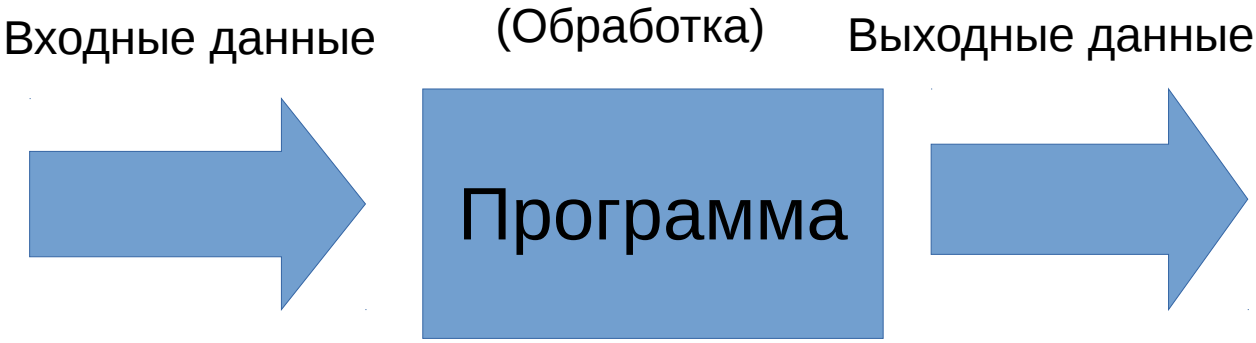
Типы данных в Си

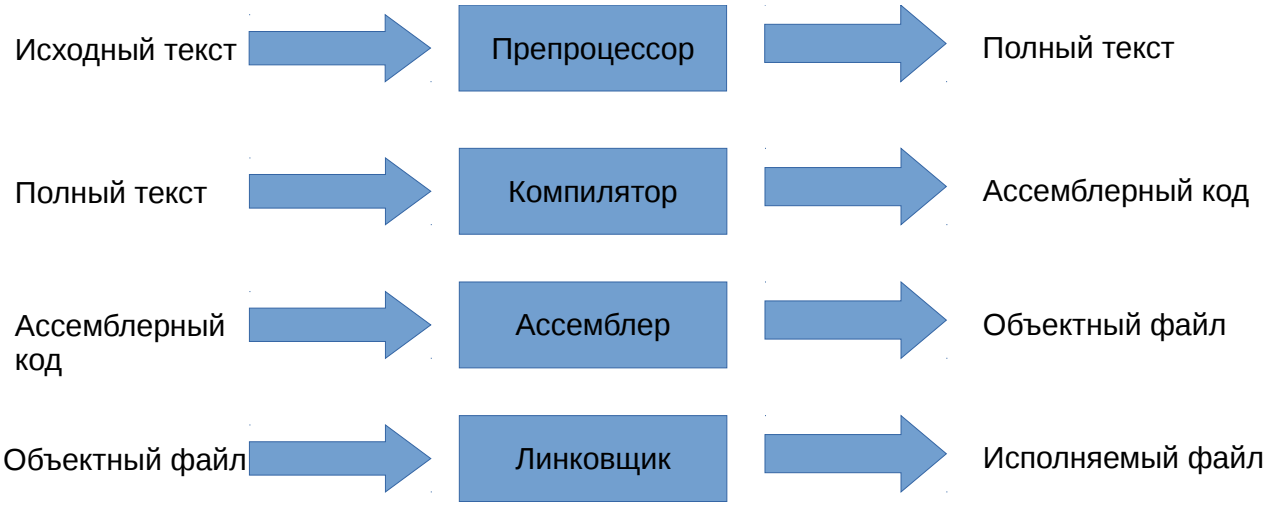
Любая программа в компьютере в том или ином виде осуществляют обработку данных.



- Опции и их значения
- Конфигурационные файлы
- Графический ввод
- Ввод с терминала

- Текстовый вывод
- Вывод в файл
- Вывод в графический интерфейс
- Запись в память (сеть)

Уже знакомую нам программу — компилятор gcc можно также представить в виде совокупности программ, работающих как конвеер, но каждая их которых имеет собственные входные и выходные данные.



При написании программ на языке Си мы также должны принимать какие-то данные, обрабатывать и выдавать на выход. В процессе работы мы также должны осуществлять временное хранение. Для временного хранения каких либо значений в языках высокого уровня существует абстракция, называемая переменная.

