Отчет лаба 1, с++, Носкова Алёна ит-5-2024

**Задание 1. Методы**

1 задача:

Сумма знаков.

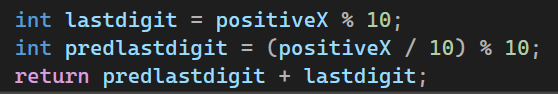
Дана сигнатура функции: **int sumLastNums** (**int x**);

Необходимо реализовать функцию таким образом, чтобы она возвращала результат сложения двух последних знаков числа**х**, предполагая, что знаков в числе не менее двух.

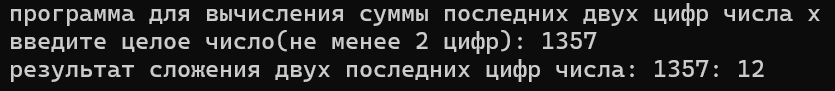
Пример: x=4568 результат: 14

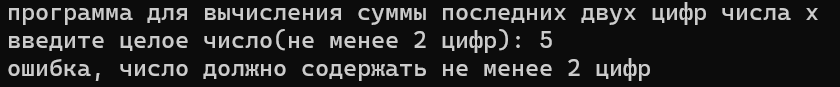
Алгоритм решения:

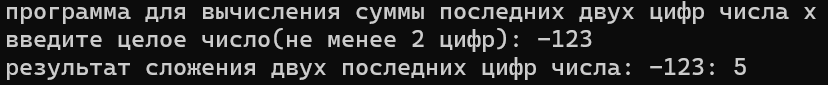
1. Проверяем, чтобы в числе было >= 2 цирф
2. Берем модуль числа(чтобы работать с положительными числами и сумма последних двух цифр не была отрицательной)
3. Выделяем последнюю и предпоследнюю цифры числа и складываем их



Пример работы:







2 задача:

Есть ли позитив.

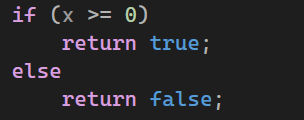
Дана сигнатура функции:  **bool isPositive** (**intx**);

Необходимо реализовать функцию таким образом, чтобы она принимала число **x** и возвращала **true**, если оно положительное.

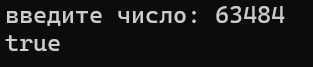
Алгоритм решения:

1. Проверяем чтобы Х был >= 0

* Если больше – возвращаем true
* Если меньше – возвращаем false



Пример работы:





3 задача:

Большая буква.

Дана сигнатура функции:  **bool isUpperCase** (**char x**);

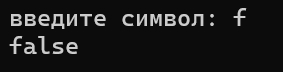
Необходимо реализовать функцию таким образом, чтобы она принимала символ **x** и возвращала **true**, если это большая буква в диапазоне от ‘A’ до ‘Z’.

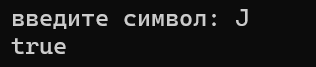
Алгоритм решения:

1. **Проверка** является ли символ заглавной буквой через функцию isUpperCase()



Пример работы:





4 задача:

Делитель.

Дана сигнатура функции: **bool isDivisor** (**int a, int b**);

Необходимо реализовать функцию таким образом, чтобы она возвращала **true**, если любое из принятых чисел делит другое нацело.

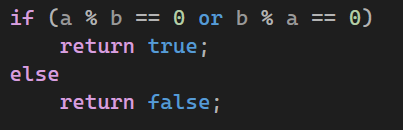
Алгоритм решения:

1. Проверяем делится ли a на b
2. Проверяем делится ли b на a
3. Если выполняется одно из условий

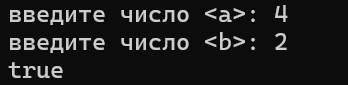
* Возвращаем true

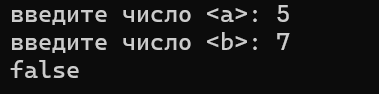
1. Иначе

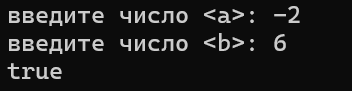
* Возвращаем false



Пример работы:







5 задача:

Многократный вызов.

Дана сигнатура функции: **int lastNumSum**(**int a, int b**)

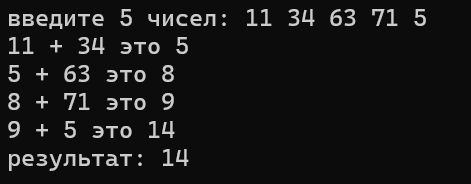
Необходимо реализовать функцию таким образом, чтобы она считала сумму цифр двух чисел из разряда единиц. Выполните с его помощью последовательное сложение пяти чисел и результат выведите на экран.

Алгоритм решения:

1. Получаем последнюю цифру числа а: а % 10
2. Получаем последнюю цифру числа b: b % 10
3. Складываем полученные цифры
4. Возвращаем результат сложения



Пример работы:



**Задание 2. Условия**

1 задача:

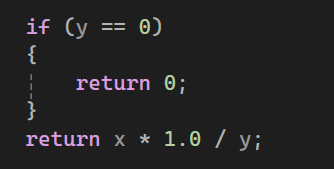
Безопасное деление.

