Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

Воронежский государственный технический университет

ОСНОВЫ ПРОГРАММИРОВАНИЯ И АЛГОРИТМИЗАЦИИ

Методические указания
к выполнению курсового проекта
для студентов бакалавриата направления 09.03.02
«Информационные системы и технологии»

Воронеж 2016

УДК 004.424
ББК 32.973.26-018

Составитель О.В. Минакова, О.В. Курипта

Основы программирования и алгоритмизации: метод.указания к выполнению курсового проекта для студ. бакалавриата направления 09.03.02 «Информационные системы и технологии» / Воронежский государственный технический университет ; сост.: О.В. Минакова, О.В. Курипта. – Воронеж, 2016. – 37 с.

Приводятся методические рекомендации по подготовки курсового проекта по дисциплине «Основы программирования и алгоритмизации, рекомендуемая последовательность выполнения курсового проекта, а также правила оформления кода и пояснительной записки к проекту.

Предназначены для студентов бакалавриата направления 09.03.02 «Информационные системы и технологии» всех форм обучения.

Ил. 17. Табл. 3 Библиогр. 10 назв.

УДК 004.424
ББК 32.973.26-018

Печатается по решению учебно-методического совета

Рецензент – А.А. Кононов, профессор каф. информатики и графики ВГТУ
д-р. техн. наук

ВВЕДЕНИЕ

Курсовой проект представляет собой этап самостоятельного изучения программирования на языке высокого уровня. Отличительной особенностью курсового проектирования является наличие законченного инженерного решения – проекта, модели, программы. Результатом курсового проектирования по дисциплине «Основы программирования и алгоритмизации» является реально функционирующая программа, решающая поставленную задачу.

Целью проектирования является разработка программного продукта с заданным функциональным назначением, включающая весь процесс создания программы от описания понимания поставленной задачи и ее анализа, многократных ее реализациях на языке программирования (редактирование, поиск и исправление ошибок) до появления отлаженного законченного программного продукта с детальным описанием.

В процессе работы над курсовым проектом студенты должны уточнить постановку задачи, произвести анализ технической литературы, провести проектирование программы, кодирование программных функций, сборку и испытание системы, правильно оформить пояснительную записку и программные документы, и защитить курсовой проект.

Выполнение проекта требует творческого подхода к выбору метода решения и построению алгоритма, при этом трудоемкость и сложность процесса кодирования и отладки требует от студента целеустремленности, высокой трудоспособности и дисциплины.

В результате выполнения курсового проекта студент должен научиться работать с научно-технической справочной литературой, выполнять самостоятельную разработку несложного программного продукта, оформлять проектную документацию в соответствии с требованиями ЕСКД.

1. Организация выполнения курсового проекта

Курсовой проект (далее проект) – учебная работа, содержащая решение поставленной задачи по отдельной учебной дисциплине, оформленная в виде конструкторских, технологических, программных и других проектных документов. Основной целью и содержанием проект должен способствовать выработке определенных компетенций и развитию навыков и умений путем решения конструкторских или (и) технологических задач, проведения инженерных расчетов, оформления графической части проекта, а также подготовке студентов к творческому решению конкретных задач при выполнении выпускной квалификационной работы (ВКР).

Целью курсового проектирования по дисциплине «Основы программирования и алгоритмизации» в соответствии с требованиями ФГОС

ВО направления подготовки 09.03.02 «Информационные системы и технологии» является формирование следующих компетенций:

- владение культурой мышления, способность к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения, умение логически верно, аргументировано и ясно строить устную и письменную речь (ОК-1);
- понимание социальной значимости своей будущей профессии, обладание высокой мотивацией к выполнению профессиональной деятельности (ОК-4);
- способность проводить предпроектное обследование объекта проектирования, системный анализ предметной области, их взаимосвязей (ПК-1);
- способность проводить техническое проектирование (ПК-2);
- способность проводить рабочее проектирование (ПК-3).

Задание на курсовое проектирование (ТЗ – техническое задание) выдается индивидуально каждому студенту на 1-2 недели семестра. Примерное содержание представлено в приложении 1 (задача А– первый уровень сложности, задача Б – второй).В ходе выполнения проекта руководитель может, при успешном решении задачи А или Б, изменить исходные данные или расширить функционал приложения для повышения их уровня сложности.

Тема курсового проекта может быть предложена самим студентом при условии обоснования им ее целесообразности, соответствия содержания дисциплине и возможности освоения необходимых компетенций.

В процессе выполнения курсового проекта студенты должны:

1. Составить развёрнутое описание поставленной задачи;
2. Выполнить анализ задания, выбрать модель решения задачи (алгоритм, формулы), обосновать структуру входных и выходных данных;
3. Разработать программу, отладить ее и подготовить документацию к ней.

Календарный план-график курсового проектирования

Неделя семестра	1-2	3-4	5-11	11	13-15	16	17
Этап	Подготовка ТЗ	Выбор модели/алгоритма решения. Уточнение ТЗ	Кодирование, отладка программы	Проверка реализации на соответствие ТЗ	Оформление работы	Проверка ПЗ	Защита

Руководство курсовым проектом осуществляется путем индивидуальных и групповых консультаций. В ходе консультаций преподавателем разъясняются элементы балльной раскладки по отдельным элементам выполнения проекта, поясняется назначение, конкретизируется содержание и порядок выполнения отдельных частей курсовой работы, даются ответы на вопросы студентов.

По результатам выполнения работы оформляется пояснительная записка (так называется текстовый документ курсового проекта). Структура представлена на рис.1.

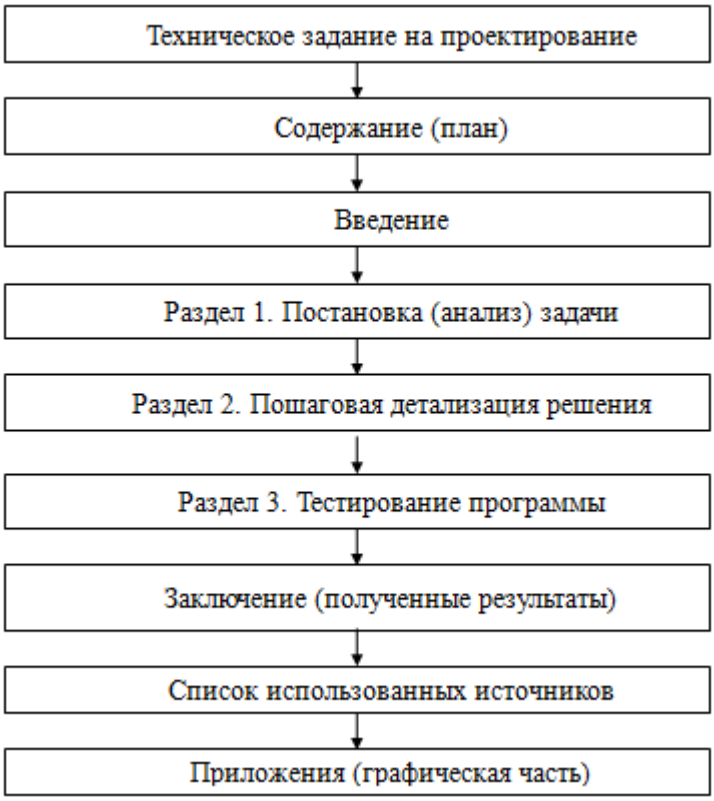


Рис. 1. Примерная структура курсового проекта

Во введении приводится актуальность темы курсового проекта, назначение и возможная область применения создаваемого приложения, степень новизны, значимость и границы разработки, формулируются цель курсового проекта, основные задачи, решаемые в работе.

Основная часть включает:

- подробное описание выбранного или разработанного самостоятельно алгоритма;
- обоснование выбора структур данных для представления исходных данных, результатов и промежуточных вычислений;
- руководство для пользователя, в котором описывается, как применять созданную программу;
- описание тестирования программы;
- результаты применения программы для решения поставленной задачи.

Графическая часть проекта может содержать схемы примененных алгоритмов, структуры исходных и обработанных в программе данных, листинги программных функций, структурную схему программы (иерархию функций и их описание), скриншоты примеров работы и результаты прохождения контрольных тестов (содержание выходных файлов).

В заключении приводятся основные выводы, характеризующие в сжатом

виде итоги проделанной работы, излагаются предложения и рекомендации по внедрению полученных результатов и дальнейшему развитию темы. В заключении не допускается повторения разделов содержания, введения или основной части. Заключение должно соответствовать основным пунктам задания на курсовую работу.

Пояснительная записка должна быть сдана на проверку не менее чем за 3 дня до даты защиты.

Защита курсового проекта является обязательным заключительным этапом курсового проектирования. Защита проводится до последней недели перед началом сессии. На защиту студент предоставляет:

– техническое задание;

– программное средство (ПС);

– пояснительную записку, содержащую описание этапов разработки ПС и соответствующие иллюстрации, а также разработанную программную документацию.

На защите курсового проекта студент коротко (2–3 мин.) докладывает об основных проектных решениях, принятых в процессе разработки, и отвечает на вопросы.

Оценка работы складывается из следующих составляющих.

1 – поставленная задача реализована в программе;

2 – в пояснительной записке представлены алгоритмы каждой из разработанных функций, модель их взаимодействия, описаны входные и выходные данные;

3 – правильно оформленный код программы, точно соответствующий обоснованному алгоритму и выбранным структурам данных;

4 – полно реализован функционал с проверками корректности ввода данных и необходимыми пояснениями, разработан понятный, красивый и удобный интерфейс ввода/вывода;

5 – соответствующее правилам оформление пояснительной записки и успешный ответ на защите работы.

Студенту, не предоставившему курсовой проект до окончания зачетной недели, в ведомости выставляется «не аттестован», и он считается неуспевающим по дисциплине.

2. Разработка основной части курсового проекта

2.1 Постановка (анализ) задачи

Разработка любой программы начинается с анализа поставленной задачи, на этом этапе важно правильно выделить функции, которые будут в дальнейшем реализовываться. На этом этапе важно определить все требования, включая передаваемые параметры, глобальные структуры и переменные, вызываемые подпрограммы и т.д.

Одним из важных технологических приемов программирования является декомпозиция решаемой задачи на подзадачи, т.е. более простые с точки зрения программирования части исходной задачи. Это естественная логическая последовательность мышления конструктора – постепенное углубление в детали, в частности при программирования на языке Си осуществляется деление на функции (рис.2)

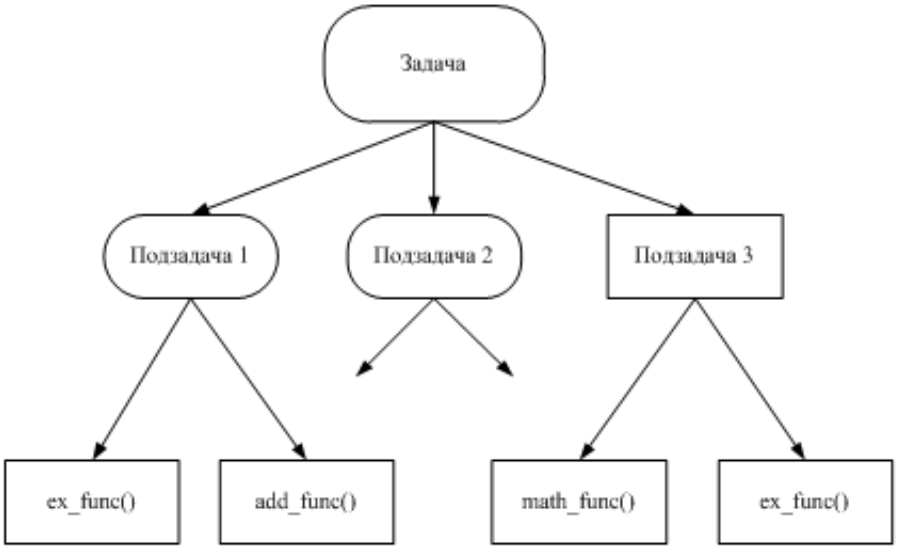


Рис. 2. Декомпозиция решаемой задачи

Суть метода пошаговой детализации состоит в анализе исходной задачи с целью выделения подзадач (пример 1,2) и построения иерархии этих подзадач (схема на рис. 2) [2,3].