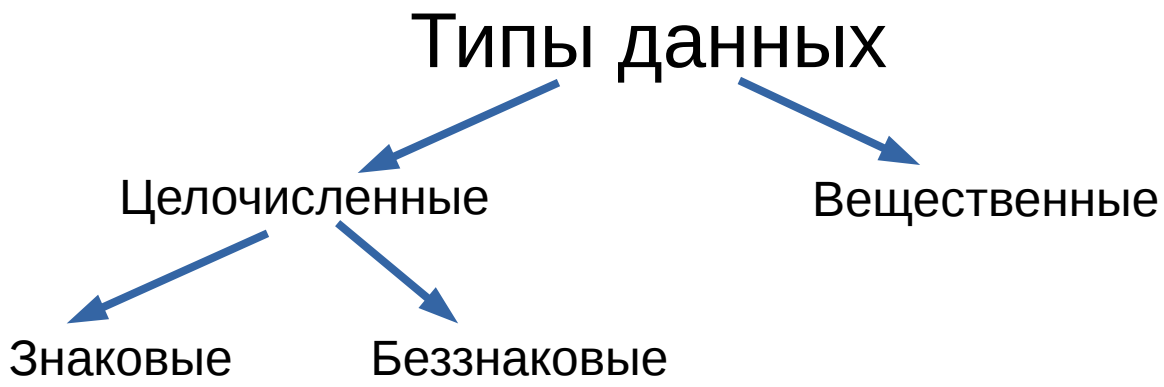
Определим переменную как именованную область памяти, хранящую определенное значение. Помимо значения в типизированных языках (Си — типизированный язык программирования) переменная имеет некоторый тип. Так что же такое тип?

Для компьютера любые входные и выходные данные — есть совокупность байт.

Чтобы упростить разработку программ и сделать программы более наглядными в языках высокого уровня существует понятие типов данных. Рассмотрим самые простые данные —

числовые.

Итак, в Си, числовые типы данных можно разделить на целочисленные и вещественные.



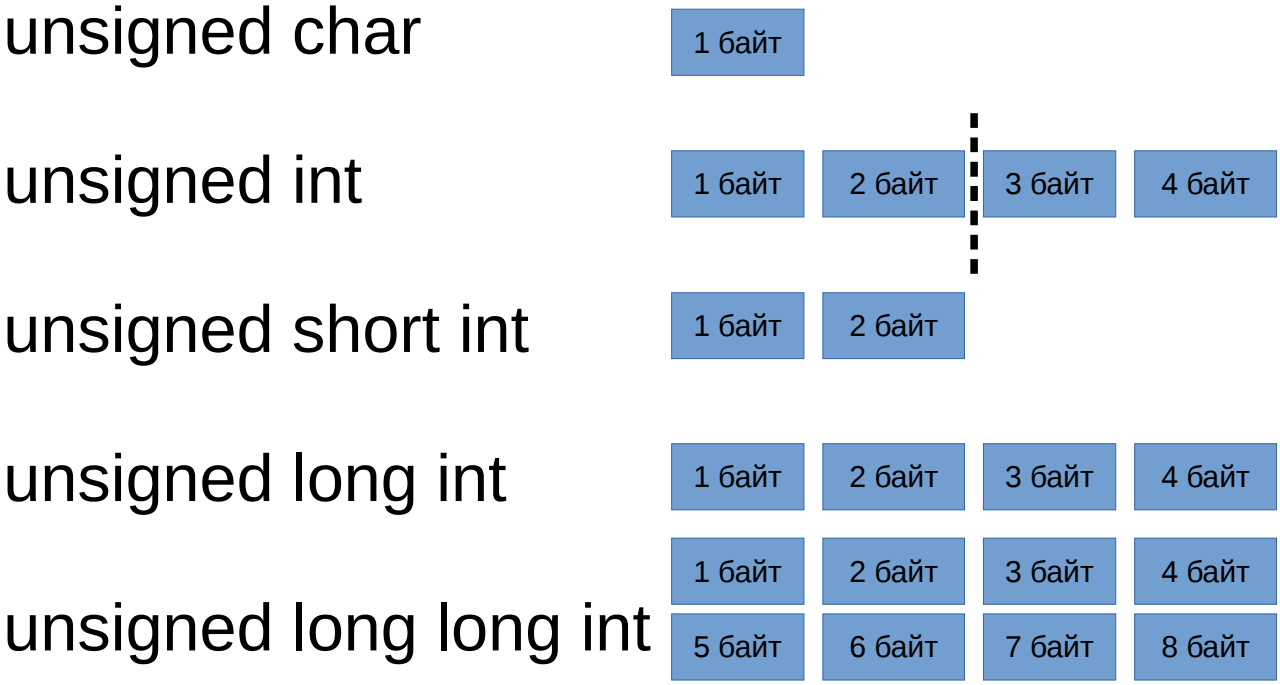
Целочисленные в свою очередь можно разделить на знаковые и беззнаковые. Начнем рассмотрение представление этих типов данных в компьютере с беззнаковых целых чисел.

