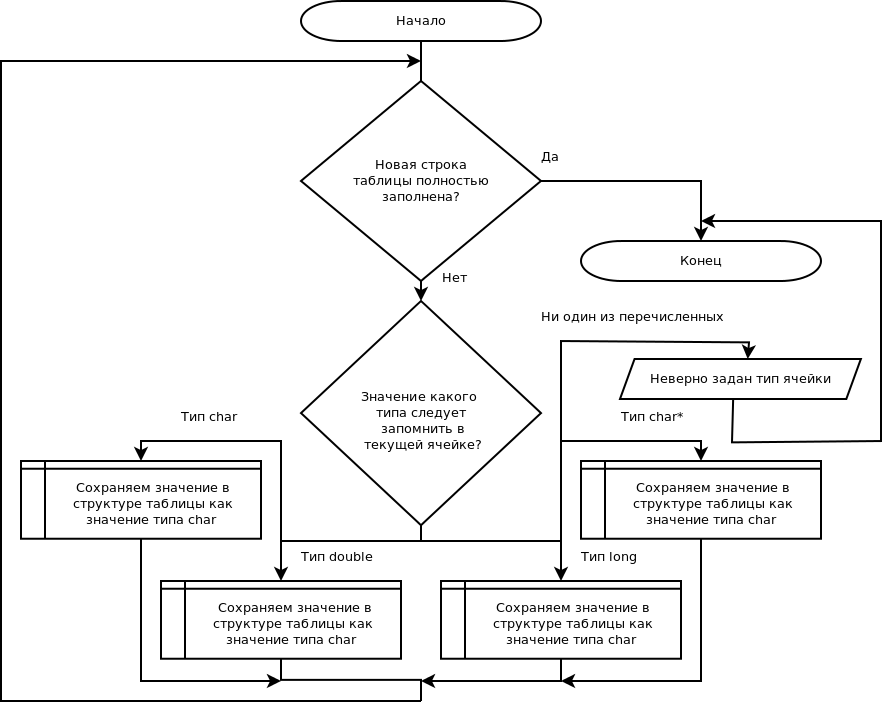
Функция tioGetS()



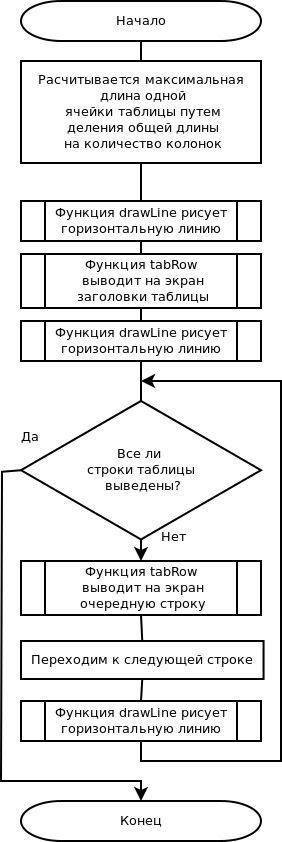
Функция tioTableBegin()



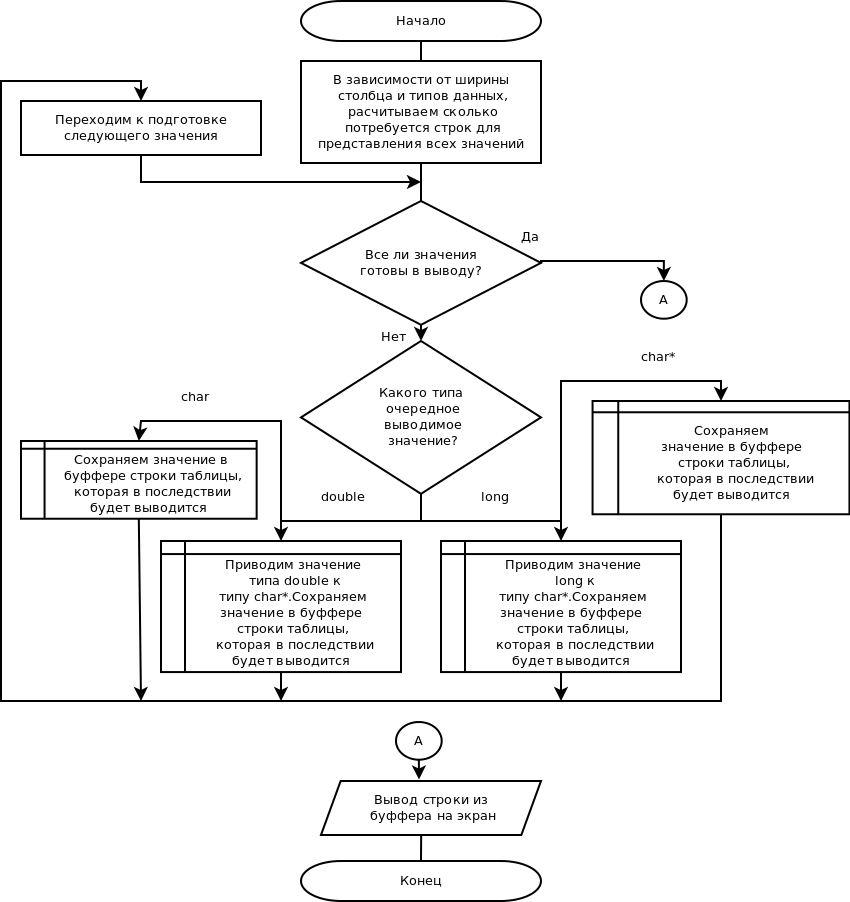
Функция tioTableRecord()



Функция tioTableEnd()



Функция tabRow()



* 1. 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 . 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 что бы убедиться в корректности работы функции решения квадратных уравнений, напишем 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, запускающая 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 .

