

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное учреждение высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА  
ЗАЩИЩЕНА С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ

доц., к.ф.-м.н., доцент  
должность, уч. степень, звание

подпись, дата

М. В. Фаттахова  
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ

ТЕМА КУРСОВОЙ РАБОТЫ: Компания «Корвет»

по дисциплине: ПРИКЛАДНЫЕ МОДЕЛИ ОПТИМИЗАЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4932

15.12.2021  
подпись, дата

Н.С. Иванов  
инициалы, фамилия

Санкт-Петербург 2021

## Оглавление

Задача .....	2
Этап 1. Математическая модель .....	4
Этап 2. Решение с помощью MS Excel .....	5
Этап 3. Приложение-интерфейс к задаче.....	6
Приложения .....	6
Список литературы.....	6
Код .....	6

## Задача

### **Задача 4. Компания «Корвет»**

Компания «Корвет» производит программное обеспечение на CD-ROM, которое продается в пакете с драйверами CD-ROM основными производителями компьютерного оборудования. Она оценивает возможность развития 6 новых программных приложений. В таблице представлена информация о затратах и ожидаемой чистой приведенной прибыли от продажи приложения:

Приложение	Ожидаемые траты на развитие, \$	Требуемое число программистов	Ожидаемая чистая приведенная прибыль, \$
1	400 000	6	2 000 000
2	1 100 000	18	3 600 000
3	940 000	20	4 000 000
4	760 000	16	3 000 000
5	1 260 000	28	4 400 000
6	1 800 000	34	6 200 000

У «Корвета» 60 программистов. Фирма может выделить 3,5 млн долларов на развитие новых программных приложений.

Каков оптимальный набор приложений, которые следует развивать, если:

- ожидается, что клиенты, заинтересованные в приложении 4, будут заинтересованы и в приложении 5, и наоборот. Таким образом, если одно из приложений решено развивать, другое тоже должно быть развито.
- Приобретение приложения 2 имеет смысл, только если в пакет включено приложение 1. Таким образом, приложения 1 и 2 тоже должны быть развиты вместе.

- с) Приложения 3 и 6 эксплуатируют одну и ту же тему. Следовательно, если одно из них развивается, то другое определённо нет.
- d) Стремясь обеспечить качество продукции, «Корвет» не склонен развивать более 3 программных продуктов.

## Этап 1. Математическая модель

Приложение	Ожидаемые траты на развитие, млн \$	Требуемое число программистов	Ожидаемая чистая приведенная прибыль, млн \$
1	0,40	6	2,00
2	1,10	18	3,60
3	0,94	20	4,00
4	0,76	16	3,00
5	1,26	28	4,40
6	1,80	34	6,20

$$x_i = \begin{cases} 0, i \text{ приложение не разрабатывается} \\ 1, i \text{ приложение разрабатывается} \end{cases}$$

$$\begin{cases} 6x_1 + 18x_2 + 20x_3 + 16x_4 + 28x_5 + 34x_6 \leq 60 \\ 0,4x_1 + 1,1x_2 + 0,94x_3 + 0,76x_4 + 1,26x_5 + 1,80x_6 \leq 3,5 \\ x_4 = x_5 \\ x_2 \geq x_1 \\ \sum x_i \leq 3 \\ x_3 + x_6 \leq 1 \end{cases}$$

$$x_i - bin$$

$$z = \max (2x_1 + 3,6x_2 + 4x_3 + 3x_4 + 4,4x_5 + 6,2x_6)$$

## Этап 2. Решение с помощью MS Excel

Таблица описывающая затраты и доход разработки ПО.

