ХОЧУ В ЛИЦЕЙ!

10

Учебные материалы
для подготовки
к вступительным испытаниям
по информатике
для поступающих в 10-й класс
Лицея НИУ ВШЭ

Т.А. Клюева,
Н.В. Копытова,
В.В. Куренков

Под общей редакцией
В.В. Куренкова

Авторы заданий: Т. А. Клюева,
Н. В. Копытова,
В. В. Куренков

Под общей редакцией В. В. Куренкова

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Пособие «Хочу в Лицей!» по информатике разработано для учащихся школ и лицеев, желающих самостоятельно подготовиться к выполнению задания по информатике, являющегося частью комплексного теста для поступающих в 10-й класс Лицея НИУ ВШЭ.
Книга состоит из девяти глав, в которых авторы последовательно и подробно разбирают структуру и особенности заданий.
В пособии подробно разобран демовариант и типовые задачи, которые могут встретиться на вступительных испытаниях по информатике. Разобраны типовые ошибки, которые допускают ученики на вступительных испытаниях. Даны рекомендации по работе с тестирующей системой.
В конце пособия приведены краткие учебники по Python и C++, которые содержиат короткие пояснения некоторых базовых структур алгоритмов, типовых алгоритмов работы с массивами, за которыми идут примеры кода. Примеры могут быть использованы как основа для собственных решений задач вступительного экзамена по информатике в 10-й класс Лицея НИУ ВШЭ.

ГЛАВА 1

СТРУКТУРА ЗАДАНИЯ: ОСОБЕННОСТИ УСЛОВИЯ

В этой главе мы рассмотрим, как выглядят условия заданий по информатике, как описаны ограничения на входные и выходные данные в заданиях.

Условие задания состоит из нескольких частей: описания условия, описания данных, которые подаются на вход, и требований, что программа должна получить в результате работы. Рассмотрим несколько примеров заданий.

ПРИМЕР ЗАДАНИЯ

Задание 1. На вход подается натуральное число N, необходимо удалить из записи числа N цифры 0 и 1, сохранив порядок остальных цифр. Вывести полученный результат на экран.
Входные данные:
2400915
Выходные данные:
2495

ПРИМЕР ЗАДАНИЯ

Задание 5. Дана матрица A размером 6 x 6, заполненная целыми числами, не превосходящими по модулю 10^4. Требуется поменять местами максимальный элемент нижнего треугольника (включая главную диагональ) матрицы A с максимальным элементом верхнего треугольника.

Входные данные:

```
 1  2  3  4  5  6
 7  8  9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
```

Выходные данные:

```
 1  2  3  4  5  6
 7  8  9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 36
31 32 33 34 35 30
```

Далее идет часть, в которой описываются входные данные к задаче:

- в первом примере входные данные представлены натуральным числом N;
- во втором примере вводятся шесть строк, каждая из которых состоит из шести чисел, разделенных пробелами.

В описании формата входных данных указываются **ограничения** на эти данные:
- в первом примере число N является натуральным, при этом количество разрядов в числе неизвестно;
- во втором – элементами матрицы являются целые числа, не превосходящие по модулю 10^4.

Ограничения на входные данные позволяют понять, какие типы данных использовать для хранения чисел (например, если входное число должно быть меньше 10^6, то для хранения этого числа понадобится 32-битный целочисленный тип, например, тип **integer** для языка **Pascal** не подойдет).

В следующей части описываются требования к тому, что должна вывести программа:
- в первом примере ответом будет одно натуральное число, полученное из исходного и не содержащее цифр 0 и 1;
- во втором – матрица исходной размерности, в которой поменяли местами максимальный элемент нижнего треугольника (включая главную диагональ) с максимальным элементом верхнего треугольника.

Программы проверяются с помощью автоматической системы. Поэтому выходные данные должны точно соответствовать формату, описанному в условии задания. Для лучшего понимания в конце условия даются примеры того, что должно быть выведено программой при разных входных данных.
Автоматическая система проверки запускает программу несколько раз на заранее подготовленных наборах данных – тестах. Программа должна считать вводимые данные, провести обработку и вывести ответ. Этот ответ сравнивается с правильным, и, если ответы совпадают, тест засчитывается как пройденный. Если программа проходит все тесты успешно, вы получаете полный балл. В противном случае задание считается решенным неверно. Частичные баллы не предусмотрены.
Проверка происходит автоматически, без участия членов комиссии.

ПОЛЕЗНЫЕ СОВЕТЫ

1. Внимательно прочитать задание, разобраться, с каким типом данных придется работать.
2. Тщательно изучить, что должно получиться в результате работы программы. Это необходимо для четкого понимания, что именно требуется сделать, чтобы избежать недочетов и ошибок при сдаче в тестирующую систему.

ГЛАВА 2

СТРУКТУРА ЗАДАНИЯ: ТРЕБОВАНИЯ К РЕШЕНИЮ

В этой главе мы остановимся на основных требованиях, которые предъявляются к программам перед отправкой в тестирующую систему.

Решением каждого из пяти заданий является программа на одном из допустимых языков программирования. Проверка осуществляется автоматически. В заданиях 1–2 количество попыток проверки программы неограниченно. В заданиях 3–5 за каждое неверное решение, отправленное в тестирующую систему, снимается 1 балл (но не более двух). Поэтому, чтобы не лишиться баллов из-за пустяков, рассмотрим, как должно быть оформлено решение задачи.

- Программа должна считывать данные, подаваемые на вход. Для этого используются стандартные функции ввода-вывода языков программирования, например **input** – **print** в **Python, cin – cout** в **C++** или **read – write** в **Pascal**.

- Проверка решения происходит без участия человека, перед вводом данных не стоит выводить какие-то дополнительные сообщения, улучшающие интерфейс, например **print** («*Введите число*») и др. Эти сообщения не смогут быть проверены автоматической системой.

- Не нужно проверять данные на корректность (например, что число $X < 100$). Все ограничения в условии задания уже учтены в тестах. Лишняя проверка никак не повлияет на результат, так как никогда не сработает.

- Организуя вывод данных, не нужно выводить ничего лишнего, например дополнительное сообщение **print** («*Ответ:*»), поясняющие слова «*км*», «*минуты*» и др. Если требуется вывести два числа, то выводятся только эти числа, разделенные пробелом или переходом на новую строку (в зависимости от условия).

- В некоторых языках программирования часто ставят дополнительную задержку – **system (pause)** в C, **readln** в Pascal. Это тоже не нужно, так как программа должна просто завершить свою работу, не дожидаясь каких-либо действий со стороны пользователя.

- При выводе данных следует обратить внимание на формат выводимых значений. Так как действительное число может выводиться в экспоненциальной форме (например, 4.16e-4) или фиксированным количеством знаков после запятой.

ПОЛЕЗНЫЕ СОВЕТЫ

1. Рассмотреть примеры, приведенные в условии задания, определиться с количеством и форматом вводимых и выводимых данных.
2. Не использовать нестандартные возможности компилятора (IDE) или приемы, «украшающие» интерфейс программы.

ГЛАВА 3

РАЗБОР ЗАДАНИЯ 1: РЕШЕНИЕ, ТИПОВЫЕ ОШИБКИ

В главе приведен разбор задания 1 из демонстрационного варианта, а также некоторых возможных вариантов заданий по теме «Ввод, вывод, использование арифметических выражений, условный оператор, операторы итерации».

В данном задании требуется написать программу, реализующую ввод данных, обработку цифр числа и вывод.

Пример (демо). На вход подается натуральное число N, удалить из записи числа N цифры 0 и 1, сохранив порядок остальных цифр. Вывести полученный результат на экран.

Python

```
n = int(input())
i = 1
n_new = 0
while n != 0:
  if n % 10 != 0 and n % 10 != 1:
      n_new += n % 10 * i
      i *= 10
  n //= 10
print(n_new)
```

C++

```
#include<iostream>

using namespace std;

int main()
{
   int n, i = 1, n_new = 0;
   cin >> n;
   while (n != 0) {
       if(((n%10)!= 0)&&((n%10)!= 1)){
           n_new += (n % 10) * i;
           i *= 10;
        }
        n /= 10;
    }
   cout<<n_new;
   return 0;
 }
```

1. Для решения данной задачи будем использовать следующий алгоритм:

- найдем остаток от деления на 10 исходного числа, тем самым получим последнюю его цифру;
- если полученная цифра удовлетворяет условию (не «0» и не «1»), добавим эту цифру к новому числу;
- разделим нацело на 10 исходное число – избавимся от последней цифры в нем;
- снова найдем остаток от деления на 10 того, что осталось от первого числа, запомним эту цифру;
- после проверки условия умножим на 10 второе число, тем самым увеличим его разрядность до двух и сдвинем в разряд десятков;
- добавим к полученному второму числу запомненную ранее цифру из первого числа;
- будем повторять действия п. 3–6, пока исходное число не уменьшится до нуля, т.е. пока не избавимся от всех его разрядов.

2. Ввод данных.

Python: с помощью функции `int()` преобразуем введенную строку в целое число.

Потребуется переменная для хранения разрядов `i = 1` и переменная для формирования нового числа `n_new = 0`.
C++: перед вводом данных `cin >> n` объявим, что переменная `n` – целого типа – `int n`.
Указать тип переменных `int` потребуется и для переменных `int i = 1` и `int n_new = 0`.

3. Организуем проверку цифр числа.
   Так как количество разрядов числа неизвестно, проверку каждой цифры числа на соответствие условию будем выполнять с помощью цикла `while`. Условием прекращения работы цикла будет `n = 0`.

4. Чтобы получить **последнюю цифру числа**, нужно поделить число на основание системы счисления. Результат деления – остаток будет последней цифрой числа. Так как число записано в десятичной системе счисления, делим на 10. В обоих языках программирования запись будет выглядеть `n % 10`.

5. **Для уменьшения числа n на один разряд** используется операция «целочисленное деление»:
   Python: `n // 10`
   C++: `n / 10`
   В некоторых языках для этого есть особый оператор. Например, // в Python или div в Pascal. В C++ если оператор / применяется к целым числовым типам, то результатом деления будет целое число, если хотя бы одно из чисел (делимое или делитель) вещественное, то и результат будет вещественным.

6. Вывод результата можно организовать строкой:
   Python: `print(n_new)`
   C++: `cout << n_new`.

Предполагается, что данная задача должна быть решена арифметическим способом. Но в Python и C++ ее можно реализовать, используя строки.

Python

```
n = input()
s = ''
for i in n:
  if i != '0' and i != '1':
      s = s + i
print(s)
```

C++

```
# include <iostream>
# include <string>

using namespace std;

int main() {
    string n;
    string s;
    cin >> n;
    for (char i : n) {
        if ((i != '0' ) && (i != '1')){
            s = s + i;
        }
    }
    cout << s;
return 0;
}
```

Пример 2. На заданном промежутке [A, B], где A и B – натуральные четырехзначные числа, требуется найти числа, у которых десятичная запись симметрична. Вывести найденные цифры в строку через пробел.

Python

```
a = int(input())
b = int(input())
for i in range(a, b + 1):
  if i//100 - (i%10*10 + i%100//10) == 0:
    print(i, end=' ')
```

C++

```
#include <iostream>
using namespace std;

int main()
{
  int a, b;
  cin >> a >> b;
  for (int i = a; i <= b; ++i){
    if (i/100-(i%10*10 + i%100/10) == 0){
      cout<<i<<' ';
    }
  }
  return 0;
}
```

Если число симметрично, то два старших разряда равны двум младшим, записанным в обратном порядке, например 1221, 3553. Чтобы получить 0, достаточно из числа, образованного двумя старшими разрядами `n // 100` (в C++ записывается как `n / 100`), вычесть число из двух младших разрядов, записанных в обратном порядке: `n % 10 * 10 + n % 100 / 10`.

Пример 3. На вход подается натуральное число N, требуется найти и вывести на экран сумму максимальных цифр числа N.

Python

```
n = int(input())
count = 0
maximum = 0
while n > 0:
  if n % 10 > maximum:
    maximum = n % 10
    count = 1
  elif n % 10 == maximum:
    count += 1
  n = n // 10
print(count * maximum)
```

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, count = 0, maximum  = 0;
    cin >> n;
    while (n > 0){
        if (n % 10 > maximum){
            maximum = n % 10;
            count = 1;
        }
        else {
            if (n % 10 == maximum){
                count = count + 1;
            }
        }
        n = n / 10;
    }
    cout << count * maximum;
    return 0;
}
```

1. **Python:** с помощью функции `int()` преобразуем введенную строку в целое число. Потребуется переменная для хранения максимальной цифры – `maximum` и переменная `count` для подсчета количества цифр, равных максимальной.
   C++: перед вводом данных `cin >> n` объявим, что переменная `n` – целого типа – `int n`. Также задать тип потребуется для переменных `maximum` и `count`.

2. Переменной `maximum` присвоим 0 (в случае ввода числа 0 это значение будет единственной и максимальной цифрой), а цикл будем выполнять только при условии, что `n > 0`.

3. На каждой итерации цикла будем отделять последнюю цифру числа и сравнивать со значением `maximum`. Если она больше, заменять `maximum` этой цифрой. Также при каждом обновлении максимума переменной-счетчику `count` будем присваивать начальное значение `count = 1`. Если условие `n % 10 > maximum` не выполняется, это означает, что число или меньше, или в числе встретился еще один максимум, который тоже нужно учесть. Для этого используем проверку условия `n % 10 == maximum` и конструкцию `elif`.

4. Значение переменной `count` будем увеличивать на единицу в случае повторного появления максимальной цифры в числе (если `n % 10 == maximum`).

5. **Для уменьшения числа n на один разряд** используется операция «целочисленное деление»:
   Python: `n // 10`
   C++: `n / 10`
   В некоторых языках для этого есть особый оператор. Например, // в Python или div в Pascal. В C++ если оператор / применяется к целым числовым типам, то результатом деления будет целое число, если хотя бы одно из чисел (делимое или делитель) вещественное, то и результат будет вещественным.

6. Вывод результата можно организовать строкой:
   Python: `print(count * maximum)`
   C++: `cout << count * maximum`.

ТИПОВЫЕ ОШИБКИ

1. Входные данные считываются не так, как в условии. Например, в описании к заданию указано, что данные вводятся в разных строках, а пользовательской программой предусмотрено считывание данных из одной строки.
2. Отсутствие описания переменной (в C++) или функции `int, float` (в Python) при вводе числовых данных.
3. Неверное использование арифметических или логических операций: остаток от деления, целочисленное деление, возведение в степень, квадратный корень числа, логическое умножение (and), логическое сложение (or), логическое отрицание (not).
4. Несоответствие типов данных при работе с арифметическими операциями и выводе.
5. Ошибки в написании условного оператора, особенно при использовании множественного условия.

ГЛАВА 4

РАЗБОР ЗАДАНИЯ 2: РЕШЕНИЕ, ТИПОВЫЕ ОШИБКИ

В главе приведен разбор задания 2 из демонстрационного варианта, а также некоторых возможных вариантов заданий по теме «Операторы итерации».

Для выполнения этого задания требуется хорошо владеть использованием операторов цикла в изучаемом языке программирования. Как и при выполнении других заданий, здесь нужно найти закономерность, связывающую исходные данные и результат, а также определить величины, зависящие от итератора (счетчика) цикла.

Пример (демо)
Найти сумму чисел последовательности `s`:

$$s = \frac{5}{8} + \frac{7}{10} + \cdots + \frac{31}{34}$$

Результат вычислений вывести с точностью до трех знаков после запятой.

Python