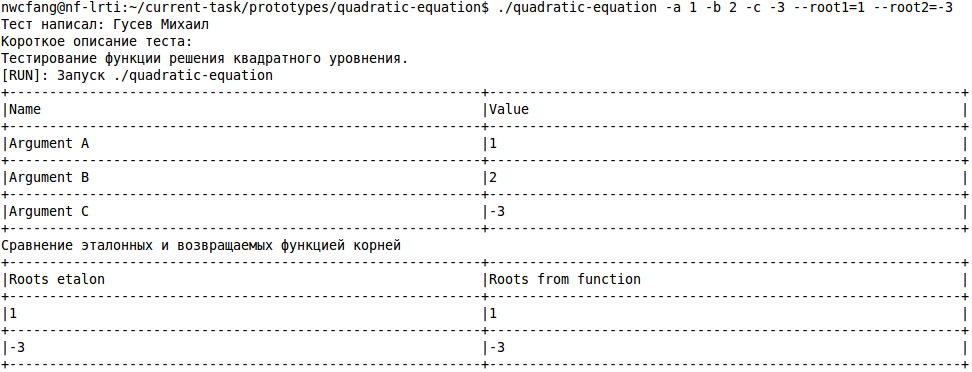


Рис. .

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*, переданному при вызове тестирующей программы, использующей библиотеку *libtio*, вместо выполнения теста на экран выведет список аргументов, которые можно передать из командной строки ().

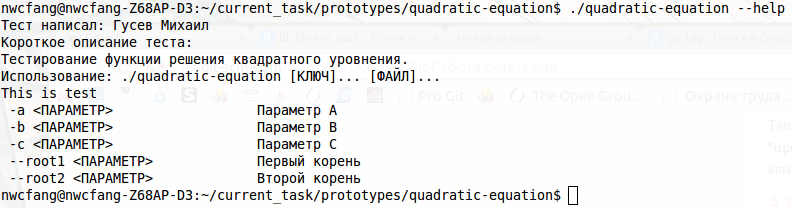


Рис. .

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

C:\Users\mishlen\diplom_report\prototypes\quadratic-equation\version.png

Рис. .

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, // minimu m transfered 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, //minimu m transfered 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. .

# Технологическая часть

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

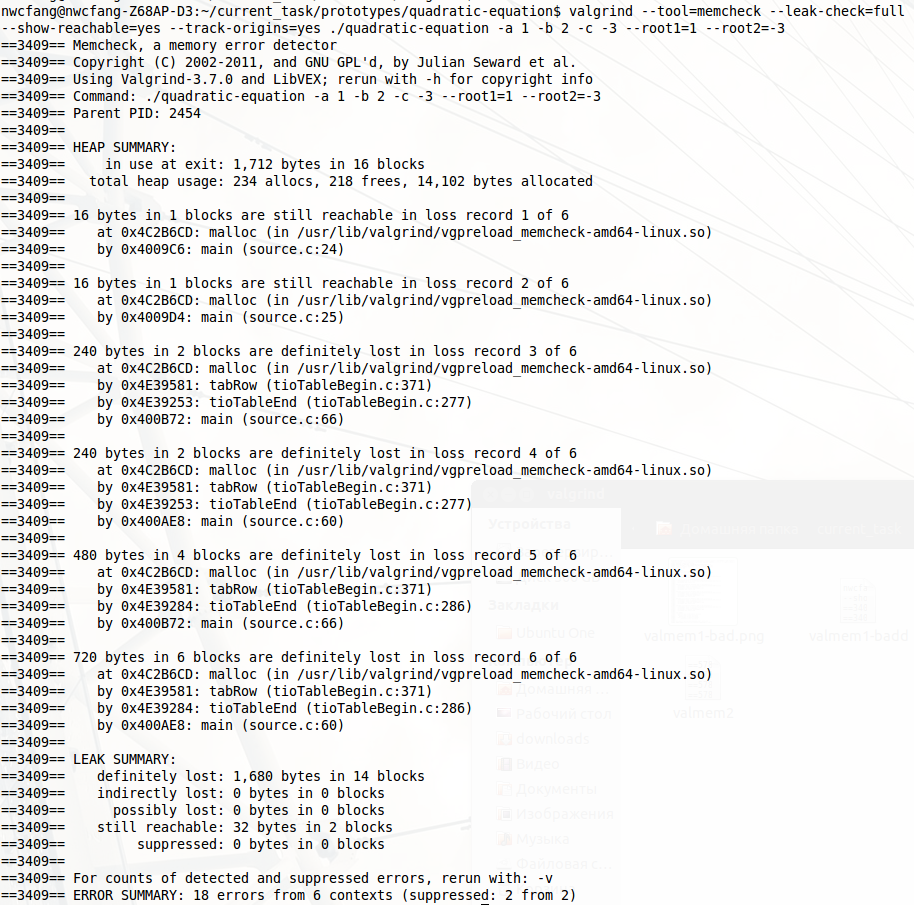


Рис. .

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 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.

bad-callgraph.tif

Имя вызываемой функции

Число операций процессора

Количество вызовов функции

Рис. .

* 1. Анализ производительности библиотеки интерфейсов

По результатам работы профилировщика *Valgrind*(инструмент профилировки *Memcheck*) из 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 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 .