Дана сигнатура функции: **double safeDiv** (**int x, int y**);

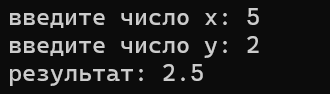
Необходимо реализовать функцию таким образом, чтобы она возвращала деление **x** на **y**, и при этом гарантировала, что не будет выкинута ошибка деления на 0. При делении на 0 следует вернуть из функции число **0**.

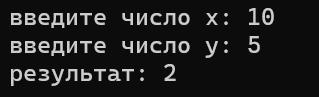
Алгоритм решения:

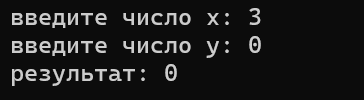
1. Проверяем делитель y( если у == 0, возвращаем 0)
2. Преобразовываем х в вещественное число (х \* 1.0)
3. Делим на у
4. Возвращаем результат



Пример работы:







2 задача:

Строка сравнения.

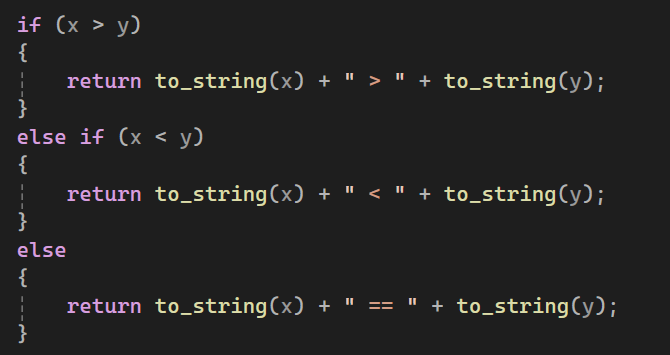
Дана сигнатура функции: **String makeDecision** (**int x, int y**);

Необходимо реализовать функцию таким образом, чтобы она возвращала строку, которая включает два принятых функцией числа и корректно выставленный знак операции сравнения (больше, меньше, или равно).

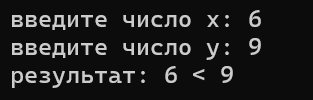
Алгоритм решения:

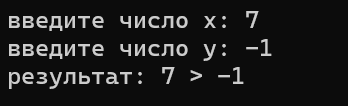
1. Сравнить числа х и у

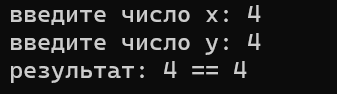
* Если x > y, преобразовать х и у в строки, объединить to\_string(x) + " > " + to\_string(y), вернуть результат
* Если x < y, преобразовать х и у в строки, объединить to\_string(x) + " < " + to\_string(y), вернуть результат
* Если x == y, преобразовать х и у в строки, объединить to\_string(x) + " == " + to\_string(y), вернуть результат



Пример работы:







3 задача:

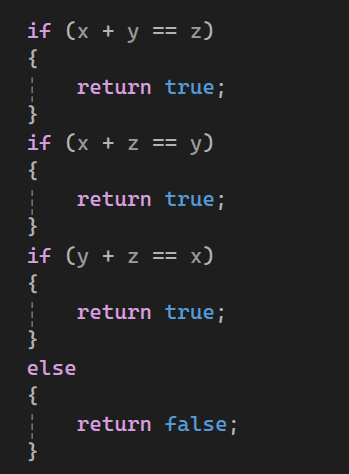
Тройная сумма.

Дана сигнатура функции: **bool sum3** (**int x, int y, int z**);

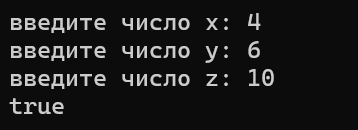
Необходимо реализовать функцию таким образом, чтобы она возвращала **true**, если два любых числа (из трех принятых) можно сложить так, чтобы получить третье.

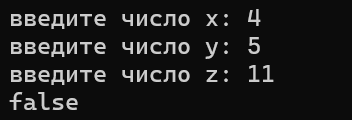
Алгоритм решения:

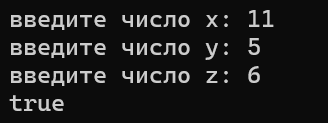
1. Проверить первую пару:
   * Вычислить сумму чисел x и y
   * Сравнить полученную сумму с числом z
   * Если x + y == z то вернуть true
2. Проверить вторую пару:
   * Вычислить сумму чисел x и z
   * Сравнить полученную сумму с числом y
   * Если x + z == y то вернуть true
3. Проверить третью пару:
   * Вычислить сумму чисел y и z
   * Сравнить полученную сумму с числом x
   * Если y + z == x то вернуть true
4. Если ни одно условие не выполнилось:
   * Вернуть false



Пример работы:







4 задача:

Дана сигнатура функции: **String age** (**int x**);

Необходимо реализовать функцию таким образом, чтобы она возвращала строку, в которой сначала будет число **х**, а затем одно из слов:

* год
* года
* лет

Слово “год” добавляется, если число **х** заканчивается на 1, кроме числа 11. Слово “года” добавляется, если число **х** заканчивается на 2, 3 или 4, кроме чисел 12, 13, 14.

