МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)

Кафедра программного обеспечения вычислительной
техники и автоматизированных систем

Лабораторная работа №1
по дисциплине: «Алгоритмы и структуры данных»
по теме: **«Встроенные структуры данных (C)»**

Выполнил/а: ст. группы ПВ-231
Чупахина София Александровна
Проверил: Акиньшин Данил Иванович

Белгород, 2024

Цель работы: изучение базовых типов данных языка С как структур данных (СД).

Задания

1. Для типов данных (см. Варианты заданий в таблицах 1,2) определить:
   1.1. Абстрактный уровень представления СД:
      1.1.1. Характер организованности и изменчивости.
      1.1.2. Набор допустимых операций.
   1.2. Физический уровень представления СД:
      1.2.1. Схему хранения.
      1.2.2. Объем памяти, занимаемый экземпляром СД.
      1.2.3. Формат внутреннего представления СД и способ его интерпретации.
      1.2.4. Характеристику допустимых значений.
      1.2.5. Тип доступа к элементам.
   1.3. Логический уровень представления СД.
      1.3.1. Способ описания СД и экземпляра СД на языке программирования.
2. Для заданных типов данных определить набор значений, необходимый для изучения физического уровня представления СД.
3. Преобразовать значения в двоичный код.
4. Преобразовать двоичный код в значение.
5. Разработать и отладить программу, выдающую двоичное представление значений, заданных СД.
6. Обработать программой значения, полученные в результате выполнения пункта 3 задания. Сделать выводы.
7. Разработать и отладить программу, определяющую значение переменной по ее двоичному представлению.
8. Обработать программой значения, полученные в результате выполнения пункта 4 задания. Сделать выводы.

Вариант 21 (Язык С)

Типы данных:

1. signed char
2. long long int
3. float массив [3]

Содержание:

Задание 1:

Задание 1.1

Абстрактный уровень представления СД типа **signed char:**
1.1.1: signed char – тип данных, представляющий собой целое число со знаком. Он представляет собой непрерывную последовательность битов (один байт), которая интерпретируется как число в диапазоне от -128 до 127 или (в случае, если число положительно) как символ, код которого в кодировке ASQII соответствует этому числу. Эта последовательность битов рассматривается как единое целое, и в процессе выполнения программы значение СД типа signed char может быть изменено только с помощью перезаписи.
1.1.2: Для СД типа signed char определены следующие операции:

- Операция инициализации,
- Операции сложения, вычитания, умножения, целочисленного деления и взятия остатка при делении одного числа на другое,
- Операции побитового сдвига числа влево и вправо,
- Операции побитового умножения, сложения, сложения по модулю 2 для двух чисел,
- Операция побитового отрицания числа,
- Операции логического умножения, сложения, отрицания,
- Операции сравнения: равно, не равно, меньше, больше, меньше или равно, больше или равно,
- Операция присваивания и операции присваивания с выполнением любой арифметической или побитовой операции с двумя операндами,
- Операции инкремента и декремента – операции перезаписи с увеличением или уменьшением исходного значения на 1,
- Операция приведения типов.

Абстрактный уровень представления СД типа **long long int:**
1.1.1: long long int – также целый знаковый тип данных. Последовательность битов, представляющая значение этого типа, занимает восемь байт, и сами числа в итоге могут принимать значения от $-(2^{63})$ до $(2^{63}) - 1$. СД этого типа также может быть изменена только с помощью перезаписи.
1.1.2: Для СД типа long long int определены все те же операции, что и для signed char, поскольку они оба относятся к целым знаковым типам:

- Операция инициализации,
- Операции сложения, вычитания, умножения, целочисленного деления и взятия остатка при делении одного числа на другое,
- Операции побитового сдвига числа влево и вправо,
- Операции побитового умножения, сложения, сложения по модулю 2 для двух чисел,
- Операция побитового отрицания числа,
- Операции логического умножения, сложения, отрицания,

- Операции сравнения: равно, не равно, меньше, больше, меньше или равно, больше или равно,
- Операция присваивания и операции присваивания с выполнением любой арифметической или побитовой операции с двумя операндами,
- Операции инкремента и декремента – операции перезаписи с увеличением или уменьшением исходного значения на 1,
- Операция приведения типов.

Абстрактный уровень представления СД типа **float массив [3]:**
1.1.1: **float массив [3]:** – линейная структура. В абстрактном смысле она является последовательностью: ее элементы (элементы массива) находятся в некотором линейном порядке, так что для каждого элемента, кроме первого и последнего, можно указать предыдущий и следующий элемент. Все элементы этой СД относятся к одному и тому же вещественному типу float с диапазоном значений ($1.1754943*10^{-38}$...$1.175494*10^{+38}$) и точностью представления числа примерно в 6-7 знаков после запятой в десятичной записи. СД float массив [3], в отличие от предыдущих, уже не является неделимой, к каждому ее элементу можно обратиться отдельно, перезаписывая его значение.
1.1.2: Для СД типа **float массив [3]** (как и для массива любой длины) определены следующие операции:

- Операция инициализации,
- Обращение к элементу по его индексу и перезапись элемента,
- Операция присваивания по отношению ко всему массиву,
- Все операции над отдельными элементами, определенные для типа float.

