МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

# ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
# ПО ДИСЦИПЛИНЕ «ИНФОРМАТИКА»
# «Работа с текстом»

Факультет: АВТ
Группа: ДТ-460а
Студент: Пантюхин Артём Евгеньевич

Преподаватель:
Копылова Оксана Андреевна

Новосибирск, 2025 г.

Содержание

## Задание

3. Для двух чисел, представленных в виде своих цифр и символов A...F, определить системы счисления каждого, в которых они будут равны, например, 134 и 30 - пятеричная и двенадцатеричная.

## Теоретические сведения

Строка – последовательность (массив) символов, ограниченная символом с кодом 0, иначе говоря, ‘\0’.

Коды символов от ‘0’ до ‘9’ расположены аналогично символам, в порядке возрастания. Аналогично для символов латинского алфавита, заглавного и прописного набора. Так, превратить любой циферный символ в целочисленную цифру можно отняв из него ‘0’ (‘5’-‘0’ == 5, ‘B’-‘A’+10 = 11).

Конвертация из любой системы счисления в десятичную:

Сумма произведений каждой цифры числа на основание системы счисления, возведенное в степень, соответствующую разряду

$1101_2 = 1*2^0+0*2^1+1*2^2+1*2^3 = 1+0+4+8 = 13_{10}$

Для дробных чисел правило сохраняется, разряды после точки идут с отрицательной степенью.

Конвертация из десятичной системы счисления в любую:

Исходное число последовательно делится на основание целевой системы счисления до тех пор, пока не станет равным 0.

Полученные в ходе деления остатки – цифры числа в целевой системе счисления, выписываются в порядке, обратном ходу получения. (так, остаток, полученный последним, будет старшим разрядом числа в целевой системе счисления.

## Проектирование программы

Основные идеи алгоритма

Исходя из формулировки задания, предполагается работа с целыми беззнаковыми числами.
Исходя из предоставленного примера, подразумевается двойная конверсия – введённых чисел в десятичную систему счисления и далее – из чисел в десятичной системе в произвольные с основанием от 2 до 16.
Получая на вход две строки, каждая из которых содержит в себе число в произвольной (неизвестной нам) системе счисления, обрабатываем строки в рамках циклов четырёх уровней вложенности:

1) В первом цикле первое число приводится к десятичной системе счисления, считая, что основание исходной системы – от минимально возможного для символов входного числа (если в числе есть 4 – значит, минимально возможная система - 5).
2) Во втором цикле для каждой из возможных систем первого числа программа аналогично приводит второе число к десятичной системе счисления.
3) В третьем цикле, переводим десятичное представление первого числа в системы счисления от 2 до 16.
4) В четвёртом цикле производим аналогичную операцию для второго числа.

В результате работы, для нашего примера (132 и 30), получаем набор возможных конверсий 132i, 30j -> <число1>, <число2> –> число1k, число2m, где i = [4,16], j = [4,16], k, m = [2,16].

Далее, если строковые представления числа1 и числа2 в системах счисления k и m - одинаковы, программа выводит путь конверсий и итоговый результат.

По завершению работы программы будет получен список всех возможных конверсий, удовлетворяющих условию задания, либо выведено сообщение об отсутствии таких конверсий для двух введённых пользователем чисел.

«Составные части» программы

Получение ввода пользователя.

```
#define NUMERIC_CHARS "0123456789abcdefABCDEF"

//Проверка, является ли символ шестнадцатеричной цифрой.
int isdigit_hex(char ch) {
    return strchr(NUMERIC_CHARS, ch) ≠ NULL;
}

int get_user_input(char* out1, char* out2) {
    char* out_ptrs[2] = {out1, out2};
    char ch = 'g';
    for (int i = 0; i < 2 && ch ≠ EOF; i++) {
        while (!isdigit_hex(ch) && ch ≠ EOF) {
            ch = fgetc(stdin);
        }
        char* inptr = out_ptrs[i];
        while (isdigit_hex(ch)) {
            *(inptr++) = tolower(ch); //Приведение A...F к a...f
            ch = fgetc(stdin);
        }
        *inptr = 0;
    }
    return ch ≠ EOF;
}
```

