ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Р.Е. Алексеева»

Институт Радиоэлектроники и Информационных Технологий

Кафедра "Вычислительные системы и технологии"

ОТЧЁТ

По курсовой работе

«Справочно-обучающая программа “FPU Intel x87”»

по дисциплине «Принципы и методы организации системных программных средств»

Выполнил:

Проверил:

Кочешков А.А.

Нижний Новгород

Содержание:

1. Введение
2. Теория
   1. Язык программирования C
   2. GTK
   3. FPU Intel x87
3. Практика
   1. Структура
   2. Дизайн
   3. Описание исходных кодов
   4. Результат работы программы
   5. Вывод
4. Исходные коды

Введение

Поскольку тема для затребованной программы является комплексной и относительно трудной к её пониманию и подаче, в данной работе основное внимание уделено программе, ресурсам, на которых она построена, а также логике её работы и обработки изменяемого массива данных. Материал для программы был взят и обработан из документа Intel “ Programming With the x87 Floating-Point Unit”.

Программа была написана на языке программирования C, под операционной системой linux, дистрибутив fedora workstation 36.

Программа была задумана с графическим интерфейсом, и для его реализации была выбрана библиотека GTK 4 версии, как наиболее простая и совместимая с языком C.

Чтобы скомпилировать исполняемый файл программы, требуется установить зависимости и пакеты, необходимые для сборки проектов, разрабатываемых на GTK.

На Debian/Ubuntu существует 3 пакета:

- Бинарный libgtk-4-1;

- Пакет разработки libgtk-4-dev;

- Дополнительный пакет с примерами gtk-4-examples;

Для Fedora имеются два пакета:

- Бинарный gtk-4;

- Пакет разработки gtk4-devel;

Чтобы создавать проекты на основе GTK4 под Fedora, достаточно установить пакет gtk4-devel.

Установка производится в терминале с помощью команды:

sudo dnf install gtk4-devel

При компиляции исходных кодов, также необходимо использовать pkg-config для указания флагов компиляции (--cflags) и библиотек (--libs). В обоих случаях, нужно прописать gtk4. Для сборки проекта, я использовал функцию make и прописал инструкции сборки в make файле. Подробнее с его содержанием можно ознакомиться в разделе «Исходные коды».

Теория

Язык программирования C

C – процедурный язык программирования. Разработан в начале семидесятых годов Кеном Томпсоном и Денисом Ритчи с целью использования в операционной системе UNIX, и написания её ядра. С того момента, язык не потерял актуальности, а разросся на многие платформы и стал одним из наиболее широко используемых.

Следование процедурной парадигме позволяет создавать программы, в которых каждое действие выполняется шаг за шагом, что даёт широкую гибкость в ходе разработки и реализации программных решений.

Данный язык отличают:

Скромный синтаксис относительно современных языков программирования;

Статическая слабая типизация – у всех данных есть определённые типы, однако допустимы неявные их преобразования;

Низкоуровневый доступ к памяти;

Поддержка указателей на переменные и функции – указатель соответствует машинному адресу объекта, а имеющаяся арифметика с указателями позволяет манипулировать адресами памяти;

Препроцессор, обрабатывающий исходные коды перед компиляцией;

Возможность разбивать код на файлы и компилировать их отдельно друг от друга, а затем линковать друг с другом.

C – компилируемый язык, что значит, программа, написанная на данном языке, преобразуется в набор инструкций, доступных процессору, благодаря чему, достигаются высокая скорость и эффективность исполнения.

По причине своих простоты и гибкости, данный язык хорошо подошёл для создания программы на его основе – с помощью доступных средств и библиотек, мной были реализованы методы обработки данных.

GTK

GTK – это библиотека элементов графического интерфейса. Она предоставляет широкий набор доступных решений, относительно проста в применении. Данный фреймворк довольно популярен для разработки программ под linux. GTK написана на языке Си, однако является объектно-ориентированной.

GTK состоит из двух компонентов: GTK и GDK.

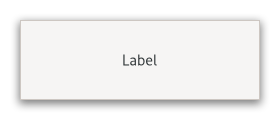
Первый компонент содержит набор элементов пользовательсокго интерфеса, или виджетов – таких, как кнопка, поле с текстом, и другие.

Второй отвечает за вывод на экран. С версии 2.8,  GDK широко заменяется системой отрисовки векторной графики Cairo.

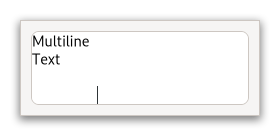
Далее, приведены классы, определённые в библиотеке, которые были использованы в программе.

GtkWidget — это базовый класс, от которого происходят все виджеты в GTK. Он управляет жизненным циклом, состояниями и стилем виджета.

Виджет GtkLabel отображает небольшое количество текста.



GtkTextView - Виджет, отображающий содержимое GtkTextBuffer. Позволяет как создать виджет с большим количеством текста, так и поле для ввода текста. Есть возможность сделать активным или неактивным редактирование отображаемого текста.



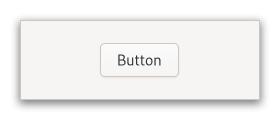
GtkTextBuffer - Сохраняет текст и атрибуты для отображения в GtkTextView.

GtkPicture – отображает изображения. Предоставляет возможность рисовать на загруженных изображениях. Доступно отображение изображения из буфера GdkPixbuf.

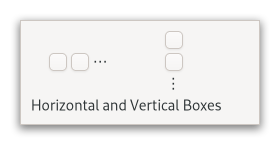


GdkPixbuf – пиксельный буфер. Содержит информацию об изображении. Позволяет гибко настроить загружаемое изображение, определить его цвета, размеры и прочие атрибуты.

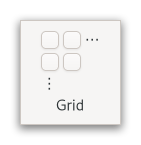
GtkButton – виджет, соответствующей кнопке. Обычно используется для запуска функции обратного вызова, которая вызывается при нажатии кнопки. Его интересной особенностью является то, что он может содержать в себе дочерний элемент, такой, как лейбл или изображение.



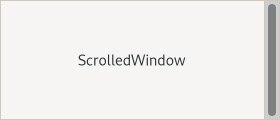
GtkBox - упорядочивает дочерние виджеты в одну строку или столбец.



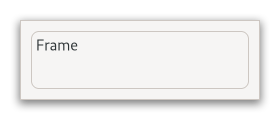
GtkGrid — это контейнер, который упорядочивает дочерние виджеты по строкам и столбцам.



GtkScrolledWindow — это контейнер, который делает свой дочерний элемент прокручиваемым.

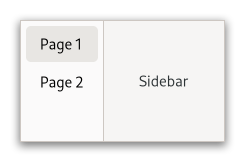


GtkFrame — это виджет, который окружает своего потомка декоративной рамкой и опциональной меткой.



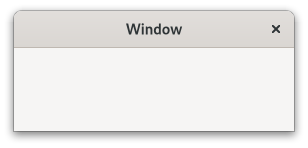
GtkStack — это контейнер, который одновременно показывает только одного из своих дочерних элементов. GtkStack не предоставляет пользователям средств для изменения видимого дочернего элемента. Вместо этого с GtkStack можно использовать отдельный виджет, такой как GtkStackSwitcher или GtkStackSidebar, для обеспечения этой функциональности.

GtkStackSidebar представляет собой боковую панель для переключения между страницами GtkStack.



GtkWindow — это виджет окна, которое может содержать другие виджеты.

Окна обычно имеют декорации, которые находятся под управлением оконной системы и позволяют пользователю манипулировать окном (изменять его размер, перемещать, закрывать и т. д.).



FPU Intel x87

FPU intel x87 является математическим сопроцессором. FPU – *floating point unit*, в переводе, модуль операций с плавающей запятой - используется для решения широкого спектра математических операций над вещественными числами.

Процессоры семейства x86 с 8086/8088 по 386 не обладали встроенным модулем для решения операций с плавающей запятой, поэтому для них, впоследствии, была разработана микросхема, названная математическим сопроцессором.

Сопроцессор не может функционировать самостоятельно и является дополнением к основному процессору. Он подключается к шинам центрального процессора и имеет несколько специальных сигналов для синхронизации процессоров между собой.

