|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» | |
|  | Институт компьютерных наук и технологий |
| ОТЧЕТ  по лабораторной работе **№6**  по дисциплине Языки программирования  «**ООП**» | |
|  | Работу выполнила студентка  Группы ПМИ 1-2, курс 2  Института компьютерных наук и технологий  **Арланова Анна Андреевна** |
|  | Работу принял:  ассистент кафедры Информационных технологий  Ракина В.Д. |
| Пермь, 2023 | |

Оглавление

[1 Постановка задачи 3](#_Toc128521975)

[2 Проектирование 4](#_Toc128521976)

[3 Тестирование 6](#_Toc128521977)

[4 Приложение 13](#_Toc128521978)

# Постановка задачи

1. Бинарные файлы, содержащие числовые данные (исходный файл заполнить случайными данными, заполнение организовать отдельным методом). Вычислить произведение нечетных отрицательных компонент файла.
2. Бинарные файлы, содержащие числовые данные (исходный файл заполнить случайными данными, заполнение организовать отдельным методом. Скопировать элементы заданного файла в квадратную матрицу размером n×n (если элементов файла недостает, заполнить оставшиеся матрицы нулями). Поменять местами в каждом столбце минимальный и максимальный элементы.
3. Бинарные файлы, содержащие величины типа struct (заполнение исходного файла организовать отдельным методом). Информация о багаже пассажира описывается массивом, где каждый элемент содержит название единицы багажа (чемодан, сумка, коробка и т.д.) и ее массу. Дан файл, содержащий сведения о багаже нескольких пассажиров. Найти число пассажиров, имеющих более двух единиц багажа и число пассажиров, количество единиц багажа которых превосходит среднее число единиц багажа.
4. Решить задачу с использованием структуры «текстовый файл» (в файле хранятся целые числа по одному в строке). В файле найти сумму квадратов элементов.
5. Решить задачу с использованием структуры «текстовый файл» (в файле хранятся целые числа по несколько в строке). Вычислить произведение элементов.
6. Решить задачу с использованием структуры «текстовый файл» (в файле хранится текст). Переписать в другой файл строки, имеющие заданную длину m.

# Проектирование

Код для лабораторной работы я разделила на три файла:

* Program.fs – менюшка с рекурсивным вызовом функций из следующих двух файлов
* Task1.fs – решение задачи 1
* Task2.fs – решение задачи 2

## Добавить к каждой строке заданный символ в начало

Тип данных Tree:

Описание: определяет новый тип данных для бинарного дерева.

Принцип работы: есть два определение для узлов дерева. Первое представляет собой узел, состоящий из трех элементов: значение, первое дерево и второе дерево. Деревья здесь также являются бинарными деревьями, которые описывают в этом пункте. Второе определение узла дерева – пустое, нулевое значение. Необходимо чтобы обозначить где заканчивается дерево.

Функция mapTree:

Описание: создает новое дерево на основе старого.

Принцип работы: с пустыми узлами дерева мы ничего не делаем. С узлами, имеющие данные, мы применяем для значения узла функцию, которую была задана при вызове функции. Поддеревья проходятся рекурсивно.

Функция addSymbol:

Описание: возвращает строку с добавленным к ней символом.

Принцип работы: мы применяем функцию sprintf, которая возвращает отформатированную строку, как если использовали обычный printf.

Функция readChar:

Описание: ввод корректного символа

Принцип работы: функция запрашивает у пользователя ввести символ, далее обрабатывается на то, чтобы это не была пустая строка и чтобы длина этой строки составляла ровно 1 символ. В случаях, когда эти условия не соблюдаются, выводится сообщение об ошибке и при помощи рекурсии функция просит снова ввести символ.

Функция readInteger:

Описание: ввод не отрицательного числа

Принцип работы: функция запрашивает у пользователя ввести число. Далее фукнция пытается преобразовать строку в число и проверяет, чтобы оно было больше 0. В случае успеха функция возвращает число, в ином случае выводит сообщение об ошибке и при помощи рекурсии функция снова запрашивает число.

