

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"Московский технологический университет"**

**МИРЭА**



Институт Информационных Технологий

*(наименование института)*

Кафедра Общей Информатики

*(наименование кафедры)*

**КУРСОВОЙ ПРОЕКТ (РАБОТА) по дисциплине**

**«**Информатика**»**

*(наименование дисциплины)*

**Тема курсового проекта (работы): «Структуры и алгоритмы обработки данных; перевод чисел в системы счисления с другим основанием»**

*(наименование темы)*

Студент группы ИВБО-02-15

*(учебная группа)*

Руководитель курсового проекта (работы)

*должность, звание, ученая степень*

Рецензент (*при наличии*)

*должность, звание, ученая степень*

|  |  |
| --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2017 г. |
| «Допущен к защите» | «\_\_»\_\_\_\_\_\_\_2017 г. |

Москва 2017

Прохоров А.В.

Карпов Д.А.

к.т.н.

*(подпись студента)*

*(подпись руководителя)*

|  |  |
| --- | --- |
| для прик эмбл | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"Московский технологический университет"**  **МИРЭА** | |
| Институт Информационных Технологий  *(наименование института)*  Кафедра Общей Информатики  *(наименование кафедры)* | |
|  | |
|  | **Утверждаю** |
|  | Заведующий кафедрой: Карпов  Дмитрий Анатольевич  *Подпись ФИО* |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_2017 г. |
| **ЗАДАНИЕ** | |
| **на выполнение курсового проекта (работы)** по дисциплине | |
| «Информатика» | |

Студент: Прохоров Андрей Валерьевич Группа: ИВБО-02-15

**Тема: Структуры и алгоритмы обработки данных; перевод чисел в системы счисления с другим основанием**

**Исходные данные: разработать программу перевода чисел из одной системы счисления в другую; познакомиться с основными типами данных и их структур.**

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

|  |
| --- |
| **Срок представления к защите курсового проекта (работы):** **до** «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. |
|  |
| **Задание на выполнение курсовой проект (работу) выдал \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Карпов Д.А.)**  *Подпись руководителя Ф.И.О руководителя* |

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017г.**

**Задание на курсовой проект (работу) получил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Прохоров А.В.)**

*Подпись обучающегося* *Ф.И.О. исполнителя*

Москва 2017

|  |
| --- |
|  |



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"Московский технологический университет"**

**МИРЭА**



Институт Информационных Технологий

*(наименование института)*

Кафедра Общей Информатики

*(наименование кафедры)*

**Протокол заседания комиссии по защите курсового проекта**

от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017г. №\_\_\_\_\_\_\_\_\_\_

Состав комиссии: зав. кафедрой, к.т.н. Карпов Дмитрий Анатольевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(должность, ученая степень, ученое звание)*

Утверждена распоряжением заведующего кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(наименование кафедры)*

от «\_\_\_» \_\_\_\_\_\_\_\_ 2017 г. №\_\_\_\_\_\_\_\_.

Слушали защиту курсового проекта (работы): «Структуры и алгоритмы обработки данных; перевод чисел в системы счисления с другим основанием»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по дисциплине «Информатика»

студента группы ИВБО-02-15 Прохорова Андрея Валерьевича

*(группа)* *(Ф.И.О.)*

Во время защиты курсового проекта (работы) были заданы следующие вопросы:

1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Итоговая (комплексная) оценка выполнения и защиты курсового проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)* *(Ф.И.О.)*

Москва 2017

**Содержание**

**Введение** 5

**Теоретическая часть** 7

Данные в ЭВМ 7

Логический уровень данных 7

Структуры данных 8

Физический уровень данных 10

Системы счисления в ЭВМ 10

Принцип перевода чисел 11

Алгоритмы обработки данных 12

Методы разработки алгоритмов 13

**Практическая часть** 14

Описание алгоритма программы 14

Схема алгоритма программы 15

Исходный код программы с комментариями 18

Контрольные прогоны программы 21

**Заключение** 22

**Список литературы** 22

**Приложение** 23

**Введение**

Нас окружают данные. Звуковые, зрительные, осязательные, обонятельные, текстовые. Это все различные виды информации. Если же речь заходит о вычислительной технике, то данные и работа с ними — то, ради чего компьютеры и были созданы. Если бы мы могли обрабатывать столь значительные объемы данных без помощи ЭВМ — вопрос об их необходимости и не поднимался.