```
s = 0
for i in range(5, 31 + 1, 2):
   s = s + i / (i + 3)
print('%.3f' % s)
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    float s = 0;
    for (int i = 5; i <= 31; i += 2){
        s = s + i / (i + 3.0);
    }
    cout << fixed << setprecision(3)<< s;
    return 0;
}
```

1. Инициализируем переменную `s`:
   Python: `s = 0`
   C++: объявим, что переменная `s` – действительное число типа `float`. Выбор такого типа данных связан с особенностью оператора деления «/». Для получения результата в виде числа с плавающей точкой необходимо, чтобы одно из чисел – делимое или делитель – было действительного типа.

2. Для организации цикла можно использовать числитель, изменяющийся от 5 до 31 с шагом 2, а знаменатель выразить через числитель (он отличается от числителя на 3).

3. Возможны и другие способы организации вычислений: для организации цикла в данном примере можно использовать специальную переменную (счетчик), определяющую номер итерации. Для определения общего количества повторений цикла (n) можно использовать формулу $n = (i_{кон} – i_{нач}) / h+1$. В данном случае при $i_{кон} = 31$, $i_{нач} = 5$, $h = 2$ получаем $n = 14$. Тогда программа будет иметь вид:

Python

```
s = 0
xh = 5, xk = 31, h = 2
a = xh
n = int((xk – xh) / h + 1)
for i in range(1, n + 1):
    s = s + i / (i + 3)
    a = a + h
print('%.3f' % s)
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    float s = 0, a;
    int n;
    double xh = 5, xk = 31, h = 2;
    a = xh;
    n = int((xk – xh) / h + 1);
    for (int i = 1; i <= n; i += 1){
        s = s + i / (i + 3.0);
        a = a + h;
    }
    cout << fixed << setprecision(3)<< s;
    return 0;
}
```

5. Используем форматированный вывод, чтобы ограничить количество знаков после запятой (оставляем три знака) `'%.3f'`.
   В C++ для форматированного вывода используют специальную функцию `setprecision()`, которой в качестве аргумента передается количество знаков после запятой. `Fixed` необходим для того, чтобы указать, что ограничения накладываются на размер дробной части числа, а целая выводится целиком, сколько бы цифр ни составляли ее запись.

Пример 2

На вход подается натуральное число N, не превосходящее 100. Напишите программу вычисления выражения:

$$\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\cdots+\frac{1}{N\cdot(N+1)}$$

Результат вычислений вывести с точностью до 3-ех знаков после запятой.

Python

```
n = int(input())
sum = 0
for i in range(1, n + 1):
    sum += 1 /(i * (i + 1))
print('%.3f' % sum)
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double n, sum;
    sum = 0;
    cin >> n;
    for (double i = 1; i <= n; ++i)
        sum += 1 /(i * (i + 1));
    cout << fixed << setprecision(3) << sum;
    return 0;
}
```

6. Введем значение переменной `n`:

   Python: с помощью функции `int` преобразуем введенную строку в целое число;

   C++: перед вводом данных `cin>> n` объявим, что переменные `n, sum` – действительные числа типа `double`. Выбор такого типа данных связан с особенностью оператора деления «/». Для получения результата в виде числа с плавающей точкой необходимо, чтобы одно из чисел – делимое или делитель – было действительного типа.

7. Так как количество итераций заранее известно, используем цикл `for`. Закономерность выражена формулой $\frac{1}{N\cdot(N+1)}$, где N возрастает от 1 с шагом 1. В программе используем вместо N итератор `i`, который меняет свое значение от 1 до N + 1

8. Используем форматированный вывод, чтобы ограничить количество знаков после запятой (оставляем три знака) `'%.3f'`.

   В C++ для форматированного вывода используют специальную функцию `setprecision()`, которой в качестве аргумента передается количество знаков после запятой. `Fixed` необходим для того, чтобы указать, что ограничения накладываются на размер дробной части числа, а целая выводится целиком, сколько бы цифр ни составляли ее запись.

Стоит отметить, что данную задачу можно решить и без цикла, если заметить, что

$$\frac{1}{k\cdot(k+1)}=\frac{1}{k}-\frac{1}{k+1}.$$

Тогда значение искомой суммы вычисляется следующим образом:

$$\Sigma_{k=1}^{n}\frac{1}{k\cdot(k+1)}=1-\frac{1}{n+1}.$$

Пример 3. Даны *n* целых чисел $a_1, a_2, \ldots a_n$. Найти сумму чисел последовательности, которые удовлетворяют условию $|a_i| < i^2$.

Входные данные

Вводится число n – количество чисел в последовательности, затем сами числа, каждое в новой строке. Гарантируется, что n не превосходит 10^4. Числа последовательности не превышают 1000.

Выходные данные

Выводится одно число – сумма тех чисел, которые удовлетворяют условию задания.

Python

```
n = int(input())
sum  = 0
for i in range(1, n + 1):
    a =  int(input())
    if abs(a) < i**2:
        sum += a
print(sum)
```

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, a, sum;
    sum  = 0;
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
    cin >> a;
    if (abs(a) < i*i){
        sum += a;
    }
    }
cout << sum;
return 0;
}
```

1. Введем значение переменной `n` – количество чисел последовательности:
   Python: с помощью функции `int` преобразуем введенную строку в целое число;
   C++: перед вводом данных `cin>> n` объявим, что переменные `n, a` и `sum` – целого типа – `int n, a, sum`.

2. Так как количество чисел в последовательности задается (равно `n`), ввод и проверку чисел осуществляем через цикл `for`. Сравнение чисел последовательности происходит с квадратом итератора *i*, поэтому указываем начальное значение 1, конечное *n* + 1.

3. Если число a_i по модулю меньше i_2, добавляем a_i к сумме `sum += a`.

4. После проверки всех чисел последовательности полученную сумму выводим на экран.

Пример 4. На вход подается натуральное число N, не превышающее 100. Напишите программу вычисления значения выражения:

$$1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\cdots+\frac{1}{N^2}.$$

Python

```
n = int(input())
sum  = 0
for i in range(1, n + 1):
    sum += 1 / i ** 2
print('%.3f' % sum)
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double n, sum;
    sum = 0;
    cin >> n;
    for (double i = 1; i <= n; ++i)
        sum += 1 / (i*i);
    cout << fixed << setprecision(3) << sum;
    return 0;
}
```

1. Введем значение переменной `n`:
   Python: с помощью функции `int` преобразуем введенную строку в целое число;
   C++: перед вводом данных `cin>> n` объявим, что переменные n, sum – действительные числа типа `double`. Выбор такого типа данных связан с особенностью оператора деления «/». Для получения результата в виде числа с плавающей точкой необходимо, чтобы одно из чисел – делимое или делитель – были действительными числами.

2. Так как количество итераций заранее известно, используем цикл `for`. Закономерность выражена формулой $\frac{1}{N^2}$, где N возрастает от 1 с шагом 1. В программе используем вместо N итератор *i*, который меняет свое значение от 1 до N + 1.

3. Используем форматированный вывод, чтобы ограничить количество знаков после запятой (оставляем три знака) `'%.3f'`.

Пример 5. На вход подается натуральное число N, не превышающее 10 000. Сколько чисел последовательности

$$\frac{1\cdot 2}{2};\frac{2\cdot 3}{2};\frac{3\cdot 4}{2};\ldots;\frac{N\cdot(N+1)}{2}$$

имеют в десятичной записи последнюю цифру 5?

Python

```
n = int(input())
a = 0
count = 0
for i in range(1, n + 1):
    a = (i * (i + 1) // 2)
    if a % 10 == 5:
        count += 1
print(count)
```

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, a, count;
    count = 0; a = 0;
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        a = (i * (i + 1) / 2);
        if (a % 10 == 5){
            count += 1;
    }
    }
    cout <<  count;
    return 0;
}
```

1. Введем значение переменной `n` – количество чисел последовательности:
   Python: с помощью функции `int` преобразуем введенную строку в целое число;
   C++: перед вводом данных `cin >> n` объявим, что переменные `n, a` и `count` – целого типа – `int n, a, count`.

2. Инициализируем переменные `a = 0, count = 0`.

3. Так как количество итераций заранее известно, используем цикл `for`. Закономерность выражена формулой $\frac{N\cdot(N+1)}{2}$, где N возрастает от 1 с шагом 1. В программе используем вместо N итератор *i*, который меняет свое значение от 1 до N + 1.

4. В цикле вычисляем каждый член последовательности `a = (i * (i + 1) // 2)` и проверяем последнюю цифру числа. Если она равна 5, увеличиваем счетчик `count` на единицу.

5. Выводим результат.

ТИПОВЫЕ ОШИБКИ

1. Часто не учитывается последняя итерация цикла или, напротив, добавляется лишняя. Это может быть связано с непониманием того, как обрабатывается условие цикла, или с ошибочной обработкой строгого неравенства как нестрогого или наоборот.
2. Отсутствует инициализация переменных, или переменная не объявлена в разделе описания (для C++).

ГЛАВА 5

РАЗБОР ЗАДАНИЯ 3: РЕШЕНИЕ, ТИПОВЫЕ ОШИБКИ

В главе приведен разбор задания 3 из демонстрационного варианта, а также некоторых возможных вариантов заданий по теме «Работа со строками».

В этом задании требуется написать программу, реализующую простую обработку строк.

Пример (демо)
Дано сообщение. Сформировать новое сообщение, в котором отсутствует самое короткое слово, оканчивающееся на символ «e». Сообщение состоит из слов, записанных латинскими буквами через пробел.

Python

```
s = input()
s += ' '
s_copy = s
min = len(s) + 1

# поиск min
while s != '':
  i = s.find(' ')
  if s[i - 1] == 'e':
    if i < min:
      min = i
  s = s[i + 1:]

while s_copy != '':
  i = s_copy.find(' ')
  if s_copy[i - 1]=='e' and i==min:
    s_copy = s_copy[i + 1:]
  else:
    print(s_copy[0 : i + 1], end='')
    s_copy = s_copy[i + 1:]
```

C++

```
# include <iostream>
# include <string>
using namespace std;

int main() {
    string s, s_copy; int i, min;

    getline (cin, s);
    s += " ";
    s_copy = s;
    min = s.size() + 1;

// поиск min
    while (s != ""){
        i = s.find(" ");
        if (s[i - 1] == 'e'){
            if (i < min){
                min = i;
            }
        }
        s.erase(0, i + 1);
        cout<<min;
    }

    while (s_copy != ""){
        i = s_copy.find(" ");
        if (s_copy[i - 1] == 'e' && i == min){
            s_copy.erase(0, i + 1);
        }
        else{
            cout<<s_copy.substr(0,i+1);
            s_copy.erase(0, i + 1);
        }
    }
    return 0;
}
```

Python: для ввода строки используем функцию `input()`.
C++: строки представляются как массивы элементов типа char, заканчивающиеся нуль-терминатором `\0`. Поэтому считываем строку с помощью функции `getline()`. Благодаря ей будут считываться введенные символы с пробелами до тех пор, пока во вводимом потоке не встретится код клавиши `Enter`. Если использовать операцию `cin`, то из всего введенного считается последовательность символов до первого пробела.

После каждого слова в строке есть пробел, кроме, может быть, последнего слова, поэтому добавляем в конец строки пробел: `s += ' '`.

Для реализации задачи нам понадобятся два цикла: первый – для поиска наименьшей длины слова, оканчивающегося на «е», второй – для обработки строки в соответствии с заданием.
Для поиска наименьшей длины исходную строку предполагается уменьшать, поэтому создадим копию строки `s` – переменную `s_copy`.
Начальное значение `min` на единицу больше, чем длина строки, т.е. заведомо больше, чем длина любого слова.
Python: `min = len(s) + 1`
C++: `min = s.size() + 1`

В первом цикле, пока строка не пустая, с помощью метода `find`() находим первое вхождение пробела. Сохраняем значение в переменной i. Эта же переменная будет являться длиной слова. Если символ слева от пробела «е», проверяем, будет ли его длина меньше `min`. Если длина слова меньше, обновляем значение переменной `min`.
Уменьшаем строку на длину i:
Python: используем для уменьшения строки срез `s[i + 1:]`, от `i+1` символа до конца строки.
C++: `s.erase(0, i + 1);` // метод `erase`() удаляет из строки `s` символы, начиная с символа с индексом `0` в количестве `i + 1` или до конца строки, если `0 + (i + 1) > s.size()`.

Во втором цикле для обработки строки будем использовать копию строки `s_copy`, которую создали в начале программы. Находим первое вхождение пробела и, если слово слева от пробела оканчивается на «е» и его длина равна значению переменной min, уменьшаем строку `s_copy` на длину слова, иначе выводим на экран символы слова (символы, стоящие слева от пробела), строку также уменьшаем.
Python: используем для вывода символов слова срез `s_copy[0:i + 1]`, от `0` символа до символа с индексом `i + 1`.
C++: `s_copy.substr(0, i+1); //` метод возвращает подстроку строки `s_copy` начиная с символа с индексом `0` количеством `i + 1` или до конца строки, если `0+(i + 1) > s.size()`.

Данную задачу можно реализовать, преобразовав строку s в массив слов a из n элементов. После этого преобразования задача сводится к поиску в массиве элемента, который является самым коротким словом, оканчивающимся на «е».
Например, в Python данная реализация может выглядеть так:

Python

```
s = input().split()
n = len(s) # длина списка – количество слов в сообщении

# начальное значение min на единицу больше, чем длина строки, т.е. заведомо
больше, чем длина любого слова
min = len(s) + 1
```

```
for i in range(n):
  len_element = len(s[i]) # длина слова

# если слово заканчивается на «е» и короче минимального, то необходимо запомнить
его.
  if s[i].endswith('e') and len_element < min:
    min = len_element
    short_word = s[i]