	A	B	C	D	E	F	G	H	I	J	K
1									Сумма		Ограничение
2	Программисты, шт	6	18	20	16	28	34		58 <=		60
3	Траты на разработку, млн \$	0,4	1,1	0,94	0,76	1,26	1,8		3,3 <=		3,5
4	Ожидаемый доход, млн \$	2	3,6	4	3	4,4	6,2		11,8		
5											
6	Разрабатывается?	1	1	0	0	0	1		1		
7		x1	x2	x3	x4	x5	x6		x3+x6		
8											
9											
10											
11											
12											
13											

## Параметры поиска решения

Параметры поиска решения

Оптимизировать целевую функцию:

До: ☒ Максимум ☐ Минимум ☐ Значения:

Изменяя ячейки переменных:

В соответствии с ограничениями:

\$B\$6:\$G\$6 <= 3  
\$B\$6:\$G\$6 = бинарное  
\$C\$6 >= \$B\$6  
\$E\$6 = \$F\$6  
\$I\$2 <= \$K\$2  
\$I\$3 <= \$K\$3  
\$I\$6 <= 1

Добавить  
Изменить  
Удалить  
Сбросить  
Загрузить/сохранить

☒ Сделать переменные без ограничений неотрицательными

Выберите метод решения:

Метод решения  
Для гладких нелинейных задач используйте поиск решения нелинейных задач методом ОПГ, для линейных задач - поиск решения линейных задач симплекс-методом, а для негладких задач - эволюционный поиск решения.

Справка

### Этап 3. Приложение-интерфейс к задаче

#### Интерфейс приложения

Untitled

№	Программисты	Траты на разработку	Ожидаемый доход
1	6	0.4	2.01
2	18	1.1	3.6
3	20	0.94	4.0
4	16	0.76	3.0
5	28	1.26	4.4
6	34	1.8	6.2

ЗагрузитьСохранить

### Приложения

#### Список литературы

1. <https://kotlinlang.org> Kotlin Programming Language
2. <https://developer.android.com/jetpack/compose> Jetpack Compose | Android Developers

#### Код

```
// @filename \src\main\kotlin\App.kt
import Data.ExcelReader
import Data.UpdateExcel
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.text.BasicTextField
import androidx.compose.material.Button
import androidx.compose.material.ButtonDefaults
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
```

```

import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import repository.ExpensesRepository
import repository.IncomeRepository
import repository.ProgrammerRepository

```

```

@Composable
fun App() {
    Column {
        tableView()
        downButton()
    }
}

```

```

@Composable
fun downButton() {
    Row {
        Button(
            onClick = { ExcelReader.read() },
            content = { Text("Загрузить") },
            modifier = Modifier.padding(10.dp),
            colors = ButtonDefaults.buttonColors(backgroundColor = Color.Gray)
        )
        Button(
            onClick = { UpdateExcel.changeCalls() },
            content = { Text("Сохранить") },
            modifier = Modifier.padding(10.dp),
            colors = ButtonDefaults.buttonColors(backgroundColor = Color.Gray)
        )
    }
}

```

```

fun checkToDouble(string: String) =
    if (""["^\\d\\."]"".toRegex().containsMatchIn(string)) null
    else string

```

```

fun checkToInt(string: String) =
    if (""["\\D"]"".toRegex().containsMatchIn(string) ) null
    else string

```

```

val column1Weight = .1f
val column2Weight = .3f
val column3Weight = .3f
val column4Weight = .3f