Целочисленные данные

Самым простым целочисленным беззнаковым типом является unsigned char. Представляет собой двоичное представление числа, ограниченное 1 байтом (8-ю битами).

Далее типы идут в порядке увеличения байт, необходимых для хранения их значения. Самое большое unsigned long long int занимает в памяти 8 байт.

Отдельно следует отметить тип unsigned int, поскольку нельзя однозначно утверждать, что для его хранения в памяти необходимо 4 байта. Существуют архитектуры, где для хранения целочисленного числа (int, unsigned int) выделяется всего 2 байта.



Чтобы избежать путаницы с размерами (а значит и с максимально возможным значением), а также сократить запись определения типа (unsigned long long int — целых четыре слова!) в стандартной библиотеке Си существует заголовочный файл stdint.h который определяет

целочисленные типы с явным указанием размера.

Тип	Размер	Максимальное значение
uint8_t	1 byte	2^8 - 1 =255
uint16_t	2 bytes	2^16 - 1 = 65535
uint32_t	4 bytes	2^32 - 1 = 4294967295
uint64_t	8 bytes	2^64 -1 = Очень много

Такой подход значительно удобнее, когда необходимо точно знать о максимальном возможном значении для переменной. Также явное указание делает код наглядным, что бывает важно при работе с протоколами передачи данных, где размеры передаваемых или хранимых данных строго фиксированы.

Небольшой пример для вывода количества занимой памяти переменной (можно поэкспериментировать и со стандартными типами int, long int, short int):

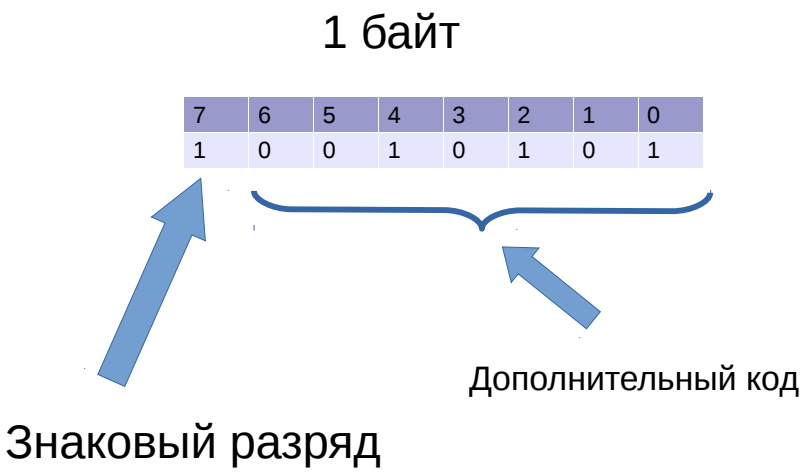
```
#include <stdio.h>
#include <stdint.h>

int main()
{
    uint8_t t1;
    uint16_t t2;
    uint32_t t3;
    uint64_t t4;

    printf("size of uint8_t = %ld\n", sizeof(t1));
    printf("size of uint16_t = %ld\n", sizeof(t2));
    printf("size of uint32_t = %ld\n", sizeof(t3));
    printf("size of uint64_t = %ld\n", sizeof(t4));

    return 0;
}
```

Со знаковыми типами ситуация несколько сложнее, но не намного. Для представления числа со знаком старший бит числа обозначается как знаковый. Если его значение равно 1, то число отрицательное, если 0, то положительное.



