



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ «Информатика и системы управления»

КАФЕДРА _____ «Теоретическая информатика и компьютерные технологии»

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент _____ Яровикова Анастасия Сергеевна
(Фамилия, Имя, Отчество)

Группа _____ ИУ9-61Б

Тип практики _____ Производственная

Название предприятия _____ Институт Программных Систем им. А.К. Айламазяна РАН

Студент _____ ИУ9-61Б
(Группа)

(Подпись, дата)

(И.О. Фамилия)

Рекомендуемая оценка _____

Руководитель практики
от предприятия

(Подпись, дата)

(И.О. Фамилия)

Руководитель практики

(Подпись, дата)

(И.О. Фамилия)

Оценка _____

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Характеристика предприятия ИПС А.К. Айламазяна РАН . .	4
2 Индивидуальное задание	5
2.1 Постановка задачи	5
2.2 Изучение предметной области	6
2.2.1 Стандартные алгоритмы перевода из одной позиционной системы в другую	7
2.2.2 Нестандартные алгоритмы Содена и Розье перевода из одной позиционной системы в другую	8
2.2.3 Обоснование новых методов перевода десятичных и восьмеричных чисел	11
2.3 Разработка и реализация	14
2.3.1 Особенности программы	14
2.3.2 Доработка алгоритма перевода десятичных натуральных чисел в восьмиричную систему, показанного на примерах в книге «Системы счисления и их применение» Гашкова С. Б.	15
2.3.3 Тестирование	17
ЗАКЛЮЧЕНИЕ	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	20
ПРИЛОЖЕНИЕ А	21

ВВЕДЕНИЕ

Целью производственной практики на предприятии ИПС А.К. Айламазяна РАН является закрепление знаний по изучаемым дисциплинам и получение студентами практических навыков в период пребывания на предприятии, в связи с чем можно выделить следующие задачи данной практики:

1. Ознакомление с предприятием, программными продуктами предприятия, основными направлениями исследований, проводимых на предприятии, кругом задач, решаемыми специалистами предприятия, планируемыми эффектами проводимых исследований и их значением для отделов разработки.
2. Изучение новых дисциплин, связанных с предметной областью задания.
3. Применение навыков программирования и разработки алгоритмов.
4. Получение опыта работы с руководителем и полноценным техническим заданием.
5. Приобретение навыка научно-исследовательской деятельности в процессе выполнения поставленной руководителем исследовательской задачи.

1 Характеристика предприятия ИПС А.К. Айламазяна РАН

Производственная практика проходила в институте программных систем Российской академии наук [1]. Институт программных систем был создан в апреле 1984 года как Филиал Института проблем кибернетики АН СССР по решению Правительства СССР, направленному на развитие вычислительной техники и информатики в стране. Руководителем ФИПК АН СССР был назначен д.т.н., профессор Альфред Карлович Айламазян. В 1986 году Филиал Института проблем кибернетики был преобразован в Институт программных систем АН СССР, а в 2008 году институту было присвоено имя его первого директора профессора А.К. Айламазяна. С момента создания основными научными направлениями деятельности института являлись:

- высокопроизводительные вычисления
- программные системы для параллельных архитектур
- автоматизация программирования
- телекоммуникационные системы и медицинская информатика

Сегодня Институт программных систем имени А.К. Айламазяна РАН объединяет пять исследовательских центров и является динамично развивающимся коллективом, работающим в Отделении нанотехнологий и информационных технологий РАН.

Исследовательские центры ИПС РАН:

- Исследовательский центр мультипроцессорных систем (ИЦМС)
- Исследовательский центр медицинской информатики (ИЦМИ Интерин)
- Исследовательский центр искусственного интеллекта (ИЦИИ)
- Исследовательский центр процессов управления (ИЦПУ)
- Исследовательский центр системного анализа (ИЦСА)

2 Индивидуальное задание

2.1 Постановка задачи

Функциональные языки программирования в настоящее время пользуются вполне заслуженной популярностью. Один из старейших членов этого семейства (впервые реализован в 1968 году в России, где используется и поныне), РЕФАЛ соединяет в себе математическую простоту с практической ориентацией на написание больших и сложных программ [2].

РЕФАЛ (РЕкурсивных Функций АЛгоритмический язык) - это функциональный язык программирования, ориентированный на обработку символьных строк (например, алгебраические выкладки), перевод с одного языка (искусственного или естественного) на другой, решение проблем, связанных с искусственным интеллектом. Данный язык программирования имеет различные диалекты, одним из которых является Рефал-5, проект которого поддерживается и оптимизируется в настоящее время.

Целью моей работы стала реализация на Си стандартных и нестандартных, основанных на схеме Горнера, алгоритмов перевода натуральных чисел из одной позиционной системы счисления в другую, а также последующее сравнение по скорости данных алгоритмов. Стоит отметить, что основная задача, поставленная передо мной на практике, имела экспериментальный характер, и от её результатов будут зависеть решения о развитии и оптимизации арифметики в системе Рефал-5.

Индивидуальное задание состояло из следующих задач:

1. Реализация нового алгоритма перевода из 10-й системы в 2^{64} -ричную без деления, используя алгоритм, предложенный математиком Ш. Розье [3].
2. Реализация стандартного алгоритма перевода из 10-й системы в 2^{64} -ричную.
3. Систематизация результатов тестирования в сравнительные таблицы времен двух разных алгоритмов перевода из 10-й системы в систему счисления по основанию 2^{64} .
4. Реализация нового алгоритма перевода из системы счисления по основанию 2^{64} в 10-ю без деления, используя алгоритм, предложенный Соденом [3].