Задание 1.2

Физический уровень представления СД типа **signed char:**
1.2.1: Единственный и неделимый элемент данной СД, переменная одноименного типа char с модификатором signed хранится в памяти компьютера как непрерывная последовательность битов, интерпретируемая в соответствии с правилами, описанными в пункте 1.2.3.
1.2.2: СД этого типа занимает ровно 1 байт (8 бит).
1.2.3: Значение этой СД хранится как единая последовательность битов, в которой первый бит отвечает за знак, а остальные 7 – за модуль хранимого числа. Записать значение СД в виде такой последовательности можно, переведя его в двоичную систему счисления (если рассматривать значение char как символ, то нужно сначала отыскать код этого символа в кодировке ASQII), а потом преобразовав его в дополнительный код следующим образом:

- Если число положительное, первый бит (бит знака) обратить в ноль, в оставшуюся часть записать двоичное представление числа без изменений
- Если число отрицательное, первый бит (бит знака) обратить в единицу, в оставшуюся часть записать двоичное представление числа, предварительно сначала побитово инвертировав его, а потом прибавив единицу к полученному числу.

1.2.4: СД signed char может принимать только целые значения от -128 до 127.
1.2.5: Доступ к данной СД осуществляется прямо; можно обратиться к области памяти, идентификатором которой является имя переменной соответствующего типа.

Физический уровень представления СД типа **long long int:**
1.2.1: Единственный и неделимый элемент данной СД, переменная одноименного типа long long int хранится в памяти компьютера как непрерывная последовательность битов, интерпретируемая в соответствии с правилами, описанными в пункте 1.2.3.
1.2.2: СД этого типа занимает 8 байт (64 бит).
1.2.3: Значение этой СД хранится как последовательность битов, в которой первый бит обозначает знак числа, а остальные 63 бита кодируют модуль этого числа. Знак числа кодируется следующим образом: первый бит равен 0, если число положительно, и 1, если оно отрицательно. Идущая следом последовательность из 63 битов равна двоичной записи модуля значения, переведенной в дополнительный код (или дополнение до двух). Двоичное число переводится в дополнительный код следующим образом:

- Если первый бит числа равен нулю, то есть число положительно, то число остается записанным без изменений;
- Если первый бит числа равен единице, то есть число отрицательно, то число (кроме знака) сначала побитово инвертируется (вместо разрядов-нулей записываются единицы и наоборот), а потом к этому инвертированному числу прибавляется 1.

1.2.4: СД long long int может принимать только целые значения от -2^{63} до $2^{63} - 1$.
1.2.5: Доступ к данной СД осуществляется прямо; можно обратиться к области памяти, идентификатором которой является имя переменной соответствующего типа.

Физический уровень представления СД типа **float массив [3]:**
1.2.1: Данная СД имеет прямоугольную схему хранения, то есть ее элементы хранятся в памяти компьютера последовательно, в соответствии с их логическим порядком: последовательности битов, представляющие нулевой, первый и второй элемент, идут друг за другом по возрастанию адреса без разделителей. В языке С массив по формату представляет собой указатель на область памяти заданного типа, на нулевой элемент этого массива. Сами же элементы все так же представляют собой непрерывные последовательности битов, интерпретируемые по правилам, описанным в пункте 1.2.3.
1.2.2: Вычислить объем памяти, занимаемый СД данного типа, можно как объем его базового элемента (переменной типа float), умноженный на количество элементов в массиве (3). Таким образом, СД этого типа занимает 4 * 3 = 12 байт (32 * 3 = 96 бит).
1.2.3: Значение этой СД хранится как три идущих подряд последовательности битов, в каждой из которых первый бит обозначает знак вещественного числа, следующие 8 бит кодируют порядок числа, а оставшиеся 23 – его мантиссу. Знак

числа кодируется так же, как и для целых чисел: первый бит равен 0, если число положительно, и 1, если оно отрицательно. Далее в общем случае интерпретацию дробного числа можно описать так: дробное число переводится в двоичную систему счисления, затем делится на 2 в некоторой степени e (не всегда положительной) таким образом, чтобы двоичная дробь была записана в виде 1.m, где m – некоторая последовательность нулей и единиц. Эта последовательность m и хранится в 23-х разрядах, отведенных для мантиссы, а степень двойки e, на которую пришлось разделить число (и на которую надо умножить мантиссу, чтобы она приняла реальное значение числа), хранится в поле для порядка, причем порядок является смещенным: в то время как реальный порядок можно принимать значения от -128 до 127, в 8 битах, отведенных для порядка, хранится число от 0 до 255.
1.2.4: Каждый из элементов массива может принимать значения от $1.1754943*10^{-38}$...$1.175494*10^{+38}$ с непостоянным шагом (интервал между возможными значениями увеличивается по мере удаления от нуля), в среднем с точностью 6-7 знаков после запятой. Соответственно, сама СД может представлять собой все варианты упорядоченных последовательностей длины 3 из элементов с такими значениями.
1.2.5: Доступ к элементам данной СД осуществляется с помощью индексов от 0 до 2: операция обращения имеет синтаксис array[i], где array – название массива, i – индекс элемента. Этот доступ является прямым, то есть не зависит от размера массива. Поскольку сам массив представляет собой указатель на область памяти, где хранится нулевой элемент, то можно прибавлять к нему целые значения от 0 до 2, чтобы получить указатели на соответствующие элементы.