# удаление самого короткого слова, заканчивающегося на 'e', – слова short_word.
# оно может встречаться в строке не один раз, поэтому надо удалить все слова,
равные short_word. В Python можно использовать функцию del, но в примере рассмотрим
другой способ: формирование нового значения строки путем склейки слов, не равных
short_word. Вариант с использованием функции del будет описан ниже
ss = ''
for i in range(n):
  if s[i] != short_word:
    ss = ss + s[i] + ' '
print(ss)

# удаление слов предложения с использованием функции del
...
i = n - 1
while i >= 0:
  if s[i] == short_word:
    del s[i]
  i -= 1
print(' '.join(s))
```

Пример 2

В исходном тексте одно слово от другого отделено одним пробелом. Сформировать текст, в котором одно слово от другого отделяется двумя пробелами.

Python

```
s1 = input()
s2 = ''
for i in range(len(s1)):
  if s1[i] == ' ':
    s2 = s2 + s1[i]
  s2 = s2 + s1[i]
print(s2)
```

C++

```
# include <iostream>
# include <string>
using namespace std;
int main() {
    string s1, s2;
    getline (cin, s1);
    s2 = "";
    for (int i = 0; i < s1.size(); i++){
        if (s1[i] == ' '){
            s2 = s2 + s1[i];
        }
        s2 = s2 + s1[i];
     }
    cout<<s2;
    return 0;
}
```

Пример 3

Подсчитать, сколько слов в тексте начинается на букву `t`. Слова в тексте записаны строчными буквами и разделены пробелами.

Python

```
# подсчет количества вхождений
подстроки ' t' в строку s.

s = input()
print(s.count(' t'))



# вариант решения с помощью
списка:

s = input().split()
count = 0;
for i in s:
  if i[0] == 't':
    count += 1
print(count)
```

C++

```
# include <iostream>
# include <string>
using namespace std;

int main() {
    string s; int count;
    getline (cin, s);
    count = 0;
    for (int i = 0; i < s.size(); i++){
       if (s[i] == ' ' && s[i + 1] == 't'){
           count ++;
        }
    }
    cout<<count;
    return 0;
}
```

ТИПОВЫЕ ОШИБКИ

1. Строка заканчивается признаком окончания строки – числом ноль. Отдельный тестовый символ нельзя рассматривать как строку, содержащую один символ. Из-за данного различия можно столкнуться с проблемой преобразования при работе со строками (для C++).
2. Использование cin вместо getline () для считывания строки из потока ввода (для C++).
3. Ошибки в логических выражениях, например использование операции присваивания = вместо сравнения = = или использование побитовых & и | вместо логических && и || .
4. Написание машинально точки с запятой в конце условного оператора или оператора цикла задает тело оператора пустым (для C++).

ГЛАВА 6
РАЗБОР ЗАДАНИЯ 4: РЕШЕНИЕ, ТИПОВЫЕ ОШИБКИ

В главе приведен разбор задания 4 из демонстрационного варианта, а также некоторых возможных вариантов заданий по теме «Работа с одномерными массивами».

В этом задании требуется написать программу, реализующую простую обработку целочисленного массива.

Пример (демо)
Со стандартного потока ввода данных считываются значения одномерного массива размером 20 элементов. Преобразовать исходный массив, вставив после максимального элемента новый элемент, значение которого равно сумме положительных элементов, расположенных перед максимальным элементом. Исходные данные являются целыми числами в диапазоне от -10^6 до 10^6. Гарантируется, что в массиве всегда есть два и больше положительных элементов. Все элементы последовательности содержатся в одной строке через пробел. Необходимо вывести элементы преобразованного массива через пробел.

Python

```
x = list(map(int, input().split()))

# поиск максимального элемента – amax и его индекса – imax
amax = x[0]
imax = 0
for i in range(len(x)):
  if x[i] > amax:
    amax = x[i]
    imax = i

# нахождение суммы положительных элементов, расположенных перед максимальным
элементом
summa = 0
for i in range(imax):
  if x[i] > 0:
    summa += x[i]

# вставка нового элемента
for i in range(20 – 2, imax , -1):
  x[i + 1] = x[i]
x[imax + 1] = summa
for i in range(20):
  print(x[i], end=' ')
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;
```

```
int main()
{
    int x[20];
    for (int i = 0; i < 20; ++i)
    cin >> x[i];

// поиск максимального элемента – amax и его индекса – imax
    double amax = x[0];
    int imax = 0;
    for (int i = 0; i < 20; i++)
    {
        if (x[i] > amax)
            {
            amax = x[i];
            imax = i;
            }
    }

// нахождение суммы положительных элементов, расположенных перед максимальным элементом
    double s = 0.0;
    for (int i = 0; i < imax; i++)
    {
        if (x[i] > 0)
        {
        s += x[i];
        }
    }

// вставка нового элемента
    for (int i = 20 – 2; i >= imax + 1; i--){
        x[i + 1] = x[i];
    }
    x[imax + 1] = s;
    for (int i = 0; i < 20; i++){
         cout << x[i]<< ' ';
    }
    return 0;
}
```

Выше реализован универсальный вариант поиска максимального элемента и его индекса. В Python для этого можно использовать функцию `max()` и метод `index()`:

```
# поиск максимального элемента – amax и его индекса – imax
amax = max(x)
imax = x.index(amax)
```

Пример 2

Со стандартного потока ввода данных считываются значения одномерного массива размером 20 элементов. Преобразовать исходный массив, заменив нулями элементы между первым и вторым отрицательными. Исходные данные являются целыми числами в диапазоне от -10^6 до 10^6. Гарантируется, что в массиве всегда есть два и больше отрицательных элементов. Все элементы последовательности содержатся в одной строке через пробел. Необходимо вывести элементы преобразованного массива через пробел.

Python

```
#  neg1, neg2 – переменные для хранения индексов первого и второго
отрицательных элементов

a = list(map(int, input().split()))
neg1 = -1
neg2 = -1

# цикл – поиск первого и второго отрицательных чисел
# если введенный элемент отрицательный и он первый (neg1 == -1), запоминаем его
индекс neg1 = i

for i in range(0, len(a)):
      if a[i] < 0 and neg1 == -1:
             neg1= i

# если очередной введенный элемент отрицательный и он не первый (neg1 != -1),
то запоминаем индекс этого элемента в neg2 = i. Выходим из цикла

      elif a[i] < 0 and neg1 != -1:
            neg2 = i
             break

# цикл – замена элементов, находящихся между первым и вторым отрицательными
числами, нулями; все остальные элементы выводятся без изменений

for i in range(0, len(a)):
      if i <= neg1 or i >= neg2:
          print(a[i], end=" ")
      else:
          print(0, end=" ")
```

C++

```
#include <iostream>
using namespace std;

// neg1, neg2 – переменные для хранения индексов первого и второго отрицательных
элементов

int main ()
{
  int a[20];

// инициализируем позиции -1, поскольку индекс массива не может быть отрицательным.
Вы можете использовать любое другое число, которому не может равняться индекс
массива.

int neg1 = -1; int neg2 = -1;

// считываем значения массива. Это нельзя делать в следующем цикле, поскольку,
когда находим отрицательный элемент, мы выходим из цикла и необязательно доходим
до конца.

for (int i = 0; i < 20; ++i){
    cin >> a[i];
}
```

```
// цикл – поиск первого и второго отрицательных чисел
// если введенный элемент отрицательный и он первый (neg1 == -1), запоминаем его индекс neg1 = i

for (int i = 0; i < 20; ++i){
      if (a[i] < 0 && neg1 == -1){
          neg1 = i;
      }

// если очередной введенный элемент отрицательный и он не первый (neg1 != -1), то запоминаем индекс этого элемента в neg2 = i. Выходим из цикла

      else if (a[i] < 0 && neg1 != -1){
           neg2 = i;
           break;
       }
   }

// цикл – замена элементов, находящихся между первым и вторым отрицательными числами, нулями; все остальные элементы выводятся без изменений

for (int i = 0; i < 20; ++i){
      if(i <= neg1 || i >= neg2){
          cout << a[i] << " ";
      }
      else{
          cout << 0 << " ";
      }
 }
 return 0;
}
```

Пример 3

Со стандартного потока ввода данных считываются значения одномерного массива размером 20 элементов. Преобразовать исходный массив, заменяя отрицательные элементы массива средним арифметическим положительных элементов, следующих за вторым отрицательным. Исходные данные являются целыми числами в диапазоне от -10^6 до 10^6. Гарантируется, что в массиве всегда есть два и больше отрицательных элементов. Все элементы последовательности содержатся в одной строке, через пробел. Необходимо вывести элементы преобразованного массива через пробел.

Python

```
#  neg1, neg2 – переменные для хранения индексов первого и второго отрицательных элементов

a = list(map(int, input().split()))
neg1 = -1
neg2 = -1

# цикл – поиск первого и второго отрицательных чисел
# если введенный элемент отрицательный и он первый (neg1 == -1), запоминаем его индекс neg1 = i
```

```
for i in range(0, len(a)):
      if a[i] < 0 and neg1 == -1:
             neg1= i

# если очередной введенный элемент отрицательный и он не первый (neg1 != -1), то
запоминаем индекс этого элемента в neg2 = i. Выходим из цикла

      elif a[i] < 0 and neg1 != -1:
             neg2 = i
             break

# складываем все положительные элементы, следующие за вторым отрицательным
sum = 0
count = 0
for i in range(neg2 + 1, len(a)):
    if a[i] > 0:
        sum += a[i]
        count += 1

# цикл – замена отрицательных элементов на среднее арифметическое
for i in range(0, len(a)):
    if a[i] < 0:
        print(sum / count, end=" ")
    else:
        print(a[i], end=" ")
```

C++

```
#include <iostream>
using namespace std;

// neg1, neg2 – переменные для хранения индексов первого и второго отрицательных
элементов

int main ()
{
  int a[20];
  int neg1 = -1; int neg2 = -1;
  double sum = 0;
  for (int i = 0; i < 20; ++i){
     cin >> a[i];
  }
// цикл – поиск первого и второго отрицательных чисел
// если введенный элемент отрицательный и он первый (neg1 == 0), запоминаем его
индекс neg1 = i

  for (int i = 0; i < 20; ++i){
      if (a[i] < 0 && neg1 == -1){
           neg1 = i;
  }

// если очередной введенный элемент отрицательный и он не первый (neg1 != 0), то
запоминаем индекс этого элемента в neg2 = i. Выходим из цикла
```

```
      else if (a[i] < 0 && neg1 != -1){
           neg2 = i;
           break;
       }
   }
// складываем все положительные элементы, следующие за вторым отрицательным
  int count = 0;
  for (int i =  neg2 + 1; i < 20; ++i){
      if (a[i] > 0){
        sum += a[i];
        count += 1;
      }
  }
// цикл – замена отрицательных элементов на среднее арифметическое

  for (int i = 0; i < 20; ++i){
      if (a[i] < 0){
          cout << sum / count << " ";
      }
      else{
          cout << a[i] << " ";
      }
  }
return 0;
}
```

ТИПОВЫЕ ОШИБКИ

1. В цикле происходит выход за границу массива.
2. Не инициализируется или неверно инициализируется искомое значение.
3. Исходный массив не изменяется.
4. Изменяются не все требуемые элементы (например, только первый или последний из них).
5. Отсутствует вывод ответа, или ответ выводится не полностью (например, только один элемент массива ввиду пропущенного цикла вывода элементов или операторных скобок).
6. Используется переменная, не объявленная в разделе описания переменных.
7. Не указано или неверно указано условие завершения цикла.

ГЛАВА 7
РАЗБОР ЗАДАНИЯ 5: РЕШЕНИЕ, ТИПОВЫЕ ОШИБКИ

В главе приведен разбор задания 5 из демонстрационного варианта, а также некоторых возможных вариантов заданий по теме «Работа с двумерными массивами».

В этом задании требуется написать программу, реализующую простую обработку целочисленного двумерного массива.

Пример (демо)
Дана матрица A размером 6 x 6, заполненная целыми числами, не превосходящими по модулю 10^4. Требуется поменять местами максимальный элемент нижнего треугольника (включая главную диагональ) матрицы A с максимальным элементом верхнего треугольника.

Python

```
# вводятся 6 строк, каждая из которых состоит из 6 чисел, разделенных пробелами.
n = 6
x = []
for i in range(n):
    row = input().split()
    for i in range(len(row)):
        row[i] = int(row[i])
    x.append(row)

# поиск максимального элемента нижнего треугольника матрицы (включая главную диагональ), в id, jd – запоминаем номер строки и столбца, где находится максимальный элемент
xd = x[0][0]
id = 0
jd = 0
for i in range(n):
  for j in range(i + 1):
    if x[i][j] > xd:
      xd = x[i][j]
      id = i
      jd = j

# поиск максимального элемента верхнего треугольника матрицы, в iu, ju – запоминаем номер строки и столбца, где находится максимальный элемент

xu = x[0][0]
iu = 0
ju = 0
for i in range(n - 1):
  for j in range(i + 1, n):
      if x[i][j] > xu:
          xu = x[i][j]
          iu = i
          ju = j
```

```
# меняем значения максимальных элементов нижнего и верхнего треугольников матрицы
x[id][jd],  x[iu][ju] =  x[iu][ju],  x[id][jd]

# выводим на экран измененную матрицу
for i in range(n):
    for j in range(n):
      print(x[i][j], end=' ')
    print()
```

C++

```
# include <iostream>
# include <string>

using namespace std;

int main() {
#  вводятся 6 строк, каждая из которых состоит из 6 чисел, разделенных пробелами.
    const int n = 6;
    int x [n][n];
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            cin >> x[i][j];
        }
    }
# поиск максимального элемента нижнего треугольника матрицы (включая главную диагональ), в id, jd – запоминаем номер строки и столбца, где находится максимальный элемент

    int xd = x[0][0], id = 0, jd = 0;
    for (int i = 0; i < n; i++){
        for (int j = 0; j <= i; j++){
            if (x[i][j] > xd){
                xd = x[i][j];
                id = i;
                jd = j;
            }
        }
    }
# поиск максимального элемента верхнего треугольника матрицы, в iu, ju – запоминаем номер строки и столбца, где находится максимальный элемент

    int xu = x[0][0], iu = 0, ju = 0;
    for (int i = 0; i < n – 1; i++){
        for (int j = i + 1; j < n; j++){
            if (x[i][j] > xu){
                xu = x[i][j];
                iu = i;
                ju = j;
            }
        }
    }
```

```
# меняем значения максимальных элементов нижнего и верхнего треугольников матрицы

    int t = x[id][jd];
    x[id][jd] = x[iu][ju];
    x[iu][ju] = t;

# вывод обработанного двумерного массива на экран
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
             cout << x[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Python: возможны и другие способы заполнения матрицы числами из потока ввода:

```
# с помощью функции map()
      x = []
      for i in range(n):
          x.append(list(map(int, input().split())))

# с помощью списочного выражения(генератора):

      x = [list(map(int, input().split())) for i in range(n)]
```

Пример 2

Дан двумерный массив размера N x N, заполненный целыми числами, не превосходящими по модулю 10^6. Требуется выполнить преобразование: последний отрицательный элемент каждого столбца заменить нулем.