5. Реализация стандартного алгоритма перевода из системы счисления по основанию 2^{64} в 10-ю.
6. Систематизация результатов тестирования в сравнительные таблицы времен двух разных алгоритмов перевода из системы счисления по основанию 2^{64} в 10-ю.
7. Участие в тестировании инсталляции новой версии реализации Рефала-5.

2.2 Изучение предметной области

Перед выполнением практической части работы, было необходимо провести изучение теоретической базы по данной задаче а именно вспомнить стандартные алгоритмы перевода чисел из одной системы счисления в другую и изучить алгоритмы перевода Содена и Розье, основанные на классической схеме Горнера.

Согласно условиям поставленной задачи, на вход алгоритму поступает число в десятичной системе. Далее необходимо произвести перевод в систему счисления по основанию 2^{64} . Ввиду того, что основание целевой системы счисления является достаточно большим числом, и при этом степенью двойки, было решено разбить алгоритм перевода десятичного числа в целевую систему счисления на этапы:

1. Перевод из 10-й системы в 8-ю систему счисления.
2. Перевод из 8-й системы счисления в систему по основанию 2^{64} .

Отметим, что последний этап упрощается в связи с тем, что основания обеих систем счисления являются степенями двойки.

В свою очередь, обратный перевод чисел из системы по основанию 2^{64} в привычную человеку десятичную систему счисления выполнен по аналогичной схеме:

1. Перевод из системы по основанию 2^{64} в 8-ю систему счисления.
2. Перевод из 8-й системы счисления в 10-ю систему .

При таком распределении задачи на этапы тщательному рассмотрению подлежит вопрос перевода из десятичной системы счисления в восьмиричную, и обратно: из восьмиричной системы счисления в десятичную.

2.2.1 Стандартные алгоритмы перевода из одной позиционной системы в другую

Как было отмечено ранее, нас будут интересовать стандартные алгоритмы перевода из десятичной системы счисления в восьмиричную и из восьмиричной системы счисления в десятичную. Первый алгоритм основывается на получении остатков от деления натуральных чисел на основание целевой системы счисления.

Алгоритм перевода натуральных десятичных чисел в восьмиричную систему счисления.

1. Последовательно выполнять деление десятичного числа и получаемых целых частных на 8, до тех пор, пока частное не станет равным 0.
2. Для получения ответа в восьмиричной системе, необходимо записать, полученные, в результате деления остатки, в обратном порядке.

Ниже представлен пример применения алгоритма для числа 1234_{10} . Результатом в данном примере является восьмиричное число 2322_8 .

Пример 1. *Стандартный метод перевода десятичного числа в восьмиричную систему счисления.*

При делении $1234 : 8$ получаем частное 154 и остаток 2. Далее рекурсивно выполняем деление до получения нулевого частного: при делении $154 : 8$ и получаем частное 19 и остаток от деления 2, при делении 19 на 8 – частное 2 и остаток 3, результат целого деления 2 на 8 дает 0. Осталось инверсивно записать последовательность полученных остатков от деления: 2322.

Ответ: $1234_{10} = 2322_8$

Второй алгоритм базируется на развернутой форме записи числа для позиционной системы счисления:

$$A_n = a_{n-1} * q^{n-1} + a_{n-2} * q^{n-2} + ... + a_0 * q^0; \quad (1)$$

где A — число, q — основание системы счисления, а n — количество разрядов числа.

Зная, что 8 — основание системы счисления, выведем формулу перевода:

$$A_8 = a_{n-1} * 8^{n-1} + a_{n-2} * 8^{n-2} + ... + a_0 * 8^0; \quad (2)$$

Пример 2. Перевод числа 7471 из восьмеричной системы в десятичную.

$$7471_8 = 7 * 8^3 + 4 * 8^2 + 7 * 8^1 + 1 * 8^0 = 7 * 512 + 4 * 64 + 7 * 8 + 1 * 1 = \\ 3584 + 256 + 56 + 1 = 3897_{10};$$

Ответ: $7471_8 = 3897_{10}$

Таким образом, **алгоритм перевода натурального восьмеричного числа в десятичную систему счисления** состоит в том, чтобы число представить в виде суммы произведений степеней основания восьмеричной системы счисления на соответствующие цифры в разрядах восьмеричного числа.

2.2.2 Нестандартные алгоритмы Содена и Розье перевода из одной позиционной системы в другую

Описания алгоритмов перевода из десятичной системы счисления в восьмиричную и из восьмиричной системы счисления в десятичную взяты нами из книг «Системы счисления и их применение» [4] и «Занимательная компьютерная арифметика: Математика и искусство счета на компьютерах и без них» [5] профессора Гашкова С. Б.

Данные алгоритмы представляют собой нестандартные методы взаимного перевода десятичных и двоичных чисел, предложенные Соденом в 1953 году и Розье в 1962 году, и основанные на так называемой схеме Горнера. При этом в данных методах используются промежуточные переводы в восьмиричную систему счисления, что как раз отвечает поставленной перед нами задаче. Поэтому методами Содена и Розье можно осуществлять и взаимный перевод десятичных и восьмеричных чисел. Ниже будут описаны алгоритмы взаимных переводов из восьмиричной системы в десятичную, и обратно.

Алгоритм перевода натуральных восьмеричных чисел в десятичную систему счисления.