Функция generateBinaryTree:

Описание: генерирует строковое бинарное дерево, где узлы являются рандомными целыми числами

Принцип работы: Если глубина меньше или равна 0, функция возвращает пустой узел (Nil). В противном случае: генерируется случайное значение и преобразуется в строку, далее рекурсивно вызывается функция для левого поддерева с уменьшенной глубиной, аналогично и для правого поддерева. Затем создается новый узел со сгенерированным значением, левым и правым поддеревом.

Функция addSpaces:

Описание: выводит пробелы до заданного уровня, вспомогательная функция для вывода дерева

Принцип работы: С использованием цикла for, выводится в консоль несколько пробелов.

Функция printTree:

Описание: выводит отформатированное дерево справа налево

Принцип работы: В случае, если текущий узел - это узел с данными, то рекурсивно вызывается printTree для правого поддерева с увеличением уровня на 1. Затем вызывается функция addSpace для добавления отступа на текущем уровне. Выводится значение узла, при этом если текущий узел не находится на 0 уровне, то выводится также для аккуратного вывода строка: “ |--- ”. Затем рекурсивно вызывается printTree для левого поддерева с увеличением уровня на 1. В случае, если текущий узел - это пустой узел, то просто вызывается функция addSpace для добавления отступа на текущем уровне. Если обобщать, выходит так, что обходится дерево справа, выводится строка с данными узла, а затем рекурсивно вызывается для левого поддерева. Когда достигается конец ветви, функция возвращает управление на уровень выше, и там продолжается обход левой части родительского узла.

Функция Output:

Описание: Вывод решения задачи

## Что больше: сумма четных чисел или сумма нечетных чисел в дереве

Тип данных Tree:

Описание: определяет новый тип данных для бинарного дерева.

Принцип работы: есть два определение для узлов дерева. Первое представляет собой узел, состоящий из трех элементов: значение, первое дерево и второе дерево. Деревья здесь также являются бинарными деревьями, которые описывают в этом пункте. Второе определение узла дерева – пустое, нулевое значение. Необходимо чтобы обозначить где заканчивается дерево.

Функция generateBinaryTree:

Описание: генерирует целочисленное бинарное дерево, где узлы являются рандомными целыми числами

Принцип работы: Если глубина меньше или равна 0, функция возвращает пустой узел (Nil). В противном случае: генерируется случайное значение, далее рекурсивно вызывается функция для левого поддерева с уменьшенной глубиной, аналогично и для правого поддерева. Затем создается новый узел со сгенерированным значением, левым и правым поддеревом.

Функция readInteger:

Описание: ввод не отрицательного числа

Принцип работы: функция запрашивает у пользователя ввести число. Далее фукнция пытается преобразовать строку в число и проверяет, чтобы оно было больше 0. В случае успеха функция возвращает число, в ином случае выводит сообщение об ошибке и при помощи рекурсии функция снова запрашивает число.

Функция addSpaces:

Описание: выводит пробелы до заданного уровня, вспомогательная функция для вывода дерева

Принцип работы: С использованием цикла for, выводится в консоль несколько пробелов.

Функция printTree:

Описание: выводит отформатированное дерево справа налево

Принцип работы: В случае, если текущий узел - это узел с данными, то рекурсивно вызывается printTree для правого поддерева с увеличением уровня на 1. Затем вызывается функция addSpace для добавления отступа на текущем уровне. Выводится значение узла, при этом если текущий узел не находится на 0 уровне, то выводится также для аккуратного вывода строка: “ |--- ”. Затем рекурсивно вызывается printTree для левого поддерева с увеличением уровня на 1. В случае, если текущий узел - это пустой узел, то просто вызывается функция addSpace для добавления отступа на текущем уровне. Если обобщать, выходит так, что обходится дерево справа, выводится строка с данными узла, а затем рекурсивно вызывается для левого поддерева. Когда достигается конец ветви, функция возвращает управление на уровень выше, и там продолжается обход левой части родительского узла.