Python

```
n = int(input()) # размерность матрицы

# ввод данных в массив
x = []
for i in range(n):
    row = input().split()
    for i in range(n):
        row[i] = int(row[i])
    x.append(row)

# проходим элементы каждого столбца двумерного массива снизу вверх
# обход по столбцу завершается, как только найден отрицательный элемент
# т.к. обход организован снизу вверх, найденный отрицательный элемент(в том #
числе, если он единственный) будет последним отрицательным элементом в  # столбце
for j in range(0, n):
  i = n - 1
  while i >= 0 and x[i][j] >= 0:
    i -= 1
```

```
# условие для предотвращения обнуления элементов 0-й строки после завершения цикла
  if i >= 0:
    x[i][j] = 0

# вывод обновленного двумерного массива на экран
for i in range(n):
    for j in range(n):
      print(x[i][j], end=' ')
    print()
```

C++

```
# include <iostream>


using namespace std;

int main() {
    int n, i;
    cin >> n;
// ввод данных в массив
    int x[n][n];
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            cin >> x[i][j];
        }
    }

// проходим элементы каждого столбца двумерного массива снизу вверх
// обход по столбцу завершается, как только найден отрицательный элемент
// т.к. обход организован снизу вверх, найденный отрицательный элемент(в  // том числе, если он единственный) будет последним отрицательным элементом // в столбце
    for (int j=0; j < n; j++){
        i = n - 1;
        while (i >= 0 && x[i][j] >= 0){
            i -= 1;
        }

// условие для предотвращения обнуления элементов 0-й строки после завершения цикла
        if (i >= 0){
            x[i][j] = 0;
        }
    }

// вывод обновленного двумерного массива на экран
    for (int i = 0; i < n ; i++){
        for (int j = 0; j < n; j++){
            cout << x[i][j] << " ";
        }
        cout << endl;
    }
   return 0;
}
```

Пример 3

Дан двумерный массив размера N x N, заполненный целыми числами, не превосходящими по модулю 10^6. Требуется найти сумму элементов верхней и нижней четвертей. Элементы диагоналей (главной и побочной) не учитываются в сумме.

Python

```
n = int(input()) # размерность матрицы
s = 0
# ввод данных в массив
x = []
for i in range(n):
    row = input().split()
    for i in range(n):
        row[i] = int(row[i])
    x.append(row)

# для верхней четверти справедливо соотношение: i<j и i<N-1-j
i = 0
j = n - 1
while i < j:
  while i < j:
    # проверка, что i и j не находятся на главной и побочной диагоналях
    if i != j and j != n - i - 1:
        s += x[i][j]
    j -= 1
  i += 1
  j = n - i - 1

# для нижней четверти справедливо соотношение: i>j и i>N-1-j
i = n - 1
j = 0
while i > j:
  while i >j:
    # проверка, что i и j не находятся на главной и побочной диагоналях
    if i != j and j != n - i - 1:
        s += x[i][j]
    j += 1
  i -= 1
  j = n - i - 1

# вывод на экран суммы элементов верхней и нижней четвертей (без учета диагоналей)
print(s)
```

C++

```
# include <iostream>

using namespace std;

int main() {
    int n, i, j, s = 0;
    cin >> n;    // размерность матрицы
    int x[n][n];
```

```
// ввод данных в массив
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            cin >> x[i][j];
        }
    }

// для верхней четверти справедливо соотношение: i<j и i<N-1-j
    i = 0;
    j = n – 1;
    while (i < j){
        while (i < j){
        // проверка, что i и j не находятся на главной и побочной диагоналях
            if (i != j && j != n – i – 1){
                s += x[i][j];
            }
            j -= 1;
        }
        i += 1;
        j = n – i – 1;
    }

// для нижней четверти справедливо соотношение: i>j и i>N-1-j
    i = n – 1;
    j = 0;
    while (i > j){
        while (i >j){
        // проверка, что i и j не находятся на главной и побочной диагоналях
            if (i != j and j != n – i – 1){
                s += x[i][j];
            }
            j += 1;
        }
        i -= 1;
        j = n – i – 1;
    }
// вывод на экран суммы элементов верхней и нижней четвертей (без учета
диагоналей)
    cout << s;
    return 0;
}
```

ТИПОВЫЕ ОШИБКИ

1. В цикле происходит выход за границу массива.
2. Не инициализируется или неверно инициализируется искомое значение.
3. Исходный массив не изменяется.
4. Изменяются не все требуемые элементы.
5. Отсутствует вывод ответа, или ответ выводится не полностью, или ответ не выводится в виде матрицы.
6. Используется переменная, не объявленная в разделе описания переменных.
7. Не указано или неверно указано условие завершения цикла.

ГЛАВА 8
КРАТКИЙ СПРАВОЧНИК ПО ЯЗЫКУ PYTHON

Установка Python. Среды разработки

Для начала работы потребуется установить интерпретатор языка Python и интегрированную среду разработки для удобства написания программ.

Установка интерпретатора

Для операционных систем Windows, Mac OS дистрибутивы для установки можно найти на сайте https://www.python.org/downloads/.

Для Android подойдет пакет QPython3.

Установка IDE

Python относится к языкам, в которых программный код можно писать в простейшем текстовом редакторе на Windows. Но среда разработки IDE (*Integrated development environment*) предоставляет программисту дополнительные возможности – инспекцию кода, подсветку ошибок, подбор отступа, инструменты отладки и др.

Для начала обучения и понимания основ языка Python подойдут следующие среды:

Python IDLE – редактор, поставляемый вместе с Python;

Wing IDE 101 – https://wingware.com/downloads/wing-101;

PyCharm – https://www.jetbrains.com/ru-ru/pycharm/.

1. Переменные. Типы переменных. Идентификаторы. Литералы

Python относится к языкам с неявной динамической типизацией, т.е. объявление переменной происходит автоматически во время присвоения значения. Тип определяет способ представления значения переменной в памяти, множество допустимых значений, а также набор операций, которые можно выполнять с этими данными.

В Python выделяют несколько встроенных типов данных: целые числа, числа с плавающей точкой (вещественные), логический, строки, последовательности, множества, словари.

Для создания переменной достаточно присвоить некоторому идентификатору значение при помощи оператора присваивания «=».

```
a = 10          # переменная a  хранит значение типа int (целое число)
b = 3.1415926   # переменная b  – типа float  (вещественное число)
c = 'Hello'     # переменная c  – типа str  (строка)
d = [1, 2, 3]   # переменная d  – типа list (список из трех целых чисел)
f = True        # переменная f  – типа bool  (может принимать логические
                                  значения True и False)
```

Пример работы с типом `bool`:

```
ch = 13 > 5
print(ch)      # выведет True
```

Слева от знака присваивания указывается идентификатор, справа литерал (например, число или строка).
Идентификатор – это «ссылка» на хранимые в памяти данные. Записывается как последовательность заглавных латинских букв, строчных латинских букв, цифр и символа подчеркивания. Не может начинаться с цифры и должен быть отличным от ключевых (зарезервированных) слов. Идентификаторы чувствительны к регистру: `count` и `Count` – разные идентификаторы.

Следующие ключевые слова являются зарезервированными:

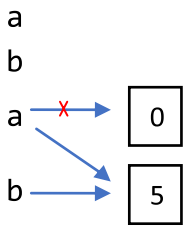
False	assert	del	for	in	or	while
None	break	elif	from	is	pass	with
True	class	else	global	lambda	raise	yield
and	continue	except	if	nonlocal	return	
as	def	finally	import	not	try	

2. Инициализация переменных. Присваивание. Обмен значениями

Инициализация переменной – первое присваивание переменной значения в процессе работы программы. Переменная должна быть проинициализирована перед использованием в выражении.

Во время присваивания в памяти сохраняется объект (число, строка или массив данных), а идентификатор (имя) переменной становится ссылкой на этот объект.

```
a = 0
b = 5
a = b
```



Оператор присваивания имеет несколько модификаций.

1. Левая часть может содержать несколько имен переменных, разделенных запятой, в правой части должно находиться столько же значений (или выражений), разделенных запятой.

   Пример:
   a, b = 4, 'hello'

2. Для инициализации одной переменной может быть также использована другая, инициализированная ранее переменная.

   Пример:
   ```
   a = 13
   b = a
   c = b
   ```

В данном примере переменным `a`, `b`, `c` присваивается одно и то же значение в следующей последовательности:

- переменной `a` присваивается значение `13`,
- переменной `b` присваивается значение `a`,
- переменной `c` присваивается значение `b`.

3. Допустимо использование множественного присваивания `a = b = c = 13`.

4. Для обмена переменных значениями в Python есть особый способ:

   `a, b = b, a`.

   Переменным, расположенным слева от знака «=», присваиваются значения переменных справа в указанном порядке. Обмен значениями можно производить с переменными разных типов.

3. Ввод данных

Для считывания данных со стандартного потока ввода используется функция `input()`. Данные считываются в виде строки и возвращаются в указанную переменную.

Пример:

```
a = input()
b = input()
```

Для преобразования строки в целочисленное или вещественное значение используются функции `int` и `float` соответственно. Считывание данных и преобразование типов можно объединить.

Пример:

```
a = int(input())
b = float(input())
```

При считывании данных нескольких переменных (или данных списка) в одной строке, разделенной пробелами, используется метод `split()`. Это специальная функция, доступная для строк, списков, множеств. Метод возвращает список строк, разрезая исходную строку на части по пробелам.

Пример:

```
a, b = input().split()
a = int(a)
b = int(b)
```

Для преобразования разделенных строк в числовой тип (например, в `int`) используется функция `map`, которая применяет для каждого элемента заданную функцию.

```
a, b, c = map(int, input().split())
```

4. Вывод данных

Для вывода данных используется функция `print()`, позволяющая выводить значения переменных или выражений, разделяя их значением параметра `sep=''` и завершая вывод значением параметра `end=''`. По умолчанию `sep` равен пустой строке, а `end` равен \n (символ переноса на следующую строку). То есть при отсутствии именованных параметров в функции выводимые значения разделяются одним пробелом, а каждый вывод организован с новой строки.

Пример:

```
a = 13
b = 5
c = 0
print(a, '+', b, '=', a + b)    # будет выведено a + b = 18
print(a, b, a + b, sep=';')     # будет выведено 13; 5; 18;

print(a, b, sep='', end='')     # значения переменных a,b,c будут
print(c)                        выведены без пробелов в одну
                                строку 1350
```

5. Встроенные операторы. Приоритет и ассоциативность операторов

Для построения выражений используют операторы. Различные операторы имею разный приоритет. Например, приоритет умножения и деления одинаковый, а приоритет сложения и вычитания ниже, чем у умножения и деления.

Пример. Рассмотрим строку кода `a = 2 + 2 * 2`.

Здесь выражением является часть кода, расположенная после простого оператора присваивания. Порядок выполнения операций:

- сначала будет выполнена операция умножения `2 * 2`, в результате наше выражение будет эквивалентно `2 + 4`;
- далее будет выполнено сложение `2 + 4`, получим `6`;
- полученное в результате значение будет присвоено переменной `a`.

В таблице ниже приведен не полный список операторов. Полный список всех операторов можно посмотреть в документации: https://docs.python.org/3/reference/expressions.html#operator-precedence.

Оператор	Описание
`:=`	Присваивание
`or`	Логическое «ИЛИ»
`and`	Логическое «И»
`not x`	Логическое «НЕ»
`in, not in`	Проверка принадлежности
`is, is not`	Проверка тождественности
`<, <=, >, >=, !=, ==`	Сравнения
`+, -`	Сложение и вычитание
`*, /, //, %`	Умножение, деление, целочисленное деление и остаток от деления
`+x, -x`	Положительное, отрицательное
`**`	Возведение в степень
`x[индекс]`	Обращение по индексу
`x[индекс1:индекс2]`	Вырезка

В таблице некоторые группы включают по нескольку операторов. Это значит, что представители одной группы имеют один уровень приоритетности. При наличии двух или более операторов с одинаковым уровнем порядок определяет ассоциативность.

Большая часть из них (за исключением оператора возведения в степень **) поддерживает ассоциативность слева направо.

Пример. Проверка ассоциативности с оператором степени.

```
print(4 ** 3 ** 2)        # будет выведено 262144
print((4 ** 3) ** 2)      # будет выведено 4096
```

Некоторые операторы (например, присваивания и сравнения) не поддерживают ассоциативность. Для них применяются специальные правила порядка, в которых ассоциативность не принимает участия.

6. Приведение и преобразование типов

Преобразования бывают явными и неявными. Так, приведение от меньшего типа к большему, от целого к вещественному происходит безопасно и в неявном виде.

```
a = 123
b = 1.2
a = b          # неявное преобразование типов, из int к float
b = int(b)     # явное преобразование типов, из float к int
print(a, b)    # будет выведено 1.2 1
```

В результате приведения вещественного числа к целому отбрасывается дробная часть числа.

При операциях с целыми типами результат выражения не всегда будет целочисленный. Сложение, вычитание, умножение двух целых чисел вернет целое число. При делении целого числа на целое результатом будет вещественное число. Для получения целочисленного результата используют оператор // (целая часть числа), в результате дробная часть отбрасывается и получается целое число. Для получения остатка используется операция % (остаток от деления).

Пример. Деление.

```
a = 3 * 2 / 2 * 2;
print(a)            # выведет 6.0
```

Порядок выполнения операций:

- сначала будет выполнена операция умножения `3 * 2`, в результате наше выражение будет эквивалентно `6 / 2 * 2`;
- далее будет выполнено деление `6 / 2`, выражение будет эквивалентно `3 * 2`;
- далее будет выполнено умножение `3 * 2`, получим `6.0`;
- полученное в результате значение будет присвоено переменной `a`.

При операциях, где хотя бы один операнд – вещественное число, результат выражения будет вещественный.