Пример 1. *Разрабатываемая программа предназначена для анализа успеваемости студента на основе информации о текущих оценках, содержащейся в текстовом файле.*

Для выполнения поставленной задачи программа должна выполнять следующие функции:

– открыть заданный файл указанный пользователем;

– определить количество студентов с «плохими оценками»;

– сортировать записи о студентах, имеющих оценки с баллами два и ниже по убыванию;

– выполнить запись результатов сортировки в текстовый файл.

Пример 2. *Дано арифметическое выражение, записанное двумя рациональными числами, между которыми расположен знак операции. Необходимо после знака равно выводить результат выполнения операции.*

Следовательно, пользователем задается символьная строка вида:

A#B=res

где A, B – произвольные рациональные числа;

*# – знак арифметической операции (возможно сложение +, умножение *, деление /, вычитание -, возведение в степень ^);*

= – обязательный символ;

res – значение результата расчета.

Программу решения такой задачи назовем «Простой калькулятор». Простой калькулятор должен расшифровывать содержание строки и выполнять соответствующую арифметическую операцию.

Для выполнения поставленной задачи программа должна выполнять следующие функции:

1. Ввод строки.

*2. Синтаксический контроль (строка должна состоять из цифр, знаков операций +, *, /, -, ^ обязательного символа =, пробелов. Проверка осуществляется для выявления недопустимых символов операций и корректности ввода операндов).*

3. Семантический контроль (строка должна быть построена по схеме A#B=res, сообщение об ошибке должно выводиться, если какой-то элемент отсутствует или нарушен их порядок).

4. Выполнение операции (в зависимости от заданной арифметической операции).

5. Вывод результата.

Эта декомпозиция будет предопределять и методику решения задачи (пример 3), а также выбор типа входных и выходных данных. Функционирование любого программного приложения можно рассматривать, как обработку некоторого входного потока данных. Данные поступают от входа, преобразуются по правилам решения задачи, и в преобразованном виде передаются к выходу (внешним пользователям).

Поэтому задача определения способа хранения, порядка и правил организации данных, является очень важным элементом разработки программы. Четкое структурирование данных, выполняемое в процессе разработки, способствует уменьшению сложности системы и снижает вероятность ошибок из-за их неправильного использования.

Существуют различные способы организации и хранения информационных объектов: файлы, списки, строки, массивы, структуры, правильный выбор которых позволяет значительно упростить решение задачи, и, как следствие, размер кода. Поэтому использование множества переменных при проектировании неэффективно и не позволяет продемонстрировать полученные в ходе изучения дисциплины навыки работы со структурами данных [5].

Пример 3. *Задачу нахождения расстояния от точки до точки обычно рассматривают либо на координатной прямой, либо в прямоугольной декартовой системе координат на плоскости или в трехмерном пространстве.*

В данной постановке задачи требуется вычислять расстояние между точками по их координатам.

Формула вычисления расстояния между двумя точками на плоскости $M_1(x_1, y_1)$ *и* $M_2(x_2, y_2)$ *имеет вид:*

$$d = \sqrt{(x_1 - x_2)^2 + (y_2 - y_1)^2} \qquad (1)$$

Порядок точек не играет роли. Расстояние считается положительным, поэтому корень берется с одним знаком (плюс).

В ходе этого этапа должны быть описаны исходные данные и для них должны быть обосновано: выбраны типы или структуры данных, исходя из обсуждения методов решения поставленной задачи.

Следовательно, исходными данными являются целые числа со знаком, при этом результат вычисления по формуле 1 может иметь дробную часть.

Поэтому для хранения координат каждой точки выбраны две переменные типа int и для результата – double.

Структура для хранения данных о координатах и расстояния между точками содержит:

int x;//координата x.

int y;//координата y.

*double *distance;//указатель для создания динамического массива предназначенного для хранения расстояний от точек.*

Поскольку для каждой точки требуется хранить расстояния до всех остальных точек, то удобна структура типа одномерный массив, но поскольку размер этого массива до момента считывания неизвестен, то он должен быть динамический. В качестве переменной для хранения расстояний использован указатель на double.

Поскольку это переменные различных типов, то для их совместного хранения и использования удобна запись типа структура данных.

Любые ограничения и допущения должны быть также подробно описаны (пример 4).

Пример 4. *Перед вычислением значения функции необходимо проверить область допустимых значений аргументов. В выбранном варианте решения аргумент равный нулю приводит к делению на ноль, поэтому при вводе такого значения вызова функции не осуществляется, а пользователю выводится диагностическое сообщение «недопустимое значение операнда».*

В этом разделе следует описать, каким образом будут водиться исходные данные, выводиться окончательные и промежуточные результаты.

Чем лучше проработана модель разрабатываемой программы, чем четче определены подзадачи и структуры данных, хранящие входную, промежуточную и выходную информацию, тем проще проектирование и реализация и меньше вероятность ошибок, для исправления которых потребуется существенно изменять программу.

Пример 5. *Исходная информация представлена в файле input.txt, в котором на первой строке указано количество элементов, каждая следующая строка содержит три целых чисел разделенных пробелами.*

Для проверки правильности работы программы после завершения отладки следует провести ее тестирование. Тест – это такой вариант решения задачи, для которого заранее известны результаты. Как правило, один тестовый

вариант не доказывает правильность программы. Необходимо построить множество тестов для исчерпывающего испытания всей программы.

Пример 6. *Тесты программы «Простой калькулятор» должны продемонстрировать, что при правильном вводе исходной строки всегда получаются верные результаты, а при наличии ошибок (синтаксических, семантических, выхода за диапазон допустимых значений) будут получены соответствующие сообщения. План тестирования представлен ниже.*

№	*Входные данные*	*Результат*
1	*25+75*	*100*
2	*500-74*	*426*
5	*5^7*	*859375*
6	*25.17*1.0*	*25.17*
7	*5=6-1*	*Неверное выражение*

Успешное прохождение всех тестов есть необходимое условие подтверждения правильности работы программы.

2.2. Пошаговая детализация решения

Каждая выделенная на первом этапе задача должна быть разделена на небольшое количество более простых подзадач. Это постепенное иерархическое разложение (пошаговая детализация) продолжается до тех пока не будет выделена последовательность простых действий, соответствующих операторам языка программирования.

Важно, чтобы код программы был структурирован, т. е. разделен на функции и программные модули. При проектировании необходимо определить, из каких частей, модулей, функций будет состоять программа, и как программные элементы будут взаимодействовать между собой. На рис. 3 представлен пример структуры программы на языке Си, реализующих модульный подход к построению программ. Модульная декомпозиция представляет собой разбиение программы на несколько отдельных файлов, каждый из которых решает отдельную конкретную задачу и, как правило, облегчает процесс ее работы.

Рекомендуется выполнять пояснение назначения и выполняемых каждым модулем действий, как например, в табл. 1.Описание каждого программного объекта (модуль или функция) должно включать:

– название модуля/функции;

– краткое описание его назначения;

– описание входных и выходных параметров с указанием типов, допустимых значений;

– список используемых (вызываемых) функций;

– алгоритм функционирования.

Поскольку программа предназначена для работы некоторого пользователя, то ключевым элементом проектирования (после разработки

математической модели) является реализация взаимодействия пользователя с программой.

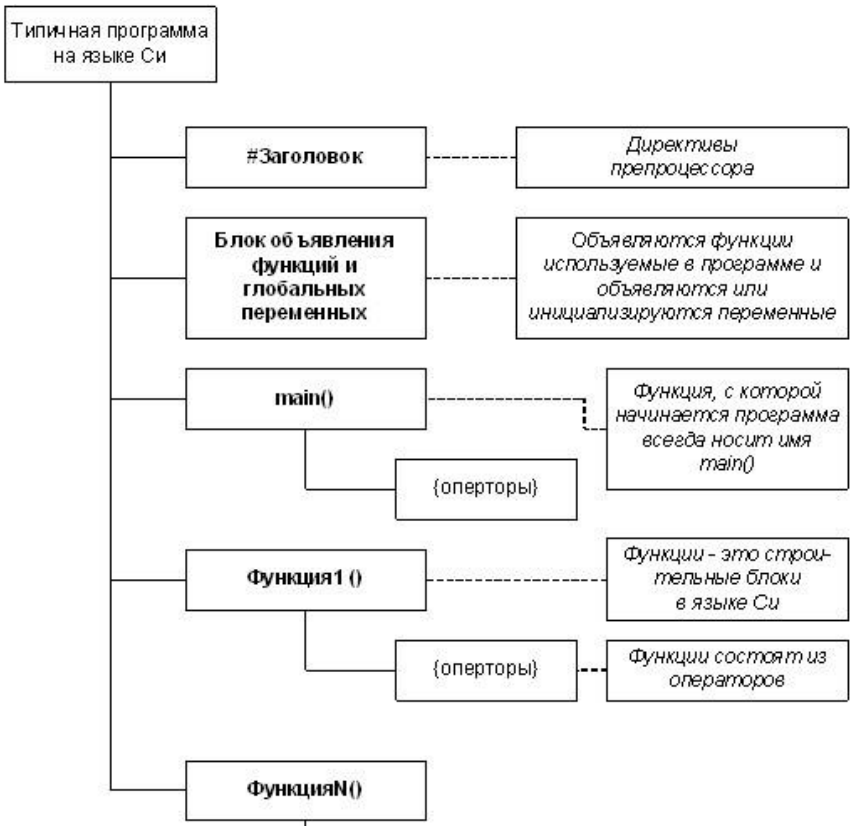


Рис. 3. Модульная декомпозиция программы на языке Си

При работе с программой пользователь должен знать:

- что он хочет получить на выходе (решение задачи);
- как минимум одну последовательность действий, приводящую к успешному результату;
- где ему найти все объекты, участвующие в процедуре решения;
- как определять пригодность объектов для их использования;
- как управляться с объектами.

Чтобы помочь пользователю, необходимо предусмотреть в конструкции диалога по обмену данными между программой и пользователем возможность выбора режима работы, ввода исходных данных и вывода результатов, а также включить подсказки и диагностические сообщения. Пример схемы диалога с пользователем на рис. 4.

В пояснительной записке к программе следует отметить, что в соответствии с проведенным выше анализом задачи разрабатываемая программа должна содержать: (указать какие) исходные данные для эксперимента, определяющие условия решения; выбор вида решения (расписать) и вида результатов (охарактеризовать в соответствии с ранее определенными функциями). В результате может быть сформирована структурная схема функционирования интерфейса подобная тому, как это представлено на рис.4.