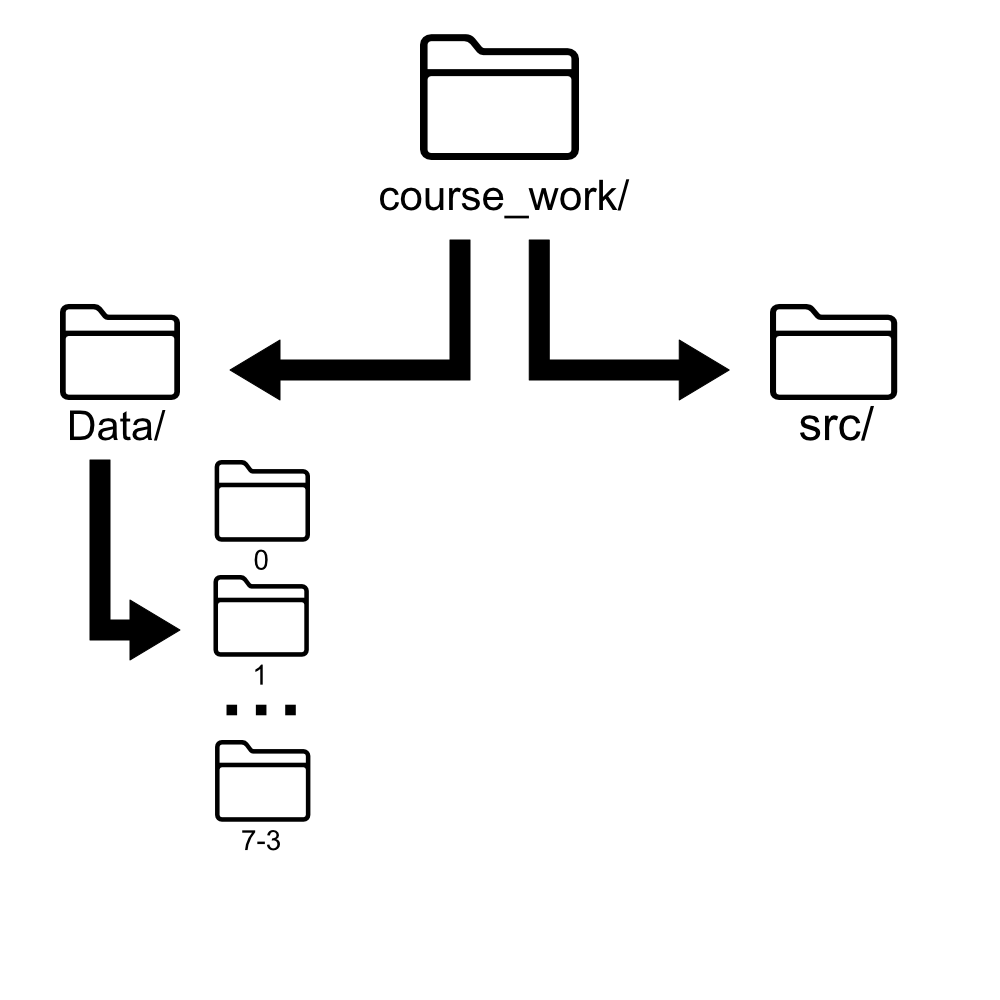
Начиная с процессора Intel486DX модуль операций с плавающей запятой был интегрирован в центральный процессор и назван FPU.

Особенностью данных сопроцессоров является то, что регистры FPU организованы не в виде массива, а как регистровый стек. Это означает, что команды всегда используют верхнее значение в стеке для проведения операций, а доступ к другим хранящимся значениям обычно обеспечивается в результате манипуляций со стеком.

Практика

Структура

Поскольку проект представляет собой программу, отображающую массив информации, он разделён на две части: данные и исходный код.



В материале Intel представлено 44 страницы, 55 заголовков. Это большой объём данных, и перенос информации непосредственно в код было бы неоправданно затруднительным процессом. По этой причине, я решил разбить материал по заголовкам и для каждого такого раздела генерировать отдельную страницу в программе, на основе универсального метода.

В папке Data/ я разместил 55 папок – каждая соответствует странице программы. Каждая папка вмещает в себя файлы по трём заданным шаблонам:

* label.txt;
* picture-N.jpg – где N – порядковый номер картинки;
* text-N.txt – где N – порядковый номер текстового блока.

Файл label.txt хранит название заголовка. На каждую страницу приходится один заголовок.

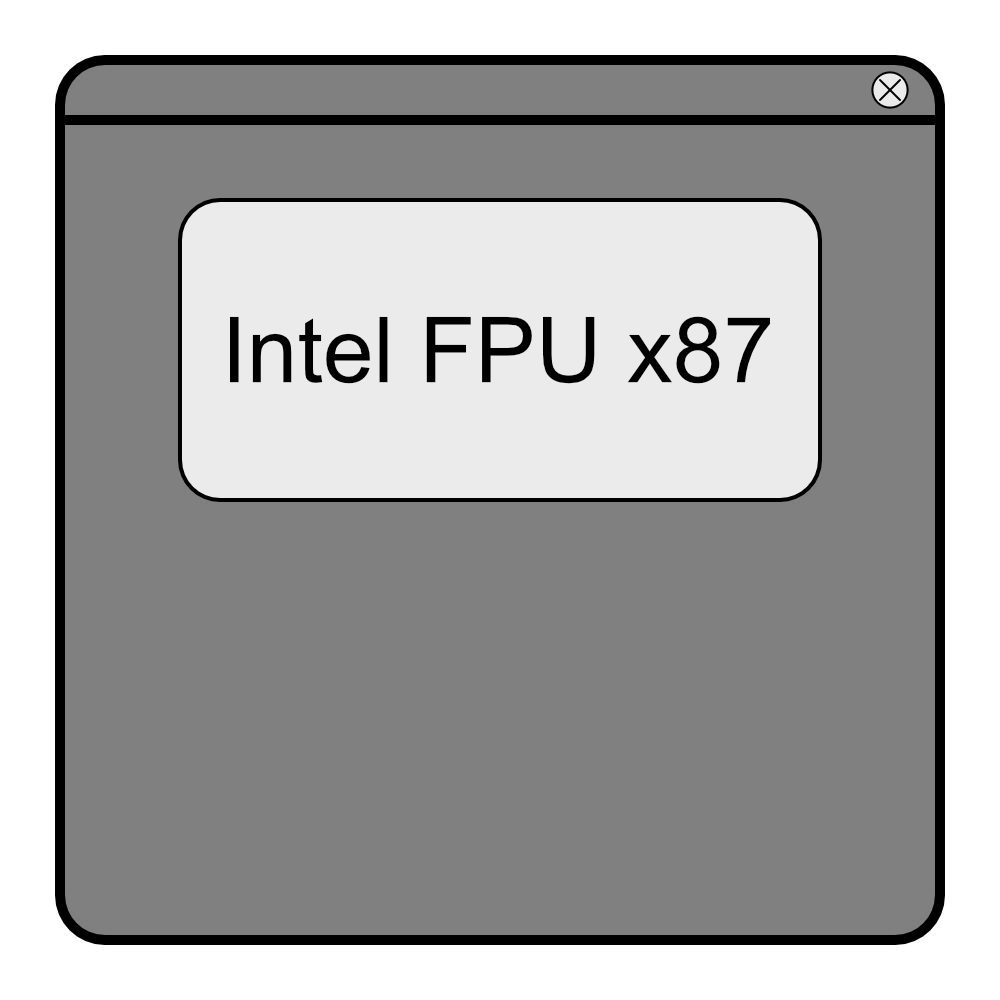
Файлы picture-N.jpg относятся к изображениям, а файлы text-N.txt к текстовым блокам.

В разделах могут как присутствовать одно или несколько изображений, так и отсутствовать полностью. Неизменным остаётся наличие текста. Вариативность присутствия изображений создаёт трудность в реализации шаблона страницы, поскольку материал может быть построен логически таким образом, чтобы определённые изображения следовали до или после конкретных текстовых блоков. Поэтому, я решил разбить данные на последовательные части и реализовать логику их обработки.

