Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологий программирования

**Методические указания**

**к лабораторной работе № 2  
 по курсу «Основы алгоритмизации   
и программирования»**

«Операторы объявлений, присвоения,   
вызова функций и приведения типов.   
Математические, логические   
и условная операции.   
Построение выражений»

Преподаватель: Магеров В.В.

Составитель: Войтехович А.В.

Полоцк, 2016

**ЦЕЛЬ РАБОТЫ**

Познакомить студентов с принципами работы основных операторов и операций языка С++. Научить писать программы с использованием этих операторов и операций. Выработать у студентов навык их корректного использования.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

# 1 Операторы языка С++

Операторы управляют процессом выполнения программы. Набор операторов языка С++ содержит все управляющие конструкции структурного программирования.

Составной оператор ограничивается фигурными скобками. Все другие операторы заканчиваются точкой с запятой.

* **Пустой оператор – ;**

Пустой оператор – это оператор, состоящий только из точки с запятой. Он может появиться в любом месте программы, где по синтаксису требуется оператор. Выполнение пустого оператора не меняет состояния программы.

* **Составной оператор – {...}**

Действие составного оператора состоит в последовательном выполнении содержащихся в нем операторов, за исключением тех случаев, когда какой-либо оператор явно передает управление в другое место программы.

* **Оператор разрыва – break**;

Оператор разрыва прерывает выполнение операторов цикла и ветвления. Он может содержаться только в теле этих операторов. Управление передается оператору программы, следующему за прерванным. Если оператор разрыва записан внутри вложенных операторов цикла, то он завершает только непосредственно охватывающий его составной оператор.

* **Оператор продолжения – continue**;

Оператор продолжения передает управление на следующую итерацию в операторах цикла. Он может содержаться только в теле этих операторов.

* **Оператор возврата –** return [*<выражение>*];

Оператора возврата заканчивает выполнение функции, в которой он содержится, и возвращает управление в вызывающую функцию. Управление передается в точку вызывающей функции, непосредственно следующую за оператором вызова. Значение ***выражения*,** если она задано, вычисляется, приводится к типу, объявленному для функции, содержащей оператор возврата, и возвращается в вызывающую функцию. Если ***выражение*** опущено, то возвращаемое функцией значение не определено.

С формальной точки зрения операторы **break**, **continue** и **return** не являются операторами структурного программирования. Однако их использование в ограниченных количествах оправдано, когда они упрощают понимание программы и позволяют избегать больших вложенных структур.

**Ввод/вывод** не является частью языка С++, а осуществляется функциями, входящими в состав стандартной библиотеки.

# 2 Операторы объявления переменных

Операторы объявления служат для ввода имён в программу. Процедура ввода имени переменной предполагает не только создание отличного от любого ключевого слова идентификатора, но и кодирование дополнительной информации о характеристиках объекта, с которым будет связано объявляемое имя.

К характеристикам объекта относятся тип объекта, класс памяти, время жизни объекта, множество других свойств, представляемых различными модификаторами.

# 2.1 Объявление переменных

Оператор объявления состоит из спецификации типа и списка имён переменных, разделённых запятой. В конце обязательно должна стоять точка с запятой.

Оператор имеет следующий формат:

**[модификаторы] специф.\_типа идентификатор [, идентификатор] ...**

**Модификаторы** - ключевые слова **signed**, **unsigned**, **short**, **long**.

**Спецификатор типа** - ключевое слово **char**, **int** и другие, определяющее тип объявляемой переменной.

**Идентификатор** - имя переменной.

Пример:

**char** x;

**int** a, b, c;

**unsigned long long** y;

Таким образом, будут объявлены переменные **x**, **a**, **b**, **c**, **y**. В переменную **x** можно будет записывать значения от -128 до 127. В переменные **a**, **b**, **c** - от -32 768 до +32 767. В переменную **y** - от 0 до 18 446 744 073 709 551 615.

Идентификатор должен удовлетворять правилам именования переменных в C/C++, и не могут быть использованы в одной области переменные с одинаковыми именами. Переменные автоматически не инициализируются значениями, они имеют значение (*мусор*), которое хранится в памяти по адресу переменной.

Правила именования переменных гласят:

* имя переменной не должно начинаться с цифры;
* имя переменной не должно включать следующие символы: **,**, **/**, **:**,\*, **?**, **"**, **<**, **>**, **|**.