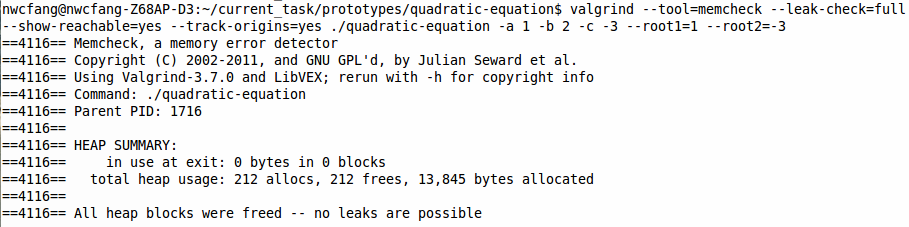


Рис. .

Теперь проанализируем данные о числе вызовов функция, полученных с помощью инструмента *Callgrind*. Из 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* (). Теперь оно составляет около 5% от общего времени выполнения программы.

good-callgraph (2).tif

Рис. .

* 1. 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 рисунках – .

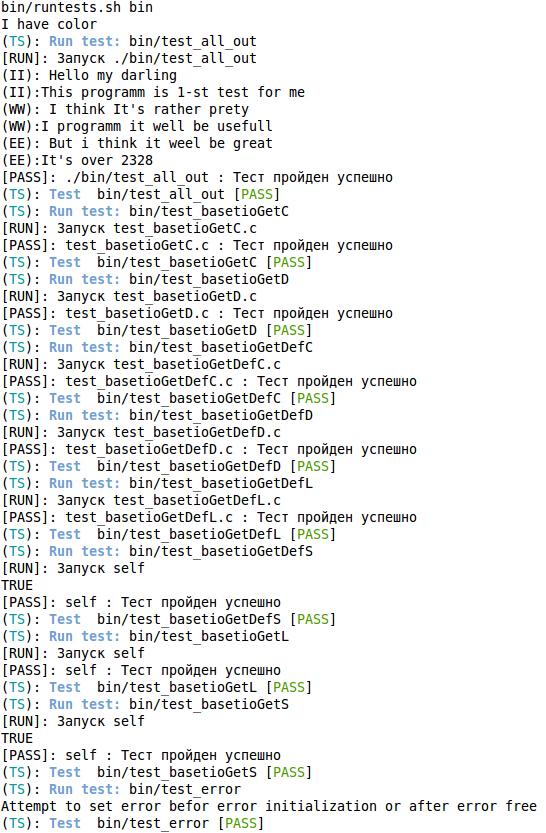
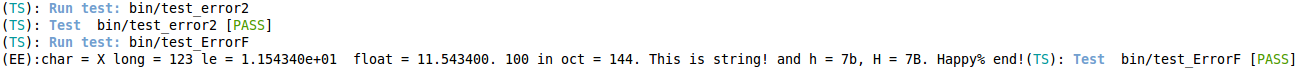


Рис. .



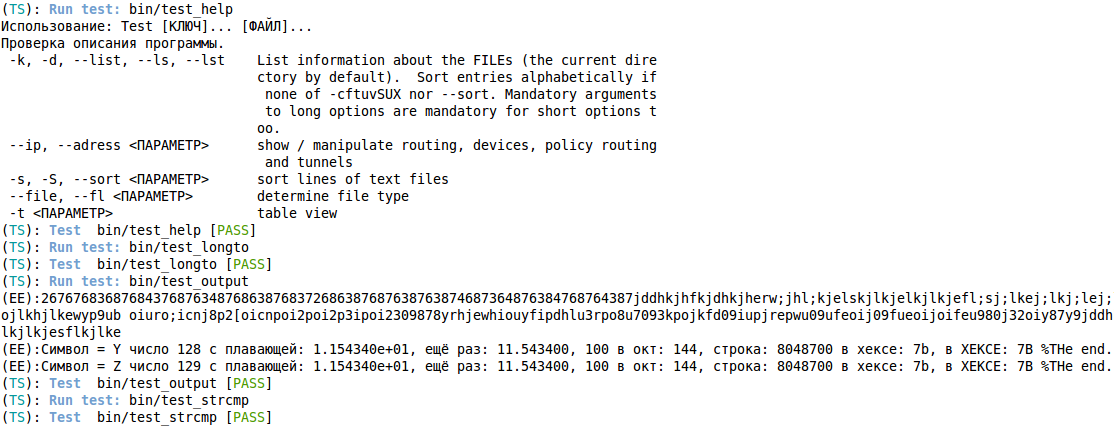


Рис. .

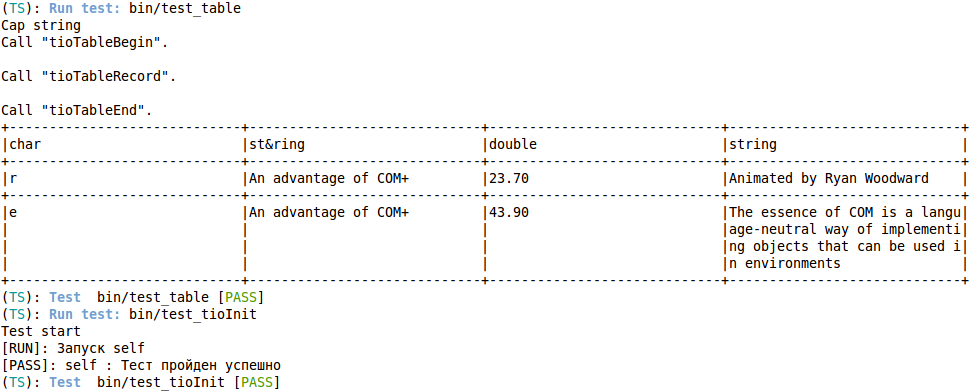


Рис. .