Задание 1.3

Логический уровень представления СД типа **signed char:**
Переменные этого типа объявляются с использованием ключевого слова (спецификатора) char и модификатора signed со следующим синтаксисом: signed char <имя переменной>:

```
signed char num;
```

Логический уровень представления СД типа **long long int:**
Переменные этого типа объявляются с использованием ключевого слова int и модификаторов long long со следующим синтаксисом: long long int <имя переменной>; допустим также вариант с опущенным спецификатором, тогда его роль выполнят модификаторы: long long <имя переменной>

```
long long big_num1;
```

Логический уровень представления СД типа **float массив [3]:**
Данная СД задается следующей синтаксической конструкцией: float <имя массива> [3]:

```
float array[3];
```

Задание 2:

Для каждой СД подберем данные, которые помогут достаточно полно проанализировать физический уровень. Для целых чисел достаточно одного положительного и одного отрицательного числа в том диапазоне, который данная СД позволяет хранить. Пусть для СД типа **signed char** это будут числа 35 и -111, для типа **long long int** это будут числа 7802697 и -585911. В СД типа **float массив [3]** можно записать три различных дробных числа: положительное целое число, отрицательное число с дробной частью и положительное число с дробной частью, с большим количеством знаков после запятой: {35, -49.875, 299.8675}

Задание 3:

Примечание: в дальнейшем будет апострофами (') разбивать двоичные представления значений для удобства, чтобы отделить друг от друга биты знака и модуля, мантиссы и порядка и т. д.
Преобразуем в двоичный код значения, выбранные для СД типа **signed char.**
Переведем число 35 в двоичную систему путем деления на два и записи остатков в обратном порядке:
35 / 2 = 17 (ост. 1); 17 / 2 = 8 (ост. 1); 8 / 2 = 4 (ост. 0); 4 / 2 = 2 (ост. 0); 2 / 2 = 1 (ост. 0), 1 / 2 = 0 (ост. 1).
Записывая остатки в обратном порядке, получим число 100011. Поскольку оно положительное, бит знака приравнивается к нулю, число не нуждается в переводе в дополнительный код, и перед записью нужно лишь заполнить недостающие биты слева нулями так, чтобы их стало 7. По итогу в двоичном коде СД signed char со значением 35 будет представлена как 0'0100011.
Перейдем к числу -111. Переведем его модуль в двоичную систему путем деления на два и записи остатков в обратном порядке:
111 / 2 = 55 (ост. 1); 55 / 2 = 27 (ост. 1); 27 / 2 = 13 (ост. 1); 13 / 2 = 6 (ост. 1); 6 / 2 = 3 (ост. 0), 3 / 2 = 1 (ост. 1), 1 / 2 = 0 (ост. 1).
Записывая остатки в обратном порядке, получим число 1101111. Поскольку оно отрицательное, бит знака приравнивается к единице, а само число переводится в дополнительный код: сначала каждый его бит инвертируется, в результате чего мы получим последовательность 0010000, а потом к нему прибавляется единица, что даст результат 0010001. Эта последовательность уже имеет длину 7 и в добавлении единиц (так как число отрицательное) слева не нуждается. По итогу в двоичном коде СД signed char со значением -111 будет представлена как 1'0010001.

Преобразуем в двоичный код значения, выбранные для СД типа **long long int.**
Здесь порядок действий будет почти таким же, как и для целых значений типа signed char, с тем лишь различием, что числа здесь куда больше. По этой причине будет неудобно переводить их в двоичную систему путем деления на 2, и мы поступим так: переведем их в шестнадцатеричную систему путем деления на 16 и

записи остатков в обратном порядке, а получившееся шестнадцатеричное число переведем в двоичную, используя двоично-шестнадцатеричную систему. Итак:
7802697 / 16 = 487668 (ост. 9); 487668 / 16 = 30479 (ост. 4); 30479 / 16 = 1904 (ост. 15, F); 1904 / 16 = 119 (ост. 0); 119 / 16 = 7 (ост. 7), 7 / 16 = 0 (ост. 7).
Записывая остатки в обратном порядке, получим число 770F49. Переведя каждую из цифр в двоичную систему:
($0_{16} = 0000_2$, $4_{16} = 0100_2$, $7_{16} = 0111_2$, $9_{16} = 1001_2$, $F_{16} = 1111_2$),
...получим 111011100001111101001001. Поскольку оно положительное, бит знака приравнивается к нулю, число не нуждается в переводе в дополнительный код, и перед записью нужно лишь заполнить недостающие биты слева нулями так, чтобы их стало 64. По итогу в двоичном коде СД long long int со значением 7802697 будет представлена как 0'0000000000000000000000000000000000000000011101110000111101001001.
Перейдем к числу -585911. Переведем его модуль в двоичную систему тем же путем:
585911 / 16 = 36619 (ост. 7); 36619 / 16 = 2288 (ост. 11, B); 2288 / 16 = 143 (ост. 0); 143 / 16 = 8 (ост. 15, F); 8 / 16 = 0 (ост. 8).
Записывая остатки в обратном порядке, получим число 8F0B7. Переведя каждую из цифр в двоичную систему:
($0_{16} = 0000_2$, $7_{16} = 0111_2$, $8_{16} = 1000_2$, $B_{16} = 1011_2$, $F_{16} = 1111_2$),
...получим 10001111000010110111. Поскольку оно отрицательное, бит знака приравнивается к единице, а само число переводится в дополнительный код: сначала каждый его бит инвертируется, в результате чего мы получим последовательность 01110000111101001000, а потом к нему прибавляется единица, что даст результат 01110000111101001001. К этой последовательности стоит прибаить столько единиц (так как число отрицательное) слева, чтобы ее длина достигла 63. По итогу в двоичном коде СД long long int со значением -585911 будет представлена как 1'1111111111111111111111111111111111111111111101110000111101001001.