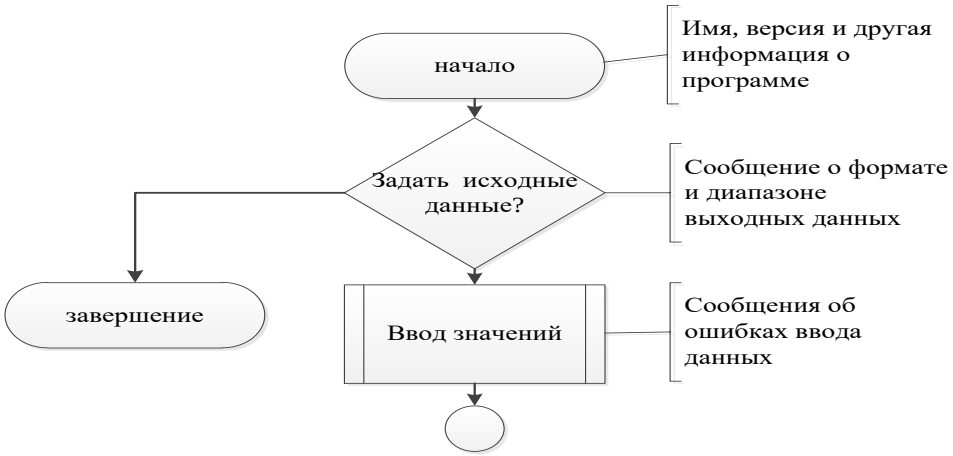


Рис.4. Структурная схема интерфейсной части программы

Таблица 1

Описание модулей программы

Наименование	Назначение	Примечание
main	Главная функция	Выводит информацию о программе и вызывает другие функции
readfile	Чтение данных из файла	Читает заданный пользователем тестовый файл
select_file	Диалог с пользователем для задания имени файла данных	Предлагает пользователю ввести имя файла
add_file	Проверка существования заданного файла в текущем каталоге	Проверяет существует ли файл с указанным именем, если такого файла нет, то заменяет введенное значение на input.txt

Собственные функции следует описывать следующим образом.

```
Функцияfloatsqr(floatx) – возвращает вещественное значение корня квадратного из переданного ей аргумента x, вещественного типа.
```

или

```
Функция sqr предназначена для вычисления квадратного корня.
Прототип функции имеет вид:
floatsqr(floatx)
Функция возвращает вещественного значениетипа float, если значение аргумента отрицательно и не возникает ошибка из-за выхода за пределы области допустимых значений (в этом случе функция возвращает NULL).
```

Каждый алгоритм должен быть представлен:

1) таблицей и (или) списком используемых в нем глобальных переменных;

2) блок-схемой алгоритма, использующей имена переменных, приведенных в таблице или списке; (алгоритм может быть представлен в псевдокодах, если разработчику более удобно);

3) описанием процесса обработки данных в соответствии с приведенной

схемой алгоритма.

Описание каждого алгоритма должно включать:

- функциональное назначение алгоритма;
- входные и выходные данные (результаты выполнения);
- список формальных параметров и их назначение;
- пример вызова модуля или подпрограммы;
- используемые технические средства;
- ссылку на таблицу переменных алгоритма;
- ссылку на рисунок со схемой алгоритма;
- описание процесса обработки данных в соответствии со схемой;
- если имеется приложение с полным текстом программы, то ссылку на соответствующую страницу приложения.

При описании процесса обработки данных в соответствии со схемой алгоритма необходимо пояснить все циклы, каждую альтернативу ветвления, принятое решение по результатам анализа альтернатив и последующие действия.

Тексты описания алгоритмов должны быть структурными, предложения короткими. Описание алгоритма должно отражать суть процесса обработки.

Пример описания функции алгоритмом представлен на рис. 5.

Пример 7. float mean_time(float *mas, intn)

Функция предназначена для определения среднего времени хода за одну игру. Данная функция возвращает среднее время хода. Блок-схема представлена на рис.5.

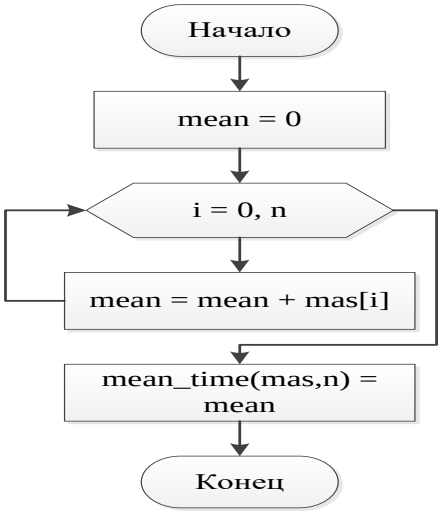
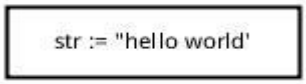
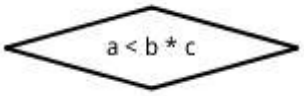
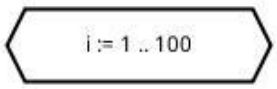
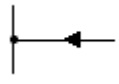


Рис.5. Блок-схема функции для определения среднего времени

При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий [1]. Такое графическое представление принято описывать в виде блок-схемы. Для изображения основных алгоритмических структур и блоков на блок-схемах используют специальные графические фигуры, соединенные стрелками, указывающими порядок их выполнения и показанные в табл. 2.

Таблица 2

Основные блочные символы

Обозначение	Функция
	Начало, конец алгоритма, вход и выход в подпрограмму
	Процесс. В блоке операций обычно размещают одно или несколько (ГОСТ не запрещает) операций присваивания, не требующих вызова внешних функций.
	Ввод-вывод.
	Решение (ветвление). В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет. Имеет один вход и несколько подписанных выходов. В случае, если блок имеет 2 выхода (соответствует оператору ветвления), на них подписывается результат сравнения — «да/нет». Если из блока выходит большее число линий (оператор выбора), внутри него записывается имя переменной, а на выходящих дугах — значения этой переменной.
	Цикл с параметром. Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.
	Соединитель. В случае, если блок-схема не умещается на лист, используется символ соединителя, отражающий переход потока управления между листами.
	Предопределенный процесс. Процедуры и функции (использование ранее созданных и отдельно описанных алгоритмов или программ). Используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.
	Горизонтальные и вертикальные потоки. Линии связей между блоками, направление потоков
	Комментарий может быть соединен как с одним блоком, так и группой. Группа блоков выделяется на схеме пунктирной линией.
	Слияние линий потоков

3. Требования к структуре и оформлению кода

3.1 Структура программы

Любая программа на языке Си может быть размещена в одном или нескольких файлах.

Файл, содержащий программный код имеет определенную структуру:

1) Директивы препроцессора;
2) Объявление прототипов пользовательских функций и внешних и/или глобальных переменных;
3) Главная функция main().
4) Определение пользовательских функций.

Работа препроцессора осуществляется с помощью специальных директив (указаний), начинающихся знаком решетка #.

Препроцессор — это специальная программа, являющаяся частью компилятора языка Си. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы и вводить макроопределения [2,4]. Настоятельно рекомендуется использовать #define для обеспечения простой возможности переопределения числовых значений.

Лучше макро определение, чем «магические цифры»
`#define PIXEL_WIDTH_OF_PLAY_AREA  800` `#define PIXEL_HEIGHT_OF_PLAY_AREA 600`

Если в программе более трех собственных функций имеет смысл разместить их в заголовочном файле.

Активно используете структурирование программы, не перегружайте функцию main, оставьте в ней диалоговую часть и логику управления ходом работы. Для всех подзадач и последовательностей операторов, связанных общей задачей, используйте функции.

Без необходимости не используйте внешние (глобальные) переменные, реализуйте обмен данными между функциями через механизм передачи параметров.

Не пренебрегайте возможностью идентифицировать программные объекты. Правильно выбранные имена могут сделать программу поистине самодокументированной, практически не требуя явных комментариев. Плохо выбранные имена всегда добавляют ненужную сложность в вашу программу.

При разработке программы учитывайте следующие правила выбора идентификаторов.

1. Не используйте однобуквенные имена. Типичной ошибкой начинающих является стремление давать всем переменным неосмысленные однобуквенные имена, например **m**, **n**, **a**, **s**, **p** и т.п. Это глубоко порочная практика, поскольку при этом теряется сам смысл понятия *имя*. Одно буквенные имена принято давать только индексам! Исключением являются

случаи, когда количество переменных в процедуре очень мало (порядка 1-3 переменных) и смысл их хорошо понятен из контекста или комментариев.

2. Придумывайте осмысленные имена переменным. Все переменные, имеющие сколько-нибудь важное значение в программе, необходимо снабжать именами, характеризующими их назначение.

filename	имя файла
int_vector	целочисленный вектор
n_book_market	число книг в магазине

При плохом знании английского языка можно использовать звуковые аналогии русским буквам, например **nazv_faila**, **razmer**, **summa** и т.п. Это не в полной мере соответствует представлению о «хорошем стиле», но, несомненно, гораздо нагляднее использования совсем бессмысленных имен.

В общем случае в именах не должны использоваться сокращения или аббревиатуры. Это приводит к ухудшению читаемости имен [10].

3. Давайте осмысленные имена функциям. Такие имена часто делают довольно длинными для улучшения понимаемости их смысла, а отдельные компоненты имен начинают с большой буквы, например **ObjectList**, **ArcSet** и т.п. Имена функций при этом рекомендуется начинать с глагола, например **GetPersonName**, **SetNewDate** и т.п. Альтернативным способом выделения компонент в сложных названиях является использование символа **_**, например **add_record**, **copy_object**. Главное - не смешивать в одной программе несколько стилей.

4. Придумывайте осмысленные имена файлам. Все сказанное вполне относится и к именам файлов с программами. Имена файлов должны нести обязательную смысловую нагрузку, поясняя свое «содержимое». Например, заголовочный файл, содержащий описание класса **vector,** логичнее всего назвать **vector.h**, а файл, содержащий реализацию методов этого класса - **vector.c**. Все современные операционные системы поддерживают длинные имена файлов, вследствие чего искусственно упрощать и укорачивать их нет необходимости. Более того, при разработке крупного продукта количество исходных файлов измеряется сотнями. В этой ситуации невозможно «помнить», в каком файле что находится, следовательно, длинные и понятные имена становятся жизненно необходимыми.

5. Имена макросов и констант должны записываться ЗАГЛАВНЫМИ БУКВАМИ. Особенно это важно для макросов, определяемых директивой **#define**, так как они часто вызывают побочные эффекты. Поэтому полезно иметь возможность определить с первого взгляда, что у вас является макросом или константой. Конечно, вы не должны использовать только заглавные буквы для чего-нибудь помимо макросов и констант, иначе вы не достигнете цели данного правила.

3.2 Рекомендации по оформлению кода

Для улучшения читаемости исходного текста программы следует располагать *не более одного оператора в строке*, что вызвано особенностями

человеческого восприятия текста. Кроме того, это облегчает пошаговую отладку в символьных отладчиках.

Неправильно	Правильно

`int *ptr;ptr = new int [100];ptr[0]=0;`	`int *ptr;`
	`ptr = new int [100];`
	`ptr[0] = 0;`

Два оператора в строке *вполне допустимы*, если второй подчинен первому, причем является единственным подчиненным (несоставным), например:

```
for( i=0; i<size; i++) m[i] = 0;
```

Использование двух и более операторов в строке не только допустимо, но и желательно, если это позволяет подчеркнуть некоторую систему в локальной последовательности операторов, например:

```
x1 = Tr1[0];  y1 = Tr1[1];  z1 = Tr1[2];
x2 = Tr2[0];  y2 = Tr2[1];  z2 = Tr2[2];
x3 = Tr3[0];  y3 = Tr3[1];  z3 = Tr3[2];
```

Правильное использование сдвигов, или, иначе, отступов (indentation), является ключевым методом обеспечения читаемости. Идея состоит в том, что отступы зрительно показывают подчиненность (иерархию) операторов. При этом директивы препроцессора (**#include**, **#define** и т.д.), описания классов, структур, типов, глобальных данных и определения функций всегда имеют наивысший приоритет, то есть начинаются с крайней левой позиции, например:

```
#include <stdio.h>
#define NAME_SIZE 256

void main()
{
 //...
}
```

Соблюдайте следующие правила использования отступов и пробелов.

