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 Емельяненко Дарья 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, его программными продуктами и предметной областью, для которой предназначены эти продукты;
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. обобщение полученных результатов и навыков, составление отчёта по практике.

1 Характеристика предприятия

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 одним из самых 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

Функциональные языки программирования - это класс языков программирования, ориентированных на работу с функциями как основными строительными блоками программ. Они основываются на математическом понятии функции и применяют функциональные концепции в разработке программ.

Примером функционального языка выступает РЕФАЛ [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 [3].

2.1 Цель работы

Цель работы - сравнение и реализация на Си стандартного алгоритма умножения «столбиком» и алгоритма, основанного на Китайской теореме об остатках. 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.

2.2 Поставленные задачи

1. ознакомление с необходимой теорией для реализации заданных алгоритмов;
2. реализация алгоритма перевода числа из 10-тичной системы счисления в систему счисления 2^{64}
3. реализация стандартного алгоритма умножения на языке C;
4. реализация алгоритма умножения, основанного на Китайской теореме об остатках, на языке C;
5. генерирование различных тестовых данных;
6. измерение времени выполнения каждого алгоритма на тестовых данных;

7. сравнение результатов;
8. систематизация результатов тестирования в сравнительные таблицы.

2.3 Изучение предметной области

Перед реализацией необходимо было ознакомиться с теорией, а именно вспомнить стандартный алгоритм перемножения «столбиком» для чисел в системе счисления 2^{64}, Китайскую теорему об остатках, алгоритм, основанный на этой теореме. Также необходимо было ознакомиться с алгоритмом перевода из 10-тичной системы счисления в систему счисления по основанию 2^{64}.

Исходя из условия поставленной задачи, ниже представлены алгоритмы, которые нужно было реализовать.

Алгоритм перевода из десятичной системы счисления в систему счисления по основанию 2^{64}:

1. число переводится в двоичную систему счисления алгоритмом стандартного деления в столбик;
2. из двоичной системы счисления переводится в систему счисления по основанию 2^{64};

Алгоритм стандартного умножения: Пусть заданы два целых числа по основанию b - первое число $u_1u_2...u_n$, второе число $v_1v_2...v_m$. Алгоритм находит их произведение $w_1w_2...w_{m+n}$.

1. устанавливаем все значения $w_{m+1}w_{m+2}...w_{m+n}$ равными нулю;
2. устанавливаем j=m (индекс цифры второго сомножителя);
3. если $v_j = 0$, устанавливаем $w_j = 0$ и передаем управление на шаг 8;
4. устанавливаем i=n (индекс цифры первого сомножителя), k=0 (цифра переноса);
5. устанавливаем t=$u_i * v_j + w_{i+j} + k$
6. устанавливаем $w_{i+j} = t \mod b$, $k = [\frac{t}{b}]$;
7. цикл по индексу i. Уменьшаем i на единицу, если i>0, то возвращаемся в шаг 5, иначе устанавливаем $w_j = k$;
8. цикл по индексу j. Уменьшаем j на единицу, если j>0, то возвращаемся в шаг 3, иначе заканчиваем алгоритм.

Пример 1. *Умножение 621 на 201 в десятичной системе счисления*

$$\begin{array}{r} 621 \\ * \quad 201 \\ \hline 621 \\ + \quad 000 \\ \quad 1242 \\ \hline 124821 \end{array}$$

Ответ: $621 * 201 = 124821$

Алгоритм умножения, основанный на Китайской теореме об остатках [4]:

1. выполняем умножение двух чисел по стандартного алгоритму умножения в системе счисления 2_{64};
2. задаем взаимно простые модули (a_1,$a_2 ... a_n$);
3. находим все остатки от деления получившегося числа на модули ($r_1 r_2 ... r_n$), которые должны являться взаимно простыми числами;
4. вычисляем M равный произведению всех модулей (M= $\Pi_{i=1}^{n} a_i$);
5. для каждого модуля вычисляем $M_i = \dfrac{M}{a_i}$;
6. для каждого M_i вычисляем его обратный элемент по модулю a_i ($Mi^{-1} \equiv \frac{1}{M_i} \mod a_i$);
7. Вычисляем искомое значение по формуле $x \equiv \sum_{i=1}^{n} r_i M_i M_i^{-1} \mod M$

При этом для корректной работы данного алгоритма должно выполняться следующее условия:

— произведение входных чисел не должно превышать произведения всех модулей.