Переменные:

ch – символ из ввода.

i – счётчик.

inptr – указатель-счётчик на массивы символов 1 и 2.

out1, out2 – указатели на выделенную память под введённые символы, «массивы»

out_ptrs – массив «массивов»

Получение минимально возможного основания для строки символов.

```
int get_min_base(const char* in) {
    //Здесь и далее уже работаем только с a...f
    const char possible[] = "0123456789abcdef";
    int out = 0;
    while (*in) {
        char* poss_ptr = strchr(possible, *in);
        int base = poss_ptr-possible;
        if (base > out) {
            out = base;
        }
        in++;
    }
    return out+1;
}
```

Переменные:

in – входной массив символов, указатель на первый (в ходе работы - последующие) элемент.

possible – упорядоченный массив констант, содержащий возможные символы

base – результат разницы указателей == минимальное основание системы счисления для данной цифры-1.

out – переменная-максимум, минимально возможная система счисления для данного in.

Конвертация символа в число.

```
int to_digit(char ch) {
    int out = 0;
    if (ch ⩾ '0' && ch ⩽ '9') {
        out = ch-48;
    } else if (ch ⩾ 'a' && ch ⩽ 'f') {
        out = ch-97+10;
    }
    return out;
}
```

Переменные:

Out – итоговое число.

Ch – символ.

Конвертация цифры в символ.

```
char to_char(int num) {
    char out = 0;
    if (num ⩾ 0 && num ⩽ 9) {
        out = 48+num;
    } else if (num ⩾ 10 && num ⩽ 16) {
        out = 97+(num-10);
    }
    return out;
}
```

Переменные:

Out – итоговый символ.

num – цифра.

Перевод строки указанного основания в число в десятичной системе счисления.

```
int strtoi_base(const char* str, int base) {
    int out = 0;
    const char* strptr = str;
    while (*(strptr++));
    strptr-=2;
    int power = 0;
    while (strptr ⩾ str) {
        int buff = to_digit(*strptr);
        for (int i = 0; i < power; i++) {
            buff*=base;
        }
        out+=buff;
        power++;
        strptr--;
    }
    return out;
}
```

Переменные:

str – входной массив символов.

base – основание системы счисления, в которой стоит воспринимать входной массив.

out – итоговое число в десятичной системе счисления.

strptr – указатель-счётчик.

power – счётчик, степень, порядок символа исходной строки для перевода.

buff – буфер, результат конверсии одного символа в число, далее умножаемый на основание base в степени power.

Перевод числа в десятичной системе счисления в строковое представление числа с указанным основанием системы счисления.

```
void strfromi_base(int value, int base, char* out) {
    char* endptr = out;
    if (!value) {
        *(endptr++) = '0';
    }
    while (value) {
        int rem = value % base;
        value /= base;
        *(endptr++) = to_char(rem);
    }
    *endptr = 0;
    endptr--;
    while (endptr > out) {
        char buff = *out;
        *out = *endptr;
        *endptr = buff;
        endptr--;
        out++;
    }
}
```

Переменные:

Value – входное число в десятичной системе счисления.

Base – основание целевой системы счисления.

Out – указатель на начало массива символов, выделенного под строковое представление результата. (В конце программы – счётчик.)

Endptr – указатель-счётчик.

Rem – остаток от деления.