# 2.2 Инициализация значения переменной при объявлении

При объявлении переменную можно проинициализировать, то есть присвоить ей начальное значение. Сделать это можно следующим образом.

1. **int** x = 100;

Таким образом, в переменную **x** при объявлении сразу же будет записано число 100.

Лучше избегать смешивания инициализируемых переменных в одном операторе описания, то есть инициализируемые переменные лучше объявлять в отдельных строках.

# 2.3 Константы

Переменная любого типа может быть объявлена как немодифицируемая. Это достигается добавлением ключевого слова **const** к спецификатору типа. Переменные с типом **const** представляют собой данные, используемые только для чтения, то есть этой переменной не может быть присвоено новое значение. Если после слова **const** отсутствует спецификатор типа (так делать не рекомендуется), то константы рассматриваются как величины со знаком, и им присваивается тип **int** или **long int** в соответствии со значением константы: если константа меньше 32 768, то ей присваивается тип **int**, в противном случае **long int**.

Пример:

**const long int** k = 25;

**const** m = -50; // подразумевается **const int** m=-50

**const** n = 100000; // подразумевается **const long int**  
 // n=100000

# 3 Операторы присвоения

Для присваивания в С++ служит знак "=". Выражение, стоящее справа от знака присваивания, вычисляется, и полученное значение присваивается переменной, стоящей слева от знака присваивания. При этом предыдущее значение, хранящееся в переменной, стирается и заменяется на новое.

Оператор "=" не следует понимать как равенство.

Например, выражение a = 5; следует читать как "присвоить переменной a значение 5".

Примеры:

x = 5 + 3; // сложить значения 5 и 3, результат

//присвоить переменной x (записать в переменную x)

b = a + 4; // прибавить 4 к значению, хранящемуся в

// переменной a, полученный результат присвоить

// переменной b (записать в переменную b)

b = b + 2; // прибавить 2 к значению, хранящемуся в

// переменной b, полученный результат присвоить

// переменной b (записать в переменную b)

В правой части значение переменной может использоваться несколько раз:

c = b \* b + 3 \* b;

# 3.1 Многочисленное присваивание

С++ позволяет присваивать нескольким переменным одни и те же значения путем использования многочисленных присваиваний в одном операторе. Например, данный фрагмент программы присваивает переменным х, у и z значение 0:

х = у = z = 0;

В профессиональных программах переменным часто присваиваются стандартные значения с использованием данного метода.

# 3.2 Дополнительные способы присваивания

Кроме простого оператора присваивания **"="**, в Си существует еще несколько комбинированных операторов присваивания: **"+=", "-=", "\*=<", "/=",** **"%="**.

Примеры:

x += y; // то же, что и x = x + y; - сложить x и y

// и записать результат в переменную x

x -= y; // то же, что и x = x - y; - отнять y от x

// и записать результат в переменную x

x \*= y; // то же, что и x = x \* y; - умножить x на y

// и записать результат в переменную x

x /= y; // то же, что и x = x / y; - разделить x на y

// и записать результат в переменную x

x %= y; // то же, что и x = x % y; - вычислить

// целочисленный остаток от деления x на y

// и записать результат в переменную x

# 4 Оператор вызова функций

Основные возможности языка С/С++ реализованы посредством функций из различных библиотек. Понятие функции будет подробно рассмотрено позже, здесь же мы ограничимся представлением функции как именованного независимого фрагмента программы, который может быть вызван (запущен) из другой функции. Функция может иметь параметры - значения, которые необходимы для выполнения функции (передаются при вызове функции).

В языке С++ оператор вызова функций, в принципе, выделен условно (во многих книгах он вообще не рассматривается как оператор). Оператор вызова функций просто определяет синтаксис описания вызова любой функции в тексте программы:

имя\_функции(параметры);

Сначала указывается имя функции, затем, в круглых скобках, указываются параметры функции. Завершается оператор вызова функций точкой с запятой.

В языке С++, если функция не имеет параметров, то пустые круглые скобки все равно должны присутствовать.