Для увеличения скорости работы алгоритма взаимно простые модули $a_1 .. a_n$ должны быть простыми числами Мерсенна.

Число Мерсенна - число вида $M_n = 2^n - 1$, где n - натуральное число; такие числа примечательны тем, что некоторые из них являются простыми при больших значениях n. Примеры простых чисел Мерсенна - 3, 7, 31, 127, 8191, 131 071, 524 287, 2 147 483 647 [5].

Пример 2. *Умножение 3675356 на 2912709, используя алгоритм умножения, основанный на Китайской теореме об остатках*

1. выберем взаимно простые модули

$$\begin{cases} a_1 = 3 \\ a_2 = 7 \\ a_3 = 31 \\ a_4 = 127 \\ a_5 = 8191 \\ a_6 = 131071 \end{cases}$$

2. находим результат умножения по стандартному алгоритму - 3675356*2912709=10705242499404;
3. найдем все $r_1 ... r_n$

$$\begin{cases} r_1 = 0, \\ r_2 = 5, \\ r_3 = 18, \\ r_4 = 102, \\ r_5 = 2537, \\ r_6 = 93393. \end{cases}$$

4. вычислим M=3*7*31*127*8191*131071=88762238935797;
5. вычислим все M_i

$$\begin{cases} M_1 = 29587412978599 \\ M_2 = 12680319847971, \\ M_3 = 2863298030187, \\ M_4 = 698915267211, \\ M_5 = 10836557067, \\ M_6 = 677207307. \end{cases}$$

6. вычислим обратные элементы к M_i

$$\begin{cases} M_1^{-1} = 1, \\ M_2^{-1} = 6, \\ M_3^{-1} = 3, \\ M_4^{-1} = 34, \\ M_5^{-1} = 5969, \\ M_6^{-1} = 93887. \end{cases}$$

7. вычислим искомое значение x = $((0 * 29587412978599 * 1) \mod 88762238935797 + (5 * 12680319847971 * 6) \mod 88762238935797 + (18 * 2863298030187 * 3) \mod 88762238935797 + (102 * 698915267211 * 34) \mod 88762238935797 + (2537 * 10836557067 * 5969) \mod 88762238935797 + (93393 * 677207307 * 93887) \mod 88762238935797) \mod 88762238935797 = 10705242499404$

Ответ: $3675356 * 2912709 = 10705242499404$

2.4 Разработка и реализация

Для достижения поставленной задачи был реализован программный продукт на языке C в программе Visual Studio Code. В результате были реализованы необходимые алгоритмы, которые принимают на вход числа произвольной длины и возвращают результат умножения этих чисел. Все данные считываются из файла input и возвращаются в виде файла output, который в случае корректной работы, то есть одинакового ответа алгоритма путем стандратного умножения и алгоритма, основанного на Китайской теореме об остатках, содержит информацию о длине чисел, а также времени работы каждого алгоритма. В случае некорректной работы файл содержит информацию о тесте, на котором произошла поломка, а также ответы, которые выдали два алгоритма.

2.5 Особенности программы

Для реализации заданных алгоритмов необходимо было решить несколько проблем. Первая из них - хранение больших чисел в памяти компьютера. Для этого было решено использовать структуру, которая представляет из себя структуру, близкую к той, которую используют в используют в реализации интерпретатора Рефала-5 для представления элементарных данных (термов) языка Рефал [2].

```
/*** 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;
```

По условию задачи необходимо было работать с числами в системе счисления 2^{64}, но одна цифра может превышать максимальное значение типа unsigned long в C, поэтому было принято решение хранить числа в системе счисления 2^{32} при этом число может быть легко представлено в нужной системе счисления. Для этого достаточно взять два разряда и домножить старший разряд на 2^{32}, данное умножение можно произвести простым побитовым сдвигом на 32 бита влево.

Следующая проблема, которая возникла передо мной, заключалась в реализации алгоритма, основанного на Китайской теореме об остатках. Для выполнения первого шага, то есть нахождения остатков от деления, необходимо было реализовать алгоритм деления в системе счисления 2^{64}. Стандратный алгоритм деления столбиком не подходит для больших систем счисления, так как компьютер будет долго «угадывать» необходимую цифру. Решение данной проблемы - использо-

вание алгоритма деления, используемого при реализации языка FLAC, который является диалектом Рефала [6].

