|  |  |
| --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Факультет |  | И |  | Информационные и управляющие системы |
|  |  | шифр |  | наименование |
| Кафедра |  | И5 |  | Информационные системы и программная инженерия |
|  |  | шифр |  | наименование |
| Дисциплина |  | Структуры и организация данных | | |

Курсовая работа

|  |
| --- |
|  |

**Программирование на ЯВУ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | |  | И596 |
| Малинина Полина Альбертовна | | | | | |
| Фамилия И.О. | | | | | |
| **РУКОВОДИТЕЛЬ** | | | | | |
| Орлов О.В. | |  |  | | |
| Фамилия И.О. Подпись | | | | | |
| Оценка |  | | | |  |
| «\_\_\_\_\_» |  | | | | 2020\_г. |

САНКТ-ПЕТЕРБУРГ

2020 г.

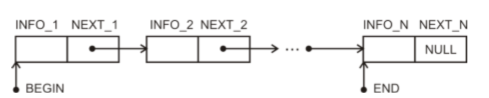
18 вариант

**Задача 1:**

Поля данных: номер поезда, станция назначения, время отправления, время в пути, наличие билетов. Вывести номера поездов и время их отправления в определенный город в заданном временном интервале. Получить информацию о наличии билетов на поезд с определенным номером.

Метод реализации :

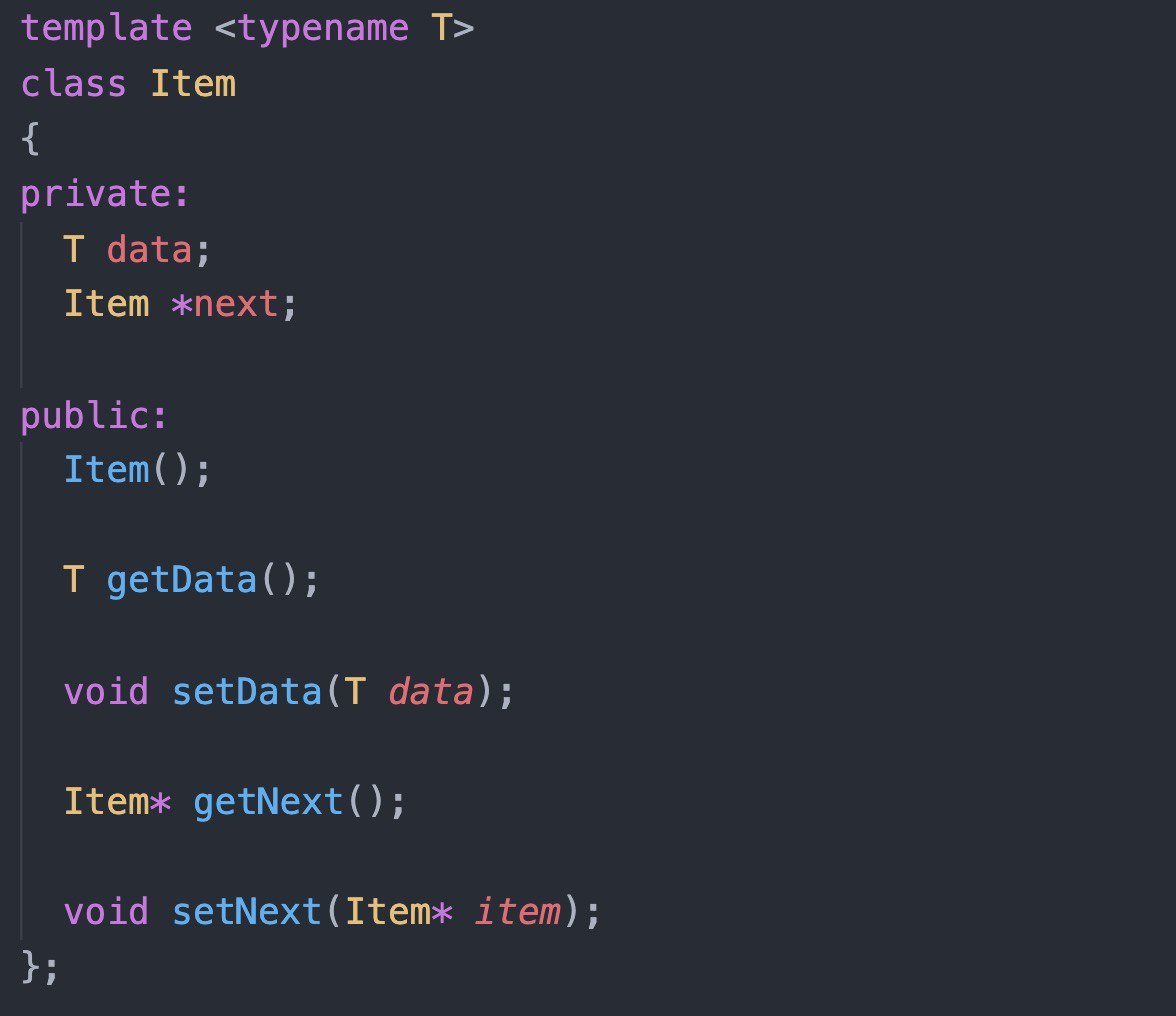
( Однонаправленный линейный список )