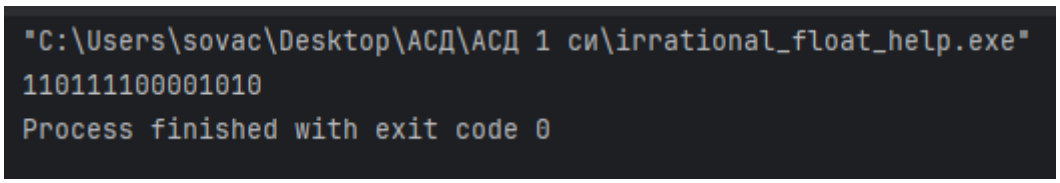
Преобразуем в двоичный код значения, выбранные для СД типа **float массив [3]**. Начнем с элемента с целым значением, 35. Оно положительно, поэтому бит знака будет равняться 0. Пунктом ранее мы уже переводили его в двоичную систему, в ней он записывается как 100011. Чтобы привести его к виду, когда в нем можно будет выделить мантиссу, его нужно разделить на 2^5, тогда он примет вид 1.00011. То есть мантисса числа равна 00011, а чтобы привести ее к реальному значению числа, ее нужно умножить на 2^5. Порядок хранится в смещенном виде, поэтому порядок будет равняться не 5, а 5+127 = 132. Переведем это число в двоичную систему:
132 / 2 = 66 (ост. 0); 66 / 2 = 33 (ост. 0); 33 / 2 = 16 (ост. 1); 16 / 2 = 8 (ост. 0); 8 / 2 = 4 (ост. 0), 4 / 2 = 2 (ост. 0); 2 / 2 = 1 (ост. 0); 1 / 2 = 0 (ост. 1).
Получим число 10000100. В конечном счете вещественное число 35 будет записано в виде 0'10000100'00011000000000000000000.

Перейдем с элементу -49.875. Оно отрицательно, поэтому бит знака будет равняться 1. Переведем в двоичную систему сначала его целую часть уже привычным способом:
49 / 2 = 24 (ост. 1); 24 / 2 = 12 (ост. 0); 12 / 2 = 6 (ост. 0); 6 / 2 = 3 (ост. 0); 3 / 2 = 1 (ост. 1), 1 / 2 = 0 (ост. 1).
Получим последовательность 110001. Для дробной части, наоборот, будем умножать ее на 2, выписывая целые части в порядке получения, пока дробная часть не станет равна 0.
0.875 * 2 = 1.75 (целое 1); 0.75*2 = 1.5 (целое 1), 0.5 * 2 = 1.0 (целое 1).
Таким образом, целиком число запишется как 110001.111 Чтобы привести его к виду, когда в нем можно будет выделить мантиссу, его нужно разделить на 2^5, тогда оно примет вид 1.10001111. То есть мантисса числа равна 10001111, а чтобы привести ее к реальному значению числа, ее нужно умножить на 2^5. Как мы уже выяснили, смещенный порядок для этого значения равен 132 и в двоичной системе представим как 10000100. В конечном итоге вещественное число -49.875 будет записано в виде 1'10000100'10001111000000000000000.

Наконец, перейдем с элементу 299.8675 Оно положительно, поэтому бит знака будет равняться 0. Переведем в двоичную систему сначала его целую часть уже привычным способом:
299 / 2 = 149 (ост. 1); 149 / 2 = 74 (ост. 1); 74 / 2 = 37 (ост. 0); 37 / 2 = 18 (ост. 1); 18 / 2 = 9 (ост. 0), 9 / 2 = 4 (ост. 1); 4 / 2 = 2 (ост. 0); 2 / 2 = 1 (ост. 0), 1 / 2 = 0 (ост. 1).
Получим последовательность 100101011. Для дробной части, наоборот, будем умножать ее на 2, выписывая целые части в порядке получения, пока дробная часть не станет равна 0.
0.8675 * 2 = 1.735 (целое 1); 0.735*2 = 1.47 (целое 1), 0.47 * 2 = 0.94 (целое 0), 0.94 * 2 = 1.88 (целое 1), 0.88 * 2 = 1.76 (целое 1)...
Большинство дробных чисел никогда не приведутся к рациональной дроби в двоичной системе. Чтобы не выполнять 24 (число бит под мантиссу) – 9 (число бит целой части) = 15 умножений вручную, воспользуемся простым кодом:

```
float num = 0.8675;
int digit = 0;
while ((digit < 24 - 9)) {
    num = num * 2.0;
    printf("%d", (int) num);
    if (num > 1.0) {
        num = num - 1.0;
    }
    digit++;
}
```

И получим следующий вывод:



Таким образом, целиком число запишется как 100101011.110111100001010. Чтобы привести его к виду, когда в нем можно будет выделить мантиссу, его нужно разделить на 2^8, тогда оно примет вид 1.00101011110111100001010. То есть мантисса числа равна 00101011110111100001010. а чтобы привести ее к реальному значению числа, ее нужно умножить на 2^8. Смещенный порядок для этого значения равен 8 + 127 = 135. Переведем его в двоичную систему счисления: 135 / 2 = 67 (ост. 1); 67 / 2 = 33 (ост. 1); 33 / 2 = 16 (ост. 1); 16 / 2 = 8 (ост. 0); 8 / 2 = 4 (ост. 0), 4 / 2 = 2 (ост. 0); 2 / 2 = 1 (ост. 0); 1 / 2 = 0 (ост. 1). В двоичной системе порядок представим как 10000111. В конечном итоге вещественное число 299.8675 будет записано в виде 0'10000111'00101011110111100001010.