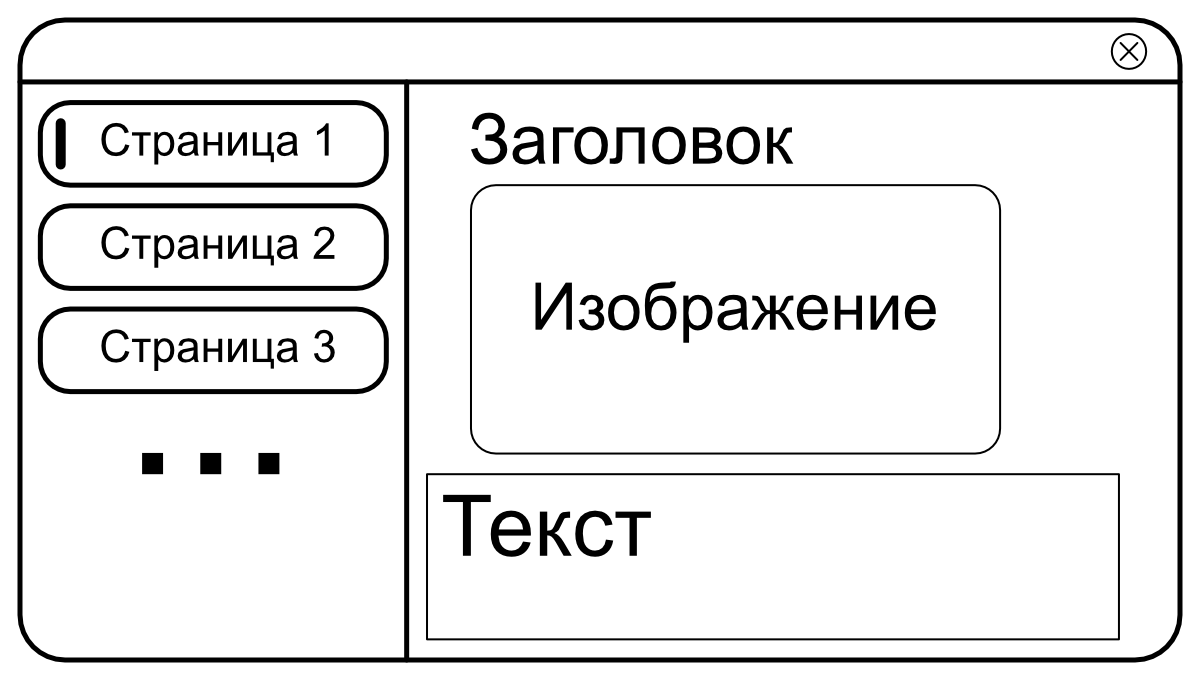
Дизайн приложения

Дизайн приложения было решено сделать простым и функциональным.

Стартовое окно должно презентовать материал. По нажатии на специальную кнопку, откроется основное окно с информацией. Такой подход также позволяет создать пространство для расширения приложения и дополнения другими материалами.

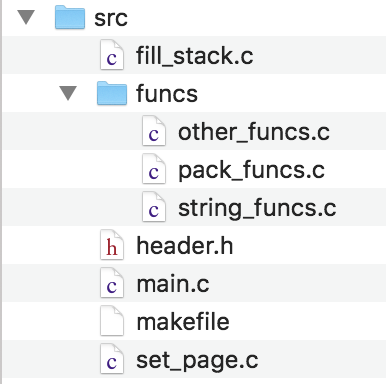


Основное окно должно отображать материал и предоставлять средство навигации по информации. Для этого, хорошо подойдёт виджет GtkStackSideBar. Страницы будут помещены в стек GtkStack, а боковое меню позволит переключаться между ними.



Описание исходных кодов

На изображении ниже приведена структура файлов исходных кодов.



Файл main.c

В файле main.c представлено три функции:

- int main() – главная функция, с которой начинается программа на Си. Описанный код внутри был взят из стандартного шаблона. Внутри создаётся объект типа GtkApplication — это класс, который обрабатывает многие важные аспекты приложения GTK удобным способом, не навязывая универсальную модель приложения. Это основа приложения, с него начинается создание графической оболочки. Функция gtk\_application\_new() возвращает указатель на экземпляр класса GtkApplication, а в качестве аргумента принимает идентификатор приложения и флаги. Функция g\_signal\_connect() вызывает функцию start\_window(), отрисовывающую старотовое окно программы. Функция g\_application\_run() запускает приложение.

GObject — это основной тип, обеспечивающий общие атрибуты и методы для всех типов объектов в GTK, и других библиотек, основанных на Gobject. Класс GObject предоставляет методы для создания и уничтожения объектов. При создании объекта класса Gobject (и его наследников), создаётся ссылка на этот объект. Когда объект более не используется, требуется вызвать функцию g\_object\_unref() для освобождения предоставленной ссылки. GObject — это тип с подсчетом ссылок. Подсчет ссылок — это форма сборки мусора.

- start\_window() – в данной функции описано создание стартового окна с кнопкой запуска основного окна с данными по материалу из папки Data/. По нажатии на кнопку, функция g\_signal\_connect() вызывает функцию main\_window(), а затем, функция g\_signal\_connect\_swapped() вызывает функцию gtk\_window\_destroy(), закрывающую стартовое окно.

- main\_window() – функция, в которой описано основное окно. Помимо виджета непосредственно окна, здесь присутствуют только три виджета: контейнер виджетов GtkGrid, стек виджетов GtkStack и переключатель дочерних элементов стека GtkStackSidebar. После создания экземпляра стека, вызывается функция fill\_stack(), заполняющая стек данными из директории Data/.

Файл fill\_stack.c

В данном файле описана функция fill\_stack(). Функция принимает в качестве аргумента виджет стека. Внутри этой функции, с помощью функции scan\_dir\_set\_pack(), директория Data/ анализируется на количество файлов внутри неё. Функция scan\_dir\_set\_pack() возвращает указатель на структуру типа Package, описанную в заголовочном файле header.h. Из данной структуры извлекается список имён файлов из директории Data/ и количество этих имён. Далее, для всех элементов полученного списка, в цикле вызывается функция set\_page(), чтобы создать страницу и поместить в стек.

Файл set\_page.c

В данном файле описана одна функция set\_page(). Функция служит для сборки страницы с данными соответствующего раздела материала. Она принимает на вход стек и строку с именем директории, в которой лежат данные для отображения. Функция условно разделена на 4 части:

В начале, с помощью функции scan\_dir\_set\_pack() получаем список файлов внутри каталога. С помощью функции set\_pack\_with\_numbers() получаем списки из номеров, соответствующих порядковой принадлежности файлов изображений и текстовых блоков. С помощью функции shake\_sort\_int() сортируем списки по возрастанию.

Во втором условном блоке, объявляем используемые виджеты, создаём контейнер с вертикальным расположением дочерних элементов.

В третьем блоке, происходит заполнение контейнера виджетами. Сначала, добавляется заголовок, или виджет GtkLabel. Если заголовок не удаётся считать из файла по указанному шаблону, ставится прочерк.

Определяется максимальный порядковый номер изображений и текстовых блоков и в соответствии с наибольшим, запускается цикл, внутри которого происходит добавление виджетов изображений и текста в соответствии с порядковыми номерами. Если счётчик цикла не совпадает с порядковым номером файла, файл не читается и счётчик цикла увеличивается. Если счётчик цикла совпадает с порядковым номером файла, файл отрисовывается и адрес в списке номеров данного виджета увеличивается на единицу, т. е. в следующей итерации, счётчик цикла будет сравниваться со следующим порядковым номером набора файлов.

В последнем блоке, контейнер добавляется в виджет перелистывания, который, в свою очередь, переносится в стек. Каждый дочерний элемент стека может быть проименован, что затем будет отображено в переключателе стека. Однако, некоторые заголовки слишком длинные, из-за чего, происходит некорректное отображение боковой панели GtkStackSidebar. Для решения этой проблемы, вызывается функция split\_long\_string\_with\_interval(), которая разбивает слишком длинные заголовки с помощью переноса на следующую строку.

Далее, освобождается вся выделенная память под вспомогательные переменные.

Файл header.h

В этом заголовочном файле описаны вспомогательные функции, используемые в реализации логики работы программы. В нём также описываются структуры и макросы. В заголовочном файле функции только объявлены, а их реализация находится в файлах папки funcs.

Макрос PACK(X) используется для конвертации динамического массива либого типа к типу void\*. Используется для присваивания значения элементу list структуры Package.

Макрос PACK\_2D\_ARRAY(X) макрос, используемый, чтобы конвертировать двухмерный массив либого типа в тип void\*\*, для передачи в функцию free\_2d\_array().

Структура Package – используется для передачи списков любого типа и его размерности их одних функций в другие.

Функция pack\_new() – передаёт указатель на новый экземпляр структуры Package.

Функция get\_pack\_with\_numbers\_new() – принимает на вход строку, список строк и размер списка, внутри генерирует новый список и заполняет его строками, чьи первые символы соответствуют символам из строки, переданной в качестве первого аргумента. Возвращает новый список и его размер в виде указателя на новый экземпляр структуры Package.