Структура данных:



Класс узла списка:



Код программы :

**//app.cpp**

#include <iostream>

#include <string>

#include "./lib/sdl2/include/SDL.h"

#include "./lib/sdl2\_ttf/include/SDL\_ttf.h"

#include "src/utils/utils.h"

#include "src/engine.h"

#include "src/menu/menu.h"

#include "src/input/input.h"

#include "src/core/linkedList/linked\_list.cpp"

#include "src/core/train/train.h"

#include "src/core/trainOrm/train\_orm.h"

#include "src/screens/screen\_config.cpp"

#include "src/screens/mainMenu/main\_menu.h"

#include "src/screens/trains/trains\_screen.h"

#include "src/screens/addTrain/add\_train\_screen.h"

#include "src/screens/queryScreen/query\_screen.h"

// --- App entry point

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

{

int width;

int height;

SDL\_DisplayMode displayMode;

SDL\_Window \*window;

SDL\_Renderer \*renderer;

SDL\_Event event;

bool quit = false;

int screen = Screen::MAIN\_MENU;

TrainORM\* trainOrm = new TrainORM();

ENGINE::init();

window = ENGINE::createWindow("Trains. Author: P. Malinina", &width, &height);

HELPERS::escapeNullptr(window, "Window creation");

renderer = ENGINE::createRenderer(window);

HELPERS::escapeNullptr(renderer, "Renderer creation");

// Screens

MainMenuScreen mainMenuScreen;

TrainsScreen trainsScreen(trainOrm);

AddTrainScreen addTrainScreen(trainOrm);

QueryScreen queryScreen(trainOrm);

while (ENGINE::shouldLoop(&event, &quit))

{

ENGINE::beforeRender(renderer);

// Render App Title

ENGINE::renderText(renderer, 12, 12, "Trains. Author: P. Malinina", 32);

switch (screen)

{

case Screen::MAIN\_MENU:

mainMenuScreen.render(renderer, event, quit, screen);

break;

case Screen::PRINT\_TRAINS:

trainsScreen.render(renderer, event, quit, screen);

break;

case Screen::ADD\_TRAIN:

addTrainScreen.render(renderer, event, quit, screen);

break;

case Screen::QUERY\_TRAINS:

queryScreen.render(renderer, event, quit, screen);

break;

}

ENGINE::afterRender(renderer);

}

delete trainOrm;

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

ENGINE::destroy();

return 0;

}

**//linked\_list.cpp**

#ifndef \_LINKED\_LIST

#define \_LINKED\_LIST

template <typename T>

class Item

{

private:

T data;

Item \*next;

public:

Item();

T getData();

void setData(T data);

Item\* getNext();

void setNext(Item\* item);

};

template <typename T>

class LinkedList

{

private:

Item<T>\* begin;

Item<T>\* end;

int count;

public:

LinkedList();

~LinkedList();

Item<T>\* getBegin();

Item<T>\* getEnd();

int getCount();

void pushBack(T data);

void pushFront(T data);

T popBack();

T popFront();

};

template <typename T>

Item<T>::Item()

{

next = nullptr;

}

template <typename T>

T Item<T>::getData()

{

return data;

}

template <typename T>

void Item<T>::setData(T \_data)

{

data = \_data;

}

template <typename T>

Item<T> \*Item<T>::getNext()

{

return next;

}

template <typename T>

void Item<T>::setNext(Item<T> \*\_next)

{

next = \_next;

}

template <typename T>

LinkedList<T>::LinkedList()

{

begin = nullptr;

end = nullptr;

}

template <typename T>

LinkedList<T>::~LinkedList()

{

Item<T> \*iterator;

while(begin != nullptr)

{

iterator = begin;

begin = begin->getNext();

delete iterator;

}

}

template <typename T>

Item<T>\* LinkedList<T>::getBegin()

{

return begin;

}

template <typename T>

Item<T>\* LinkedList<T>::getEnd()

{

return end;

}

template <typename T>

int LinkedList<T>::getCount()

{

return count;

}

template <typename T>

void LinkedList<T>::pushBack(T data)

{

Item<T>\* item = new Item<T>();

item->setData(data);

if (begin == nullptr)

{

begin = item;

}

else

{

end->setNext(item);

}

end = item;

count += 1;

}

template <typename T>

void LinkedList<T>::pushFront(T data)

{

Item<T>\* item = new Item<T>();

item->setData(data);

if (begin == nullptr)

{

begin = item;

}

else

{

Item<T>\* next = begin;

item->setNext(next);

begin = item;

}

count += 1;

}

template <typename T>

T LinkedList<T>::popBack()

{

Item<T>\* iterator = begin;

if (begin != end)

{

while (iterator->getNext() != end)

{

iterator = iterator->getNext();

}

}

else

{

begin = nullptr;

iterator = nullptr;

}

T data = end->getData();

delete end;

end = iterator;

count -= 1;

return data;

}

template <typename T>

T LinkedList<T>::popFront()

{

T data = begin->getData();

if (begin == end)

{

delete begin;

begin = nullptr;

end = nullptr;

}

else

{

Item<T>\* next = begin->getNext();

delete begin;

begin = next;

}

count -= 1;

return data;

}

#endif

**//train.cpp**

#include <cstring>

#include "train.h"

