Основы алгоритмизации и программирования

Лабораторная работа №2.

Цель работы - изучить типы данных, определение переменных и операции в языке программирования Си.

Содержание отчёта:

1. Титульный лист
2. Формулировка цели и задач работы
3. Описание результатов выполнения
4. Выводы, согласованные с целью

Описание языка программирования Си

Си —универсальный язык программирования. Он тесно связан с системой UNIX, так как был разработан в этой системе, которая как и большинство программ, работающих в ней, написаны на Си. Однако язык не привязан жестко к какой-то одной операционной системе или машине. Хотя он и назван "языком системного программирования", поскольку удобен для написания компиляторов и операционных систем, оказалось, что на нем столь же хорошо писать большие программы другого профиля.

Многие важные идеи Си взяты из языка BCPL, автором которого является Мартин Ричарде. Влияние BCPL на Си было косвенным —через язык B, разработанный Кеном Томпсоном в 1970 г. для первой системы UNIX, реализованной на PDP-7.

BCPL и B —"бестиповые" языки. В отличие от них Си обеспечивает разнообразие типов данных. Базовыми типами являются символы, а также целые и числа с плавающей точкой различных размеров. Кроме того, имеется возможность получать целую иерархию производных типов данных из указателей, массивов, структур и объединений. Выражения формируются из операторов и операндов. Любое выражение, включая присваивание и вызов функции, может быть инструкцией. Указатели обеспечивают машинно-независимую адресную арифметику.

В Си имеются основные управляющие конструкции, используемые в хорошо структурированных программах: составная инструкция ({...}), ветвление по условию (if-else), выбор одной альтернативы из многих (switch), циклы с проверкой наверху (while, for) и с проверкой внизу (do), а также средство прерывания цикла (break).

В качестве результата функции могут возвращать значения базовых типов, структур, объединений и указателей. Любая функция допускает рекурсивное обращение к себе. Как правило, локальные переменные функции —"автоматические", т. е. они создаются заново при каждом обращении к ней. Определения функций нельзя вкладывать друг в друга, но объявления переменных разрешается строить в блочно-структурной манере. Функции программы на Си могут храниться в отдельных исходных файлах и компилироваться независимо. Переменные по отношению к функции могут быть внутренними и внешними. Последние могут быть доступными в пределах одного исходного файла или всей программы.

На этапе препроцессирования выполняется макроподстановка в текст программы, включение других исходных файлов и условная компиляция.

Си —язык сравнительно "низкого уровня". Однако это вовсе не умаляет его достоинств, просто Си имеет дело с теми же объектами, что и большинство компьютеров, т.е. с символами, числами и адресами. С ними можно оперировать при помощи арифметических и логических операций, выполняемых реальными машинами.

В Си нет прямых операций над составными объектами, такими как строки символов, множества, списки и массивы. В нем нет операций, которые бы манипулировали с целыми массивами или строками символов, хотя структуры разрешается копировать целиком как единые объекты. В языке нет каких-либо средств распределения памяти, помимо возможности определения статических переменных и стекового механизма при выделении места для локальных переменных внутри функций. Нет в нем "кучи" и "сборщика мусора". Наконец, в самом Си нет средств ввода-вывода, инструкций READ(читать) и WRITE(писать) и каких-либо методов доступа к файлам. Все это —механизмы высокого уровня, которые в Си обеспечиваются исключительно с помощью явно вызываемых функций. Большинство реализованных Си-систем содержат в себе разумный стандартный набор этих функций.

В продолжение сказанного следует отметить, что Си предоставляет средства лишь последовательного управления ходом вычислений: механизм ветвления по условиям, циклы, составные инструкции, подпрограммы —и не содержит средств мультипрограммирования, параллельных процессов, синхронизации и организации сопрограмм.

