|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Стурктуры»** | |
|  | |
| Выполнил студент группы ИКБО-07-22 | Козлов К.И. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

получение навыков по реализации многоэлементных структур данных задачи С++

получение навыков по реализации многоэлементных данных средствами структуры данных – таблица

# **Постановка задачи**

1. Разработать набор операций для управления таблицей, созданной на основе динамического массива. Структура записи элемента таблицы определена вариантом индивидуального задания.
   1. Создать проект, добавить заголовочный файл. Включить в заголовочный файл:
      1. разработанный тип данных, определяющий структуру элемента таблицы (записи).
      2. определенная на базе типа struct структура хранения данных по таблице, содержащая: размер таблицы – n, массив из n элементов типа записи варианта.
      3. реализованная функция вывода таблицы на экран.
      4. операции варианта задания, оформленные как функции с параметрами.
   2. Для заполнения отдельной записи с клавиатуры разработать функцию, которая принимает в качестве параметра пустую запись, а возвращает заполненную.
   3. Наполнение таблицы данными должна выполнять функция вставки или добавления записи, включенная в список операций варианта. Эта функция должна приять через параметр заполненную запись, выполнить действие по вставке или добавления, согласно алгоритму варианта этой операции.
2. Разработать программу, демонстрирующую работу всех функций с массивом или записей. Программа должна позволять пользователю непрерывно выполнять операции над таблицей в произвольном порядке.
3. Разработать набор тестовых данных по наполнению таблицы. Включить в меню программы возможность автоматического ввода разработанных тестовых данных.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Требования к упражнениям:

Вариант №18. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | Учет проданных на поезд билетов. Структура элемента: номер поезда, номер вагона, номер места, дата отправления, стоимость билета. Таблица хранит данные по всем вагонам поезда.  Операции   1. Добавить запись о продаже билета в таблицу. 2. Определить количество билетов, проданных на поезд заданного номера и дате отправления. 3. Удалить запись из таблицы по проданному билету. 4. Вывести сведения о количестве проданных билетов в каждый вагон поезда. |

# **Решение**

В данной практической работе была описана структура Ticket. Структура билета, хранящая в себе данные о номере поезда, вагоне, месте и времени отправления вместе с ценой билета.

Структура – тип определяемый пользователем. Этот тип данных используется для представления в программе объектов, обладающих несколькими свойствами возможно различного типа. Например, объект ***Книга***, может быть определена свойствами: фамилия автора, название книги, год издания, издательство. Структура позволяет представить объект, обладающий несколькими свойствами, одной переменной.

Сведения по таблицам

Таблица – это массив, элемент которого имеет сложную структуру, реализуемую в языке С++ посредством типа struct. Элемент таблицы определяет объект предметной области, который характеризуется несколькими свойствами различного типа. Например, объект Книга, в библиотечной системе может быть определен следующим набором свойств: ISBN,Автор, Название, Издательство, Год издания, Количество страниц, Цена. Элементы таблицы содержат поле – ключ, значение которого однозначно идентифицирует данные элемента. Так ISBN – ключ для данной книги.

Таблица может быть реализована на внешнем носителе в форме файла, содержащего записи одного типа. Такие файлы состоят из записей фиксированной длины и чаще реализуются как бинарные.

Таблицу можно реализовать контейнером vector библиотеки STL, в котором в качестве элемента выступает тип данных struct.

Для решения упражнения были описаны несколько структур и написано несколько функций: структура Table используется для упорядоченного хранения записей о нарушении ПДД, структура PDD используется для хранения элементов записей(статьи, даты, и т.д.), функция inp\_PDD() – функция ввода значений элементов записи, show\_PDD() – для вывода записи на экран, show\_table() – для вывода всей таблицы на экран.

Функция ShowTickets() имеет 4 парамера: List – указатель на динамический массив структур Ticket, целочисленную переменную длинны этого массива, целочисленная переменная времени отправления и целочисленную переменную номера поезда.

