Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования
«Иркутский государственный университет»
(ФГБОУ ВО «ИГУ»)

Институт математики и информационных технологий

Кафедра математического анализа и дифференциальных уравнений

КУРСОВАЯ РАБОТА

На тему «Разработка приложения на языке функционального программирования»

Студента 3 курса очного отделения
группы 2321-ДБ
Сухоручкина Егора Олеговича

Руководитель:
к. ф.-м. н., доцент
_____Черкашин Е.А.

Иркутск – 2022

СОДЕРЖАНИЕ

Введение

Функциональное программирование — это стиль написания программ посредством компиляции набора функций. Его основной принцип заключается в том, чтобы обернуть почти все в функцию, написать множество небольших многократно используемых функций, а затем просто вызывать их одну за другой. Кроме того, структура этих функций должна соответствовать определенным правилам и решать некоторые проблемы.

Использование функциональных языков может значительно повысить производительность и качество работы программистов. Естественно, это зависит от сочетания задач, языка и навыков программирования. В этом случае программист просто описывает, что он хочет, вместо перечисления последовательности действий, необходимых для получения результата. Таким образом, разработчик сосредотачивается на высокоуровневом «что требуется» вместо того, чтобы застрять на низкоуровневом «как это сделать».

Цель работы: создание приложения, позволяющего воспроизводить звуки, для проверки динамиков на языке F# (F Sharp) функционального программирования.

Задачи работы:

1. Изучение подхода функционального программирования;
2. Создание программы проверки звука;
3. Создание интерфейса программы;
4. Реализация воспроизведения звука.

1. Функциональное программирование на языке F#

F# — мультипарадигмальный язык программирования из семейства языков .NET, поддерживающий функциональное программирование в дополнение к императивному и объектно-ориентированному программированию.

Код на языке F# является безопасным в отношении типов, часто бывает более компактным, чем аналогичный код C#, за счёт вывода типов. В F# действует строгая типизация, неявные преобразования типов полностью отсутствуют, что полностью исключает ошибки, связанные с приведением типов.

Такие возможности, как обобщённое программирование и функции высших порядков позволяют писать абстрактные обобщённые алгоритмы, которые управляют параметризованными структурами данных (например, массивами, списками, графами, деревьями).

Также в F# есть ссылочные типы и объекты, которые также могут содержать изменяемые значения. Тем не менее, бóльшая часть кода является чистыми функциями, что позволяет избежать многих ошибок и упростить отладку. Кроме того, упрощается распараллеливание программ. При всём этом код редко становится сложнее, чем аналогичный код на императивном языке.

Одна из основных идей F# заключается в том, чтобы удостовериться, что имеющийся код и типы в функциональном языке программирования могут быть легко доступны из других .NET-языков. Программы на F# компилируются в сборки CLR (файлы с расширениями .exe и .dll), однако, для их запуска необходима установка пакета среды исполнения дополнительно к .NET Framework.