Отсутствие некоторых из перечисленных средств может показаться серьезным недостатком ("выходит, чтобы сравнить две строки символов, нужно обращаться к функции?"). Однако компактность языка имеет реальные выгоды. Поскольку Си относительно мал, то и описание его кратко, и овладеть им можно быстро. Программист может реально рассчитывать на то, что он будет знать, понимать и на практике регулярно пользоваться всеми возможностями языка.

В течение многих лет единственным определением языка Си было первое издание книги "Язык программирования Си". В 1983 г. Институтом американских национальных стандартов (ANSI) учреждается комитет для выработки современного исчерпывающего определения языка Си. Результатом его работы явился стандарт для Си ("ANSI-C"), выпущенный в 1988 г. Большинство положений этого стандарта уже учтено в современных компиляторах.

Стандарт базируется на первоначальном справочном руководстве. По сравнению с последним язык изменился относительно мало. Одной из целей стандарта было обеспечить, чтобы в большинстве случаев существующие программы оставались правильными или вызывали предупреждающие сообщения компиляторов об изменении поведения.

Для большинства программистов самое важное изменение —это новый синтаксис объявления и определения функций. Объявление функции может теперь включать и описание ее аргументов. В соответствии с этим изменился и синтаксис определения функции. Дополнительная информация значительно облегчает компилятору выявление ошибок, связанных с несогласованностью аргументов; по нашему мнению, это очень полезное добавление к языку.

Следует также отметить ряд небольших изменений. В языке узаконены присваивание структур и перечисления, которые уже некоторое время широко используются. Вычисления с плавающей точкой теперь допускаются и с одинарной точностью. Уточнены свойства арифметики, особенно для беззнаковых типов. Усовершенствован препроцессор. Большинство программистов эти изменения затронут очень слабо.

Второй значительный вклад стандарта —это определение библиотеки, поставляемой вместе с Си-компилятором, в которой специфицируются функции доступа к возможностям операционной системы (например чтения-записи файлов), форматного ввода-вывода, динамического выделения памяти, манипуляций со строками символов и т. д. Набор стандартных заголовочных файлов обеспечивает единообразный доступ к объявлениям функций и типов данных. Гарантируется, что программы, использующие эту библиотеку при взаимодействии с операционной системой, будут работать также и на других машинах. Большинство программ, составляющих библиотеку, созданы по образу и подобию "стандартной библиотеки ввода-вывода" системы UNIX. Эта библиотека описана в первом издании книги и широко используется в других системах. И здесь программисты не заметят существенных различий.

Так как типы данных и управляющих структур языка Си поддерживаются командами большинства существующих машин, исполнительная система (run-timelibrary), обеспечивающая независимый запуск и выполнение программ, очень мала. Обращения к библиотечным функциям пишет сам программист (не компилятор), поэтому при желании их можно легко заменить на другие. Почти все программы, написанные на Си, если они не касаются каких-либо скрытых в операционной системе деталей, переносимы на другие машины.

Си соответствует аппаратным возможностям многих машин, однако он не привязан к архитектуре какой-либо конкретной машины. Проявляя некоторую дисциплину, можно легко писать переносимые программы, т. е. программы, которые без каких-либо изменений могут работать на разных машинах. Стандарт предоставляет возможность для явного описания переносимости с помощью набора констант, отражающих характеристики машины, на которой программа будет работать.

Си не является "строго типизированным" языком, но в процессе его развития контроль за типами был усилен. В первой версии Си хоть не одобрялся, но разрешался бесконтрольный обмен указателей и целых,что вызывало большие нарекания, но это уже давным-давно запрещено.Согласно стандарту теперь требуется явное объявление или явное указание преобразования, что уже и реализовано в хороших компиляторах. Новый вид объявления функций —еще один шаг в этом направлении. Компилятор теперь предупреждает о большей части ошибок в типах и автоматически не выполняет преобразования данных несовместимых типов.Однако основной философией Си остается то, что программисты сами знают, что делают; язык лишь требует явного указания об их намерениях.