Остальная часть числа представляется в дополнительном коде. Дополнительный код был придуман для оптимизации вычислений в ЭВМ, а именно чтобы заменить операцию вычитания, операцией сложения. В случае, если знаковый разряд равен нулю, то есть число положительное представление числа в прямом коде совпадает с представлением числа в дополнительном коде. Однако для отрицательных числе представление будет различаться. Пример для демонстрации.

```
#include <stdio.h>
#include <stdint.h>

int main()
{
    uint8_t test = 255;

    int8_t test2 = 255;

    printf("test1 value = %u\n", test);
    printf("test2 value = %d\n", test2);

    return 0;
}
```

test2 на выходе будет иметь значение -1, несмотря на то, что в двоичном виде будет представляться, как 1111 1111.

Ответ кроется в представлении числа -1 в дополнительном коде. Алгоритм следующий. Чтобы представить число в дополнительном коде сперва необходимо установить знаковый разряд в 1, а число записать в прямом коде

1000 0001.

После чего инвертировать все разряды кроме знакового (чтобы сохранить знак).

1111 1110.

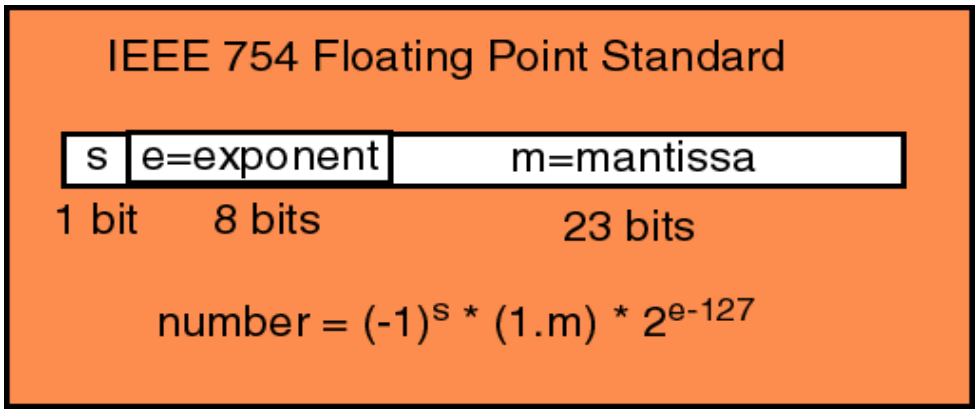
После чего к полученному результату прибавить 1.

111 1111 = -1 (для знаковых однобайтовых числел) = 255 (для беззнаковых).

Более подробно можно почитать тут (есть русская версия):

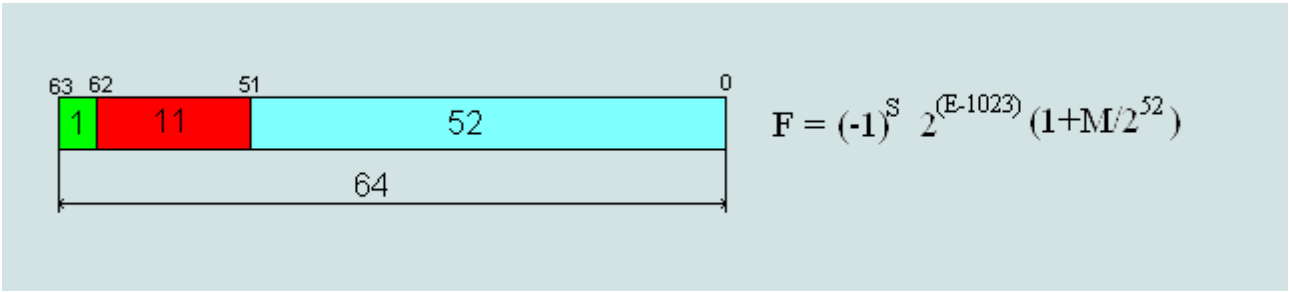
Вещественные числа

Вещественные числа или числа с плавающей запятой имеют несколько более сложное представление. Оно определено в стандарте IEEE 754 и имеет следующий вид (одинарная точность 32 бита, тип float).



Диапазон для числел одинарной точности от $3.4*10^{-38}$(наименьшее представляемое значение) до $3.4*10^{38}$ (наибольшее представляемое значение)

Вещественные числа двойной точности занимают 8 байт и представляются следующим образом



Диапазон для числел с двойной точностью от $1.7 * 10^{-308}$ до $1.7 * 10^{308}$.

Существует так же формат с повышенной точностью, для его хранения необходимо 80 бит.