Саму СД типа float массив [3] можно записать как прямую последовательность этих трех двоичных значений: 0'10000100'00011000000000000000000'1'10000100'10001111000000000000000'0'10000111'00101011110111100001010.

Задание 4:

Зная принципы, по которым формируется двоичное представление значения СД в памяти компьютера, легко выполнить и обратные преобразования, получив значение СД по ее двоичному представлению.

Выполним преобразование двоичных представлений в значения для СД типа **signed char.** По значению 0'0100011 легко понять, что оно представляет положительное число (его первый бит равен 0). Значит, перевод из дополнительного кода в прямой не требуется, и можно сразу получить запись значения в двоичной системе: 100011. $100011_2 = 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 1*32 + 0*16 + 0*8 + 0*4 + 1*2 + 1*1 = 32 + 2 + 1 = 35$.

По двоичному представлению следующего значения, 1'0010001, можно сказать, что оно отрицательно, поскольку первый его бит равен единице. Значит, требуется перевод из дополнительного кода в прямой. Выполняем шаги для перевода из прямого кода в дополнительный в обратном порядке: отнимая от числа 1, получаем 1'0010000, а инвертируя каждый его незнаковый бит, получаем число 1'1101111. $-(1101111)_2 = -(1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0) = -(1*64 + 1*32 + 0*16 + 1*8 + 1*4 + 1*2 + 1*1) = -(64 + 32 + 8 + 4 + 2 + 1) = -111$.

Как видим, значения, полученные по двоичному представлению, совпали со значениями, которые мы кодировали в этом двоичном представлении, то есть кодировка и раскодировка прошла без ошибок.

Почти тот же алгоритм мы применим и для двоичных представлений СД типа **long long int.** Значение, заданное двоичным представлением 0'0000.....11101110000111101001001 положительно, поскольку равен нулю первый бит последовательности. Значит, не требуется перевод из

дополнительного кода в прямой, и мы можем сразу получить двоичную запись числа. Переведем его в шестнадцатеричную систему, последовательно переведя в нее каждую четверку бит с конца числа: $1001_2 = 9_{16}$, $0100_2 = 4_{16}$, $1111_2 = F_{16}$, $0000_2 = 0_{16}$, $0111_2 = 7_{16}$, $111_2 = 7_{16}$, и в конечном итоге двоичное число преобразуется в шестнадцатеричное 770F49. $770F49_{16} = 7*16^5 + 7*16^4 + 0*16^3 + 15*16^2 + 4*16^1 + 9*16^0 = 7*1048576 + 7*65536 + 0*4096 + 15*256 + 4*16 + 9*1 = 7340032 + 458752 + 3840 + 64 + 9 = 7802697$.

По двоичному представлению значения 1'11111....01110000111101001001 можно сказать, что оно отрицательно, поскольку первый бит равен единице. Тогда требуется перевод в прямой код: из числа 1'11111....01110000111101001001 вычитается единица и оно становится равно 1'11111....01110000111101001000, а затем незнаковые биты инвертируются, и число принимает вид 1'00000...10001111000010110111. Переведем значимую часть числа в шестнадцатеричную систему счисления. $0111_2 = 7_{16}$, $1011_2 = B_{16}$, $0000_2 = 0_{16}$, $1111_2 = F_{16}$, $1000_2 = 8_{16}$, и двоичное число преобразуется в шестнадцатеричное -8F0B7. $-8F0B7_{16} = -(8*16^4 + 15*16^3 + 0*16^2 + 11*16^1 + 7*16^0) = -(8*65536 + 15*4096 + 0*256 + 11*16 + 7*1) = -(524288 + 61440 + 176 + 7) = -585911$. В этот раз кодируемое значение снова совпало с результатом раскодировки.

Преобразовывая СД типа **float массив [3]** из двоичного представления к значению, следует сначала разделить двоичный код так, чтобы можно было рассматривать каждое значение отдельно. СД с двоичным представлением 0'10000100'00011000000000000000000'1'10000100'10001111000000000000000'0'10000111'00101011110111100001010 легко разделится на три равных по длине кода, 0'10000100'00011000000000000000000, 1'10000100'10001111000000000000000, 0'10000111'00101011110111100001010. Процесс преобразования дробных чисел, являющихся элементами этой СД, в двоичное представление, форму с плавающей запятой, несколько более сложный. Чтобы не проходить через эту цепочку преобразований в обратном порядке, можно воспользоваться готовыми формами получения значения из двоичного представления дробного числа типа float.