Train::Train() {}

int Train::getIndex()

{

return index;

}

void Train::setIndex(int \_index)

{

index = \_index;

}

std::string Train::getDestination()

{

return std::string(dest);

}

void Train::setDestination(std::string \_dest)

{

\_dest.copy(dest, 4, 0);

}

int Train::getDepartureTimestamp()

{

return departureTimestamp;

}

void Train::setDepartureTimestamp(int timestamp)

{

departureTimestamp = timestamp;

}

int Train::getPathTimeInMs()

{

return pathTime;

}

void Train::setPathTimeInMs(int \_pathTime)

{

pathTime = \_pathTime;

}

bool Train::getHasTickets()

{

return hasTickets;

}

void Train::setHasTickets(bool \_hasTickets)

{

hasTickets = \_hasTickets;

}

**//train\_orm.cpp**

#include <iostream>

#include <fstream>

#include <cstring>

#include "train\_orm.h"

TrainORM::TrainORM()

{

fileName = "db.dat";

loadFromFile();

}

void TrainORM::addTrain(Train train)

{

list.pushBack(train);

saveToFile();

}

Train \*TrainORM::getTrainByIndex(int index)

{

Item<Train> \*iterator = list.getBegin();

while (iterator != nullptr)

{

Train train = iterator->getData();

if (train.getIndex() == index)

{

return &train;

}

iterator = iterator->getNext();

}

return nullptr;

}

LinkedList<Train> \*TrainORM::getTrains()

{

return &list;

}

LinkedList<Train>\* TrainORM::queryTrainsByDestinationAndTimeInterval(

std::string destination,

int fromTimestamp,

int toTimestamp)

{

LinkedList<Train>\* filteredTrainList = new LinkedList<Train>();

Item<Train> \*iterator = list.getBegin();

while (iterator != nullptr)

{

Train train = iterator->getData();

std::string trainDestination = train.getDestination();

int trainDeparture = train.getDepartureTimestamp();

if (trainDestination == destination && trainDeparture >= fromTimestamp && trainDeparture <= toTimestamp)

{

filteredTrainList->pushBack(train);

}

iterator = iterator->getNext();

}

return filteredTrainList;

}

struct TrainMask

{

int index;

char dest[4];

int departureTimestamp;

int pathTime;

bool hasTickets;

};

void TrainORM::loadFromFile()

{

std::ifstream input;

input.open(fileName);

TrainMask mask;

while (input.read((char \*)&mask, TRAIN\_BYTE\_SIZE))

{

Train train;

train.setIndex(mask.index);

train.setDestination(std::string(mask.dest));

train.setDepartureTimestamp(mask.departureTimestamp);

train.setPathTimeInMs(mask.pathTime);

train.setHasTickets(mask.hasTickets);

list.pushBack(train);

}

input.close();

}

void TrainORM::saveToFile()

{

std::ofstream output;

output.open(fileName);

Item<Train>\* iterator = list.getBegin();

while (iterator != nullptr)

{

Train train = iterator->getData();

TrainMask mask;

mask.index = train.getIndex();

strcpy(mask.dest, train.getDestination().c\_str());

mask.departureTimestamp = train.getDepartureTimestamp();

mask.pathTime = train.getPathTimeInMs();

mask.hasTickets = train.getHasTickets();

output.write((char \*)&mask, TRAIN\_BYTE\_SIZE);

iterator = iterator->getNext();

}

output.close();

}

**//add\_train\_screen.cpp**

#include "add\_train\_screen.h"

#include "../screen\_config.cpp"

AddTrainScreen::AddTrainScreen(TrainORM \*\_trainOrm)

{

trainOrm = \_trainOrm;

inputFieldIndex = 0;

inputs[0].setMaxLength(9);

inputs[1].setMaxLength(3);

inputs[2].setMaxLength(9);

inputs[3].setMaxLength(9);

inputs[4].setMaxLength(1);

}

void AddTrainScreen::validateInputField(int &inputFieldIndex)

{

switch (inputFieldIndex)

{

case 0:

{

int trainIndex;

try

{

trainIndex = std::stoi(inputs[inputFieldIndex].getCurrentValue());

train.setIndex(trainIndex);

inputFieldIndex += 1;

}

catch (std::invalid\_argument e)

{

shouldShowErrorMessage = true;

errorMessage = "Given train index is not number. Please, try again.";

inputs[inputFieldIndex].clear();

}

}

break;

case 1:

{

std::string destination = inputs[inputFieldIndex].getCurrentValue();

if (destination == "")

{

shouldShowErrorMessage = true;

errorMessage = "Given train destination is empty. Please, specify destination.";

}

else

{

train.setDestination(destination);

inputFieldIndex += 1;

}

}

break;

case 2:

{

int departureTimestamp;

try

{

departureTimestamp = std::stoi(inputs[inputFieldIndex].getCurrentValue());

train.setDepartureTimestamp(departureTimestamp);

inputFieldIndex += 1;

}

catch (std::invalid\_argument e)

{

shouldShowErrorMessage = true;

errorMessage = "Given train timestamp is not number. Please, try again.";

inputs[inputFieldIndex].clear();

}

}

break;

case 3:

{

int pathTime;

try

{

pathTime = std::stoi(inputs[inputFieldIndex].getCurrentValue());

train.setPathTimeInMs(pathTime);

inputFieldIndex += 1;

}

catch (std::invalid\_argument e)

{

shouldShowErrorMessage = true;

errorMessage = "Given train path time is not number. Please, try again.";

inputs[inputFieldIndex].clear();

}

}

break;

case 4:

{

bool hasTickets;

try

{

hasTickets = std::stoi(inputs[inputFieldIndex].getCurrentValue());

train.setHasTickets(hasTickets == 1);

inputFieldIndex += 1;

}

catch (std::invalid\_argument e)

{

shouldShowErrorMessage = true;

errorMessage = "You should enter 1, if train has tickets to sell, and 0 if not.";

inputs[inputFieldIndex].clear();

}

}

break;

}

}

void AddTrainScreen::render(SDL\_Renderer \*renderer, SDL\_Event event, bool &quit, int &state)

{

const bool inputFiledIndexInBounds = inputFieldIndex < 5;

const int xOffset = 12;

const int yFieldOffset = 64;

const int fieldTitleFontSize = 22;

std::string fieldTitle;

switch (inputFieldIndex)

{

case 0:

fieldTitle = "Enter train index:";

break;

case 1:

fieldTitle = "Enter train destination (max 3 symbols):";

break;

case 2:

fieldTitle = "Enter train departure timestamp:";

break;

case 3:

fieldTitle = "Enter train path time in ms:";

break;

case 4:

fieldTitle = "Has train tickets to sell? (Enter 1, if has, and 0, if not)";

break;

case 5:

state = Screen::MAIN\_MENU;

(\*trainOrm).addTrain(train);

SDL\_WaitEvent(&event);

reset();

break;

}

if (inputFiledIndexInBounds)

{

ENGINE::renderText(renderer, xOffset, yFieldOffset, fieldTitle, fieldTitleFontSize);

inputs[inputFieldIndex].handleEvent(event);

inputs[inputFieldIndex].render(renderer, xOffset, yFieldOffset + 28);

// Handle enter

if (event.type == SDL\_KEYDOWN)

{

const SDL\_Keycode keycode = event.key.keysym.sym;

if (keycode == SDLK\_RETURN)

{

validateInputField(inputFieldIndex);

}

if (keycode == SDLK\_ESCAPE)

{

state = Screen::MAIN\_MENU;

SDL\_WaitEvent(&event);

reset();

}

}

// Show error message

if (shouldShowErrorMessage && errorMessage.size())

{

if (inputs[inputFieldIndex].getCurrentValue().size() == 0)

{

ENGINE::renderText(renderer, xOffset, yFieldOffset + 44, errorMessage, 16);

}

else

{

shouldShowErrorMessage = false;

}

}

}

}

void AddTrainScreen::reset()

{

inputFieldIndex = 0;

for (int i = 0; i < 5; i++)

{

inputs[i].clear();

}

train = Train();

}

**//main\_menu.cpp**

#include "main\_menu.h"

#include "../screen\_config.cpp"

MainMenuScreen::MainMenuScreen()

{

std::string options[] = {

"1. Print trainsDB",

"2. Add train",

"3. Query trains by destination and time interval",

"4. Exit program"};

menu = Menu(4, options);

}

void MainMenuScreen::render(SDL\_Renderer \*renderer, SDL\_Event event, bool &quit, int &screen)

{

const int xOffset = 12;

ENGINE::renderText(renderer, xOffset, 64, "Choose action:", 22);

menu.render(renderer, xOffset, 100);

if (event.type == SDL\_KEYDOWN)

{

SDL\_Keycode keycode = event.key.keysym.sym;

switch (keycode)

{

case SDLK\_1:

screen = Screen::PRINT\_TRAINS;

SDL\_WaitEvent(&event);

break;

case SDLK\_2:

screen = Screen::ADD\_TRAIN;

SDL\_WaitEvent(&event);

break;

case SDLK\_3:

screen = Screen::QUERY\_TRAINS;

SDL\_WaitEvent(&event);

break;

case SDLK\_4:

quit = true;

break;

}

}

}

**//query\_screen.cpp**

#include "query\_screen.h"

#include "../screen\_config.cpp"

#include "../../engine.h"

QueryScreen::QueryScreen(TrainORM \*orm)

{

trainOrm = orm;

inputFieldIndex = 0;

inputs[0].setMaxLength(3);

inputs[1].setMaxLength(9);

inputs[2].setMaxLength(9);

}

void QueryScreen::validateInputField(int &inputFieldIndex)

{

switch (inputFieldIndex)

{

case 0:

{

std::string destination = inputs[inputFieldIndex].getCurrentValue();

if (destination == "")

{

shouldShowErrorMessage = true;

errorMessage = "Given train destination is empty. Please, specify destination.";

}

else

{

dest = destination;

inputFieldIndex += 1;

}

}

break;

case 1:

{

int \_fromInterval;

try

{

\_fromInterval = std::stoi(inputs[inputFieldIndex].getCurrentValue());

fromInterval = \_fromInterval;

inputFieldIndex += 1;

}

catch (std::invalid\_argument e)

{

shouldShowErrorMessage = true;

errorMessage = "Given interval is not number. Please, try again.";

inputs[inputFieldIndex].clear();