Эта функция выводит количество билетов по заданному номеру поезда и времени отправления.

|  |
| --- |
| int ShowTickets(Ticket\* List, int length, int trainNumber, int d, int m, int y)  {  int count = 0;  for (int i = 0; i < length; i ++)  {  if ((List[i].TrainNumber == trainNumber) && matchDate(List[i].dateOfDep, d, m, y))  count += 1;  }  return(count);  } |

Функция IsTicket() имеет все параметры функции ShowTickets() и дополнительные: указатели на целочисленные переменные цены билета, места в вагоне и номер вагона и целочисленные переменные даты отправления.

Функция проверяет наличие билета в массиве всех билетов.

|  |
| --- |
| bool IsTicket(Ticket\* List, int length, int\* currPrice, int\* currWagon, int\* currSeat, int\* currTrainNumber, int d, int m, int y)  {  for (int i = 0; i < length; i++)  {  if (List[i].TrainNumber == \*currTrainNumber && List[i].Wagon == \*currWagon && List[i].Seat == \*currSeat && List[i].CostOfTicket == \*currPrice && matchDate(List[i].dateOfDep, d, m, y))  {  return(true);  }  }  return(false);  } |

Функция BuyTicket () имеет такие же параметры как и функция IsTicket ().

Функция добавляет в динамический массив билет по заданным его параметрам (реализация покупки билета).

|  |
| --- |
| Ticket\* BuyTicket(Ticket\* List, int length, int\* currPrice, int\* currWagon, int\* currSeat, int\* currTrainNumber, int d, int m, int y)  {  Ticket\* temp = (Ticket\*)realloc(List, (sizeof(Ticket) \* (length)));  if (temp == NULL)  return (NULL);  List = temp;  List[length - 1] = Ticket(\*currTrainNumber, \*currWagon, \*currSeat, \*currPrice, d, m, y);  return(List);  } |
|  |

Функция SellTicket() имеет такие же параметры как и функция IsTicket ().

Функция удаляет из динамического массива билет по заданным его параметрам (реализация продажи билета).

|  |
| --- |
| Ticket\* SellTicket(Ticket\* List, int length, int\* currPrice, int\* currWagon, int\* currSeat, int\* currTrainNumber, int d, int m, int y)  {  for (int i = 0; i < length; i++)  {  if (List[i].TrainNumber == \*currTrainNumber && List[i].Wagon == \*currWagon && List[i].Seat == \*currSeat && List[i].CostOfTicket == \*currPrice && matchDate(List[i].dateOfDep, d, m, y))  {  while (i != length - 1)  {  std::swap(List[i], List[i + 1]);  i++;  }  length--;  Ticket\* temp = (Ticket\*)realloc(List, sizeof(Ticket) \* length);  if (temp != NULL)  {  List = temp;  return(temp);  }  else  {  return (NULL);  }  }  }  } |
|  |

Функция OutTakenTicket() имеет параметры: целочисленная переменная длины массива уникальных номеров вагонов, длинна массива билетов, указатель на динамический массив билетов и указатель на динамический массив уникальных номеров вагонов.

Функция выводит все занятые места(купленные билеты) для каждого вагона в поезде.

|  |
| --- |
| void OutTakenSeats(int wagonLength, int length, Ticket\* List, int\* wagons)  {  for (int i = 0; i < wagonLength; i++)  {  for (int j = 0; j < length; j++)  {  if (List[j].Wagon == wagons[i])  {  std::cout << "Train Number " << List[j].TrainNumber << std::endl;  std::cout << "Wagon " << List[j].Wagon << std::endl;  std::cout << "Seat " << List[j].Seat << std::endl;  std::cout << "Cost " << List[j].CostOfTicket << std::endl;  std::cout << "Time " << List[j].dateOfDep.day << ':' << List[j].dateOfDep.month << ':' << List[j].dateOfDep.year << std::endl;  }  }  std::cout << std::endl;  }  } |

При запуске программы пользователь увидит начальный интерфейс программы



Рисунок 1. Интерфейс программы

# **4. Тестирование**

Протестируем работу программы. Для этого введем 2 – запись (покупка) билета, и затем введем 1. Далее укажем параметры лишь одного введенного до этого билета.