```

```

@Composable
fun tableView() {

```

```
println("tableView")
```

```
var p1 by remember { mutableStateOf(ProgrammerRepository.data[0]) }  
var p2 by remember { mutableStateOf(ProgrammerRepository.data[1]) }  
var p3 by remember { mutableStateOf(ProgrammerRepository.data[2]) }  
var p4 by remember { mutableStateOf(ProgrammerRepository.data[3]) }  
var p5 by remember { mutableStateOf(ProgrammerRepository.data[4]) }  
var p6 by remember { mutableStateOf(ProgrammerRepository.data[5]) }
```

```
var i1 by remember { mutableStateOf(IncomeRepository.data[0]) }  
var i3 by remember { mutableStateOf(IncomeRepository.data[2]) }  
var i4 by remember { mutableStateOf(IncomeRepository.data[3]) }  
var i2 by remember { mutableStateOf(IncomeRepository.data[1]) }  
var i5 by remember { mutableStateOf(IncomeRepository.data[4]) }  
var i6 by remember { mutableStateOf(IncomeRepository.data[5]) }
```

```
var e1 by remember { mutableStateOf(ExpensesRepository.data[0]) }  
var e3 by remember { mutableStateOf(ExpensesRepository.data[2]) }  
var e4 by remember { mutableStateOf(ExpensesRepository.data[3]) }  
var e2 by remember { mutableStateOf(ExpensesRepository.data[1]) }  
var e5 by remember { mutableStateOf(ExpensesRepository.data[4]) }  
var e6 by remember { mutableStateOf(ExpensesRepository.data[5]) }
```

```
LazyColumn(Modifier.fillMaxSize(0.9f).padding(16.dp)) {  
    item {  
        Row(Modifier.background(Color.Gray)) {  
            TableCell(text = "№", weight = column1Weight)  
            TableCell(text = "Программисты", weight = column2Weight)  
            TableCell(text = "Траты на разработку", weight = column3Weight)  
            TableCell(text = "Ожидаемый доход", weight = column4Weight)  
        }  
    }  
  
    item {  
        Row(Modifier.fillMaxWidth()) {  
            TableCell(text = "1", weight = column1Weight)  
            BasicTextField(  
                value = p1.toString(),  
                onValueChange = {  
                    checkToInt(it)?.let {  
                        p1 = it.toInt()  
                        ProgrammerRepository.data[0] = it.toInt()  
                    }  
                },  
                Modifier  
                    .border(1.dp, Color.Black)  
                    .weight(column2Weight)  
                    .padding(8.dp)  
            )  
            BasicTextField(  
                value = e1.toString(),
```



```

        onChange = {
            checkToDouble(it)?.let {
                e1 = it.toDouble()
                ExpensesRepository.data[0] = it.toDouble()
            }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column3Weight)
            .padding(8.dp)
    )
    BasicTextField(
        value = i1.toString(),
        onChange = {
            checkToDouble(it)?.let {
                i1 = it.toDouble()
                IncomeRepository.data[0] = it.toDouble()
            }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column4Weight)
            .padding(8.dp)
    )
}
}
}

```

```

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "2", weight = column1Weight)
        BasicTextField(
            value = p2.toString(),
            onChange = {
                checkToInt(it)?.let {
                    p2 = it.toInt()
                    ProgrammerRepository.data[1] = it.toInt()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = e2.toString(),
            onChange = {
                checkToDouble(it)?.let {
                    e2 = it.toDouble()
                    ExpensesRepository.data[1] = it.toDouble()
                }
            }
        )
    }
}

```

```

    },
    Modifier
        .border(1.dp, Color.Black)
        .weight(column3Weight)
        .padding(8.dp)
    )
    BasicTextField(
        value = i2.toString(),
        onChange = {
            checkToDouble(it)?.let {
                i2 = it.toDouble()
                IncomeRepository.data[1] = it.toDouble()
            }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column4Weight)
            .padding(8.dp)
    )
}
}
item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "3", weight = column1Weight)
        BasicTextField(
            value = p3.toString(),
            onChange = {
                checkToInt(it)?.let {
                    p3 = it.toInt()
                    ProgrammerRepository.data[2] = it.toInt()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = e3.toString(),
            onChange = {
                checkToDouble(it)?.let {
                    e3 = it.toDouble()
                    ExpensesRepository.data[2] = it.toDouble()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
        BasicTextField(