}

}

break;

case 2:

{

int \_toInterval;

try

{

\_toInterval = std::stoi(inputs[inputFieldIndex].getCurrentValue());

toInterval = \_toInterval;

inputFieldIndex += 1;

}

catch (std::invalid\_argument e)

{

shouldShowErrorMessage = true;

errorMessage = "Given interval time is not number. Please, try again.";

inputs[inputFieldIndex].clear();

}

}

break;

}

}

void QueryScreen::render(SDL\_Renderer \*renderer, SDL\_Event event, bool &quit, int &screen)

{

const bool inputFiledIndexInBounds = inputFieldIndex < 3;

const int xOffset = 12;

const int yFieldOffset = 64;

const int fieldTitleFontSize = 22;

std::string fieldTitle;

switch (inputFieldIndex)

{

case 0:

fieldTitle = "Enter train destination (max 3 symbold):";

break;

case 1:

fieldTitle = "Enter 'fromInterval' as timestamp:";

break;

case 2:

fieldTitle = "Enter 'toInterval' as timestamp:";

break;

case 3:

trainList.setList(trainOrm->queryTrainsByDestinationAndTimeInterval(dest, fromInterval, toInterval));

break;

}

if (inputFiledIndexInBounds)

{

ENGINE::renderText(renderer, xOffset, yFieldOffset, fieldTitle, fieldTitleFontSize);

inputs[inputFieldIndex].handleEvent(event);

inputs[inputFieldIndex].render(renderer, xOffset, yFieldOffset + 28);

// Handle enter

if (event.type == SDL\_KEYDOWN)

{

const SDL\_Keycode keycode = event.key.keysym.sym;

if (keycode == SDLK\_RETURN)

{

validateInputField(inputFieldIndex);

}

if (keycode == SDLK\_ESCAPE)

{

screen = Screen::MAIN\_MENU;

SDL\_WaitEvent(&event);

reset();

}

}

// Show error message

if (shouldShowErrorMessage && errorMessage.size())

{

if (inputs[inputFieldIndex].getCurrentValue().size() == 0)

{

ENGINE::renderText(renderer, xOffset, yFieldOffset + 44, errorMessage, 16);

}

else

{

shouldShowErrorMessage = false;

}

}

}

else

{

ENGINE::renderText(renderer, xOffset, yFieldOffset, "Filtered trains:", 24);

trainList.render(renderer, xOffset, yFieldOffset + 28);

// Handle quit

if (event.type == SDL\_KEYDOWN)

{

const SDL\_Keycode keycode = event.key.keysym.sym;

if (keycode == SDLK\_ESCAPE)

{

screen = Screen::MAIN\_MENU;

SDL\_WaitEvent(&event);

reset();

}

}

}

}

void QueryScreen::reset()

{

inputFieldIndex = 0;

for (int i = 0; i < 3; i++)

{

inputs[i].clear();

}

dest = "";

fromInterval = -1;

toInterval = -1;

}

**//trains\_screen.cpp**

#include "trains\_screen.h"

#include "../screen\_config.cpp"

#include "../../engine.h"

TrainsScreen::TrainsScreen(TrainORM \*orm)

{

trainOrm = orm;

trainList.setList(trainOrm->getTrains());

}

void TrainsScreen::render(SDL\_Renderer \*renderer, SDL\_Event event, bool &quit, int &screen)

{

const int xOffset = 12;

ENGINE::renderText(renderer, xOffset, 64, "Trains:", 24);

trainList.render(renderer, xOffset, 100);

if (event.type == SDL\_KEYDOWN)

{

const SDL\_Keycode keycode = event.key.keysym.sym;

if (keycode == SDLK\_ESCAPE)

{

screen = Screen::MAIN\_MENU;

}

}

}

**//train\_list.cpp**

#include <string>

#include "train\_list.h"

#include "../screens/screen\_config.cpp"

#include "../engine.h"

Item<Train> \*getIteratorFromPage(LinkedList<Train> \*list, int page, int limit)

{

Item<Train> \*iterator = list->getBegin();

int count = 0;

while (iterator != nullptr)

{

count += 1;

int \_page = count / limit;

if (\_page == page)

{

return iterator;

}

iterator = iterator->getNext();

}

return nullptr;

}

void TrainList::setList(LinkedList<Train> \*list)

{

trainList = list;

page = 0;

}

void TrainList::nextPage()

{

int pagesCount = getPagesCount();

page += 1;

if (page + 1 > pagesCount)

{

page = pagesCount - 1;

}

}

void TrainList::prevPage()

{

page -= 1;

if (page < 0)

{

page = 0;

}

}

void TrainList::setPage(int \_page)

{

if (page < getPagesCount() && page > -1)

{

page = \_page;

}

}

int TrainList::getPagesCount()

{

int pagesCount = (trainList->getCount() / limit) + (trainList->getCount() % limit != 0 ? 1 : 0);

return pagesCount;

}

void TrainList::render(SDL\_Renderer \*renderer, int x, int y)

{

if (trainList->getCount() == 0)

{

return;

}

int index = 0;

Item<Train> \*iterator = getIteratorFromPage(trainList, page, limit);

while (iterator != nullptr)

{

if (index + 1 > limit)

{

break;