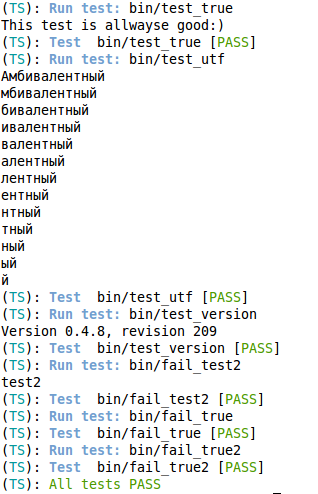


Рис. .

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

Вначале сценария приписываем в переменную окружения LD\_LIBRERY\_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\_LIBRERY\_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 ().

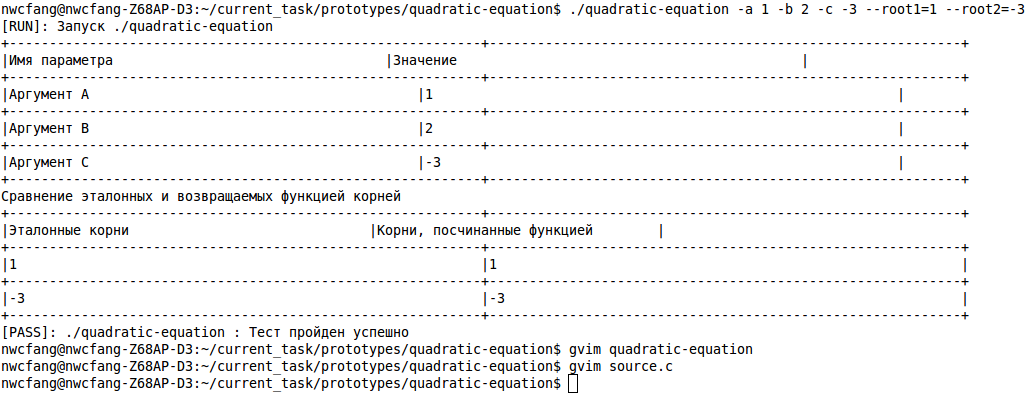


Рис. .

При отладке было установлено что, так как 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 ().

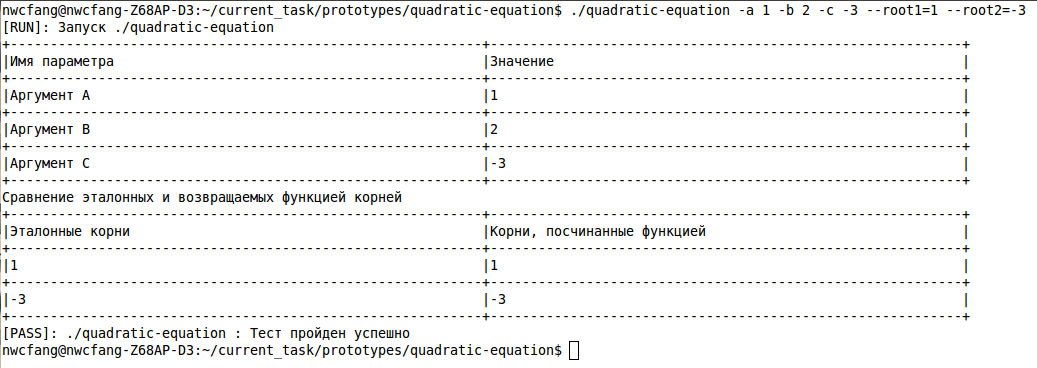


Рис. .

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

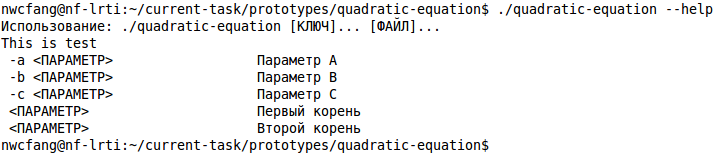


Рис. .

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 ().

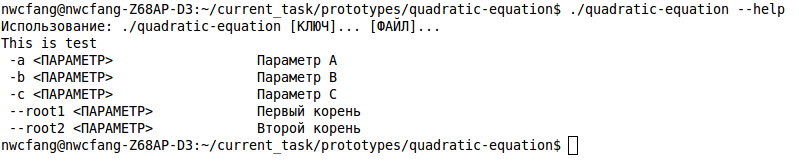


Рис. .

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 ().

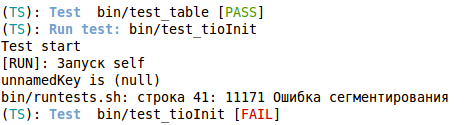


Рис. .

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 .

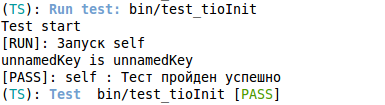


Рис. .

# 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.

* 1. Анализ условий труда инженера-программиста
     1. Характеристика условий труда инженера-программиста
        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.

* + - 1. 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/м2 |
| Контрастность | 1000:1 |
| Потребляемая мощность | при работе: 40 Вт, в спящем режиме: 0.50 Вт |
| Частота обновления экрана | 60Гц |

* + - 1. Количество работающих людей

В помещении работают 6 человек.

