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

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, основыми 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 стенах института 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 одним из самых динамично развивающихся коллективов, работающих 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 широко 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 различрые диалекты, одним из которых является Рефал-5, проект которого поддерживается и оптимизируется в настоящее время.

Целью моей работы стала реализация на Си 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.
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.
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.

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 ознакомиться с новыми алгоритмами перевода.

Согласно условиям поставленной задачи, на вход алгоритму поступает число в десятичной системе. Далее необходимо произвести перевод в сисстему счисления по основанию 2^{64}. Ввиду того, что целевая система 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 будет выполнен по аналогичной схеме:

1. Перевод из 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 при делении на основание системы счисления.

Алгоритм перевода целых десятичных чисел в восьмеричную 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 восьмеричной системе, необходимо записать, полученные, в результате деления остатки, в обратном порядке.

На рисунке 1 представлен пример применения алгоритма для числа 1234_{10}. Результатом в данном примере является восьмиричное число 2322_8.

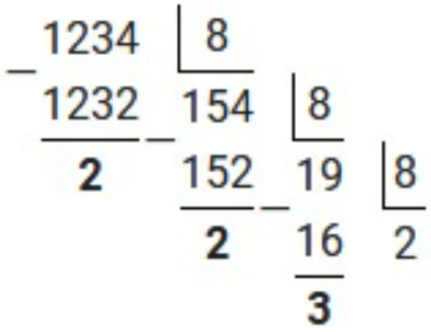

Рисунок 1 — Пример перевода десятичных чисел в восьмиричную систему счисления.

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

$$A_n = a_{n-1} * q^{n-1} + a_{n-2} * q^{n-2} + ... + a_0 * q^0\,; \tag{1}$$

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

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

$$A_8 = a_{n-1} * 8^{n-1} + a_{n-2} * 8^{n-2} + ... + a_0 * 8^0\,; \tag{2}$$

Пример 1. *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 целого 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 Новые алгоритмв перевода из одной позиционной системы в другую

Новые алгоритмы перевода из десятичной системы счисления в восьмиричную и из восьмиричной системы счисления в десятичную были описаны в книгах «Системы счисления и их применение» [3] и «Занимательная компьютерная арифметика: Математика и искусство счета на компьютерах и без них» [4] профессора Гашкова С. Б.

Описанные в книгах методы представляют собой нестандартрные 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 задачей. Поэтмоу методами Содена и Розье можно осуществлять и только взаимный перевод десятичных и восьмеричных чисел. Ниже будут описаны алгоритмы взаимных переводов из восьмиричной системы в десятичную, и обратно.

Алгоритм перевода целых восьмеричных чисел в десятичную систему счисления.

Для n-значного восьмиричного числа $u = (u_n...u_1)_8$ выполняются $(n-1)$ шагов по переводу его в десятичную систему.

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

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

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

Пример 2. *Перевод числа 3747 из восьмеричной системы в десятичную.*

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

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 конечные 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 целых десятичных чисел в восьмеричную систему счисления.

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

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

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

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

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

Пример 3. *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 восьмиричную систему, а сам перевод состоит из трех шагов. Основным нюансом данного алгоритма является проведение всех арифметических операций (умножение и сложение) в восьмиричной системе.

На первом шаге к числу 2023 прибавляется удвоенное произведение его первой цифры на 100 (на 10^{n-2}). С целью экономии времени конечные 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 отсутсвия операции деления данный алгоритм также получается проще по числу операций, чем стандартный алгоритм перевода десятичных чисел в восьмиричную систему счисления.

2.2.3 Обоснвоание новых методов перевода десятичных и восьмеричных чисел

Как было обозначено ранее, методы Содена и Розье были 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 восьмеричной арифметике:

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

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

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

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

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

а остаток r можно вычислить по формулам

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

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 целых 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 2 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 восьмиричное число переводится в десятичное в помощью развернутой формы записи числа, а именно:

$$3747_8 = 3 * 8^3 + 7 * 8^2 + 4 * 8 + 7. \tag{3}$$

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

$$3747_8 = 3 * (10 - 2)^3 + 7 * (10 - 2)^2 + 4 * (10 - 2) + 7 =$$
$$= ((3 * (10 - 2) + 7) * (10 - 2) + 4) * (10 - 2) + 7.$$

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 удвоенно и вычтено из полученной разности.