Пример. Умножение с вещественным результатом.

```
a = 3 * 2.0 * 2;
print(a)            # выведет 12.0
```

7. Логические операторы

Оператор	Значение	Пример
and	Возвращает значение True, если оба утверждения верны	x < 5 and x < 10
or	Возвращает True, если хотя бы одно из утверждений верно	x < 5 or x < 4
not	Меняет результат, возвращает False, если результат True	not(x < 5 and x < 10)

8. Основные операторы языка (инструкции)

Приведем список операторов, которые будут рассмотрены далее.

Оператор выбора `if-else` реализует разветвление.
Оператор итераций `while` реализует цикл с предусловием.
Оператор итераций `for`, как правило, используется для реализации циклов с известным количеством итераций.
Операторы перехода `break` и `continue`.
Рассмотрим более подробно каждый из перечисленных операторов.

8.1. Оператор выбора if-else

Оператор `if-else` управляет условным ветвлением. Для записи используются ключевые слова if и else, двоеточие и отступ в четыре пробела.

Синтаксис:

```
if   условие:
     операторы1
else:                #не обязательно
     операторы2
```

Операторы `операторы1` в блоке `if` выполняются только в том случае, если параметр `условие` имеет ненулевое значение или `True`, блок `else`, если он есть, в данном случае пропускается, т.е. операторы `операторы2` не будут выполнены.

Если значение `условие` равно нулю или `false`, то операторы `операторы1` пропускаются и выполняются операторы `операторы2` блока `else`, если он есть.

Рассмотрим несколько примеров.

Пример 1. Максимум из двух чисел.
Вводятся два целых числа. Необходимо вывести наибольшее из данных чисел.

```
a = int(input())
b = int(input())
if a > b:
    print(a)
else:
    print(b)
```

Для сокращения записи можно использовать каскадные конструкции `elif` и объединять операторы сравнения в цепочки.

Пример 2. Максимум из трех чисел.
Вводятся три целых числа. Необходимо вывести наибольшее из данных чисел.

```
a, b, c = map(int, input().split())
if b < a > c:
    print(a)
elif a < b > c:
    print(b)
else:
    print(c)
```

Другая реализация данного примера. Фактически мы упорядочиваем три числа по невозрастанию.

```
a, b, c = map(int, input().split())
if a < b:
    a, b = b, a
if a < c:
    a, c = c, a
if b < c:
    b, c = c, b
print(a)
```

По завершении данного алгоритма минимум всегда будет в `c`, максимум в `a`.

8.2. Оператор while

Операторы цикла `while` реализуют цикл по условию с проверкой условия перед прохождением цикла. Общий вид цикла:

```
while  условие:
    операторы
```

Выполняет инструкции `операторы` циклически, пока выражение `условие` не станет равно нулю или `false`.

Пример 1. Выведем целые числа от 1 до 5 через пробел.

```
i = 1
while i < 6:
    print(i, end=' ')
    i += 1
```

Пример 2. Список квадратов.

Задано единственное натуральное число n. Необходимо вывести квадраты натуральных чисел, не превосходящие данного числа n.

```
n = int(input())
k = 1
while k * k <= n:
    print(k * k, end=' ')
    k += 1
```

Пример 3. Список степеней двойки.

По данному числу n распечатайте все целые степени двойки, не превосходящие n, в порядке возрастания.

```
n = int(input())
k = 1
while k <= n:
    print(k, end=' ')
    k *= 2
```

Пример 4. Сумма цифр.
Дано натуральное число n. Необходимо вычислить сумму цифр числа n.

```
n = int(input())
sum = 0
while n > 0:
    sum += n % 10
    n //= 10
print(sum)
```

Пример 5. Количество нулей.
Дано натуральное число n. Необходимо вычислить количество нулей среди всех цифр числа n.

```
n = int(input())
cnt = 0
while n > 0:
    if n % 10 == 0:
          cnt += 1
    n //= 10
print(cnt)
```

Пример 6. Наибольшая цифра.
Дано натуральное число n. Необходимо найти максимальную цифру числа n.

```
n = int(input())
mx = 0
while n > 0:
    if n % 10 > mx:
          mx = n % 10
    n //= 10
print(mx)
```

8.3. Оператор цикла for

Оператор цикла `for` предназначен для выполнения одного оператора или группы (блока) операторов заданное количество раз.

Общий вид оператора `for`:

```
for i in (...):
    блок кода (тело цикла)
```

Вместо i может быть использована любая другая последовательность символов, которая может служить именем переменной.

(…) обозначает некоторый способ задать диапазон значений.

Один из наиболее часто используемых вариантов – задать диапазон при помощи ключевого слова `range`. Это генератор, который создает последовательность элементов. Например, `range(n)` создаст последовательность чисел от 0 (включительно) до `n` (не включительно). Генератору можно задать стартовое (по умолчанию равно 0), конечное значения и шаг (по умолчанию равен 1). Если очередной элемент оказывается не меньше конечного числа, то генератор прекращает работу. Например, `range(2, 15, 3)` сгенерирует последовательность чисел {2, 5, 8, 11, 14}.

Еще один часто применимый способ задать диапазон – это на место (…) вставить имя списка, по элементам которого мы хотим «пройтись» в цикле. Данный способ применим, если нам не нужно менять элементы списка, а нужно только получить некоторую информацию, например посчитать сумму этих элементов.

Пример 1. Выведем целые числа от 1 до n через пробел.

```
n = int(input())
for i in range(1, n + 1):
    print(i, end=' ')
```

Оператор `for` выполняется следующим образом: range генерирует последовательность значений, в данном случае от 1 до n включительно, и i последовательно принимает каждое из этих значений. То есть на первой итерации цикла i =1, на второй i = 2 и так далее, пока мы не дойдем до конца последовательности.
На каждой итерации выполняется тело цикла. Так, на каждой итерации выводится значение переменной i.

Пример 2. На вход подается натуральное число n. Далее, каждое в отдельной строке, задаются n целых чисел, по модулю не превышающих 1000. Необходимо вывести первое четное число из этой последовательности и строку Error, если такого числа в последовательности нет.

```
n = int(input())
output = "Error"
for i in range(n):
    x = int(input())
    if x % 2 == 0:
          output = x
          break
print(output)
```

В переменной output мы храним значение, которое будет выведено. Изначально оно равно Error – строка, которую нужно вывести, если четное число в последовательности не встречается. Когда мы встречаем подходящее число, мы присваиваем это значение переменной output.

Поскольку нам нужно первое четное, то, если мы уже нашли такое, остальные значения нас не интересуют. Для того чтобы досрочно выйти из цикла, используется оператор `break`.

Пример 3. На вход подается натуральное число n. Далее, каждое в отдельной строке, задаются n натуральных чисел, не превышающих 1000. Посчитать сумму чисел последовательности, а также вычислить сумму последних цифр всех элементов последовательности, кратных 3.

```
n = int(input())
sum_of_elements = 0
sum_of_last_digits = 0
for i in range(n):
    x = int(input())
    sum_of_elements += x
    if x % 3 != 0:
         continue
    sum_of_last_digits += x % 10
print(sum_of_elements, sum_of_last_digits)
```

В данном случае удобно использовать оператор continue, который завершает текущую итерацию цикла и переходит к следующей. В данном примере можно было использовать условный оператор с противоположным условием, но в более сложных случаях эта замена может быть неравноценной. Например, если бы для чисел, кратных 3, нужно было бы выполнить еще какие-то проверки и в зависимости от результатов этих проверок выполнять разные действия. В таком случае оператор `continue` помогает нам избежать большой степени вложенности и делает код более читаемым.

Пример 4. Четные числа. Вводятся целые числа `a` и `b`. Гарантируется, что a не превосходит b. Выведите (через пробел) все четные числа от a до b (включительно).

```
a = int(input())
b = int(input())
for i in range(a, b + 1):
    if i % 2 == 0:
         print(i, end=' ')
```

Пример 5. Вычислите сумму данных n натуральных чисел. Вводится число n, а затем n чисел, сумму которых необходимо вычислить. Выведите единственное число – сумму введенных чисел.

```
n = int(input())
sum = 0
for i in range(0, n):
    x = int(input())
    sum += x
print(sum)
```

Пример 6. Напишите программу, вычисляющую 2^n. Вводится целое неотрицательное число n, которое не превосходит 30. Выведите число 2^n.

```
n = int(input())
k = 1
x = 0
for i in range(0, n):
    k *= 2
print(k)
```

9. Списки

Список представляет собой упорядоченное множество элементов, доступ к которым осуществляется с помощью обращения по индексу через оператор [ ]. Индексы – это числа от 0 до размера списка невключительно. Соответственно, первый элемент имеет индекс 0, второй – индекс 1.

Пример 1. В данном примере создадим список из пяти элементов. Выведем на экран значение начального элемента списка.

```
a = [10, 20, 30, 40, 50]
print(a[0])
```

Пример 2. Ввод-вывод элементов списка. Введем 10 чисел с консоли и выведем их в консоль.

Способ 1: создадим список из 10 элементов, каждый из которых равен 0. Это делается посредством специальной конструкции. `[x] * y` – создает список из y элементов, равных `x`.
Далее в цикле считаем значение каждого элемента, а потом выведем их на экран.

```
a = [0] * 10
for i in range(0, 10):
    a[i] = int(input())
for i in range(0, 10):
    print(a[i], end=' ')
```

Способ 2: создадим пустой список, а в цикле на каждой итерации считаем очередное значение и добавим его в список.

```
a = []
for i in range(0, 10):
    a.append(int(input()))
for i in range(0, 10):
    print(a[i], end=' ')
```

Пример 3. С консоли вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо найти и вывести максимальный элемент.

```
a = [0] * 10
for i in range(0, 10):
    a[i] = int(input())
mx = a[0]
for i in range(1, 10):
    if a[i] > mx:
          mx = a[i]
print(mx)
```

Пример 4. С консоли вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо найти и вывести индекс первого максимального элемента последовательности.

```
a = [0] * 10
for i in range(0, 10):
    a[i] = int(input())
```

```
imx = 0
for i in range(1, 10):
    if a[i] > a[imx]:
         imx = i
print(imx)
```

Пример 5. С консоли вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо найти и вывести индекс последнего максимального элемента последовательности.

```
a = [0] * 10
for i in range(0, 10):
    a[i] = int(input())
imx = 0
for i in range(1, 10):
    if a[i] >= a[imx]:
         imx = i
print(imx)
```

Пример 6. Вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо найти и вывести индекс первого отрицательного элемента массива.

```
a = [0] * 10
for i in range(0, 10):
a[i] = int(input())
neg = -1
for i in range(0, 10):
    if a[i] < 0:
         neg = i
         break
if neg == -1:
    print("no negative value")
else:
    print(neg)
```

Пример 7. Вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо найти и вывести индекс последнего отрицательного элемента массива.

```
a = [0] * 10
for i in range(0, 10):
    a[i] = int(input())
neg = -1
for i in range(0, 10):
    if a[i] < 0:
         neg = i
if neg == -1:
    print("no negative value")
else:
    print(neg)
```

Пример 8. Вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо поменять местами первый отрицательный и последний максимальный элементы массива. Гарантируется, что отрицательный элемент есть во входных данных. Вывести полученный массив в консоль.

```
a = [0] * 10
for i in range(0, 10):
    a[i] = int(input())
neg = -1
for i in range(0, 10):
    if a[i] < 0:
        neg = i
        break
imx = 0
for i in range(1, 10):
    if a[i] >= a[imx]:
        imx = i
tmp = a[neg]
a[neg] = a[imx]
a[imx] = tmp
for i in range(0, 10):
    print(a[i], end=' ')
```

Пример 9. Вводится последовательность из 10 элементов. Каждый элемент вводится в отдельной строке. Необходимо заменить первый элемент масссива (0 индекс) средним значением всех элементов массива. Вывести полученный массив в консоль.

```
a = [0] * 10
sum = 0.0
for i in range(0, 10):
    a[i] = int(input())
    sum += a[i]
a[0] = sum / 10
for i in range(0, 10):
    print('%.2f' % a[i], end=' ')
```

ГЛАВА 9
КРАТКИЙ СПРАВОЧНИК ПО ЯЗЫКУ С++

Документация по Visual C++
https://docs.microsoft.com/ru-ru/cpp/?view=vs-2019.

Справка по установке средства C++ в Visual Studio доступна по ссылке
https://docs.microsoft.com/ru-ru/cpp/build/vscpp-step-0-installation?view=vs-2019.

Создание проекта консольного приложения C++
https://docs.microsoft.com/ru-ru/cpp/get-started/tutorial-console-cpp?view=vs-2019.

Прежде чем перейти к изложению средств языка, рассмотрим простейший пример программы, написанной на языке C++.

Пример. Переменной *x* присвоить значение 10 и вывести значение *x* на экран (консоль).

```
#include <iostream>

using namespace std;

int main()
{
    int x = 10;
    cout << x; // вывод значения переменной в консоль
    system("pause");
    return 0;
}
```

В Visual Studio разработка программы осуществляется в рамках *проекта*. Для создания нового проекта необходимо войти в Visual Studio и в меню «Файл» выбрать «Создать проект». Появится диалоговое окно «Создание проекта».

Создадим консольное приложение. Для создания консольного приложения необходимо в появившемся окне «Создание проекта» выбрать шаблон «Пустой проект» и нажать «OK».

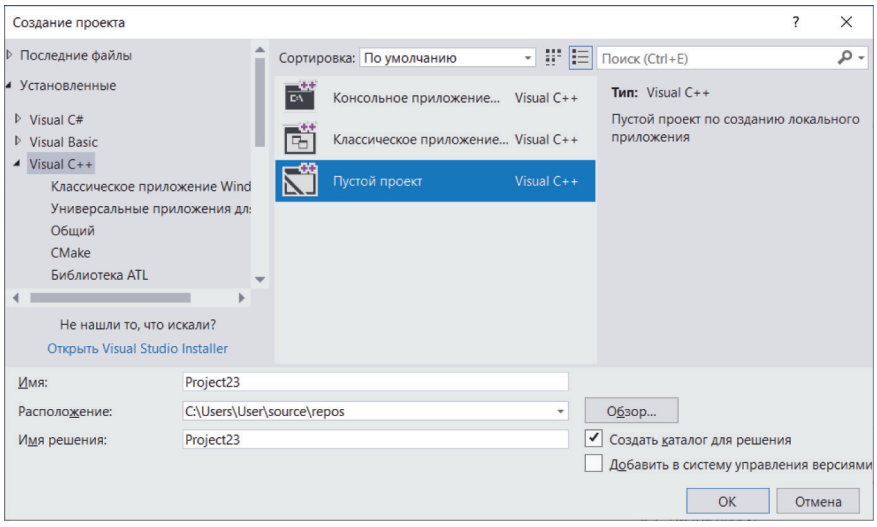


Рис. 1 Окно «Создание проекта». Microsoft Visual Studio Code 2019

Приложению автоматически присваивается имя `Project1` (при желании это имя можно заменить любым другим). Если проект с таким именем существует, то будет создан проект со следующим свободным порядковым номером – `Project2`.