* + - 1. Характеристика помещения

Работа ведется в офисном помещении длиной L = 6,6м, шириной W = 5,4м и высотой H = 4м. Соответственно, площадь помещения составляет S = L·W = 6,6·5,4 = 35,64 м2, а его объём V = S·H = 35,64·4 = 142,56 м3.

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 T8 фирмы OSRAM L 18W/640 25X1 (световой поток Φл = 1350 лм) и имеют сертификат соответствия ГОСТ:

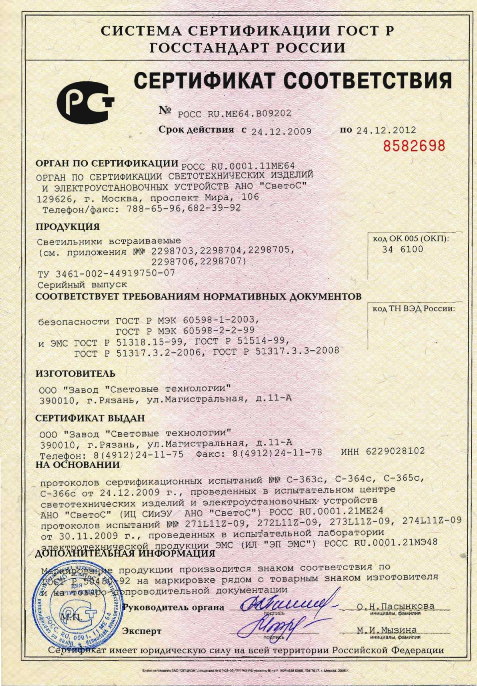


Рис. .

* + 1. Анализ освещения, микроклимата и визуальных параметров устройства отображения информации.
       1. Освещение

1. **Искусственное освещение**
2. **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 это точки а и б).

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

где:

* *H* – высота потолка;
* *hс* – высота свеса светильника;
* *h*р.п. – высота рабочеё поверхности.

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

а

б

4,2 м

6,6 м

3,3 м

1

2

4

3

1,5м

2,4 м

5,4 м

Рис. .. Осветительная установка.

1. **Формула расчёта фактической освещённости точек а и б.**

Фактическая освещённость определяется по формуле:

,

где

* Ф – суммарный световой поток всех источников, лм;
* μ = 1,1 …1,2 – коэффициент, учитывающий отражённую составляющую света и действий удалённых светильников (μ =1,1);
* Σε – сумма относительных освещённостей от нескольких светящих линий;
* kз – 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 (kз = 1,5);

* h – высота подвеса светильников над рабочей поверхностью;
* L – общая длина светящих линий, м

1. **Определение суммарного светового потока от всех источников, Φ.**

,

где

* Φл – световой поток лампы, лм.

Световой поток от 1 лампы (см. п.) Φл = 1350 лм.

* – количество ламп в одном светильнике;
* – количество светильников в одном ряду;
* – количество рядов светильников.

Следовательно:

1. **L – общая длина светящих линий.**

Где:

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

Высота рабочей поверхности над полом равна 0,8м.

Отсюда:

* – длина светильника;

Таким образом:

1. **Определение суммы относительных освещенностей от нескольких светящих линий (Σε).**

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

*и ,*

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

и – приведенные размеры

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

**Для определения относительной освещенности ε, лк, находим:**

1. 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.

1. Определяем p, L, p', L' для каждой точки.

**Точка *а*:**

для всех полурядов ⇒ ;

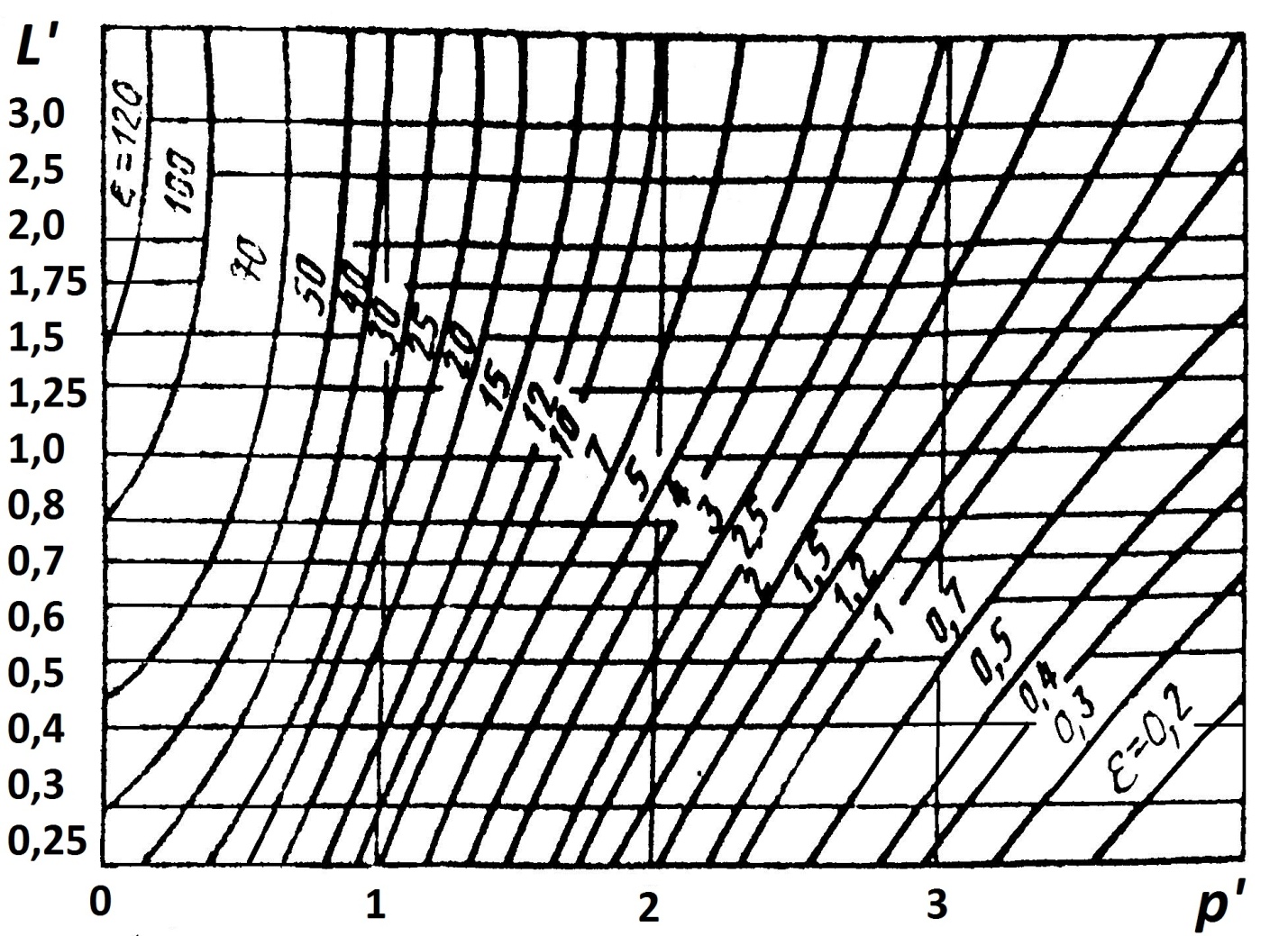
для полурядов 1, 2 и для полурядов 3, 4 ( и соответственно).