Для n -значного восьмиричного числа $u = (u_n \dots u_1)_8$ выполняются $(n - 1)$ шагов для перевода его в десятичную систему.

На k -м шаге выполняем над полученной на предыдущем шаге записью в десятичной арифметике действия:

$$\overline{u_n \dots u_{n-k-1}} - 2 * \overline{u_n \dots u_{n-k}} = \overline{v_{n+1} \dots v_{n-k-1}}$$

и получаем запись $\overline{v_{n+1} \dots v_{n-k-1} . u_{n-k-2} \dots u_1}$. При этом старшие разряды могут оказаться нулевыми и в реальных вычислениях участвовать не будут. Десятичную запись числа и получаем на $(n - 1)$ -м шаге.

Пример 3. *Перевод числа 3747 из восьмеричной системы в десятичную.*

$$\begin{array}{r} 3.747 \\ - \\ 6 \\ \hline 31.47 \\ - \\ 62 \\ \hline 252.7 \\ - \\ 504 \\ \hline 2023 \end{array}$$

Ответ: $3747_8 = 2023_{10}$

В данном примере приведен процесс перевода четырехзначного восьмиричного числа в десятичную систему. Соответственно, перевод состоит из трех шагов.

На первом шаге необходимо из числа 3747, рассматривая его как десятичное, вычесть удвоенное произведение его первой цифры на 100 (на 10^{n-2}). Отметим, что с целью экономии времени хвостовые нули можно не выписывать.

На втором шаге надо из полученной разности (также рассматривая ее как десятичную) вычесть удвоенное произведение двузначного числа, образованного первой и второй цифрами разности, на 10 (на 10^{n-3}).

На третьем, в данном случае последнем, шаге вычитается удвоенное произведение трехзначного числа, образованного тремя первыми цифрами последней разности, на 1 (на 10^0 , или на 10^{n-4}).

Полученное число и будет искомым десятичным.

Видно, что при переводе используются только умножение на два и вычитание. Именно по этой причине алгоритм получается значительно проще по числу операций, чем стандартный алгоритм с делением.

Алгоритм перевода натуральных десятичных чисел в восьмеричную систему счисления.

Данный метод почти такой же, как описанный выше (только, можно сказать, «наоборот»).

Для n -значного десятичного числа $u = (u_n \dots u_1)_{10}$ выполняются $(n - 1)$ шагов для перевода его в восьмеричную систему.

На k -м шаге над полученной на предыдущем шаге записью выполняем действия (умножение и сложение) в восьмеричной арифметике:

$$\overline{u_n \dots u_{n-k-1}} + 2 * \overline{u_n \dots u_{n-k}} = \overline{v_{n+1} \dots v_{n-k-1}}$$

и получаем запись $\overline{v_{n+1} \dots v_{n-k-1} . u_{n-k-2} \dots u_1}$. При этом старшие $(n + 1)$ -е разряды могут оказаться нулевыми и в реальных вычислениях участвовать не будут. Восьмеричную запись числа u получаем на $(n - 1)$ -м шаге.

Пример 4. Перевод числа 2023 из десятичной системы в восьмеричную.

$$\begin{array}{r} 2.023 \\ + \quad 4 \\ \hline 24.23 \\ + \quad 50 \\ \hline 312.3 \\ + \quad 624 \\ \hline 3747 \end{array}$$

Ответ: $2023_{10} = 3747_8$

В данном случае разобран процесс перевода четырехзначного десятичного числа в восьмеричную систему, а сам перевод состоит из трех шагов. Основным нюансом данного алгоритма является проведение всех арифметических операций (умножение и сложение) в восьмеричной системе.

На первом шаге к числу 2023 прибавляется удвоенное произведение его первой цифры на 100 (на 10^{n-2}). С целью экономии времени хвостовые нули могут быть опущены (не выписаны).

На втором шаге надо к полученной записи прибавляется удвоенное произведение двузначного числа, образованного первой и второй цифрами полученной на предыдущем шаге записи, на 10 (на 10^{n-3}).

На третьем, в данном случае последнем, шаге складываются полученное на предыдущем шаге число и удвоенное произведение трехзначного числа, образованного тремя первыми цифрами последней полученной записи, на 1 (на 10^0 , или на 10^{n-4}).

Итоговое число является искомым восьмиричным.

Отметим, что при переводе используются только умножение на два и сложение. Из-за отсутствия операции деления данный алгоритм также получается проще по числу операций, чем стандартный алгоритм перевода десятичных чисел в восьмиричную систему счисления.

2.2.3 Обоснование новых методов перевода десятичных и восьмиричных чисел

Как было обозначено ранее, методы Содена и Розье основаны на так называемой схеме Горнера. Опишем как данный алгоритм устроен и связан с методами перевода восьмиричных и десятичных чисел, на примерах обоснуем справедливость данных методов.

Схема Горнера (или правило Горнера, метод Горнера, метод Руффини-Горнера) — это алгоритм вычисления значения многочлена, записанного в виде суммы мономов (одночленов), при заданном значении переменной. Также данный алгоритм применяется для вычисления частного и остатка от деления многочлена $p(x)$ на $x - a$.