Функция set\_pack\_with\_numbers() делает то же самое, но вместо создания нового экземпляра структуры, она принимает на вход существующий и заполняет данные в него.

set\_pack\_with\_numbers() – сканирует директорию, путь к которой указан во втором аргументе, и помещает список найденных файлов в переданный первым аргументом экземпляр структуры Package, вместе с размером списка.

length() – функция принимает на вход строку и возвращает количество символов в ней, до нулевого ‘\0’.

compare\_strings() – принимает на вход две строки, соответствующие шаблонным именам папок в директории Data/. Возвращает 1, если первая строка «больше», и 2, если вторая; 0, если равны.

shake\_sort\_data() – принимает на вход писок строк и его размер, в соответствии с результатами функции compare\_strings() сортирует элементы списка. Используется для сортировки списка каталогов в директории Data/, поскольку, при использовании функций библиотеки dirent.h, список элементов получается неупорядоченный.

glue\_strings() – соединяет строки из первого и второго аргументов последовательно и помещает их в строку, переданную третьим аргументом.

string\_append() – присоединяет с строке, переданной первым аргументом, строку из второго аргумента. Строка из первого аргумента должна быть димачической.

get\_string\_from\_num() – функция, принимающая на вход целочисленное значение и возвращающая его в качестве новой строки.

add\_num\_to\_string() – функция добавляет целочисленное значение, переданной вторым аргументом, в конец строки, переданной в качестве первого аргумента. Строка должна быть динамической.

set\_string\_from\_num() - переводит целочисленное значение в строку, переданную первым аргументом. Если в строке были данные, они окажутся стёрты.

string\_new() – возвращает указатель на пустой строковый массив.

get\_string\_with\_text\_new() – возвращает динамическую строку с текстом, переданным в качестве аргумента. Используется, чтобы преобразовать статические строки в динамические.

string\_from\_file\_new() – считывает файл, путь к которому указан в аргументе функции, и возвращает новую строку с текстом из файла или нулевой указатель, если файл не удалось прочитать.

set\_string\_from\_file() – делает то же самое, но вместо возврата новой строки, функция помещает данные в строку, переданную вторым аргументом.

insert\_letter\_to\_string() – функция вставляет букву из третьего аргумента, на позицию, указанную во вором аргументе, в строку, переданную в качестве первого аргумента. Если позиция указана вне диапазона элементов строки, возвращается строка без изменений.

split\_long\_string\_with\_interval() – функция разбивает переданную строку в первом аргументе переходами на новую строку. Разбиение происходит с интервалом, указанным во втором аргументе. Вставка символа перехода на новую строку происходит на место первого пробела в сторону убывания порядкового номера символов в строке, или, если пробелов не обнаружено, переход на новую строку встраивается в строку после последнего элемента в рамках текущего отрезка. Следующий отрезок вычисляется от места вставки сивола перехода на новую строку плюс интервал.

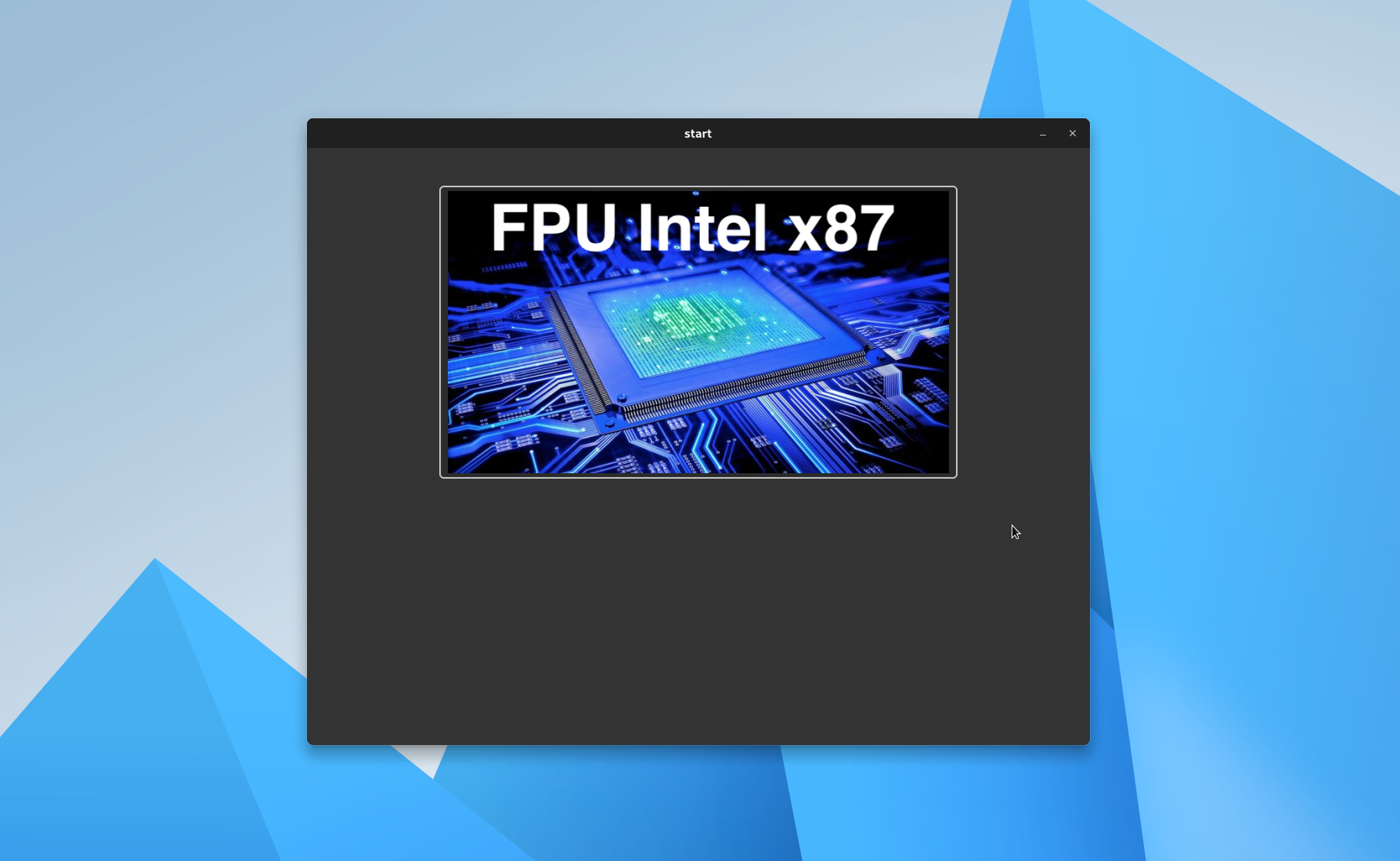
free\_2d\_array() – функция освобождает память любого двухмерного массива.

shake\_sort\_int() - функция сортирует целочисленный массив по принципу шейкерной сортировки.

pow\_i() – возводит целочисленное значение из первого аргумента, в степень, указанную вторым элементом. Второй элемент должен быть неотрицательным и целочисленным.