Функции sumEvenNumbers и sumOddNumbers:

Описание: рекурсивно подсчитывают сумму четных/нечетных чисел

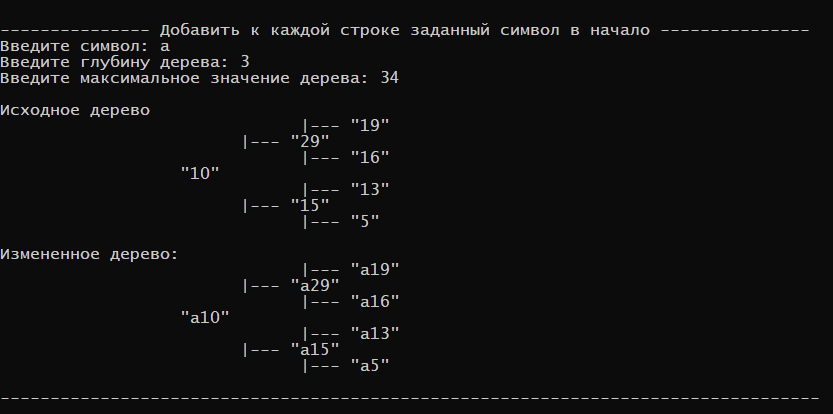
Принцип работы: рекурсивно обходит дерево и суммирует значения только в тех узлах, где значение является четным/нечетным числом. Если значение узла четное/нечетное, оно добавляется к сумме значений левого и правого поддеревьев; в противном случае, суммируются только значения левого и правого поддеревьев.

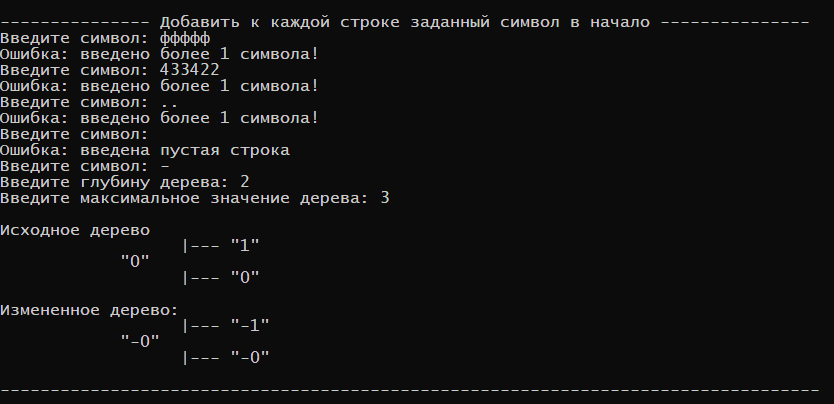
Функция Output:

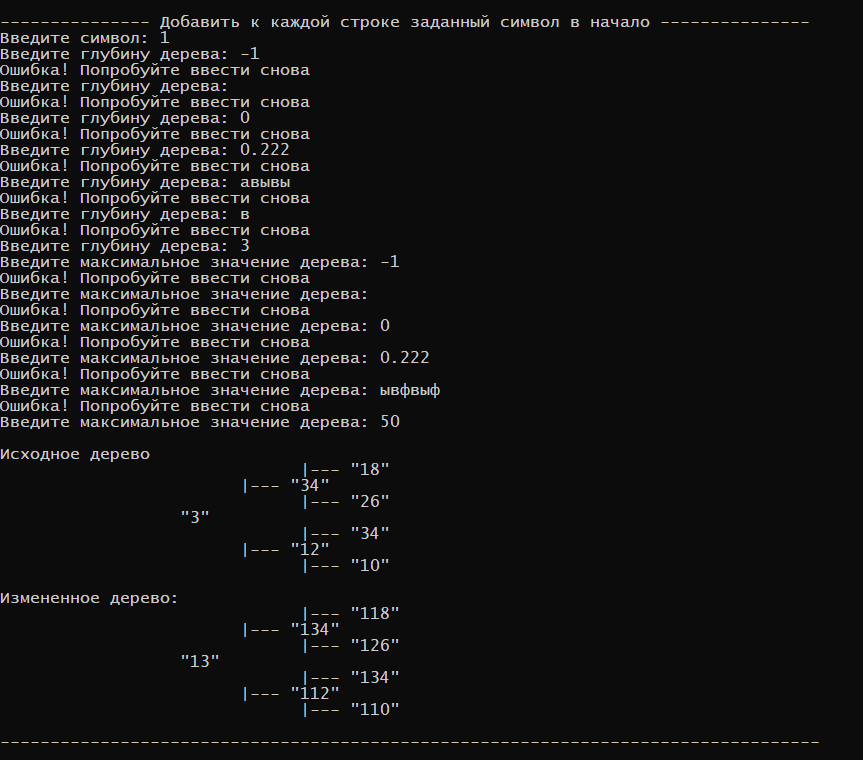
Описание: Вывод решения задачи и сравнение чья сумма больше четных или нечетных чисел.