Итак, пусть дан произвольный многочлен в восьмиричной арифметике:

$$p(x) = u_n * x^n + \dots + u_1 * x + u_0.$$

Деление этого многочлена на $x - a$ — это представление его в виде

$$p(x) = (x - a) * h(x) + r, \quad h(x) = v_{n-1} * x^{n-1} + \dots + v_1 * x + v_0.$$

Можно непосредственно проверить, что коэффициенты частного $h(x)$ можно найти по формулам

$$v_{n-1} = u_n, v_{n-2} = u_{n-1} + a * v_{n-1}, \dots, v_0 = u_1 + a * v_1,$$

а остаток r можно вычислить по формулам

$$\begin{aligned} r &= u_0 + a * v_0 = u_0 + a * (u_1 + a * v_1) = u_0 + a * (u_1 + a * (u_2 + a * v_2)) = \dots \\ &= u_0 + a * (u_1 + a * (\dots(u_{n-1} + a * u_n)\dots)) = u_n * a^n + \dots + u_1 * a + u_0 = p(a). \end{aligned}$$

Описанный выше метод вычисления и называется схемой Горнера.

Теперь по порядку опишем взаимосвязь рассмотренных алгоритмов перевода Содена и Розье со схемой Горнера.

Напомним, что в алгоритме перевода натуральных восьмеричных чисел в десятичную систему счисления, предложенном Соденом, на каждом шаге полученное число рассматривается как десятичное. Тогда, воспользуясь схемой Горнера, представим число 3747 из примера 3 следующим образом:

$$3747_{10} = 3 * 10^3 + 7 * 10^2 + 4 * 10 + 7 = ((3 * 10 + 7) * 10 + 4) * 10 + 7.$$

Вспомним, что по стандартному алгоритму восьмеричное число переводится в десятичное в помощью развернутой формы записи числа, а именно:

$$3747_8 = 3 * 8^3 + 7 * 8^2 + 4 * 8 + 7. \quad (3)$$

А теперь, воспользовавшись схемой Горнера и тем фактом, что $8 = 10 - 2$ перепишем (3):

$$\begin{aligned} 3747_8 &= 3 * (10 - 2)^3 + 7 * (10 - 2)^2 + 4 * (10 - 2) + 7 = \\ &= ((3 * (10 - 2) + 7) * (10 - 2) + 4) * (10 - 2) + 7. \end{aligned}$$

Видно, откуда в алгоритм Содена появилось вычитание удвоенного числа. Если рассматривать каждую скобку по отдельности, то можно проследить каждый шаг алгоритма, описанного ранее. Например, первая скобка $(3 * (10 - 2) + 7)$ соответствует первому шагу алгоритма, когда из числа 3747 вычитается шестерка, т.е. удвоенная первая цифра 3. Действительно, $(3 * (10 - 2) + 7) = 3 * 10 - 6 + 7 = 31$.

В результате первого шага получаем число 31, которое является двузначным числом из той самой разности, полученной после первого шага алгоритма Содена, и которое на втором шаге будет удвоено и вычтено из полученной разности.

Аналогично рассматривается оставшаяся запись из уравнения (3). В итоге, действительно, получаем верное десятичное число:

$$\begin{aligned} 3747_8 &= ((3 * (10 - 2) + 7) * (10 - 2) + 4) * (10 - 2) + 7 = \\ &= 3 * 8^3 + 7 * 8^2 + 4 * 8 + 7 = 1536 + 448 + 32 + 7 = 2023_{10}. \end{aligned}$$

Наконец перейдем к алгоритму Розье. Так же воспользуемся схемой Горнера и представим число 2023 из примера 4 следующим образом:

$$2023_{10} = 2 * 10^3 + 0 * 10^2 + 2 * 10 + 2 = ((2 * 10 + 0) * 10 + 2) * 10 + 3.$$

Теперь представляем 10 в виде суммы 8 и 2:

$$2023_{10} = ((2 * (8 + 2) + 0) * (8 + 2) + 2) * (8 + 2) + 3.$$

Стало видно откуда появилось сложение удвоенного числа в алгоритме Розье.

Напомним, что в алгоритме Розье перевода десятичного числа в восьмиричную систему арифметика выполняется в восьмиричной системе. Вычисления проводятся именно в той системе счисления, в которой мы хотим получить результат. С учетом этого нюанса рассматриваем каждую скобку полученного уравнения по отдельности:

1. $(2 * (8 + 2) + 0) = 2 * 8 + 2 * 2 + 0 = 20 + 4 + 0 = 24$
2. $((2 * (8 + 2) + 0) * (8 + 2) + 2) = 24 * (8 + 2) + 2 = 240 + 50 + 2 = 312$
3. $((2 * (8 + 2) + 0) * (8 + 2) + 2) * (8 + 2) + 3 = 312 * (8 + 2) + 3 = 3120 + 629 + 3 = 3747$

Наглядно видно, что в результате первого шага получаем число 24, которое является двузначным числом из той суммы, полученной после первого шага алгоритма Розье, и которое на втором шаге будет удвоено и прибавлено к полученной сумме. Результатом второго шага является число 312, а в результате третьего, последнего, шага получено число 3747, являющееся восьмиричной записью десятичного числа 2023.

Листинг 1: Структуры, близкие структурам реализации интерпретатора Рефала-5

```
1  /** Refal structures. */
2  typedef struct link {
3      char ptype; /* type of the link */
4      union {
5          struct link *b; /* bracket: ptr to the pair */
6          char *f; /* function or compound symbol: ptr to label. */
7          char c; /* symbol: actual value. */
8          /* unsigned long u; /* unicode symbol */
9          unsigned int u; /* unicode symbol */
10         unsigned long n; /* macro-digit */
11         unsigned short us_1, us_2;
12     } pair;
13     struct link *prec; /* ptr to preceding link */
14     struct link *foll; /* ptr to following link */
15 } LINK;
```

2.3 Разработка и реализация

Для достижения поставленных задач был реализован программный продукт на языке С. В результате была создана программа, которая генерирует числовые примеры заданной длины и запускает алгоритмы перевода в системы счисления.

