**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Программирование

**Отчет по лабораторной работе № 1**

**Тема:** «Управляющие структуры языка Си»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-148 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Фаизова А. Н. |  |  |  |
| Принял | Гайнетдинова А. А. |  |  |  |

**Уфа 2017**

**Требования к составу отчета:**

Отчет к лабораторной работе оформляется в текстовом процессоре Microsoft Word или OpenOffice (LibreOffice) Write в соответствии с требованиями стандарта СТО УГАТУ 016-2007 и содержать

• титульный лист,

• описание цели работы и краткую теоретическую справку по использованным операторам и функциям языка программирования,

• описание выполнения задания:

1. формулировка задания, как в методичке,
2. блок-схема каждой разработанной функции,
3. исходный текст разработанного приложения,
4. скриншоты примеров выполнения программы.

• выводы

• список использованной литературы,

**Требования к оформлению отчета**

* Текст отчета набирается шрифтом Times New Roman, размер шрифта 14pt или 12pt единый во всем документе. Исходный код должен быть набран любым моноширинным шрифтом (например, Courier New), размер символов можно уменьшить до 10 pt. Поля страницы: верхнее и нижнее – 2 см., левое – 2 см, правое – 1.5 см.
* Абзац должен начинаться с красной строки, за исключением тех случаев, когда абзац разорван каким-либо математическим выражением. Выравнивание внутри текстовых абзацев «по ширине», выравнивание в блоке с исходным кодом – «по левому краю».
* Все блок-схемы оформляются в соответствие с ГОСТ 19.701. Описание ГОСТ и пример блок-схемы приведены в разделе Справочник, методических указаний по выполнению лабораторных работ на сайте. Если блок-схемы составляются в сторонних приложениях, следует предусмотреть возможность их правки в учебном классе. В противном случае отчет с некорректно составленной блок-схемой и исправлениями вручную приниматься не будут.
* Описание выполнения каждого задания начинается с новой страницы. Следует использовать заголовок с текстом «**Индивидуальное задание №\_\_**», набранный полужирным шрифтом с выравниванием по левому краю, без красной строки. Текст задания, текст в блок-схеме и описание программы следует набирать шрифтом Times New Roman с прямым начертанием (не курсив!!!).
* Текст на скриншотах должен быть читаемым. Скриншоты должны содержать только окно с результатами выполнения программы, а не весь рабочий стол. Размер шрифта на скриншотах должен соответствовать по размеру окружающему его тексту.

Ниже и выше красным выделены места, которые необходимо изменить.

**Цель:**  ознакомиться с базовыми принципами построения программ и основными конструкциями языка C.

**Теоретический материал**

Любой файл начинается с директив #include, вставляющих в текст программы так называемые заголовочные файлы, которые содержат описания функций, используемых в этом файле. В нашем примере это описания стандартных функций ввода-вывода (#include <stdio.h>) и математических функций (#include <math.h>).

Основная программа всегда называется именем main.

Любая программа содержит две основные части: информацию, которую она обрабатывает и набор команд, при выполнении которых происходит обработка этой информации.

Информацию, с которой работает программа, называют *данными*, и в программе она находиться в виде чисел, которые хранятся в ячейках оперативной памяти. Виды ячеек, которые можно использовать в программе для хранения чисел, называют *типами данных*.

Тип данных определяет размер ячейки, и внутреннее представление числа, хранящегося в ней, которое может быть целым со знаком, целым без знака и вещественным(числом с дробной частью).

Для обозначения целых типов используются следующие ключевые слова:

* int– целое число со знаком размером 2 байта(для 8, 16-разрядных ЭВМ) или 4 байта (для32, 64-разрядных ЭВМ);

Для обозначения символьных типов данных:

* char - символьный размером 1 байт, его можно рассматривать как целое -128...+127 (иногда 0...255);

Для обозначения вещественных типов:

* float– число с плавающей точкой одинарной точности размером

4 байта;

* double– число с плавающей точкой двойной точности размером

8 байт.

*Переменные* – это поименованные ячейки оперативной памяти, которые предназначены для хранения данных в программе. Переменные позволяют производить считывание и запись информации в связанные с ними ячейки.

Для того, что бы в программе получить переменные, их необходимо продекларировать. Это можно сделать в тексте программы после открывающейся фигурной скобки предложением, составленным по следующей форме:

тип\_данных список\_имен\_переменных;

список\_имен\_переменных– одно или несколько имен переменных, перечисленных через запятую.

Пример:

int abc ;

double x, y ;

В переменных, декларируемых таким образом, начальное значение заранее не определено (мусор). Для того чтобы задать переменной нужное начальное значение, необходимо после имени переменной в строке декларации поставить знак равно и указать это значение.

Для осуществления *обработки информации*, в программе необходимо записать предложения, содержащие наборы действий над нужными данными. Такие предложения на языке Си имеют следующую форму:

имя\_переменной= выражение;

выражение – любой набор констант, переменных и функций, связанных знаками операций.

! В конце каждого простого предложения в языке Си должна стоять точка с запятой, как признак окончания предложения. !

В языке С точка с запятой является элементом оператора и его завершающей частью, а не разделителем операторов, как в языке Pascal.