}

Train train = iterator->getData();

std::string row = "";

row += std::to\_string(train.getIndex()) + " ";

row += train.getDestination() + " ";

row += std::to\_string(train.getDepartureTimestamp()) + " ";

row += std::to\_string(train.getPathTimeInMs()) + " ";

row += std::to\_string(train.getHasTickets());

ENGINE::renderText(renderer, x, y + index \* 20, row, 16);

iterator = iterator->getNext();

index += 1;

}

// Render pagination info

ENGINE::renderText(renderer, x, y + 100, "Pages count: " + std::to\_string(getPagesCount()), 14);

ENGINE::renderText(renderer, x, y + 120, "<-", 14);

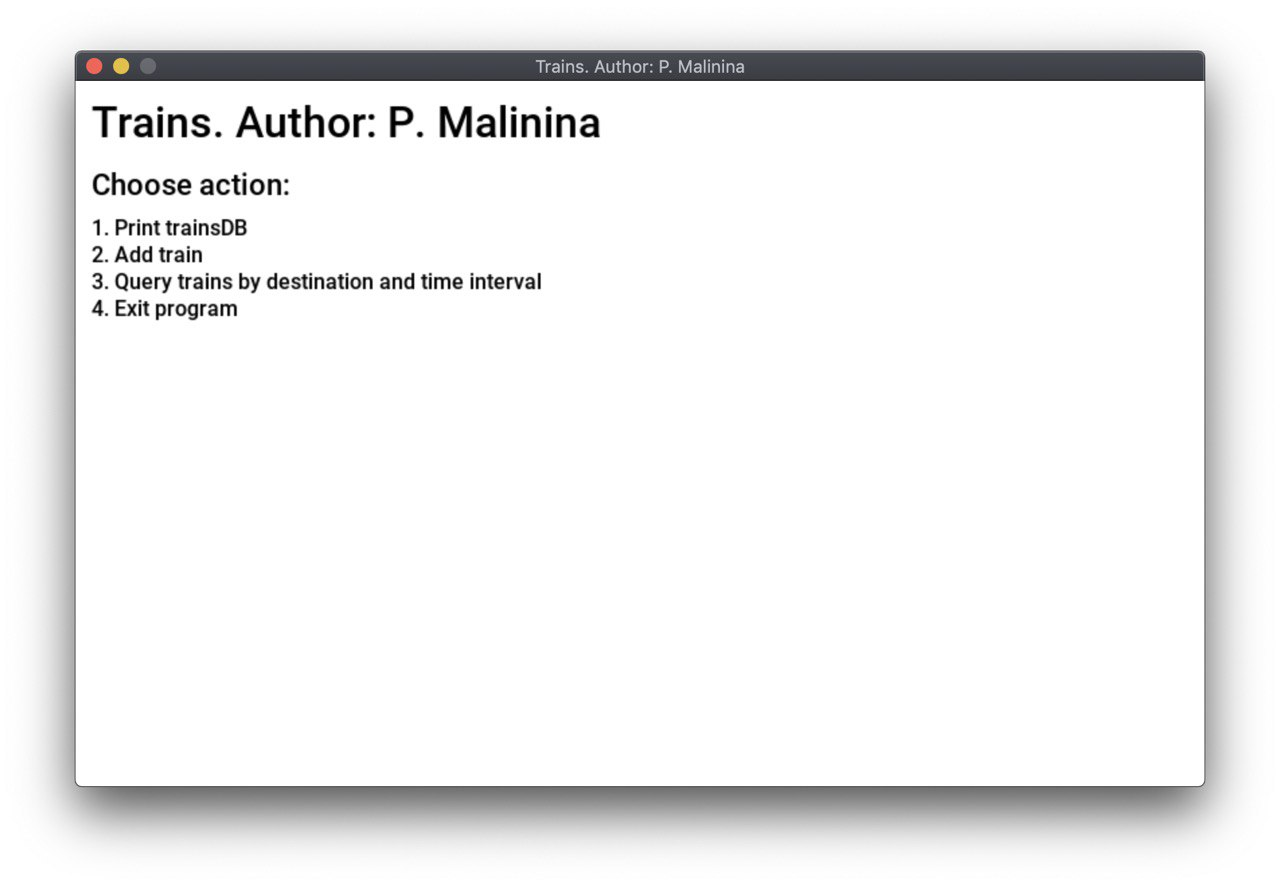
ENGINE::renderText(renderer, x + 18, y + 120, std::to\_string(page + 1), 14);

ENGINE::renderText(renderer, x + 32, y + 120, "->", 14);