Особенностью оператора вызова функций является то, что он может содержать в себе и сам может быть включен в другие операторы языка С++. Пример: вычисление квадратного корня куба вещественной переменной *x*:

sqrt(pow(x,3.0));

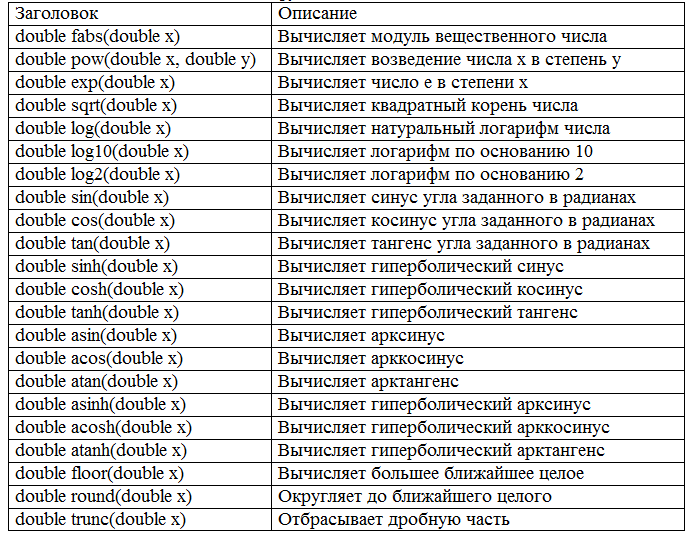
В этом примере осуществляется вызов функции **sqrt**, вычисляющий квадратный корень, в качестве параметра которой осуществляется вызов функции **pow**, выполняющей возведение переменной *x* в куб. Параметрами функции **pow** являются число, которое необходимо возвести в степень (переменная *x*), и значение степени (вещественная константа 3.0). Константа 3.0 является вещественной, так как функция **pow** работает с вещественными параметрами. Функции **sqrt** и **pow** описаны в библиотеке математических функций языка С **math.h**, а также в аналогичной библиотеке языка C++ **<cmath>**. Следует обратить внимание на то, что точка с запятой ставится только в конце выражения.

В таблице 1 приведены основные функции для математических вычислений из математической библиотеки <**cmath**>.

Все приведенные ниже функции оперируют значениями вещественного типа двойной точности (**double**). В библиотеке **<cmath>** содержатся и функции, соответствующие перечисленным выше функциям, но для вещественных чисел одинарной (**float**) и повышенной (**long double**) точности. Данные функции имеют то же имя, но с добавлением в конце символа ‘**f**’ - для вещественных чисел одинарной точности, ‘**l**’ - для вещественных чисел повышенной точности. Например, имя функции вычисления синуса для значения типа **float** будет **sinf**, а для значения типа **long double** - **sinl**.

В таблице приведены далеко не все функции, содержащиеся в библиотеке **<cmath>**. С полным набором функций можно ознакомиться, обратившись к соответствующему разделу помощи среды разработки.

**Таблица 1** - Основные функции для математических вычислений



**5 Оператор приведения типов**

В C++ различают явное и неявное преобразование [типов данных](http://cppstudio.com/obuchenie_cpp/tipy_dannyh). Неявное преобразование типов данных выполняет компилятор С++, ну а явное преобразование данных выполняет сам программист.

Результат любого вычисления будет преобразовываться к наиболее точному типу данных, из тех типов данных, которые участвуют в вычислении. Для наглядного примера ниже представлена таблица 2 с преобразованиями типов данных. В таблице рассмотрена операцию деления. В качестве целочисленного типа данных взят тип **int**, а вещественный тип данных по умолчанию **float**.

**Таблица 2** — Явное и неявное преобразование типов данных в С++

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **y** | **Результат деления** | **Пример** |
| *делимое* | *делитель* | *частное* | *x = 15 y = 2* |
| int | int | int | 15 / 2=7 |
| int | float | float | 15 / 2,0=7,5 |
| float | int | float | 15,0 / 2=7,5 |

Из таблицы видно, что меняя переменные различных [типов данных](http://cppstudio.com/obuchenie_cpp/tipy_dannyh) местами, результат остается тот же (в нашем случае это делимое и делитель). О неявном преобразовании типов данных все.

Что же касается явного преобразования, то оно необходимо для того чтобы выполнять некоторые манипуляции, тем самым меняя результат вычисления. Допустим, необходимо разделить целые числа 15 и 2, делим: **15 / 2 = 7**. Результат тот же, что и в таблице. Но если сделать незначительные преобразования, например: **15,0 / 2 = 7,5** при таком делении число 15 является вещественным, значит и результат будет вещественный. Само число 15 с точки зрения математики не изменилось, ведь **15 = 15,0**. Этот же прием можно было применить к двойке, результат был бы тем же, а можно было сразу к двум числам, но зачем, если хватает одного из двух.