Но мы используем ЭВМ для сложных расчетов и операций с большим количеством данных, доступ к которым должен быть максимально быстрым и полным. Для реализации этой полноты и оперативности используются структуры данных. Однако не стоит забывать, что любые данные, записанные в компьютере — числа. И десятичная система счисления, привычная человеку, для компьютера не является оптимальной. ЭВМ используют различные системы счисления для своих задач.

Базовые знания о системах счисления люди получают еще в рамках школьной программы информатики, однако многим кажется, что при «реальном программировании» нет необходимости в переводе чисел в системы с разным основанием. И это заблуждение. Зачастую в программе намного удобнее использовать не десятичную систему счисления. При этом понятность кода программы не уменьшается.

В большинстве случаев данные хранятся и обрабатываются алгоритмами программ. Поэтому важна задача написания алгоритмов, оптимальных во времени выполнения и затрачиваемой вычислительной мощности. Знать основные типы данных и их структуры, принципы их записи и обработки в компьютере, а также базовые алгоритмы работы с ними необходимо любому человеку, связавшему себя с информатикой или разработкой ПО.

В связи с описанными выше причинами я считаю эту работу актуальной для изучения специалисту ИТ-сферы. Также, в учебной литературе и интернет-ресурсах есть большое количество информации на данную тему, избавляющее от ряда трудностей.

В данной работе мы постараемся дать наиболее понятное представление о современных и распространенных типах данных, их структурах, некоторых алгоритмах работы с ними, а также о работе с числовыми данными в ЭВМ. Сформулируем для этого цели курсовой работы, ее задачи, изучим теорию по теме и разработаем программу перевода чисел в любую систему счисления, описываемую арабскими цифрами и английским алфавитом. Программу оформим в виде консольного приложения на одном из актуальных языков программирования. Из всего проделанного сделаем выводы в виде заключения.

**Цели работы**

Познакомиться с основными типами данных, их структурами и алгоритмами обработки данных. Изучить популярные системы счисления, алгоритм перевода чисел в другие системы счисления, принцип работы компьютера с числами. Получить практические навыки работы с системами счисления и компьютерной интерпретацией чисел путем написания программы.

**Постановка задачи**

Изучить теоретический материал по теме. Разработать программу, реализующую алгоритм перевода чисел, введенных пользователем, в другие системы счисления. Осуществлять проверку ввода. Ограничиться 36-ричной системой счисления и положительными числами. Сделать выводы из проделанной работы.

**1. Теоретическая часть**

**1.1. Данные в ЭВМ**

Данные — представление информации в формализованном виде, пригодном для передачи и обработки. Различают три уровня представления данных:

**1.** Предметный (пользовательский уровень) — представление данных, понятных пользователю. Пример: список студентов ВУЗа.

**2.** Логический уровень — тип организации данных в программе. Пример: список студентов ВУЗа представляет собой массив данных.

**3.** Физический уровень — организация данных в физической памяти компьютера. Пример: зона ячеек памяти, содержащая массив со студентами ВУЗа.

Далее в данной работе данные и их структуры будут рассмотрены на физическом уровне ячеек памяти и без учета представления в машинной памяти — в логическом виде.

**1.2. Логический уровень данных**

На логическом уровне организации данных различаю следующие их типы:

**Логические** (булевы) — данные, принимающие одно из двух состояний: «истина» или «ложь». Обычно обозначаются как bool или boolean. «Истина» может обозначаться как true, а «ложь», соответственно, false. В языках C и C++ любое ненулевое число трактуется как «истина», а ноль — как «ложь».