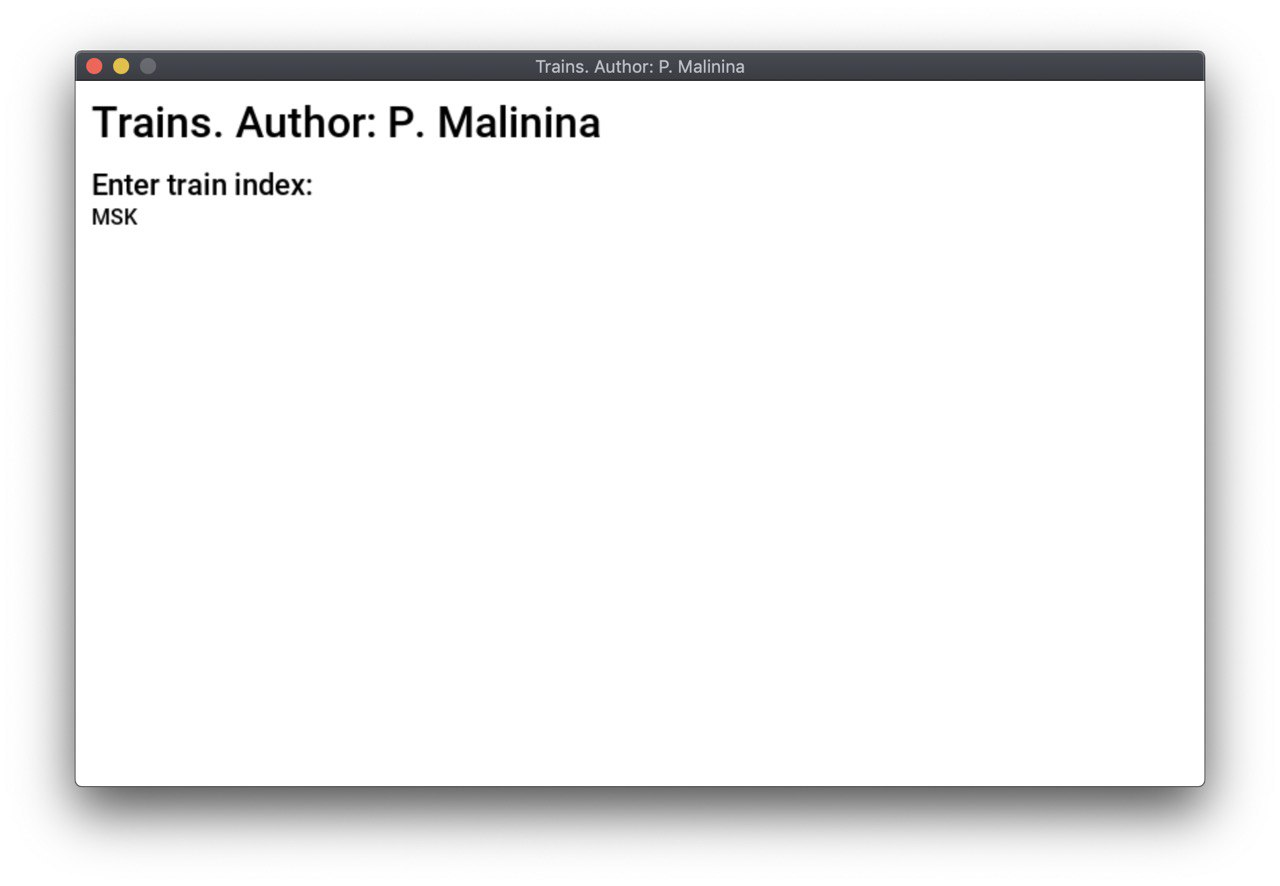
}

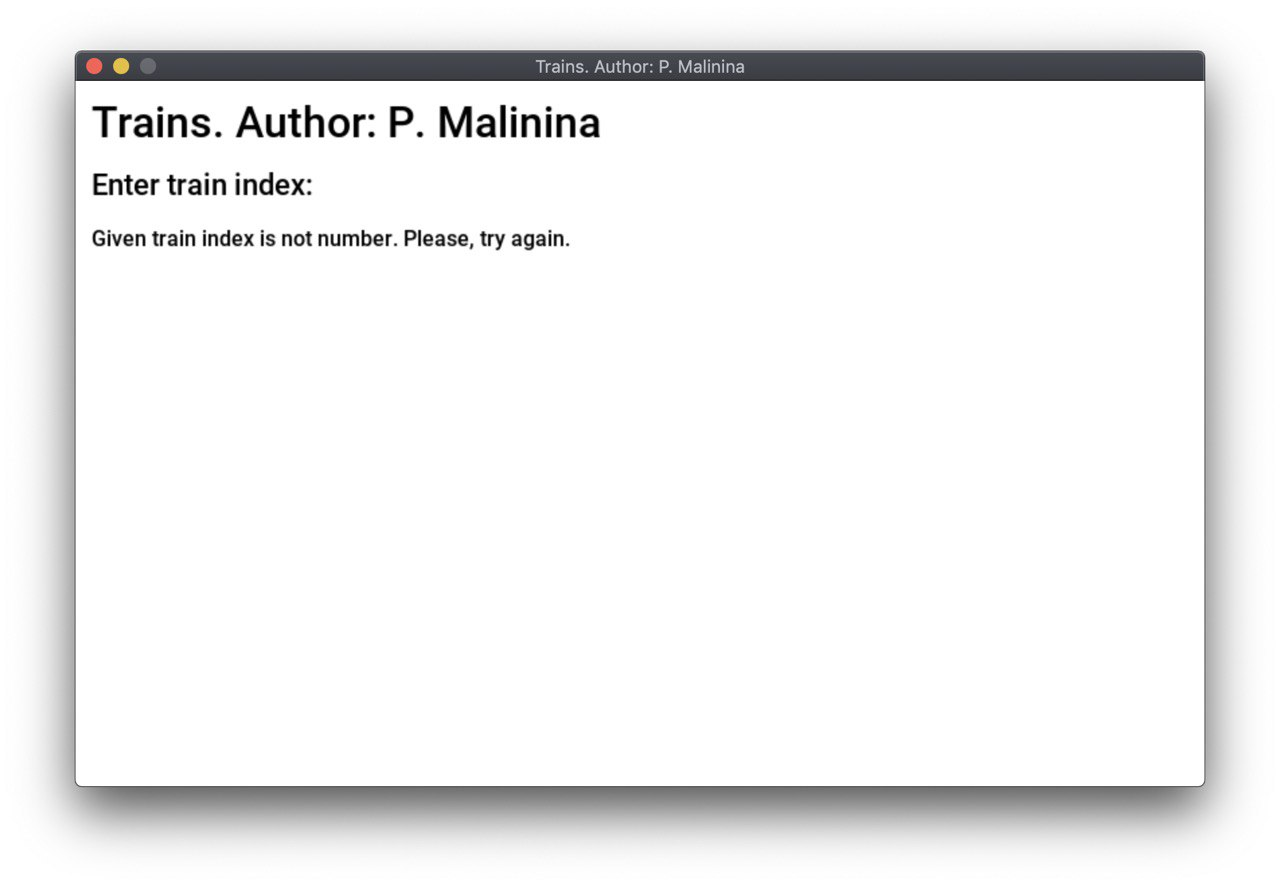
**Результат работы программы:**

1. Главное меню