**Пример работы программы**

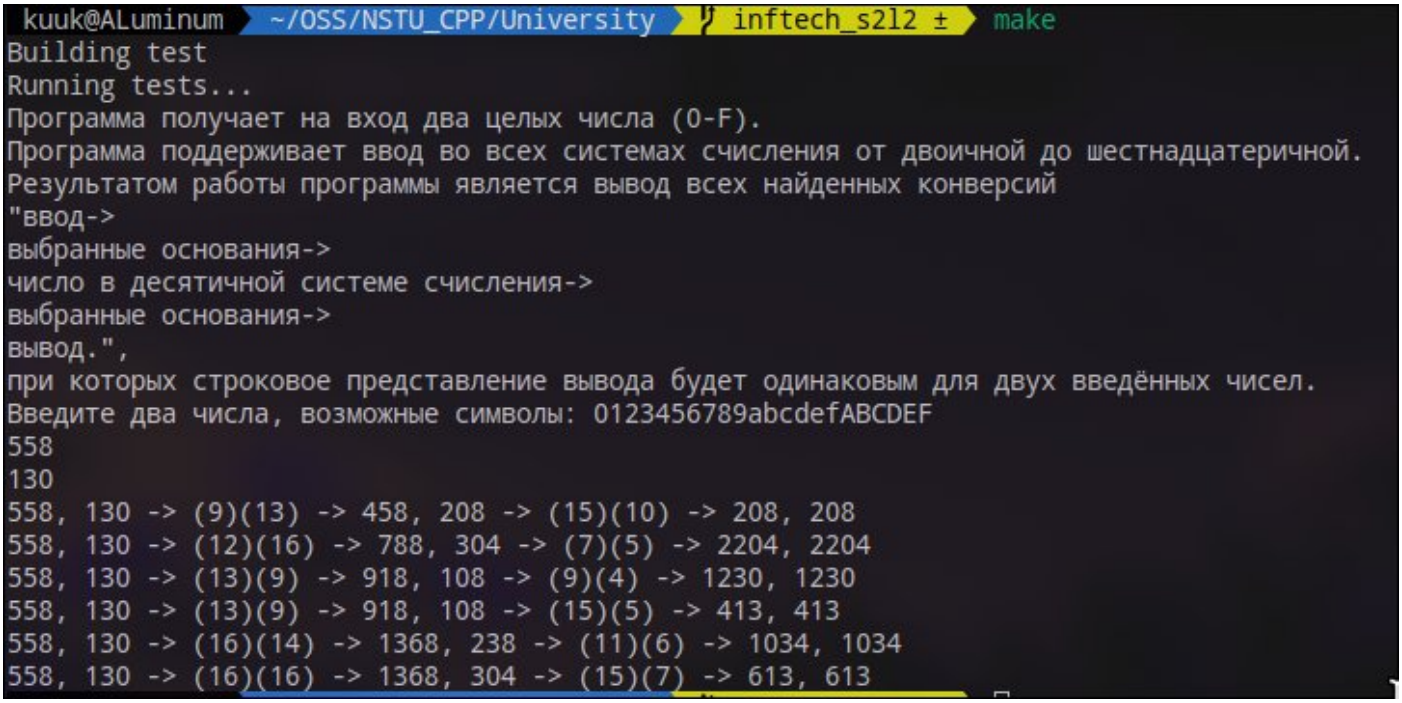


Рисунок 1 – Пример работы программы, успешное нахождение.