# Тестирование

## Добавить к каждой строке заданный символ в начало

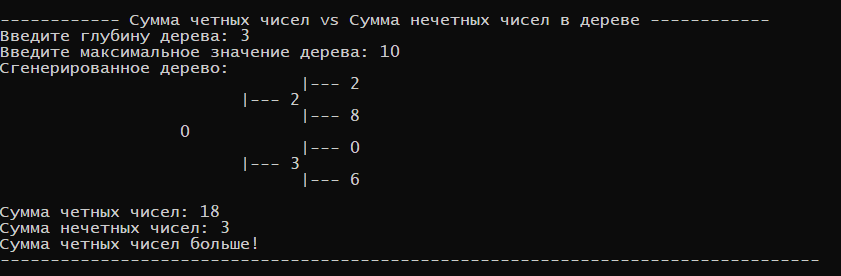
1ый тест

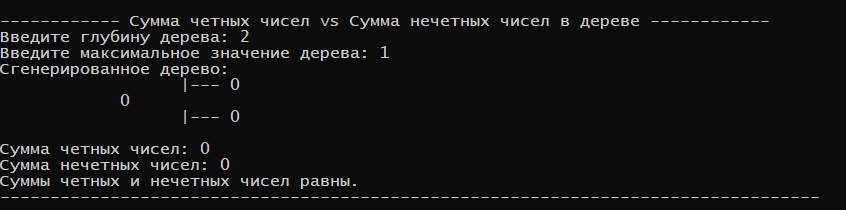
2ой тест

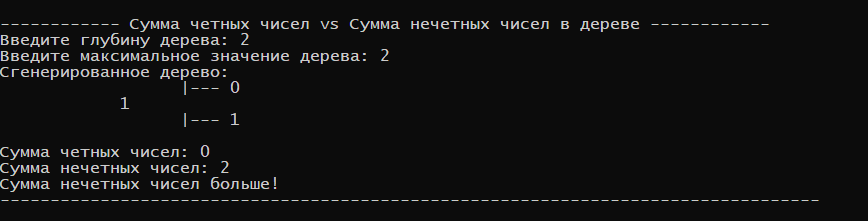
3ий тест

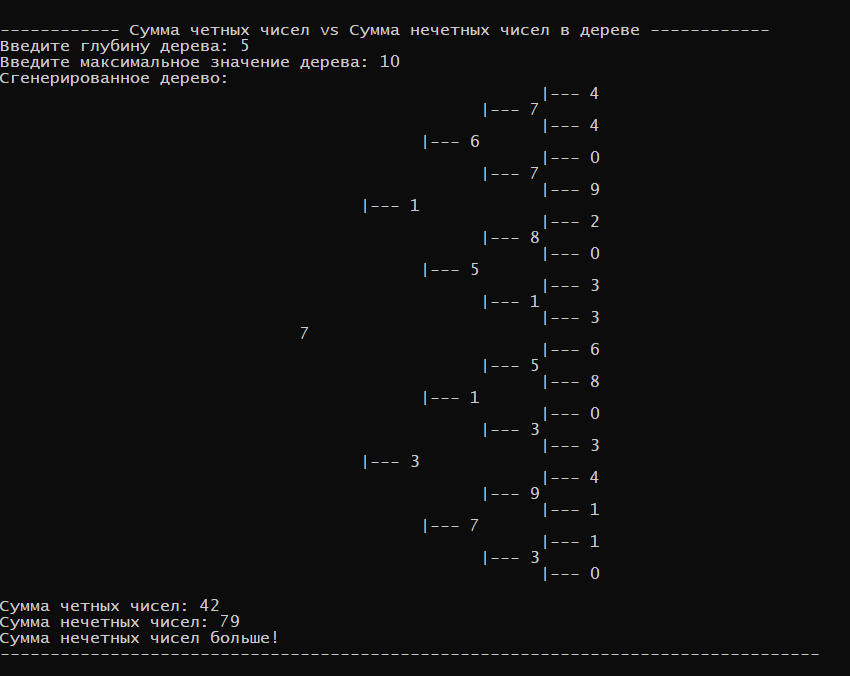
4ый тест

## Что больше: сумма четных чисел или сумма нечетных чисел в дереве

1ый тест

2ой тест

3ий тест

4ый тест

# Приложение

Program.fs

1. open System
2. printfn "----------------------------- Лабораторная работа №4 -----------------------------"
3. printfn "1: Первая задача (Добавить к каждой строке элемента дерева заданный символ в начало)"
4. printfn "2: Вторая задача (Выяснить, что больше: сумма четных чисел или сумма нечетных чисел в дереве)"
5. printfn "Enter: Выход"
6. printfn "----------------------------------------------------------------------------------"
7. let rec userInputLoop () =
8. printf "\n\nВведите номер задачи: "
9. let userInput = Console.ReadLine()
10. printfn ""
11. if userInput = "" then
12. ()
13. else
14. match userInput with
15. | "1" -> Task1.Output ()
16. | "2" -> Task2.Output ()
17. | \_ -> printfn "Не сопоставлено"
18. userInputLoop ()
19. userInputLoop ()

Task1.fs

1. // Дерево содержит строки. Добавить к каждой строке заданный символ в начало
2. module Task1
3. open System
4. // Структура бинарного дерева
5. type Tree<'t> =
6. | Node of 't \* Tree<'t> \* Tree<'t>
7. | Nil
8. // Функция для добавления символа к каждой строке в дереве
9. let rec mapTree f tree =
10. match tree with
11. | Nil -> Nil
12. | Node (data, left, right) -> Node (f data, mapTree f left, mapTree f right)
13. // Функция, добавляющая символ к строке
14. let addSymbol ch str =
15. sprintf "%c%s" ch str
16. // Функция для ввода символа
17. let rec readChar () : char =
18. printf "Введите символ: "
19. let input = Console.ReadLine()
20. match input with
21. | "" ->
22. printfn "Ошибка: введена пустая строка"
23. readChar ()
24. | str ->
25. match str.Length with
26. | 1 -> str.[0]
27. | \_ ->