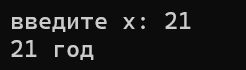
Слово “лет”добавляется во всех остальных случаях.

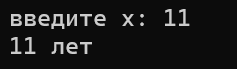
Алгоритм решения:

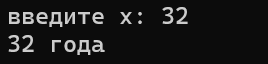
1. Вычислить последнюю цифру: lastdigit = x % 10
2. Вычислить две последние цифры: twolastdigit = x % 100
3. Проверить особый случай (11-14 лет):
   * Если twolastdigit от 11 до 14 → вернуть "x лет"
4. Проверить окончание для 1:
   * Если lastdigit == 1 → вернуть "x год"
5. Проверить окончание для 2-4:
   * Если lastdigit от 2 до 4 → вернуть "x года"
6. Для всех остальных случаев:
   * Вернуть "x лет"



Пример работы:







5 задача:

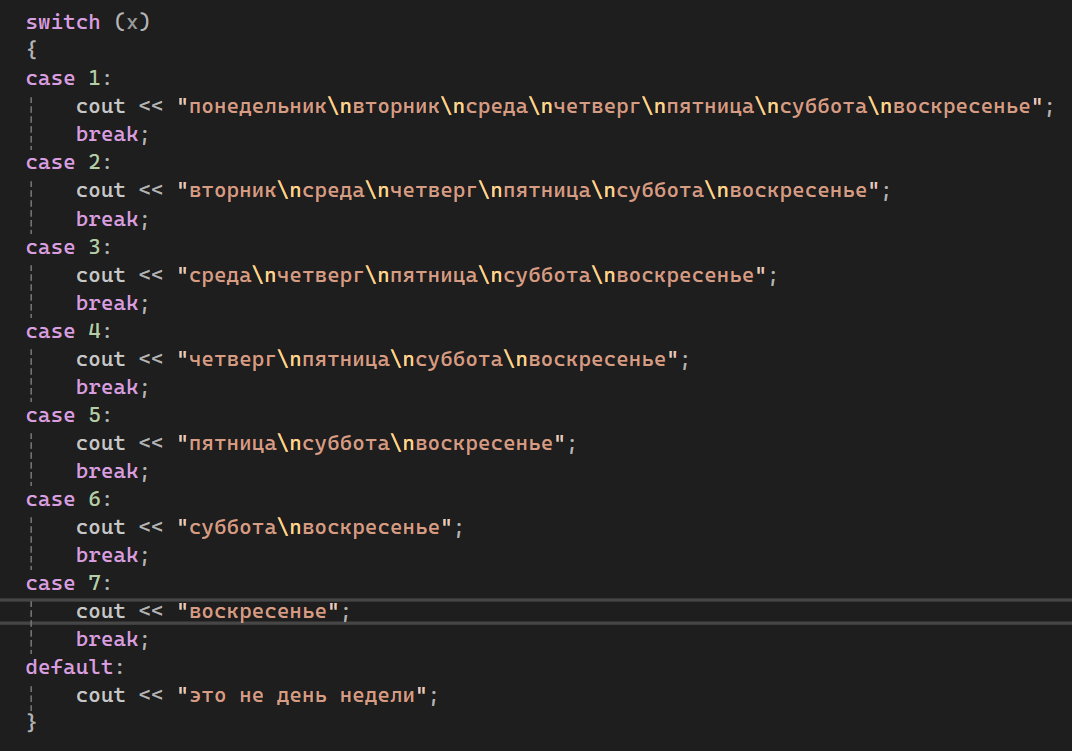
Вывод дней недели.

Дана сигнатура функции:  **void printDays** (int **x**);

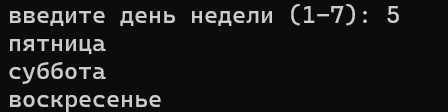
В качестве параметра функция принимает число x, обозначаются день недели. Необходимо реализовать функцию таким образом, чтобы она выводила на экран название переданного в него дня и всех последующих до конца недели дней. Если в качестве параметра передан не день (число, не в диапазоне от 1 от 7), то выводится текст “это не день недели”. Первый день понедельник, последний – воскресенье. Вместо **if** в данной задаче используйте **switch**.