Алгоритм:

Пусть первое число $n_1 = a_n a_{n-1} ... a_0$, второе число $n_2 = b_m b_{m-1} ... b_0$

— необходимо найти $\delta = min([\frac{a_n * \beta + a_{n-1}}{b_m}, \beta - 1])$;

— домножаем делимое и делитель на δ - $a_n a_{n-1} ... a_0 * \beta = a_l a_{l-1} ... a_0$, $b_m b_{m-1} ... b_0 * \beta = b_k b_{k-1} ... b_0$;

— если $a_l \geq b_k$, то в ответ записывается цифра 1;

— иначе цифра = $min([\frac{a_l * \beta + a_{l-1}}{b_m}, \beta - 1])$;

— домножаем делитель на цифру и проверяем, что число меньше, чем делимое, иначе уменьшаем цифру на 1;

— повторяем предыдущие шаги, пока число не становится меньше делимого.

Последней проблемой стало нахождение обратного числа по определенному модулю. Алгоритм для его нахождения:

— используем расширенный алгоритм Евклида для нахождения x и y, таких что ax+ny=d, где d - наибольший общий делитель a,n [7];

— если d>1, то обратного элемента не существует, иначе возвращаем x.

3 Тестирование

Целью тестирования - сравнение двух алгоритмов по времени их выполнения с целбю определения наиболее быстрого варианта.

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

Таблица 1 — Среднее время работы стандартного и нового алгоритмов умножения в системе счисления по основанию 2^{64}.

Длина входного десятичного числа	Время выполнения стандартного алгоритма, с.	Время выполнения Китайской теоремы об остатках, с.
1	0.000001	0.000003
2	0.000001	0.00001
4	0.000001	0.00003
8	0.000001	0.000031
16	0.000001	0.000008
20	0.000001	0.000045
32	0.000003	0.00050
64	0.000007	0.00055
128	0.00002	0.0003

3.1 Вывод

Таким образом, сравнивая времена двух алгоритмов, для чисел, длина которых в десятичной системе не превышает 128 символов, можно сделать вывод о том, что стандартный алгоритм работает быстрее, чем алгоритм, основанный на Китайской теореме об остатках. Происходит это по причине дополнительных затрат времени на алгоритм деления, который активно используется во втором алгоритме, наибольший выигрыш прослеживается когда разложение модуля состоит из одного простого числа. Возможным решением проблемы является оптимизация алгоритма деления.

ЗАКЛЮЧЕНИЕ

В результате практики поставленные задачи были успешно выполнены. Я изучила необходимую теорию для реализации стандартного алгоритма умножения, а также алгоритма, основанного на Китайской теореме об остатках. Также создала функцию, которая переводит число из десятичной системы счисления в число в системе счисления по основанию 2^{64}. Мною был реализован стандратный алгоритм умножения длинных чисел, а также алгоритм умножения длинных чисел, основанный на Китайской теореме об остатках. Также были реализованы функции необходимые для корректной работы вышеперечисленных алгоритмов, а именно вся арифметика для чисел по основанию 2^{64} и расширенный алгоритм Евклида для нахождения обратного элемента по заданному модулю. Также была создана функция, которая способна генерировать примеры, содержащие числа, заданной пользователем длины. На данных примерах была протестирована работа стандартного алгоритма умножения, а также алгоритма, основанного на Китайской теореме об остатках, и было осуществлено сравнение времен выполнения вышеперечисленных алгоритмов.

В ходе разработки был получен опыт работы с языком программирования Си. Результаты работы подтвердили гиппотезу о том, что умножение больших чисел, основанное на Китайской теореме об остатках, работает быстрее, чем стандатный алгоритм умножения, но на числах очень большой длины. Таким образом для вычисления умножения чисел небольшой длины рекомендуется использовать стандартный алгоритм умножения.

В дальнейшем видится совершенствование проекта, а именно усовершенствование алгоритма деления и нахождения остатка по модулю по причине уменьшения времени выполнения алгоритма, основанного на Китайской теореме об остатках, и улучшения его работоспособности на небольших числах (числах длины ≤ 128).

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] Рефал [Электронный ресурс]. url: https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BB.