**Точка *б*:**

для всех рядов⇒ ;

для ряда 1-2 и для ряда 3-4 ( и соответственно).

1. По графику линейных изолюкс () по p', L' определяем относительную освещенность ε для каждого полуряда и ряда светильников, которые освещают точку (сначала точку а, а потом – точку б).



0,75

0,66

0

1,3

Рис. .. 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  3, 4 | 0  2,4 | 2,1  2,1 | 0  0,75 | 0,66  0,66 | 2×85=170  2×38=76  Σε=246 |
| б | 1-2  3-4 | 2,4  0 | 4,2  4,2 | 0,75  0 | 1,31  1,31 | 1×50=130  1×120=120  Σε=170 |

1. **Окончательный расчёт фактической освещённости**

Учитывая, что , , имеем:

1. **Сравнение полученных результатов с нормативными значениями**

Работа за пультами ЭВМ, дисплеев относится к III разряду зрительных работ (подразряд *г*) 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.

1. **Естественное освещение**
2. **Коэффициент естественного освещения**

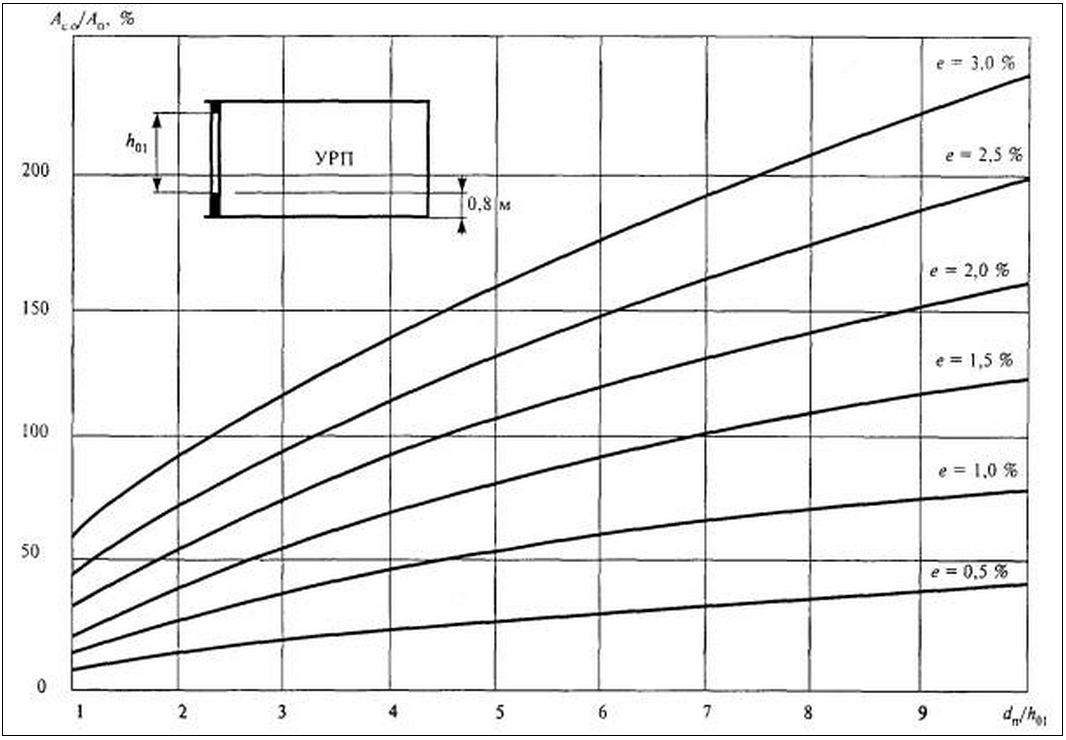
В соответствие с характеристикой помещения, приведённой в п., площадь офисного помещения, в котором работает инженер-программист, равна Ап = 35,64 м2, а его глубина (расстояние от стены, где расположены оконные проёмы, до противоположенной стены) – dп = 6,6 м.

