

ПРАКТИЧЕСКАЯ РАБОТА №1.

СТРУКТУРА ПРОГРАММЫ, ОСНОВНЫЕ ТИПЫ ДАННЫХ, ВВОД/ВЫВОД

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №1. Форма проведения – групповая работа в

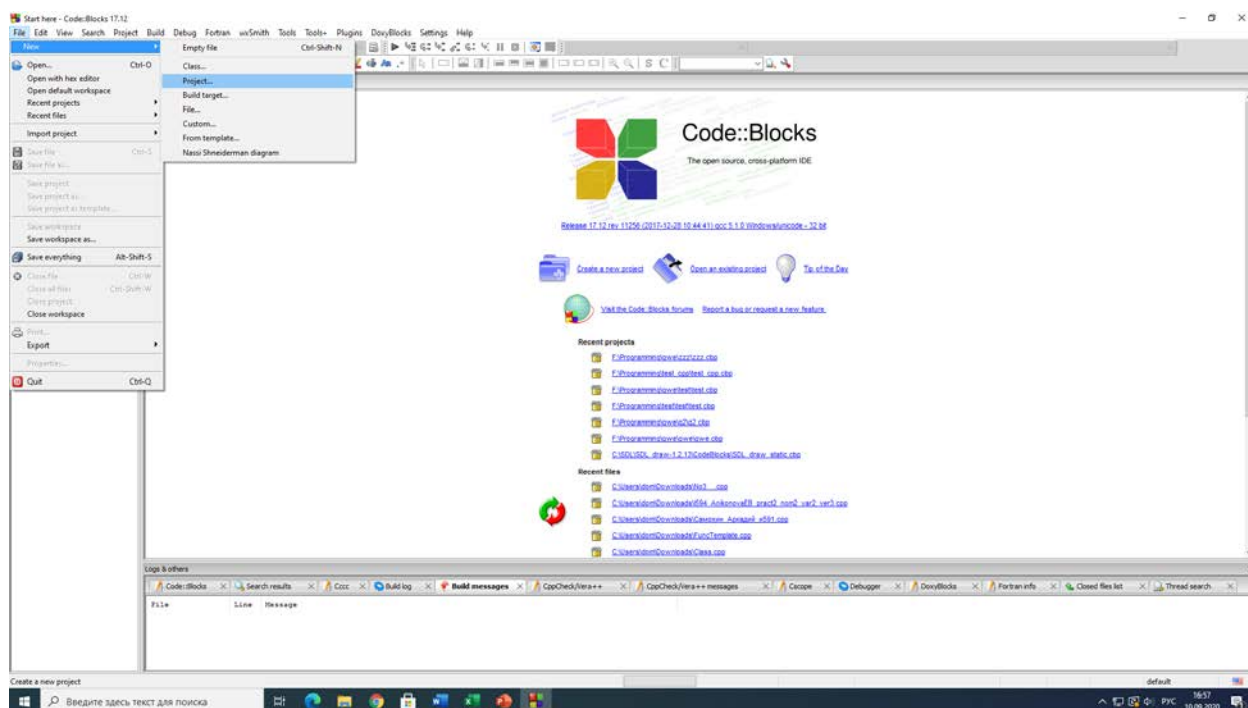
компьютерном классе. Отрабатываемые вопросы:

- организация работы в кафедральной компьютерной сети или компьютерной сети вычислительного центра (ВЦ), знакомство с интегрированной средой разработки Code::Blocks, создание проекта, написание простейшей программы («Hello, world!») на языке Си;
- написание простой программы на языке Си с использованием данных типов `int` и `double`, арифметических операций и операций присваивания, функций форматного ввода и вывода;
- ввод и вывод данных в разных форматах;
- разбор часто встречающихся ошибок при написании форматных строк.