2.3.1 Особенности программы

Одной из особенностей программной реализации является проблема переполнения памяти при выполнении арифметических операций с большими числами. Числа в системе 2^{64} необходимо хранить в структурах, близких структурам реализации интерпретатора Рефала-5 (см. Листинг 1). К сожалению, число 2^{64} достаточно большое, поэтому при произведении и сложении больших чисел возможно переполнение памяти, с которым необходимо бороться. Для решения проблемы было решено воспользоваться равенством $2^{64} = (2^{32})^2$.

Данное равенство говорит о том, что цифры числа N в системе счисления по основанию 2^{64} строятся следующим образом: начиная с младших разрядов, цифра числа N по основанию 2^{64} есть последовательность двух цифр этого же числа по основанию 2^{32} . В таком случае, нечетные цифры по основанию 2^{32} укладываются в младшую половину цифры по основанию 2^{64} , а четные – в старшую половину.

При использовании такой техники было необходимо реализовать взаимный перевод восьмиричных и двоичных чисел, двоичных чисел и чисел по основанию 2^{32} , а также заполнение структур, близких структурам реализации интерпретатора Рефала-5 в соответствии описанным выше методом.

Другой важной особенностью программы являясь модификация алгоритма Розье перевода из десятичной системы счисления в восьмиричную.

2.3.2 Доработка алгоритма перевода десятичных натуральных чисел в восьмиричную систему, показанного на примерах в книге «Системы счисления и их применение» Гашкова С. Б.

Вопрос о модификации алгоритма был рассмотрен в процессе тестирования первой версии программы. Было замечено, что в новом алгоритме в книгах [4] и [5], которые были использованы в качестве источника, представлены примеры только для тех чисел, у которых в старшем разряде стоит цифра от 1 до 4. Таким образом, на первом шаге алгоритма при умножении на два в результате не получается двузначное число, и, соответственно, на каждом шаге алгоритма не возникает ситуации, при которой число-результат увеличивалось бы в длине. Однако с числами, начинающимися на цифру от 5 до 9 описанный в книгах алгоритм давал сбой.

Решение проблемы было выявлено в процессе экспериментов с различными входными данными. При этом стабильно прослеживалось то, что на каждом шаге результат умножения на 2 необходимо записывать для последующего сложения так, что его последняя цифра стоит сразу за точкой, отделяющей число, которое мы умножали. То есть удвоенное число сдвигается относительно цифр исходного числа, на которое мы умножали, вправо ровно на одну цифру.

В итоге, алгоритм был модифицирован следующим образом:

На k -м шаге над полученной на предыдущем шаге записью выполняем действия (умножение и сложение) в восьмиричной арифметике и при этом следим за результатом умножения на два:

1. если получается число, длина которого не изменилась (например, у 2 длина единица, и $2 * 2 = 4_8$ также имеет длину 1), тогда результат умножения

записывается в столбик под исходное число, начиная со второго после старшего разряда исходного.

2. если получается число, длина которого больше, изначального (например, у 6 длина единица, и $6 * 2 = 14_8$ имеет длину 2), тогда результат умножения записывается в столбик под исходное число, начиная со старшего разряда исходного.

Более того, на каждом шаге необходимо следить за значением суммы исходного числа с удвоенным:

1. если после сложения получается число, длина которого не изменилась (например, после первого шага для числа 6297, получаем 7697 - длина 4, так же как и у 6297), тогда результат сложения записывается записывается в столбик под исходное число, как обычно.
2. если после сложения получается число, длина которого больше, чем длина изначального (например, после второго шага для числа 6297 получаем 11657 - длина 5), то происходит сдвиг самого числа-результата влево, тогда точка «как бы» передвигается не на одну позицию вправо, а на две позиции.

Данные рассуждения проиллюстрированы на примере ниже:

Пример 5. *Перевод числа 6297 из десятичной системы в восьмеричную.*

$$\begin{array}{r}
 6.297 \\
 + \quad 14 \\
 \hline
 76.97 \\
 + \quad 174 \\
 \hline
 1165.7 \\
 + \quad 2353 \\
 \hline
 14231
 \end{array}$$

Ответ: $6297_{10} = 14231_8$

Посняим данный пример, используя схему Горнера.

$$\begin{aligned}
 6297_{10} &= ((6 * 10 + 2) * 10 + 9) * 10 + 7 = \\
 &= ((6 * (8 + 2) + 2) * (8 + 2) + 9) * (8 + 2) + 7.
 \end{aligned}$$

Как и в предыдущих примерах, рассмотрим каждую скобку отдельно:

$$1. (6 * (8 + 2) + 2) = 6 * 8 + 6 * 2 + 2 = 60 + 14 + 2 = 62 + 14 = 76_8$$

Видим, что результат умножения 6 на 2 является двузначным, поскольку мы работаем в восьмиричной системе и $6 * 2 = 14_8$, соответственно возникает перенос единицы в старший разряд.

$$2. ((6 * (8 + 2) + 2) * (8 + 2) + 9) = 76 * (8 + 2) + 9 = 76 * 8 + 76 * 2 + 9 = 760 + 174 + 9 = 769 + 174 = 1165_8$$