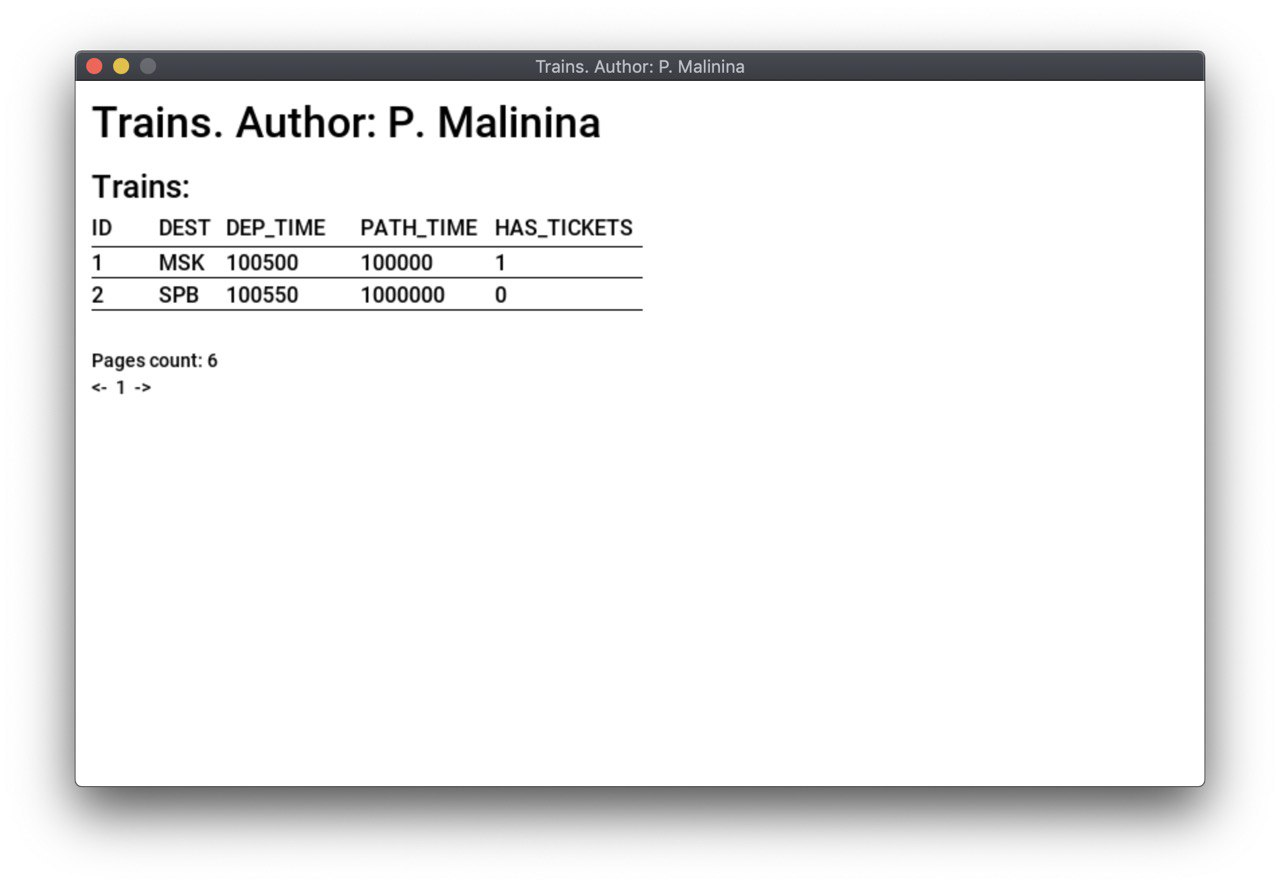


1. Создание базы в файле





4)Просмотр списка



5)Выбор по временному интервалу и станции прибытия