Еще один способ явного преобразования типов данных:

float(15) / 2 // результат равен 7,5, число 15

//преобразуется в вещественный тип данных float.

double(15) / 2 // результат равен 7,5 – тоже самое

В С++ также предусмотрена унарная операция приведения типа:

static\_cast </\*тип данных\*/>(/\*переменная или число\*/)

Пример:

static\_cast <float> (15) / 2 результат равен 7,5

Пример с переменной:

int ret=15;

static\_cast<float>(ret)/2 //результат равен 7,5

В случае с переменной надо понимать, что в строке 2 переменная ***ret*** не преобразуется в тип данных **float**, а всего лишь на всего создается временная копия переменной ***ret*** с типом данных **float.**

# 6 Математические операции

В языке С++, как и в некоторых других языках программирования, математические операции делятся на две группы:

* математические операции для вещественных и целочисленных вычислений;
* математические операции только для целочисленных вычислений.

К **математическим операциям** для вещественных и целочисленных вычислений языка С относят обычные арифметические операции:

* сложения (+),
* вычитания (),
* умножения (\*),
* деления (/).

Пример:

x = 3; // переменной x будет присвоено значение 3

y = x + 5; // к значению, хранящемуся в переменной x,   
 // будет прибавлено число 5, полученный   
 //результат будет записан в переменную y

z = x \* y; // значения переменных x и y будут   
 // перемножены, результат будет записан в переменную z

z = z - 1; // от значения, хранящегося в переменной z,   
 //будет отнято 1 результат будет записан в   
 // переменную z. В результате z = 23

Математические операции для вещественных и целочисленных вычислений являются простейшими математическими действиями и относятся к классу бинарных операций. Так как эти операции предназначены для использования, как в целочисленных, так и вещественных вычислениях, то тип результирующего значения этих операций зависит от их операндов. Соответствие типа результата от типов операндов приведено в таблице 3.

**Таблица 3** - Соответствие типа результата от типов операндов

|  |  |  |
| --- | --- | --- |
| **Тип первого операнда** | **Тип второго операнда** | **Тип результата** |
| Целый | Целый | Целый |
| Целый | Вещественный | Вещественный |
| Вещественный | Целый | Вещественный |
| Вещественный | Вещественный | Вещественный |

Вторую группу математических операций составляют операции, предназначенные только для целочисленных вычислений. К ним относятся:

* операция взятия остатка от деления;
* побитовые операции;
* операции сдвигов;
* операции инкремента и декремента.

**Операция взятия остатка** от деления является бинарной операцией и в языке С++ обозначается символом процента (%).

Пример вычисления:

**int** a = 10, b = 3, c;

c = a % b;

После выполнения данного фрагмента переменная ***c*** будет содержать значение 1.

**Битовые операции** — это тестирование, установка или сдвиг битов в байте или слове, которые соответствуют стандартным типам языка С++ **char** и **int**. Битовые операторы не могут использоваться с **float**, **double, long double, void** и другими сложными типами. Таблица 4 содержит имеющиеся операторы.

**Таблица 4** – Битовые операторы

|  |  |
| --- | --- |
| **Оператор** | **Действие** |
| & | И |
| | | ИЛИ |
| ^ | Исключающее ИЛИ |
| ~ | Дополнение |
| >> | Сдвиг вправо |
| << | Сдвиг влево |

Битовые операторы **И, ИЛИ, НЕ** используют ту же таблицу истинности, что и их логические эквиваленты, за тем исключением, что они работают побитно. В таблицах 5 – 7 представлены таблицы истинности для операторов «исключающее ИЛИ», «И» и «ИЛИ».

**Таблица 5** – Таблица истинности оператора «исключающее ИЛИ»

|  |  |  |
| --- | --- | --- |
| **р** | **q** | **p^q** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Как следует из таблицы, исключающее ИЛИ выдает истину, если только один из операндов истинен. В противном случае получается ложь.