28. printfn "Ошибка: введено более 1 символа!"
29. readChar ()
30. // Генератор случайного бинарного дерева
31. let rec generateBinaryTree depth maxValue : Tree<string> =
32. let rand = Random()
33. if depth <= 0 then
34. Nil
35. else
36. let value = string(rand.Next(maxValue))
37. let leftSubtree = generateBinaryTree (depth - 1) maxValue
38. let rightSubtree = generateBinaryTree (depth - 1) maxValue
39. Node(value, leftSubtree, rightSubtree)
40. // Ввод не отрицательного числа
41. let rec readInteger strMessage : int =
42. printf "%s" strMessage
43. let input = Console.ReadLine()
44. match Int32.TryParse(input) with
45. | true, \_ when int(input) > 0 -> int(input)
46. | \_ ->
47. printfn "Ошибка! Попробуйте ввести снова"
48. readInteger strMessage
49. // Добавление пробелов до уровня
50. let addSpace lvl =
51. for \_ in 1..lvl do
52. printf " "
53. // Функции отформатированного вывода дерева
54. let rec printTree tree lvl =
55. match tree with
56. | Node (data, left, right) ->
57. printTree right (lvl + 1)
58. addSpace lvl
59. printfn "%s%A" (if lvl = 0 then "" else "|--- ") data
60. printTree left (lvl + 1)
61. | Nil ->
62. addSpace lvl
63. // Вывод основной программы
64. let Output () = (
65. printf "\n--------------- Добавить к каждой строке заданный символ в начало ---------------\n"
67. // Чтение символа на добавление
68. let symbolToAdd = readChar ()
69. // Задача параметров генерируемого дерева
70. let depth = readInteger "Введите глубину дерева: "
71. let maxValue = readInteger "Введите максимальное значение дерева: "
73. // Генерация нового дерева
74. let originalTree = generateBinaryTree depth maxValue
75. // Применение функции к дереву
76. let modifiedTree = mapTree (addSymbol symbolToAdd) originalTree
77. // Вывод исходного дерева
78. printfn "\nИсходное дерево"
79. printTree originalTree 0
80. // Вывод измененного дерева
81. printfn "\nИзмененное дерево:"
82. printTree modifiedTree 0
83. printf "\n----------------------------------------------------------------------------------")