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

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

Наконец перейдем к алгоритму Розье. Так же воспользуемся схемой Горнера и представим число 2023 из примера 3 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 новом алгоритме перевода десятичного числа в восьмиричную систему арифметика выполняется в восьмиричной системе. С учетом этого нюанса рассмотриваем 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 удвоенно 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.

2.3 Разработка и реализация

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

Листинг 1: Структуры, близкие структурам Рефала

```
/*** Refal structures. ***/
typedef struct link {
    char ptype; /* type of the link */
    union {
        struct link *b; /* bracket: ptr to the pair */
    char *f; /* function or compound symbol: ptr to label. */
    char c; /* symbol: actual value. */
    /* unsigned long u; /+* unicode symbol */
    unsigned int u;  /* unicode symbol */
    unsigned long n; /*  macro-digit   */
    unsigned short us_1, us_2;
    } pair;
    struct link *prec; /* ptr to preceding link */
    struct link *foll; /* ptr to following link */
} LINK;
```

2.3.1 Особенности программы

Одной из особенностей программной реализации является проблема переполнения памяти в случае с большими числами, поскольку числа в системе 2^{64} необходимо хранить в структурах, близких структурам Рефала (см. Листинг 1), где макро-цифра имеет тип данных unsigned long. К сожалению число 2^{64} на ного больше максимально возможного числа типа unsigned long. 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, макроцифра числа N по основанию 2^{64} есть последовательность двух макро-цифр этого же числа по основанию 2^{32}. В таком случае, нечетные макроцифры по основанию 2^{32} укладываются в младшую половину макро-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 Рефала в соответствии описанным выше методом.

Другой важной особенностью программы являеся модификация нового алгоритма перевода из десятичной системы счисления в восьмиричную. Вопрос о модификации алгоритма был рассмотрен в процессе тестирования первой версии

программы. Было замечено, что в новом алгоритме в книгах [3] и [4], 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, начиньающимися 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 в восьмеричной 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 4. *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.

$$6297_{10} = ((6 * 10 + 2) * 10 + 9) * 10 + 7 =$$
$$= ((6 * (8 + 2) + 2) * (8 + 2) + 9) * (8 + 2) + 7.$$

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, поулченного 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.

2.3.2 Тестирование

Целью тестирования являлось желание получить подтверждение или опровержение гипотезы о том, что новые алгоритмы взаимного перевода из систем счисления имеют меньшее время выполнения, 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 — Время работы стандартного и нового алгоритмов перевода из 10-й системы в систему счисления по основанию 2^{64}.

Входное десятичное число	Время выполнения стандартного алгоритма, с.	Время выполнения нового алгоритма, с.
123	0.012	0.001

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

Входное десятичное число	Время выполнения стандартного алгоритма, с.	Время выполнения нового алгоритма, с.
123	0.012	0.001

ЗАКЛЮЧЕНИЕ

В результате практики поставленные задачи были успешно выполнены, и была создана программа, реализующая сравнение нового и стандартного алгоритма перевода натуральных чисел из десятичной системы счисления в систему счисления по основанию 2^{64}.

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 поставленую гипотезу об оптимальности нового алгоритма без деления.

В дальнейшем видится совершенствование проекта, его тестирование и внедрение описанных алгоритмов в Рефал-5 для оптиммальной работы с натуральными числами.

ПРИЛОЖЕНИЕ А

Листинг 2: Новый алгоритм. Перевод десятичного числа в восьмиричную систему. Умножение на 2 в восьмиричной системе

```
char* mul_x_2_octal(char* code, long n) {
    char c;
    int i;
    char* code1;
    code1 = malloc(n + 1);
    long digit, dec, res;
    long carry = 0;
    code1[0] = '0';
    for (i = n - 1; i >= 0; i--) {
        c = code[i];
        char *pChar = &c;
        digit = (long)atoi(pChar);
        digit = digit * 2;
        dec = digit / 8;
        res = digit%8 + carry;
        if (res > 9) {
            dec = res /8;
            code1[i + 1] = (res%10) + '0';
        } else {
            code1[i + 1] = res + '0';
        }
        if (dec > 0) {
            carry = dec;
        } else {
            carry = 0;
        }
    }
    if (i == -1 && carry > 0) {
        code1[0] = (long)(code1[0] - '0') + carry + '0';
    }
    code1[n+1] = '\0';
    return code1;
}
```