$v = (-1)^s*2^{(e-127)}*1.m$, если $0 < e < 255$

$v = (-1)^s*2^{(126)}*0.m$, если $e = 0$ и $m \neq 0$

$v = (-1)^s$, если $e = 0$ и $m = 0$

$v = (-1)^s*\text{Inf}$, если $e = 255$ и $m = 0$

$v = \text{NaN}$, если $e = 255$ и $m \neq 0$

Анализируя значение 0'10000100'00011000000000000000000, мы можем заметить, что его порядок в двоичной записи равен 10000100 – это значение точно не равно 0 или 255, поэтому применим первую формулу из списка. Для данного случая значение знакового бита s = 0 значение порядка e = $10000100_2 = 1*2^7 + 0*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0 = 1*128 + 0*64 + 0*32 + 0*16 + 0*8 + 1*4 + 0*2 + 0*1 = 128 + 4 = 132$; значение 1.m (мантиссы с целым разрядом) = $1.00011 = 1*2^0 + 0*2^{-1} + 0*2^{-2} + 0*2^{-3} + 1*2^{-4} + 1*2^{-5} = 1*1 + 0*1/2 + 0*1/4 + 0*1/8 + 1*1/16 + 1*1/32 = 1 + 0.0625 + 0.03125 = 1.09375$. Вставив эти значения в формулу, получим $(-1)^0*2^{(132-127)}*1.09375 = 1*2^5*1.09375 = 32*1.09375 = 35$.

Для следующего значения, 1'10000100'10001111000000000000000, также справедливо, что порядок, в двоичной системе тоже записываемый как 10000100, не равен ни 0, ни 255, и к этому представлению применима первая формула. Для данного случая значение знакового бита s = -1; значение порядка e = 10000100_2 = 132 (повторяет значение порядка для примера выше); значение 1.m (мантиссы с целым разрядом) = 1.10001111 = $1*2^0 + 1*2^{-1} + 0*2^{-2} + 0*2^{-3} + 0*2^{-4} + 1*2^{-5} + 1*2^{-6} + 1*2^{*-7} + 1*2^{-8}$ = 1*1 + 1*1/2 + 0*1/4 + 0*1/8 + 0*1/16 + 1*1/32 + 1*1/64 + 1*1/128 + 1*1/256 = 1 + 0.5 + 0.03125 + 0.015625 + 0.0078125 + 0.00390625 = 1.55859375. Вставив эти значения в формулу, получим $(-1)^1*2^{(132 - 127)}$*1.55859375 = $-1*2^5$*1.55859375 = -32*1.55859375 = -49.875.

Наконец, для последнего значения, 0'10000111'00101011110111100001010, порядок в двоичной записи выражен как 10000111, что точно не равно 0 или 255, и мы все так же имеем право применить первую формулу. В этом случае s = 0; e = 10000111_2 = $1*2^7 + 0*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0$ = 1*128 + 0*64 + 0*32 + 0*16 + 0*8 + 1*4 + 1*2 + 1*1 = 128 + 4 + 2 + 1 = 135; значение 1.m (мантиссы с целым разрядом) = 1.00101011110111100001010. Ручной перевод этого числа в десятичную форму будет слишком громоздким, а машинный может привести к значительной потере знаков. Поэтому, помня, что для вычисления значения нам потребуется выполнить умножение $(-1)^0*2^{(135 - 127)}$*1.m, выполним эти дейтсвия без перевода в десятичную систему. $(-1)^0*2^{(135 - 127)}$*1.m = $1*2^8$*1.m = 2^8*1.m = 2^8*$1.00101011110111100001010_2$ = $100101011.110111100001010_2$. Это число уже можно попробовать перевести в десятичную систему. Его целую часть переведем вручную: 100101011_2 = $1*2^8 + 0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$ = 1*256 + 0*128 + 0*64 + 1*32 + 0*16 + 1*8 + 0*4 + 1*2 + 1*1 = 256 + 32 + 8 + 2 +1 = 299. Дробную часть числа переведем в десятичную систему с помощью простой программы.

```
char *bin_string = "110111100001010";
float value = 0;
float cur_digit_value = 0.5;
while (*bin_string != '\0') {
    if (*bin_string - '0') {
        value += cur_digit_value;
    }
    cur_digit_value /= 2;
    bin_string += 1;
}
printf("\n%f", value);
```

Ее вывод будет следующим:

```
"C:\Users\sovac\Desktop\АСД\АСД 1 си\notmain.exe"

0.867493
Process finished with exit code 0
```

То есть итоговое значение элемента с данным двоичным представлением будет равно 299.867493.

Как видим, для типа float массив [3] все значения, кроме последнего, точно совпали с исходными: последнее отклоняется от исходного не более чем на 0.0001, что является нормой для типа float, представляющего числа только примерно.

Задание 5:

Для начала реализуем данные функции следующим образом. Функция PrintByte выводит на экран двоичное представление переменной типа unsigned char, целого числа без знака, занимающего один байт, получая доступ к значениям разрядов, от старшего к младшему, с помощью побитовых операций. Следующая функция, PrintVar, преобразует переменную произвольного типа к массиву значений unsigned char и выводит двоичное представление переменной на экран, применяя функцию PrintByte для каждого байта этой переменной.

```
//Выводит на экран двоичное представление числа типа unsigned
char, целого числа в диапазоне от 0 до 255
void PrintByte(unsigned char a) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", a>>i & 1);
    }
    printf(" ");
}
```

```
//Выводит на экран двоичное представление переменной по адресу a
произвольного типа, занимающей size байт
void PrintVar(void *a, unsigned int size) {
    unsigned char *a_in_chars = (unsigned char *) a;
    for (int i = 0; i < size; i++) {
        PrintByte(a_in_chars[i]);
    }
}
```