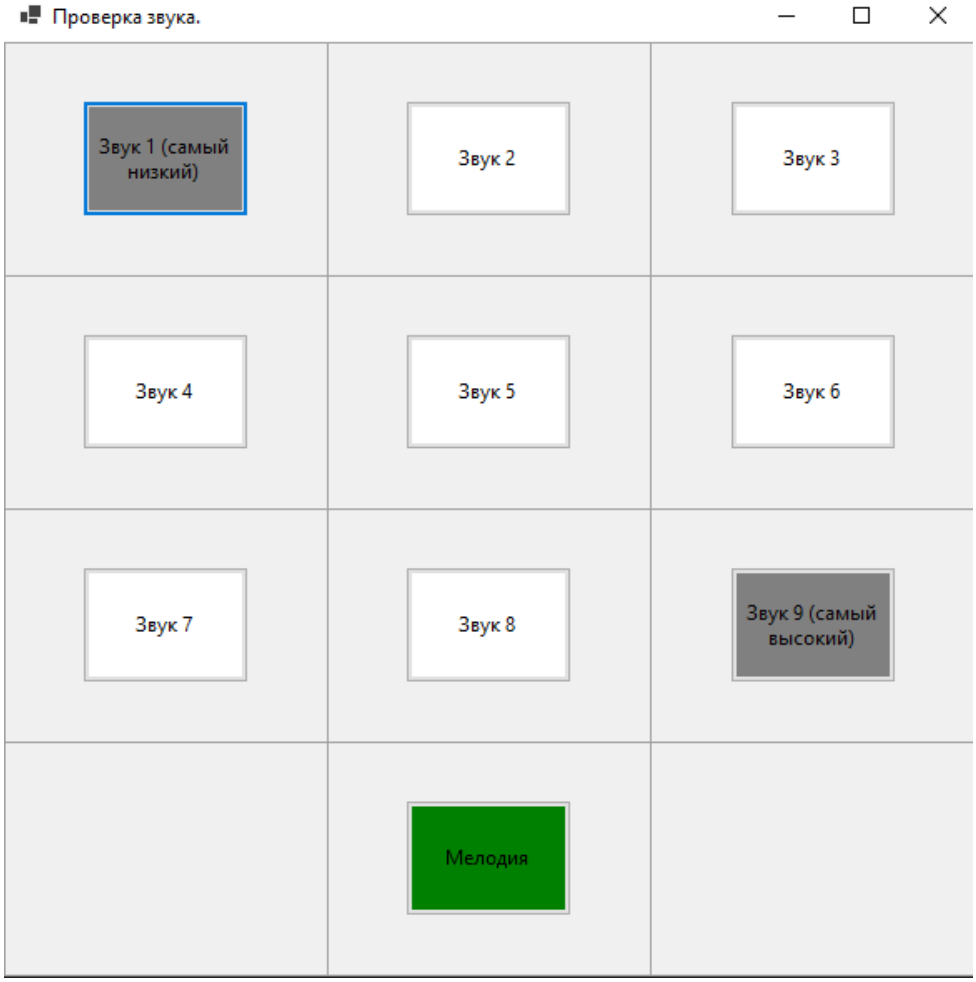
2. Разработка приложения на языке F#

2.1 Архитектура

Создается таблица, в которой уже будут располагаться наши кнопки. При нажатии на кнопки воспроизводятся разные звуки. Также кнопки меняются в размерах после воспроизведения звука – это сделано для того, чтобы человек мог понять, что звук уже произвёлся.

2.2 Интерфейс

Интерфейс отображается в виде окна с десятью кнопками.



Каждая кнопка со звуком воспроизводит писк разных частот по возрастанию. Кнопка мелодия воспроизводит небольшую композицию.

2.3 Реализация

Таблица создается путем добавления строк и столбцов

```
let tableLayoutPanel =
    let tlp = new TableLayoutPanel()
    tlp.CellBorderStyle <- TableLayoutPanelCellBorderStyle.Single
    tlp.ColumnCount <- 3
    tlp.RowCount <- 4
    tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F)) |> ignore
    tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F)) |> ignore
    tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F)) |> ignore
    tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
    tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
    tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
    tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
    tlp.Dock <- DockStyle.Fill
    tlp.Name <- "tlp"
    tlp
```

При нажатии проверяется, кликнули вы на кнопку и на какую из десяти. После этого вызывается звук «Beep» разных частот и продолжительности:

```
let buttons_Click (sender:System.Object) (e:System.EventArgs) : unit =
    let button: Button = sender :?> Button

    if button = button1 then Console.Beep(500, 500)
    if button = button2 then Console.Beep(1000, 500)
    if button = button3 then Console.Beep(1500, 500)
    if button = button4 then Console.Beep(2000, 500)
    if button = button5 then Console.Beep(2500, 500)
    if button = button6 then Console.Beep(3000, 500)
    if button = button7 then Console.Beep(3500, 500)
    if button = button8 then Console.Beep(4000, 500)
    if button = button9 then Console.Beep(4500, 500)

    if button = button10 then (
    Console.Beep(659, 400)
    Console.Beep(659, 400)
    Console.Beep(659, 800)
    Console.Beep(659, 400)
    Console.Beep(659, 400)
    Console.Beep(659, 800)
    Console.Beep(659, 400)
    Console.Beep(783, 400)
    Console.Beep(523, 400)
    Console.Beep(587, 400)
    Console.Beep(659, 800)
    )
```

Изменение размера кнопки задается двумя состояниями: plumpButton(увеличенная) и standardButton(стандартная).

```
if isStandart button then
    button.Size <- plumpButton
else
    button.Size <- standardButton
```

```
let standardButton = Size(100, 70)
let plumpButton = Size(150,120)
```

Заключение.