Листинг 3: Новый алгоритм. Перевод десятичного числа в восьмиричную систему. Сложение в восьмиричной системе

```
char* my_octal_add(char* code1, char* code2, long n) {
    char *code;
    char c1, c2;
    int digit, digit1, digit2, sd, dec, res, carry = 0;
    code = malloc(n);
    unsigned long long i, len1 = strlen(code1), len2 = strlen(code2);
    c1 = code1[0];
    sd = c1-'0';
    code[0] = sd + '0';
    if (len1 == len2) {
        for (i = len1 - 1; i > 0; i--) {
            c1 = code1[i];
            c2 = code2[i];
            digit1 = c1-'0';
            digit2 = c2-'0';
            digit = digit1 + digit2;
            dec = digit / 8;
            res = digit%8 + carry;
            if (res > 7) {
                dec = res /8;
                code[i] = (res%8) + '0';
            } else {
                code[i] = res + '0';
            }
            if (dec > 0) {
                carry = dec;
            } else {
                carry = 0;
            }
            if (dec > 0 && i == 1) {
                code[0] = sd + carry + '0';
            }
        }
        if (sd != 0) {
            code[0] = sd + (code2[0] - '0') + carry + '0';
        }
        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: Новый алгоритм. Перевод восьмиричного числа в десятичную систему. Умножение на 2 в десятичной системе

```
char* mul_x_2(char* code, long n) {
    char c;
    int i;
    char* code1;
    code1 = malloc(n + 1);
    long digit, dec, res;
    long carry = 0;
    code1[0] = '0';
    for (i = n - 1; i >= 0; i--) {
        c = code[i];
        char *pChar = &c;
        digit = (long)atoi(pChar);
        digit = digit * 2;
        dec = digit / 10;
        res = digit%10 + carry;
        if (res > 9) {
            dec = res / 10;
            code1[i + 1] = (res%10) + '0';
        } else {
            code1[i + 1] = res + '0';
        }
        if (dec > 0) {
            carry = dec;
        } else {
            carry = 0;
        }
    }
    if (i == -1 && carry > 0) {
        code1[0] = (long)(code1[0] - '0') + carry + '0';
    }
    code1[n+1] = '\0';
    return code1;
}
```

Листинг 5: Новый алгоритм. Перевод восьмиричного числа в десятичную систему. Вычитание в десятичной системе

```
char* my_decimal_sub(char* code1, char* code2, long n) {
    char *code;
    int digit, digit1, digit2, sd, dec, res, carry = 0;
    code = malloc(n + 1);
    sd = code1[0] - '0';
    code[0] = sd + '0';
    for (int i = n - 1; i > 0; i--) {
        digit1 = code1[i]-'0';
        digit2 = code2[i]-'0'; ;
        digit = digit1 - digit2 - carry;
        if (digit < 0) {
            carry = 1;
            code[i] = (digit + 10) + '0';
        } else {
            carry = 0;
            code[i] = digit + '0';
        }
        if (carry != 0 && i == 1) {
            code[0] = sd - carry + '0';
        }
    }
    code[n] = '\0';
    return code;
}
```

Листинг 6: Стандартный алгоритм. Перевод из десятичной системы в восьмиричную систему.

```
char* convert_from_decimal_to_octal(char* code, long n) {
    bool new_step = false;
    char* result;
    result = (char*)malloc((n + 2) * sizeof(char));
    int j = 0, d;
    char* code2;
    char* first_two_digits;
    first_two_digits = malloc(3);
    long digits, rem, quot, len = n;
    int first_digit = code[0] - '0', flag = 0;
    if (first_digit < 8) {
        slice(code, first_two_digits, 0, 2);
        first_two_digits[2] = '\0';
        digits = atol(first_two_digits);
        quot = digits / 8;
        rem = digits - quot * 8;
        flag = 1;
    } else {
        quot = first_digit / 8;
        rem = first_digit - quot * 8;
    }
    while( len != 1) {
        if (new_step) {
            new_step = false;
            first_digit = code[0] - '0',
            flag = 0;
            if (first_digit < 8) {
                slice(code, first_two_digits, 0, 2);
                first_two_digits[2] = '\0';
                digits = atol(first_two_digits);
                quot = digits / 8;
                rem = digits - quot * 8;
                flag = 1;
                free(code2);
            } else {
                quot = first_digit / 8;
                rem = first_digit - quot * 8;
```