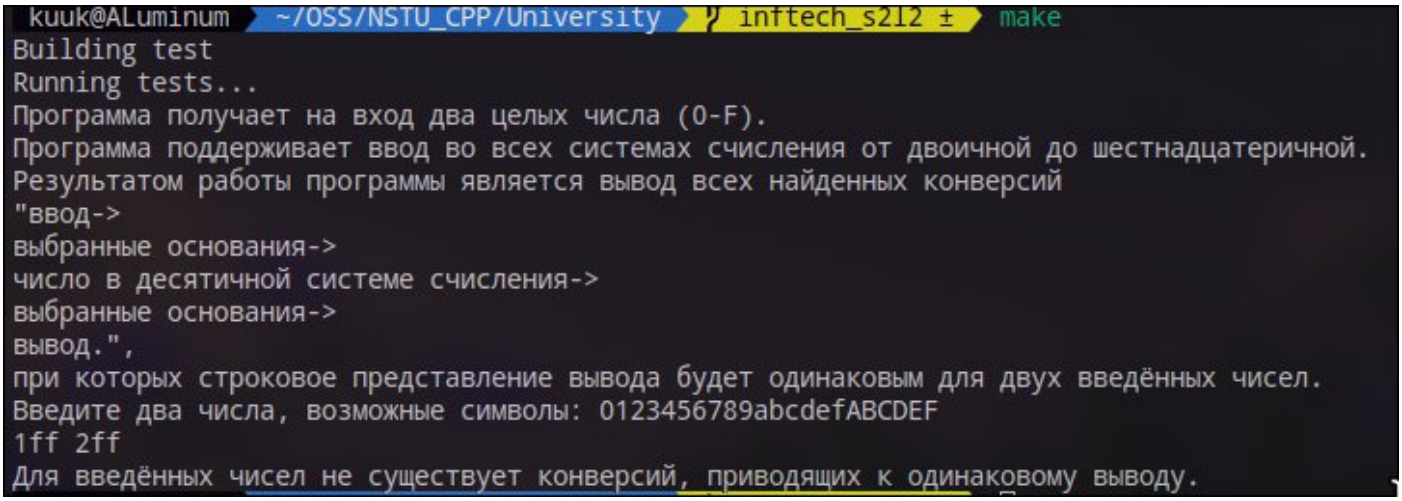


Рисунок 2 – Пример работы программы, нет подходящих.

### Ошибки и неточности

В задании нет явного указания на то, в какой системе счисления стоит воспринимать введённое число. Приведённый пример является результатом результатом работы исходной программы только для одного случая из череды успешно найденных, при этом задание не указывает, необходимо ли продолжать работу после первого успешного нахождения, но системы счисления примера, пусть и могут иметь иную логику работы, взяты из середины итогового списка найденных значений.

### Выводы

В процессе выполнения лабораторной работы были приобретены навыки работы с массивами символов, выбор реализации движений по массивам при помощи указателей положительно сказался на восприятии кода, позволив сократить множество команд до одной операции.

Отметим, что реализованные алгоритмы не подходят для работы с дробными числами, а так же не предполагают ввода отрицательных чисел. Поддержка работы с дробными числами потребует внесения радикальных правок в работу алгоритмов конверсии чисел, поддержка работы с отрицательными числами не затронет алгоритм, но не подразумевается в рамках задания.

В силу неполного описания задания предположения о желаемом результате были получены путём опытной проверки работы программы на числах из примера, пока не был найден вариант работы, при котором можно было получить указанные значения. Данный фактор сказался на уровне декомпозиции программы – несмотря на то, что некоторые функции выполняются лишь единожды, вынести их было необходимо для ускорения проектирования.

**Список литературы**

1. Подбельский В.В., Фомин С.С. Программирование на языке Си: Учеб.пособие. – 2-е доп. Изд. – М.: Финансы и статистика, 2004. – 600 с.

2. Романов Е. Л. Си/Си++. От дилетанта до профессионала. Электронное учебное пособие по дисциплинам "Информатика", "Программирование", "Технология программирования" для студентов 1–2 курсов направления 230100 : учеб. пособие / Е. Л. Романов. – Новосибирский государственный технический университет, № гос регистрации 0321000528, 2010. - 581 с.

3. Си/Си++ от дилетанта до профессионала [Электронный ресурс]. URL: http://ermak.cs.nstu.ru/cprog/HTML/index.htm (дата обращения 01.10.2022).