**Целочисленные** типы содержат в себе значения чисел (знаковых и беззнаковых).

**Числа с плавающей запятой** используются для представления вещественных чисел. В этом случае число записывается в виде x=a\*10^b. Где 0<=a<1, а b — некоторое целое число из определённого диапазона. a —мантисса числа, b — порядок. У мантиссы хранятся несколько цифр после запятой, а b — хранится полностью.

**Строковый тип** — последовательность символов, которая рассматривается как единое целое в контексте переменной. В разных языках программирования накладываются разные ограничения на строковые переменные.

**Указатель** — переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения для обозначения того, что в данный момент в переменной ничего не записано.

Для языка С++ тип данных, диапазон принимаемых ими значений и размер отражены в таблице №1 (см. Приложение).

**1.2.1 Структуры данных**

Данные в памяти могут быть организованы в различные виды структур данных, таких как массивы, связанные списки или объекты. Под структурой данных в общем случае понимают множество элементов данных и множество связей между ними. Такое определение охватывает все возможные подходы к структуризации данных, но в каждой конкретной задаче используются те или иные его аспекты. Структуры данных могут хранить данные различных типов, включая числа, строки и другие структуры данных.

*Линейные структуры данных* — это упорядоченные структуры, в которых адрес элемента однозначно определяется его номером. Линейные структуры данных обладают следующими свойствами:

**1.** Каждый элемент имеет не более 1 предшественника

**2.** Два разных элемента не могут иметь одинакового последователя

К линейным структурам данных можно отнести: массив, вектор, список, стек, очередь, дек.

**Массив** — одна из простейших и наиболее широко применяемых в компьютерных программах линейных структур данных. В любом языке программирования массивы имеют несколько общих свойств:

**1.** Все элементы массива имеют одинаковый тип, поэтому массивы называют однородными структурами данных.

**2.** Существует прямой доступ к элементам массива, или же — существует индексирование элементов.

**Вектор** — схожая с массивом структура данных, отличающаяся возможность менять размер в процессе работы с ним. Обладает поддержкой функций контейнера.

**Список** — простейший тип данных структуры, состоящей из элементов (узлов). Каждый узел включает в себя в классическом варианте два поля:

**1.** данные, хранящиеся в узле

**2.** указатель на следующий узел в списке.

Существуют односвязные списки – движение только «вперед» и двусвязные — «вперед» и «назад».

**Стек** — структура данных с доступом к элементам по принципу LIFO (Last In First Out - последний пришел - первый вышел). Данные добавляются в начало (конец, кому как удобно), оттуда же и извлекаются.

**Очередь —** структура данных с доступом к элементам по принципу FIFO (First In First Out - Первый пришел - Первый вышел). Данные добавляются в конец, а извлекаются из начала.

**Дек** — это сокращенная фраза «double-ended-queue» — двусторонняя очередь. Контейнер дек очень похож на контейнер вектор. Так же, как и вектора, деки являются динамическими массивами. Разница между вектором и деком состоит лишь в том, что в деках динамический массив открыт с двух сторон. Это и позволяет очень быстро добавлять новые элементы как в конец, так и в начало контейнера.

*Нелинейные структуры данных* **—** это структуры, подразумевающие нелинейный порядок обхода. Их нельзя линеаризовать, а элементы могут иметь множественные связи и ветвление.

**Дерево** – иерархическая структура данных.  Элемент этой структуры данных называется вершиной. Дерево представляет собой вершину, имеющую ограниченное число связей (ветвей) к другим деревьям. Нижележащие деревья для текущей вершины называются поддеревьями, а их головные вершины —потомками. По отношению к потомкам текущая вершина называется предком. Предок может иметь несколько потомков, но не наоборот. Пример — см. Приложение (рис. №1).

**Сеть** — совокупность непустого множества вершин и наборов пар вершин. В отличии от дерева, потомки в графе могут иметь несколько предков. Пример — см. Приложение (рис. №2).