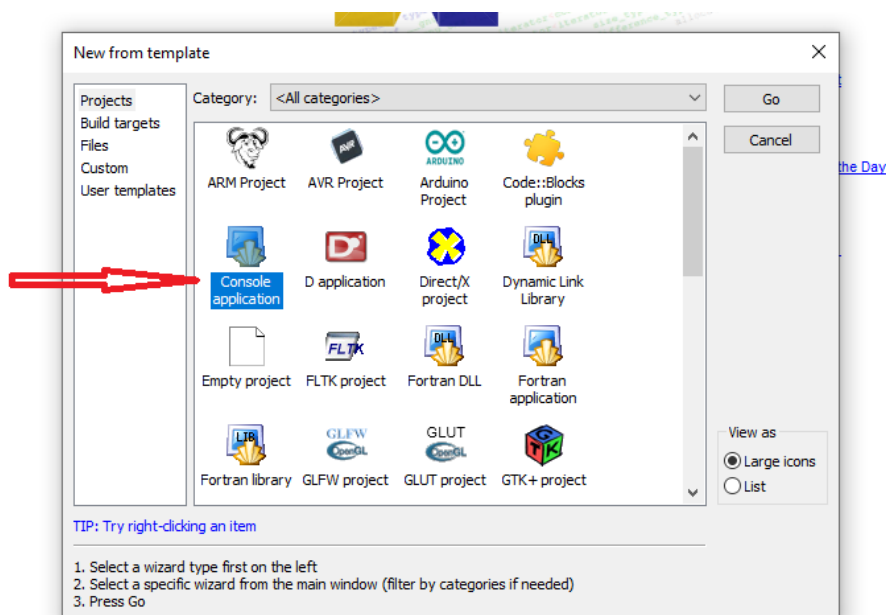
Задание 0.

Создайте новую папку, дайте ей имя по номеру группы, по желанию можно добавить фамилию. При задании имени во избежание возможных проблем предпочтительно не использовать русские буквы. В этой папке Вы будете сохранять все создаваемые файлы.

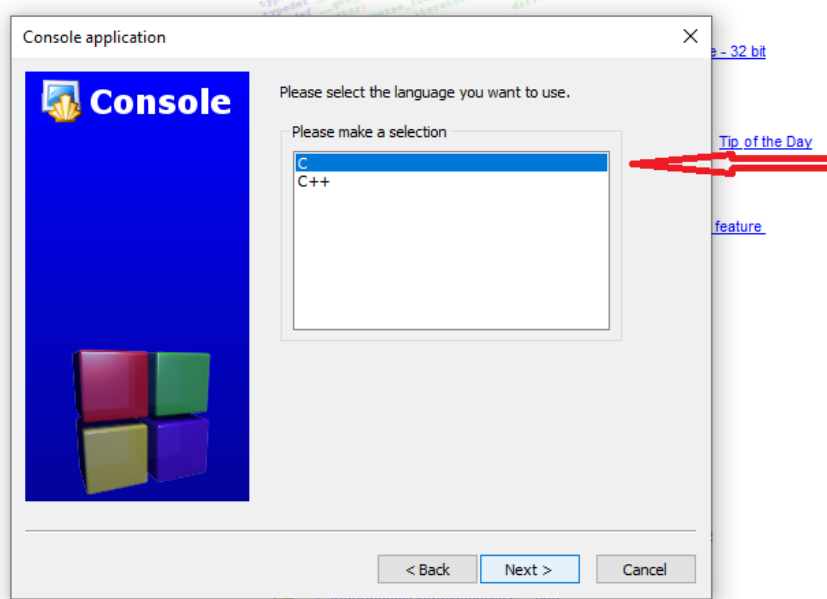
Запустите Code::Blocks и создайте новый проект:



В появившемся окне выберите *Console application*:



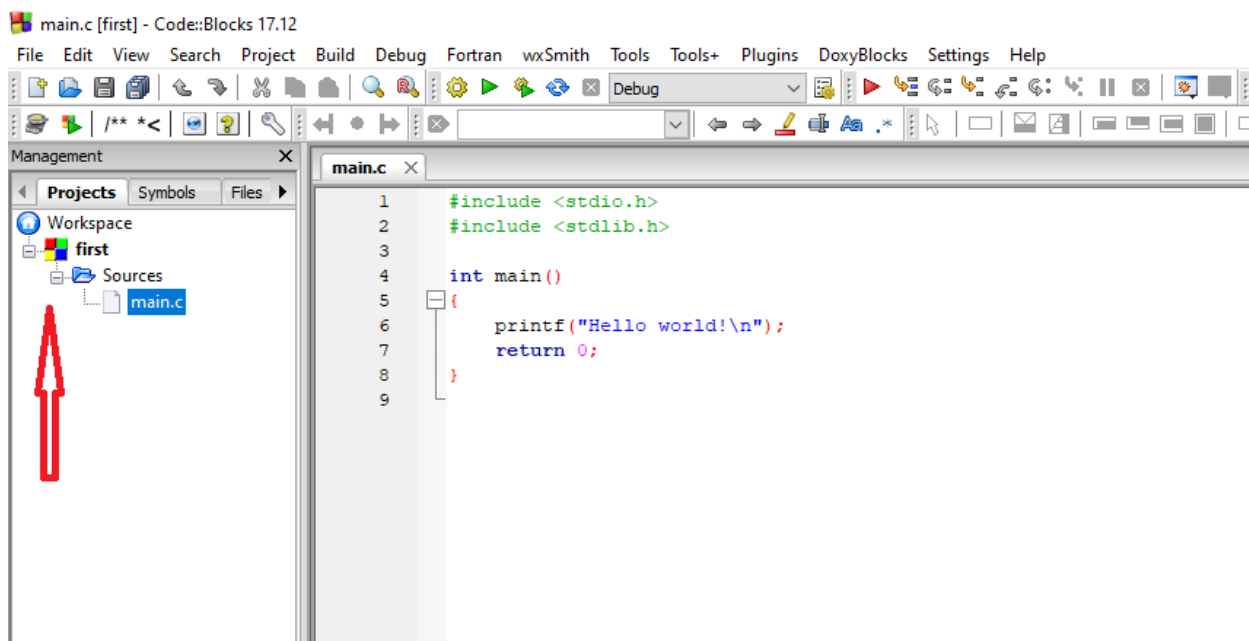
В окне выбора языка программирования выберите C



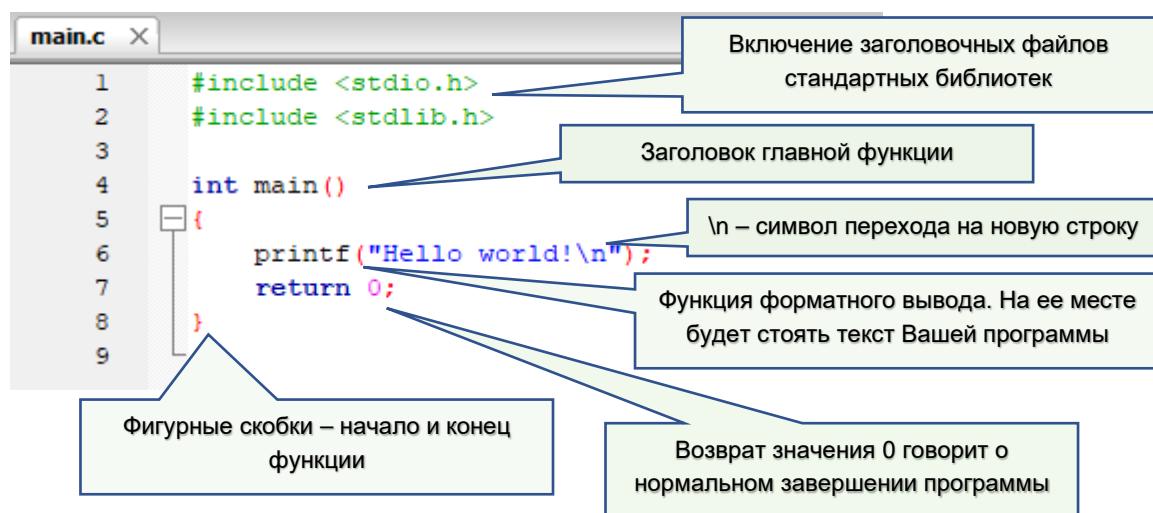
В следующем окне задайте имя проекта ЛАТИНСКИМИ буквами (это важно, Code::Blocks не «дружит» с кириллицей, могут возникнуть трудно интерпретируемые ошибки) и укажите путь к созданной Вами папке.

В следующем окне (выбор компилятора) ничего менять не надо, жмите кнопку *Finish*.