Результат работы программы

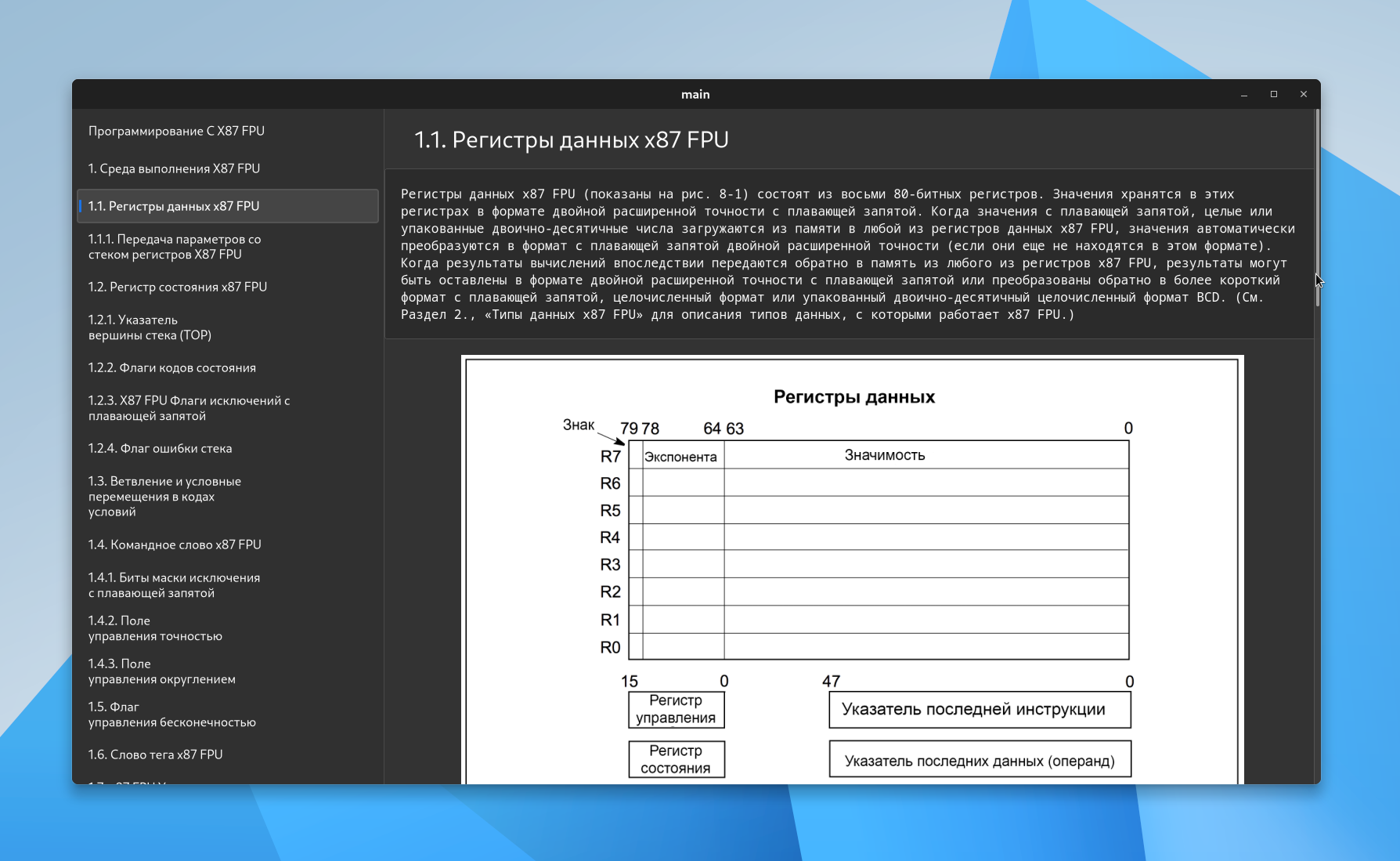
При запуске программы, появляется стартовое окно, описанное в функции start\_window(), реализованной в файле main.c.

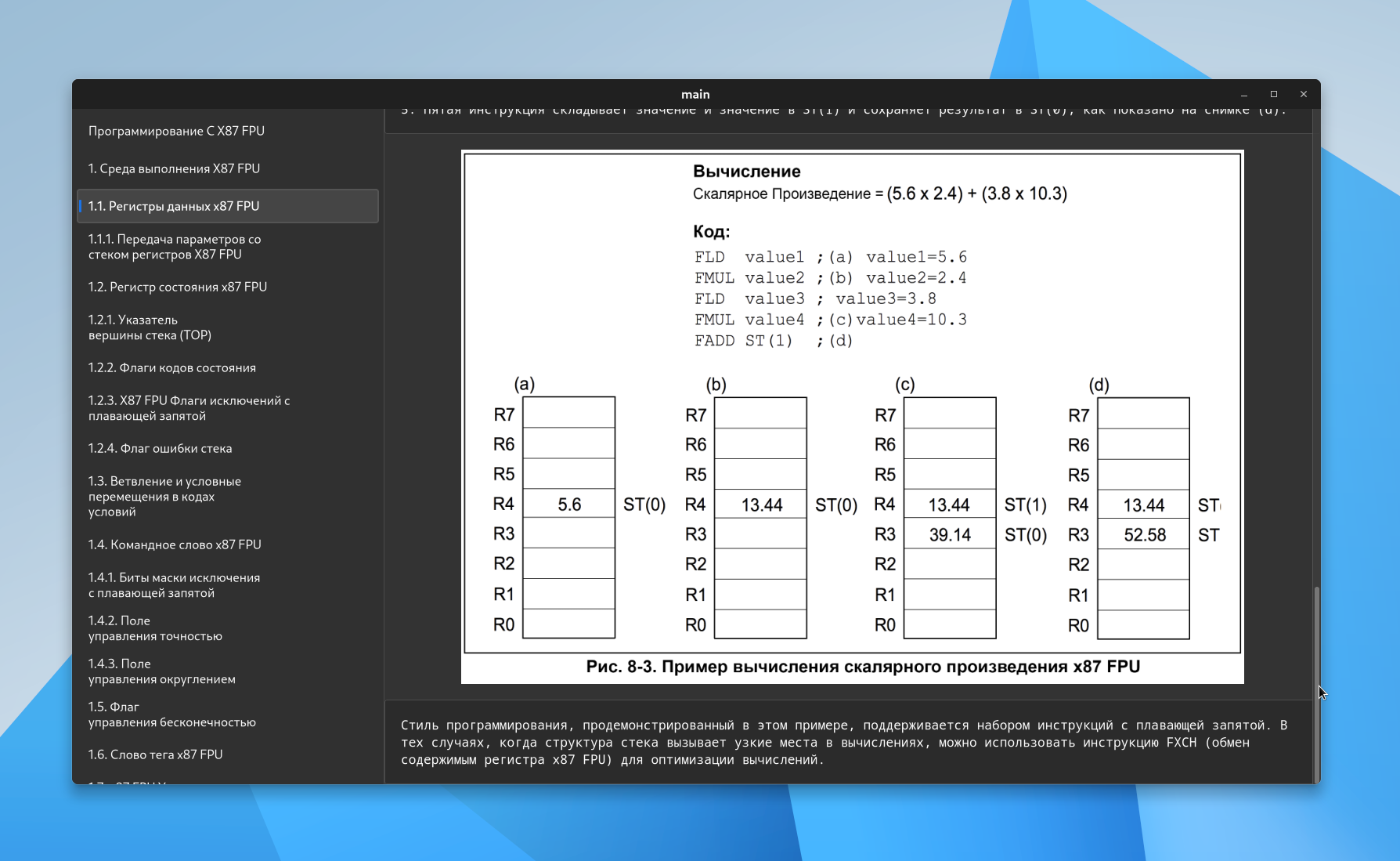


По нажатии на изображение, окно закрывается и открывается главное окно, описанное в функции main\_window(). Здесь можно увидеть отображение виджета GtkStackSidebar и контента из первого элемента стека.



При выборе другого раздела в боковом баре, меняется и содержимое правой части окна, то есть стека. В разделах, где контент не умещается на площади, можно увидеть работу виджета GtkScrolledWindow. В правой части приложения появляется ползунок, который позволяет пролистывать данные по вертикали.





Вывод

В этой работе я изучил бибилотеку создания графических интерфейсов GTK, и на основе предлагаемых ею средств разработал приложение с графическим интерфейсом, отображающее данные по объёмной теме Intel FPU x87, и ознакомился с информацией об этом сопроцессоре и его средствах. Я широко применял на практике гибкие возможности языка C, разрабатывая логику программы, её поведение и способы обработки больших объёмов нестандартной структуры данных. Опыт, который я приобрёл, поможет мне создавать приложения с графическим интерфейсом – более дружелюбные и наглядные для их пользователей.