[4] Китайская теорема об остатках[Электронный ресурс]. url: https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D1%82%D0%B0%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%BE%D0%B1_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B0%D1%85.

[5] Число Мерсенна [Электронный ресурс]. url: https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%9C%D0%B5%D1%80%D1%81%D0%B5%D0%BD%D0%BD%D0%B0.

[6] В. Ф. Турчин, Сборник трудов по функциональному языку программирования. Переславль-Залесский: СБОРНИК, 2014.

[7] С. М. Окулов, Алгоритмы компьютерной арифметики. Москва: Лаборатория знаний, 2021.

ПРИЛОЖЕНИЕ А

Листинг 1: Реализация стандартного алгоритма умножения

```
LINK* alghorithm_multyplication(LINK* first_num,LINK* second_num){
    LINK* res=NULL;
    res=(LINK*)malloc(sizeof(LINK));
    res->foll=NULL;
    res->prec=NULL;
    if (((first_num->ptype=='d')&&(second_num->ptype=='n'))||((first_num->ptype=='n')&&(second_num->ptype=='d')))
      res->ptype='n';
    else res->ptype='d';
    res->pair.n=0;
    unsigned long long size_first_num=Size_Link(first_num);
    unsigned long long size_second_num=Size_Link(second_num);
    if (((size_first_num==1)&&(first_num->pair.n==0))||((size_second_num==1)&&(second_num->pair.n==0))) return res;
    if ((size_first_num>size_second_num)||(size_first_num==size_second_num)){
        LINK* dop_for_dop_res2=NULL;
        LINK* dop_for_res=res;
        LINK* dop_for_res2=res;
        LINK* dop_for_first_num=first_num;
        while (second_num!=NULL){
        while (dop_for_first_num!=NULL){
            unsigned long long mnog=0;
            unsigned long mainpart=0;
            unsigned long ost=0;
            //Проверка на переполнение
            if (dop_for_res2==NULL){
                dop_for_res2=(LINK*)malloc(sizeof(LINK));
                dop_for_res2->ptype='d';
                dop_for_res2->foll=NULL;
                dop_for_res2->prec=dop_for_dop_res2;
                dop_for_res2->pair.n=0;
            }
                mnog=second_num->pair.n*dop_for_first_num->pair.n;
                mnog=dop_for_res2->pair.n+mnog;
                if (mnog){
```

```
33                  mainpart=mnog>>32;
34                  ost=mnog-(mainpart<<32);
35              }
36              else{
37                  mainpart=0;
38                  ost=0;
39              }
40          dop_for_res2->pair.n=ost;
41              if (dop_for_res2->foll==NULL){
42                  dop_for_res2->foll=(LINK*)malloc(sizeof(LINK));
43                  dop_for_res2->foll->ptype='d';
44                  dop_for_res2->foll->pair.n=0;
45                  dop_for_res2->foll->foll=NULL;
46                  dop_for_res2->foll->prec=dop_for_res2;
47              }
48              dop_for_res2->foll->pair.n+=mainpart;
49          dop_for_dop_res2=dop_for_res2;
50          dop_for_res2=dop_for_res2->foll;
51          dop_for_first_num=dop_for_first_num->foll;
52      }
53      dop_for_first_num=first_num;
54      second_num=second_num->foll;
55      dop_for_res=dop_for_res->foll;
56      dop_for_res2=dop_for_res;
57      }
58  }
59  else{
60      LINK* dop_for_dop_res2=NULL;
61      LINK* dop_for_res=res;
62      LINK* dop_for_res2=res;
63      LINK* dop_for_second_num=second_num;
64      while (first_num!=NULL){
65      while (dop_for_second_num!=NULL){
66          unsigned long long mnog=0;
67          mnog=first_num->pair.n*dop_for_second_num->pair.n;
68          if (dop_for_res2==NULL){
69              dop_for_res2=(LINK*)malloc(sizeof(LINK));
70              dop_for_res2->ptype='d';
71              dop_for_res2->foll=NULL;
72              dop_for_res2->prec=dop_for_dop_res2;
```

```
73                  dop_for_res2->pair.n=0;
74              }
75              mnog=dop_for_res2->pair.n+mnog;
76              unsigned long mainpart=mnog>>32;
77              unsigned long ost=mnog-(mainpart<<32);
78              dop_for_res2->pair.n=ost;
79                  if (dop_for_res2->foll==NULL){
80                      dop_for_res2->foll=(LINK*)malloc(sizeof(LINK));
81                      dop_for_res2->foll->ptype='d';
82                      dop_for_res2->foll->pair.n=0;
83                      dop_for_res2->foll->foll=NULL;
84                      dop_for_res2->foll->prec=dop_for_res2;
85                  }
86                  dop_for_res2->foll->pair.n+=mainpart;
87              dop_for_dop_res2=dop_for_res2;
88              dop_for_res2=dop_for_res2->foll;
89              dop_for_second_num=dop_for_second_num->foll;
90          }
91          dop_for_second_num=second_num;
92          first_num=first_num->foll;
93          dop_for_res=dop_for_res->foll;
94          dop_for_res2=dop_for_res;
95          }
96      }
97      if (res!=NULL) res=DELETE_ZERROW(res);
98      return res;
99  }
```