```

```

        value = i3.toString(),
        onValueChange = {
            checkToDouble(it)?.let {
                i3 = it.toDouble()
                IncomeRepository.data[2] = it.toDouble()
            }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column4Weight)
            .padding(8.dp)
    )
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "4", weight = column1Weight)
        BasicTextField(
            value = p4.toString(),
            onValueChange = {
                checkToInt(it)?.let {
                    p4 = it.toInt()
                    ProgrammerRepository.data[3] = it.toInt()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = e4.toString(),
            onValueChange = {
                checkToDouble(it)?.let {
                    e4 = it.toDouble()
                    ExpensesRepository.data[3] = it.toDouble()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = i4.toString(),
            onValueChange = {
                checkToDouble(it)?.let {
                    i4 = it.toDouble()
                    IncomeRepository.data[3] = it.toDouble()
                }
            }
        )
    }
}

```

```

    },
    Modifier
        .border(1.dp, Color.Black)
        .weight(column4Weight)
        .padding(8.dp)
    )
}
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "5", weight = column1Weight)
        BasicTextField(
            value = p5.toString(),
            onChange = {
                checkToInt(it)?.let {
                    p5 = it.toInt()
                    ProgrammerRepository.data[4] = it.toInt()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = e5.toString(),
            onChange = {
                checkToDouble(it)?.let {
                    e5 = it.toDouble()
                    ExpensesRepository.data[4] = it.toDouble()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = i5.toString(),
            onChange = {
                checkToDouble(it)?.let {
                    i5 = it.toDouble()
                    IncomeRepository.data[4] = it.toDouble()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column4Weight)
                .padding(8.dp)
        )
    }
}

```

```

    }
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "6", weight = column1Weight)
        BasicTextField(
            value = p6.toString(),
            onChange = {
                checkToInt(it)?.let {
                    p6 = it.toInt()
                    ProgrammerRepository.data[5] = it.toInt()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = e6.toString(),
            onChange = {
                checkToDouble(it)?.let {
                    e6 = it.toDouble()
                    ExpensesRepository.data[5] = it.toDouble()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = i6.toString(),
            onChange = {
                checkToDouble(it)?.let {
                    i6 = it.toDouble()
                    IncomeRepository.data[5] = it.toDouble()
                }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column4Weight)
                .padding(8.dp)
        )
    }
}

}
}

```

```

@Composable
fun RowScope.TableCell(
    text: String,
    weight: Float
){
    Text(
        text = text,
        Modifier
            .border(1.dp, Color.Black)
            .weight(weight)
            .padding(8.dp)
    )
}

```

```

// @filename \src\main\kotlin\Main.kt
import Data.ExcelReader
import Data.UpdateExcel
import androidx.compose.ui.window.Window
import androidx.compose.ui.window.application

```

```

fun main() = application {

    //Загрузка при запуске
    ExcelReader.read()

    Window(onCloseRequest = ::exitApplication) {
        App()
    }
}

```

```

// @filename \src\main\kotlin\Data\ExcelReader.kt
package Data

```

```

import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.*
import org.apache.poi.ss.util.NumberToTextConverter;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import repository.ProgrammerRepository
import repository.ExpensesRepository
import repository.IncomeRepository

```

```

import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.HashMap;

```

```

object ExcelReader {
    // Чтение файла
    fun read(filename: String = "calc.xlsx") {

```

```

val workbook = loadWorkbook(filename)
val sheetIterator = workbook?.sheetIterator()
if (sheetIterator != null) {
    while (sheetIterator.hasNext()) {
        val sheet: Sheet = sheetIterator.next()
        processSheet(sheet)
        println()
    }
}
else
    println("sheetIterator is null")
}

// Загрузка файла
private fun loadWorkbook(filename: String): Workbook? {
    val extension = filename.substring(filename.lastIndexOf(".") + 1)
    val file = FileInputStream(File(filename))
    return when (extension) {
        "xls" -> HSSFWorkbook(file)
        "xlsx" -> XSSFWorkbook(file)
        else -> {
            println("Unknown Excel file extension: $extension")
            null
        }
    }
}