Далее необходимо открыть «Обозреватель решений», нажать правой кнопкой мыши на каталог «Исходные файлы» и вызвать контекстное меню. Выбрать пункт меню «Добавить», далее «Создать элемент».

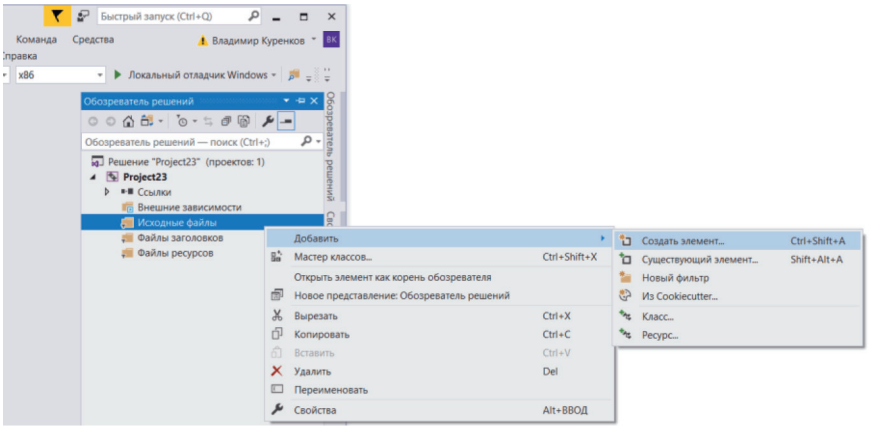


Рис. 2 Окно «Обозреватель решений». Microsoft Visual Studio Code 2019

В открывшемся окне «Добавление нового элемента» выбрать «Файл c++» и нажать кнопку «Добавить».

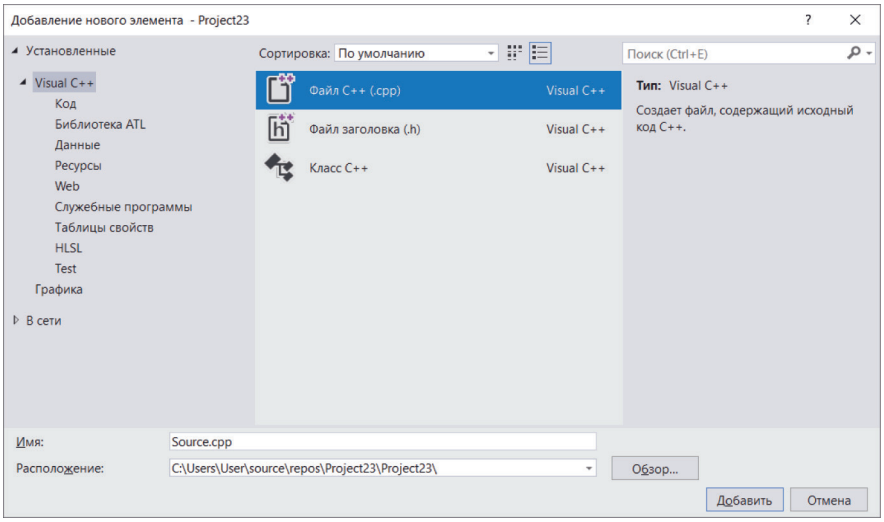


Рис. 3 Добавление нового элемента. Microsoft Visual Studio Code 2019

В открывшийся файл Source.cpp вписать код программы.

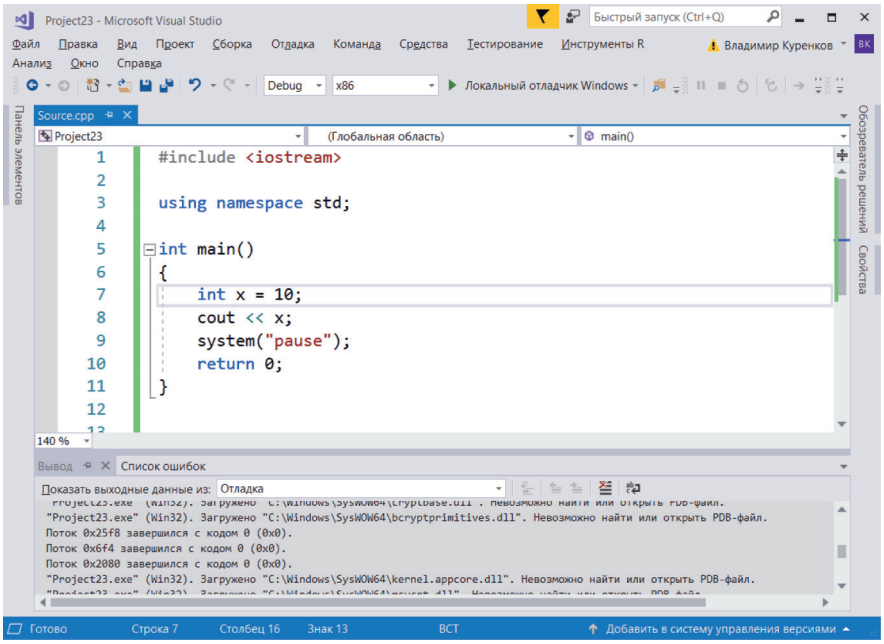
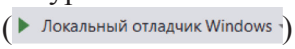


Рис. 4 Создание программы. Microsoft Visual Studio Code 2019

Для выполнения программы нажмите клавишу F5 или Ctrl/F5 на клавиатуре. Также можно выполнить программу, нажав кнопку «Локальный отладчик Windows» (

) на панеле.

Функция `main` является точкой входа каждого приложения C++ и вызывается при запуске программы. Рассмотрим, что делает каждая строчка нашей программы:

- `int x = 10;` – в данной строке мы объявляем переменную x типа `int` и инициализируем ее значением 10, т.е. присваиваем переменной x значение 10;
- `cout << x;` – выводим значение переменной x в консоль;
- `// вывод значения переменной в консоль` – комментарий к тексту программы. Комментарии не оказывают никакого влияния на результат компиляции программы;
- `system("pause");` – приостанавливает выполнение программы, что позволяет увидеть результат. В противном случае консоль сразу после того, как программа выполнится, будет закрыта;
- `return 0;` – завершает выполнение функции `main`.

Запустив программу, мы увидим следующее консольное окно:

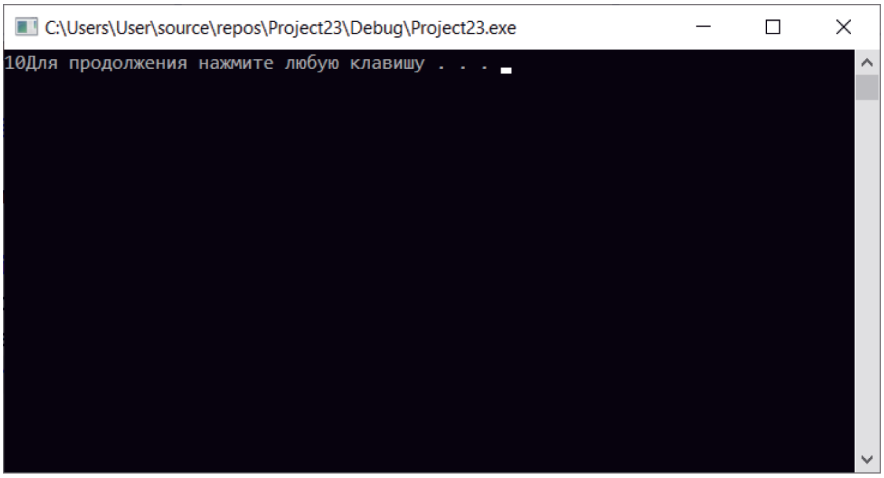


Рис. 5 Консольное окно программы Microsoft Visual Studio Code 2019

1. Переменные. Типы переменных. Идентификаторы

В C++ переменные объявляются с определенным типом. Тип определяет способ представления значения переменной в памяти, множество допустимых значений, а также набор операций, которые можно выполнять с этими данными. Встроенные (фундаментальные) типы в C++ делятся на три категории: целочисленные, с плавающей запятой и void.

Таблицу с типами и диапазонами их значений можно посмотреть в документации: https://docs.microsoft.com/ru-ru/cpp/cpp/data-type-ranges?view=vs-2019.

Основные используемые в пособии типы:

- целое 32-разрядное число
  int i = 5;
- вещественное 64-разрядное число
  double d = 6.7;
- логический тип данных, переменная этого типа может принимать значения true и false. При выводе в консоль или применении арифметических операций будет вести себя как число 1 или 0 соответственно.
  bool b = true.

Пример работы с типом `bool`:

```
#include <iostream>
using namespace std;
int main() {
    bool b = true;
    cout << b << '\n'; // выведет «1»
    cout << b * 5; // выведет «5»
    system("pause");
    return 0;
}
```

Другие типы будут рассмотрены в пособии позднее.

Для обозначения переменных используются *идентификаторы*: сочетание букв, цифр и знака подчеркивания.

Например: `int ans = 5`
`ans` – идентификатор (имя переменной).
Следующие символы можно использовать в качестве любого символа идентификатора:
a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Следующие символы могут быть любым символом в идентификаторе, кроме первого:
0 1 2 3 4 5 6 7 8 9.

Подробнее про идентификаторы можно прочитать в документации:
https://docs.microsoft.com/ru-ru/cpp/cpp/identifiers-cpp?view=vs-2019.

C++ имеет зарезервированный набор слов для собственного использования. Эти слова называются ключевыми словами. Данные слова нельзя использовать в качестве идентификатора. Например, `int` является зарезервированным словом и нельзя его использовать в качестве идентификатора (имени переменной):

```
int int = 5.
```

2. Литералы

Литерал – это представление значения в исходном коде.
Пример:

```
int i = 5;
int d = 4.5;
char ch = 'a';
string s = "Happy!";
```

В данном примере:
- i, d, ch, s – переменные;
- 5, 4.5, 'a', "Happy!" – литералы.

Подробнее про литералы можно прочитать в документации:
https://docs.microsoft.com/ru-ru/cpp/cpp/string-and-character-literals-cpp?view=vs-2019.

3. Объявления и инициализация переменных

Объявления вводят имена в программе, например имена переменных. В объявлениях также указывается тип данных. Перед использованием имени его необходимо объявить, при необходимости переменной может быть присвоено значение.

Инициализация переменной – первое присваивание переменной значения в процессе работы программы. Переменные можно инициализировать при объявлении.

В C++ инициализация переменной не является обязательным условием.

Для инициализации одной переменной может быть также использована другая, инициализированная ранее переменная:

```
int x = 1;
int y = x;
int z;
z = y;
```

В данном примере переменным x, y, z присваивается одно и то же значение в следующей последовательности:

1. Переменной x присваивается значение 1.
2. Переменной y присваивается значение x.
3. Переменной z присваивается значение y.

Допустима также запись

```
int x, y, z;
x = y = z = 1;
```

Операция присвоения правоассоциативна. Присваивание происходит справа налево, в данном примере переменным x, y, z присваивается одно и то же значение в следующей последовательности:

1. Переменной z присваивается значение 1.
2. Переменной y присваивается значение z.
3. Переменной x присваивается значение y.

Константы задают неизменяемые в программе значения. Константа обязательно должна быть проинициализирована при объявлении. Например:

```
const int prime = 1000000009;
```

4. Ввод/вывод

В C++ для ввода-вывода подключается библиотека iostream. Данная библиотека определяет три стандартных потока:

- `cin` – стандартный входной поток;
- `cout` – стандартный выходной поток;
- `cerr` – стандартный поток вывода сообщений об ошибках.

`cin` и `cout` относятся к пространству имен std. Директива `using` позволяет использовать все имена в пространстве имен без указания имени пространства имен, соответственно, для использования `cin` и `cout` необходимо:

- либо прописать строку `using namespace std`;
- либо указывать пространство имен `std::cin, std::cout`.

Пример 1. Выведем текст в консоль.

```
#include <iostream>

using namespace std;

int main() {
      cout << "Happy!";
      system("pause");
      return 0;
}
```

```
#include <iostream>

int main() {
      std::cout <<
"Happy!";
      system("pause");
      return 0;
}
```

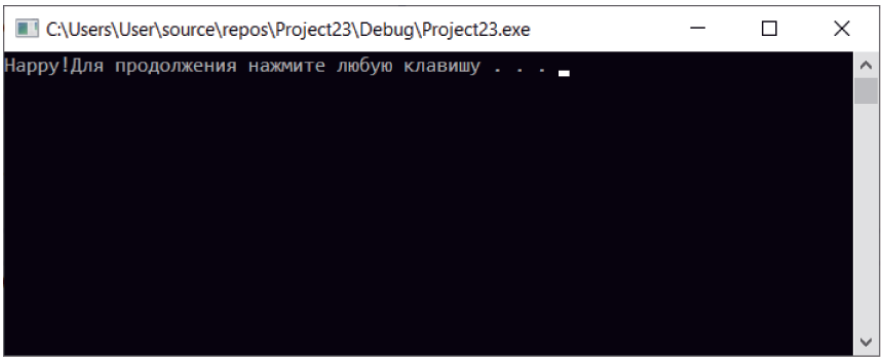


Рис. 6 Ввод данных в консоль. Microsoft Visual Studio Code 2019

Пример 2. Введем и выведем через пробел два целых числа.

1. Объявим две переменные целого типа a и b, проинициализируем их значением ноль.
2. Введем значения данных переменных через консоль.
3. Выведем значения данных переменных в консоль через пробел.

```
#include <iostream>

using namespace std;

int main() {
    int a = 0, b = 0;
    cin >> a >> b;
    cout << a << " " << b;
    system("pause");
    return 0;
}
```

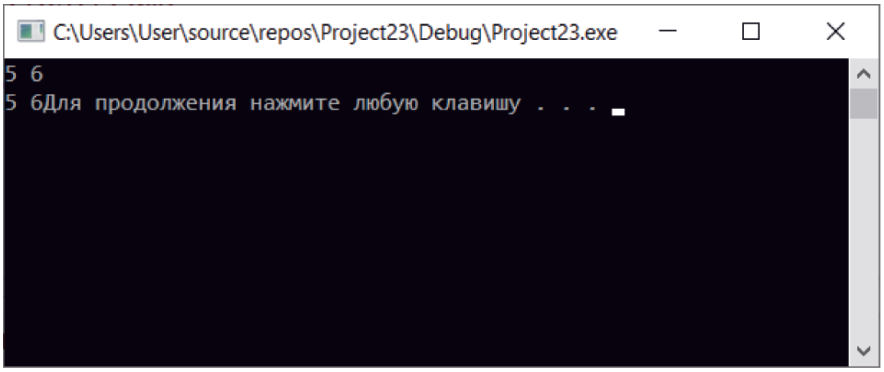


Рис. 7 Вывод результата в консоль. Microsoft Visual Studio Code 2019

Пример 3. Посчитать сумму двух вещественных чисел.