Листинг 2: Реализация Китайской теоремы об остатках

```
LINK* chinese_theorema(M_I* m_i,M_I* r_i,LINK* M,M_I* y_i){
    LINK* x=(LINK*)malloc(sizeof(LINK));
    x->foll=NULL;
    x->pair.n=0;
    x->prec=NULL;
    x->ptype='d';
    while (m_i!=NULL){
        char* check;
        LINK* s_i=extended_euclid(y_i->num,m_i->num);
        LINK* c_i=alghorithm_multyplication(r_i->num,s_i);
        c_i=FIND_MOD(c_i,m_i->num);
        LINK* x_dop=NULL;
        x_dop=alghorithm_multyplication(c_i,y_i->num);
        x_dop=FIND_MOD(x_dop,M);
        x=ADD_LINK(x,x_dop);
        m_i=m_i->next;
        r_i=r_i->next;
        y_i=y_i->next;
    }
    x=FIND_MOD(x,M);
    if ((Size_Link(x)!=1)&&(x->pair.n!=0)) x=DELETE_ZERROW(x);
    return x;

}
```

Листинг 3: Реализация алгоритма деления в системе счисления по основанию 2^{64}

```
LINK* Div_in_32_system(LINK*a, LINK* b){
    if ((a==NULL)||(b==NULL)) return NULL;
    if (Compare(a,b)==-1){
        LINK* res;
        res=(LINK*)malloc(sizeof(LINK));
        res->pair.n=0;
        res->foll=NULL;
        res->prec=NULL;
        res->ptype='d';
        return res;
    }
    if (((Size_Link(a)==Size_Link(b))&&(Size_Link(a)==1))){
        if (!b->pair.n){
            printf("Деление на 0 запрещено\n");
            return NULL;
        }
        LINK* res;
        res=(LINK*)malloc(sizeof(LINK));
        unsigned long long res1=a->pair.n/b->pair.n;
        if ((res1)&&(((a->ptype=='d')&&(b->ptype=='n'))||((a-
      >ptype=='n')&&(b->ptype=='d'))))
       res->ptype='n';
        else res->ptype='d';
        res->prec=NULL;
        res->foll=NULL;
        res->pair.n=res1;
        return res;
    }
    if ((Size_Link(a)==1)&&(a->pair.n==0)){
        LINK* res;
        res=(LINK*)malloc(sizeof(LINK));
        res->foll=NULL;
        res->pair.n=0;
        res->prec=NULL;
        res->ptype='d';
        return res;
    }
```

```
36      LINK* bn=reverse(b);
37      LINK* normal_a;
38      LINK* normal_b;
39      LINK* vector_for_normalization=NULL;
40      vector_for_normalization=(LINK*)malloc(sizeof(LINK));
41      vector_for_normalization->ptype='d';
42      vector_for_normalization->prec=NULL;
43      unsigned long long base=4294967296;
44      vector_for_normalization->pair.n=(unsigned long) (base/(bn->pair.n+1));
45      vector_for_normalization->foll=NULL;
46      unsigned long long base_without_1=4294967296-1;
47      normal_a=alghorithm_multyplication(a,vector_for_normalization);
48      normal_b=alghorithm_multyplication(b,vector_for_normalization);
49      //Результат
50      LINK* res;
51      res=(LINK*)malloc(sizeof(LINK));
52      res->foll=NULL;
53      res->prec=NULL;
54      res->pair.n=0;
55      res->ptype='d';
56      LINK* normal_a_reverse;
57      LINK* normal_b_reverse;
58      normal_a_reverse=reverse(normal_a);
59      normal_b_reverse=reverse(normal_b);
60      //Остаток
61      LINK* q;
62      q=(LINK*)malloc(sizeof(LINK));
63      q->foll=NULL;
64      q->pair.n=normal_a_reverse->pair.n;
65      q->prec=NULL;
66      q->ptype='d';
67      unsigned long long first_num=normal_a_reverse->pair.n;
68      unsigned long long second_num=normal_a_reverse->prec->pair.n;
69      while (Compare(q,normal_b)==-1){
70          if (normal_a_reverse->prec==NULL){
71              LINK* ost;
72              ost=(LINK*)malloc(sizeof(LINK));
73              ost->foll=NULL;
74              ost->ptype='d';
75              ost->prec=NULL;
```

```
76              ost->pair.n=0;
77              return ost;
78          }
79          normal_a_reverse=normal_a_reverse->prec;
80          q->prec=malloc(sizeof(LINK));
81          q->prec->ptype='d';
82          q->prec->foll=q;
83          q->prec->pair.n=normal_a_reverse->pair.n;
84          q->prec->prec=NULL;
85          q=q->prec;
86      }
87      if (first_num>=normal_b_reverse->pair.n){
88          res->pair.n=1;
89          LINK* mnog;
90          mnog=alghorithm_multyplication(normal_b,res);
91          q=SUB_LINK(q,mnog);
92      }
93      else{
94          LINK* delta;
95          delta=(LINK*)malloc(sizeof(LINK));