Исходные коды

Папка src.

header.h

#ifndef TEST\_HEADER\_H

#define TEST\_HEADER\_H

#include <gtk/gtk.h>

#define PACK(X) ((void\*)X)

#define PACK\_2D\_ARRAY(X) ((void\*\*)X)

**typedef** **struct** {

**void** \*list;

**int** size;

} Package;

// pack\_funcs.c

Package\* pack\_new(**void**);

Package\* get\_pack\_with\_numbers\_new(**char**\* str, **char**\*\* list, **int** size);

**void** set\_pack\_with\_numbers(Package\* pack, **char**\* str, **char**\*\* list, **int** size);

**int** scan\_dir\_set\_pack(Package\* pack, **char**\* path);

//// pack\_funcs.c //

// string\_funcs.c

**int** length(**char**\* str);

**int** compare\_strings(**char**\* str1, **char**\* str2);

**void** shake\_sort\_data(**char** \*\*str\_arr, **int** size);

**char**\* glue\_strings(**char**\* str1, **char**\* str2, **char**\* str);

**char**\* string\_append(**char**\* str1, **char**\* str2);

**char**\* get\_string\_from\_num(**int** num);

**char**\* add\_num\_to\_string (**char**\* str, **int** num);

**char**\* set\_string\_from\_num(**char**\* str, **int** num);

**char**\* string\_new();

**char**\* get\_string\_with\_text\_new (**char**\* str);

**char** \*string\_from\_file\_new(**char**\* filename);

**char** \*set\_string\_from\_file(**char**\* filename, **char**\* str);

**char**\* insert\_letter\_to\_string(**char**\* str, **int** pos, **char** letter);

**char**\* split\_long\_string\_with\_interval (**char**\* str, **int** num);

//// string\_funcs.c //

// other\_funsc.c

**void** free\_2d\_array (**void** \*\*arr, **int** size);

**void** shake\_sort\_int (**int** \*arr, **int** size);

**int** pow\_i(**int** a, **unsigned** **int** b);

//// other\_funcs.c //

// set\_page.c

**int** set\_page (GtkStack\* stack, **char**\* dir);

//// set\_page.c

// fill\_stack.c

**int** fill\_stack(GtkStack\* stack);

//// fill\_stack.c

#endif // TEST\_HEADER\_H

makefile

CFLAGS = $$(pkg-config --cflags gtk4)

LIBS = $$(pkg-config --libs gtk4)

h\_f = pack\_funcs.c string\_funcs.c other\_funcs.c

header\_funcs = funcs/\*.c

main\_funcs = set\_page.c fill\_stack.c main.c

app: $(header\_funcs) $(main\_funcs)

gcc $(CFLAGS) -o $@ $^ $(LIBS)

clean:

rm app

main.c

#include <gtk/gtk.h>

#include "header.h"

**static** **void** main\_window (GtkWidget\* widget, gpointer data) {

GtkWidget \*window;

GtkWidget \*grid;

GtkWidget \*stack;

GtkWidget \*stackSidebar;

window = gtk\_application\_window\_new(GTK\_APPLICATION (data));

gtk\_window\_set\_title (GTK\_WINDOW (window), "main");

gtk\_window\_set\_default\_size (GTK\_WINDOW (window), 1600, 900);

gtk\_window\_set\_resizable(GTK\_WINDOW (window), **TRUE**);

gtk\_widget\_show(window);

grid = gtk\_grid\_new();

gtk\_grid\_set\_row\_homogeneous (GTK\_GRID(grid), **TRUE**);

gtk\_grid\_set\_column\_homogeneous (GTK\_GRID(grid), **TRUE**);

stack = gtk\_stack\_new();

fill\_stack(GTK\_STACK(stack));

stackSidebar = gtk\_stack\_sidebar\_new();

gtk\_stack\_sidebar\_set\_stack(GTK\_STACK\_SIDEBAR (stackSidebar), GTK\_STACK(stack));

gtk\_grid\_attach(GTK\_GRID(grid), stackSidebar, 0, 0, 1, 1);

gtk\_grid\_attach(GTK\_GRID(grid), stack, 1, 0, 3, 1);

gtk\_window\_set\_child (GTK\_WINDOW (window), grid);

}

**static** **void** start\_window (GApplication \*app, gpointer \*user\_data) {

GtkWidget \*window;

GtkWidget\* box;

GtkWidget \*pic;

GdkPixbuf \*pixbuf;

window = gtk\_application\_window\_new(GTK\_APPLICATION (app));

gtk\_window\_set\_title (GTK\_WINDOW (window), "start");

gtk\_window\_set\_default\_size (GTK\_WINDOW (window), 1000, 800);

gtk\_window\_set\_resizable(GTK\_WINDOW (window), **FALSE**);

box = gtk\_box\_new (GTK\_ORIENTATION\_VERTICAL, 50);

gtk\_widget\_set\_halign (box, GTK\_ALIGN\_CENTER);

gtk\_widget\_set\_valign (box, GTK\_ALIGN\_START);

gtk\_widget\_set\_margin\_top(box, 50);

pixbuf = gdk\_pixbuf\_new\_from\_file\_at\_size("../Data/0/picture-1.jpg", 640, 360, **NULL**);

pic = gtk\_picture\_new\_for\_pixbuf(pixbuf);

GtkWidget \*bttn;

bttn = gtk\_button\_new();

gtk\_button\_set\_child(GTK\_BUTTON (bttn), pic);

g\_signal\_connect (bttn, "clicked", G\_CALLBACK (main\_window), app);

g\_signal\_connect\_swapped (bttn, "clicked", G\_CALLBACK (gtk\_window\_destroy), window);

gtk\_box\_append(GTK\_BOX(box), bttn);

gtk\_window\_set\_child (GTK\_WINDOW (window), box);

gtk\_widget\_show(window);

}

**int** main (**int** argc, **char** \* argv[]) {

GtkApplication \*app;

**int** status;

app = gtk\_application\_new ("Dima-s.application", G\_APPLICATION\_FLAGS\_NONE);

g\_signal\_connect (app, "activate", G\_CALLBACK (start\_window), **NULL**);

status = g\_application\_run (G\_APPLICATION (app), argc, argv);

g\_object\_unref (app);

**return** status;

}

fill\_stack.c

#include <stdio.h>

#include <stdlib.h>

#include <gtk/gtk.h>

#include "header.h"

**int** fill\_stack(GtkStack\* stack) {

**int** i = 0;

**char** \*data\_path = "../Data/";

Package\* pack = pack\_new();

scan\_dir\_set\_pack(pack, data\_path);

**if** (pack == **NULL**)

{

perror("Package failed\n");

**return** -1;

}

**char**\*\* dir\_list = (**char**\*\*) pack->list;

**int** list\_size = pack->size;

**char**\* new\_path = string\_new();

**for** (i = 0; i < list\_size; i++)

{

new\_path = glue\_strings(data\_path, dir\_list[i], new\_path);

set\_page(stack, new\_path);

}

free\_2d\_array(PACK\_2D\_ARRAY (dir\_list), list\_size);

free(new\_path);

free(pack);

**return** 0;

}

set\_page.c

#include <stdio.h>

#include <stdlib.h>

#include <gtk/gtk.h>

#include "header.h"

**int** set\_page (GtkStack\* stack, **char**\* dir) {

// 1 //

// pack cleared

Package\* pack = pack\_new();

**if** (scan\_dir\_set\_pack(pack, dir) == -1)

{

perror("No such directory\n");

**return** -2;

}

// files cleared

**char**\*\* files = (**char**\*\*) pack->list;

**int** f\_size = pack->size;

set\_pack\_with\_numbers(pack, "picture-", files, f\_size);

// pic\_nums cleared

**int**\* pic\_nums = (**int**\*) pack->list;

**int** p\_size = pack->size;

shake\_sort\_int(pic\_nums, p\_size);

set\_pack\_with\_numbers(pack, "text-", files, f\_size);

// text\_nums cleared

**int**\* text\_nums = (**int** \*) pack->list;

**int** t\_size = pack->size;

shake\_sort\_int(text\_nums, t\_size);

///// 1

// 2 //

GtkWidget \*scrolledWindow;

GtkWidget \*vBox;

GtkWidget \*label;

GtkWidget \*\*pictures;

GtkWidget \*\*frames;

GtkWidget \*\*textViews;

GtkTextBuffer \*textBuf;

GdkPixbuf\* pixbuf;

vBox = gtk\_box\_new (GTK\_ORIENTATION\_VERTICAL, 20);

gtk\_box\_set\_baseline\_position (GTK\_BOX (vBox), GTK\_BASELINE\_POSITION\_CENTER);;

gtk\_widget\_set\_margin\_top(vBox, 20);

// path cleared

**char**\* path = string\_new();

//// 2

// 3 //

**int** i = 0, p = 0, t = 0;

**int** p\_max = \*(pic\_nums + p\_size - 1);

**int** t\_max = \*(text\_nums + t\_size - 1);

// clear text 6

**char**\* text = string\_new();

//### label #################################

path = glue\_strings(dir, "/label.txt", path);

// title cleared

**char**\* title = string\_from\_file\_new (path);

**if** (title == **NULL**) title = get\_string\_with\_text\_new ("-");

label = gtk\_label\_new (title);