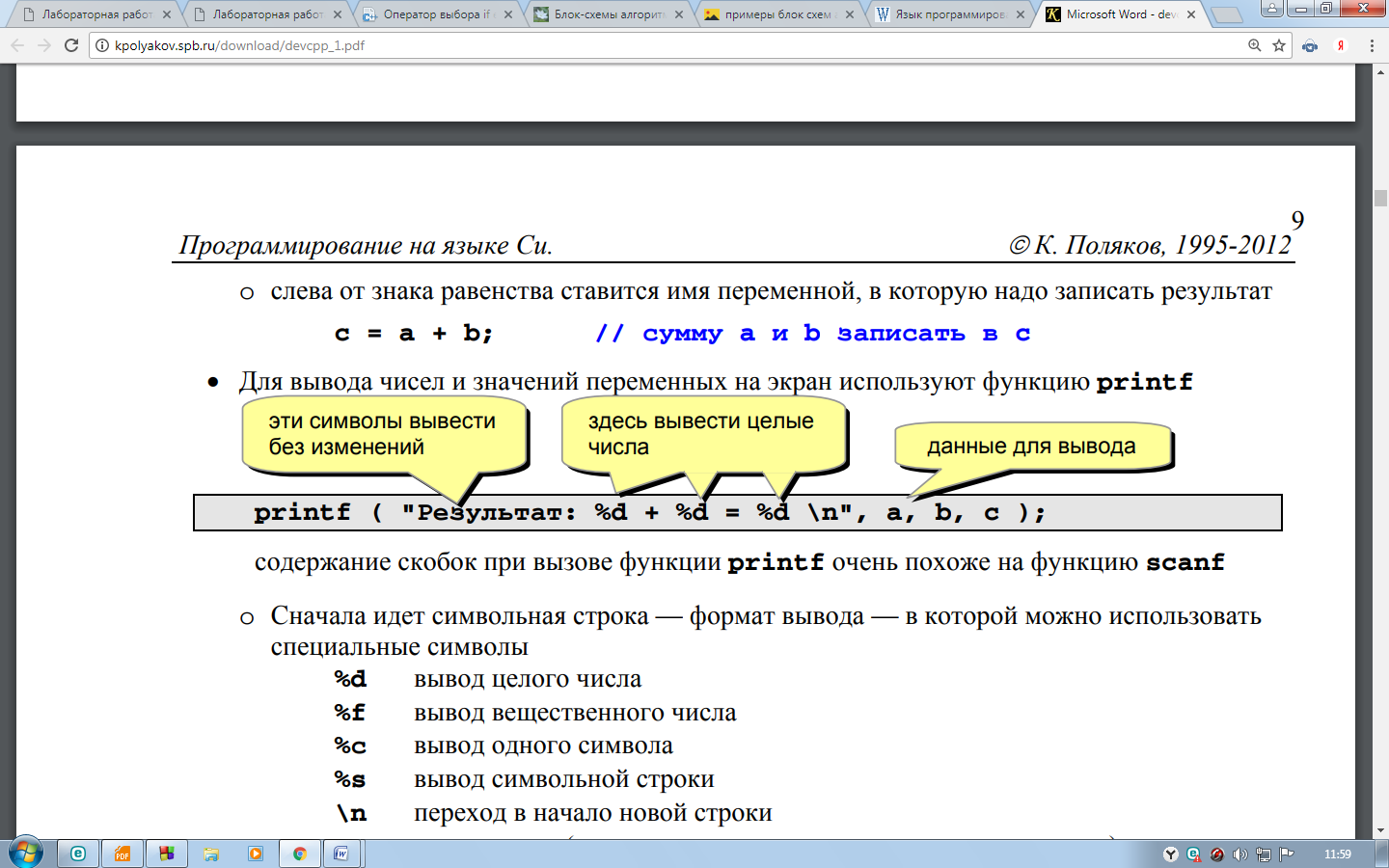
Фигурные скобки, { и }, служат для группировки объявлений и операторов в составные операторы, или блоки, синтаксически эквивалентные одному оператору. То есть несколько операторов, заключенных в фигурные скобки, рассматриваются компилятором как один оператор. Такие конструкции часто называют сложным или составным оператором. Можно также встретить понятие блок операторов, которое означает то же самое. Фигурные скобки, окружающие операторы тела функции, — это самый очевидный пример такого блока, а еще один — это скобки вокруг группы из нескольких операторов. После правой скобки, закрывающей блок, точка с запятой не ставится.

Организация ввода/вывода

printf(‘форматная\_строка’ , список\_информации )– функция осуществляет вывод на экран числовых и символьных данных в разных форматах.

*форматная\_строка* - строка-литерал. В форматной строке может быть любой набор символов, который без изменения будет выведен на экран, за исключением символа %. Этот символ интерпретируется функцией, как начало задания формата вывода данных. С помощью формата программист указывает функции место вывода информации, а также, в каком виде, и какого типа данные выводить на экран.

*список\_информации* - если в форматной строке заданы форматы вывода данных, то после форматной строки должен идти список переменных или выражений, перечисленных через запятую и их количество, а также порядок следования, должны совпадать с заданными форматами в форматной строке.



Ниже показаны примеры использования функции printf.