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

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

Таким образом:

Для определения КЕО воспользуемся графиком, представленным на .



36

2,2

Рис. .. График для определения относительной площади световых проемов Ас.о./Ап при боковом освещении помещений общественных зданий.

По графику находим, что e = 1,3%.

Нормированное значение КЕО, еN, определяем по формуле

где

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

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

Окончательно, *eN* = 1,2·1 = 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.

* + - 1. 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 Iа) – 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 - Iа | 22-24 | 21- 25 | 18-26 | 40-60 | 75 | 0,1 | 0,1 |
| Теплый | Легкая - Iа | 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.

* + - 1. 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 +-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 × 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. , можно сделать вывод, что монитор соответствует требованиям СанПиН 2.2.2/2.4.1340-03.

* 1. Разработка мероприятий по уменьшению отрицательного воздействия производственных факторов
     1. Микроклимат

Исходя из анализа микроклимата в офисном помещении, проведенного в п. , 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. ..

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).

* + 1. 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 012а1663/0921-04, 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.

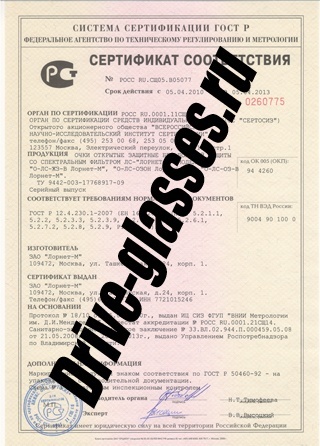


Рис. 4.6. Сертификат подтверждает, что очки компании ЗАО Лорнет-М являются Средством Индивидуальной Защиты (СИЗ) и соответствуют ГОСТ Р 12.4.230.1-2007.



Рис. 4.. 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 требованием 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 требованием безопасности.

Список литературы

1. *Сибаров Ю.Г. и др*. Охрана труда в вычислительных центрах. Учебник. – М.: Машиностроение, 1990.
2. *Березин В.М., Дайнов М.И.* Защита от вредных производственных факторов при работе на ПЭВМ. – М.: Изд. МАИ, 2003.
3. ГОСТ 12.1.005-88 «Общие санитарно-гигиенические требования к воздуху рабочей зоны».
4. СНиП 23-05-95 «Естественное и искусственное освещение».
5. СанПиН 2.2.2./2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организация работы».
6. СанПиН 2.2.4.548–96 «Физические факторы производственной среды гигиенические требования к микроклимату производственных помещений»

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

* 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 .

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

Индекс технического уровня проектируемого изделия:

где - уровень i-й функционально-технической характеристики соответственно базового и нового программного продукта;

– 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

Значение коэффициента влияния Кв выбирается из таблицы 5.2

|  |  |
| --- | --- |
| Наименование проектируемой техники | Кв |
| 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 .

Разработанное устройство относится к пункту “Прочая комплектующая техника”, следовательно Kв =0,15

Подставив коэффициент в формулу для технического уровня получим:

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

* 1. Определение трудоёмкости создания программного продукта

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

|  |  |  |
| --- | --- | --- |
| N 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 .

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 (tПР Т) в чел.-час. Определяются следующим образом:

Где:

tи - затраты труда на изучение и постановку задачи;

tА – затраты труда на разработку алгоритма решения задачи;

tк – затраты труда на программирование по блок-схеме;

tот – затраты труда на отладку программы;

tд – затраты труда на подготовку документации по ПП;

Для расчетов необходимо знать:

* q- количество этапов и элементарных процедур преобразования информации; (q=120)
* Kc – коэффициент сложности программы Kc=1,25;…;2,0; (Kc=1,4)
* Kk – коэффициент коррекции, при разработке Kk = 0,05…;0,1;(Kk=0,065)
* n- количество коррекций;(n=60)
* K – коэффициент квалификации разработчика, программиста;(K=0,8 стаж до 2х лет)
* B = 1,2…;3,0;- увеличение затрат на изучение и постановку задачи вследствие ее сложности и новизны. (В=2)

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

Теперь подставим значение Q в формулы для определения затрат труда:

Таким образом, суммарная трудоемкость работы составляет:

(чел.-час.)

* 1. 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:

 ,

где Тэi – трудоемкость этапа, чел.-ч.;

tрд – 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 .

* 1. Определение затрат на создание программного продукта

Зарплата персонала по стадиям работ рассчитывается по формуле:

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

- 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 .

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: | | ЗПП=381886,596 | 100 |

Таблица .

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 определятся по отношению к заработной плате основных исполнителей:

Где - коэффициент накладных расходов; для ПП принимается на уровне: 1 ≤ ≤2,0

Таким образом

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

где Зпп – текущие затраты на создание ПП;

ЗПпп – оплата труда персонала в общих текущих затратах на создание ПО;

ρзп – уровень рентабельности (прибыли по отношению к оплате труда персонала), обеспечивающий безубыточность деятельности (ρзп = 200-400%).

Примем ρзп = 200% ,тогда цена разрабатываемого программного продукта будет равна:

* 1. 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 .