**Таблица 6** – Таблица истинности оператора «И»

|  |  |  |
| --- | --- | --- |
| **р** | **q** | **p^q** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Таблица 7** – Таблица истинности оператора «ИЛИ»

|  |  |  |
| --- | --- | --- |
| **р** | **q** | **p^q** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Операторы сдвига >> и <<** сдвигают биты в переменной вправо и влево на указанное число. Общий вид оператора сдвига вправо:

переменная >> число сдвигов

Общий вид оператора сдвига влево:

переменная << число сдвигов

Помните, что сдвиг — это не то же самое, что и вращение, то есть биты, сдвигающиеся на один конец, не появляются с другого. Сдвинутые биты теряются, а с другого конца появляются нули. В том случае, если вправо сдвигается отрицательное число, слева появляются единицы (поддерживается знаковый бит).

Операторы битового сдвига могут использоваться для выполнения быстрого умножения и деления целых чисел. Сдвиг влево равносилен умножению на 2, а сдвиг вправо -  делению на 2, как показано в таблице 8.

**Таблица 8** – Изменение переменной ***x*** типа **char** операторами сдвига >> и <<

|  |  |  |
| --- | --- | --- |
| **char х;** | **Битовое представление х после выполнения каждого оператора** | **Значение х** |
| x = 7;  x = x << 1;  x = x << 3;  x = x << 2;  х = х >> 1;  x = x >> 2; | 00000111  00001110  01110000  11000000  01100000  00011000 | 7  14  112  192  96  24 |

Каждый сдвиг влево приводит к умножению на 2. Обратите внимание, что после сдвига **х << 2** информация теряется, поскольку биты сдвигаются за конец байта.

Каждый сдвиг вправо приводит к делению на 2. Обратите внимание, что деление не вернуло потерянные биты.

**Оператор дополнение, ~,** инвертирует состояние каждого бита указанной переменной, то есть 1 устанавливается в 0, а 0 — в 1.

Надо обратить внимание, что в результате выполнения двух битовых дополнений получаем исходное число.

Если необходимо изменить значение переменной на 1, то используют **инкремент** или **декремент**.

**Инкремент** - операция увеличения на единицу значения, хранящегося в переменной.

Пример:

x++; // значение переменной x будет увеличено на 1

**Декремент** - операция уменьшения на единицу значения, хранящегося в переменной.

Пример:

x--; // значение переменной x будет уменьшено на 1

**Инкремент и декремент** относятся к операциям присваивания. При использовании декремента и инкремента совместно с оператором присваивания **"="** применяют постфиксную **(x++)** или префиксную **(++x)** запись. Первой выполняется префиксная запись.

Пример:

y = x++;

Предположим, что в переменной **x** хранилось значение 5. Тогда в **y** будет записано значение 5, после чего значение переменной **x** будет увеличено на 1. Таким образом, в **y** будет 5, а в **x** - 6.

y = --x;

Если в **x** хранилось значение 5, то сначала будет выполнено уменьшение **x** до 4, а затем это значение будет присвоено переменной **y**. Таким образом, **x** и **y** будет присвоено значение 4.

# 7. Операторы отношения и логические операторы

Операторы отношения применимы для вычисления соотношений между операндами. Логические операторы, используя правила формальной логики, также возвращают соотношения между операндами. Поскольку операторы отношения и логические операторы часто применяются вместе, то мы рассмотрим их также вместе.

Ключевой концепцией операторов отношения и логических операторов является понятие истины и лжи. В С/С++ истине соответствует любое значение, кроме 0, а лжи — 0. Выражение, использующее операторы отношения или логические операторы, возвращают 0(false) для лжи и 1(true) для истины. Список операторов и их описание представлены в таблице 9.

В таблице 10 представлена таблица истинности для описанных операторов. Она образована с использованием на входах 1 и 0.

**Таблица 9 -** Операторы отношения и логические операторы

|  |  |
| --- | --- |
| **Операторы отношения** | |
| **Оператор** | **Действие** |
| > | Больше чем |
| >= | Больше чем или равно |
| < | Меньше чем |
| <= | Меньше чем или равно |
| == | Равно |
| != | Не равно |
| **Логические операторы** | |
| && | И |
| || | ИЛИ |
| ! | НЕ |

**Таблица 10** – Таблица истинности для логических операторов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p** | **q** | **p&&q** | **p||q** | **!p** |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |

Как операторы отношения, так и логические операторы имеют более низкий приоритет по сравнению с арифметическими операторами. Это означает, что выражение типа **10 > 1+12** вычисляется как **10 > (1 + 12).** Результатом, естественно, будет ***ложь***.