Си, как и любой другой язык программирования, не свободен от недостатков. Уровень старшинства некоторых операторов не является общепринятым, некоторые синтаксические конструкции могли бы быть лучше. Тем не менее,как оказалось, Си — чрезвычайно эффективный и выразительный язык, пригодный для широкого класса задач.

Быстрый старт

Единственный способ выучить новый язык программирования —это писать на нем программы. При изучении любого языка первой, как правило, предлагают написать приблизительно следующую программу:

Напечатать слова hello, world

Си-программа, печатающая "hello, world", выглядит так:

```
#include <stdio.h>

int main(void) {
    printf("hello, world\n");
    return 0;
}
```

Программа на Си, каких бы размеров она ни была, состоит из функций и переменных. Функции содержат инструкции, описывающие вычисления, которые необходимо выполнить, а переменные хранят значения, используемые в процессе этих вычислений. Функции в Си похожи на подпрограммы и функции Фортрана или на процедуры и функции Паскаля. Приведенная программа — это функция с именем main. Обычно вы вольны придумывать

любые имена для своих функций, но "main" —особое имя: любая программа начинает свои вычисления с первой инструкции функции main.

Обычно main для выполнения своей работы пользуется услугами других функций; одни из них пишутся самим программистом, а другие берутся готовыми из имеющихся в его распоряжении библиотек. Первая строка программы:

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

сообщает компилятору, что он должен включить информацию о стандартной библиотеке ввода-вывода. Эта строка встречается в начале многих исходных файлов Си-программ.

Один из способов передачи данных между функциями состоит в том, что функция при обращении к другой функции передает ей список значений, называемых аргументами. Этот список берется в скобки и помещается после имени функции. В нашем примере main определена как функция, которая не ждет никаких аргументов, что отмечено пустым списком (void).

Перед функцией main стоит int. Это указывает на то, что функция main возвращает в качестве результата своей работы какое-то целое число. Этим целым числом является код выполнения программы. Если код равен нулю, то принято считать, что программа завершилась успешно (без ошибок). Но если же код не равен нулю (допустим = 2), то говорят, что программа была завершена с ошибкой. В конце функции main инструкция return 0 как раз говорит о том, чтобы функция main вернула 0, как результат успешного исполнения.

Инструкции функции заключаются в фигурные скобки {}. Функция main содержит две инструкции: printf("hello, world\n") и return 0.

Функция вызывается по имени, после которого, в скобках, указывается список аргументов. Таким образом, приведенная выше строка —это вызов функции printf с аргументом "hello, world\n". Функция printf—это библиотечная функция (из библиотеки <stdio.h>), которая в данном случае напечатает последовательность символов, заключенную в двойные кавычки.

Последовательность символов в двойных кавычках, такая как "hello, world\n", называется строкой символов, или строковой константой. Пока что в качестве аргументов для printf и других функций мы будем использовать только строки символов.

В Си комбинация \n внутри строки символов обозначает символ новой строки и при печати вызывает переход к левому краю следующей строки. Если вы удалите \n (стоит поэкспериментировать), то обнаружите, что, закончив печать, машина не переходит на новую строку. Символ новой строки в текстовый аргумент printf следует включать явным образом. Если вы попробуете выполнить, например,

```
printf("hello, world
");
```

компилятор выдаст сообщение об ошибке.

Символ новой строки никогда не вставляется автоматически, так что одну строку можно напечатать по шагам с помощью нескольких обращений к printf. Нашу первую программу можно написать и так:

```
#include <stdio.h>

int main(void) {
    printf("hello, ");
    printf("world");
    printf("\n");
```

```
    return 0;
}
```

В результате ее выполнения будет напечатана та же строка, что и раньше.