## Приложение 1. Исходный код программы

```
#include <ctype.h>
#include <locale.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZ_SHORT 255

#define NUMERIC_CHARS "0123456789abcdefABCDEF"
#define GREETINGS_STRING \
"Программа получает на вход два целых числа (0-F).\n\
Программа поддерживает ввод во всех системах счисления \
от двоичной до шестнадцатеричной.\n\
Результатом работы программы является вывод всех найденных конверсий\n\
\"ввод->\n\
выбранные основания->\n\
число в десятичной системе счисления->\n\
выбранные основания->\n\
вывод.\",\n\
при которых строковое представление вывода будет одинаковым для двух
введённых чисел.\n"

//Проверка, является ли символ шестнадцатеричной цифрой.
int isdigit_hex(char ch) {
    //strchr ищет элемент в указанной строке, если найдёт - вернёт
указатель, иначе (void*) 0.
    return strchr(NUMERIC_CHARS, ch) != NULL;
}

int get_user_input(char* out1, char* out2) {
    //Чтобы не копировать код, соберём наши указатели в массив.
    char* out_ptrs[2] = {out1, out2};
    char ch = 'g';
    //Пробежимся по массиву
    for (int i = 0; i < 2 && ch != EOF; i++) {
        //Пропустим мусор
        while (!isdigit_hex(ch) && ch != EOF) {
            ch = fgetc(stdin);
        }
        char* inptr = out_ptrs[i];
        //Пока получаем из stdin шестнадцатеричные цифры - пишем их в
out 1 или 2, сдвигая позицию.
        while (isdigit_hex(ch)) {
```

```
            *(inptr++) = tolower(ch); //приведём A...F к a...f сразу,
чтобы непосредственно алгоритм не тратил время.
            ch = fgetc(stdin);
        }
        *inptr = 0;
    }
    //Если поймали EOF при вводе из файла - значит, в этом файле чего-то
не хватало.
    return ch != EOF;
}

int get_min_base(const char* in) {
    //Здесь и далее уже работаем только с a...f
    const char possible[] = "0123456789abcdef";
    int out = 0;
    //Пока не конец строки ('\0' == 0 == false)
    while (*in) {
        /*
        Для каждого(ptr++) символа из строки на входе
        находим его позицию в нашем массиве возможных
        */
        char* poss_ptr = strchr(possible, *in);
        /*
        Массив уже в нужном порядке, поэтому достаточно просто
        Вычесть указатель на начало из найденного, чтобы получить
наибольшее число базы
        Да, система счисления будет на единичку больше.
        */
        int base = poss_ptr-possible;
        //В рамках первичной конверсии, нас интересует именно
максимальное значение.
        if (base > out) {
            out = base;
        }
        in++;
    }
    /*
    Да, мы можем получить тут базу "1", но
    в реалиях нашего алгоритма нас это абсолютно полностью устраивает,
    ведь это лишь значит, что либо оба числа будут 0,
    либо они никогда не станут равны.
    */
    return out+1;
}
```

```
//Конвертируем символ в десятичное число(да, написано "в_цифру")
int to_digit(char ch) {
    int out = 0;
    //символы 0-9 идут в ряд, аналогично для a-f поэтому мы можем так
сделать
    if (ch ⩾ '0' && ch ⩽ '9') {
        //Можно бы и -'0' писать, но так понятнее, что и зачем.
        out = ch-48;
    } else if (ch ⩾ 'a' && ch ⩽ 'f') {
        //+10 потому, что вычитание уронит нас в [0,5]
        out = ch-97+10;
    }
    return out;
}

//Перевод строки в произвольной базе в десятичное число.
int strtoi_base(const char* str, int base) {
    int out = 0;
    const char* strptr = str;
    //Проматываем до конца строки
    while (*(strptr++));
    //Вычитаем 2: -1 за лишний ++ и -1 за \0, который тоже нам не нужен.
    strptr-=2;
    int power = 0;
    /*
    Идём с конца в начало.
    Можно и идти из начала в конец,
    но длину знать всё равно нужно для степеней.
    */
    while (strptr ⩾ str) {
        //Получаем просто число
        int buff = to_digit(*strptr);
        for (int i = 0; i < power; i++) {
            /*
            Домножаем его на основание, в зависимости от положения в
строке
            первый с конца умножится 0 раз,
            второй 1, третий 2 и так далее.
            */
            buff*=base;
        }
        //Добавляем к итоговому числу результат
        out+=buff;
```

```
        //Увеличили степень для следующего раза.
        power++;
        //Сдвинулись ближе к началу
        strptr--;
    }
    return out;
}

char to_char(int num) {
    char out = 0;
    //Аналогично to_digit() с той лишь разницей, что теперь прибавляем,
а не отнимаем
    if (num ⩾ 0 && num ⩽ 9) {
        out = 48+num;
    } else if (num ⩾ 10 && num ⩽ 16) {
        out = 97+(num-10);
    }
    return out;
}

/*
Конверсия из десятичного числа в строку произвольной базы.