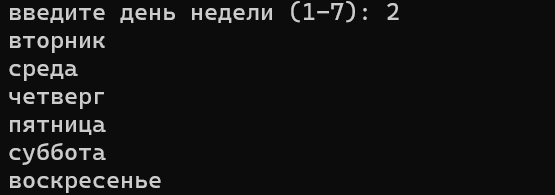
Алгоритм решения:

1. Проверить значение x через switch-case
2. Для каждого случая:
   * case 1: Вывести дни с понедельника по воскресенье
   * case 2: Вывести дни со вторника по воскресенье
   * case 3: Вывести дни со среды по воскресенье
   * case 4: Вывести дни с четверга по воскресенье
   * case 5: Вывести дни с пятницы по воскресенье
   * case 6: Вывести субботу и воскресенье
   * case 7: Вывести воскресенье
   * default: Сообщение об ошибке



Пример работы:







**Задание 3. Циклы**

1 задача:

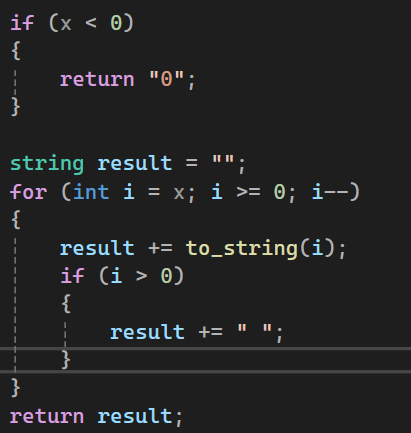
Числа наоборот.

Дана сигнатура функции: **String reverseListNums** (**int x**);

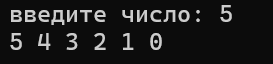
Необходимо реализовать функцию таким образом, чтобы она возвращала строку, в которой будут записаны все числа от **x** до **0** (включительно).

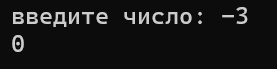
Алгоритм решения:

1. Проверить отрицательное число:
   * Если x < 0 → вернуть "0"
2. Инициализировать пустую строку: result = ""
3. Цикл от x до 0 (включительно):
   * Начальное значение: i = x
   * Условие: i >= 0
   * Шаг: i-- (уменьшение на 1)
4. В каждой итерации цикла:
   * Преобразовать число i в строку
   * Добавить к результату: result += to\_string(i)
   * Если i > 0 → добавить пробел: result += " "
5. Вернуть полученную строку



Пример работы:





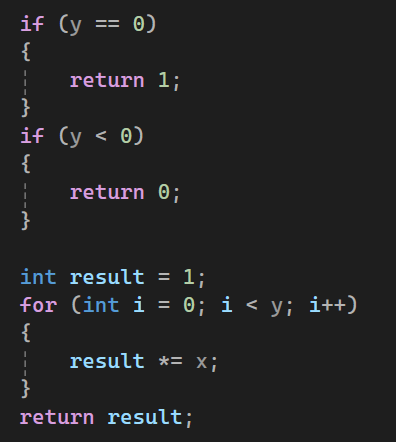
2 задача:

Дана сигнатура функции: **int pow** (**int x, int y**);

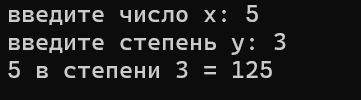
Необходимо реализовать функцию таким образом, чтобы она возвращала результат возведения **x** в степень **y**.

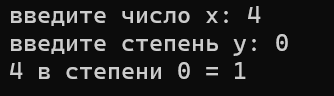
Алгоритм решения:

1. Проверить особые случаи:
   * Если y == 0 → вернуть 1 (любое число в степени 0 равно 1)
   * Если y < 0 → вернуть 0
2. Инициализировать результат: result = 1
3. Цикл умножения:
   * Повторить y раз: for (int i = 0; i < y; i++)
   * В каждой итерации: result = result \* x
4. Вернуть полученный результат



Пример работы:





3 задача:

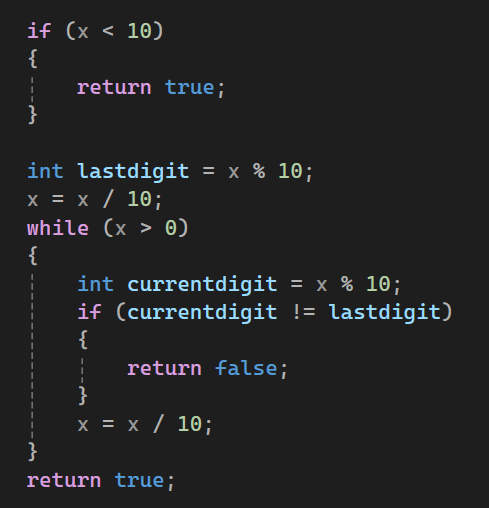
Одинаковость.

Дана сигнатура функции:  **bool equalNum** (**int x**);

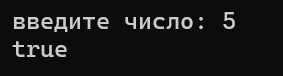
Необходимо реализовать функцию таким образом, чтобы она возвращала **true**, если все знаки числа одинаковы, и **false** в ином случае.