Заметим, что \n обозначает только один символ. Такие особые комбинации символов, начинающиеся с обратной наклонной черты, как \n, и называемые эскейп-последовательностями, широко применяются для обозначения трудно представимых или невидимых символов. Среди прочих в Си имеются символы \t, \b, \", \\, обозначающие соответственно табуляцию, возврат на один символ назад ("забой" последнего символа), двойную кавычку, саму наклонную черту. Всё это будет представлено далее.

Типы данных

Переменные и константы являются основными объектами данных, с которыми имеет дело программа. Переменные перечисляются в объявлениях, где устанавливаются их типы и, возможно, начальные значения. Операции определяют действия, которые совершаются с этими переменными. Выражения комбинируют переменные и константы для получения новых значений, Тип объекта определяет множество значений, которые этот объект может принимать, и операций, которые над ними могут выполняться.

Стандартом ANSI было утверждено значительное число небольших изменений и добавлений к основным типам и выражениям. Любой целочисленный тип теперь может быть со знаком, signed, и без знака, unsigned. Предусмотрен способ записи беззнаковых констант и шестнадцатеричных символьных констант. Операции с плавающей точкой допускаются теперь и с одинарной точностью. Введен тип longdouble, обеспечивающий повышенную точность. Строковые константы конкатенируются("склеиваются") теперь во время компиляции. Частью языка стали перечисления (enum), формализующие для типа установку диапазона значений. Объекты для защиты их от каких-либо изменений разрешено помечать как const. В связи с введением новых типов расширены правила автоматического преобразования из одного арифметического типа в другой.

Разумно давать переменным осмысленные имена в соответствии с их назначением, причем такие, чтобы их было трудно спутать друг с другом.Мы предпочитаем короткие имена для локальных переменных, особенно для счетчиков циклов, и более длинные для внешних переменных.

В Си существует всего лишь несколько базовых типов:

1. char—единичный байт, который может содержать один символ из допустимого символьного набора;
2. int—целое, обычно отображающее естественное представление целых в машине;
3. float—число с плавающей точкой одинарной точности;
4. double—число с плавающей точкой двойной точности.

Имеется также несколько квалификаторов, которые можно использовать вместе с указанными базовыми типами. Например, квалификаторы short(короткий) и long(длинный) применяются к целым:

Имеется также несколько квалификаторов, которые можно использовать вместе с указанными базовыми типами. Например, квалификаторы short(короткий) и long(длинный) применяются к целым:

```
short int sh;
long int counter;
```

В таких объявлениях слово int можно опускать, что обычно и делается.

Если только не возникает противоречий со здравым смыслом, short int и long int должны быть разной длины, а int соответствовать естественному размеру целых на данной машине. Чаще всего для представления целого, описанного с квалификатором short, отводится 16 битов, с квалификатором long—32 бита, а значению типа int—или 16, или 32 бита. Разработчики компилятора вправе сами выбирать подходящие размеры, сообразуясь с характеристиками своего компьютера и соблюдая следующие ограничения: значения типов short и int представляются по крайней мере 16 битами; типа long—по крайней мере 32 битами; размер short не больше размера int, который в свою очередь не больше размера long.

Квалификаторы signed(со знаком) или unsigned(без знака) можно применять к типу char и любому целочисленному типу. Значения unsigned всегда положительны или равны нулю и подчиняются законам арифметики по модулю 2^n, где n—количество битов в представлении типа. Так, если значению char отводится 8 битов, то unsigned char имеет значения в диапазоне от 0 до 255, а signed char — от -128 до 127 (в машине с двоичным дополнительным кодом). Являются ли значения типа просто char знаковыми или беззнаковыми, зависит от реализации, но в любом случае коды печатаемых символов положительны.

Тип long double предназначен для арифметики с плавающей точкой повышенной точности. Как и в случае целых, размеры объектов с плавающей точкой зависят от реализации; float, double и long double могут представляться одним размером, а могут двумя или тремя разными размерами.