1. Подчиненные операторы должны быть сдвинуты вправо по отношению к управляющей конструкции, образуя следующий уровень иерархии:

Неправильно	Правильно

`if( f == NULL )`
`cout << "No file\n";`
`else`
`cout << "Start..\n";`
	`if( f == NULL )`
	`cout << "No file\n";`
	`else`
	`cout << "Start..\n";`

2. Операторы одного и того же уровня иерархии (не подчиненные друг другу) должны иметь равный отступ[3,6]:

Неправильно	Правильно

`cout << "Enter dimension: ";`
`cin >>dim;`
`ptr = new int [dim];`
`ptr[0] = 0;`
	`cout << "Enter dimension: ";`
	`cin >>dim;`
	`ptr = new int [dim];`
	`ptr[0] = 0;`

3. Размер (шаг) сдвига должен быть постоянным:

Неправильно	Правильно

`if( ptr == NULL )`
`return -1;`
`for( i=0; i<dim; i++)`
`ptr[i] = i;`
	`if( ptr == NULL )`
	`return -1;`
	`for( i=0; i<dim; i++ )`
	`ptr[i] = i;`

Шаг сдвига должен быть постоянным во всех функциях и даже во всех файлах программы, а лучше всего, и во всех программах, которые вы пишете. Часто для сдвигов применяют табуляцию (по умолчанию 4 пробела), устанавливая при этом для нее желаемый шаг. Однако лучше использовать пробелы – это более универсальный способ и рекомендован в [9].Возможность установить количество пробелов для отступа в начале строки поддерживается большинством интегрированных сред разработчика. В среде разработки MVS выбрать меню ***Сервис->Параметры***, выбрать ***Текстовый редактор***, и настроить ***Табуляцию*** (в папке Все языки) и/или установить в разделе ***Отсупы*** правило перехода на следующую строку.

4. Используйте классический стиль для размещения операторных скобок. Классический стиль подразумевает, что открывающаяся скобка помещается строкой ниже под управляющей конструкцией, строго на уровне её левой границы, а закрывающаяся скобка – строго под открывающейся:

```
int factorial( int n )
{
if( n > 1 )
return n*factorial( n-1 );
if( n < 0 )
{
cerr <<"Factorial error: negative argument\n";
exit( 1 ); //Аварийный выход
}
return 1;
}
```

Такой прием обеспечивает улучшенную наглядность. Открывающаяся и закрывающаяся скобки при этом располагаются строго друг под другом, что помогает находить начало и конец составного оператора, функции или описания класса. Большинство редакторов кода обеспечивают проверку скобки на парность (В MVS– Автоматическое выделение разделителей на странице <***Общие***>, папка <***Текстовый редактор***>, диалоговое окно <***Параметры***>), но отсутствие зрительного разделения ведет к ухудшению читаемости кода.

5. Добавляйте пробелы к другим элементам синтаксиса. Отдельные элементы текста необходимо отделять пробелами, несмотря на то, что первые, возможно, уже отделены другими знаками препинания. Добавляйте пробелы: после запятой и точки с запятой; после открывающейся круглой скобки и перед закрывающейся круглой скобкой; вокруг знаков арифметических и логических операций.

Неправильно	Правильно
`while(i++<dim)` `move(a,b,ptr[base+off*i]);`	`while( i++ < dim )` `move( a, b, ptr[base + off*i] );`

6. Используйте пустые строки для выделения:

– объявления переменных от их использования;

```
char str[80];//объявление
int counter = 0;

fgets( str, 79, infile);//использование
counter++;
```

– последовательности однотипных инструкций;

```
#include<math.h>// заголовочные файлы
#include <stdio.h>
#include <stdlib.h>

#define NAME_SIZE 256//опредление чисел
#define MAX_LEN 3000
```

– любые логически завершенные блоки кода (каждая законченная мысль - нуждается в выделении);

```
cout <<"Enter size and delta:";//Блоквводаданных
cin >>size >>delta;

for( i=0; i<size; i++ ) //Блокиспользованияданных
{
a[i] -= delta;
b[i] += delta;
}
```

7. Отделяйте определения функций строками с комментариями, желательно содержащими их описание

```
/*------------главная функция, ---------------------------------------
-------------выполняет диалог с пользователем----------------------*/
intmain()
{
. . .
}

//------------получить имя файла
char *get_name(FILE *f)
{
. . .
}
```

8. Разбивайте слишком длинные строки. Если строка трудна для чтения из-за своей длины или тем более не помещается в окне, ее следует разбить на несколько частей.

Плохо и не понятно

```
char errMsg0[] = "Fatal Error\nError code 0:\tNot enough memory\n";
```

Наглядно

```
char errMsg0[] = "Fatal Error\n"
                 "Error code 0:\t"
                 "Not enough memory\n";
```

Плохо и не понятно

```
if( (2*a*c <= vec.getsize()) || (matr[i][j] > max_lengh/num_obj) ||
something_else() || one_more_condition() ) return true;
```

Наглядно

```
if( ( 2*a*c <= vec.getsize() ) ||
    ( matr[i][j] > max_lengh/num_obj ) ||
( something_else()               ) ||
( one_more_condition()           )
)
  return true;
```

Плохо и не понятно

```
int SetCoord( double x1, double y1, double x2, double y2, double x3, double y3);
```

Наглядно

```
int SetCoord( double x1, double y1,
      doublex2, doubley2,
      doublex3, doubley3 );
```

Время, потраченное на написание комментариев, многократно окупится при любых модификациях программы. Однако и здесь есть ряд тонкостей.

Комментарии должны быть предложениями. Они должны быть хорошо составлены и иметь правильную пунктуацию, по возможности без сокращений. Не превращайте свои комментарии в секретный код, применяя странные сокращения и изобретая свой собственный грамматический строй.

Комментарий не должен подтверждать очевидное [7,9]. Не стоит комментировать все подряд, включая очевидные действия.

Плохо и бессмысленно

```
size = 10; //Присвоить size значение 10
for( i=0; i<size; i++) //Циклпо i от 0 до size
{
}
```

Комментируйте закрытие вложенных операторов. Если существует большая глубина вложенности операторов наряду со значительным расстоянием операторных скобок друг от друга, то очень трудно определять соответствие операторов и закрывающих скобок.

Например, на первой странице:

```
while( a < b )
{
while( something_else() )
{
for( i = 10; --i >= 0; )
{
for( j = 10; --j >= 0; )
{ // далее идет масса кода
```

На какой-то из последующих страниц:

```
}
}
}
}
```

Если вложенность небольшая, можно использовать комментарии вида

```
} //for
} //for
} //while
} //while
```

В более сложных случаях эти комментарии слишком кратки, чтобы быть полезными, и завершающие блок комментарии должны полностью описывать (повторять) управляющий оператор:

```
} //for( j = 10; --j >= 0; )
    } //for( i = 10; --i >= 0; )
  } //while( something_else() )
} //while( a < b )
```

Комментируйте заголовок функции. Функции рекомендуется отделять друг от друга штриховыми линиями, расположенными перед заголовком. Комментарии должны пояснять назначение аргументов и смысл самой функции, а при необходимости – алгоритм[8,10]:

```
//-------------------------------------------------------
void descriptive_name( type descriptive_name )
{ /* Если имена функции и аргументов недостаточно содержательны,
  *поместить здесь комментарий, описывающий, что она делает.
*  1. Описать возвращаемое значение (значения) и аргументы,
  *если это не очевидно.
  * 2. Прокомментировать основные особенности алгоритма.
  */
  //Далее идет код...
}
```

Комментируйте любые места, которые трудны для быстрого понимания. Особенно следует отмечать в комментариях использование программных «трюков», нестандартных приемов и находок.

4. Требования к структуре и оформлению пояснительной записки

Общий объём пояснительной записки в пределах 20-30 листов машинописного текста, не считая приложений. Курсовой проект должен быть оформлен в соответствии с требованиями ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».

Размер бумаги – А4.

Поля: левое – 30 мм, правое – 15 мм, верхнее – 20 мм, нижнее – 20 мм.

Настройки основного стиля:

- ***Шрифт – 14, типа Times New Roman***
- ***Межстрочный интервал – полуторный***
- ***Отступ красной строки – 1,25 см***
- ***Отступы до и после абзаца – 0***
- ***Выравнивание – по ширине***
- ***Размеры полей: левое – 3 см, верхнее – 2 см, нижнее – 2 см, правое – 1,5 см***

Страницы нумеруются внизу в центре, начиная с титульного листа, на котором номер страницы не ставиться.

Текст должен быть разделен на разделы и, при необходимости, подразделы (заголовки 1-го и 2-го уровней). Заголовки должны быть сформулированы кратко, на первом месте должно стоять имя существительное.

Все заголовки иерархически нумеруются. Номер помещается перед названием, после каждой группы цифр ставится точка. В конце заголовка точка не ставится. Такие разделы как «СОДЕРЖАНИЕ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ», «ПРИЛОЖЕНИЕ» обычно не нумеруются.

Заголовки одного уровня оформляются одинаково по всему тексту.

Каждый раздел (заголовок 1-го уровня) следует начинать с новой страницы. Заголовок 1-го уровня следует располагать в середине строки и набирать прописными буквами. Заголовки 2-го уровня и ниже следует начинать с абзацного отступа и печатать с прописной буквы. Переносы в заголовках не допускаются. Заголовки следует отделять от окружающего текста 1 строкой. Подчеркивать заголовки запрещается. После любого заголовка должен следовать текст, а не рисунок, формула, таблица или новая страница.

Цифровой материал большого объема выполняется в виде таблиц, которые располагают непосредственно в тексте после первого упоминания. На все таблицы должны быть ссылки в тексте. Номер и название таблицы располагают над ней слева без абзацного отступа. *Пример:*

Таблица 1. – Основные характеристики системы

Рисунки входят в общий объём и нумеруются в пределах всей пояснительной записки (Рисунок 1 и т.д.). Все иллюстрации должны иметь ссылки в тексте. Номер и название помещают ниже рисунка и подрисуночного текста с поясняющими данными.

Пример:

Рисунок 1. – Структурная схема системы

Сведения об использованных источниках следует располагать в порядке появления ссылок в тексте работы и нумеровать. Сведения об учебниках, методических пособиях и монографиях должны включать автора, название, издательство и год издания. Интернет источники должны включать название материала, автор, полная ссылка на сайт, включающая html-страницу.

Пример:

1.Иоффе, Д.С. СтандартIEEE 1284. http://www.rusdoc.ru /material/hardware/iee/ieee_1284.htm.

Приложения идентифицируются номерами или буквами, например «ПРИЛОЖЕНИЕ 1» или «ПРИЛОЖЕНИЕ А». На следующей строке, при необходимости, помещается название приложения, например «ЛИСТИНГ ПРОГРАММЫ», которое оформляется как заголовок 1-го уровня без нумерации. В раздел «СОДЕРЖАНИЕ» названия приложений, как правило, не помещают.

ЗАКЛЮЧЕНИЕ

Навыки разработки программ относятся к необходимым базовым техникам будущего ИТ-специалиста и начиная с первого проекта необходимо формировать не только умение кодировать, но последовательно проходить основные этапы разработки программного продукта – анализ требований, проектирование, реализация, тестирование. На стадии проектирования приобретаются навыки структурирования программы и построения алгоритма. Процесс реализации состоит в составлении текста программы и его отладки на заранее продуманном наборе тестов. Тестирование программы – это очень

важная и одновременно сложная задача, позволяющая не только оценить достигнутый результат, но сформировать навыки анализа задач.