**1.3. Физический уровень данных**

Компьютер хранит данные в ячейках памяти в виде нулей и единиц — отсутствия электрического тока и его наличия соответственно. Это двоичная система записи чисел. Однако компьютер использует не только двоичную систему счисления, но и другие позиционные системы: восьмеричную, шестнадцатеричную. Последняя, например, применяется для адресации ячеек памяти и устройств. Чтобы понять, почему ЭВМ используют именно эти системы счисления, рассмотрим их подробнее.

**1.3.1.** **Системы счисления в ЭВМ**

**Бинарная** система счисления (двоичная) — позиционная система счисления с основанием два. Использует цифры 0 и 1 в качестве средства записи чисел. Получила широкое распространение в цифровой технике благодаря следующим преимуществам:

**1.** Устойчивость — имеет лишь два состояния, сродни наличию тока и его отсутствию.

**2.** Отсутствие электрических помех, способных создать трудности при распознавании промежуточных значений.

**3.** Возможность использования булевой алгебры.

**Восьмеричная** система счисления — используется из-за главного недостатка двоичной системой: быстрый рост разрядов. Восьмеричная система использует в три раза меньше разрядов, чем бинарная для записи одного и того же числа. Используемые ифры: 0, 1, 2, 3, 4, 5, 6, 7.

**Десятеричная** система счисления — классическая и знакомая с детства система счета, сочетающая простоту и как следствие — понятность человеку. Использует цифры от 0 до 9.

**Шестнадцатеричная** система счисления — дальнейшее развитие восьмеричной системы. Экономит в 4 раза больше разрядов в сравнении с двоичной. Так же применяется для адресации устройств в микропроцессорах. Использует цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

**1.3.2. Принцип перевода чисел**

Компьютер хранит все в двоичном коде, но способен осуществлять перевод в любую желаемую систему счисления. Для перевода целого числа с основанием b1 в другую с основанием b2 требуется последовательно делить число и его частные на основание b2, пока остаток от деления не будет превышать b2. Этот принцип называется **методом деления** и прост для понимания. Пример его использования можно увидеть на рис. №3:

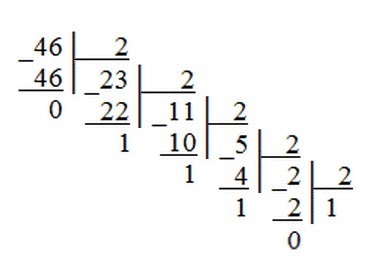


Рис. №3 – перевод числа 4610 в двоичную систему счисления

Аналогично осуществляется перевод и в другие системы (рис. №4).

Для обратного перевода требуется сложить коэффициенты при разрядах, умноженные на основание в соответствующем порядке.

Пример: 101110 = 1\*25+0\*24+1\*23+1\*22+1\*21 = 32+8+4+2=46.

**1.4. Алгоритмы обработки данных**

Алгоритм — это точная инструкция исполнителю на понятном ему языке, определяющая выполнение операции, ведущая от варьируемых начальных данных к искомому результату. В процессе решения прикладных задач выбор подходящего алгоритма — важная часть разработки. Алгоритм должен удовлетворять следующим критериям:

**1.** быть простым для понимания человеком и перевода в программный код

**2.** эффективно использовать вычислительную мощность и выполняться по максимально быстро, иными словами — быть эффективным

При написании алгоритма стоит учитывать это. Если решение задачи требует значительных вычислительных затрат, то стоимость выполнения программы может превысить стоимость написания программы. Прежде чем принимать решение об использовании того или иного алгоритма, необходимо оценить сложность и эффективность этого алгоритма, условия его выполнения.