*/
void strfromi_base(int value, int base, char* out) {
    char* endptr = out;
    /*
    Если число ⩵ 0, то просто поставим '0', остальное алгоритм доделает
позже
    Сама проверка нужна потому, что иначе в цикл мы не попадём, и
поставить
    нолик можно будет только нам.
    */
    if (!value) {
        *(endptr++) = '0';
    }
    /*
    Далее, пока от числа ничего не останется, делим его на основание,
    остаток пишем в строку, остальное - до следующей итерации.
    */
    while (value) {
        int rem = value % base;
        value ⁄⩵ base;
        *(endptr++) = to_char(rem);
    }
```

```
    /*
    К этому моменту endptr будет всегда указывать на элемент после
последнего значащего
    Обозначим конец строки и вернёмся в область работы
    */
    *endptr = 0;
    endptr--;
    /*
    Мне почему-то нравится как это работает
    Пока указатель на правую сторону больше указателя на левую,
    мы меняем местами символы в нашей строке
    Если символов чётное число - замечательно, поменяли последний, стали
меньше и разошлись
    Если нечётное - встретились в серединке, а ничего и менять не надо.
    */
    while (endptr > out) {
        char buff = *out;
        *out = *endptr;
        *endptr = buff;
        endptr--;
        out++;
    }
}

int main() {
    setlocale(LC_ALL, "Russian");
    printf(GREETINGS_STRING);
    printf("Введите два числа, возможные символы: %s\n", NUMERIC_CHARS);
    //Создадим два массива под наши строки с числами.
    char in1[BUFSIZ_SHORT] = "";
    char in2[BUFSIZ_SHORT] = "";
    //Тут будет 0 только если вводили из файла и файл закончился раньше
чем всё получилось.
    if (get_user_input(in1, in2)) {
        //Счётчик просто для сообщения о том, что одинаковых не нашли.
        int printed = 0;
        //Начинается страх и ужас, итак:
        //В системе счисления от минимально возможной для введённых
символов 1 числа, и до 16:
        for (int in_base1 = get_min_base(in1); in_base1 ⩽ 16;
in_base1++) {
            //Узнаём как выглядит первое число в десятичной системе.
            int in_val1 = strtoi_base(in1, in_base1);
```

```
            //В системе счисления от минимально возможной для введённых
символов 2 числа, и до 16:
            for (int in_base2 = get_min_base(in2); in_base2 ⩽ 16;
in_base2++) {
                //Узнаём как выглядит второе число в десятичной системе.
                int in_val2 = strtoi_base(in2, in_base2);
                //В системах счисления от 2 до 16,
                for (int out_base1 = 2; out_base1 ⩽ 16; out_base1++) {
                    //Создаём буфер под первое число
                    char out_val1[BUFSIZ_SHORT] = "";
                    //Переводим первое число из десятичной в 2-16
                    strfromi_base(in_val1, out_base1, out_val1);
                    //В системах счисления от 2 до 16,
                    for (int out_base2 = 2; out_base2 ⩽ 16;
out_base2++) {
                        //Создаём буфер под второе число
                        char out_val2[BUFSIZ_SHORT] = "";
                        //Переводим второе число из десятичной в 2-16
                        strfromi_base(in_val2, out_base2, out_val2);
                        //Если по результатам сравнения итоговых строк
вышло, что они одинаковы
                        if (strcmp(out_val1, out_val2)==0) {
                            printed++;
                            /*
                            Выводим:
                            Ввод1, Ввод2 ->
                            <восприняли в>(системе 1.1)(системе 2.1) ->
                            <тогда десятичное> число1, число2 ->
                            <конвертируется в> (систему 1.2)(систему
2.2) ->
                            <и на выходе> Вывод1, Вывод2.
                            */
                            printf("%s, %s -> (%d)(%d) -> %d, %d ->
(%d)(%d) -> %s, %s\n",
                                    in1, in2,
                                    in_base1, in_base2,
                                    in_val1, in_val2,
                                    out_base1, out_base2,
                                    out_val1, out_val2);
                        }
                    }
                }
            }
        }
```

```
        if (!printed) {
            printf("Для введённых чисел не существует конверсий,
приводящих к одинаковому выводу.\n");
        }
    } else {
        printf("Ошибка, проверьте введённые числа.\n");
    }
    //Даже при ошибке ввода, де-юре программа отработала нормально, с
памятью проблем не возникло, значит всё равно нолик.
    return 0;
}
```