В выражении может объединяться несколько операторов, как показано ниже:

10 > 5 && !(10 < 9) || 3 <= 4

в результате чего получаем истину.

Ниже показаны приоритеты выполнения операторов отношения и логических операторов:

|  |  |
| --- | --- |
| высший | ! > >= < <= && |
| низший | || |

Как и в арифметических выражениях, возможно использование круглых скобок для изменения естественного порядка вычисления операторов отношения или логических операторов. Например:

! 1 && 0

даст в результате 0, поскольку **!** вычисляется первым, а затем вычисляется &&. Если в этом выражении поставить скобки следующим образом:

! (1 && 0)

то получится истина.

Надо помнить, что все выражения с операторами отношения и логическими операторами дают в результате **0** или **1**. Поэтому следующий фрагмент программы не только корректен, но и печатает число 1:

int x;

x = 100;

printf("%d", x > 1);

# 8 Условная операция

Оператор **?** имеет следующий вид:

выражение1 ? выражение2 : выражение3;

где выражение1, выражение2 и выражениеЗ - это выражения.

Оператор **?** работает следующим образом: вычисляется ***выражение1;*** если оно истинно, то вычисляется ***выражение2*** и все выражение получает это значение; а если оно ложно, то вычисляется ***выражение3*** и все выражение получает это значение. Например:

х = 10;

у = х > 9 ? 100 : 200;

В данном примере ***у*** получает значение 100. Если бы ***х*** было меньше, чем 9, то ***у*** получило бы значение 200.

ЗАДАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

1. Считать из консоли значение переменной типа а), привести это значение к типу b). Вывести результат в консоль. Типы а) и b) указаны в таблице.
2. Разработать программу согласно условию варианта задания. При вводе и выводе данных использовать функции **scanf** и **printf** из стандартной библиотеки ввода и вывода языка «С»: **stdio.h**.
3. Разработать программу для вычисления значения выражения. При реализации использовать только условную операцию (?:). Все переменные и результат имеют вещественный тип, если это не оговаривается отдельно в условии задания.
4. \*Разработать программу для вычисления логического выражения. Тип переменных определяется из специфики предметной области задачи. Система исчисления по умолчанию: десятичная. Должно быть построено логическое выражение, результат которого присваивается целочисленной переменной. Результат выводится на экран в виде значения «1» - если истина и значения «0» - если ложь.
5. \*\*Разработать программу для вычисления математического выражения. Все переменные и результат имеют вещественный тип, если это не оговорено непосредственно в тексте задания. При реализации программы рекомендуется все числовые константы записывать рационально, а сложные математические выражения разбивать на части и использовать при их вычислении операторы +=, -= и т.д. Под переменной ***e*** в выражениях подразумевается значение константы e=2.71828. Варианты заданий смотрите в таблице.