Константы

Целая константа, например 1234, имеет тип int. Константа типа long завершается буквой l или L, например 123456789L; слишком большое целое, которое невозможно представить как int, будет представлено как long. Беззнаковые константы заканчиваются буквой u или U, а окончание ul или UL говорит о том, что тип константы unsigned long.

Константы с плавающей точкой имеют десятичную точку (123.4), или экспоненциальную часть (1e-2), или же и то и другое. Если у них нет окончания, считается, что они принадлежат к типу double. Окончание f или F указывает на тип float, а l или L —на тип long double.

Целое значение помимо десятичного может иметь восьмеричное или шестнадцатеричное представление. Если константа начинается с нуля, то она представлена в восьмеричном виде, если с 0x или с 0X, то —в шестнадцатеричном. Например, десятичное целое 31 можно записать как 037 или как 0X1F. Записи восьмеричной и шестнадцатеричной констант могут завершаться буквой L (для указания на тип long) и U (если нужно показать,что константа беззнаковая). Например, константа 0XFUL имеет значение 15 и тип unsigned long.

Символьная константа есть целое, записанное в виде символа, обрамленного одиночными кавычками, например 'x'. Значением символьной константы является числовой код символа из набора символов на данной машине. Например, символьная константа '0'в кодировке ASCII имеет значение 48, которое никакого отношения к числовому значению 0 не имеет. Когда мы пишем '0', а не какое-то значение (например,48), зависящее от способа кодировки, мы делаем программу независимой от частного значения кода, к тому же она и легче читается. Символьные константы могут участвовать в операциях над числами точно так же, как и любые другие целые, хотя чаще они используются для сравнения с другими символами.

Некоторые символы в символьных и строковых константах записываются с помощью эскейп-последовательностей, например \n (символ новой строки); такие последовательности изображаются двумя символами, но обозначают один. Кроме того, произвольный восьмеричный код можно задать в виде

'\ooo'
где ooo — одна, две или три восьмеричные цифры (0...7) или
'\xhh'
где hh — одна, две или более шестнадцатеричные цифры (0...9, a...f,A...F). Таким образом, мы могли бы написать

```
#define VTAB'\013'  /* вертикальная табуляция в ASCII*/
#define BELL'\007' /* звонок в ASCII*
```

или в шестнадцатеричном виде:

```
#define VTAB'\xb' /* вертикальная табуляция в ASCII*/
#define BELL'\x7' /* звонок в ASCII*/
```

Полный набор эскейп-последовательностей таков:

\a - сигнал-звонок
\\ - обратная наклонная черта
\b - возврат-на-шаг (забой)
\? - знак вопроса
\f - перевод-страницы
\’ - одиночная кавычка
\n - новая-строка
\” - двойная кавычка
\r - возврат-каретки
\ooo - восьмеричный код
\t - горизонтальная-табуляция
\xhh - шестнадцатеричный код
\v - вертикальная-табуляция

Символьная константа '\0' — это символ с нулевым значением, так называемый символ null (нуль-терминатор). Вместо просто 0 часто используют запись '\0', чтобы подчеркнуть символьную природу выражения, хотя и в том и другом случае запись обозначает нуль.

Объявления переменных

Все переменные должны быть объявлены раньше, чем будут использоваться, при этом некоторые объявления могут быть получены неявно—из контекста. Объявление специфицирует тип и содержит список из одной или нескольких переменных этого типа, как, например, в следующем фрагменте:

```
int lower, upper, step;
char c, line [1000];
```

Переменные можно распределять по объявлениям произвольным образом, так что указанные выше списки можно записать и в следующем виде

```
int lower;
int upper;
int step;
char c;
char line[1000];
```

Последняя форма записи занимает больше места, тем не менее,она лучше, поскольку позволяет добавлять к каждому объявлению комментарий. Кроме того, она более удобна для последующих модификаций.