1. Объявим две переменные вещественного типа a и b, проинициализируем их значением ноль.
2. Введем значения данных переменных через консоль.
3. Выведем сумму значений данных переменных в консоль.

```
#include <iostream>

using namespace std;

int main() {
    double a = 0.0, b = 0.0;
    cin >> a >> b;
    cout << a + b;
    system("pause");
    return 0;
}
```

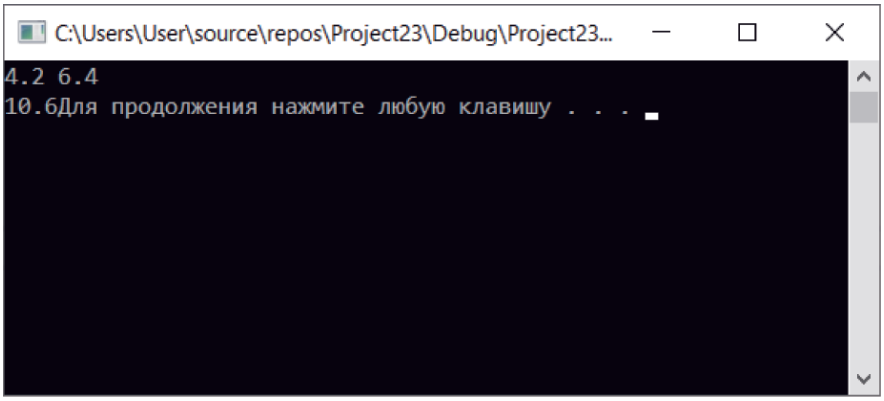


Рис. 8 Демонстрация работы программы подсчета суммы двух вещественных чисел. Microsoft Visual Studio Code 2019

5. Вычисление значения выражения из операндов. Встроенные операторы. Приоритет и ассоциативность операторов

Для построения выражений используют операторы. Различные операторы имеют разный приоритет, так, например, приоритет умножения и деления одинаковый, а приоритет сложения и вычитания ниже, чем у умножения и деления.

Пример 1. Рассмотрим строку кода `int a = 2 + 2 * 2;`

Здесь выражением является часть кода, расположенная после простого оператора присваивания. Порядок выполнения операций:

- сначала будет выполнена операция умножения 2 * 2, в результате наше выражение будет эквивалентно
  `2 + 4;`
- далее будет выполнено сложение 2 + 4. Получим
  6;
- полученное в результате значение будет присвоено переменной `a`.

В таблице ниже приведен не полный список операторов. Полный список всех операторов можно посмотреть в документации:

https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-built-in-operators-precedence-and-associativity?view=vs-2019.

Операторы, приведенные в таблице, разбиты на группы в порядке по уменьшению их приоритета. Стрелочками показана ассоциативность операторов:

- → ассоциативность слева направо
- ← ассоциативность справа налево

1	`::`	Оператор разрешения области	`std::cin`
2 →	`[]`	Индекс массива	`int arr[5] = { 0, 1, 2, 3, 4 };` `cout << arr[2]; // выведет «1»`
	`()`	Оператор вызова функции	`double d = sin(3.14);` `cout << d; // выведет «0.00159265»`

	`++`	Постфиксный инкремент	`int a = 5;`
			`a++;`
			`cout << a; // выведет «6»`

	`--`	Постфиксный декремент	`int a = 5;`
			`cout << a-- << " ";`
			`/* выведет «5», сначала значение переменной будет передано в консоль для печати, потом переменная будет уменьшена на единицу*/`
			`cout << a; // выведет «4»`

3 ←	`++`	Префиксный инкремент	`int a = 5;`
			`cout << ++a;`
			`/* выведет «6», сначала значение переменной будет увеличено на 1, потом передано в консоль для печати */`

	`--`	Префиксный декремент	`int a = 5;`
			`--a;`
			`cout << a; // выведет «4»`

	`!`	Логическое не	`bool b = true;`
			`cout << b << " "; // выведет «1»`
			`cout << !b; // выведет «0»`

	`-`	Унарное отрицание	`int a = 5;`
			`a = -a;`
			`cout << a; // выведет «-5»`

	`+`	Унарный плюс	`int a = 5;`
			`a = +a;`
			`cout << a;`
			`/* выведет «5», значение не будет изменено */`

4 →	`*`	Умножение	`int a = 5;` `cout << a * 2; // выведет «10»`
	`/`	Деление	`int a = 10;` `cout << a / 2; // выведет «5»`

	`%`	Остаток от деления	`int a = 15;`
			`cout << a % 4 << " " << a % 10;`
			`/* выведет «3 5» */`

5 →	`+`	Сложение	`int a = 5, b = 7;` `cout << a + b – 4; // выведет «8»`
	`-`	Вычитание	

6 →	<	Меньше	`int x = 5, y = 5;`
			`bool b = x < y;`
			`cout << b << " ";`
			`cout << (x >= y); // выведет «0 1»`

	>	Больше	
	<=	Меньше или равно	
	>=	Больше или равно	

7 →	==	Равенство	`int x = 5, y = 5;`
			`bool b = x == y;`
			`cout << b << " ";`
			`cout << (x != y); // выведет «1 0»`

	!=	Неравенство	
8 →	&&	Логическое И	`int x = 5;` `cout << (x > 3 && x < 9); //выведет «1»`
9 →	||	Логическое ИЛИ	`int x = 6;` `cout << (x < 5 || x > 9); //выведет «0»`

9 ←	?:	Условный	`int x = 6;`
			`cout << (x < 7 ? 2 : 3) * 10;`
			`//выведет «20»`

10 ←	=	Присваивание	`int x = 5;`
			`x += 5;`
			`cout << x << " ";`
			`x *= 2;`
			`cout << x;`
			`//выведет «10 20»`

	*=	Умножение с присваиванием	
	/=	Деление с присваиванием	
	%=	Присваивание остатка	
	+=	Сложение с присваиванием	
	-=	Вычитание с присваиванием	

Приведенные в таблице примеры будут работать, если их добавить в следующий шаблон:

```
#include <iostream>

using namespace std;

int main() {
    // в данном месте добавить пример кода из таблицы.
    system("pause");
    return 0;
}
```

6. Приведение и преобразование типов

Преобразования бывают явными и неявными.

```
int a = 123;
double b = a;   // implicit conversion from int to double
                // неявное преобразование типов, из int к double
int c = (int)b; // explicit conversion from double to int
                // явное преобразование типов, из double к int
```

Явное и неявное преобразование типов происходит в нескольких случаях. Например, при присвоении переменной значения.

```
int a = 8.2; // неявное приведение
```

Здесь целочисленной переменной присвоен литерал вещественного числа. Это не приведет к ошибке компиляции, но компилятор выдаст предупреждение. В данном случае рекомендуется выполнить явное преобразование типов.

```
double d = 8.2;
int a = int(d); // явное приведение
cout << a;      // выведет 8
```

В результате приведения вещественного числа к целому отбрасывается дробная часть числа.

Приведение от меньшего типа к большему или от целого к вещественному происходит безопасно в неявном виде.

```
int i = 8;
double d = i; // неявное приведение
```

При операциях с целыми типами результат выражения будет целочисленный. Сложение, вычитание, умножение и деление двух целых чисел вернет целое число. При делении целого числа на целое дробная часть отбрасывается и получается целочисленное деление.

Пример 1. Целочисленное деление.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    int a = 3 * 3 / 2 * 2;
    cout << a; // выведет 8
    system("pause");
    return 0;
}
```

Порядок выполнения операций:

- сначала будет выполнена операция умножения `3 * 3`, в результате наше выражение будет эквивалентно
  `9 / 2 * 2`;
- далее будет выполнено целочисленное деление `9 / 2`, дробная часть при делении будет отброшена, в результате наше выражение будет эквивалентно
  `4 * 2`;
- далее будет выполнено умножение `4 * 2`, получим
  `8`;
- полученное в результате значение будет присвоено переменной `a`.

При операциях, где хотя бы один операнд – вещественное число, результат выражения будет вещественный.

Пример 2. Деление с вещественным результатом.

```
#include <iostream>
#include <iomanip>

using namespace std;
```

```
int main() {
    double a = 3.0 / 2 * 2;
    cout << a; // выведет 3
    system("pause");
    return 0;
}
```

Порядок выполнения операций:

- сначала будет выполнена операция деления `3.0 / 2`, в результате наше выражение будет эквивалентно
  `1.5 * 2`;
- далее будет выполнено умножение `1.5 * 2`, получим
  `3.0`;
- полученное в результате значение будет присвоено переменной `a`.

Полную информацию о преобразовании типов можно прочитать в документации: https://docs.microsoft.com/ru-ru/cpp/cpp/standard-conversions?view=vs-2019.

В языке C++ определены преобразования между его основными типами. Эти преобразования называются стандартными преобразованиями.

7. Логические операторы.

Унарный ! (логическое отрицание) оператор.

Бинарные & (логическое И), | (логическое ИЛИ), а также ^ (логическое исключающее ИЛИ) операторы. Эти операторы всегда обрабатывают оба операнда.

Бинарные && (условное логическое И) и || (условное логическое ИЛИ) операторы. Эти операторы вычисляют правый операнд, только если это необходимо.

8. Основные операторы языка (инструкции)

Приведем список операторов, которые будут рассмотрены далее:

оператор выбора `if-else` реализует разветвление;
оператор итераций `while` реализует цикл с предусловием;
оператор итераций `do-while` реализует цикл с постусловием;
оператор итераций `for`, как правило, используется для реализации циклов с известным количеством итераций;
операторы перехода `break` и `continue`.

Рассмотрим более подробно каждый из перечисленных операторов.

8.1. Оператор выбора if-else

Оператор `if-else` управляет условным ветвлением.

Синтаксис:

```
if ( условие )
{
   Операторы1;
   ...
}
else  // не обязательно
{
   Операторы2;
   ...
}
```

Операторы `Операторы1` в блоке *`if`* выполняются только в том случае, если параметр `условие` имеет ненулевое значение или *`true`*, блок *`else`*, если он есть, в данном случае пропускается, т.е. операторы `Операторы2` не будут выполнены.

Если значение `условие` равно нулю или *`false`*, то операторы `Операторы1` пропускаются и выполняются операторы `Операторы2` блока *`else`,* если он есть.

Подробнее можно прочитать в документации
https://docs.microsoft.com/ru-ru/cpp/cpp/if-else-statement-cpp?view=vs-2017.

Рассмотрим несколько примеров:

Пример 1. Максимум из двух чисел.
Вводятся два целых числа. Необходимо вывести наибольшее из данных чисел.

```
#include <iostream>

using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    if (a > b) {
         cout << a;
    } else {
         cout << b;
    }
    system("pause");
    return 0;
}
```

Также можно реализовать данную задачу при помощи тернарного оператора.

```
#include <iostream>

using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    cout << (a > b ? a : b);
    system("pause");
    return 0;
}
```

Пример 2. Какое из чисел больше.
Вводятся два целых числа. Необходимо вывести число 1, если первое число больше второго, число -1, если второе больше первого, число 0, если они равны.

```
#include <iostream>

using namespace std;

int main()
{
```

```
    int a, b;
    cin >> a >> b;
    if (a > b) {
        cout << 1;
    } else if (a < b) {
        cout << -1;
    } else {
        cout << 0;
    }
    system("pause");
    return 0;
}
```

Пример 3. Максимум из трех чисел.
Вводятся три целых числа. Необходимо вывести наибольшее из данных чисел.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    if (a >= b && a >= c) {
        cout << a;
    } else if (b >= a && b >= c) {
        cout << b;
    } else {
        cout << c;
    }
    system("pause");
    return 0;
}
```

Другая реализация данного примера. Фактически мы упорядочиваем три числа по невозрастанию. Функция `swap` меняет местами значения двух чисел, т.е. если `x = 5`, а `y = 7`, то после `swap(x, y)`, `x = 7`, а `y = 5`.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    if (a < b) swap(a, b);
    if (a < c) swap(a, c);
    if (b < c) swap(b, c);
    cout << a;
    system("pause");
    return 0;
}
```

По завершении данного алгоритма минимум всегда будет в `c`, максимум в `a`.

8.2. Оператор while

Операторы цикла `while` реализуют цикл по условию с проверкой условия перед прохождением цикла (цикл с предусловием). Общий вид цикла:

```
while ( условие ) {
   Операторы
}
```

Выполняет инструкции `Операторы` циклически, пока выражение `условие` не станет равно нулю или *`false`*.

Пример 1. Выведем целые числа от 1 до 5 через пробел.

```
#include <iostream>

using namespace std;

int main()
{
    int i = 1;
    while (i < 6)
    {
         cout << i << " ";
         i++;
    }
    system("pause");
    return 0;
}
```

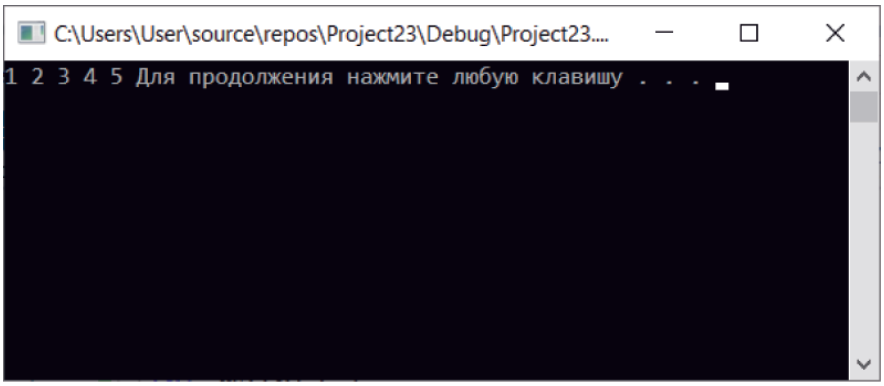


Рис. 9 Последовательный вывод в строку чисел от 1 до 5. Microsoft Visual Studio Code 2019

Пример 2. Список квадратов.

Задано единственное натуральное число n. Необходимо вывести квадраты натуральных чисел, не превосходящие данного числа n.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0, k = 1;
    cin >> n;
```

```
    while (k * k <= n) {
        cout << k * k << " ";
        k++;
    }
    system("pause");
    return 0;
}
```

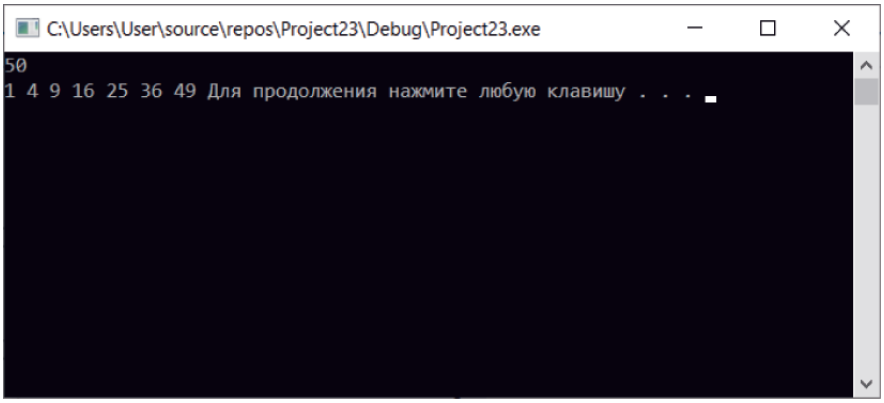


Рис. 10 Последовательный вывод квадратов чисел не превосходящих n. Microsoft Visual Studio Code 2019

Пример 3. Список степеней двойки.