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 определяется укрупнено на основе анализа динамики эксплуатационных затрат по отношению к повышению качества диагностики:

Где:

– эксплуатационные затраты на выполнение диагностических процедур для одного объекта, до применения ПП и в новом варианте:

Где:

- затраты на оплату труда персонала, осуществляющего диагностику

- годовые амортизационные отчисления по вычислительной технике

- затраты на электроэнергию по вычислительной технике

- прочие затраты

- количество объектов, диагностируемых за год

- коэффициент опережения повышения качества диагностики по сравнению с ростом эксплуатационных затрат

Где:

- индекс изменения эксплуатационных затрат

- уровень качества диагностики

Выполним расчеты:

* количество объектов, диагностируемых за год
* годовые амортизационные отчисления по вычислительной технике

Где:

* *СВТ* – стоимость вычислительной техники
* руб.
* *НАВТ* – годовая норма амортизационных отчислений (25 %)
* *dис* – коэффициент использования мощности информационной системы для решения данной задачи

ТМ.Г. – машинное время, используемое в течение года для реализации данного ПП, час.;

Fэфф.ВТ – годовой эффективный фонд времени работы вычислительной техники, час.

Таким образом:

* затраты на электроэнергию

Где:

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

СЭЛ – стоимость одного кВт·ч электроэнергии, руб. (2,05)

* затраты на оплату труда персонала, осуществляющего диагностику одного объекта, руб.:
* Затраты в базовом варианте
* Затраты в новом варианте
* прочие затраты, руб.:
* Затраты в базовом варианте
* Затраты в новом варианте

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 .

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

Где:

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

Ц – цена нового продукта

(см. п.3.4)

Тогда:

Рассчитаем срок окупаемости затрат:

**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.

# 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.

# 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 заием 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 Колличество ключей

\*

\* @return 0 при упешном выполнени 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 == ll && 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[ll-1] == ':')

{

obj[i].per->has\_key=1;

obj[i].str[ll-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 индеса к старшему исключить что более короткий

\* ключ перекроет более длинный

\*

\* @param[in,out] obj Массив подвергающийся сортировки.

\*

\* @return 0 в случае успеха и неноль 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 терминируя их

\* записью содержащей в поле per значение NULL. Записи str в longkey

\* заполнются 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 при работе с shortkeys 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 полей keys и skeys

\*

\* @param keys Сгруперованный массив структур tio\_key\_string в котором

\* если два элемента ссылаются на родительский элемент а межу ними не

\* можеты 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 введеных параметров, перенос из входной структуры в

\* tio\_param\_rec с преобразованием записи в естественный вид ключа и

\* группировкой по имени параметра.

\*

\* @param params Список пораметров переданых 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;

}

/\*\*

\* Фцнкуия извлечения параметров библиотеки из переданных программепараметров

\*

\* @param argc Количество параметров передаваемых функции

\*

\* @param argv Массив параметров для обработки

\*

\* @param[out] cnt Колличество оставшихся элементов

\*

\* @return Массив строк с параметрами не подпадающими под шаблоню Все

\* эти строки должны быть освобожден 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;

}

}

}

}

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 TEEPICFAILL;

#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

}

if (extractparams(0, cnt, mass))

{

#ifndef NDEBUG

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 ресчурсов 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%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%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)

{

#ifndef NDEBUG

fputs("Internal pointers for deque already initializated", stderr);

#endif

pthread\_mutex\_unlock(&errlock);

return TEINTMC;

}

if(!(obj=malloc(sizeof(errdeque))))

{

#ifndef 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 )

{

#ifndef NDEBUG

fputs("Attempt to set error befor 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))))

{

#ifndef 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))))

{

#ifndef 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)

{

#ifndef NDEBUG

fputs("Attemp to use uninitializate 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);

}

}

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; /\*Колличество символов в строке\*/

int countColum; /\*Колличество 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("Неправельный 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->countColum;

//printf("Размер колонки = %d\n", lenColCon);

if( (masType = (int \*) malloc(datTab->countColum \* sizeof(int))) == NULL)

{

printf("ERROR!");

exit(EXIT\_FAILURE);

}

/\*Draw line\*/

drawLine(lenColCon);

for( i = 0; i < datTab->countColum; ++ i )

{

masType[i] = 4;

}

/\*Display cap\*/

tabRow( (void \*)datTab->cap, masType, datTab->countColum, lenColCon);

/\*Draw line\*/

drawLine(lenColCon);

while( datTab->ptr->n != NULL )

{

/\*Print table row\*/

tabRow( datTab->ptr->s, datTab->bufType, datTab->countColum, 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->countColum; ++ 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->countColum; ++ 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 countColum, 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(countColum \* sizeof(int))) == NULL )

{

printf("ERROR!");

exit(EXIT\_FAILURE);

}

/\*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];

}

/\*Allocating memory for data\*/

if((data = (char \*\*\*) malloc (countColum \* sizeof(char \*\*))) == NULL)

{

printf("ERROR!\n");

exit(EXIT\_FAILURE);

}

for( i = 0; i < countColum; ++ 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 < countColum; ++ 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 < countColum; ++ i )

printf("|%s", data[i][strCounter]);

printf("|\n");

}

/\*FREE \*/

for( i = 0; i < countColum; ++ 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 countColum, char \*\*cap, int lenColCon )

{

int i;

int j;

int tmp;

for(i = 0; i < countColum; ++ 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 А"},

{"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, посчинанные 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))

{

DEBUGPARAMETRS("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(RECIVEREADYMAGIC);

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");

DEBUGPARAMETRS("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, //minimu m transfered 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 цыкле размещает в COM 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 успеного 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("Starnge 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 скороси

\*/

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