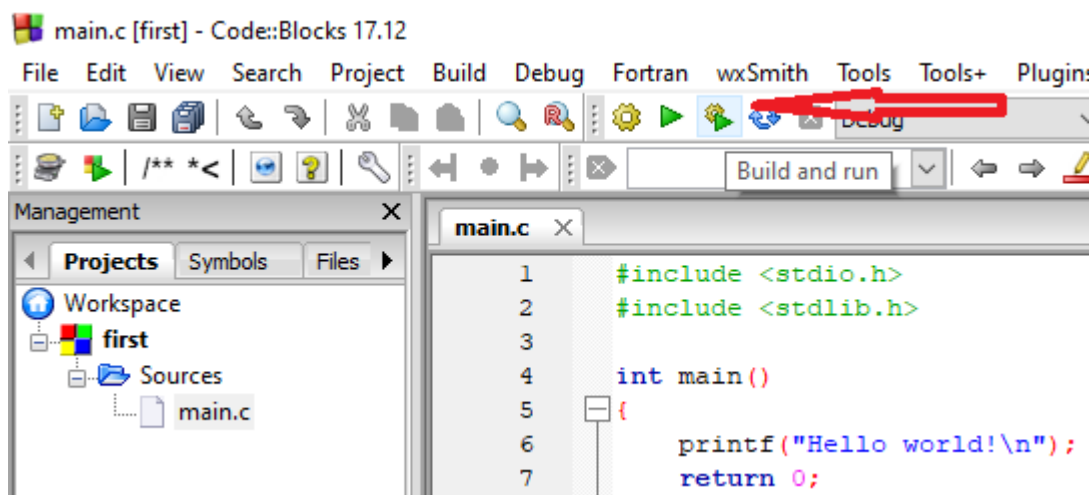
Проект создан и активирован. Разверните список источников, и откройте файл *main.c*



Рассмотрите структуру файла:



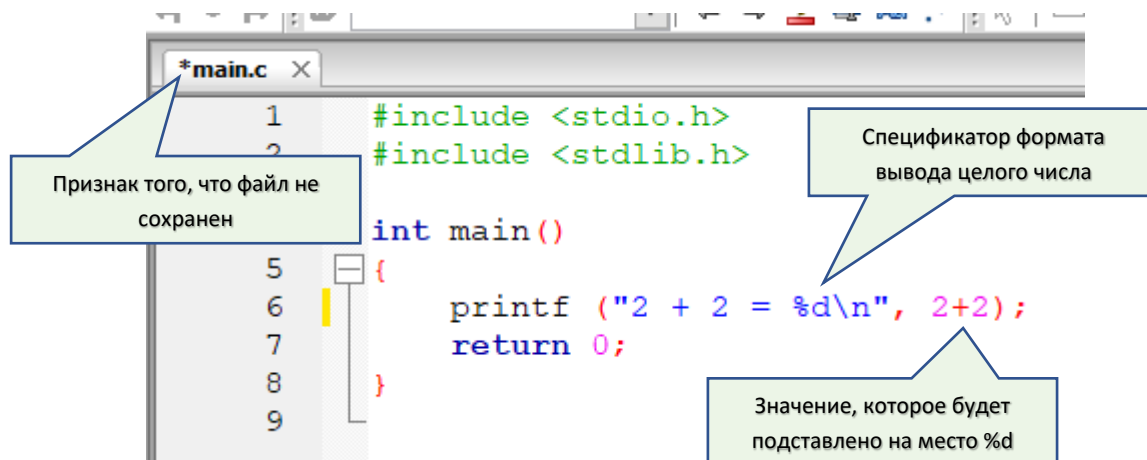
Запустите программу на исполнение, выбрав кнопку «Build and run» на панели инструментов:



Убедились, что программа работает. Внесите в программу изменения, чтобы она выводила сумму 2+2, для этого замените строку вывода на такую:

```
printf ("2 + 2 = %d\n", 2+2);
```

Должно получиться так:



Запустите программу, посмотрите, что получилось.

Задание 1.

Находить только сумму 2+2 не интересно. Напишем программу, которая будет находить сумму любых двух целых чисел, введенных с клавиатуры.