```
            flag = 0;
        }
    }
    int k = 0;
    code2 = malloc(len + 1);
    code2[k++] = (int)quot + '0';
    for (int i = 1; i < len; i++) {
        if (flag == 1) {
            i++;
            if (i == len) {
                break;
            }
        }
        d = code[i] - '0';
        flag = 0;
        if (rem != 0) {
            digits = rem * 10 + d;
            quot = digits / 8;
            rem = digits - quot * 8;
        } else {
            if (d < 8 && i != len - 1) {
                code2[k++] = '0';
                slice(code, first_two_digits, i, i + 2);
                first_two_digits[2] = '\0';
                digits = atol(first_two_digits);
                quot = digits / 8;
                rem = digits - quot * 8;
                i++;
            } else {
                quot = d / 8;
                rem = d - quot * 8;
            }
        }
        code2[k++] = (int)quot + '0';
    }
    result[j++] =  (int)rem + '0';
    new_step = true;
    code2[k] = '\0';
    memcpy(code, code2, k);
    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}.

```
unsigned long convert_to_2_in_32_system(char* code) {
    unsigned long res = 0;
    unsigned long cur;
    int deg = 0;
    int len = strlen(code);
    for (int i = len - 1; i >= 0; i--) {
        cur = code[i] - '0';
        res += cur * pow(2, deg);
        ++deg;
    }
    return res;
}
```

Листинг 8: Заполнение структуры Рефала LINK.

```
LINK* start_number = (LINK*)malloc(sizeof(LINK));
LINK* end_number;
LINK* prev_number;
prev_number = NULL;
bool flag_chetn = false;
unsigned long long old_len, len = strlen(binary);
while (len > 0) {
    LINK* number;
    number = (LINK*)malloc(sizeof(LINK));
    if (prev_number != NULL) {
        NEXT(prev_number) = number;
    }
    TYPE(number) = 'd';
    PREV(number) = prev_number;
    char* str = malloc(33 * sizeof(char));
    old_len = len;
    if (len >= 32) {
        slice(binary, str, len-32, len);
        len -= 32;
    } else {
        slice(binary, str, 0, len);
        len -= len;
```

```
    }

    PAIR(number).n = convert_to_2_in_32_system(str);
    if (old_len == strlen(binary)) {
        end_number = number;
    }
    free(str);
    prev_number = number;
    flag_chetn = !flag_chetn;
}

if (len == 0) {
    if (flag_chetn) {
        LINK* number = (LINK*)malloc(sizeof(LINK));
        if (prev_number != NULL) {
            NEXT(prev_number) = number;
        }
        TYPE(number) = 'd';
        PREV(number) = prev_number;
        PAIR(number).n = 0;
        start_number = number;
    } else {
        start_number = prev_number;
    }
}
```

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

```
void print_Link_64(LINK* l, FILE* output) {
    if (output != NULL) {
        if (l != NULL && PREV(l) != NULL) {
            unsigned long senior = l->pair.n;
            unsigned long junior = l->prec->pair.n;
            char* macrodigit = malloc(40);
            char* jun = malloc(20);
            sprintf(macrodigit, "%ld", senior);
            sprintf(jun, "%ld", junior);
            strcat(macrodigit, jun);
            fputs(macrodigit, output);
            print_Link_64(l->prec->prec, output);
        }
    } else {
        if (l != NULL && PREV(l) != NULL) {
            unsigned long senior = l->pair.n;
            unsigned long junior = l->prec->pair.n;
            char* macrodigit = malloc(40);
            char* jun = malloc(20);
            sprintf(macrodigit, "%ld", senior);
            sprintf(jun, "%ld", junior);
            strcat(macrodigit, jun);
            printf("%s",macrodigit);
            print_Link_64(l->prec->prec, NULL);
        }
    }
}
```

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. Гашков С. Б. Системы счисления и их применение. — Москва : Московский центр непрерывного математического образования, 2012.
4. Гашков С. Б. Занимательная компьютерная арифметика: Математика и искусство счета на компьютерах и без них. — Москва : Едиториал УРСС, 2017.