Для оценки сложности алгоритма применяют величину О(). Она отражающая порядок величины требуемого ресурса (времени или задействованной памяти) в зависимости от сложности задачи. Можно привести следующую классификацию алгоритмов, при n входных данных:

**1**. постоянные – сложность не зависит от n: O(1);

**2**. линейные – сложность О(n);

**3**. полиномиальные – сложность O(n\*m), где m – константа;

**4**. экспоненциальные – сложность O(t\*f(n)), где t – константа, которая больше 1, а f(n) – полиномиальная функция;

**5**. супер полиномиальные – сложность O(с\*f(n)), где с – константа, а f(n) — функция возрастающая быстрее постоянной, но медленнее линейной;

Простые задачи (решаемые) – это задачи, решаемые за полиномиальное время. Труднорешаемые задачи не решаются за полиномиальное время, либо алгоритм решения за полиномиальное время не найден. Для примера приведем числа, иллюстрирующие скорость роста для нескольких функций, которые часто используются при оценке временной сложности алгоритмов — см. Приложение (Таблица. №2). Если считать, что числа соответствуют микросекундам, то для задачи с 1048476 элементами алгоритму со временем работы T(log\*n) потребуется 20 микросекунд, а алгоритму со временем работы T(n2) – более 12 дней. Отсюда следует важность использования наиболее быстрого алгоритма.

**1.4.1. Методы разработки алгоритмов**

**Метод грубой силы –** (brute force), решение «в лоб», представляющий собой полный перебор.

**Метод декомпозиции** — методом разбиения, возможно, является самым важным и наиболее широко применимым методом проектирования эффективных алгоритмов. Он предполагает разбиение задачи на более мелкие задачи. На основе решений этих более мелких задач можно легко получить решение исходной задачи.

**Динамическое программирование** — запись полученных на различных этапах выполнения программы значений и последующее их использование позволяет ускорит выполнение задачи на порядки.

**Поиск с возвратом** — общий метод нахождения решений задачи, в которой требуется полный перебор  всех возможных вариантов в некотором множестве М. Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше.

**Локальный поиск** — метод использования группа алгоритмов, в которых поиск ведется только на основании текущего состояния, а ранее пройденные состояния не учитываются и не запоминаются.

Это самые популярные алгоритмы работы с данными, более подробно с каждым их них можно ознакомиться в специализированной литературе. В практической части будет использоваться метод декомпозиции (разбиение на подпрограммы).

**2. Практическая часть**

Для разработки алгоритма и его реализации был выбран язык программирования С++. Алгоритм обрабатывает данные, введенные пользователем, а именно — переводит числа из одной системы счисления в другую. При работе программы используются структуры данных — массивы.

**2.1. Описание алгоритма программы**

Программа получает с клавиатуры число в системе счисления с снованием 2<=base<=36. Значения, выходящие за этот отрезок отбрасываются функцией cin.ignor(). Программа требует повторного корректного ввода. Введенное число проходит на символьный состав с помощью функции filter(): допускаются только арабские цифры и буквы английского алфавита. Числа могут быть введены в любом регистре — функция toupper() переводит символы в верхний регистр для корректного дальнейшего перевода. Если число содержит символы, выходящие за пределы основания системы счисления, программа потребует повторный ввод основания. В случае верности введенных данных функция stoi() осуществит перевод введенного числа в десятичную систему счисления — она выступает системой-посредником. Далее вводится основание желаемой системы счисления — newbase. В случае удовлетворяющего значения нового основания, функция ten\_base() переводит переданное от stoi() десятичное число в систему с указанным пользователем основанием. Число в новой системе счисления выводится на экран.

**2.2. Схема алгоритма программы**

Начало

Инициализация number, temp, base, numeral, newbase, newnumber, finalchar

Ввод number

Инициализация up, i, chislo

Нет

сhislo[i]

Да

chislo[i] = toupper(chislo[i])

temp.push\_back(toupper(chislo[i]))

i++

number.clear(); number = temp

1

Рис. 5. Схема алгоритма программы

1

Ввод base

Нет

2<=base<=36

i=base, j=0;

Да

j<(number.length() + 1)

Нет

2

i <36

Да

Нет

Да

chislo[j] =numeral[i]

Нет

Да

Ввод base

i++

j++

Рис. 5. Схема алгоритма программы

2

Ввод newbase

2<=newbase<=36

Нет

Инициализация newnumber

Да

int newnumber = stoi(number, nullptr, base)

