

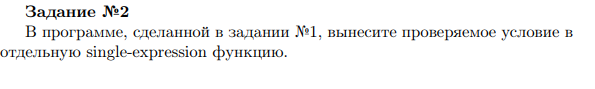
fun main()  
{  
 *print*("Введите неотрицательное целое число: ")  
 // val num = readLine()?.toIntOrNull()  
 val num = 910  
 if (num == null)  
 {  
 *println*("Введено некорректное значение.")  
 return  
 }  
 if (num == 0)  
 {  
 *println*("Результат не может быть вычислен для нулевого значения.")  
 return  
 }  
 if (num < 0)  
 {  
 *println*("Результат не может быть вычислен для отрицательного значения.")  
 return  
 }  
 if(*MinDigitNum*(num) % 3 == 0)  
 *println*("Ошибка: Результат кратен 3")  
 else  
 *println*("Минимальная цифра, некратная трем: ${*MinDigitNum*(num)}")  
}  
fun MinDigitNum(num: Int): Int {  
 var minDigit = 9  
 for (digit in num.toString()) {  
 val digitInt = digit.*digitToInt*()  
 if (*IsMultipleOfThree*(digitInt)) //0 % 3 = 0  
 continue  
 if (*IsMore*(digitInt, minDigit))  
 continue  
 minDigit = digitInt  
 }  
 return minDigit  
}  
fun IsMultipleOfThree(num: Int): Boolean = num % 3 == 0  
fun IsMore(num1: Int, num2: Int): Boolean = num1 >= num2

(+)

2)

fun main()  
{  
 *print*("Введите неотрицательное целое число: ")  
 // val num = readLine()?.toIntOrNull()  
 val num = 920  
 if (num == null)  
 {  
 *println*("Введено некорректное значение.")  
 return  
 }  
 if (num == 0)  
 {  
 *println*("Результат не может быть вычислен для нулевого значения.")  
 return  
 }  
 if (num < 0)  
 {  
 *println*("Результат не может быть вычислен для отрицательного значения.")  
 return  
 }  
 if(*findMinDigitNotMultipleOfThree*(num) % 3 == 0)  
 *println*("Ошибка: Результат кратен 3")  
 else  
 *println*("Минимальная цифра, некратная трем: ${*findMinDigitNotMultipleOfThree*(num)}")  
}  
tailrec fun findMinDigitNotMultipleOfThree(num: Int, currentMin: Int = 9): Int {  
 return when {  
 num <= 0 -> currentMin  
 num % 10 % 3 != 0 && num % 10 < currentMin -> *findMinDigitNotMultipleOfThree*(num / 10, num % 10)  
 else -> *findMinDigitNotMultipleOfThree*(num / 10, currentMin)  
 }  
}

(+)



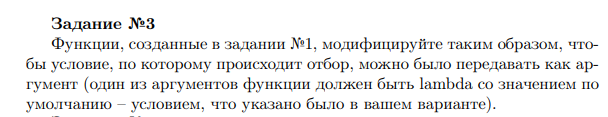
fun main()  
{  
 *print*("Введите неотрицательное целое число: ")  
 // val num = readLine()?.toIntOrNull()  
 val num = 910  
 if (num == null)  
 {  
 *println*("Введено некорректное значение.")  
 return  
 }  
 if (num == 0)  
 {  
 *println*("Результат не может быть вычислен для нулевого значения.")  
 return  
 }  
 if (num < 0)  
 {  
 *println*("Результат не может быть вычислен для отрицательного значения.")  
 return  
 }  
 if(*MinDigitNum*(num) % 3 == 0)  
 *println*("Ошибка: Результат кратен 3")  
 else  
 *println*("Минимальная цифра, некратная трем: ${*MinDigitNum*(num)}")  
}  
fun MinDigitNum(num: Int): Int {  
 var minDigit = 9  
 for (digit in num.toString()) {  
 val digitInt = digit.*digitToInt*()  
 if (*IsMultipleOfThree*(digitInt)) //0 % 3 = 0  
 continue  
 if (*IsMore*(digitInt, minDigit))  
 continue  
 minDigit = digitInt  
 }  
 return minDigit  
}  
fun IsPositiveNum(num: Int): Boolean = num > 0  
fun IsMultipleOfThree(num: Int): Boolean = num % 3 == 0  
fun IsMore(num1: Int, num2: Int): Boolean = num1 >= num2