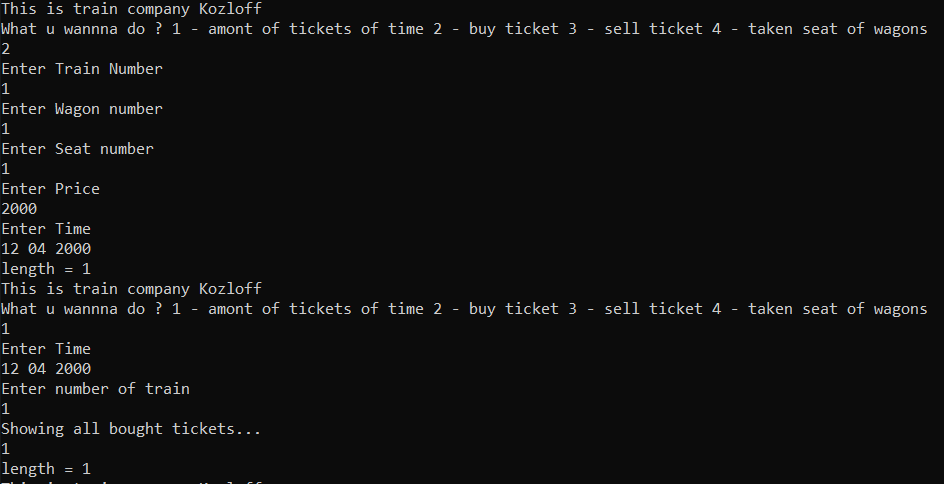


Рисунок 2. Проверка корректности ввода новой записи

Проверим корректность удаления. Введем значения одного из билетов и заново введем 1.

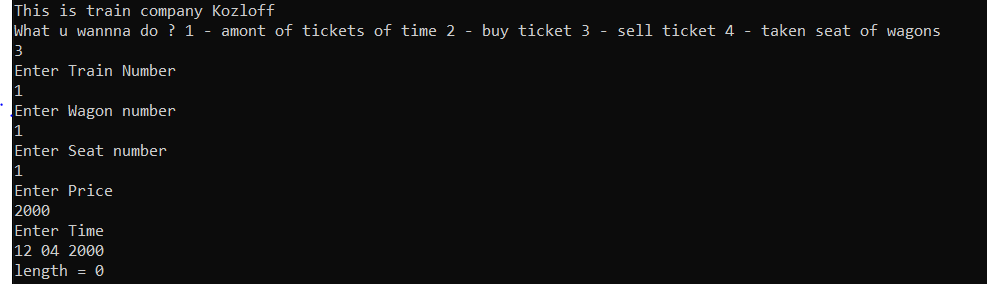


Рисунок 3. Проверка правильности выбора записей по статье

Правильность вывода всех занятых мест в вагонах. Для этого добавим билеты и выберем пункт 4 в меню.

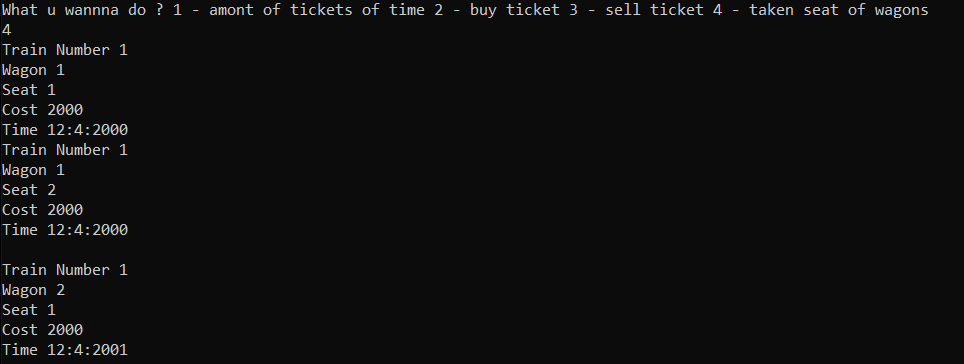


Рисунок 4. Проверка корректности удаления записей по номеру авто

# **5. Выводы**

В результате выполнения работы я:

1. Освоил навыки по реализации многоэлементных структур данных задачи С++
2. Освоил навыки по реализации многоэлементных данных средствами структуры данных – таблица

# **6. Исходный код программы**

|  |
| --- |
| #include <iostream>  struct Date  {  int day;  int month;  int year;  Date()  {  day = month = year = 0;  }  Date(int d, int m, int y)  {  day = d;  month = m;  year = y;  }  };  struct Ticket  {  int TrainNumber;  int CostOfTicket;  int Wagon;  int Seat;  Date dateOfDep;  Ticket(int number, int i, int j, int UserCostOfTicket, int d, int m, int y)  {  TrainNumber = number;  Wagon = i;  Seat = j;  CostOfTicket = UserCostOfTicket;  dateOfDep.day = d;  dateOfDep.month = m;  dateOfDep.year = y;  }  };  bool matchDate(Date dateOfDep, int d, int m, int y)  {  if (dateOfDep.day != d || dateOfDep.month != m || dateOfDep.year != y)  {  return (false);  }  return (true);  }  int ShowTickets(Ticket\* List, int length, int trainNumber, int d, int m, int y)  {  int count = 0;  for (int i = 0; i < length; i ++)  {  if ((List[i].TrainNumber == trainNumber) && matchDate(List[i].dateOfDep, d, m, y))  count += 1;  }  return(count);  }  bool IsTicket(Ticket\* List, int length, int\* currPrice, int\* currWagon, int\* currSeat, int\* currTrainNumber, int d, int m, int y)  {  for (int i = 0; i < length; i++)  {  if (List[i].TrainNumber == \*currTrainNumber && List[i].Wagon == \*currWagon && List[i].Seat == \*currSeat && List[i].CostOfTicket == \*currPrice && matchDate(List[i].dateOfDep, d, m, y))  {  return(true);  }  }  return(false);  }  Ticket\* BuyTicket(Ticket\* List, int length, int\* currPrice, int\* currWagon, int\* currSeat, int\* currTrainNumber, int d, int m, int y)  {  Ticket\* temp = (Ticket\*)realloc(List, (sizeof(Ticket) \* (length)));  if (temp == NULL)  return (NULL);  List = temp;  List[length - 1] = Ticket(\*currTrainNumber, \*currWagon, \*currSeat, \*currPrice, d, m, y);  return(List);  }  Ticket\* SellTicket(Ticket\* List, int length, int\* currPrice, int\* currWagon, int\* currSeat, int\* currTrainNumber, int d, int m, int y)  {  for (int i = 0; i < length; i++)  {  if (List[i].TrainNumber == \*currTrainNumber && List[i].Wagon == \*currWagon && List[i].Seat == \*currSeat && List[i].CostOfTicket == \*currPrice && matchDate(List[i].dateOfDep, d, m, y))  {  while (i != length - 1)  {  std::swap(List[i], List[i + 1]);  i++;  }  length--;  Ticket\* temp = (Ticket\*)realloc(List, sizeof(Ticket) \* length);  if (temp != NULL)  {  List = temp;  return(temp);  }  else  {  return (NULL);  }  }  }  }  void OutTakenSeats(int wagonLength, int length, Ticket\* List, int\* wagons)  {  for (int i = 0; i < wagonLength; i++)  {  for (int j = 0; j < length; j++)  {  if (List[j].Wagon == wagons[i])  {  std::cout << "Train Number " << List[j].TrainNumber << std::endl;  std::cout << "Wagon " << List[j].Wagon << std::endl;  std::cout << "Seat " << List[j].Seat << std::endl;  std::cout << "Cost " << List[j].CostOfTicket << std::endl;  std::cout << "Time " << List[j].dateOfDep.day << ':' << List[j].dateOfDep.month << ':' << List[j].dateOfDep.year << std::endl;  }  }  std::cout << std::endl;  }  }  int main()  {  Ticket\* List = (Ticket\*)malloc(sizeof(Ticket));  if (List == NULL)  return 1;  int Control;  int length = 0;  int wagonLength = 0;  int\* wagons = (int\*)malloc(sizeof(int) \* wagonLength);  do  {  std::cout << "This is train company Kozloff" << std::endl;  std::cout << "What u wannna do ? 