Вывод number в 10-ой системе (newnumber)

ten\_base(newnumber, newbase)

Вывод newnumber в системе счисления newbase

cin.clear()

Конец

Рис. 5. Схема алгоритма программы

**2.3. Исходный код программы с комментариями**

#include <iostream>

#include <string>

#include <cstring>

#include <cstdlib>

using namespace std;

string finalchar; //строка, в которую записываются лишь символы от 0 до 9, A-Z

char numeral[36] = { '0','1','2','3','4','5','6','7','8','9', //все возможные цифры

'A','B','C','D','E','F','G','H','I','J','K','L','M','N',

'O','P','Q','R','S','T','U','V','W','X','Y','Z' };

char newsystem\_number(int newnumber) { //функция записи числа из десятичной с.с. в новой

return numeral[newnumber];

}

void ten\_base(int newnumber, int newbase) { //функция подсчета числа из десятичной с.с. в систему с основанием newbase

if (newnumber<newbase) {

cout << newsystem\_number(newnumber);

return;

}

ten\_base(newnumber / newbase, newbase);

cout << newsystem\_number(newnumber % newbase);

}

string filter(const string &to, const string &remove) //фильтр ввода числа, удаляющие из строки символы-аргументы функции

{

for (string::const\_iterator it = to.begin(); it != to.end(); ++it)

{

if (remove.find(\*it) == string::npos)

{

finalchar += \*it;

}

}

return finalchar;

}

int main()

{

setlocale(LC\_ALL, "Russian");

int base, newbase; //введенное пользователем основание и основание той системы, в которую будет осуществлен перевод

string number, temp; //введенное число, промежуточная переменная для хранинеия числа в верхнем регистре

cout << "Введите положительное число: ";

getline(cin, number); //ввод числа

filter(number, "'\\,<\".<-+\*&!?"); //фильтр ввода

number = finalchar; //очищеннон от лишнего число

char \*chislo = new char[number.length() + 1]; //сивмольный массив, содержащий в себе введенное число

strcpy(chislo, number.c\_str());

char up;

int i = 0;

while (chislo[i])

{

chislo[i] = toupper(chislo[i]); //все введенные символы становятся заглаными

temp.push\_back(toupper(chislo[i])); //запись заглавных чисел в temp

i++;

}

number.clear();

number = temp; //введенное число из заглавных символов

cout << "Введите основание, не превышающее 36: "; //ввод начального основания числа

cin >> base;

if (base<2 || base>36 || cin.fail()) //проверка на ввод символов или значений, не принадлежащих промежутку

cout << "Неверно. Введите число от 2 до 36." << endl;

cin.clear(); //очистка потока вводных данных, предотвращающая циклический вывод предупреждения

cin.ignore(cin.rdbuf()->in\_avail()); //игнорирование (ignore) колличества символов (avail) буфера (rdbuf)

while (base<2 || base >36) {

cout << "Введите основание, не превышающее 36: ";

cin >> base;

if (base<2 || base>36 || cin.fail())

cout << "Неверно. Введите число от 2 до 36." << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

}

for (int j = 0; j<(number.length() + 1); j++) //проверка на соответсвие символам числа его основанию

{

for (int i = base; i<36; i++)

{

if (chislo[j] == numeral[i])

{

cout << "Неправильное основание системы счисления. Введите корректное: ";

cin >> base;

i = base;

j = 0;

}

}

}

cout << "Введите новое основание системы счисления, не превышающее 36: "; //ввод нового основания числа

cin >> newbase;

if (newbase<2 || newbase>36 || cin.fail()) //проверка

cout << "Неверно. Введите число от 2 до 36." << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

while (newbase<2 || newbase >36) {

cout << "Введите основание, не превышающее 36: ";

cin >> newbase;

if (newbase<2 || newbase>36 || cin.fail())

cout << "Неверно. Введите число от 2 до 36." << endl;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

}

int newnumber = stoi(number, nullptr, base); //перевод числа из системы с указанным основанием в десятичную

cout << endl;

cout << "Число " << number << " в 10-ой системе счисления: " << newnumber << '\n';

cout << endl;

cout << "Число " << number << " в " << newbase << "-ой системе счисления: ";

ten\_base(newnumber, newbase); //запись числа из 10-ой системы счисления в желаемую пользователем

cout << endl;

cout << endl;

cin.clear(); //очиска буфера ввода

system("Pause");

return 0;

}

**2.4. Контрольные прогоны программы**

Тест №1 — перевод из 17-ричной в 24-ричную систему счисления. Пример проверка ввода.