// вывод строки без указания форматов

printf("Hello World\n") ;

// вывод значения одной переменнойa типа float

printf ("Сторона квадрата равна %f см.\n", a) ;

// вывод значения двух переменныхs иp типа double

printf("Площадь равна %lf см. кв., периметр равен%lf см.\n",

s, p) ;

Aрифметические операции

+ сложение двух операндов;

- вычитание второго операнда из первого;

\* умножение двух операндов;

/ деление первого операнда на второй;

% остаток от деления целых чисел;

++ инкремент, или увеличение переменной на единицу;

-- декремент, или уменьшение переменной на единицу.

Операции отношений и логические операции

< меньше;

> больше;

<= меньше или равно;

>= больше или равно;

== равно;

!= не равно;

&&- логическое И.

||- логическое ИЛИ.

Операторы:

Ветвление.

Оператор *if-else* выражает процесс принятия альтернативных решений. Его формальный синтаксис таков:

if (выражение) оператор1

else оператор2

Часть, начинающаяся со слова else, необязательна. Условный оператор в котором отсутствует часть else называется сокращенным, а условный оператор, в котором присутствует часть else – полным.

Циклы.

Счетный оператор цикла for

Первый оператор цикла в Си *for* часто называют счетным оператором цикла. Связано это с тем, что в заголовке определяется некоторая управляющая переменная (переменная-счетчик), ей присваивается некоторое начальное значение, после каждого прохода переменная счетчик изменяется (увеличивается или уменьшается), а цикл завершается когда переменная счетчик достигает некоторого критического значения.

for(<переменная счетчик>=<нач.знач.>; <условие завершения>; <изменение переменной-счетчика>)

{

<тело цикла>

}

Первая строка, включающая в себя служебное слово for и выражения в круглых скобках, называется заголовком цикла, а все, что указывается после заголовка внутри фигурных скобок, телом цикла.

Условный оператор цикла с пред-условием while

Общий вид условного оператора цикла с предусловие выглядит следующим образом:

while(<условие>)

{

<тело цикла>

}

Пока условие в скобках истинно (не ложно) выполняется тело цикла.

Условный оператор цикла с пост-условием do ... while

Этот цикл очень похож на предыдущий, но условие в нем проверяется по завершении выполнения тела цикла. Таким образом, тело цикла с пост-условием всегда хотя бы один раз будет выполнено. Общий вид:

do

{

<тело цикла>

} while(<условие>)

**Ответы на контрольные вопросы**

1. Перечислите суффиксы, определяющие тип целой константы. Какой тип имеет целочисленная константа без суффикса?

Целую константу можно представить в виде беззнакового целого с помощью суффикса U (или u). Например, Константе 200U выделяется 1 байт, и старший бит используется для представления одного из разрядов кода числа и диапазон значений становится от 0 до 255. Суффикс L (или l) позволяет выделить целой константе 8 байт (long int).

Совместное использование в любом порядке суффиксов U (или u) и L (или l) позволяет приписать целой константе тип unsigned long int, и она займет в памяти 64 разряда, причем знаковый разряд будет использоваться для представления разряда кода (а не знака).

1. Перечислите суффиксы, определяющие тип вещественной константы. Какой тип имеет вещественная константа без суффикса?

По умолчанию компилятор присваивает вещественному числу тип double. Если программиста не устраивает тип, который компилятор приписывает константе, то тип можно явно указать в записи константы с помощью следующих суффиксов:

F (или f) — float для простых вещественных констант,

L (или l) — long double для вещественных констант двойной расширенной точности.

Примеры:

3.14159F — константа типа float, занимающая 4 байта;

3.14L — константа типа long double, занимающая 10 байт.

1. Что такое «переменная»?

Переменная — поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются значением этой переменной.

Чтобы программа могла обратиться к переменной (области памяти), например, для того, чтобы получить исходные данные для расчета по формуле или сохранить результат, переменная должна иметь имя. Имя переменной придумывает программист.

1. Объясните разницу между префиксной и постфиксной формами операций инкремента и декремента.

[Инкремент](http://codelessons.ru/cplusplus/syntax/inkrement-i-dekrement-v-c.html#inc) — увеличивает переменную на единицу.

[Декремент](http://codelessons.ru/cplusplus/syntax/inkrement-i-dekrement-v-c.html#dec) — уменьшает переменную на единицу.

Чтобы увеличить переменную при помощи инкремента, нужно после имени переменной или до него написать два плюса (++). Применение декремента аналогично инкременту. Однако для уменьшения переменной мы должны вместо двух знаком сложения прописать два минуса (—).

Рассмотрим префиксную и постфиксную варианты записи изучаемых нами операторов:

* До переменной (++x --y) :

Данное положения является префиксным. При префиксном инкременте или декременте, наша переменная будет уменьшена (увеличена) до того, как будет использовано ее значение в нуждах программы. Для лучшего понимания, давайте рассмотрим простой пример, который наглядно демонстрирует префиксное расположение инкремента:

int v = 4, sum = ++v; // sum и v равны пяти

* После переменной (x++ y--) :

Данное расположения называется постфиксным. При использовании постфиксного инкремента или декремента, переменная будет увеличена после использования ее значения для выполнения сторонних операций. Вот тот же пример, лишь с применением постфиксного инкремента:

int v = 4, sum = v++; // sum равна четырём

1. К каким операндам применимы операции инкремента и декремента?