Задание 6:

Создадим переменные типов, соответствующих рассматриваемым нами СД: **signed char, long long int, float[3]**, и присвоим им значения, которые мы выбрали для изучения физического уровня в задании 2. Затем остается только вызвать функцию PrintVar для этих переменных и сравнить выведенные на экран двоичные представления с полученными вручную.

```c
int main() {
    signed char num1 = 35;
    signed char num2 = -111;
    long long big_num1 = 7802697;
    long long big_num2 = -585911;
    float array[3] = {35, -49.875, 299.8675};
    PrintVar(&num1, sizeof(signed char));
    printf("\n");
    PrintVar(&num2, sizeof(signed char));
    printf("\n");
    PrintVar(&big_num1, sizeof(long long));
    printf("\n");
    PrintVar(&big_num2, sizeof(long long));
    printf("\n");
    PrintVar(&array, sizeof(float[3]));
    printf("\n");
    return 0;
}
```

```
"C:\Users\sovac\Desktop\АСД\АСД 1 си\bin_input_output.exe"
00100011
10010001
01001001 00001111 01110111 00000000 00000000 00000000 00000000 00000000
01001001 00001111 11110111 11111111 11111111 11111111 11111111 11111111
00000000 00000000 00001100 01000010 00000000 10000000 01000111 11000010 00001010 11101111 10010101 01000011

Process finished with exit code 0
```

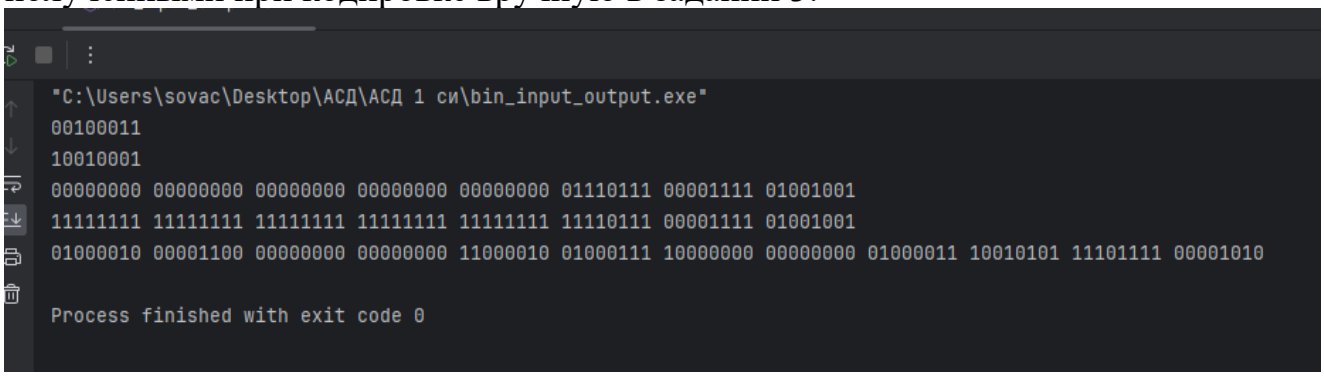
Как мы видим, правильно оказались переведены только значения типа signed char, занимающие ровно 1 байт. Остальные двоичные коды на первый взгляд не имеют ничего общего с кодами, полученными в задании 3, но можно заметить, что коды на экране могут быть получены из кодов задания 3, если расположить их байты в обратном порядке, не инвертируя при этом каждый из них внутри. Причем для массива байты каждого элемента расположены в обратном порядке, но порядок элементов также не нарушен. Значит, для исправления этого недостатка следует модифицировать программу из пункта 5, а именно: 1) выводить на экран байты от конца приведенного массива типа unsigned char к началу, а не наоборот; 2) создать отдельную функцию для вывода массива заданной длины и с заданным размером базового типа, чтобы обеспечить вывод элементов в правильном порядке (этот порядок нарушится после внесения правки из пункта 1).

Получим следующие варианты функций:

```c
//Выводит на экран двоичное представление числа типа unsigned char, целого числа в диапазоне от 0 до 255
void PrintByte(unsigned char a) {
```

```
    for (int i = 7; i >= 0; i--) {
        printf("%d", a>>i & 1);
    }
    printf(" ");
}
//Выводит на экран двоичное представление переменной по адресу a
произвольного типа, занимающей size байт
void PrintVar(void *a, unsigned int size) {
    unsigned char *a_in_chars = (unsigned char *) a;
    for (int i = size - 1; i >= 0; i--) {
        PrintByte(a_in_chars[i]);
    }
}
//Выводит на экран двоичное представление массива длиной
array_size по адресу a из элементов произвольного типа, каждый
экземпляр которого имеет размер size байт
void PrintArray(void *a, unsigned int type_size, unsigned int
array_size) {
    for (int i = 0; i < array_size; i++) {
        PrintVar(a + i*type_size, type_size);
    }
}
```

Теперь можем повторно запустить программу и сравнить результаты с кодами, полученными при кодировке вручную в задании 3:



В этот раз коды совпадают.

Задание 7:

Реализуем функцию GetVar, которая по заданному адресу записывает значение СД, имеющее двоичное представление, представленное заданной строкой.

Примечание: будем считать, что поданная на вход строка содержит в себе незначащие нули и единицы, при их наличии.