# ВАРИАНТЫ ЗАДАНИЙ

|  |  |
| --- | --- |
| **Вариант** | **Задание** |
| **1** | 1) а)int(от 0 до 255); b)char;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти их сумму, а затем сумму сдвинуть на два разряда влево. Вывести результат в восьмеричной системе исчисления.  3)  4) Дано вещественное число. Проверить: принадлежит ли данное число промежутку: [-1;1].  5) |
| **2** | 1) а)int; b)double;  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Разделить первое на второе, а затем частное сдвинуть на один разряд вправо. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дано целое число. Проверить: является ли данное число положительным пятизначным числом.  5) |
| **3** | 1) а)int; b)float;  2) Два незнаковых целых числа вводятся с клавиатуры в шестнадцатеричной системе исчисления. Найти их произведение, а затем провести операцию побитового «И» над произведением и вторым числом. Вывести результат в десятичной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой латинского алфавита (нижний регистр).  5) |
| **4** | 1) а)char; b)float;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти остаток от деления первого числа на второе, а затем провести операцию побитового «ИЛИ» над остатком и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой русского алфавита (верхний регистр).  5) |
| **5** | 1) а)char; b)double;  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Разделить первое число на второе, а затем провести операцию побитового «Исключающего ИЛИ» над частным и первым числом. Вывести результат в десятичной системе исчисления.  3)  4) Даны три вещественных числа. Проверить: образуют ли данные числа возрастающую последовательность.  5) |
| **6** | 1) а)char; b)int;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти их произведение, а затем произведение сдвинуть на 4 разряда влево. Вывести результат в восьмеричной системе исчисления.  3)  4) Даны три целых числа. Проверить: образуют ли данные числа убывающую последовательность.  5) |
| **7** | 1) а)double(2 знака после запятой); b)float(4 знака после запятой);  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Умножить первое на второе, а затем произведение сдвинуть на два разряда вправо. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дано целое число. Проверить является ли данное число отрицательным трехзначным числом.  5) |
| **8** | 1) а)double(4 знака после запятой); b)float(1 знак после запятой);  2) Два незнаковых целых числа вводятся с клавиатуры в шестнадцатеричной системе исчисления. Разделить первое на второе, а затем частное сдвинуть на три разряда влево. Вывести результат в десятичной системе исчисления.  3)  4) Дано вещественное число. Проверить: принадлежит ли данное число промежутку: [0;3].  5) |
| **9** | 1) а)double(2 знака после запятой); b)int;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти их произведение, а затем провести операцию побитового «И» над произведением и вторым числом. Вывести результат в восьмеричной системе исчисления.  3)  4) Дано вещественное число. Проверить: принадлежит ли данное число промежутку: [-5;0].  5) |
| **10** | 1) a)float(4 знака после запятой); b)double(2 знака после запятой);  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти их сумму, а затем провести операцию побитового «И» над суммой и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дано целое число. Проверить: является ли данное число положительным трехзначным числом.  5) |
| **11** | 1) а)int(от 0 до 255); b)char;  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Найти остаток от деления первого числа на второе, а затем провести операцию побитового «ИЛИ» над остатком и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дано целое число. Проверить: является ли данное число отрицательным двухзначным числом.  5) |
| **12** | 1) а)int; b)double;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти остаток от деления первого числа на второе, а затем провести операцию побитового «И» над остатком и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дано целое число. Проверить: является ли данное число положительным пятизначным числом.  5) |
| **13** | 1) а)int; b)float;  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Умножить первое число на второе, а затем провести операцию побитового «Исключающего ИЛИ» над произведением и первым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой латинского алфавита от “a” до “p” (нижний регистр).  5) |
| **14** | 1) а)char; b)float;  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Разделить первое на второе, а затем частное сдвинуть на один разряд вправо. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой латинского алфавита от “s” до “y” (нижний регистр).  5) |
| **15** | 1) а)char; b)double;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти остаток от деления первого числа на второе, а затем провести операцию побитового «ИЛИ» над остатком и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой русского алфавита от “а” до “н” (нижний регистр).  5) |
| **16** | 1) а)char; b)int;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти их сумму, а затем провести операцию побитового «И» над суммой и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой русского алфавита от “о” до “я” (нижний регистр).  5) |
| **17** | 1) а)double(2 знака после запятой); b)float(4 знака после запятой);  2) Два незнаковых целых числа вводятся с клавиатуры в восьмеричной системе исчисления. Найти остаток от деления первого числа на второе, а затем провести операцию побитового «ИЛИ» над остатком и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой русского алфавита от “А” до “М” (верхний регистр).  5) |
| **18** | 1) а)double(4 знака после запятой); b)float(1 знак после запятой);  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти их произведение, а затем произведение сдвинуть на 4 разряда влево. Вывести результат в восьмеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой русского алфавита от “Н” до “Я” (верхний регистр).  5) |
| **19** | 1) а)double(2 знака после запятой); b)int;  2) Два незнаковых целых числа вводятся с клавиатуры в десятичной системе исчисления. Найти остаток от деления первого числа на второе, а затем провести операцию побитового «И» над остатком и вторым числом. Вывести результат в шестнадцатеричной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой латинского алфавита от “A” до “N” (верхний регистр).  5) |
| **20** | 1) a)float(4 знака после запятой); b)double(2 знака после запятой);  2) Два незнаковых целых числа вводятся с клавиатуры в шестнадцатеричной системе исчисления. Найти их произведение, а затем провести операцию побитового «И» над произведением и вторым числом. Вывести результат в десятичной системе исчисления.  3)  4) Дан символ. Определить: является ли данный символ буквой латинского алфавита от “O” до “Z” (верхний регистр).  5) |