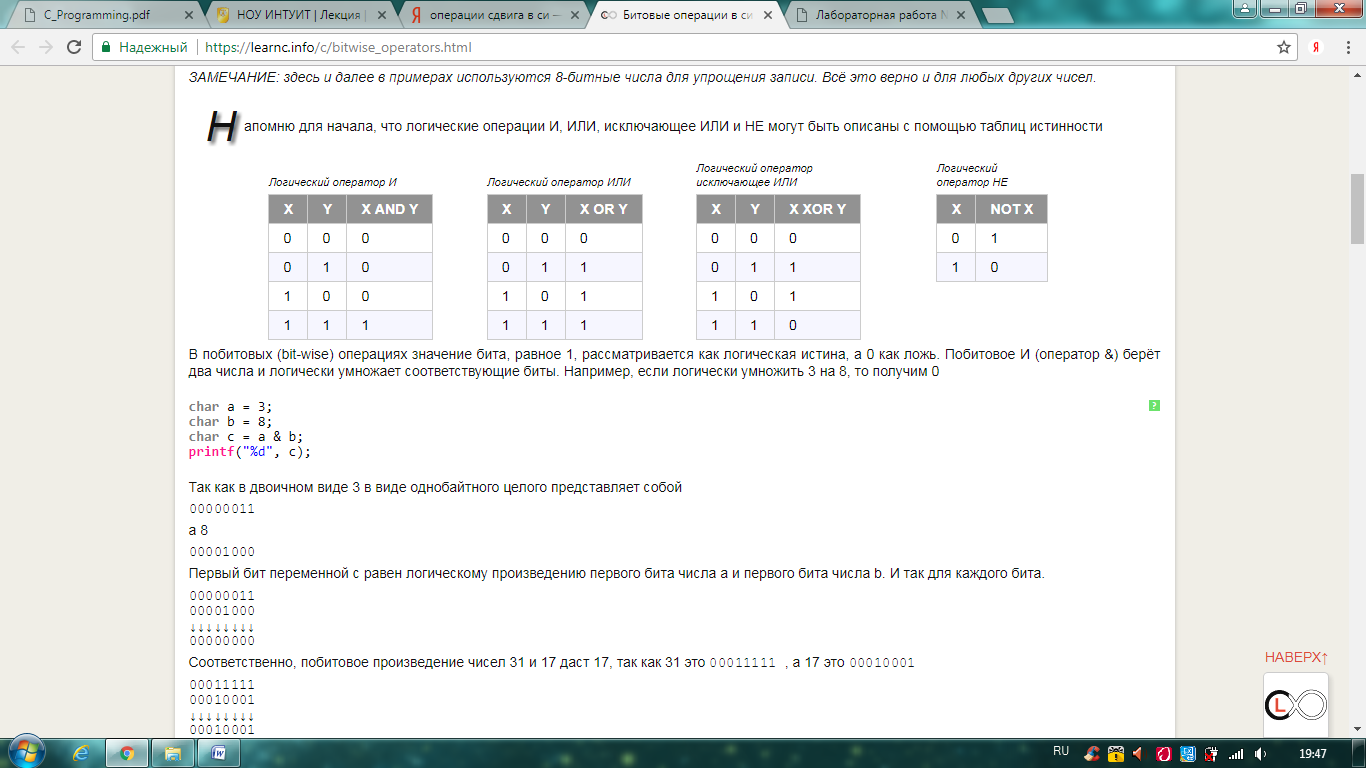
Операции инкремента и декремента могут применяться только к переменным и не могут – к константам и выражениям.

1. Приведите примеры использования поразрядных операций и операций сдвига.

Побитовые И, ИЛИ, НЕ, исключающее ИЛИ

ЗАМЕЧАНИЕ: здесь и далее в примерах используются 8-битные числа для упрощения записи. Всё это верно и для любых других чисел.

Логические операции И, ИЛИ, исключающее ИЛИ и НЕ могут быть описаны с помощью таблиц истинности



В побитовых (bit-wise) операциях значение бита, равное 1, рассматривается как логическая истина, а 0 как ложь. Побитовое И (оператор &) берёт два числа и логически умножает соответствующие биты. Например, если логически умножить 3 на 8, то получим 0

char a = 3;

char b = 8;

char c = a & b;

printf("%d", c);

Первый бит переменной c равен логическому произведению первого бита числа a и первого бита числа b. И так для каждого бита.

00000011

00001000

↓↓↓↓↓↓↓↓

00000000

Соответственно, побитовое произведение чисел 31 и 17 даст 17, так как 31 это 00011111 , а 17 это 00010001

00011111

00010001

↓↓↓↓↓↓↓↓

00010001

Побитовое произведение чисел 35 и 15 равно 3.

00100011

00001111

↓↓↓↓↓↓↓↓

00000011

Аналогично работает операция побитового ИЛИ (оператор |), за исключением того, что она логически суммирует соответствующие биты чисел без переноса.