(+)

2)

fun main() {  
 *println*("Введите неотрицательное целое число: ")  
 // val num = readLine()?.toIntOrNull()  
 val num = 900  
 if (num == null) {  
 *println*("Введено некорректное значение.")  
 return  
 }  
 if (num == 0) {  
 *println*("Результат не может быть вычислен для нулевого значения.")  
 return  
 }  
 if (num < 0) {  
 *println*("Результат не может быть вычислен для отрицательного значения.")  
 return  
 }  
 if (*isMultipleOfThree*(*MinDigitNum*(num))) {  
 *println*("Ошибка: Результат кратен 3: ${*MinDigitNum*(num)}")  
 } else {  
 *println*("Минимальная цифра, некратная трем: ${*MinDigitNum*(num)}")  
 }  
}  
  
tailrec fun findMinDigitNotMultipleOfThree(num: Int, currentMin: Int = 9): Int {  
 return when {  
 num <= 0 -> currentMin  
 num % 10 % 3 != 0 && num % 10 < currentMin -> *findMinDigitNotMultipleOfThree*(num / 10, num % 10)  
 else -> *findMinDigitNotMultipleOfThree*(num / 10, currentMin)  
 }  
}  
fun MinDigitNum(num: Int): Int {  
 return *findMinDigitNotMultipleOfThree*(num)  
}  
fun isPositiveNum(num: Int): Boolean = num > 0  
fun isMultipleOfThree(num: Int): Boolean = num % 3 == 0  
fun isMore(num1: Int, num2: Int): Boolean = num1 >= num2

(+)

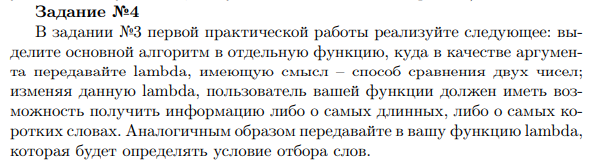


fun main() {  
 *print*("Введите неотрицательное целое число: ")  
 val num = 901  
  
 if (num == null) {  
 *println*("Введено некорректное значение.")  
 return  
 }  
 if (num == 0) {  
 *println*("Результат не может быть вычислен для нулевого значения.")  
 return  
 }  
 if (num < 0) {  
 *println*("Результат не может быть вычислен для отрицательного значения.")  
 return  
 }  
  
 var res = *MinDigitNum*(  
 num,  
 **{** digit **->** digit % 3 != 0 **}**, // Изменено, чтобы получить цифры, не кратные трём  
 **{** num1, num2 **->** num1 >= num2 **}** )  
  
 if (res % 3 == 0) {  
 *println*("Ошибка: Результат кратен 3: $res")  
 } else {  
 *println*("Минимальная цифра, некратная трём: $res")  
 }  
}  
  
fun MinDigitNum(  
 num: Int,  
 IsNotMultipleOfThree: (Int) -> Boolean = **{** digit **->** digit % 3 != 0 **}**,  
 IsMore: (Int, Int) -> Boolean = **{** num1, num2 **->** num1 >= num2 **}**): Int {  
 var minDigit = 9  
 for (digit in num.toString()) {  
 val digitInt = digit.*digitToInt*()  
 if (!IsNotMultipleOfThree(digitInt)) // Изменено на отрицание, чтобы исключить кратные трём цифры  
 continue  
 if (IsMore(digitInt, minDigit))  
 continue  
 minDigit = digitInt  
 }  
 return minDigit  
}

(2)