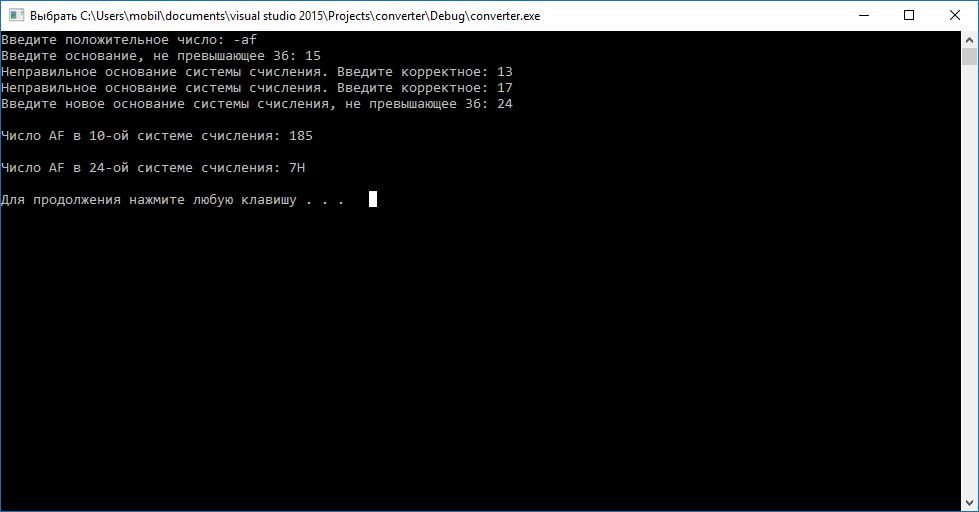


Рис. №6 – результаты теста №1

Тест №2 — обратная проверка правильности алгоритма перевода. Проверка ввода основания системы.

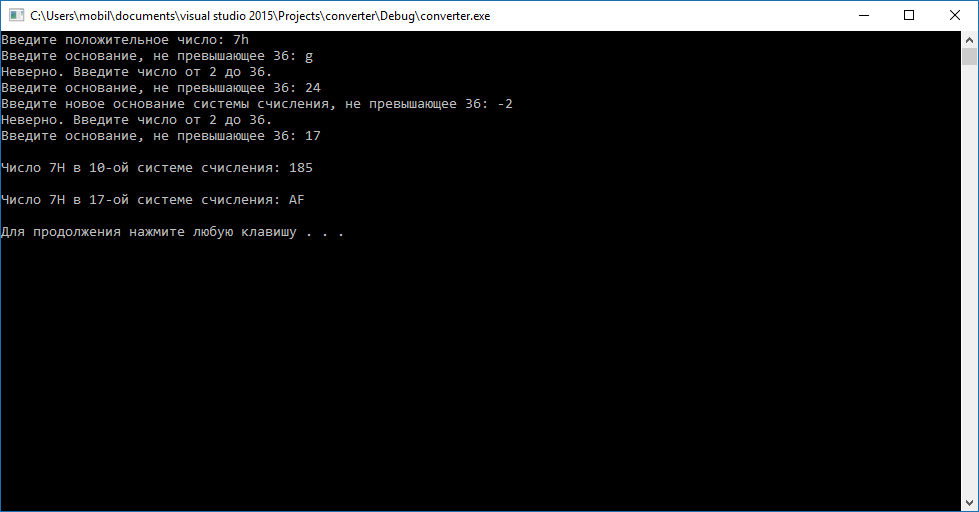


Рис. №7 – результаты теста №2

**Заключение**

В ходе выполнения курсовой работы мы познакомились с основными структурами данных, их физическим и логическим представлением. Поняли необходимость написания наиболее оптимальных (быстрых) алгоритмов обработки данных, использования восьмеричных и шестнадцатеричных систем счисления для оптимизации алгоритмов обработки данных. Написали программу, реализующую алгоритм обработки числовых данных и оформили его на языке С++. Иными словами — все поставленные задачи были выполнены.