```

```

// Обработка строки
private fun processSheet(sheet: Sheet) {
    println("Sheet: " + sheet.getSheetName())
    val data = HashMap<Int, MutableList<Any>>()
    val iterator = sheet.rowIterator()
    var rowIndex = 0
    while (iterator.hasNext()) {
        val row = iterator.next()
        processRow(data, rowIndex, row)
        rowIndex++
    }
}

```

```

ProgrammerRepository.data.clear()
ExpensesRepository.data.clear()
IncomeRepository.data.clear()

```

```

for ((k,v) in data)
    when ( k ) {
        1 -> for (i in 1..6) ProgrammerRepository.data.add(v[i]
            .toString().toInt())
        2 -> for (i in 1..6) ExpensesRepository.data.add(v[i]
            .toString().replace(',', '.').toDouble())
        3 -> for (i in 1..6) IncomeRepository.data.add(v[i]
            .toString().replace(',', '.').toDouble())
    }

```

```

        else -> {}
    }

    print("Sheet data:")
    println(data)

    println(ExpensesRepository.data.toString())
    println(ProgrammerRepository.data.toString())
    println(IncomeRepository.data.toString())
}

// Обработка строки
private fun processRow(data: HashMap<Int, MutableList<Any>>, rowIndex: Int, row:
Row) {
    data[rowIndex] = ArrayList()
    for (cell in row) {
        processCell(cell, data[rowIndex]!!)
    }
}

// Обработка ячейки
private fun processCell(cell: Cell, dataRow: MutableList<Any>) {
    when (cell.cellType) {
        CellType.STRING -> dataRow.add(cell.stringCellValue)
        CellType.NUMERIC -> if (DateUtil.isCellDateFormatted(cell)) {
            dataRow.add(cell.localDateTimeCellValue)
        } else {
            dataRow.add(NumberToTextConverter.toText(cell.numericCellValue))
        }
        CellType.BOOLEAN -> dataRow.add(cell.booleanCellValue)
        CellType.FORMULA -> dataRow.add(cell.cellFormula)
        else -> dataRow.add(" ")
    }
}

// @filename \src\main\kotlin\Data\UpdateExcel.kt
package Data

import org.apache.poi.ss.usermodel.Row
import org.apache.poi.xssf.usermodel.XSSFWorkbook
import repository.ProgrammerRepository
import repository.ExpensesRepository
import repository.IncomeRepository
import java.io.File
import java.io.FileInputStream
import java.io.FileOutputStream
import java.io.IOException
import java.util.*

object UpdateExcel {

```



```

fun changeCalls() {

    val file = FileInputStream(File("calc.xlsx"))
    val workbook = XSSFWorkbook(file)
    val sheet = workbook.getSheetAt(0)

    val programmer: Row = sheet.getRow(1)
    for (i in 0 until ProgrammerRepository.data.size)
        programmer.getCell(1+i).setCellValue(ProgrammerRepository.data[i].toDouble())

    val expenses: Row = sheet.getRow(2)
    for (i in 0 until ExpensesRepository.data.size)
        expenses.getCell(1+i).setCellValue(ExpensesRepository.data[i])

    val income: Row = sheet.getRow(3)
    for (i in 0 until IncomeRepository.data.size)
        income.getCell(1+i).setCellValue(IncomeRepository.data[i])

    try {
        val out = FileOutputStream(File("calc.xlsx"))
        workbook.write(out)
        out.close()
        println("Значения успешно изменены")
    } catch (e: IOException) {
        e.printStackTrace()
    }
}

val checkedInputDigit: Double
    get() {
        val digit = Scanner(System.`in`).nextDouble()
        if (digit < 0) {
            println("Значения не могут быть отрицательными. Повторите попытку! : ")
            checkedInputDigit
        }
        return digit
    }
}

// @filename \src\main\kotlin\repository\ExpensesRepository.kt
package repository

// Затраты
object ExpensesRepository {
    val data = mutableListOf<Double>()
}

// @filename \src\main\kotlin\repository\IncomeRepository.kt
package repository