Например,

char a = 15;

char b = 11;

char c = a | b;

printf("%d", c);

выведет 15, так как

a=00001111

b=00001011

↓↓↓↓↓↓↓↓

c=00001111

Побитовое ИЛИ для чисел 33 и 11 вернёт 43, так как 33 это 00100001, а 11 это 00001011

00100001

00001011

↓↓↓↓↓↓↓↓

00101011

Побитовое отрицание (оператор ~) работает не для отдельного бита, а для всего числа целиком. Оператор инверсии меняет ложь на истину, а истину на ложь, для каждого бита. Например,

char a = 65;

char b = ~a;

printf("%d", b);

Выведет -66, так как 65 это 01000001, а инверсия даст 10111110

что равно -66. Кстати, вот алгоритм для того, чтобы сделать число отрицательным: для нахождения дополнительного кода числа его надо инвертировать и прибавить к нему единицу.

char a = 107;

char b = ~a + 1;

printf("a = %d, -a = %d", a, b);

Исключающее ИЛИ (оператор ^) применяет побитово операцию XOR. Например, для чисел

char a = 12;

char b = 85;

char c = a ^ b;

printf("%d", c);

будет выведено 89, так как a равно 00001100, а b равно 01010101. В итоге получим 01011001

Иногда логические операторы && и || путают с операторами & и |. Такие ошибки могут существовать в коде достаточно долго, потому что такой код в ряде случаев будет работать. Например, для чисел 1 и 0. Но так как в си истиной является любое ненулевое значение, то побитовое умножение чисел 3 и 4 вернёт 0, хотя логическое умножение должно вернуть истину.

int a = 3;

int b = 4;

printf("a & b = %d\n", a & b); //выведет 0

printf("a && b = %d\n", a && b);//выведет не 0 (конкретнее, 1)

-----------------------------------------------------------------------------------

Операций сдвига две – битовый сдвиг влево (оператор <<) и битовый сдвиг вправо (оператор >>). Битовый сдвиг вправо сдвигает биты числа вправо, дописывая слева нули. Битовый сдвиг влево делает противоположное: сдвигает биты влево, дописывая справа нули. Вышедшие за пределы числа биты отбрасываются.

Например, сдвиг числа 5 влево на 2 позиции

00000101 << 2 == 00010100

Сдвиг числа 19 вправо на 3 позиции

00010011 >> 3 == 00000010

Так как сдвиг вправо (>>) дописывает слева нули, то для целых чисел операция равносильна целочисленному делению пополам, а сдвиг влево умножению на 2. Произвести битовый сдвиг для числа с плавающей точкой без явного приведения типа нельзя. Это вызвано тем, что для Си не определено представление числа с плавающей точкой. Однако можно переместить число типа float в int, затем сдвинуть и вернуть обратно

float b = 10.0f;

float c = (float) (\*((unsigned int\*)&b) >> 2);

printf("%.3f >> 2 = %.3f", b, c);

Но мы, конечно же, получим не 5.0f, а совершенно другое число.

Особенностью операторов сдвига является то, что они могут по-разному вести себя с числами со знаком и без знака, в зависимости от компилятора. Действительно, отрицательное число обычно содержит один бит знака. Когда мы будем производить сдвиг влево, он может пропасть, число станет положительным. Однако, компилятор может сделать так, что сдвиг останется знакопостоянным и будет проходить по другим правилам. То же самое и для сдвига вправо.

----------------------------------------------------------------------------------------

Побитовые операторы и операторы сдвига не изменяют значения числа, возвращая новое. Они также как и арифметические операторы, могут входить в состав сложного присваивания

int a = 10;

int b = 1;

a >>= 3;

a ^= (b << 3);

1. Знаки каких бинарных операций могут использоваться в составных операциях присваивания?

Ответ: + \* / % << >> & ^ |

Кроме простого присваивания имеется целая группа операций присваивания, которые объединяют простое присваивание с одной из бинарных операций. Такие операции называются составными операциями присваивания и имеют вид:

(операнд-1) (бинарная операция) = (операнд-2)

(i += 2;).

Составное присваивание по результату эквивалентно следующему простому присваиванию:

(операнд-1) = (операнд-1) (бинарное операция) (операнд-2)

(i = i + 2;),

Отметим, что выражение составного присваивания с точки зрения реализации не эквивалентно простому присваиванию, так как в последнем операнд-1 вычисляется дважды.

Каждая операция составного присваивания выполняет преобразования, которые осуществляются соответствующей бинарной операцией.

Большинству бинарных операторов (аналогичных + и имеющих левый и правый операнды) соответствуют

операторы присваивания ор=, где ор — один из операторов

+ \* / % << >> & ^ |

Если выр1 и выр2 — выражения, то

выр1 ор= выр2

эквивалентно

выр1 = (выр1) ор (выр2)

с той лишь разницей, что выр1 вычисляется только один раз. Обратите внимание на скобки вокруг выр2:

x \*= y + 1

эквивалентно

x = x \* (y + 1)

но не

x = x \* y + 1

Кроме того, в сложных выражениях вроде

yyval[yypv[p3+p4] + yypv[p1+p2]] += 2

благодаря оператору присваивания += запись становится более легкой для понимания, так как читателю при такой записи не потребуется старательно сравнивать два длинных выражения, совпадают ли они, или выяснять, почему они не совпадают. Следует иметь в виду и то, что подобные операторы присваивания могут

помочь компилятору сгенерировать более эффективный код.

Типом и значением любого выражения присваивания являются тип и значение его левого операнда после завершения присваивания.

1. Назовите обязательные этапы обработки, которые проходит исходная программа, подготовленная на языке Си в виде текстового файла.

Исходная программа, подготовленная на языке Си в виде текстового файла, проходит три обязательных этапа обработки:

• препроцессорное преобразование текста;

• компиляция;

• компоновка (редактирование связей или сборка).