Для начала проанализируем, какие в нашей программе входные и выходные данные и как их можно запрограммировать. Входные – слагаемые, два целых числа. Выходное данные – сумма, тоже целое число. Все данные программа берет из оперативной памяти, в которой они размещаются в переменных. *Переменная* – это область памяти, которая имеет имя, тип и значение. Назовем первое слагаемое *a*, второе – *b*, а сумму – *s*. Поскольку все данные у нас целочисленные, то пусть наши переменные будут иметь стандартный целый тип *int*. Запишем результаты анализа в отчет.

Значения переменных *a* и *b* будет задавать пользователь, вводя их с клавиатуры, а значение переменной *s* должно быть вычислено в программе как сумма значений *a* и *b*. Все переменные в программе на языке Си должны быть предварительно объявлены. Объявление наших переменных выглядит так:

```
int a, b, s;
```

Теперь прикинем, что должно получиться и построим небольшую таблицу тестирования:

Входные данные	Ожидаемый результат	Результат работы программы
a = 2, b = 2	4	
a = 2000, b = -2000	0	
a = 2000000000, b = 2000000000	4000000000	

Таблицу сразу включаем в отчет.

Запишем последовательность действий, которую должна будет выполнить программа – алгоритм:

1. ввести значения a и b ;
2. вычислить $s = a + b$;
3. вывести значение s на экран.

На языке Си этот алгоритм записывается так:

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

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

```
s = a + b;
```

```
printf ("%d + %d = %d\n", a, b, s);
```

%d – спецификатор формата ввода, означает, что вводится целое
& - операция получения адреса

Для того, чтобы пользователь догадался, что от него требуется, перед каждым вызовом функции *scanf()* нужно вывести какую-нибудь подсказку. Готовая программа будет выглядеть так:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int a, b, s;          /* объявление переменных */
7      printf ("a = ");      /* печать сообщения */
8      scanf ("%d", &a);      /* ввод с клавиатуры целого числа и запись его в переменную a */
9      printf ("b = ");      /* печать следующего сообщения */
10     scanf ("%d", &b);      /* ввод с клавиатуры целого числа и запись его в переменную b */
11     s = a + b;             /* вычисление суммы и запись ее в переменную s */
12     printf ("%d + %d = %d\n", a, b, s); /* вывод результата в формате число + число = число */
13     return 0;
14 }
15
```

Запустите программу, введите значения, записанные в первой графе таблицы, получите результат и запишите в третью графу. Сделайте выводы. Все действия фиксируйте в отчете.

Такая последовательность решения задач: анализ входных и выходных данных, подготовка тестовых наборов, составление алгоритма, запись программы, тестирование программы, выводы и фиксация всех действий в отчете – должна постоянно использоваться при выполнении индивидуальных практических заданий (и в дальнейшем тоже 😊).

Задание 2.

Используя этот алгоритм, напишите программу деления одного целого числа на другое. При подготовке тестового набора включите в него пару чисел, в которой делимое делится на делитель без остатка, пары, в которых делимое нацело на делитель без остатка не делится (в одной паре делимое больше делителя, в другой - меньше), и пару, в которой делитель равен нулю. Протестируйте написанную программу, сделайте выводы.

Задание 3.

Измените тип переменных на *double* (стандартный вещественный тип). В функциях *scanf()* и *printf()* поменяйте спецификаторы формата на *%lf*. Должно получиться примерно так:

```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      double a, b, res;      /* объявления переменных */
7      printf ("a = ");      /* печать сообщения */
8      scanf ("%lf", &a);    /* ввод с клавиатуры вещественного числа и запись его в переменную a */
9      printf ("b = ");      /* печать следующего сообщения */
10     scanf ("%lf", &b);    /* ввод с клавиатуры вещественного числа и запись его в переменную b */
11     res = a / b;           /* вычисление частного и запись ее в переменную res */
12     printf ("%lf / %lf = %lf\n", a, b, res); /* вывод результата в формате число / число = число */
13     return 0;
14 }
15
```

Протестируйте программу на тех же тестовых наборах, что и в предыдущем задании. Сделайте выводы.

Поменяйте формат вывода чисел, изменив количество знаков, выводимых после точки. Для этого в спецификаторе *%lf* между *%* и *l* поставьте точку и запишите требуемое число знаков, например, *%.2lf*. Если указать после точки 0 или не записывать число вовсе (*%.lf*), то точка и дробная часть выведены не будут. Обратите внимание на округление чисел. Сделайте выводы.

Задание 4.

Измените тип делимого и делителя обратно на *int*, результат оставьте типа *double*. Не забудьте поменять спецификатор формата в функции *scanf()* на *%d*. Протестируйте программу на тех же тестовых наборах, что и в задании 3. Сделайте выводы.

Внесите изменение в инструкцию вычисления результата – запишите (*double*) перед делимым:

```
res = (double) a / b;
```

Протестируйте программу, сделайте выводы.

Задание 5.

С помощью преподавателя или самостоятельно сделайте несколько ошибок при вызове функций *scanf()* и *printf()*. Важно! Каждый раз делайте только ОДНУ ошибку, запускайте программу, анализируйте результат, записывайте его, после чего возвращайте правильную запись. Наиболее распространенные ошибки:

- отсутствие *&* перед именем переменной в *scanf()*;
- наличие *&* перед именем переменной в *printf()* при выводе значения переменной;
- тип спецификатора формата не совпадает с типом переменной;
- количество спецификаторов формата не совпадает с количеством вводимых или выводимых значений.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №2. Форма проведения – групповая работа в компьютерном классе. Отрабатываемые вопросы:

- простые типы данных языка Си: предельные значения, точность представления;
- анализ ситуаций, приводящих к выходу за границы типа;
- правила неявного преобразования типа;
- анализ результатов работы программы при обработке данных одного типа и в комбинациях, использование явного и неявного преобразования типов;
- анализ проблем, возникающих при использовании данных вещественных типов, и методов их решения.

Задание 0. Повторите сведения о кодировании целых чисел.

Целые беззнаковые числа представляются в компьютере *прямым кодом* – своим двоичным представлением, дополненным в старших разрядах нулями до размера разрядной сетки.

Дополнительный код (англ. two's complement, иногда twos-complement) — наиболее распространённый способ представления *отрицательных целых* чисел в компьютерах. Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел, чем упрощает архитектуру ЭВМ.

Дополнительный код для отрицательного числа можно получить инвертированием его двоичного модуля и прибавлением к инверсии единицы или вычитанием числа из нуля.

При записи числа в дополнительном коде старший разряд является знаковым. Если его значение равно 0, то в остальных разрядах записано положительное двоичное число, совпадающее с прямым кодом.

Двоичное 8-разрядное число со знаком в дополнительном коде может представлять любое целое в диапазоне от –128 до +127. Если старший разряд равен нулю, то наибольшее целое число, которое может быть записано в оставшихся 7 разрядах, равно $2^7-1=127$.

Примеры:

Десятичное представление	Двоичное представление (8 бит)		
	прямой	обратный	дополнительный
127	0111 1111	0111 1111	0111 1111
1	0000 0001	0000 0001	0000 0001
0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	
-1	1000 0001	1111 1110	1111 1111
-2	1000 0010	1111 1101	1111 1110
-3	1000 0011	1111 1100	1111 1101
-4	1000 0100	1111 1011	1111 1100

-5	1000 0101	1111 1010	1111 1011
-6	1000 0110	1111 1001	1111 1010
-7	1000 0111	1111 1000	1111 1001
-8	1000 1000	1111 0111	1111 1000
-9	1000 1001	1111 0110	1111 0111
-10	1000 1010	1111 0101	1111 0110
-11	1000 1011	1111 0100	1111 0101
-127	1111 1111	1000 0000	1000 0001
-128	---	---	1000 0000

Задание 1. (В отчет войдет под номером 6)

Познакомьтесь с типами данных *char* и *unsigned char*.

Создайте новый проект, скопируйте в файл *main.c* содержимое файла *pr_1.c*. Запустите программу, расположите окно программы таким образом, чтобы видеть и текст программы, и результаты ее работы. С помощью преподавателя проанализируйте работу программы, запишите в комментариях, как вычисляются значения.

Задание 2. (В отчет войдет под номером 7)

Познакомьтесь с типами данных *int*, *short int*, *long int* и *unsigned int*.

Создайте новый проект, скопируйте в файл *main.c* содержимое файла *pr_2.c*. Запустите программу, расположите окно программы таким образом, чтобы видеть и текст программы, и результаты ее работы. С помощью преподавателя проанализируйте работу программы, запишите в комментариях, как вычисляются значения.

Задание 3. Повторите сведения о кодировании вещественных чисел.

В компьютерной технике вещественными называются числа, имеющие дробную часть. Вещественные числа обычно представляются в виде чисел с плавающей запятой. Числа с плавающей запятой – один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений, его можно считать аналогом экспоненциальной записи чисел, но только в памяти компьютера.