```

```

// Доходы
object IncomeRepository {
    val data = mutableListOf<Double>()
}

// @filename \src\main\kotlin\repository\ProgrammerRepository.kt
package repository

object ProgrammerRepository {
    val data = mutableListOf<Int>()
}

// @filename \src\main\kotlin\screen\lineView.kt
package screen

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.*
import repository.ProgrammerRepository

@Composable
fun lineView() {
    println("lineView")
    var viewLine by remember { mutableStateOf(0) }

    /*
    var name by remember { mutableStateOf(ProgrammerRepository.data[viewLine].name)
    }
    var count by remember { mutableStateOf(ProgrammerRepository.data[viewLine].count)
    }
    var tp1 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp1) }
    var tp2 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp2) }
    var tp3 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp3) }
    var tp4 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp4) }
    var tp5 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp5) }
    var tp6 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp6) }
    var tp7 by remember { mutableStateOf(ProgrammerRepository.data[viewLine].tp7) }
    var cost by remember { mutableStateOf(ProgrammerRepository.data[viewLine].cost) }

    Column {
        Row {
            for (i in 1..5) {
                Button(
                    onClick = {
                        println("onClick $i")
                        viewLine = i-1
                        name = ProgrammerRepository.data[viewLine].name
                    }
                )
            }
        }
    }
    */
}

```

```

        count = ProgrammerRepository.data[viewLine].count
        tp1 = ProgrammerRepository.data[viewLine].tp1
        tp2 = ProgrammerRepository.data[viewLine].tp2
        tp3 = ProgrammerRepository.data[viewLine].tp3
        tp4 = ProgrammerRepository.data[viewLine].tp4
        tp5 = ProgrammerRepository.data[viewLine].tp5
        tp6 = ProgrammerRepository.data[viewLine].tp6
        tp7 = ProgrammerRepository.data[viewLine].tp7
        cost = ProgrammerRepository.data[viewLine].cost
    },
    content = {
        Text("Линия $i")
    }
}
}
}
}

```

```

Column {
    TextField(
        label = { Text("Название") },
        value = name,
        onValueChange = {
            name = it
            ProgrammerRepository.data[viewLine].name = it
        },
    )
    TextField(
        label = { Text("Количество линий") },
        value = count.toString(),
        onValueChange = {
            count = it.toInt()
            ProgrammerRepository.data[viewLine].count = it.toInt()
        }
    )
    TextField(
        label = { Text("Время производства продукта 1") },
        value = tp1.toString(),
        onValueChange = {
            tp1 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp1 = it.toDouble()
        }
    )
    TextField(
        label = { Text("Время производства продукта 2") },
        value = tp2.toString(),
        onValueChange = {
            tp2 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp2 = it.toDouble()
        }
    )
    TextField(

```

```

        label = { Text("Время производства продукта 3") },
        value = tp3.toString(),
        onChange = {
            tp3 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp3 = it.toDouble() }
    )
    TextField(
        label = { Text("Время производства продукта 4") },
        value = tp4.toString(),
        onChange = {
            tp4 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp4 = it.toDouble() }
    )
    TextField(
        label = { Text("Время производства продукта 5") },
        value = tp5.toString(),
        onChange = {
            tp5 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp5 = it.toDouble() }
    )
    TextField(
        label = { Text("Время производства продукта 6") },
        value = tp6.toString(),
        onChange = {
            tp6 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp6 = it.toDouble() }
    )
    TextField(
        label = { Text("Время производства продукта 7") },
        value = tp7.toString(),
        onChange = { tp7 = it.toDouble()
            ProgrammerRepository.data[viewLine].tp7 = it.toDouble() }
    )
    TextField(
        label = { Text("Цена") },
        value = cost.toString(),
        onChange = {
            cost = it.toDouble()
            ProgrammerRepository.data[viewLine].cost = it.toDouble() }
    )
    }
}*/
}