По данному числу n распечатайте все целые степени двойки, не превосходящие n, в порядке возрастания.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0, k = 1;
    cin >> n;
    while (k <= n) {
        cout << k << " ";
        k *= 2;
    }
    system("pause");
    return 0;
}
```

Пример 4. Сумма цифр.

Дано натуральное число n. Необходимо вычислить сумму цифр числа n.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0, sum = 0;
    cin >> n;
```

```
    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }
    cout << sum;
    system("pause");
    return 0;
}
```

Пример 5. Количество нулей.

Дано натуральное число n. Необходимо вычислить количество нулей среди всех цифр числа n.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0, cnt = 0;
    cin >> n;
    while (n > 0) {
        if (n % 10 == 0)
            cnt++;
        n /= 10;
    }
    cout << cnt;
    system("pause");
    return 0;
}
```

Пример 6. Наибольшая цифра.

Дано натуральное число n. Необходимо найти максимальную цифру числа n.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0,  mx = 0;
    cin >> n;
    while (n > 0) {
        if (n % 10 > mx)
            mx = n % 10;
        n /= 10;
    }
    cout << mx;
    system("pause");
    return 0;
}
```

8.3. Операторы цикла do-while

Операторы цикла `do-while` реализуют цикл по условию с проверкой условия после первого прохождения цикла (цикл с постусловием). Общий вид цикла:

```
do
{
   Операторы
}
while (условие);
```

Выполняет инструкции `Операторы` циклически, пока выражение `условие` не станет равно нулю или *`false`*.

Пример 1. Выведем целые числа от 1 до 5 через пробел.

```
#include <iostream>

using namespace std;

int main()
{
    int i = 1;
    do
    {
         cout << i << " ";
         i++;
    } while (i < 6);
    return 0;
}
```

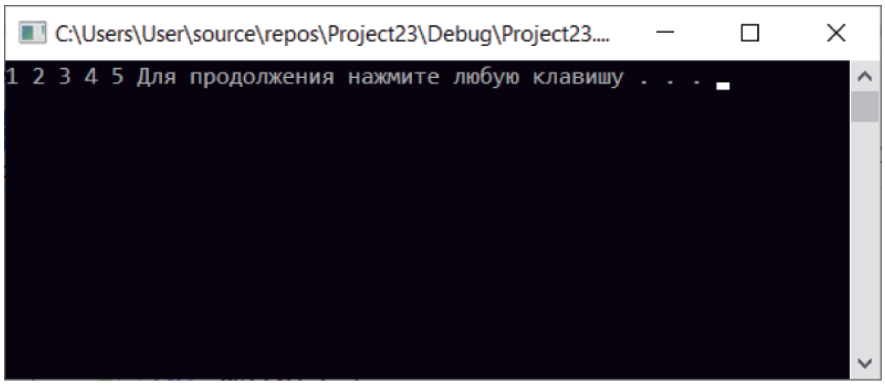


Рис. 11 Последовательный вывод через пробел целых чисел от 1 до 5. Microsoft Visual Studio Code 2019

8.4. Оператор цикла for

Оператор цикла `for` предназначен для выполнения одного оператора или группы (блока) операторов заданное количество раз.

Общий вид оператора `for`:

```
for ([ инициализация]; [условие];[приращение])
{
     Операторы;
}
```

Если в цикле выполняется один оператор, то фигурные скобки необязательны. Однако в этом случае их рекомендуется использовать для наглядности.

Пример 1. Выведем целые числа от 1 до 5 через пробел.

```
#include <iostream>

using namespace std;

int main()
{
    for (int i = 1; i <= 5; i++)
    {
         cout << i << " ";
    }
    system("pause");
    return 0;
}
```

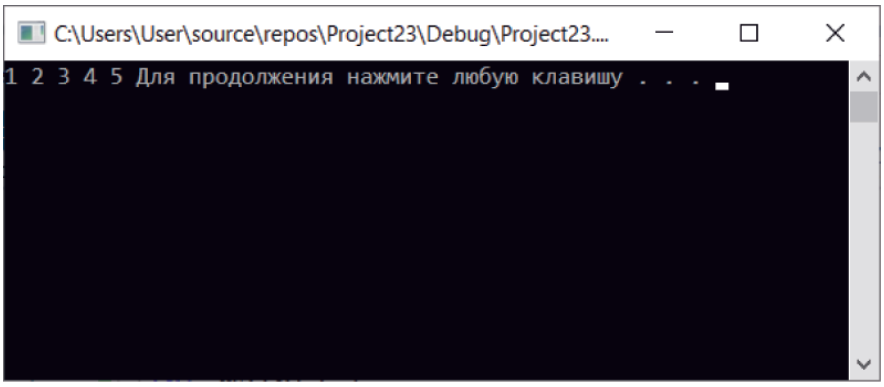


Рис. 12 Последовательный вывод через пробел целых чисел от 1 до 5. Microsoft Visual Studio Code 2019

Оператор `for` выполняется следующим образом: переменной цикла `i` присваивается начальное значение 1. Проверяется условие. Здесь при `i` = 1 оно выполняется (имеет значение `true`). Далее выполняется оператор в цикле и на консоль выводится значение 1. Затем `i` увеличивается на 1 и снова проверяется условие. И так до тех пор, пока не станет `i` > 5 (условие получает значение `false`) и не произойдет выход из цикла.

Все параметры оператора `for` являются необязательными (при определении оператора каждый параметр заключен в квадратные скобки) и, следовательно, могут отсутствовать либо по отдельности, либо все вместе.

Оператор *`break`* позволяет завершить выполнение цикла.

Пример 2. Выведем целые числа от 1 до 5 через пробел.

```
#include <iostream>

using namespace std;

int main() {
    for (int i = 1; ; i++) {
         if (i > 5)
              break;
         cout << i << " ";
    }
    system("pause");
    return 0;
}
```

Оператор *continue* возобновляет выполнение цикла с первого оператора, игнорируя следующие за ним операторы.

Пример 3. Выведем целые числа от 1 до 5 через пробел.

```
#include <iostream>

using namespace std;

int main() {
    for (int i = 1; ; i++) {
         cout << i << " ";
         if (i < 5)
              continue;
         else
              break;
    }
    system("pause");
    return 0;
}
```

Пример 4. Четные числа. Вводятся целые числа a и b. Гарантируется, что a не превосходит b. Выведите (через пробел) все четные числа от a до b (включительно).

```
#include <iostream>

using namespace std;

int main() {
    int a = 0, b = 0;
    cin >> a >> b;
    for (int i = a; i <= b ; i++) {
         if (i % 2 == 0)
              cout << i << " ";
    }
    system("pause");
    return 0;
}
```

Пример 5. Вычислите сумму данных n натуральных чисел. Вводится число n, а затем n чисел, сумму которых необходимо вычислить. Выведите единственное число – сумму введенных чисел.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0, sum = 0, x = 0;
    cin >> n;
```

```
    for (int i = 0; i < n ; i++) {
         cin >> x;
         sum += x;
    }
    cout << sum;
    system("pause");
    return 0;
}
```

Пример 6. Напишите программу, вычисляющую 2^n. Вводится целое неотрицательное число n, которое не превосходит 30. Выведите число 2^n.

```
#include <iostream>

using namespace std;

int main() {
    int n = 0, k = 1;
    cin >> n;
    for (int i = 0; i < n ; i++) {
         k *= 2;
    }
    cout << k;
    system("pause");
    return 0;
}
```

9. Массивы

Массив представляет собой структуру данных, содержащую определенное число переменных, доступ к которым осуществляется с помощью индексов. Содержащиеся в массиве переменные также называются элементами массива и имеют одинаковый тип, который называется типом элементов массива.

Все массивы должны быть объявлены и инициализированы перед их использованием. При объявлении массива нужно указать тип элементов массива.

Индексы массива начинаются с нуля.

Пример 1. Объявление массива. В данном примере создадим массив из пяти элементов. Выведем на экран значение нулевого элемента массива.

```
#include <iostream>

using namespace std;

int main() {
    int a[5] = {10, 20, 30, 40, 50};
    cout << a[0]; // выведет 10
    system("pause");
    return 0;
}
```

Пример 2. Ввод-вывод элементов массива. Введем 10 элементов массива с консоли и выведем их в консоль.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
    for (int i = 0; i < 10; i++) {
         cin >> arr[i];
    }
    for (int i = 0; i < 10; i++) {
         cout << arr[i] << " ";
    }
    system("pause");
    return 0;
}
```

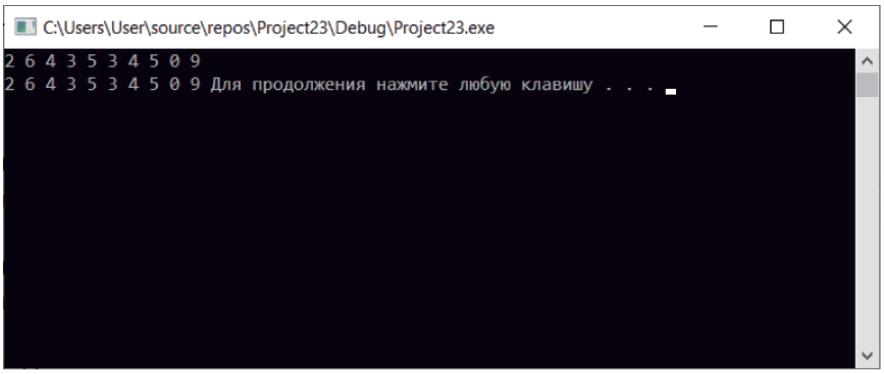


Рис. 13 Ввод-вывод элементов массива. Microsoft Visual Studio Code 2019

Пример 3. Вводится 10 элементов массива с консоли. Необходимо найти и вывести максимальный элемент массива.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
    for (int i = 0; i < 10; i++) {
         cin >> arr[i];
    }
    int mx = arr[0];
    for (int i = 1; i < 10; i++) {
         if (arr[i] > mx)
              mx = arr[i];
    }
    cout << mx;
    system("pause");
    return 0;
}
```

Пример 4. Вводится 10 элементов массива с консоли. Необходимо найти и вывести индекс первого максимального элемента массива.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
    for (int i = 0; i < 10; i++) {
         cin >> arr[i];
    }
    int imx = 0;
    for (int i = 1; i < 10; i++) {
         if (arr[i] > arr[imx])
              imx = i;
    }
    cout << imx;
    system("pause");
    return 0;
}
```

Пример 5. Вводится 10 элементов массива с консоли. Необходимо найти и вывести индекс последнего максимального элемента массива.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
    for (int i = 0; i < 10; i++) {
         cin >> arr[i];
    }
    int imx = 0;
    for (int i = 1; i < 10; i++) {
         if (arr[i] >= arr[imx])
              imx = i;
    }
    cout << imx;
    system("pause");
    return 0;
}
```

Пример 6. Вводится 10 элементов массива с консоли. Необходимо найти и вывести индекс первого отрицательного элемента массива.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
```

```
    for (int i = 0; i < 10; i++) {
        cin >> arr[i];
    }
    int in = -1;
    for (int i = 0; i < 10; i++) {
        if (arr[i] < 0) {
            in = i;
            break;
        }
    }
    if (in == -1)
        cout << "no negative value";
    else
        cout << in;
    system("pause");
    return 0;
}
```

Пример 7. Вводится 10 элементов массива с консоли. Необходимо найти и вывести индекс последнего отрицательного элемента массива.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
    for (int i = 0; i < 10; i++) {
        cin >> arr[i];
    }
    int in = -1;
    for (int i = 0; i < 10; i++) {
        if (arr[i] < 0)
            in = i;
    }
    if (in == -1)
        cout << "no negative value";
    else
        cout << in;
    system("pause");
    return 0;
}
```

Пример 8. Вводится 10 элементов массива с консоли. Необходимо поменять местами первый отрицательный и последний максимальный элементы массива. Гарантируется, что отрицательный элемент есть во входных данных. Вывести полученный массив в консоль.

```
#include <iostream>

using namespace std;

int main() {
    int arr[10];
```

```
    for (int i = 0; i < 10; i++) {
         cin >> arr[i];
    }
    int in = -1;
    for (int i = 0; i < 10; i++) {
         if (arr[i] < 0) {
              in = i;
              break;
         }
    }
    int imx = 0;
    for (int i = 1; i < 10; i++) {
         if (arr[i] >= arr[imx])
              imx = i;
    }
    int tmp = arr[in];
    arr[in] = arr[imx];
    arr[imx] = tmp;
    for (int i = 0; i < 10; i++) {
         cout << arr[i] << " ";
    }
    system("pause");
    return 0;
}
```

Пример 9. Вводится 10 элементов массива с консоли. Необходимо заменить первый элемент масссива (0 индекс) средним значением всех элементов массива. Вывести полученный массив в консоль.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    double arr[10];
    double sum = 0.0;
    for (int i = 0; i < 10; i++) {
         cin >> arr[i];
         sum += arr[i];
    }
    arr[0] = sum / 10;
    for (int i = 0; i < 10; i++) {
         cout << setprecision(2) << fixed << arr[i] << " ";
    }
    system("pause");
    return 0;
}
```