В методических указания даны теоретические сведения и описаны практические действия, необходимые для разработки программного продукта с заданным функциональным назначением и включающие весь процесс создания программы от описания понимания поставленной задачи и ее анализа, многократных ее реализациях на языке программирования до появления отлаженного законченного программного продукта.

Особое внимание в курсе «Основы программирования и алгоритмизации» отводится на формирование хорошего собственного стиля программирования – простого и легко читаемого кода. Следование правилам хорошего стиля программирования значительно уменьшает вероятность появления ошибок на этапе набора текста, делает программу легко читаемой, что, в свою очередь, облегчает процессы отладки и внесения изменений. Это достигается не только использованием особых правил записи текста и понятных имен, но прежде всего надежностью и дружественностью программы (контроль исходных данных и их доступности, наличие справочной системы, разумное и предсказуемое, с точки зрения пользователя, поведение программы) и курсовой проект прекрасная возможность это продемонстрировать.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ГОСТ 19.701-90 – Схемы алгоритмов программ, данных и систем
2. Ашарина, И.В. Основы программирования на языках С и С++[Текст]: учебн. пособие для вузов / И.В. Ашарина.– Москва: Горячая линия – Телеком, 2002. – 207 с.
3. Давыдов, В.Г. Программирование и основы алгоритмизации [Текст]: учебн. пособие для вузов /В.Г. Давыдов. – Москва: Высшая школа, 2003. – 447 с.
4. Павловская, Т.А. С/С++. Программирование на языке высокого уровня[Текст]: учебн. пособие для вузов / Т.А. Павловская. – СПб: Питер, 2013. – 461 с.: ил.
5. Подбельский, В.В., Фомин, С.С. Курс программирования на языке Си[Текст]: учебн. пособие для вузов / В.В. Подбельский, С.С. Фомин. – М.: ДМК Пресс, 2007. – 239 с.
6. Семакин, И. Г., Шестаков, А. П.Основы программирования[Текст]: учебник для вузов / И. Г. Семакин,А. П. Шестаков. – М.: Академия, 2009. - 432с.
7. Курипта, О.В., Минакова, О.В.Основы программирования и алгоритмизации: практикум / О.В. Курипта, О.В. Минакова, Д.К. Проскурин; Воронежский ГАСУ. – Воронеж, 2015. – 132 с.
8. Шилдт, Г. С: Полное руководство, классическое издание [Текст]: учебник для вузов / Г.С. Шилдт. – Москва: Издательский дом «Вильямс», 2002. – 704 с.
9. Соглашение по оформлению кода на Си.

10. Документация по MDSNhttps://msdn.microsoft.com/ru-ru/library/ms229045(v=vs.100).aspx

ПРИЛОЖЕНИЕ 1

ЗАДАНИЯ ПО КУРСОВОМУ ПРОЕКТУ

Задание А.

Спроектировать три элементарных программных модуля для вычисления трех функций:

f1(x), включающую последовательность различных математических операций и функций;

кусочную f2(x), использующую многоальтернативный выбор решения;

f3(x), представляющую сумму ряда.

Вариант	Функции
1	$f1(x) = arctg\left(\frac{\cos^2(x)}{\sqrt[3]{1.7 \cdot x^4}} \cdot \ln(5.2)\right)$, $f2(x) = \begin{cases} \lvert x\rvert & при & \lvert x\rvert > 1 \\ 1+x & при & \lvert x\rvert \le -1 \\ \sin(x) & при & остальных \end{cases}$, $f3(x) = \frac{1}{2} \cdot \left(\frac{1}{x+1}\right)^2 + \frac{1}{4} \cdot \left(\frac{1}{x+1}\right)^4 + \ldots$
2	$f1(x) = \frac{\sqrt{5.9} \cdot tg^3(ar\sin(x))}{x^{1.7}}$, $f2(x) = \begin{cases} \sin(2.3x-1) & при & x > 2.5 \\ 1-3\ln\lvert 1-x\rvert & при & 0 \le x \le 2.5 \\ 2-x & при & x < 0 \end{cases}$, $f3(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots$
3	$f1(x) = x^2 - \pi \cdot x \cdot \cos(\pi \cdot x)$, $f2(x) = \begin{cases} x+1 & при & x > 0.75 \\ 1-x^5 & при & 0 \le x \le 0.75 \\ x + \ln\lvert\cos(x)\rvert & при & x < 0 \end{cases}$, $f3(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$
4	$f1(x) = \frac{\cos^2(e^{1.4})}{\ln^3(x)} \cdot \left(2.3 - \left\lvert\frac{x}{173.5\ln(x)}\right\rvert\right)$, $f2(x) = \begin{cases} x-2 & при & x \ge 2.5 \\ 1+x^2 & при & -1 < x < 2.5 \\ x\ln\lvert\cos x\rvert & при & x \le -1 \end{cases}$, $f3(x) = \sin x + \frac{\sin 3x}{3} + \frac{\sin 5x}{5} + \ldots$
5	$f1(x) = \frac{\sin(x^2 + x^{-1} + x^{1/3})}{tg(e^{\cos(\sqrt{x})})} \cdot 10^{-6}$, $f2(x) = \begin{cases} e^{-(x+0.8)} & при & x > 3.61 \\ 1 & при & 0 \le x \le 3.61 \\ 0.5x & при & x < 0 \end{cases}$, $f3(x) = \cos x + \frac{\cos 2x}{2} + \frac{\cos 3x}{3} + \ldots$

Вариант	Функции
6	$f1(x)=\ln x^2-x+4,$ $f2(x)=\begin{cases} x-2 & при \quad x\ge 2.5 \\ 1+x^2 & при \quad -1<x<2.5, \\ x\ln\lvert\cos x\rvert & при \quad x\le -1 \end{cases}$ $f3(x)=1+\frac{x^2}{2!}+\frac{x^4}{4!}+...$
7	$f1(x)=\frac{0.8\cdot tg^2(x)+1}{x^{-1/8}+\pi/4},$ $f2(x)=\begin{cases} \lvert\cos(x)\rvert^x & если \quad x>\pi \\ x^{\cos x} & при \quad x\ge\pi \end{cases},$ $f3(x)=x-\frac{x^3}{3}+\frac{x^5}{5}-.\frac{x^7}{7}..$
8	$f1(x)=\frac{ar\cos(0.5\cdot\lvert\sin(x-3)\rvert)+3x}{x^{-1/3}},$ $f2(x)=\begin{cases} \frac{x^2-7x+3}{x-e^x} & если \quad x\le 2 \\ arctg^2(x) & если \quad x>2 \end{cases},$ $f3(x)=1-\frac{3x^2}{2!}+\frac{5x^4}{4!}-.\frac{10x^6}{6!}..$
9	$f1(x)=\lg\left(\sum_{k=1}^{4}\frac{x^k}{(2k)!}\right),$ $f2(x)=\begin{cases} 3x^2 & при \quad x>4.5 \\ e^{-x} & при \quad 1\le x\le 4.5, \\ -\cos^2(2x) & при \quad x<1 \end{cases}$ $f3(x)=\sin x+\sin 3x^3+.\sin 5x^5..$
10	$f1(x)=\frac{\sqrt{4{,}28\cdot x^2}\cdot\cos^2(e^{3/4x})}{\ln^3(x)},$ $f2(x)=\begin{cases} \ln\lvert\sqrt[3]{x}\rvert & если \quad x>0 \\ \sum_{k=1}^{9}k^{x^2} & при \quad x>0 \end{cases}$ $f3(x)=1+\frac{2x}{1!}+.\frac{(2x)^2}{2!}..$

Интерфейс программы должен обеспечивать следующие возможности:

А) табулирование функций в заданном диапазоне с пользовательским значением шага вычислений;

Б) задание точности расчета (для вычисления ряда число членов и/или погрешность);

В) вывода результатов вычислений на консоль и/или записи в файл.

Задание Б.

Для заданной базы данных (см. варианты предметной области)

1) предложить структуру для описания отдельных полей (не менее пяти полей двух и более различных типов данных);

2) реализовать возможность заполнения одномерного массива структур значениями из стандартного потока (клавиатуры);

3) организовать запись и чтение данных (отдельных записей базы данных) из файла;

4) разработать функцию поиска записи по заданным пользователем значениям полей, указанных в варианте задания;

5) разработать функцию упорядочивания записей по любому из полей базы данных, указанных в варианте задания;

6*) обеспечить возможность добавления новых записей в файл базы данных.

Вариант	Тематика базы данных	Признак поиска	Возможные поля для упорядочивания
1	Пациент	Фамилия, возраст	Имя, фамилия, год рождения, район проживания, группа крови
2	Спортивная команда	Название, количество очков	Название, регион, кол-во сыгранных игр, набранных очков по категориям (выиграли/ничья/проиграли)
3	Автомобиль	Марка, серийный номер	Марка, серийный номер, кол-во дверей, год выпуска, цвет
4	Альбом музыкальный	Исполнитель, время	Исполнитель, название, кол-во песен, год выпуска, время звучания
5	Учебник	Автор, шифр направления	Название, автор, год издания, кол-во страниц, шифр направления
6	Институт	Название, код специальности	Название, количество специальностей, студентов, адрес.
7	Ученик	Кол-во учеников в классе, возраст	Фамилия, класс, пол, дата рождения
8	Поезд	Время в пути, пункт назначения	Название (код), пункт отправления/назначения, расстояние, время в пути

Интерфейс программы должен обеспечивать следующие возможности:

1) ввод данных по одному из сценариев:

А) заданное пользователем количества записей;

Б) до появления структуры с заданным признаком;

В) диалог с пользователем о необходимости продолжить ввод.

2) вывод результатов упорядочивания в заданный файл и/или на экран.

3) вывод диагностических сообщений в ходе проверки корректности ввода данных.

4) пользовательский выбор полей записи для упорядочивания массива структур;

5) задание пользователем значения поисковых полей.

ПРИЛОЖЕНИЕ 2

Пример выполнения курсового проекта

В данном методическом указании рассмотрен пример разработки курсового проекта по теме «Разработка игры «Быки и коровы»

ЗАДАНИЕ

Разработать компьютерную игру «Быки и коровы».

Программа должна поддерживать диалог с пользователем.

Необходимо предусмотреть возможность выбора различных сценариев игры, например ограничение числа ходов или времени хода.

Обязательная дополнительная функция – сбор статистики о времени каждого хода.

Требуется реализовать запись в файл результатов игры.

ВВЕДЕНИЕ

Целью настоящей работы является закрепление знаний, получаемых в процессе изучения дисциплины, приобретение необходимых практических навыков для применения работы со статическими и динамическими массивами, циклами, файлами и функциями. Задача курсовой работы состоит в разработке программного продукта – логической игры «Быки и коровы». В ходе игры будет производиться проверка ходов пользователя, пока не будет сделан верный. И тогда игра закончится, а результаты будут записаны в файл. В качестве результатов будут использоваться: количество ходов за 1 игру, среднее время хода и комментарий к игре.

Для удобной работы с программным продуктом необходимо разработать удобный пользовательский интерфейс, а так же пункт – «Правила» для ознакомления с игрой.

Для решения поставленной задачи необходимо выполнить следующие действия:

- проанализировать поставленную задачу;
- выбрать метод решения поставленной задачи;
- обосновать выбор типов и структур данных;
- составить алгоритм решения задачи;
- реализовать диалог с пользователем;
- провести отладку программы и написать сопроводительную документацию.

Программный продукт будет реализован на языке программирования C++ в среде Microsoft Visual Studio 2015.

При разработке программы используется структурный подход.