На втором шаге также результат умножения 76 на 2 имеет большую длину, является трехзначным, поскольку мы в восьмиричной системе и $76 * 2 = 174_8$, соответственно возникает перенос единицы в старший, третий, разряд.

$$3. (((6 * (8 + 2) + 2) * (8 + 2) + 9) * (8 + 2) + 7) = 1165 * (8 + 2) + 7 = 1165 * 8 + 1165 * 2 + 7 = 11650 + 2352 + 7 = 11657 + 2352 = 14231_8$$

Мы видим, что 11657, полученное после второго шага алгоритма, отличается по длине от результата 7697, полученного на первом шаге. Именно в таком случае и происходит так называемый сдвиг влево, он обоснован простым переносом в старший разряд, который возникает в связи с использованием восьмиричной арифметики.

В конечном итоге, получен верный результат 14231_8 .

С учетом всех описанных выше особенностей программа была скорректирована. Основные функции проекта, необходимые для перевода чисел в различные системы счисления, приведены в листингах 2-8 в приложении А.

2.3.3 Тестирование

Целью тестирования являлось желание получить подтверждение или опровержение гипотезы о том, что нестандартные алгоритмы, предложенные Соденом и Розье, взаимного перевода из систем счисления являются практически более эффективны по времени, по сравнению со стандартными алгоритмами.

Реализованная программа была протестирована на входных десятичных числах разной длины. Результаты были систематизированы в сравнительные таблицы 1 и 2.

Таблица 1 — Среднее время работы стандартного и нового алгоритмов перевода из 10-й системы в систему счисления по основанию 2^{64} .

Длина входного десятичного числа	Время выполнения стандартного алгоритма, с.	Время выполнения нового алгоритма, с.
3	0.00006	0.0005
4	0.00007	0.00006
5	0.00009	0.00008
10	0.000193	0.000172
20	0.000454	0.000380
30	0.00052	0.00050
40	0.000962	0.000849

Таблица 2 — Среднее время работы стандартного и нового алгоритмов перевода из системы по основанию 2^{64} в 10-ю систему счисления.

Длина входного десятичного числа	Время выполнения стандартного алгоритма, с.	Время выполнения нового алгоритма, с.
3	0.000090	0.000021
4	0.000240	0.000030
5	0.000386	0.000050
10	0.000494	0.000113
20	0.000541	0.000354
30	0.001841	0.000614
40	0.004073	0.000954

ЗАКЛЮЧЕНИЕ

В результате практики поставленные задачи были успешно выполнены. Были разработаны и реализованы функции, выполняющие взаимные переводы десятичных и восьмиричных чисел, восьмиричных и двоичных чисел, двоичных чисел и чисел по основанию 2^{64} , а также функция, сравнивающая практические временные сложности в среднем алгоритмов, основанных на схеме Горнера, и стандартных алгоритмов перевода натуральных чисел из десятичной системы счисления в систему счисления по основанию 2^{64} и обратно. К тому же была реализована функция, выполняющая автоматическую генерацию тестов, были составлены таблицы, отражающие результаты тестирования в контексте поставленных задач, а также было принято участие в тестировании инсталляции новой версии реализации Рефала-5.

В процессе выполнения поставленной задачи были изучены и реализованы алгоритмы перевода из одной позиционной системы счисления в другую с использованием схемы Горнера. В ходе разработки был получен опыт работы с языком программирования Си. Результаты работы подтвердили гипотезу о том, что практические временные сложности в среднем алгоритмов, основанных на схеме Горнера и не использующих операцию деления, меньше соответствующих сложностей стандартных алгоритмов, т.е. стандартные алгоритмы менее оптимальны.

В дальнейшем видится совершенствование проекта, продолжение тестирования на случайных множествах тестов натуральных чисел и внедрение описанных алгоритмов в реализацию арифметики системы программирования Рефал-5, которая работает с натуральными числами неограниченной длины.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт ИПС А.К. Айламазяна РАН. — URL: <http://www.psi-ras.ru/>.
2. REFAL-5 programming guide & reference manual [Электронный ресурс]. — URL: <http://refal.botik.ru/book/html/>.
3. Златопольский Д. М. Нестандартные методы перевода чисел из одной системы счисления в другую // Информатика в школе. — 2020. — № 1. — С. 28—30. — DOI: 10.32517/2221-1993-2020-19-9-28-30.
4. Гашков С. Б. Системы счисления и их применение. — Москва : Московский центр непрерывного математического образования, 2012.
5. Гашков С. Б. Занимательная компьютерная арифметика: Математика и искусство счета на компьютерах и без них. — Москва : Едиториал УРСС, 2017.