Только после успешного завершения всех перечисленных этапов формируется исполняемый машинный код программы.

Задача препроцессора – преобразование текста программы до ее компиляции. Правила препроцессорной обработки определяет программист с помощью директив препроцессора.

1. Как обозначается пустой оператор?

Пустой оператор состоит только из точки с запятой. При выполнении ничего не происходит. Используется в операторах for, while, if в случаях, когда тело оператора не требуется, хотя по синтаксису требуется хотя бы один оператор. А также, когда необходимо пометить фигурную скобку меткой. Синтаксис требует, чтобы после метки обязательно следовал оператор. Фигурная скобка оператором не является. Поэтому, если надо передать управление на фигурную скобку, необходимо использовать пустой оператор.

1. Перечислите операторы цикла языка Си.

Операторы цикла (**for, while…do**).

Счетный оператор цикла for

Первый оператор цикла в Си *for* часто называют счетным оператором цикла. Связано это с тем, что в заголовке определяется некоторая управляющая переменная (переменная-счетчик), ей присваивается некоторое начальное значение, после каждого прохода переменная счетчик изменяется (увеличивается или уменьшается), а цикл завершается когда переменная счетчик достигает некоторого критического значения.

for(<переменная счетчик>=<нач.знач.>; <условие завершения>; <изменение переменной-счетчика>)

{

<тело цикла>

}

Первая строка, включающая в себя служебное слово for и выражения в круглых скобках, называется заголовком цикла, а все что указывается после заголовка внутри фигурных скобок телом цикла.

Условный оператор цикла с пред-условием while

Общий вид условного оператора цикла с предусловие выглядит следующим образом:

while(<условие>)

{

<тело цикла>

}

Пока условие в скобках истинно (не ложно) выполняется тело цикла.

Условный оператор цикла с пост-условием do ... while

Этот цикл очень похож на предыдущий, но условие в нем проверяется по завершении выполнения тела цикла. Таким образом, тело цикла с пост-условием всегда хотя бы один раз будет выполнено. Общий вид:

do

{

<тело цикла>

} while(<условие>)

1. Перечислите операторы ветвления языка Си.

Условные операторы (**if, switсh**).

Оператор *if-else* выражает процесс принятия альтернативных решений. Его формальный синтаксис таков:

if (выражение) оператор1

else оператор2

Часть, начинающаяся со слова else, необязательна. Условный оператор в котором отсутствует часть else называется сокращенным, а условный оператор, в котором присутствует часть else – полным.

1. В чем сходства и различия операторов break и continue?

Операторы break и continue выполняют прерывание некоторых вычислительных процессов (Например: в операторе switch, в котором прерывание просто необходимо, while … do, for).

Оператор break приводит к прекращению выполнения того атомарного блока кода, в котором он применяется.

В результате применения оператора continue прерывается только текущая итерация (проход) цикла. Происходит переход к следующей итерации. В цикле while или do это означает немедленную проверку условия, тогда как в цикле for дополнительно выполняется инкрементирование (изменение переменной-счетчика). Оператор continue применим только к циклам, но не к оператору switch. Если поставить continue внутри switch, в свою очередь находящегося внутри цикла, то управление будет передано на следующий проход этого цикла.

**Индивидуальное задание №1**

Задание: Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и такого же количества обложек к ним. Количество и цена задаются соответствующими переменными. Проинициализировать переменные некоторыми значениями, исходные данные и результат расчета вывести на экран. Предусмотреть тестовый пример, который легко решается в уме.

Блок-схема

Начало

Ввод

k

k = 0

sum:=k(tetrCost++oblCost)

sum:=tetrCost++oblCost

Вывод

sum

Конец

да

нет

Ввод

tetrCost, oblCost

sum:= 0

Начало алгоритма

Ввод цены тетради и обложки

Ввод количества наборов

Стоимость покупки приравниваем к начальному значению

Проверка выполнения примера, решающегося в уме

Расчёт стоимости покупки

Расчёт стоимости покупки

Вывод стоимости покупки

Конец алгоритма

Описание программы**:**

Переменные:

k – количество наборов=тертадь+обложка (т.к. количество тетрадей и обложек одинаковы);

tetrCost - цена одной тетради;

oblCost - цена одной обложки;

sum - стоимость покупки.

Исходный код программы

#include <stdio.h>

int main() {

int k = 10, tetrCost = 5, oblCost = 8, sum = 0;

if (k == 1)

{

sum = tetrCost + oblCost;

}

else

{

sum = k\*( tetrCost + oblCost);

}

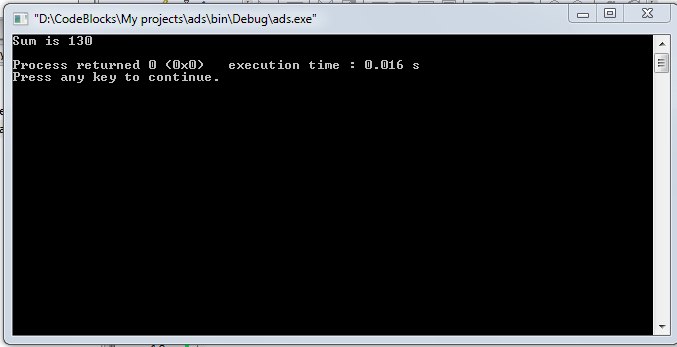
printf ("Sum is %d\n", sum);

return 0;