96          delta->ptype='d';
97          delta->foll=NULL;
98          delta->prec=NULL;
99          unsigned long long num=(first_num<<32)/normal_b_reverse->pair.n;
100         unsigned long long num2=second_num/normal_b_reverse->pair.n;
101         num=num+num2+2;
102         if (num<=base_without_1) {
103             delta->pair.n=num;
104         }
105         else delta->pair.n=base_without_1;
106         while (true){
107             LINK* mnog;
108             mnog=alghorithm_multyplication(delta,normal_b);
109             if (Compare(mnog,q)==1) delta->pair.n-=1;
110             else break;
111         }
112         res->pair.n=delta->pair.n;
113         LINK* mnog;
114         mnog=alghorithm_multyplication(res,normal_b);
115         q=SUB_LINK(q,mnog);
```

```
116         free(delta);
117     }
118     if (normal_a_reverse->prec==NULL) {
119         if (((a->ptype=='d')&&(b->ptype=='n'))||((a->ptype=='n')&&(b-
        >ptype=='d')))
          res->ptype='n';
120         return res;
121     }
122     while (normal_a_reverse!=NULL){
123         normal_a_reverse=normal_a_reverse->prec;
124         if (normal_a_reverse==NULL) break;
125         q->prec=(LINK*)malloc(sizeof(LINK));
126         q->prec->ptype='d';
127         q->prec->foll=q;
128         q->prec->pair.n=normal_a_reverse->pair.n;
129         q->prec->prec=NULL;
130         q=q->prec;
131         LINK* reverse_q=reverse(q);
132         first_num=reverse_q->pair.n;
133         if (reverse_q->prec==NULL) second_num=0;
134         else second_num=reverse_q->prec->pair.n;
135         if (Compare(q,normal_b)==-1){
136             res->prec=(LINK*)malloc(sizeof(LINK));
137             res->prec->ptype='d';
138             res->prec->foll=res;
139             res->prec->prec=NULL;
140             res->prec->pair.n=0;
141             res=res->prec;
142             continue;
143         }
144         if (first_num>=normal_b_reverse->pair.n){
145             res->prec=(LINK*)malloc(sizeof(LINK));
146             res->prec->ptype='d';
147             res->prec->foll=res;
148             res->prec->prec=NULL;
149             res->prec->pair.n=1;
150             res=res->prec;
151             LINK* for_one_num;
152             for_one_num=(LINK*)malloc(sizeof(LINK));
153             for_one_num->ptype='d';
```

```
154             for_one_num->prec=NULL;
155             for_one_num->foll=NULL;
156             for_one_num->pair.n=1;
157             LINK* mnog;
158             mnog=alghorithm_multyplication(normal_b,for_one_num);
159             free(for_one_num);
160             q=SUB_LINK(q,mnog);
161             continue;
162         }
163         else{
164             LINK* delta;
165             delta=(LINK*)malloc(sizeof(LINK));
166             delta->foll=NULL;
167             delta->ptype='d';
168             delta->prec=NULL;
169             delta->pair.n=0;
170             unsigned long long num=(first_num<<32)/normal_b_reverse->pair.n;
171             unsigned long long num2=second_num/normal_b_reverse->pair.n;
172             num=num+num2+2;
173             if (num<=base_without_1) {
174                 delta->pair.n=num;
175             }
176             else delta->pair.n=base_without_1;
177             while (true){
178                 LINK* mnog;
179                 mnog=alghorithm_multyplication(delta,normal_b);
180                 if (Compare(mnog,q)==1){
181                     delta->pair.n-=1;
182                 }
183                 else break;
184             }
185             res->prec=(LINK*)malloc(sizeof(LINK));
186             res->prec->ptype='d';
187             res->prec->foll=res;
188             res->prec->prec=NULL;
189             res->prec->pair.n=delta->pair.n;
190             res=res->prec;
191             LINK* mnog;
192             mnog=alghorithm_multyplication(delta,normal_b);
193             q=SUB_LINK(q,mnog);
```

```
194         }
195     }
196     if (((a->ptype=='d')&&(b->ptype=='n'))||((a->ptype=='n')&&(b-
          >ptype=='d')))
          res->ptype='n';
197     return res;
198 
199 }
200 
```