1. Анализ поставленной задачи

Быки и коровы — логическая игра, первоначально задуманная для двух игроков, но с появлением компьютерных версий стал превалировать вариант, когда один игрок отгадывает число, задуманное программой, то есть играет в

одиночку. Варианты игры могут зависеть от типа отгадываемой последовательности — это могут быть числа, цвета, пиктограммы или слова.

Данный программный продукт реализуется для одного человека, противником которого является компьютер. В реализуемой игре компьютер загадывает четырехзначное число, в котором цифры не повторяются. Задача пользователя угадать данное число.

Для проверки правильности числа необходимо применить реализовать следующие этапы:

- проверку на количества цифр в числе,
- проверку на повторяющиеся цифры в числе,
- проверку угаданных цифр стоящих на своей позиции (число быков) и стоящих не на своей позиции (число коров).

По результатам каждого хода необходимо предоставлять информацию о том, насколько правильно сделан ход.

Игра может закончиться, даже если пользователь не дал правильного ответа, так как время хода ограничено и составляет 30 с.

Для правильного выполнения поставленной задачи программа должна выполнять следующие операции:

- по желанию пользователя вывести на экран правила игры;
- осуществить ввод числа;
- определить время хода;
- проверить правильность хода;
- выводить комментарий к ходу;
- по окончанию игры записать результаты игры (количество ходов, среднее время хода и комментарий) в файл.

Разрабатываемая программа должна «загадать» четырехзначное число, т.е. произвести ввод элементов массива случайным образом. Затем пользователь должен вводить предполагаемое число, которое будет помещать в массив размерностью в четыре элемента. После чего будет происходить сравнения двух массивов на совпадение. И когда они будут одинаковые, игра закончится, а результат игры записывается в файл.

Работа программы состоит из четырех этапов:

1) заполнение элементов первого массива случайным образом;
2) ввод пользователем числа и помещение его во второй массив;
3) реализация алгоритма сравнения двух массивов на совпадение;
4) запись результата в файл.

Для работы программы требуются переменные, представленные в в таблице.

Таблица 1 – Переменные программы и их назначение

Переменная	Тип и значение
intgame;	переменная целочисленного типа предназначена для определения номера игры
intbulls;	переменная целочисленного типа предназначена для определения

Переменная	Тип и значение
	количества быков (отгаданная цифра предполагаемого числа, стоящих на своих местах).
intcows;	переменная целочисленного типа предназначена для определения количества коров (отгаданная цифр предполагаемого числа, не стоящих на своих местах)
intran[4];	статический массив размерностью в четыре элемента предназначен для задания элементов случайным образом
*float *time_h*	динамический массив размерностью в двадцать элементов предназначен для записи времени каждого хода последовательно
intkol_h	переменная целочисленного типа, в которую записывается количество хода, сделанных за одну игру
intN;	переменная целочисленного типа предназначена для ввода пользователем предполагаемого числа
intnum[4]	статический массив целочисленного типа размерность в четыре элемента предназначен для помещения в него предполагаемого пользователем числа, чтобы было удобно производить сравнения «загаданного» числа компьютером и предполагаемого числа пользователем

Особенностью выбранного подхода к решению является использование массива, в качестве общей структуры данных для хранения ключевой информации – «загаданной» числовой последовательности. Также для реализации дополнительного требования – учет времени хода для сбора статистики, необходимо использование динамического массива , поскольку число ходов неизвестно заранее.

Массив– это конечная последовательность однотипных данных. Каждый член этой последовательности называется элементом массива. Доступ к элементам массива производится по их номеру в последовательности, который в этом случае называется индексом. Количество элементов в массиве называется размерностью массива. Причем важно заметить, что если размерность массива равна n, то индекс этого массива изменяется от 0 до n-1.

Доступ к элементам массива выполняется при помощи оператора индексирования [ ], результатом выполнения которого является значение элемента массива с заданным индексом.

Динамическое выделение памяти необходимо для эффективного использования памяти компьютера. В C++ операции **new** и **delete** предназначены для динамического распределения памяти компьютера. Операция **new** выделяет память из области свободной памяти, а операция **delete** высвобождает выделенную память. Выделяемая память после её использования должна высвобождаться, поэтому операции **new** и **delete** используются парами. Даже если не высвобождать память явно, то она освободится ресурсами ОС по завершению работы программы

Операция **new** создает объект заданного типа, выделяет ему память и возвращает указатель правильного типа на данный участок памяти. Если память невозможно выделить, например, в случае отсутствия свободных участков, то возвращается нулевой указатель, то есть указатель вернет значение 0.

Выделение памяти возможно под любой тип данных.

2. Детализация выбранного решения

Программа считывает вводимые пользователем символы посредствам объекта **cin**, далее вводимый символ пропускается через цикл, если введена цифра «1» программа выводит «Правила игры», если введена цифра «0» программа предлагает начать игру.

Программа снова считывает вводимые пользователем символы посредством объекта **cin** , а вводимый символ так же пропускается через цикл. Если пользователь вводит «1», то игра начинается.

Каждая игра заключена в тело циклацикла, в котором сначала считывается вводимое пользователем предполагаемое число, а затем проверяется его правильность. В зависимости от ответа программа выводит следующее:

1) если в числе больше или меньше чем четыре цифры, на экран выводится посредством объекта **cout** «Ошибка! Число должно быть 4х значным» и предлагается ввести число еще раз;

2) если в числе есть повторяющиеся цифры, на экран выводится посредством объекта **cout** сообщение «Ошибка! Цифры в числе не должны повторяться» и предлагается ввести число снова;

3) если ход сделан более чем за 30 с, выводиться сообщение «Время хода вышло. К сожалению, вы проиграли!», и игра заканчивается без победы;

4) если угадано два быка и две коровы или только четыре коровы, выводиться посредством объекта **cout** сообщение «Очень сильный ход!», и игра продолжается;

5) если угадано два быка и одна корова или один бык и две коровы, выводиться посредством объекта **cout** сообщение «Хороший ход!», и игра продолжается;

6) если же угадано четыре быка, то выводиться посредством объекта **cout** сообщение «Мур... Победа!», и игра заканчивается с победой.

Далее результаты удавшейся игры записываются в файл. Данные записывающие в файл:

- номер игры;
- комментарий к игре;
- среднее время хода.

В конце программа предлагает сыграть еще раз. Она снова считывает вводимые пользователем символы посредствам объекта **cin** , а вводимый символ так же пропускается через цикл. Если пользователь вводит «1», то игра начинается заново.

На рис.1. представлена блок схема разрабатываемого программного продукта. Она содержит следующие пользовательские функции:

1) void random_array(int *mas);

Функция предназначена для задания массива случайным образом, помещаемого в массив, который указывается в качестве параметра. Данная функция ничего не возвращает. Блок-схема представлена на рис.2.

2) int input_num();

Функция предназначена для проверки введенного числа на количество цифр в нем и повторений этих цифр. Данная функция возвращает введенное число пользователем. Блок-схема представлена на рис.3.

3) void num_array(int n, int *mas);

Функция преобразует четырехзначное число в массив, указанный в качестве параметра. Данная функция ничего не возвращает. Блок-схема представлена на рис.4.

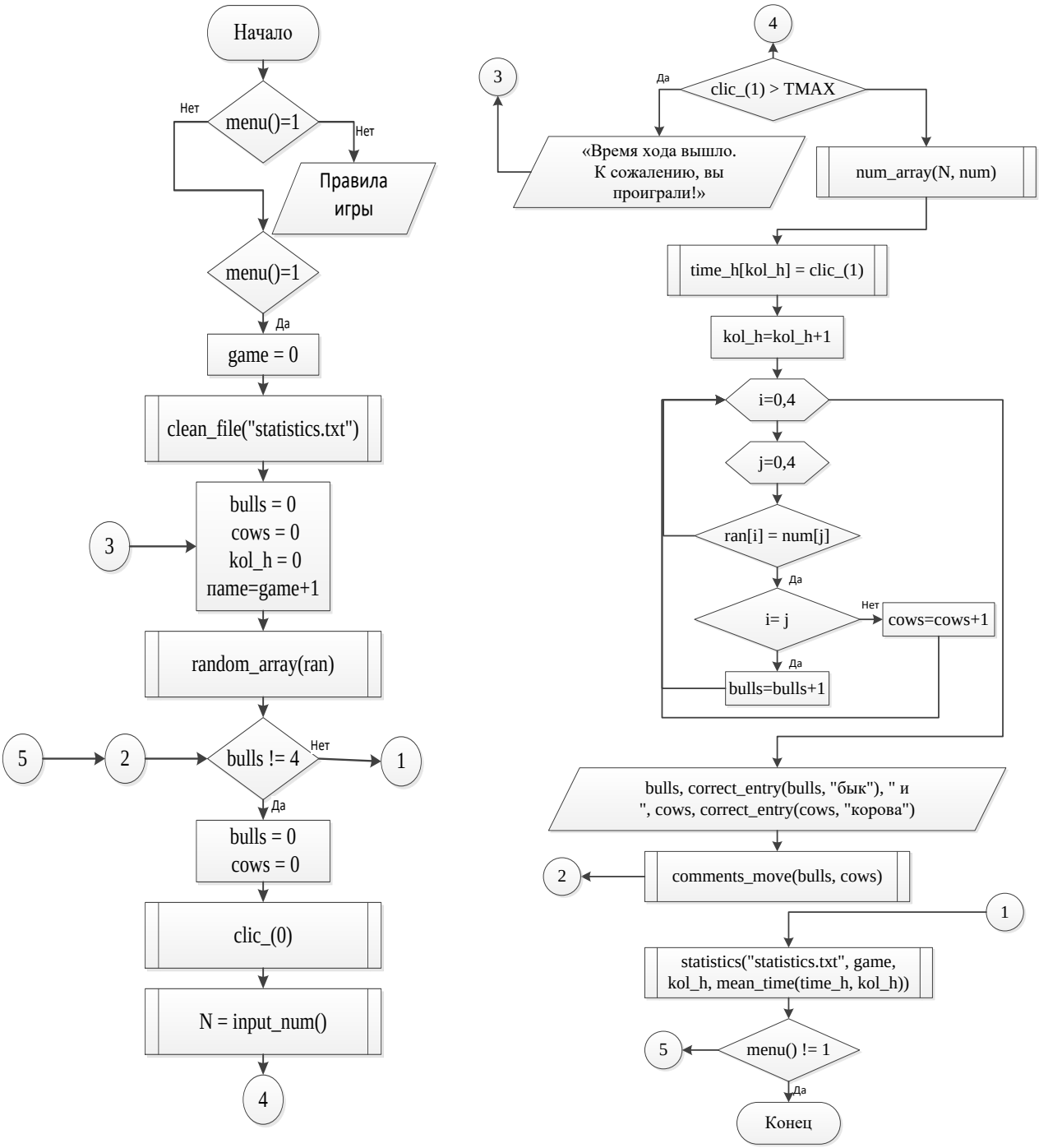


Рис. 1. Блок-схема игры «Быки и коровы»

4) string correct_entry (int num, string s);

Функция предназначена для корректного вывода количества быков и коров. Данная функция возвращает отредактированное слово (корова или бык). Блок-схема представлена на рис.5.