В своем объявлении переменная может быть инициализирована, как, например:

```
char esc = '\\';
int i = 0;
int limit = MAXLINE+1;
floateps= 1.0e-5;
```

Инициализация неавтоматической переменной осуществляется только один раз — перед тем, как программа начнет выполняться, при этом начальное значение должно быть константным выражением. Явно инициализируемая автоматическая переменная получает начальное значение каждый раз при входе в функцию или блок, ее начальным значением может быть любое выражение. Внешние и статические переменные по умолчанию получают нулевые значения. Автоматические переменные, явным образом не инициализированные, содержат неопределенные значения("мусор").

К любой переменной в объявлении может быть применен квалификатор const для указания того, что ее значение далее не будет изменяться.

```
const double e = 2.71828182845905;
const char msg[] = "предупреждение: ";
```

Применительно к массиву квалификатор const указывает на то, что ни один из его элементов не будет меняться.

Арифметические операторы

Бинарными (т. е. с двумя операндами) арифметическими операторами являются +, -, *, /, а также оператор деления по модулю %. Деление целых сопровождается отбрасыванием дробной части, какой бы она ни была. Выражение дает остаток от деления x на y и, следовательно, нуль, если x делится на y нацело. Например, год является високосным, если он делится на 4, но не делится на 100. Кроме того, год является високосным, если он делится на 400. Следовательно,

```
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
        printf("%d - leap year\n", year);
else
        printf("%d - common year\n", year);
```

Оператор % к операндам типов float и double не применяется. В какую сторону (в сторону увеличения или уменьшения числа) будет усечена дробная часть при выполнении /и каким будет знак результата операции % с отрицательными операндами, зависит от машины.

Бинарные операторы + и - имеют одинаковый приоритет, который ниже приоритета операторов *, /и %, который в свою очередь ниже приоритета унарных операторов + и -. Арифметические операции одного приоритетного уровня выполняются слева направо.

```
int x, y;
x = 5;
y = 10;
printf("%d\n", x + y); // x + y = 15
printf("%d\n", x - y); // x - y = -5
printf("%d\n", x * y); // x * y = 50
printf("%d\n", y / x); // y / x = 2
printf("%d\n", y % x); // x % y = 0
```

Операторы отношения и логические операторы

Операторами отношения являются >, >=, <, <=. Все они имеют одинаковый приоритет. Сразу за ними идет приоритет операторов сравнения на равенство: ==, !=.

Операторы отношения имеют более низкий приоритет, чем арифметические, поэтому выражение вроде i < lim-1 будет выполняться так же, как i < (lim-1), т. е. как мы и ожидаем.

Более интересны логические операторы && и ||. Выражения, между которыми стоят операторы && или ||, вычисляются слева направо. Вычисление прекращается, как только становится известна истинность или ложность результата. Многие Си-программы опираются на это свойство.

По определению численным результатом вычисления выражения отношения или логического выражения является 1, если оно истинно, и 0, если оно ложно.

Унарный оператор ! преобразует ненулевой операнд в 0, а нуль в 1. Обычно оператор ! используют в конструкциях вида

```
if(!valid)
```

что эквивалентно

```
if(valid == 0)
```

Трудно сказать, какая из форм записи лучше. Конструкция вида !valid хорошо читается ("если не правильно"), но в более сложных выражениях может оказаться, что ее не так-то легко понять.

```
int x, y;
x = 5;
y = 10;
printf("%d\n", x == y); // x = y ? НЕТ -> 0
printf("%d\n", x != y); // x != y ? ДА -> 1
printf("%d\n", x < y); // x < y ? ДА -> 1
printf("%d\n", x > y); // x > y ? НЕТ -> 0
printf("%d\n", x <= y); // x <= y ? ДА -> 1
printf("%d\n", x >= y); // x >= y ? НЕТ -> 0
printf("%d\n", x && y); // x && y ? ДА -> 1
printf("%d\n", x || y); // x || y ? ДА -> 1
```