Task2.fs

1. // Выяснить, что больше: сумма четных чисел или сумма нечетных чисел в дереве
2. module Task2
3. open System
4. // Структура бинарного дерева
5. type Tree<'t> =
6. | Node of 't \* Tree<'t> \* Tree<'t>
7. | Nil
8. // Генератор случайного бинарного дерева
9. let rec generateBinaryTree depth maxValue =
10. let rand = Random()
11. if depth <= 0 then
12. Nil
13. else
14. let value = rand.Next(maxValue)
15. let leftSubtree = generateBinaryTree (depth - 1) maxValue
16. let rightSubtree = generateBinaryTree (depth - 1) maxValue
17. Node(value, leftSubtree, rightSubtree)
18. // Ввод не отрицательного числа
19. let rec readInteger strMessage : int =
20. printf "%s" strMessage
21. let input = Console.ReadLine()
22. match Int32.TryParse(input) with
23. | true, \_ when int(input) > 0 -> int(input)
24. | \_ ->
25. printfn "Ошибка! Попробуйте ввести снова"
26. readInteger strMessage
27. // Добавление пробелов до уровня
28. let addSpace lvl =
29. for \_ in 1..lvl do
30. printf " "
31. // Функции отформатированного вывода дерева
32. let rec printTree tree lvl =
33. match tree with
34. | Node (data, left, right) ->
35. printTree right (lvl + 1)
36. addSpace lvl
37. printfn "%s%A" (if lvl = 0 then "" else "|--- ") data
38. printTree left (lvl + 1)
39. | Nil ->
40. addSpace lvl
41. // Функция для подсчета суммы четных чисел в дереве
42. let rec sumEvenNumbers tree =
43. match tree with
44. | Node (data, left, right) ->
45. if data % 2 = 0 then
46. data + sumEvenNumbers left + sumEvenNumbers right
47. else
48. sumEvenNumbers left + sumEvenNumbers right
49. | Nil -> 0
50. // Функция для подсчета суммы нечетных чисел в дереве
51. let rec sumOddNumbers tree =
52. match tree with
53. | Node (data, left, right) ->
54. if data % 2 <> 0 then
55. data + sumOddNumbers left + sumOddNumbers right
56. else
57. sumOddNumbers left + sumOddNumbers right
58. | Nil -> 0
59. let Output () = (
60. printf "\n------------ Сумма четных чисел vs Сумма нечетных чисел в дереве ------------\n"
61. // Задача параметров генерируемого дерева
62. let depth = readInteger "Введите глубину дерева: "
63. let maxValue = readInteger "Введите максимальное значение дерева: "
65. // Генерация нового дерева
66. let tree = generateBinaryTree depth maxValue
67. printfn "Сгенерированное дерево:"
68. printTree tree 0
69. // Вычисление и вывод сумм
70. let evenSum = sumEvenNumbers tree
71. let oddSum = sumOddNumbers tree
72. printfn "\nСумма четных чисел: %d" evenSum
73. printfn "Сумма нечетных чисел: %d" oddSum
74. // Сравнение сумм и вывод результата
75. if evenSum > oddSum then
76. printfn "Сумма четных чисел больше!"
77. elif evenSum < oddSum then
78. printfn "Сумма нечетных чисел больше!"
79. else
80. printfn "Суммы четных и нечетных чисел равны."
81. printf "----------------------------------------------------------------------------------"
82. )