1 - amont of tickets of time 2 - buy ticket 3 - sell ticket 4 - taken seat of wagons" << std::endl;  std::cin >> Control;  switch (Control)  {  case 1:  {  int d, m, y;  int num;  std::cout << "Enter Time" << std::endl;  std::cin >> d >> m >> y;  std::cout << "Enter number of train" << std::endl;  std::cin >> num;  std::cout << "Showing all bought tickets..." << std::endl;  std::cout << ShowTickets(List, length, num, d, m, y) << std::endl;  std::cout << "length = " << length << std::endl;  break;  }  case 2:  {  int\* currPrice = new int;  int\* currWagon = new int;  int\* currSeat = new int;  int\* currTrainNumber = new int;  int d, m, y;  std::cout << "Enter Train Number" << std::endl;  std::cin >> \*currTrainNumber;  std::cout << "Enter Wagon number" << std::endl;  std::cin >> \*currWagon;  std::cout << "Enter Seat number" << std::endl;  std::cin >> \*currSeat;  std::cout << "Enter Price" << std::endl;  std::cin >> \*currPrice;  std::cout << "Enter Time" << std::endl;  std::cin >> d >> m >> y;  length += 1;  List = BuyTicket(List, length, currPrice, currWagon, currSeat, currTrainNumber, d, m, y);  bool flag = true;  for (int i = 0; i < wagonLength; i++)  {  if (wagons[i] == \*currWagon)  {  flag = false;  break;  }  }  if (flag)  {  wagonLength++;  wagons = (int\*)realloc(wagons, sizeof(int) \* wagonLength);  wagons[wagonLength - 1] = \*currWagon;  }  std::cout << "length = " << length << std::endl;  delete currPrice;  delete currTrainNumber;  delete currSeat;  delete currWagon;  break;  }  case 3:  {  int\* currPrice = new int;  int\* currWagon = new int;  int\* currSeat = new int;  int\* currTrainNumber = new int;  int d, m, y;  std::cout << "Enter Train Number" << std::endl;  std::cin >> \*currTrainNumber;  std::cout << "Enter Wagon number" << std::endl;  std::cin >> \*currWagon;  std::cout << "Enter Seat number" << std::endl;  std::cin >> \*currSeat;  std::cout << "Enter Price" << std::endl;  std::cin >> \*currPrice;  std::cout << "Enter Time" << std::endl;  std::cin >> d >> m >> y;  if (IsTicket(List, length, currPrice, currWagon, currSeat, currTrainNumber, d, m, y))  {  List = SellTicket(List, length, currPrice, currWagon, currSeat, currTrainNumber, d, m, y);  length--;  if (length < 0)  {  length++;  }  }  delete currPrice;  delete currTrainNumber;  delete currSeat;  delete currWagon;  std::cout << "length = " << length << std::endl;  break;  }  case 4:  {  OutTakenSeats(wagonLength, length, List, wagons);  break;  }  }  } while (Control == 1 || Control == 2 || Control == 3 || Control == 4);  free(List);  } |