В результате выполнения данной работы была достигнута цель – разработана программа для проверки воспроизведения звука. Код программы был написан на функциональном языке #F, благодаря которому были рассмотрены и изучены подходы функционального программирования.

Изучение функционального языка действительно непросто, однако пытаться разобраться в нём – полезно для развития мышления и расширения кругозора.

Список литературы

1. Кубенский, А. А. Функциональное программирование : учебник и практикум для вузов / А. А. Кубенский. — Москва : Издательство Юрайт, 2021. — 348 с. — (Высшее образование). — ISBN 978-5-9916-9242-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/469863 (дата обращения: 19.01.2022).
2. Why you should think about functional programming — Текст : электронный // medium.com : [сайт]. — URL: https://medium.com/@taluyev/why-you-should-think-about-functional-programming-94fdf30936fb (дата обращения: 20.01.2022).
3. Haskell - Wikipedia. — Текст : электронный // medium.com : [сайт]. — URL: https://ru.wikipedia.org/wiki/Haskell (дата обращения: 20.01.2022).
4. Build desktop Windows apps using the Win32 API. — Текст : электронный // Microsoft technical documentation : [сайт]. — URL: https://docs.microsoft.com/en-us/windows/win32/ (дата обращения: 21.01.2022).

Приложение:

1)

```
namespace MyWindowsForm

open System.Windows.Forms
open System.Drawing
open InitControls

module FormModule =

   type MyForm() =
      inherit Form()
      do
         base.Text <- "Windows Form F#"
         base.StartPosition <- FormStartPosition.CenterScreen
         base.Size <- new Size(600, 600)
         base.TopMost <- true

   let myForm = new MyForm()

   tableLayoutPanel.SuspendLayout()
   myForm.SuspendLayout()

   tableLayoutPanel.Controls.Add(button1, 0, 0)
   tableLayoutPanel.Controls.Add(button2, 1, 0)
   tableLayoutPanel.Controls.Add(button3, 2, 0)
   tableLayoutPanel.Controls.Add(button4, 0, 1)
   tableLayoutPanel.Controls.Add(button5, 1, 1)
   tableLayoutPanel.Controls.Add(button6, 2, 1)
   tableLayoutPanel.Controls.Add(button7, 0, 2)
   tableLayoutPanel.Controls.Add(button8, 1, 2)
   tableLayoutPanel.Controls.Add(button9, 2, 2)
   tableLayoutPanel.Controls.Add(button10, 1, 3)

   myForm.Controls.Add(tableLayoutPanel)

   tableLayoutPanel.ResumeLayout(false);
   myForm.ResumeLayout(false);
```

2)

```
namespace MyWindowsForm

open System
open System.Windows.Forms
open InitControls
open FormModule
open System.Drawing

module Code =

    let myCodeForm = myForm
    myCodeForm.Text <- "Проверка звука."

    button1.Text <- "Звук 1 (самый низкий)"
    button2.Text <- "Звук 2"
    button3.Text <- "Звук 3"
    button4.Text <- "Звук 4"
    button5.Text <- "Звук 5"
    button6.Text <- "Звук 6"
    button7.Text <- "Звук 7"
    button8.Text <- "Звук 8"
    button9.Text <- "Звук 9 (самый высокий)"
    button10.Text <- "Мелодия"

    let isStandart (btn: Button) : bool = btn.Size = standardButton

    let buttons_Click (sender:System.Object) (e:System.EventArgs) : unit =
        let button: Button = sender :?> Button

        if button = button1 then Console.Beep(500, 500)
        if button = button2 then Console.Beep(1000, 500)
        if button = button3 then Console.Beep(1500, 500)
        if button = button4 then Console.Beep(2000, 500)
        if button = button5 then Console.Beep(2500, 500)
        if button = button6 then Console.Beep(3000, 500)
        if button = button7 then Console.Beep(3500, 500)
        if button = button8 then Console.Beep(4000, 500)
        if button = button9 then Console.Beep(4500, 500)

        if button = button10 then (
        Console.Beep(659, 400)
        Console.Beep(659, 400)
        Console.Beep(659, 800)
        Console.Beep(659, 400)
        Console.Beep(659, 400)
        Console.Beep(659, 800)
        Console.Beep(659, 400)
        Console.Beep(783, 400)
        Console.Beep(523, 400)
        Console.Beep(587, 400)
        Console.Beep(659, 800)
        )

        if isStandart button then
            button.Size <- plumpButton
        else
            button.Size <- standardButton

    button1.Click.AddHandler( new System.EventHandler(buttons_Click) )
    button2.Click.AddHandler( new System.EventHandler(buttons_Click) )
    button3.Click.AddHandler( new System.EventHandler(buttons_Click) )
    button4.Click.AddHandler( new System.EventHandler(buttons_Click) )
    button5.Click.AddHandler( new System.EventHandler(buttons_Click) )
    button6.Click.AddHandler( new System.EventHandler(buttons_Click) )
```

```
button7.Click.AddHandler( new System.EventHandler(buttons_Click) )
button8.Click.AddHandler( new System.EventHandler(buttons_Click) )
button9.Click.AddHandler( new System.EventHandler(buttons_Click) )
button10.Click.AddHandler( new System.EventHandler(buttons_Click) )
```

3)

```
namespace MyWindowsForm

open System.Windows.Forms
open System.Drawing

module InitControls =

    let standardButton = Size(100, 70)
    let plumpButton = Size(150,120)

    let tableLayoutPanel =
        let tlp = new TableLayoutPanel()
        tlp.CellBorderStyle <- TableLayoutPanelCellBorderStyle.Single
        tlp.ColumnCount <- 3
        tlp.RowCount <- 4
        tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F)) |> ignore
        tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F)) |> ignore
        tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, 50F)) |> ignore
        tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
        tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
        tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
        tlp.RowStyles.Add(new RowStyle(SizeType.Percent, 50F)) |> ignore
        tlp.Dock <- DockStyle.Fill
        tlp.Name <- "tlp"
        tlp

    let button1 =
        let btn = new Button()
        btn.Name <- "button1"
        btn.Size <- standardButton
        btn.TabIndex <- 0
        btn.Text <- "button1"
        btn.Anchor <- AnchorStyles.None
        btn.BackColor <- Color.Gray
        btn

    let button2 =
        let btn = new Button()
        btn.Name <- "button2"
        btn.Size <- standardButton
        btn.TabIndex <- 1
        btn.Text <- "button2"
        btn.Anchor <- AnchorStyles.None
        btn.BackColor <- Color.White
        btn

    let button3 =
        let btn = new Button()
        btn.Name <- "button3"
        btn.Size <- standardButton
        btn.TabIndex <- 2
        btn.Text <- "button3"
        btn.Anchor <- AnchorStyles.None
        btn.BackColor <- Color.White
        btn

    let button4 =
```

```
    let btn = new Button()
    btn.Name <- "button4"
    btn.Size <- standardButton
    btn.TabIndex <- 3
    btn.Text <- "button4"
    btn.Anchor <- AnchorStyles.None
    btn.BackColor <- Color.White
    btn

let button5 =
    let btn = new Button()
    btn.Name <- "button5"
    btn.Size <- standardButton
    btn.TabIndex <- 4
    btn.Text <- "button5"
    btn.Anchor <- AnchorStyles.None
    btn.BackColor <- Color.White
    btn

let button6 =
    let btn = new Button()
    btn.Name <- "button6"
    btn.Size <- standardButton
    btn.TabIndex <- 5
    btn.Text <- "button6"
    btn.Anchor <- AnchorStyles.None
    btn.BackColor <- Color.White
    btn

let button7 =
    let btn = new Button()
    btn.Name <- "button7"
    btn.Size <- standardButton
    btn.TabIndex <- 6
    btn.Text <- "button7"
    btn.Anchor <- AnchorStyles.None
    btn.BackColor <- Color.White
    btn

let button8 =
    let btn = new Button()
    btn.Name <- "button8"
    btn.Size <- standardButton
    btn.TabIndex <- 7
    btn.Text <- "button8"
    btn.Anchor <- AnchorStyles.None
    btn.BackColor <- Color.White
    btn

let button9 =
    let btn = new Button()
    btn.Name <- "button9"
    btn.Size <- standardButton
    btn.TabIndex <- 8
    btn.Text <- "button9"
    btn.Anchor <- AnchorStyles.None
    btn.BackColor <- Color.Gray
    btn

let button10 =
    let btn = new Button()
    btn.Name <- "button10"
    btn.Size <- standardButton
    btn.TabIndex <- 9
    btn.Text <- "button10"
    btn.Anchor <- AnchorStyles.None
```

```
btn.BackColor <- Color.Green
btn
```