**char**\* markup = g\_markup\_printf\_escaped("<b><span font='30px'>%s</span></b>", title);

gtk\_label\_set\_markup (GTK\_LABEL (label), markup);

gtk\_label\_set\_xalign (GTK\_LABEL(label), 0.05);

gtk\_box\_append (GTK\_BOX (vBox), label);

g\_free(markup);

//###\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_###

//###########################################

pictures = (GtkWidget\*\*) calloc (p\_size, **sizeof**(GtkWidget\*));

textViews = (GtkWidget\*\*) calloc (t\_size, **sizeof**(GtkWidget\*));

frames = (GtkWidget\*\*) calloc (t\_size, **sizeof**(GtkWidget\*));

**for** (i = 0, p = 0, t = 0; i <= ((p\_max > t\_max) ? p\_max : t\_max); i++) {

**if** (\*(pic\_nums + p) == i && p < p\_size) {

//### picture #################################

path = glue\_strings(dir, "/picture-", path);

path = add\_num\_to\_string(path, \*(pic\_nums + p));

path = string\_append(path, ".jpg");

pixbuf = gdk\_pixbuf\_new\_from\_file\_at\_scale(path, 1000, 1000, **TRUE**, **NULL**);

\*(pictures + p) = gtk\_picture\_new\_for\_pixbuf(pixbuf);

gtk\_picture\_set\_can\_shrink(GTK\_PICTURE (\*(pictures + p)), **FALSE**);

gtk\_box\_append (GTK\_BOX (vBox), \*(pictures + p));

p++;

//###\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_###

//###########################################

}

**if** (\*(text\_nums + t) == i && t < t\_size) {

//### text #################################

path = glue\_strings(dir, "/text-", path);

path = add\_num\_to\_string(path, \*(text\_nums + t));

path = string\_append(path, ".txt");

text = set\_string\_from\_file (path, text);

**if** (text == **NULL**) text = get\_string\_with\_text\_new ("\0");

\*(textViews + t) = gtk\_text\_view\_new();

textBuf = gtk\_text\_view\_get\_buffer (GTK\_TEXT\_VIEW (\*(textViews + t)));

gtk\_text\_buffer\_set\_text (textBuf, text, -1);

gtk\_text\_view\_set\_left\_margin (GTK\_TEXT\_VIEW(\*(textViews + t)), 20);

gtk\_text\_view\_set\_right\_margin (GTK\_TEXT\_VIEW(\*(textViews + t)), 20);

gtk\_text\_view\_set\_top\_margin (GTK\_TEXT\_VIEW(\*(textViews + t)), 20);

gtk\_text\_view\_set\_bottom\_margin (GTK\_TEXT\_VIEW(\*(textViews + t)), 20);

gtk\_text\_view\_set\_editable(GTK\_TEXT\_VIEW (\*(textViews + t)), **FALSE**);

gtk\_text\_view\_set\_cursor\_visible(GTK\_TEXT\_VIEW (\*(textViews + t)), **FALSE**);

gtk\_text\_view\_set\_wrap\_mode (GTK\_TEXT\_VIEW (\*(textViews + t)), GTK\_WRAP\_WORD\_CHAR);

gtk\_text\_view\_set\_monospace(GTK\_TEXT\_VIEW (\*(textViews + t)), **TRUE**);

\*(frames + t) = gtk\_frame\_new (**NULL**);

gtk\_frame\_set\_child (GTK\_FRAME(\*(frames + t)), \*(textViews + t));

gtk\_box\_append (GTK\_BOX (vBox), \*(frames + t));

t++;

//###\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_###

//###########################################

}

}

//// 3

// 4 //

scrolledWindow = gtk\_scrolled\_window\_new();

gtk\_scrolled\_window\_set\_child(GTK\_SCROLLED\_WINDOW(scrolledWindow), vBox);

split\_long\_string\_with\_interval(title, 50);

gtk\_stack\_add\_titled(stack, scrolledWindow, title, title);

free(pack);

free\_2d\_array (PACK\_2D\_ARRAY (files), f\_size);

free(pic\_nums);

free(text\_nums);

free(path);

free(text);

free(title);

free(pictures);

free(textViews);

free(frames);

**return** 0;

}

Папка funcs.

pack\_funcs.c

#include <stdio.h>

#include <dirent.h> // DIR\*, struct dirent\*, opendir, rewinddir, readdirr, closedir

#include <stdlib.h> // calloc, free

#include <string.h> // strcmp, strcpy

#include "../header.h" // Package

Package\* pack\_new() {

Package\* pack = (Package\*) calloc(1, **sizeof**(Package));

//pack->list = NULL;

//pack->size = 0;

**return** pack;

}

Package\* get\_pack\_with\_numbers\_new(**char**\* str, **char**\*\* list, **int** size) {

**int** i = 0;

**int** amount = 0;

**int**\* numSet = (**int**\*) calloc(0, **sizeof**(**int**));

**for** (i = 0; i < size; i++) {

**if** (length(list[i]) < length(str)) **continue**;

**if** (strncmp(str, list[i], length(str)) == 0) {

amount ++;

numSet = (**int**\*) realloc(numSet, amount \* **sizeof**(**int**));

\*(numSet + amount - 1) = list[i][length(str)] - '0';

}

}

Package\* pack = (Package\*) calloc(1, **sizeof**(Package));

pack->list = PACK(numSet);

pack->size = amount;

**return** pack;

}

**void** set\_pack\_with\_numbers(Package\* pack, **char**\* str, **char**\*\* list, **int** size) {

**int** i = 0;

**int** amount = 0;

**int**\* numSet = (**int**\*) calloc(0, **sizeof**(**int**));

**for** (i = 0; i < size; i++) {

**if** (length(list[i]) < length(str)) **continue**;

**if** (strncmp(str, list[i], length(str)) == 0) {

amount ++;

numSet = (**int**\*) realloc(numSet, amount \* **sizeof**(**int**));

\*(numSet + amount - 1) = list[i][length(str)] - '0';

}

}

pack->list = PACK(numSet);

pack->size = amount;

}

**int** scan\_dir\_set\_pack(Package\* pack, **char**\* path) {

**struct** dirent \*entry;

DIR \*dir;

**long** **int** files\_amount = 0;

**char** \*\*dir\_files;

**int** i = 0;

// открываем директорию Data с данными для страниц

dir = opendir(path);

// если не открылась, программа завершится с ошибкой

**if** (dir == **NULL**) {

perror("Opendir\n");

**return** -1;

}

// считаем количество файлов внутри директории, пропуская специальные файлы

// точка означает текущий каталог, две - предыдущий

// используются для навигации и запуска исполняемых файлов в каталоге через терминал

**while** (entry = readdir(dir)) {

**if** (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0) **continue**;

files\_amount++;

}

// выделяем память под имена найденных файлов (каталогов)

dir\_files = (**char**\*\*) calloc (files\_amount, **sizeof**(**char**\*));

**for** (i = 0; i < files\_amount; i++) {

dir\_files[i] = (**char**\*) calloc (260, **sizeof**(**char**));

}

files\_amount = 0;

// перемещаем указатель в потоке на начало

rewinddir(dir);

// заполняем строковый массив данными из потока

**while** (entry = readdir(dir)) {

**if** (strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0) **continue**;

strcpy(dir\_files[files\_amount], entry->d\_name);

files\_amount++;

}

// сортируем каталоги по реализованному методу

shake\_sort\_data(dir\_files, files\_amount);

closedir(dir);

pack->list = PACK(dir\_files);

pack->size = files\_amount;

**return** 0;

}

string\_funcs.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "../header.h"

**int** length(**char**\* str) {

**char** c;

**int** i = 0;

**if** (str == **NULL**) **return** i;

**while** (c = \*(str + i)) {

**if** (c == '\0') **break**;

i++;

}

**return** i;

}

**int** compare\_strings(**char**\* str1, **char**\* str2) {

**int** i = 0;

**int** result = 0;

**int** length1 = length(str1);

**int** length2 = length(str2);

**int** limit = (length1 > length2) ? length2 : length1;

**for** (i = 0; i < limit + 1; i++) {

**if** (\*(str1 + i) == \*(str2 + i)) **continue**;

**if** (\*(str1 + i) == '-' && \*(str2 + i) >= '0'

&& \*(str2 + i) <= '9') {

result = 2;

**break**;

}

**if** (\*(str2 + i) == '-' && \*(str1 + i) >= '0'

&& \*(str1 + i) <= '9') {

result = 1;

**break**;

}

**if** (\*(str1 + i) > \*(str2 + i))

{

**if** (\*(str2 + i + 1) == '\0'

|| \*(str2 + i + 1) == '-')