```

```

// @filename \src\main\kotlin\screen\LineViewState.kt
package screen

```

```

import androidx.compose.runtime.MutableState

```

```

data class LineViewState{

```

```

        val name : MutableState<String>,
        val count: MutableState<Int>,
        val tp: List<MutableState<Double>>,
        val cost : MutableState<Double>
    )
}

// @filename \src\main\kotlin\screen\productView.kt
package screen

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.text.BasicTextField
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import repository.ExpensesRepository

fun checkToDouble(string: String) =
    if ("^[^\\d\\.]"!!.toRegex().containsMatchIn(string)) null
    else string

fun checkToInt(string: String) =
    if ("^[^\\D]"!!.toRegex().containsMatchIn(string) ) null
    else string
/*
@Composable
fun productView() {
    println("productView")

    var l1 by remember { mutableStateOf(ExpensesRepository.data[0].limit) }
    var l2 by remember { mutableStateOf(ExpensesRepository.data[1].limit) }
    var l3 by remember { mutableStateOf(ExpensesRepository.data[2].limit) }
    var l4 by remember { mutableStateOf(ExpensesRepository.data[3].limit) }
    var l5 by remember { mutableStateOf(ExpensesRepository.data[4].limit) }
    var l6 by remember { mutableStateOf(ExpensesRepository.data[5].limit) }
    var l7 by remember { mutableStateOf(ExpensesRepository.data[6].limit) }

    var c1 by remember { mutableStateOf(ExpensesRepository.data[0].cost) }
    var c2 by remember { mutableStateOf(ExpensesRepository.data[1].cost) }
    var c3 by remember { mutableStateOf(ExpensesRepository.data[2].cost) }
    var c4 by remember { mutableStateOf(ExpensesRepository.data[3].cost) }
    var c5 by remember { mutableStateOf(ExpensesRepository.data[4].cost) }

```

```

var c6 by remember { mutableStateOf(ExpensesRepository.data[5].cost) }
var c7 by remember { mutableStateOf(ExpensesRepository.data[6].cost) }

val tableData = (1..7).mapIndexed { index, item ->
    index to "Item $index"
}
val column1Weight = .3f
val column2Weight = .3f
val column3Weight = .4f
LazyColumn(Modifier.fillMaxSize().padding(16.dp)) {
    // Here is the header
    item {
        Row(Modifier.background(Color.Gray)) {
            TableCell(text = "Продукт", weight = column1Weight)
            TableCell(text = "Ограничение", weight = column2Weight)
            TableCell(text = "Стоимость", weight = column3Weight)
        }
    }
    // Here are all the lines of your table.
    item {
        Row(Modifier.fillMaxWidth()) {
            TableCell(text = "1", weight = column1Weight)
            BasicTextField(
                value = l1.toString(),
                onChange = {
                    checkToInt(it)?.let { l1 = it.toInt() }
                    checkToInt(it)?.let { ExpensesRepository.data[0].limit = it.toInt() }
                },
                Modifier
                    .border(1.dp, Color.Black)
                    .weight(column2Weight)
                    .padding(8.dp)
            )
            BasicTextField(
                value = c1.toString(),
                onChange = {
                    checkToInt(it)?.let { c1 = it.toInt() }
                    checkToInt(it)?.let { ExpensesRepository.data[0].cost = it.toInt() }
                },
                Modifier
                    .border(1.dp, Color.Black)
                    .weight(column3Weight)
                    .padding(8.dp)
            )
        }
    }
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "2", weight = column1Weight)
        BasicTextField(