Более подробную информацию про стандарт, включая примеры можно найти тут.

Переменные

При работе с переменными всегда необходимо выполнять две операции: объявление и определение. При объявлении мы сообщаем компилятору, что собираемся использовать область памяти с определённым типом и с определенным именем. При определении мы задаём конкретное значение для переменной или выражение для его вычисления (компилятор при этом проверяет совместимость типов с левой и правой стороны выражения и возможно осуществляет неявное преобразование типов, если это возможно).

```
int test; //Объявление переменной test
int a,b; //Объявление переменных a и b;
double c = 1.0; //Объявление переменной c, опеределение её значением 1.0
double d = c + 42.2; // Объявлением переменной d, опеределение её выражением c + 42.2
```

```
secret = 3; //ОШИБКА. Определение переменной secret, которая не была ранее объявлена.
test = "bla bla bla"; //ОШИБКА, Определение целочисленной переменной test строкой.
```

Помимо объявления и определения с переменные можно осуществлять следующие операции

1. Арифметические операции
    1. Сложение

        ```
        int a = 4;
        int b = a + 2; // b = 6
        ```
    2. Вычитание

        ```
        int a = 4;
        int b = 6 – a; // b = 2
        ```
    3. Умножение

        ```
        int a = 4;
        int b = 6 * a; // b = 24
        ```
    4. Деление (для целочисленных чисел результат - получение целой части, для вещественных — число с плавающей запятой).

    ```
    int num = 3, den = 2;
    int result = num / den; //result = 1

    double a = 3, b = 2;
    double c = a / b; // c = 1.500000
    ```

    5. Получение остатка (для целочисленных чисел)

    ```
    int num = 5, den = 3;
    int result = num % den; //result = 2
    ```

2. Операции сравнения
    1. Больше >
    2. Меньше <
    3. Больше или равно >=
    4. Меньше или равно <=
    5. Равно ==
3. Логические операции
    1. И &&
    2. ИЛИ ||
    3. НЕ !

Поскольку операции сравнения и логические операции возвращают значение истина или ложь (1 или 0) рассмотривать примеры их применения будет удобно при рассмотрении операторов ветвления и циклов.

Форматный вывод данных

Важной частью решения любой задачи является представление результатов.

Форматирование вывода в программах часто выполняется с помощью функции printf.

В общем виде использование функции printf можно представить в следующем виде

```
int printf("СТРОКА С УКАЗАНИЕМ ФОРМАТОВ", ЗНАЧЕНИЕ1, ЗНАЧЕНИЕ2, ...);
```

int — возвращаемое значение функции нас пока интересовать не будет (оно сообщает сколько символов было выведено на стандартный вывод).

В форматной строке указываются спецификаторы выводимых данных со знаком %, а также могут выводиться некоторые литеральные символы (текстовые строки).

После форматной строки через запятую указываются переменные или выражения соответствующие порядку указания спецификаторов.

Пример:

```
int a = 42;
double b = 3.0
printf("value a = %d value b = %f \n", a, b);
```

Здесь %d — спецификатор целочисленного значения, %f — числа с плавающей запятой. a,b — выводимые значения.

Более подробную информацию, включая описания спецификаторов можно получить выполнив команду:

```
$ man 3 printf
```

Форматный ввод данных

Форматный ввод осуществляется аналогично выводу при помощи функции scanf

```
int scanf("СТРОКА С ФОРМАТАМИ", &ПЕРЕМЕННАЯ1, &ПЕРЕМЕННАЯ2, ...);
```

Основное отличие здесь состоит в указании знака амперсанда «&» перед именами переменных значение которых считывается со стандартного вывода. Это связано с особенностью передачи переменных в функции и будет рассмотрена позднее.

Вводимые значения по умолчанию разделяются знаками пробелов или возврата каретки (enter). Явно задать разделитель можно в форматной строке.

```
int a, b;
scanf("%d,%d", &a, &b); //Ожидается ввод: ЧИСЛО,ЧИСЛО<ENTER>
```

Так же более подробную информацию можно получить с помощью команды:

```
$ man 3 scanf
```

Пример программы, осуществляющей ввод и вывод двух чисел.

```
#include <stdio.h>

int main()
{
    int a, b;

    printf("Enter value a, b: \n");

    scanf("%d%d",&a, &b);

    printf("Entered a = %d, b = %d \n", a, b);

    return 0;
}
```