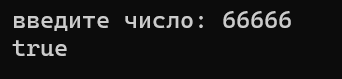
Алгоритм решения:

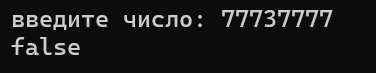
1. Проверить однозначное число:
   * Если x < 10 → вернуть true (одна цифра всегда "одинаковая")
2. Запомнить последнюю цифру: lastdigit = x % 10
3. Убрать последнюю цифру: x = x / 10
4. Цикл пока x > 0:
   * Извлечь текущую цифру: currentdigit = x % 10
   * Сравнить с последней цифрой:
     + Если currentdigit != lastdigit → вернуть false
   * Убрать обработанную цифру: x = x / 10
5. Если цикл завершился без возврата false:
   * Вернуть true (все цифры одинаковые)



Пример работы:







4 задача:

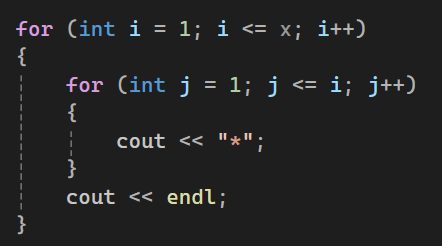
Левый треугольник.

Дана сигнатура функции: **void leftTriangle** (**int x**);

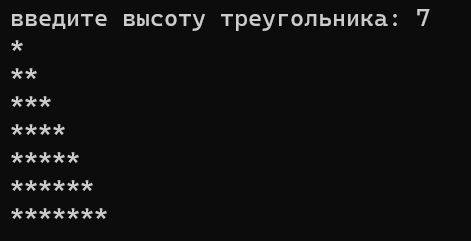
Необходимо реализовать функцию таким образом, чтобы она выводила на экран треугольник из символов ‘\*’ у которого **х** символов в высоту, а количество символов в ряду совпадает с номером строки**.**

Алгоритм решения:

1. Внешний цикл по строкам:
   * for (int i = 1; i <= x; i++) - от 1 до x строк
2. Внутренний цикл по звездочкам в строке:
   * for (int j = 1; j <= i; j++) - в строке i рисуем i звездочек
3. В каждой итерации внутреннего цикла:
   * Вывести звездочку: cout << "\*"
4. После завершения внутреннего цикла:
   * Перейти на новую строку: cout << endl



Пример работы:





5 задача:

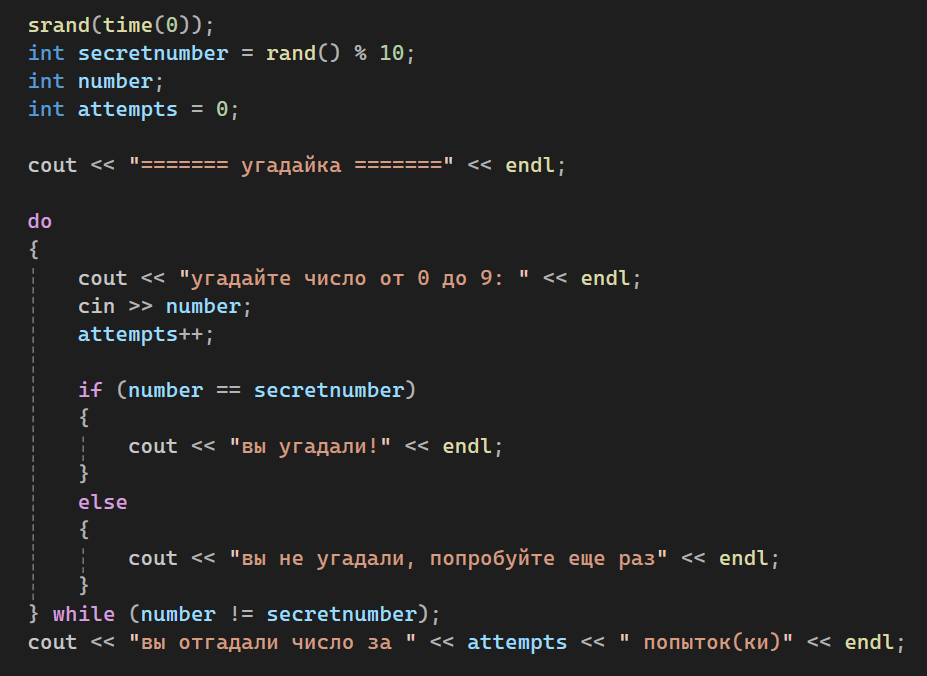
Угадайка.

Дана сигнатура функции:  **void guessGame**()

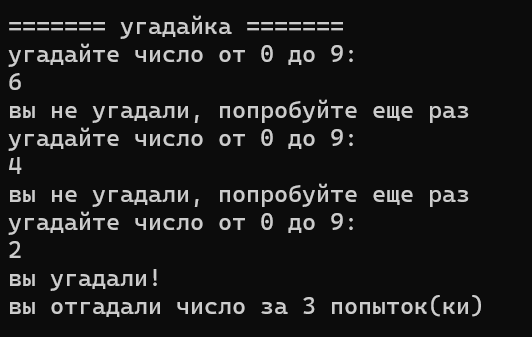
Необходимо реализовать функцию таким образом, чтобы она генерировала случайное число от 0 до 9, далее считывала с консоли введенное пользователем число и выводила, угадал ли пользователь то, что было загадано, или нет. Функция запускается до тех пор, пока пользователь не угадает число. После этого выведите на экран количество попыток, которое потребовалось пользователю, чтобы угадать число.