Операторы инкремента и дикремента

В Си есть два необычных оператора, предназначенных для увеличения и уменьшения переменных. Оператор инкремента ++добавляет 1 к своему операнду, а оператор декремента --вычитает 1. Мы уже неоднократно использовали ++ для наращивания значения переменных, как, например, в

```
if(c  ==  '\n')
        ++nl;
```

Необычность операторов ++и --в том, что их можно использовать и как префиксные (помещая перед переменной: ++n), и как постфиксные(помещая после переменной: n++) операторы. В обоих случаях значение n увеличивается на 1, но выражение ++n увеличивает n

до того, как его значение будет использовано, а n++—после того. Предположим, что n содержит 5, тогда

```
x= n++;
```

установит x в значение 5, а

```
x= ++n;
```

установит xв значение 6. И в том и другом случае n станет равным 6. Операторы инкремента и декремента можно применять только к переменным. Выражения вроде (i+j)++ недопустимы.

Если требуется только увеличить или уменьшить значение переменной (но не получить ее значение), как например

```
if(c  ==  '\n')
        ++nl;
```

то безразлично, какой оператор выбрать —префиксный или постфиксный. Но и существуют ситуации, когда требуется оператор вполне определенного типа.

Побитовые операторы

В Си имеются шесть операторов для манипулирования с битами. Их можно применять только к целочисленным операндам, т. е. к операндам типов char, short, int и long, знаковым и беззнаковым.

&—побитовое И.
|—побитовое ИЛИ.
^—побитовое исключающее ИЛИ.
<<—сдвиг влево.
>>—сдвиг вправо.
~—побитовое отрицание (унарный).

Побитовое И (&):

```
int x, y;
x = 5 // 0101
y = 11 // 1011
printf("%d", x & y); // 0101 И 1011 = 0001 = 1
```

Побитовое ИЛИ (||):

```
int x, y;
x = 5; // 0101
y = 11; // 1011
printf("%d", x || y); // 0101 ИЛИ 1011 = 1111 = 15
```

Побитовое Исключающее ИЛИ (^):

```
int x, y;
x = 5; // 0101
y = 11; // 1011
printf("%d", x ^ y); // 0101 ИЛИ 1011 = 1110 = 14
```

Сдвиг влево (<<):

```
int x;
x = 5; // 0101
printf("%d", x << 1); // 0101 << 1 = 1010 = 10
```

Сдвиг вправо (>>):

```
int x;
x = 5; // 0101
printf("%d", x >> 1); // 0101 >> 1 = 0010 = 2
```

Побитовое НЕ (~):

```
int x;
x = 5; // 0101
printf("%d", ~x); // ~0101 = 1010 = 10
```

Используемые источники:

1. Язык программирования Си. Авторы: Б. Керниган, Д. Ритчи.
2. Онлайн компилятор языка Си: https://www.onlinegdb.com/online_c_compiler

Задание:

1. Напишите программу, которая будет выдавать диапазоны значений типов char, short, int и long, описанных как signed и как unsigned, с помощью печати соответствующих значений из стандартных заголовочных файлов и путем прямого вычисления. Определите диапазоны чисел с плавающей точкой различных типов. [Инструкция sizeof(x) выдаёт размеры переменных и типов данных указанных в скобках]
2. Напечатайте ваше имя и фамилию на двух разных строках, используя функцию printf.
3. Поэкспериментируйте, удаляя некоторые части программы, и посмотрите, какие сообщения об ошибках вы получите (минимум три разных ошибки).
4. Выясните, что произойдет, если в строковую константу аргумента printf вставить \c, где c—символ, не входящий в представленный выше список.
5. Используя переменные a, b, c (с заранее заданными значениями), вычислите дискриминант $D = b^2 - 4ac$. Поместите результат в переменную D. Выведите полученный ответ.
6. Выясните, что произойдет, если прибавить к числу 255, хранимого в переменной типа unsigned char, число 1. Отобразите результат и напишите почему так произошло.