Алгоритм имеет следующие ключевые шаги: 1) создается переменная для того, чтобы отслеживать движение по строке; 2) с помощью вспомогательной функции вычисляется длина строки, и на ее основе (путем деления на 8, количество бит в байте) вычисляется длина массива байтов; 3) перед вычислением реальных значений байтов, каждый из них инициализируется нулем; 4) проходя по строке

от конца к началу, вычисляются значения соответствующих байтов (от начала к концу массива): отдельные биты обращаются в 1 или 0 в зависимости от рассматриваемого символа строки с помощью побитовых операций; 6) после получения массива байтов, он копируется по указанному адресу переменной произвольного типа.

Функция для записи реального значения массива, GetArray, будет иметь такую же структуру. Единственное различие заключается в том, что после получения массива байтов он копируется по адресу массива не целиком, а по элементам, чтобы не нарушить их порядок. Для этого группы байтов, идущие в начале массива, должны записываться в элемент с конца. Для того, чтобы это реализовать, уже необходимо знать размер базового элемента.

```
//Возвращает длину строки, начинающейся по адресу begin
size_t strlen_(char *begin) {
    char *end = begin;
    while (*end != '\0') {
        end++;
    }
    return end - begin;
}
```

```
//Записывает в переменную произвольного типа по адресу a значение,
двоичное представление которого записано в строке по адресу s
void GetVar(void *a, char *s) {
    size_t passed_length = 0;
    size_t length = strlen_(s);
    unsigned char s_in_chars[length/8];
    for (int i = 0; i < length/8; i++) {
        s_in_chars[i] = 0;
    }
    while (passed_length < length) {
        s_in_chars[passed_length / 8] += (s[length - passed_length
- 1] - '0') << passed_length % 8;
        passed_length++;
    }
    memcpy(a, s_in_chars, length/8 + (length % 8 != 0));
}
```

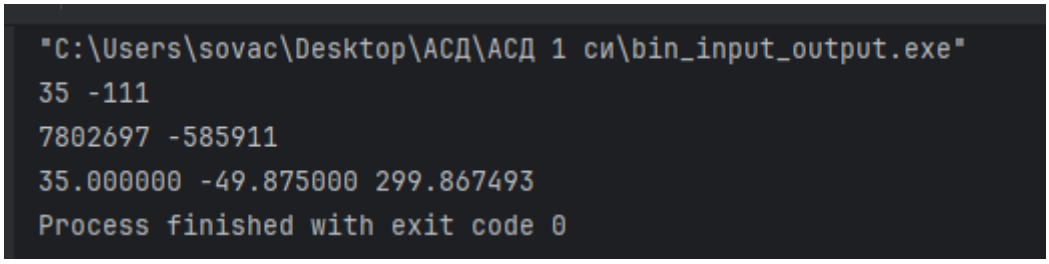
```
//Записывает в массив произвольного типа, каждый экземпляр
которого имеет размер type_size байт, по адресу a значение,
двоичное представление которого записано в строке по адресу s
void GetArray(void *a, char *s, unsigned int type_size) {
    size_t passed_length = 0;
    size_t length = strlen_(s);
    unsigned char s_in_chars[length/8];
    for (int i = 0; i < length/8; i++) {
        s_in_chars[i] = 0;
```

```
    }
    while (passed_length < length) {
        s_in_chars[passed_length / 8] += (s[length - passed_length
- 1] - '0') << passed_length % 8;
        passed_length++;
    }
    for (int i = 0; i < length / (8*type_size); i++) {
        memcpy(a + (length / (8*type_size) - i - 1)*type_size,
s_in_chars + i*type_size, type_size);
    }
}
```

Задание 8:

После того, как функции для получения значения по двоичному представлению были заданы, нам остается только создать строковые переменные, хранящие двоичное представление для всех примеров рассматриваемых СД, а также переменные, куда будет записываться реальное их значение, и вызвать функцию для каждой из пяти пар «строка» – «переменная» (а также помнить, что для массива нужно вызвать отдельную функцию). Потом остается только вывести на экран полученные значения.

```
int main() {
    char *charstring1 = "00100011";
    char *charstring2 = "10010001";
    char *longlongstring1 =
"00000000000000000000000000000000000000000000000011101110000111101001001";
    char *longlongstring2 =
"11111111111111111111111111111111111111111111111011100001111010010001";
    char *floatarraystring =
"01000010000011000000000000000000011000010010001111000000000000000010
00001110010101111011110000101010";
    char char1, char2;
    long long long1, long2;
    float array2[3];
    GetVar(&char1, charstring1);
    GetVar(&char2, charstring2);
    GetVar(&long1, longlongstring1);
    GetVar(&long2, longlongstring2);
    GetArray(array2, floatarraystring, sizeof(float));
    printf("%d %d \n", char1, char2);
    printf("%lld %lld \n", long1, long2);
    printf("%f %f %f", array2[0], array[1], array2[2]);
    return 0;
}
```



Значения (за исключением последнего значения в массиве дробных чисел) совпадают с теми значениями, которые мы выбрали в задании 2, и теми, которые вручную получили при раскодировке в задании 4.

Вывод:

В ходе лабораторной работы получили опыт работы со встроенными СД языка С, кратко описали их на абстрактном, физическом и логическом уровнях, более подробно исследовали физическое представление в памяти компьютера, перевели их в двоичную форму как вручную, так и с помощью самостоятельно написанных функций на том же языке.