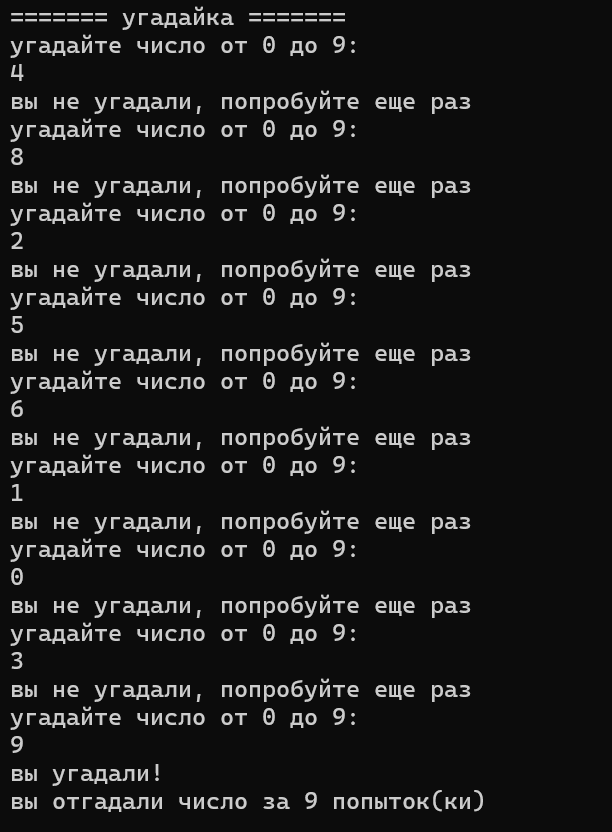
Алгоритм решения:

1. Инициализация:
   * Генерация случайного числа: secretnumber = rand() % 10
   * Инициализация счетчика попыток: attempts = 0
2. Вывод заголовка игры
3. Цикл do-while:
   * Вывод приглашения для ввода числа
   * Считывание числа от пользователя
   * Увеличение счетчика попыток
   * Проверка совпадения:
     + Если угадал → поздравление
     + Если не угадал → сообщение "попробуйте еще раз"
   * Условие продолжения: пока number != secretnumber
4. Вывод результата: количество затраченных попыток



Пример работы:





**Задание 4. Массивы**

1 задача:

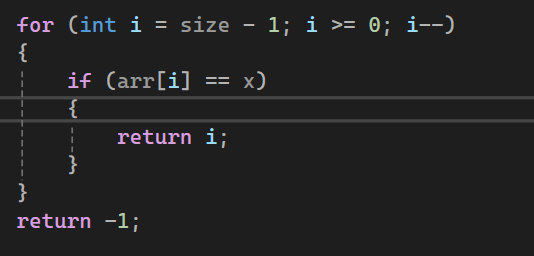
Поиск последнего значения.

Дана сигнатура функции: **int findLast** (**int arr[]**, **int x**);

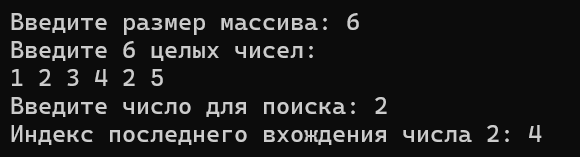
Необходимо реализовать функцию таким образом, чтобы она возвращала индекс последнего вхождения числа **x** в массив **arr**. Если число не входит в массив – возвращается -1.

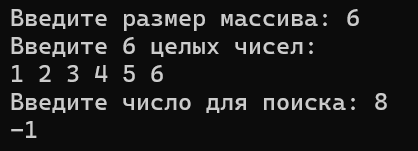
Алгоритм решения:

1. Начать с конца массива: i = size - 1
2. Цикл while (пока i ≥ 0):
   * Проверить текущий элемент: if (arr[i] == x)
   * Если нашли → вернуть индекс i
   * Перейти к предыдущему элементу: i--
3. Если цикл завершился: вернуть -1 (элемент не найден)



Пример работы:





2 задача:

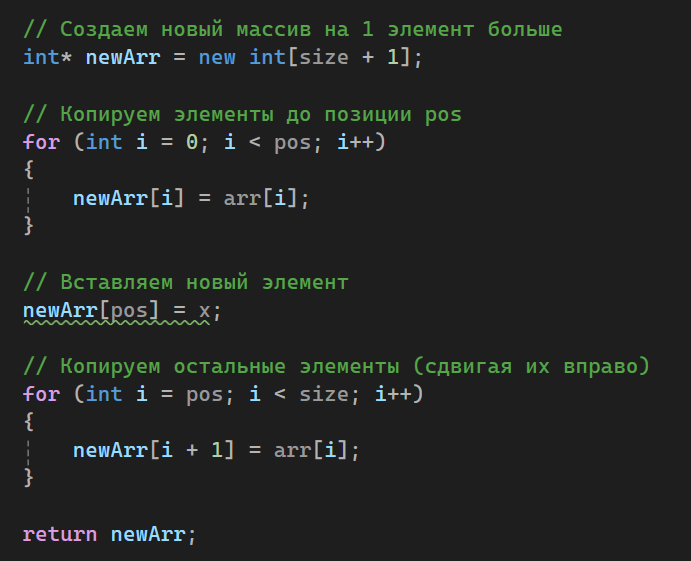
Добавление в массив.

Дана сигнатура функции: **int \* add** (**int arr[], int x, int pos**);

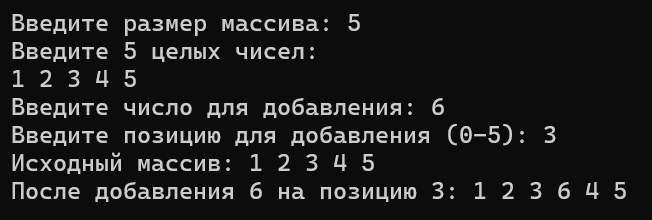
Необходимо реализовать функцию таким образом, чтобы она возвращала новый массив, который будет содержать все элементы массива **arr**, однако в позицию **pos** будет вставлено значение **x**.

Алгоритм решения:

1. Выделить память под новый массив (размер + 1)
2. Скопировать элементы до позиции вставки
3. Вставить новый элемент на указанную позицию
4. Скопировать оставшиеся элементы со сдвигом
5. Вернуть указатель на новый массив



Пример работы:

****

3 задача:

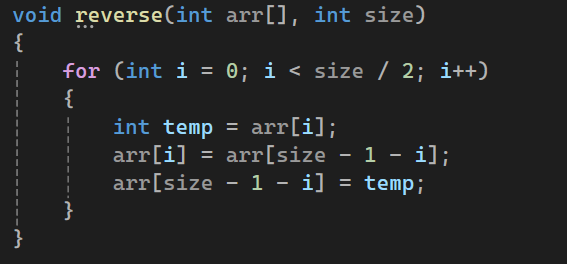
Реверс.

Дана сигнатура функции: **void reverse** (**int arr[]**);

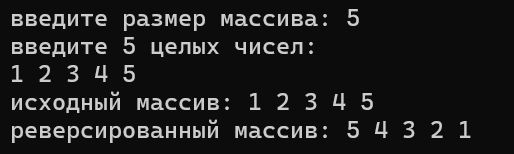
Необходимо реализовать функцию таким образом, чтобы он изменял массив **arr.** После проведенных изменений массив должен быть записан задом-наперед.

Алгоритм решения:

1. Инициализация:
   * Определить количество пар для обмена: предел = size / 2
2. Цикл попарного обмена:
   * Для i от 0 до предел - 1:
3. Обмен элементов:
   * Сохранить левый элемент: временный = arr[i]
   * Левый элемент ← правый элемент: arr[i] = arr[size - 1 - i]
   * Правый элемент ← сохраненный: arr[size - 1 - i] = временный
4. Завершение: массив полностью развернут



Пример работы:



4 задача:

Объединение.

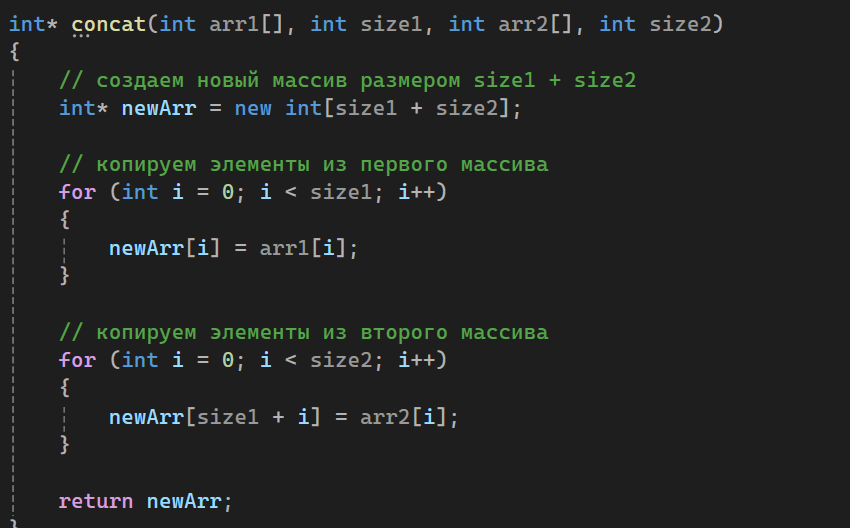
Дана сигнатура функции: **int \* concat** (**int arr1[]**,**int arr2[]**);

Необходимо реализовать функцию таким образом, чтобы она возвращала новый массив, в котором сначала идут элементы первого массива (**arr1**), а затем второго

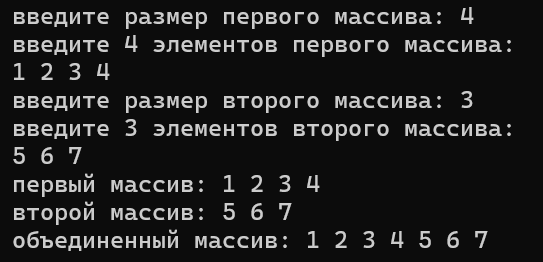
(**arr2**).