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые знак (англ. *sign*), порядок (англ. *exponent*) и мантиссу (англ. *mantis*). В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть – показатель степени, и ещё один бит используется для указания знака числа (0 – если число положительное, 1 – если число отрицательное). Порядок и мантисса – целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:

где S – знак, B – основание системы счисления, E – порядок, а M – мантисса.

Для записи числа используется *нормализованная форма* записи (англ. *normalized*), в которой мантисса десятичного числа принимает значения от 1 (включительно) до 10 (не включительно), а мантисса двоичного числа принимает значения от 1 (включительно) до 2 (не включительно). То есть в мантиссе слева от запятой до применения порядка находится ровно один знак. В такой форме любое число (кроме 0) записывается единственным образом. Ноль же представить таким образом невозможно, поэтому стандарт предусматривает специальную последовательность битов для задания числа 0 (а заодно и некоторых других полезных чисел, таких как $-\infty$ и $+\infty$). Так как старший двоичный разряд (целая часть) мантиссы вещественного числа в нормализованном виде всегда равен «1», то его не хранят, экономя таким образом один бит, т.е. имеет место так называемый скрытый разряд. Однако при аппаратном выполнении операций этот разряд автоматически восстанавливается и учитывается. Порядок числа также учитывает скрытый старший разряд мантиссы.

Знак										Мантисса (23+1 бита)																																					
Порядок (8 бит)																																															
0	0	0	0	0	0	0	0	0	0	1,	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
30										23										22										0																	

Число двойной точности (Double precision, **double**) – компьютерный формат представления чисел, занимающий в памяти 64 бита (или 8 байт).

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №3. Форма проведения – групповая и индивидуальная работа в компьютерном классе. Отрабатываемые вопросы:

- разбор и построение сложных выражений с использованием явного и неявного приведения типов, множественного присваивания, префиксной и постфиксной форм записи операций инкремента и декремента.

Задание 0. (В отчет войдет под номером 9)

Создайте новый проект, скопируйте в файл *main.c* содержимое файла *pr_4.c*. Запустите программу. Проанализируйте значения переменных после каждой операции присваивания. Сохраните скриншот. Найдите выражения, в которых изменяются значения двух и более переменных, прокомментируйте их и запишите последовательность выражений, выполняющих ту же последовательность действий, в каждом из которых будет изменяться значение только одной переменной. Например, в выражении

```
z *= x++ - ++y
```

изменяются значения трех переменных. Забираете его в комментарий и пишете три отдельных инструкции, выполняющие те же действия в том же порядке:

```
/* z *= x++ - ++y; */
++y;
z *= x - y;
++x;
```

Как вариант

```
/* z = x++ - ++y; */
y = y + 1;
z = z * ( x - y );
x = x + 1;
```

Результат выполнения измененной программы должен полностью совпасть с результатом выполнения оригинала.

Задание 1. (В отчет войдет под номером 10)

Напишите программу для вычисления значений следующих выражений:

$a=5, c=5$

$a=a+b-2$

$c=c+1, d=c-a+d$

$a=a*c, c=c-1$

$a=a/10, c=c/2, b=b-1, d=d*(c+b+a)$

Выражения, записанные в одной строке, записывайте одним оператором-выражением, не содержащим запятой. Используйте расширенные операции присваивания, операции инкремента и декремента. Переменные c и d объявите как целые, переменные a и b – как вещественные. Значения переменных b и d вводите с клавиатуры. После вычисления каждого выражения выводите на экран значения всех переменных.

Перед написанием программы не забудьте подготовить тестовые наборы и вычислить ожидаемые результаты.