Изучение структур данных и их числовых представлений, алгоритмов работы с ними позволяет научиться создавать производительное ПО.

**Список литературы**

1. Единая система программной документации (ЕСПД) ГОСТ 19.701-90.

2. ГОСТ Р30 – 2003.

3. Интернет-ресурс: <http://window.edu.ru>

4. Интернет-ресурс: <https://ru.wikipedia.org>

5. Интернет-ресурс: http://www.zaurtl.ru/UkVT/UKVT6.html

6. Ахо. А. Структуры данных и алгоритмы. Вильямс., 2001.

7. Герберт Шилдт. С++. Базовый курс. Москва. Вильямс, 2010.

8. Сыромятников В.П. «Программирование», лекции, МТУ МИРЭА, 2016/2017.

**Приложение**

|  |  |  |
| --- | --- | --- |
| **Тип** | **байт** | **Диапазон принимаемых значений** |
| целочисленный (логический) тип данных | | |
| bool | 1 | 0   /   255 |
| целочисленный (символьный) тип данных | | |
| char | 1 | 0   /   255 |
| целочисленные типы данных | | |
| short int | 2 | -32 768    /    32 767 |
| unsigned short int | 2 | 0  /  65 535 |
| int | 4 | -2 147 483 648   /   2 147 483 647 |
| unsigned int | 4 | 0     /     4 294 967 295 |
| long int | 4 | -2 147 483 648    /    2 147 483 647 |
| unsigned long int | 4 | 0     /     4 294 967 295 |
| типы данных с плавающей точкой | | |
| float | 4 | -2 147 483 648.0  / 2 147 483 647.0 |
| long float | 8 | -9 223 372 036 854 775 808 .0   /   9 223 372 036 854 775 807.0 |
| double | 8 | -9 223 372 036 854 775 808 .0   /   9 223 372 036 854 775 807.0 |

Таблица №1

Таблица №2

|  |  |  |  |
| --- | --- | --- | --- |
| n | log n | nlog n | n*2* |
| 1 | 0 | 0 | 1 |
| 16 | 4 | 64 | 256 |
| 256 | 8 | 2 048 | 65 536 |
| 4 096 | 12 | 49 152 | 16 777 216 |
| 65 536 | 16 | 1 048 565 | 4 294 967 296 |
| 1 048 476 | 20 | 20 969 520 | 1 099 301 922 576 |
| 16 775 616 | 24 | 402 614 784 | 281 421 292 179 456 |

Рисунок №1 — Пример древовидной структуры

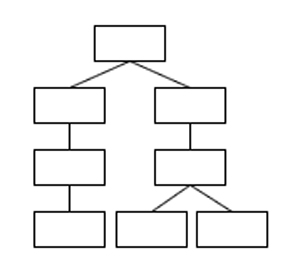


Рисунок №2 — Пример неориентированного графа

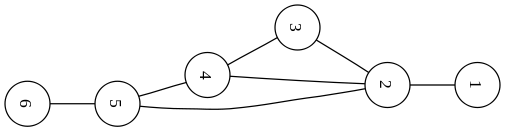
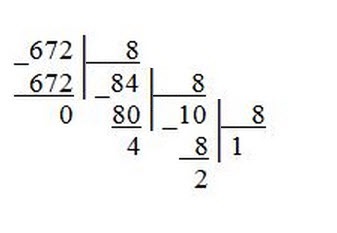


Рисунок №4 - перевод из десятичной системы счисления в восьмеричную.



67210 = 12408