Алгоритм решения:

1. Выделение памяти:
   * Рассчитать общий размер: totalSize = size1 + size2
   * Создать новый массив: newArr = new int[totalSize]
2. Копирование первого массива:
   * Для каждого индекса i от 0 до size1 - 1:
   * newArr[i] = arr1[i] (копируем элементы первого массива)
3. Копирование второго массива:
   * Для каждого индекса i от 0 до size2 - 1:
   * newArr[size1 + i] = arr2[i] (добавляем элементы второго массива)
4. Завершение:
   * Вернуть указатель на новый массив



Пример работы:



5 задача:

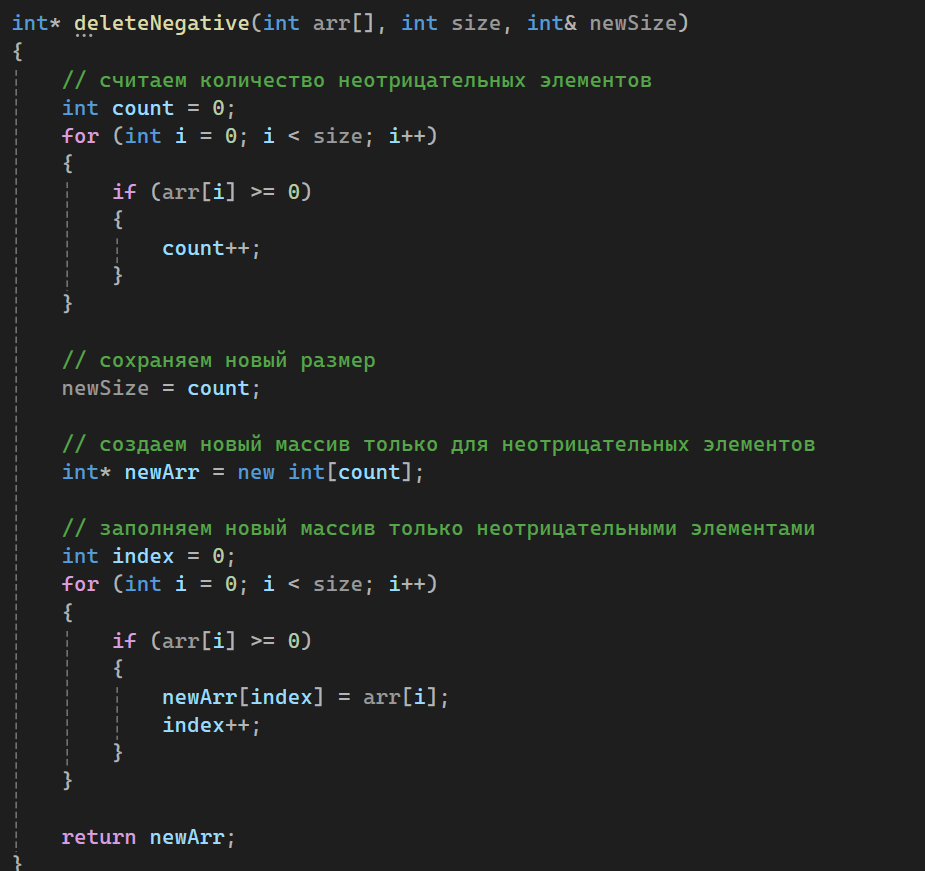
Удалить негатив.

Дана сигнатура функции: **int \* deleteNegative** (**int arr[]**);

Необходимо реализовать функцию таким образом, чтобы она возвращала новый массив, в котором записаны все элементы массива **arr** кроме отрицательных.

Алгоритм решения:

1. Подсчет неотрицательных элементов:
   * Инициализировать счетчик: count = 0
   * Для каждого элемента i от 0 до size-1:
     + Если arr[i] >= 0, то увеличить count на 1
2. Сохранение нового размера:
   * Записать newSize = count
3. Создание нового массива:
   * Выделить память: newArr = new int[count]
4. Заполнение нового массива:
   * Инициализировать индекс: index = 0
   * Для каждого элемента i от 0 до size-1:
     + Если arr[i] >= 0:
       - newArr[index] = arr[i]
       - Увеличить index на 1
5. Завершение:
   * Вернуть newArr



Пример работы:

