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»

Студентки 3 курса очного
отделения
группы 2321-ДБ
Захаровой Ольги Олеговны

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».

Существует множество причин, почему стоит постичь функциональную парадигму и в чём заключаются сложности, но не каждый программист будет заинтересован в данном вопросе в связи со сложившимся императивным мышлением.

Однако, поклонники программирования на функциональном языке Haskell считают, что программисты, использующие императивные языки, должны хотя бы попробовать научиться чистой функциональной разработке ради глубокого понимания принципов самого программирования и разницы функциональной и императивной разработки.

Цель работы: создание приложения, позволяющего сохранять изображение экрана на языке функционального программирования Haskell.

Задачи работы:

1. Изучение подхода функционального программирования;
2. Создание программы захвата экрана;
3. Создание интерфейса программы;
4. Создание функции захвата полного и активного окна;
5. Реализация копирования в буфер обмена, задержки.

1. Функциональное программирование – Haskell

Haskell — чисто функциональный типизированный язык с нестрогой (ленивой) моделью исполнения. Функциональное программирование в целом, и Haskell, как один из наиболее чистых представителей этой парадигмы, позволяет программисту освоить новый нетрадиционный подход к написанию программ.

Немаловажной особенностью Haskell является то, что он поддерживает ленивые вычисления, которые позволяют сильно ускорить работу программы и снизить нагрузку на память, а также сделать код проще. Ленивые вычисления выполняются тогда, когда это необходимо программе, а не в том моменте, когда их указал программист.

В неленивых или «энергичных» вычислениях — аргументы функции вычисляются перед выполнением самой функции. В языках программирования, которые применяют ленивые вычисления, этот процесс постоянно откладывается, и аргументы функции вычисляются только для реальной надобности, а не в том месте, где их указал разработчик. Например, если сейчас значение какой-то функций не нужно, и оно не используется, то Haskell не будет высчитывать ее аргументы.

Haskell часто используют для написания инструментов для обработки текстов, синтаксического анализа создания фильтр-систем для обработки спама и т.д... Структура Haskell позволяет достаточно просто заложить в него правила языка, в том числе и русского, и обучить алгоритмы находить взаимосвязи с ним.

У Haskell есть репозитории, из которых можно загружать библиотеки, позволяющие расширить базовые возможности языка.

Для выполнения цели работы были применены библиотеки:

- `haskell-gi` – создаёт привязки Haskell для библиотек с поддержкой проекта (`GObject Introspection`) по предоставлению машиночитаемых данных самоанализа API библиотек C. Эти данные самоанализа можно использовать в нескольких различных случаях, например, для автоматической генерации кода для привязок, проверки API и создания документации. Сюда относится и `GTK` (`GIMP Toolkit` – кроссплатформенная библиотека элементов интерфейса)
- `gi-gtk` – привязки для `GTK`, автоматически сгенерированные `haskell-gi`, используется для создания окна и внутри оконных элементов (виджетов) приложения.
- `Win32` – эта библиотека содержит прямые привязки к Windows Win32 API для Haskell, используется для создания снимков экрана и буфера обмена.

Также важную роль выполняют следующие модули:

- `Data.GI.Base` – обеспечивает удобный заголовок для основных модулей `GObject Introspection`
- `Control.Concurrent` – общий интерфейс для набора полезных абстракций параллелизма, используется для создания задержки (аналога sleep в прочих языках программирования)

2. Разработка приложения на языке Haskell

2.1 Архитектура

Основную часть работы берёт на себя библиотека gi-gtk. С помощью неё создается интерфейс программы. Однако, не стоит недооценивать библиотеку Win32 – она позволяет воспользоваться буфером обмена и создать снимки экрана в Windows.

Функция буфера обмена clip принимает два параметра – a и b.

За создание снимков экрана отвечает функция screen с параметрами take и name, откуда передаются аргументы в clip – тип съемки экрана и создание bmp-картинки.

В функции main описывается создание интерфейса и передача аргументов функции screen – тип съемки экрана и соответствующее имя сохраняемого файла в текущий каталог.

2.2 Интерфейс

Интерфейс отображается в виде небольшого окна с двумя кнопками.

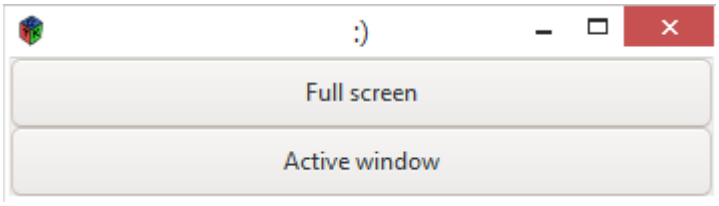

Рисунок 1. Окно программы

Первая кнопка с соответствующим названием выполняет съемку полного экрана, вторая кнопка снимает окно программы, находившейся в фокусе перед нажатием на данную кнопку.

2.3 Реализация

Точка входа – функция main, в которой происходит инициализация GTK, создание интерфейса: окно, контейнер с кнопками.

При нажатии на любую из кнопок окно исчезает из виду, выполняет действие и возвращается в исходное состояние.

Создание снимка экрана происходит следующим образом:

- После исчезновения окна, программа засыпает на 1000000 микросекунд. Данный прием необходим для исключения отображения окна на изображении в дальнейшем.
- Захватывается дескриптор области снимка экрана (полный экран или активное окно, в зависимости от нажатой кнопки).
- Берётся контекст устройства для дескриптора области снимка

экрана, а именно – строка заголовка (или строки заголовков), меню и полосы прокрутки.

- Создается новый контекст устройства памяти для дескриптора области снимка, совместимый с данным компьютером. Это необходимо для удержания скопированного изображения с последующим взаимодействием.
- Вычисляется размер конечного растрового изображения (ширина, высота).
- Выполняется создание растрового изображения, совместимое с ранее созданным контекстом.
- Текущее растровое изображение записывается в bmp-файл и копируется в буфер обмена, файл сохраняется текущем каталоге (директории данного приложения).
- Завершается снимок экрана на удалении контекста устройства.

2.4 Демонстрация

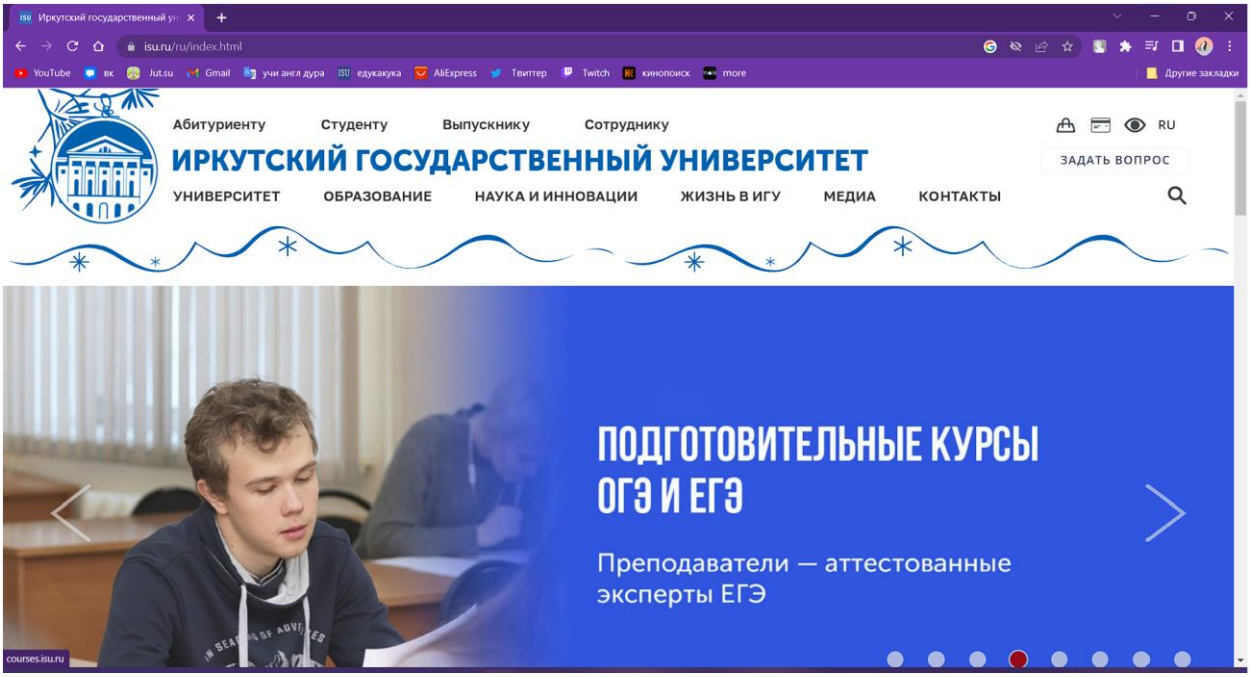

Рисунок 2. Скриншот активного окна

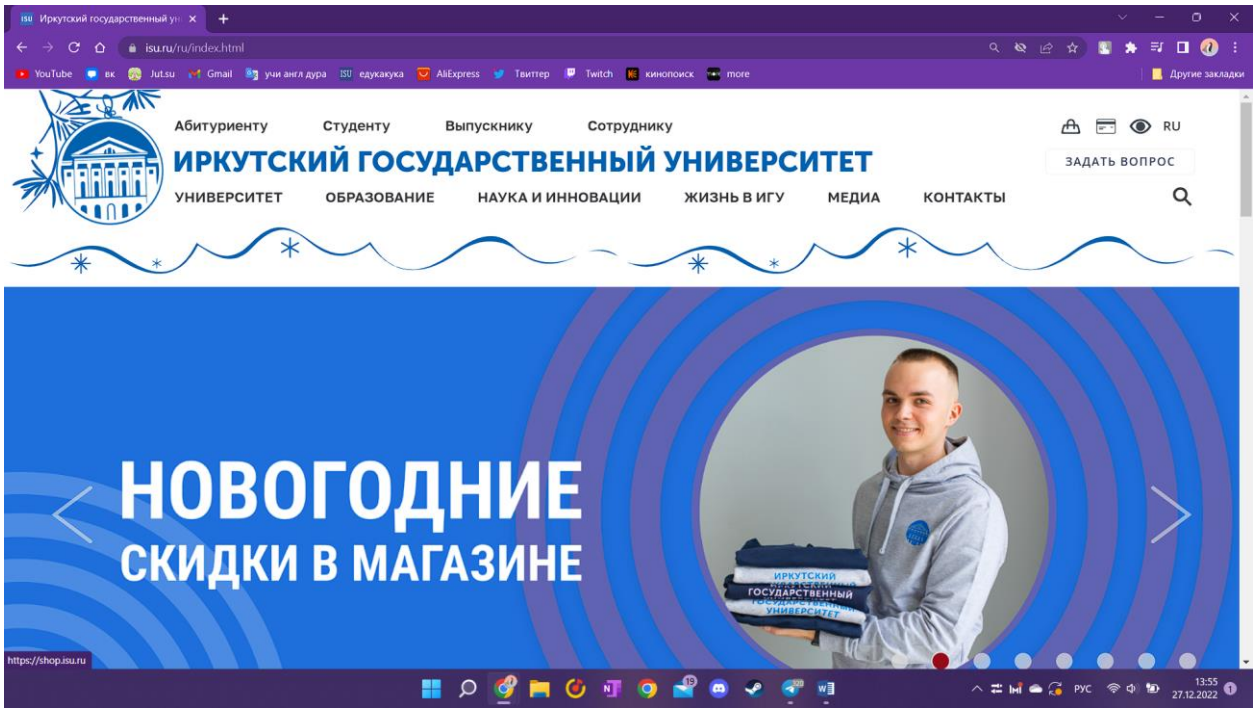

Рисунок 3. Скриншот экрана

Заключение.

В результате выполнения данной работы была достигнута цель – разработана программа для создания снимков экрана (скриншотов). Код программы был написан на функциональном языке Haskell, 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. Листинг Main.hs

```
{-# LANGUAGE OverloadedLabels  #-}
{-# LANGUAGE OverloadedStrings #-}

import Data.GI.Base
import qualified GI.Gtk as Gtk
import Graphics.Win32.Window
import Graphics.Win32.GDI.Bitmap
import Graphics.Win32.GDI.HDC
import Graphics.Win32.GDI.Graphics2D
import Graphics.Win32.GDI.Clip
import Control.Concurrent


main = do
  Gtk.init Nothing
  win <- new Gtk.Window [#title := " :) " ]
  on win #destroy Gtk.mainQuit
  #resize win 350 50
  box <- new Gtk.Box [#orientation := Gtk.OrientationVertical]
  #add win box

  full <- new Gtk.Button [#label := "Full screen"]
  #add box full
  on full #clicked (do
       #hide win
       threadDelay 1000000 --microseconds = 1 sec
       screen(getDesktopWindow,"full.bmp")
       #show win)

  active <- new Gtk.Button [#label := "Active window"]
  #add box active
  on active #clicked (do
        #hide win
        threadDelay 1000000
        screen(getForegroundWindow,"active.bmp")
        #show win)

  #showAll win
  Gtk.main


screen(take,name) = do canvas <- take
                       hdc <- getWindowDC (Just canvas)
                       (x,y,r,b) <- getWindowRect canvas
                       newDC <- createCompatibleDC (Just hdc)
                       let width  = r - x
                       let height = b - y
                       newBmp <- createCompatibleBitmap hdc width height
                       selBmp <- selectBitmap newDC newBmp
```

```
                    bitBlt newDC 0 0 width height hdc 0 0 sRCCOPY
                    createBMPFile name newBmp newDC
                    clip(canvas,newBmp)
                    deleteBitmap newBmp
                    deleteBitmap selBmp
                    deleteDC newDC


clip(a,b) = do openClipboard a
               emptyClipboard
               setClipboardData cF_BITMAP b
               closeClipboard
```