ПРИЛОЖЕНИЕ А

Листинг 2: Реализация алгоритма Розье. Перевод десятичного числа в восьмиричную систему. Умножение на 2 в восьмиричной системе.

```
1 char* mul_x_2_octal(char* code, long n) {
2     char c;
3     int i;
4     char* code1;
5     code1 = malloc(n + 1);
6     long digit, dec, res;
7     long carry = 0;
8     code1[0] = '0';
9     for (i = n - 1; i >= 0; i--) {
10         c = code[i];
11         char *pChar = &c;
12         digit = (long)atoi(pChar);
13         digit = digit * 2;
14         dec = digit / 8;
15         res = digit%8 + carry;
16         if (res > 9) {
17             dec = res / 8;
18             code1[i + 1] = (res%10) + '0';
19         } else {
20             code1[i + 1] = res + '0';
21         }
22         if (dec > 0) {
23             carry = dec;
24         } else {
25             carry = 0;
26         }
27     }
28     if (i == -1 && carry > 0) {
29         code1[0] = (long)(code1[0] - '0') + carry + '0';
30     }
31     code1[n+1] = '\\0';
32     return code1;
33 }
```

Листинг 3: Реализация алгоритма Розье. Перевод десятичного числа в восьмиричную систему. Сложение в восьмиричной системе.

```
1 char* my_octal_add(char* code1, char* code2, long n) {
2     char *code;
3     char c1, c2;
4     int digit, digit1, digit2, sd, dec, res, carry = 0;
5     code = malloc(n);
6     unsigned long long i, len1 = strlen(code1), len2 = strlen(code2);
7     c1 = code1[0];
8     sd = c1 - '0';
9     code[0] = sd + '0';
10    if (len1 == len2) {
11        for (i = len1 - 1; i > 0; i--) {
12            c1 = code1[i];
13            c2 = code2[i];
14            digit1 = c1 - '0';
15            digit2 = c2 - '0';
16            digit = digit1 + digit2;
17            dec = digit / 8;
18            res = digit % 8 + carry;
19            if (res > 7) {
20                dec = res / 8;
21                code[i] = (res % 8) + '0';
22            } else {
23                code[i] = res + '0';
24            }
25            if (dec > 0) {
26                carry = dec;
27            } else {
28                carry = 0;
29            }
30            if (dec > 0 && i == 1) {
31                code[0] = sd + carry + '0';
32            }
33        }
34        if (sd != 0) {
35            code[0] = sd + (code2[0] - '0') + carry + '0';
36        }
37        code[n] = '\0';
```

```

38         return code;
39     }
40
41     for (i = n - 1; i > len2; i--) {
42         code[i] = code1[i];
43     }
44     for (i = len2; i > 0; i--) {
45         c1 = code1[i];
46         c2 = code2[i - 1];
47         digit1 = c1 - '0';
48         digit2 = c2 - '0';
49         digit = digit1 + digit2;
50         dec = digit / 8;
51         res = digit % 8 + carry;
52         if (res > 7) {
53             dec = res / 8;
54             code[i] = (res % 8) + '0';
55         } else {
56             code[i] = res + '0';
57         }
58         if (dec > 0) {
59             carry = dec;
60         } else {
61             carry = 0;
62         }
63         if (dec > 0 && i == 1) {
64             code[0] = sd + carry + '0';
65         }
66     }
67     code[n] = '\0';
68     return code;
69 }

```

Листинг 4: Реализация алгоритма Содена. Перевод восьмиричного числа в десятичную систему. Умножение на 2 в десятичной системе.

```
1 char* mul_x_2(char* code, long n) {
2     char c;
3     int i;
4     char* code1;
5     code1 = malloc(n + 1);
6     long digit, dec, res;
7     long carry = 0;
8     code1[0] = '0';
9     for (i = n - 1; i >= 0; i--) {
10         c = code[i];
11         char *pChar = &c;
12         digit = (long)atoi(pChar);
13         digit = digit * 2;
14         dec = digit / 10;
15         res = digit%10 + carry;
16         if (res > 9) {
17             dec = res / 10;
18             code1[i + 1] = (res%10) + '0';
19         } else {
20             code1[i + 1] = res + '0';
21         }
22         if (dec > 0) {
23             carry = dec;
24         } else {
25             carry = 0;
26         }
27     }
28     if (i == -1 && carry > 0) {
29         code1[0] = (long)(code1[0] - '0') + carry + '0';
30     }
31     code1[n+1] = '\\0';
32     return code1;
33 }
```


Листинг 5: Реализация алгоритма Содена. Перевод восьмиричного числа в десятичную систему. Вычитание в десятичной системе.

```
1 char* my_decimal_sub(char* code1, char* code2, long n) {
2     char *code;
3     int digit, digit1, digit2, sd, dec, res, carry = 0;
4     code = malloc(n + 1);
5     sd = code1[0] - '0';
6     code[0] = sd + '0';
7     for (int i = n - 1; i > 0; i--) {
8         digit1 = code1[i] - '0';
9         digit2 = code2[i] - '0';
10        digit = digit1 - digit2 - carry;
11        if (digit < 0) {
12            carry = 1;
13            code[i] = (digit + 10) + '0';
14        } else {
15            carry = 0;
16            code[i] = digit + '0';
17        }
18        if (carry != 0 && i == 1) {
19            code[0] = sd - carry + '0';
20        }
21    }
22    code[n] = '\0';
23    return code;
24 }
```