Листинг 4: Реализация расширенного алгоритма Евклида

```
LINK* extended_euclid(LINK*a, LINK*b){
    if ((a==NULL)||(b==NULL)) return NULL;
    LINK* x=NULL;
    LINK* d=NULL;
    LINK* Zerrow=NULL;
    Zerrow=(LINK*)malloc(sizeof(LINK));
    Zerrow->ptype='d';
    Zerrow->foll=NULL;
    Zerrow->prec=NULL;
    Zerrow->pair.n=0;
    if ((Size_Link(b)==1)&&(b->pair.n)==0){
        if ((Size_Link(a)==1)&&(a->pair.n==1)){
            LINK* res=(LINK*)malloc(sizeof(LINK));
            res->foll=NULL;
            res->prec=NULL;
            res->ptype='d';
            res->pair.n=1;
            return res;
        }
        else{
            printf("ERROR!\n");
            LINK* res=(LINK*)malloc(sizeof(LINK));
            res->foll=NULL;
            res->prec=NULL;
            res->ptype='d';
            res->pair.n=0;
            return res;
        }
    }
    LINK* x2=(LINK*)malloc(sizeof(LINK));
    x2->foll=NULL;
    x2->pair.n=1;
    x2->prec=NULL;
    x2->ptype='d';

    LINK* x1=(LINK*)malloc(sizeof(LINK));
    x1->foll=NULL;
```

```
38      x1->pair.n=0;
39      x1->prec=NULL;
40      x1->ptype='d';
41
42      LINK* q=NULL;
43      LINK* r=NULL;
44
45      LINK* first_num=COPY_LINK_1(a);
46      LINK* second_num=COPY_LINK_1(b);
47
48      while (true){
49          if ((second_num!=NULL)&&(second_num->ptype=='n')) break;
50          if ((second_num!=NULL)&&(Size_Link(second_num)==1)&&(second_num-
        >pair.n==0))
        break;
51          q=Div_in_32_system(first_num,second_num);
52          r=alghorithm_multyplication(q,second_num);
53          r=SUB_LINK(first_num,r);
54          x=alghorithm_multyplication(q,x1);
55          x=SUB_LINK(x2,x);
56          first_num=second_num;
57          second_num=r;
58          x2=x1;
59          x1=x;
60      }
61      while (x2->ptype=='n'){
62        x2=ADD_LINK(x2,b);
63      }
64      if ((Size_Link(first_num)==1)&&(first_num->pair.n==1)) return x2;
65      printf("ERRRRRRRRRROR!!!!\n");
66      LINK* res=(LINK*)malloc(sizeof(LINK));
67      res->foll=NULL;
68      res->prec=NULL;
69      res->ptype='d';
70      res->pair.n=0;
71      return res;
72  }
```