{

result = 1;

**break**;

} **else** **if** (\*(str1 + i + 1) >= '0'

&& \*(str1 + i + 1) <= '9')

{

result = 1;

**break**;

} **else** {

result = 2;

**break**;

}

} **else**

**if** (\*(str1 + i) < \*(str2 + i))

{

**if** (\*(str1 + i + 1) == '\0'

|| \*(str1 + i + 1) == '-')

{

result = 2;

**break**;

} **else** **if** (\*(str2 + i + 1) >= '0'

&& \*(str2 + i + 1) <= '9')

{

result = 2;

**break**;

} **else** {

result = 1;

**break**;

}

}

}

**return** result;

}

**void** shake\_sort\_data(**char** \*\*str\_arr, **int** size) {

**int** i = 0;

**int** left = 0;

**int** right = size - 1;

**char** \*buf = calloc(260, **sizeof**(**char**));

**while** (left <= right) {

**for** (i = right; i > left; --i) {

**if** (compare\_strings(\*(str\_arr + i - 1), \*(str\_arr + i)) == 1) {

strcpy(buf, \*(str\_arr + i - 1));

strcpy(\*(str\_arr + i - 1), \*(str\_arr + i));

strcpy(\*(str\_arr + i), buf);

}

}

++left;

**for** (i = left; i < right; ++i) {

**if** (compare\_strings(\*(str\_arr + i), \*(str\_arr + i + 1)) == 1) {

strcpy(buf, \*(str\_arr + i));

strcpy(\*(str\_arr + i), \*(str\_arr + i + 1));

strcpy(\*(str\_arr + i + 1), buf);

}

}

--right;

}

free(buf);

}

**char**\* glue\_strings(**char**\* str1, **char**\* str2, **char**\* str) {

**int** i = 0, j = 0;

**int** l1 = length(str1);

**int** l2 = length(str2);

str = (**char**\*) realloc (str, (l1 + l2 + 1) \* **sizeof**(**char**));

**for** (i = 0; i < l1; i++) {

\*(str + i) = \*(str1 + i);

}

**for** (i = l1, j = 0; j < l2; i++, j++) {

\*(str + i) = \*(str2 + j);

}

\*(str + l1 + l2) = '\0';

**return** str;

}

**char**\* string\_append(**char**\* str1, **char**\* str2) {

**int** i = 0, j = 0;

**int** l1 = length(str1);

**int** l2 = length(str2);

str1 = (**char**\*) realloc (str1, (l1 + l2 + 1) \* **sizeof**(**char**));

**for** (i = l1, j = 0; j < l2; i++, j++) {

\*(str1 + i) = \*(str2 + j);

}

\*(str1 + l1 + l2) = '\0';

**return** str1;

}

**char**\* get\_string\_from\_num(**int** num) {

**int** i = 0;

//if (num != 0) i = 1;

**int** n = num;

**while** (n != 0) {

n = n / 10;

i++;

}

printf("i = %d\n", i);

**char**\* str = (**char**\*) calloc(i + 1, **sizeof**(**char**));

**int** j = 0;

**for** (j = i, i = 0; j > 0; j--, i++) {

\*(str + i) = (num / pow\_i(10, j - 1) % 10 + '0');

}

\*(str + i) = '\0';

**return** str;

}

**char**\* add\_num\_to\_string (**char**\* str, **int** num) {

**int** i = 0, j = 0;

**int** n = num;

**while** (n != 0) {

n = n / 10;

i++;

}

**if** (num % 10 == 0) i++;

str = (**char**\*) realloc(str, (length(str) + i + 1) \* **sizeof**(**char**) );

**for** (j = i, i = length(str); j > 0; j--, i++) {

\*(str + i) = (num / pow\_i(10, j - 1) % 10 + '0');

}

\*(str + i) = '\0';

**return** str;

}

**char**\* set\_string\_from\_num(**char**\* str, **int** num) {

**int** i = 0;

//if (num != 0) i = 1;

**int** n = num;

**while** (n != 0) {

n = n / 10;

i++;

}

printf("i = %d\n", i);

str = (**char**\*) realloc(str, (i + 1) \* **sizeof**(**char**) );

//char\* str = (char\*) calloc(i + 1, sizeof(char));

**int** j = 0;

**for** (j = i, i = 0; j > 0; j--, i++) {

\*(str + i) = (num / pow\_i(10, j - 1) % 10 + '0');

}

\*(str + i) = '\0';

**return** str;

}

**char**\* string\_new() {

**char**\* str = **NULL**;

**return** str;

}

**char**\* get\_string\_with\_text\_new (**char**\* str) {

**int** i = 0;

**char**\* string = (**char**\*) calloc (length(str) + 1, **sizeof**(**char**));

**for** (i = 0; i < length(str); i++) {

\*(string + i) = \*(str + i);

}

\*(string + i) = '\0';

**return** string;

}

**char** \*string\_from\_file\_new(**char**\* filename) {

FILE \*file;

**int** file\_size = 0;

**if** (!(file = fopen(filename, "r"))) {

**return** **NULL**;

}

fseek(file, 0, SEEK\_END);

file\_size = ftell(file);

**char** \*str = (**char**\*) calloc (file\_size, **sizeof**(**char**));

fseek(file, 0, SEEK\_SET);

fread(str, **sizeof**(**char**), file\_size, file);

\*(str + file\_size - 1) = '\0';

fclose(file);

**return** str;

}

**char** \*set\_string\_from\_file(**char**\* filename, **char**\* str) {

FILE \*file;

**int** file\_size = 0;

**if** (!(file = fopen(filename, "r"))) {

free(str);

**return** **NULL**;

}

fseek(file, 0, SEEK\_END);

file\_size = ftell(file);

str = (**char**\*) realloc (str, file\_size \* **sizeof**(**char**));

fseek(file, 0, SEEK\_SET);

fread(str, **sizeof**(**char**), file\_size, file);

\*(str + file\_size - 1) = '\0';

fclose(file);

**return** str;

}

**char**\* insert\_letter\_to\_string(**char**\* str, **int** pos, **char** letter) {

**int** i = 0, j = 0;

**if** (pos >= length(str)) **return** str;

**char**\* res = (**char**\*) calloc(length(str) + 2, **sizeof**(**char**));

**for** (i = 0; i < length(str); i++) {

\*(res + j++) = \*(str + i);

**if** (i == pos) \*(res + j++) = letter;

}

\*(res + j) = '\0';

free(str);

**return** res;

}

**char**\* split\_long\_string\_with\_interval (**char**\* str, **int** num) {

**int** i = 0, j = 0;

**int** limit = 0;

**if** (length(str) > num) {

**for** (i = num - 1; i > limit; i = i - 2)

{

**if** (\*(str + i) == ' ')

{

\*(str + i) = '\n';

limit = i;

i = i + 1 + num;

**if** (i >= length(str)) **break**;

}

**if** (i == limit + 1)

{

str = insert\_letter\_to\_string(str, limit + (num - 1) - 1, '-');

str = insert\_letter\_to\_string(str, limit + (num - 1), '\n');

limit = limit + num + 1;

i = limit + num + 1;

**if** (i >= length(str)) **break**;

}

} // for

}

**return** str;

}

other\_funcs.c

#include <stdio.h>

#include <stdlib.h>

#include "../header.h"

**void** free\_2d\_array (**void** \*\*arr, **int** size) {

**int** i = 0;

**for** (i = 0; i < size; i++) {

free(arr[i]);

}

free(arr);

}

**void** shake\_sort\_int(**int** \*arr, **int** size) {

**int** i = 0;

**int** left = 0;

**int** right = size - 1;

**int** buf = 0;;

**while** (left <= right) {

**for** (i = right; i > left; --i) {

**if** (\*(arr + i - 1) > \*(arr + i)) {

buf = \*(arr + i - 1);

\*(arr + i - 1) = \*(arr + i);

\*(arr + i) = buf;

}

}

++left;

**for** (i = left; i < right; ++i) {

**if** (\*(arr + i) > \*(arr + i + 1)) {

buf = \*(arr + i);

\*(arr + i) = \*(arr + i + 1);

\*(arr + i + 1) = buf;

}

}

--right;

}

}

**int** pow\_i(**int** a, **unsigned** **int** b) {

**int** i = 0;

**int** res = 1;

**for** (i = 0; i < b; i++) {

res \*= a;

}

**return** res;

}