Листинг 6: Реализация стандартного алгоритма. Перевод из десятичной системы в восьмиричную систему.

```
1 char* convert_from_decimal_to_octal(char* code, long n) {
2     bool new_step = false;
3     char* result;
4     result = (char*)malloc((n + 2) * sizeof(char));
5     int j = 0, d;
6     char* code2;
7     char* first_two_digits;
8     first_two_digits = malloc(3);
9     long digits, rem, quot, len = n;
10    int first_digit = code[0] - '0', flag = 0;
11    if (first_digit < 8) {
12        slice(code, first_two_digits, 0, 2);
13        first_two_digits[2] = '\0';
14        digits = atol(first_two_digits);
15        quot = digits / 8;
16        rem = digits - quot * 8;
17        flag = 1;
18    } else {
19        quot = first_digit / 8;
20        rem = first_digit - quot * 8;
21    }
22    while( len != 1) {
23        if (new_step) {
24            new_step = false;
25            first_digit = code[0] - '0',
26            flag = 0;
27            if (first_digit < 8) {
28                slice(code, first_two_digits, 0, 2);
29                first_two_digits[2] = '\0';
30                digits = atol(first_two_digits);
31                quot = digits / 8;
32                rem = digits - quot * 8;
33                flag = 1;
34                free(code2);
35            } else {
36                quot = first_digit / 8;
37                rem = first_digit - quot * 8;
```

```

38         flag = 0;
39     }
40 }
41 int k = 0;
42 code2 = malloc(len + 1);
43 code2[k++] = (int)quot + '0';
44 for (int i = 1; i < len; i++) {
45     if (flag == 1) {
46         i++;
47         if (i == len) {
48             break;
49         }
50     }
51     d = code[i] - '0';
52     flag = 0;
53     if (rem != 0) {
54         digits = rem * 10 + d;
55         quot = digits / 8;
56         rem = digits - quot * 8;
57     } else {
58         if (d < 8 && i != len - 1) {
59             code2[k++] = '0';
60             slice(code, first_two_digits, i, i + 2);
61             first_two_digits[2] = '\0';
62             digits = atol(first_two_digits);
63             quot = digits / 8;
64             rem = digits - quot * 8;
65             i++;
66         } else {
67             quot = d / 8;
68             rem = d - quot * 8;
69         }
70     }
71     code2[k++] = (int)quot + '0';
72 }
73 result[j++] = (int)rem + '0';
74 new_step = true;
75 code2[k] = '\0';
76 memcpy(code, code2, k);
77 len = k;

```

```

78
79     }
80     free(first_two_digits);
81     if (len == 1) {
82         d = code[0] - '0';
83         while (d != 0) {
84             if (d < 8) {
85                 result[j++] = d + '0';
86             } else {
87                 quot = d / 8;
88                 rem = d % 8;
89                 result[j++] = (int)rem + '0';
90             }
91             d = d/8;
92         }
93     }
94     result[j++] = '\0';
95     free(code2);
96     return result;
97 }

```

Листинг 7: Перевод из двоичной системы в систему по основанию 2^{32} .

```
1 unsigned long convert_to_2_in_32_system(char* code) {
2     unsigned long res = 0;
3     unsigned long cur;
4     int deg = 0;
5     int len = strlen(code);
6     for (int i = len - 1; i >= 0; i--) {
7         cur = code[i] - '0';
8         res += cur * pow(2, deg);
9         ++deg;
10    }
11    return res;
12 }
```

Листинг 8: Заполнение структуры Рефала LINK.

```
1 LINK* start_number = (LINK*)malloc(sizeof(LINK));
2 LINK* end_number;
3 LINK* prev_number;
4 prev_number = NULL;
5 bool flag_chetn = false;
6 unsigned long long old_len, len = strlen(binary);
7 while (len > 0) {
8     LINK* number;
9     number = (LINK*)malloc(sizeof(LINK));
10    if (prev_number != NULL) {
11        NEXT(prev_number) = number;
12    }
13    TYPE(number) = 'd';
14    PREV(number) = prev_number;
15    char* str = malloc(33 * sizeof(char));
16    old_len = len;
17    if (len >= 32) {
18        slice(binary, str, len-32, len);
19        len -= 32;
20    } else {
21        slice(binary, str, 0, len);
22        len -= len;
23    }
24 }
```

```

23     }
24
25     PAIR(number).n = convert_to_2_in_32_system(str);
26     if (old_len == strlen(binary)) {
27         end_number = number;
28     }
29     free(str);
30     prev_number = number;
31     flag_chetn = !flag_chetn;
32 }
33
34 if (len == 0) {
35     if (flag_chetn) {
36         LINK* number = (LINK*)malloc(sizeof(LINK));
37         if (prev_number != NULL) {
38             NEXT(prev_number) = number;
39         }
40         TYPE(number) = 'd';
41         PREV(number) = prev_number;
42         PAIR(number).n = 0;
43         start_number = number;
44     } else {
45         start_number = prev_number;
46     }
47 }

```

Листинг 9: Печать (в файл или стандартный поток вывода) чисел в системе счисления по основанию 2^{64} .

```
1 void print_Link_64(LINK* l, FILE* output) {
2     if (output != NULL) {
3         if (l != NULL && PREV(l) != NULL) {
4             unsigned long senior = l->pair.n;
5             unsigned long junior = l->prec->pair.n;
6             char* macrodigit = malloc(40);
7             char* jun = malloc(20);
8             sprintf(macrodigit, "%ld", senior);
9             sprintf(jun, "%ld", junior);
10            strcat(macrodigit, jun);
11            fputs(macrodigit, output);
12            print_Link_64(l->prec->prec, output);
13        }
14    } else {
15        if (l != NULL && PREV(l) != NULL) {
16            unsigned long senior = l->pair.n;
17            unsigned long junior = l->prec->pair.n;
18            char* macrodigit = malloc(40);
19            char* jun = malloc(20);
20            sprintf(macrodigit, "%ld", senior);
21            sprintf(jun, "%ld", junior);
22            strcat(macrodigit, jun);
23            printf("%s", macrodigit);
24            print_Link_64(l->prec->prec, NULL);
25        }
26    }
27 }
```