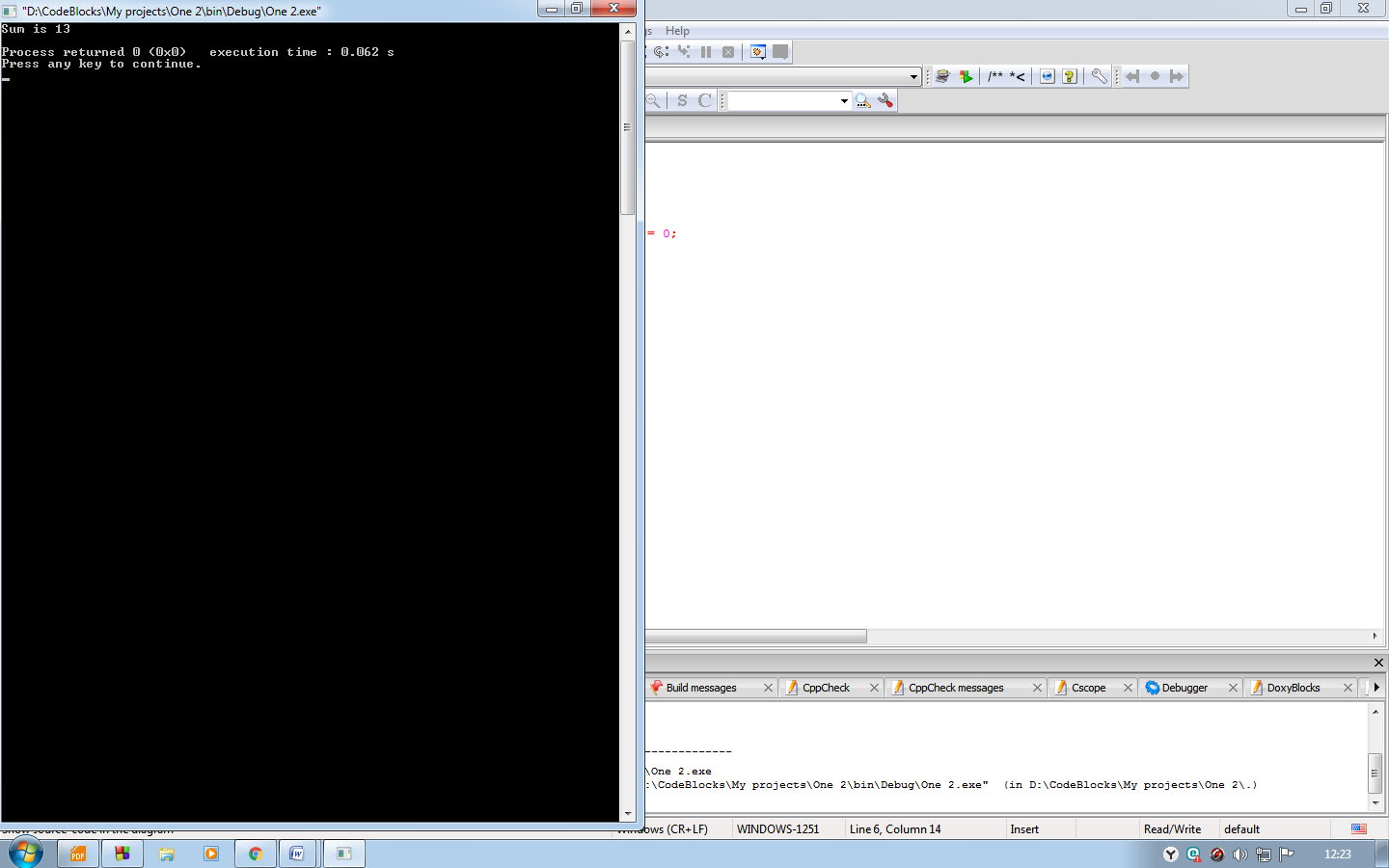
}

Пример выполнения программы

k = 10, tetrCost = 5, oblCost = 8;



k = 1, tetrCost = 5, oblCost = 8;



**Индивидуальное задание №2**

Задание: Написать программу для решения квадратного уравнения по заданным трем коэффициентам. Предусмотреть проверку некорректности исходных данных (например, уравнение не является квадратным, дискриминант отрицателен...).

Блок-схема

Расчёт значения единственного корня

Выяснение кол-ва корней квадр. ур-я

Начало алгоритма

Начало

Ввод

a, b, c

D:=0.0;

x1:=0.0;

x2:=0.0

D:=b2 - 4ac

a≠0 И D≥0

D = 0

нет

да

да

нет

Вывод

Не квадратн. ур-е или нет корней

x1:=;

x2:=

x1:=

Вывод

x1

Вывод

x1, x2

Конец

Ввод значений коэффициентов квадратного ур-я

Присвоение нулевого значения переменным, отвечающим за дискриминант, один корень, другой корень

Расчёт дискриминанта

Проверка корректности исходных данных

Вывод строки с сообщением о некорректности исх.данных

Вычисление корней квадр.ур-я

Вывод значения корня квадр.ур-я

Вывод значений корней квадр.ур-я

Конец алгоритма

Описание программы**:**

Переменные:

a – первый (главный) коэффициент;

b – второй коэффициент;

c – свободный член

D – дискриминант квадр. уравнения;

x1- первый корень;

x2 – второй корень;