```

```

        value = l2.toString(),
        onChange = {
            checkToInt(it)?.let { l2 = it.toInt() }
            checkToInt(it)?.let { ExpensesRepository.data[1].limit = it.toInt() }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column2Weight)
            .padding(8.dp)
    )
    BasicTextField(
        value = c2.toString(),
        onChange = {
            checkToInt(it)?.let { c2 = it.toInt() }
            checkToInt(it)?.let { ExpensesRepository.data[1].cost = it.toInt() }
        },
        Modifier
            .border(1.dp, Color.Black)
            .weight(column3Weight)
            .padding(8.dp)
    )
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "3", weight = column1Weight)
        BasicTextField(
            value = l3.toString(),
            onChange = {
                checkToInt(it)?.let { l3 = it.toInt() }
                checkToInt(it)?.let { ExpensesRepository.data[2].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c3.toString(),
            onChange = {
                checkToInt(it)?.let { c3 = it.toInt() }
                checkToInt(it)?.let { ExpensesRepository.data[2].cost = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
    }
}

```

```

item {
  Row(Modifier.fillMaxWidth()) {
    TableCell(text = "4", weight = column1Weight)
    BasicTextField(
      value = l4.toString(),
      onChange = {
        checkToInt(it)?.let { l4 = it.toInt() }
        checkToInt(it)?.let { ExpensesRepository.data[3].limit = it.toInt() }
      },
      Modifier
        .border(1.dp, Color.Black)
        .weight(column2Weight)
        .padding(8.dp)
    )
    BasicTextField(
      value = c4.toString(),
      onChange = {
        checkToInt(it)?.let { c4 = it.toInt() }
        checkToInt(it)?.let { ExpensesRepository.data[3].cost = it.toInt() }
      },
      Modifier
        .border(1.dp, Color.Black)
        .weight(column3Weight)
        .padding(8.dp)
    )
  }
}
item {
  Row(Modifier.fillMaxWidth()) {
    TableCell(text = "5", weight = column1Weight)
    BasicTextField(
      value = l5.toString(),
      onChange = {
        checkToInt(it)?.let { l5 = it.toInt() }
        checkToInt(it)?.let { ExpensesRepository.data[4].limit = it.toInt() }
      },
      Modifier
        .border(1.dp, Color.Black)
        .weight(column2Weight)
        .padding(8.dp)
    )
    BasicTextField(
      value = c5.toString(),
      onChange = {
        checkToInt(it)?.let { c5 = it.toInt() }
        checkToInt(it)?.let { ExpensesRepository.data[4].cost = it.toInt() }
      },
      Modifier
        .border(1.dp, Color.Black)
        .weight(column3Weight)

```



```

        .padding(8.dp)
    )
}
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "6", weight = column1Weight)
        BasicTextField(
            value = l6.toString(),
            onChange = {
                checkToInt(it)?.let { l6 = it.toInt() }
                checkToInt(it)?.let { ExpensesRepository.data[5].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c6.toString(),
            onChange = {
                checkToInt(it)?.let { c6 = it.toInt() }
                checkToInt(it)?.let { ExpensesRepository.data[5].cost = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column3Weight)
                .padding(8.dp)
        )
    }
}

item {
    Row(Modifier.fillMaxWidth()) {
        TableCell(text = "7", weight = column1Weight)
        BasicTextField(
            value = l7.toString(),
            onChange = {
                checkToInt(it)?.let { l7 = it.toInt() }
                checkToInt(it)?.let { ExpensesRepository.data[6].limit = it.toInt() }
            },
            Modifier
                .border(1.dp, Color.Black)
                .weight(column2Weight)
                .padding(8.dp)
        )
        BasicTextField(
            value = c7.toString(),
            onChange = {
                checkToInt(it)?.let { c7 = it.toInt() }
            }
        )
    }
}

```

```

        checkToInt(it)?.let { ExpensesRepository.data[6].cost = it.toInt() }
    },
    Modifier
        .border(1.dp, Color.Black)
        .weight(column3Weight)
        .padding(8.dp)
    )
}
}
}
}
}

```

```

@Composable
fun RowScope.TableCell(
    text: String,
    weight: Float
){
    Text(
        text = text,
        Modifier
            .border(1.dp, Color.Black)
            .weight(weight)
            .padding(8.dp)
    )
}
*/

```