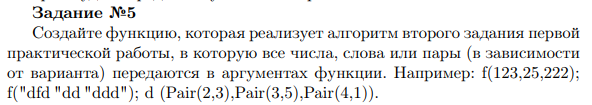
fun main() {  
 *print*("Введите неотрицательное целое число: ")  
 val num = 98620  
  
 if (num <= 0) {  
 *println*("Введено некорректное значение.")  
 return  
 }  
  
 val res = *findMinDigitNotMultipleOfThree*(num  
 ,**{** digit **->** digit % 10 % 3 != 0 **}** ,**{** digit **->** digit <= 0 **}** )  
  
 if (res % 3 == 0) {  
 *println*("Ошибка: Результат кратен 3")  
 } else {  
 *println*("Минимальная цифра, некратная трем: $res")  
 }  
}  
  
tailrec fun findMinDigitNotMultipleOfThree(  
 num: Int,  
 Condition: (Int) -> Boolean,  
 IsNotOrEqualZero: (Int) -> Boolean,  
 currentMin: Int = 9  
): Int {  
 return when {  
 num == 0 -> currentMin  
 IsNotOrEqualZero(num % 10) -> *findMinDigitNotMultipleOfThree*(num / 10, Condition, IsNotOrEqualZero, currentMin)  
 Condition(num % 10) && num % 10 < currentMin -> *findMinDigitNotMultipleOfThree*(num / 10, Condition, IsNotOrEqualZero, num % 10)  
 else -> *findMinDigitNotMultipleOfThree*(num / 10, Condition, IsNotOrEqualZero, currentMin)  
 }  
}

(показать)



fun main(args: Array<String>) {  
 val str = "111111 22 3333 4 555 6666 77 88888"  
 if (str.*isEmpty*())  
 {  
 *println*("Ответ: Строка пустая")  
 return  
 }  
 var res = *findLastChar*(  
 str  
 ,**{it**.*isWhitespace*() || **it** == '.'**}** ,**{it** % 2 == 0**}** )  
 if(res == ' ')  
 {  
 *println*("Нет ответа")  
 return  
 }  
 else  
 *println*("Последний символ в последнем самом коротком слове\n" +  
 " с чётным числом символов: ${res}") //чётное слов  
}  
fun findLastChar(str: String  
 ,IsNotSymbol:(Char) -> Boolean = **{it**.*isWhitespace*() || **it** == '.'**}** ,IsEven: (Int) -> Boolean = **{it** % 2 == 0**}**) : Char  
{  
 var size = 0  
 var oldSize = Int.MAX\_VALUE  
 var res = ' '  
 for (i in 0 *until* str.length)  
 {  
 if (IsNotSymbol(str[i]))  
 {  
 if (IsEven(size) && size <= oldSize) \*обработать >=  
 {  
 res = str[i - 1]  
 oldSize = size  
 }  
  
 size = 0  
 }  
 else  
 size++  
 }  
 if (IsEven(size) && size <= oldSize)  
 {  
 res = str[str.length - 1]  
 oldSize = size  
 }  
 return res  
}

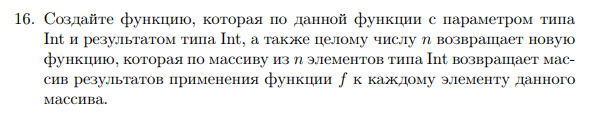
if (IsEven(size) && size <= oldSize) \*обработать >=



//f("dfd", "dd", "ddd"); //любое количество строк  
fun main(args: Array<String>)  
{  
 *println*("Количество слов с повторяющимися символами: ${*getCountDuplicatedWord*("ugu", "bugu")}")  
}  
  
fun getCountDuplicatedWord(vararg words: String): Int {  
 var count = 0  
 for (str in words)  
 {  
 var isCharacterInWord = BooleanArray(128)  
 for (symbol in str)  
 {  
 if (isCharacterInWord[symbol.*code*])  
 {  
 count++  
 break  
 }  
 isCharacterInWord[symbol.*code*] = true;  
 }  
 }  
 return count  
}

(+)





fun main() {  
 val func: (Int) -> Int = **{ it** + 1 **}** val n = 5  
 val oldArray = IntArray(n) **{** i **->** (i + 1) **}** val mapper = *getMapper*(func, n)  
 val newArray = mapper(oldArray)  
 for (el in newArray) {  
 *println*(el)  
 }  
}  
  
fun getMapper(f: (Int) -> Int, n: Int): (IntArray) -> IntArray {  
 return **{** array **->** val resultArray = IntArray(n)  
 for (i in 0 *until* n) {  
 resultArray[i] = f(array[i])  
 }  
 resultArray  
 **}**}

(+)