Исходный код программы

#include <stdio.h>

#include <math.h>

int main () {

int a = 2, b = 4, c = 7;

float D = 0.0, x1 = 0.0, x2 = 0.0;

D = (b \* b) - (4 \* a \* c);

if ( a != 0 && D >=0 )

{

if (D == 0) {x1 = ((-b) + sqrt(D))/(2 \* a);

printf(" x = %f\n", x1);}

else {

x1 = ((-b) + sqrt(D))/(2 \* a);

x2 = ((-b) + -sqrt(D))/(2 \* a);

printf (" x1 = %f, x2 = %f\n", x1, x2);

}

}

else

{

printf (" No quadratic equation or No roots \n");

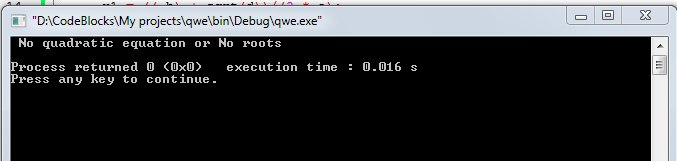
}

return 0;

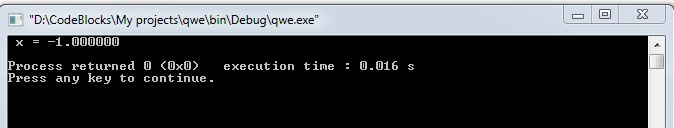
}

Пример выполнения программы

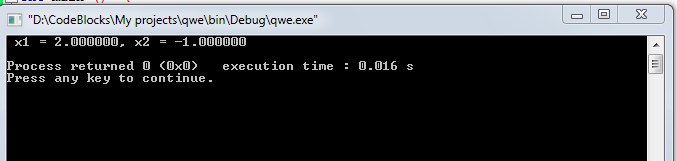
a = 2, b = 4, c = 7;



a = 1, b = 2, c = 1;



a = 1, b = -1, c = -2;



**Индивидуальное задание №3**

Задание: Задаются числа m и n. Определить k – сумму всех нечётных кратных 7 между ними. Результат расчёта вывести на экран.

Блок-схема

Начало алгоритма

Начало

Ввод

m, n

Ввод значений концов интервала

sum:=0;

i:=0

Переменной, отвечающей за сумму и переменной цикла присваиваем начальные(нулевые) значения

Цикл-проход по элементам интервала(m,n); переменная цикла i принимает начальное значение числа, следующего после нижней границы ин-ла до верхнеё границы с шагом 1

Проверка условия «Число делится на 7 и нечётно»

i mod 7 = 0

И

i mod 2≠0

i:=m+1,n;1

sum:=sum+i

Расчёт суммы необходимых чисел

Вывод значения суммы

Конец алгоритма

Вывод

sum

Конец

да

нет

Описание программы**:**

Переменные:

m – левый конец (нижняя граница) интервала;

n –правый конец (верхняя граница) интервала;

i –переменная цикла (исходная переменная цикла);

sum –сумма всех нечётных кратных 7 чисел в интервале (m, n);

Исходный код программы

#include <stdio.h>

#include <math.h>

int main() {

int m = 8, n = 22, sum = 0;

int i =0;

for ( i = m; i < n; i++)

{if ( i%7 == 0 && i%2 != 0)

{ sum = sum+i;

}

}

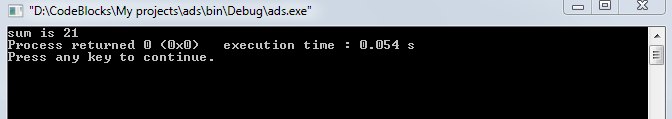
printf("sum is %d", sum);

return 0;

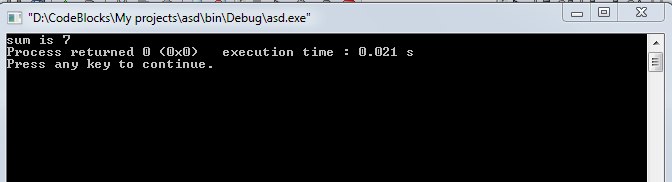
}

Пример выполнения программы

m=8, n=22



m=5, n=15



**Вывод:**

На данной лабораторной работе мы знакомились с базовыми принципами построения программ и основными конструкциями языка C.

**Список использованной литературы**

1. Куцый О. Я. Программирование на языке Си. Москва.: МГТУ им. Э. Н. Баумана. 2010. 84 с.
2. Управляющие структуры языка Си [Электронный ресурс] / Бикмеев А. Т. // <http://bikmeyevat.ugatu.su/students/CPP/Lab101/index01.html> (Дата обращения 30.10.2017).
3. Информатика и программирование. Лекции по С и С++ [Электронный ресурс] // <http://baumanki.net/lectures/10-informatika-i-programmirovanie/333-lekcii-po-s-i-s/4463-konstanty-s.html> (Дата обращения 2.10.2017).
4. Керниган Б., Ритчи Д. Язык программирования Си. Пер. с англ., 3-е изд., испр. — СПб.: "Невский Диалект", 2001. - 352 с.