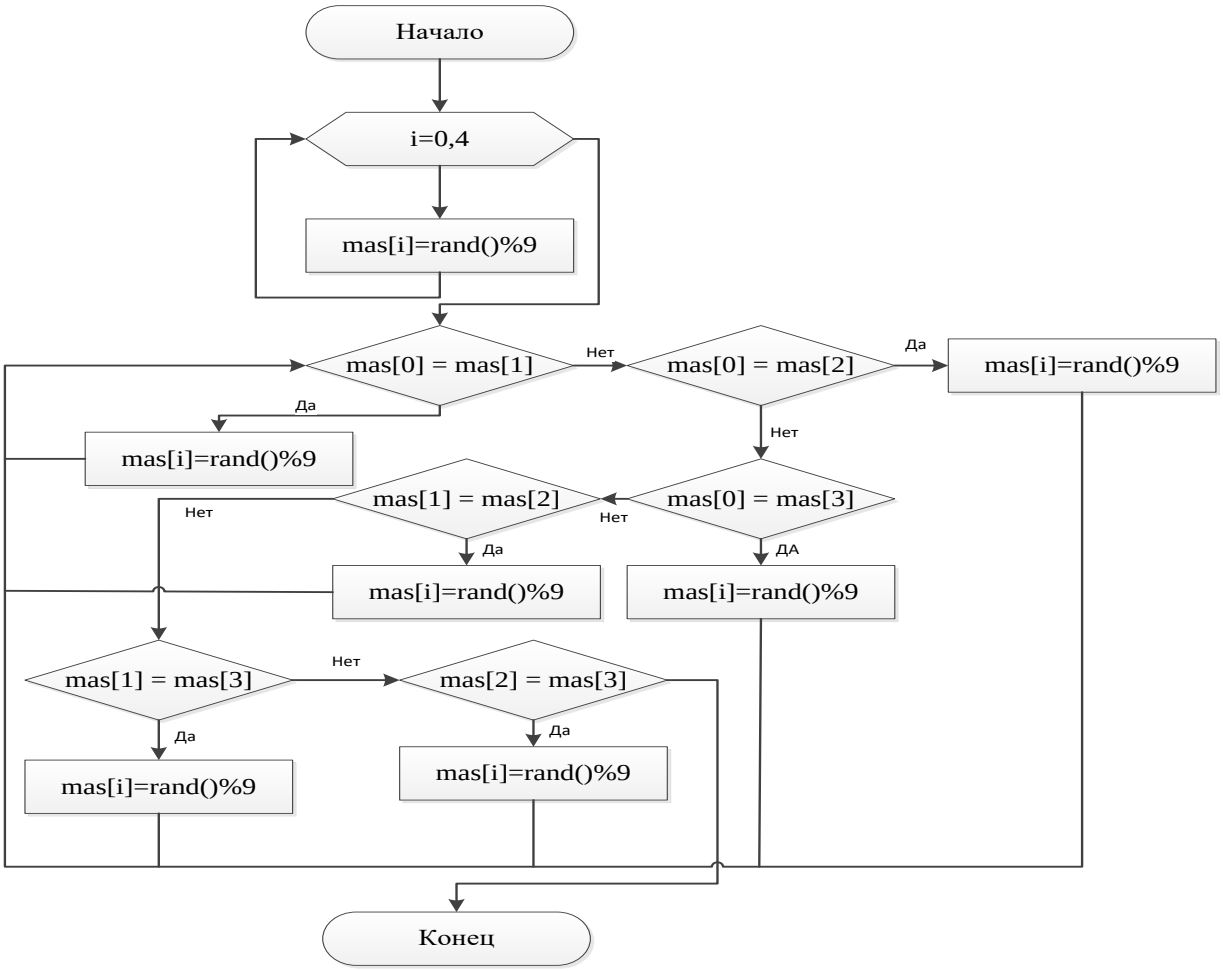


Рис.2. Блок-схема функции для задания массива случайным образом

5) void comments_move(int a, int b);

Функция предназначена для вывода комментария к ходу пользователя. Данная функция ничего не возвращает. Блок-схема представлена в приложении.

6) void clean_file(string f);

Функция очищает указанный файл. Данная функция ничего не возвращает. Блок-схема представлена в приложении.

7) void statistics(string f, int i, int kol, float s);

Функция предназначена для записи статистики об игре в указанный файл. Данная функция ничего не возвращает. Блок-схема представлена в приложении.

8) int menu();

Функция предназначена для диалога с пользователем. Данная функция возвращает 1, если ответ пользователя положителен, иначе 0. Блок-схема представлена в приложении.

9) int clic_(int init);

Функция предназначена для определения времени хода пользователя. Данная функция возвращает время хода. Блок-схема представлена в приложении.

10) float mean_time(float *mas, int n);

Функция предназначена для определения среднего времени хода за одну игра. Данная функция возвращает среднее время хода. Блок-схема представлена в приложении.

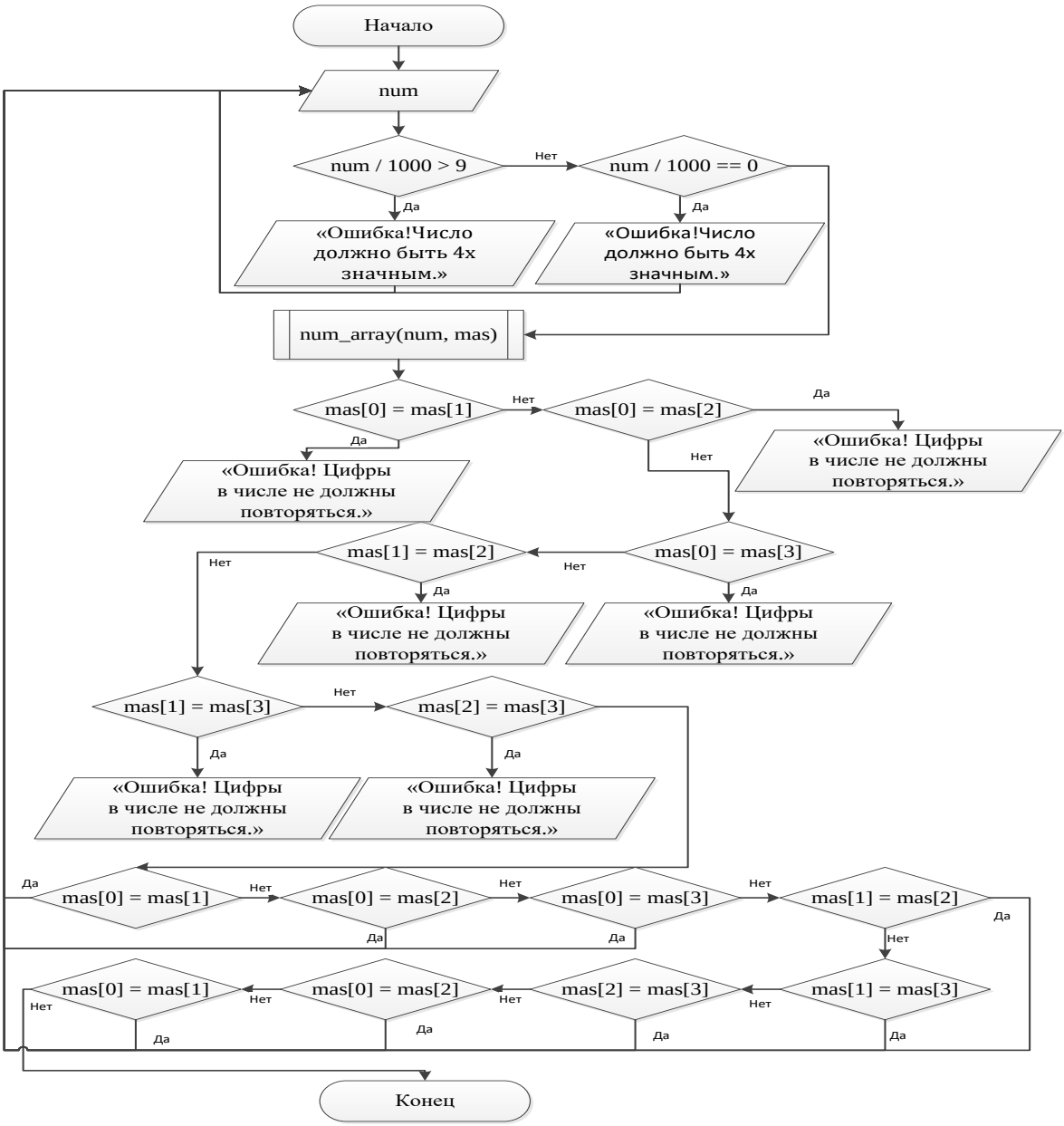


Рис.3. Блок-схема функции проверки введенного числа

Для реализации представленных алгоритмов требуется использование библиотечных функций. Так генерация основных данных – «загаданной» комбинации осуществляется с использование математических функции srand() и rand().

Ввод данных, а именно: заполнение двумерного массива происходит

случайным образом с применением функции функцией rand() из стандартной библиотеки C/C++.

Функция rand()генерирует числа в диапазоне от 0 до RAND_MAX и возвращает unsignedlong число. RAND_MAX – это константа, определенная в библиотеке <cstdlib> языка C++ и в библиотеке <stdlib.h> языка C, равная 32767, но может быть и больше, в зависимости от компилятора.

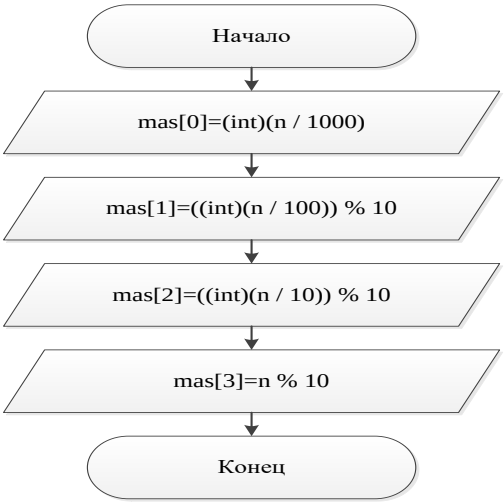


Рис.4. Блок-схема функции преобразующей число в массив

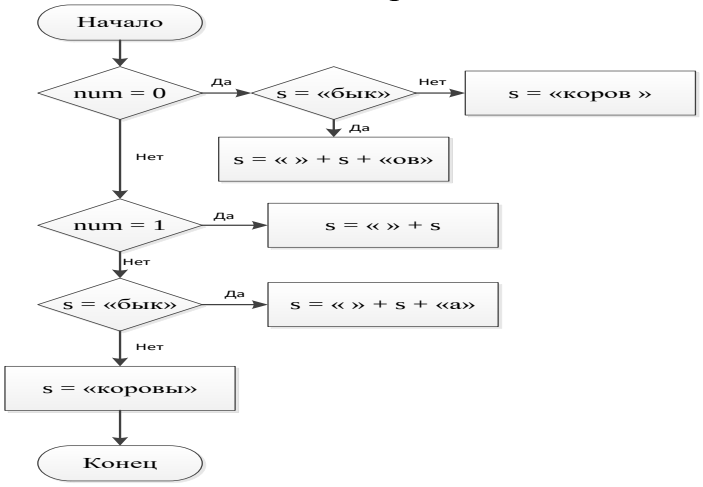


Рис.5. Блок-схема функции корректного вывода количества быков и коров

Случайное число генерируется, исходя из определенных параметров системы, поэтому для генерации разных случайных чисел, нужно в начале программы использовать функцию **srand**, которой в качестве аргумента передается число, инициирующее работу функции rand().

Алгоритм генерации случайного числа *m* следующий:

srand (time(NULL));

m = a + rand() % b;

где a – это начальная точка, с которой начинается генерация, b - это величина сдвига, определяющая интервал, в котором будет генерироваться случайное число.

Важной частью программы является – диалог с пользователем. Ввод-вывод данных в языке C++ осуществляется либо с помощью функций ввода-вывода в стили C, либо с использованием библиотеки классов C++. Преимущество объектов C++ в том, что они легче в использовании.

В данном решении для удобства пользователя использованы потоки **cin** и **cout.** Описание объектов для управления вводом-выводом содержится в файле **iostream**. При подключении этого файла с помощью директивы #include<iostream> в программе автоматически создаются виртуальные каналы связи **cin** для ввода с клавиатуры и **cout** для вывода на экран, а также операции помещения (вставки) в поток << и чтения из потока >>.

С помощью объекта **cin** и операции >> можно присвоить значение любой переменной. Например, если переменная x описана как целочисленная, то команда cin>> x; означает, что в переменную x будет записано некое целое

число, введенное с клавиатуры. Если необходимо ввести несколько переменных, то следует написать cin>>x>>y>>z;.

Объект **cout** и операция >> позволяет вывести на экран значение любой переменной или текст. Текст необходимо заключать в двойные кавычки. Запись cout>> x; означает вывод на экран значения переменной x.

Для реализации дополнительного требования – записи в файл использована стандартная последовательность вызова библиотечных функций.

1. Файл открывается. Это означает, что программа "захватывает" заданный по имени файл, сообщает Windows, что далее она будет с ним работать. Данный шаг нужен, чтобы не возникало конфликтов, когда несколько программ одновременно хотят записывать информацию в один и тот же файл. Правда, считывать данные из файла, очевидно, допустимо одновременно множеством программ, поэтому в операции открытия файла обычно уточняется, что файл открывается "на чтение" (считывание информации, которая не меняется) либо "на запись" (данные в файле модифицируются).

Операция открытия файла возвращает некий идентификатор (как правило, целое число), которое идентифицирует в программе в дальнейшем нужный открытый файл. Этот идентификатор запоминается в переменной; обычно такая переменная называется файловой переменной.

2. Ведется работа с файлом. Из него данные либо считываются, либо в него записываются.

3. Файл закрывается. После этой операции он снова доступен другим программам для обработки.

Для работы с файлами в код включен заголовочный файл <**fstream**>.

3. Тестирование и описание работы программы

Первое, что делает программа при запуске – это приветствует и предлагает ознакомиться с правилами игры (рис. 6).

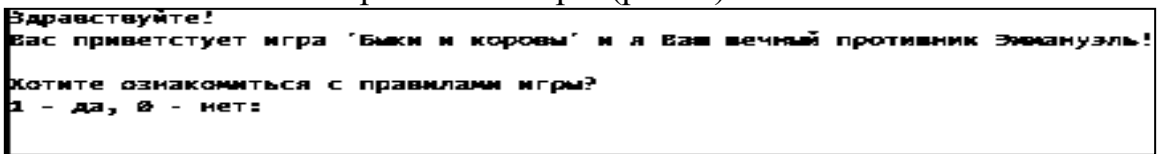


Рис.6. Окно приветствия

Если пользователь вводит «1», то на экран выводятся правила игры и следующее предложение о начале игры (рис.7.1), а если «0» - то просто следующее предложение (рис.7.2).

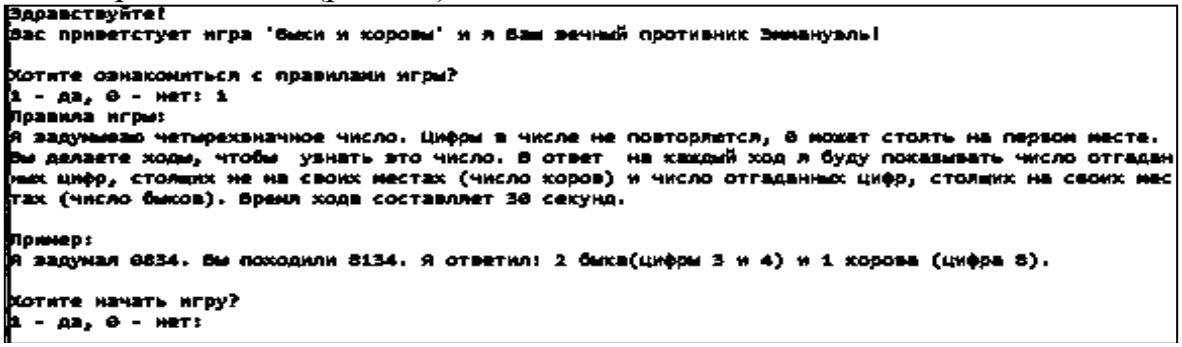


Рис.7.1. Окно ознакомления с правилами игры

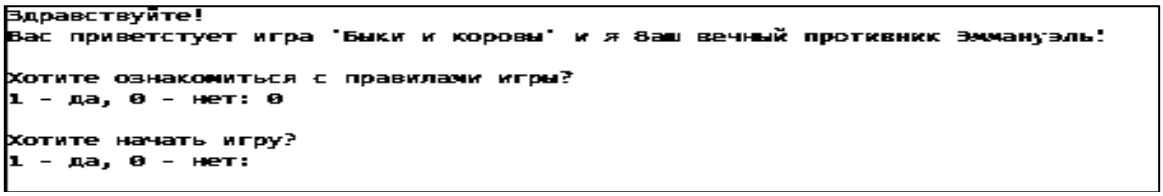


Рис. 7.2. Окно с предложение начать игру

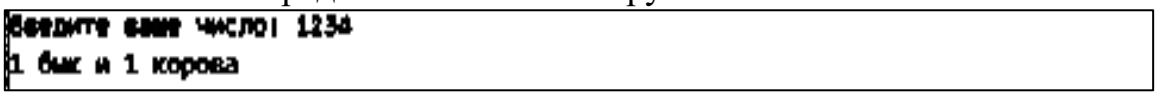


Рис.8. Результат ввода числа

Если пользователь вводит «1», то начинается игра и предлагается ввести число, а если «0» - то пользователь выходит их игр по нажатию любой клавиши. После начала игры, пользователь вводит предполагаемые числа, до тех пор, пока не угадает число, пока не будет 4 быка. После каждого хода будет выводить количество быков и коров (рис.8). Если ход будет более правильный, к нему еще будет выводиться комментарий. Если же будет введено некорректное число, программа выдаст ошибка и предложит ввести число заново (рис.9).

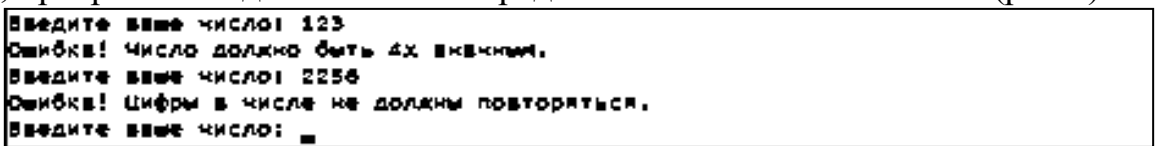


Рис.9. Ошибка ввода числа

Если пользователь сделал ход позже 30 секунд, то выводится сообщение о проигрыше, и программа предложит сыграть еще раз. Если игра прошла успешно, то на экран будет выведено сообщение о победе (рис.10).

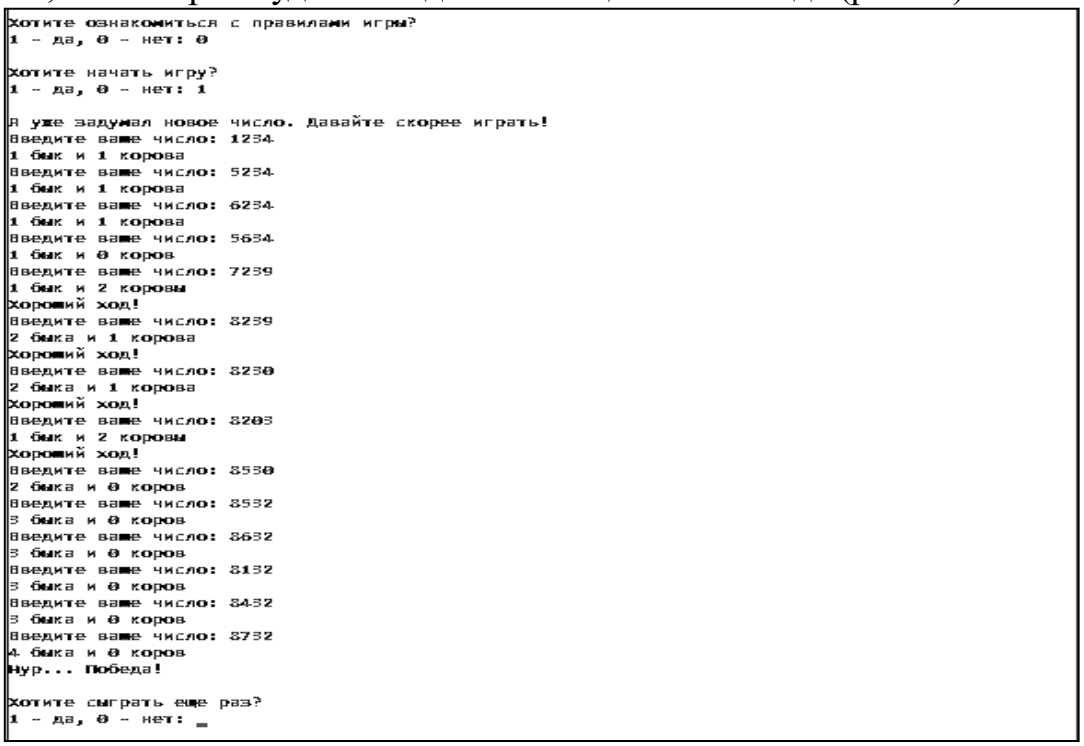


Рис.10. Игра, закончившаяся победой

После окончания игры, программа предлагает сыграть еще раз. Если пользователь вводит «1» игра начинается заново, если же «0» - игра

заканчивается. Статистика игры будет записана в файл (рис.11).

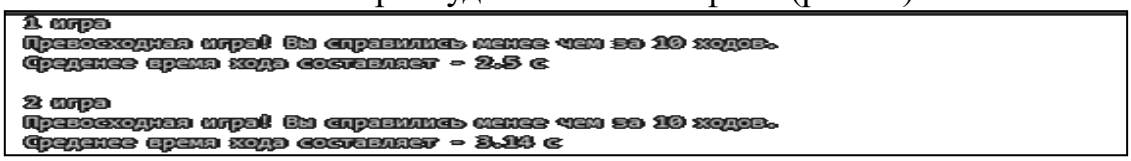


Рис.11. Статистика об игре

ЗАКЛЮЧЕНИЕ

В ходе курсового проектирования на языке высокого уровняС++ с использованием среды разработки MVS 2015 был реализован программный продукт – игра «Быки и коровы». Разработанная программа реализует два сценария игры: произвольный (без ограничение времени хода) и фиксированный (когда время хода ограничено). Статистика времени каждого хода накапливается в файле и обрабатывается по окончанию игры. В игре реализован диалог с пользователем, выводятся сообщения о каждом ходе, предлагается повторить игру, а также подбадривающие игрока фразы.

ОГЛАВЛЕНИЕ

ОСНОВЫ ПРОГРАММИРОВАНИЯ И АЛГОРИТМИЗАЦИИ

Методические указания
к выполнению курсового проекта
для студентов бакалавриата
направления 09.03.02 «Информационные системы и технологии»
всех форм обучения

Составители: Минакова Ольга Владимировна
Курипта Оксана Валериевна

Подписано в печать 01.12.2016г . Формат 60 х 84 1/16 Уч.-изд.л. 2,3.
Усл.-печ.л. 2,4. Бумага писчая. Тираж 70 экз. Заказ №_____

Отпечатано: отдел оперативной полиграфии
издательства учебной литературы и учебно-методических пособий
Воронежского государственного технического университета
396006, Воронеж, ул. 20-летия Октября,84