Департамент профессионального образования Томской области

Областное государственное бюджетное профессиональное образовательное учреждение среднего профессионального образования

**«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»**

Специальность 09.02.07 «Информационные системы и программирование»

Отчёт о лабораторно-практической работе №11 по дисциплине  
«Основы алгоритмизации и программирования»

Структуры

Студенты

«число» год                            Чуприков Д. Э.

Преподаватель

«число» месяц год оценка                            Жабин Д. И.

Томск — 2022

**Оглавление**

[1 Постановка задачи 3](#_Toc81760271)

[2 Теоретические сведения 4](#_Toc81760272)

[3 Ход работы 5](#_Toc81760273)

[4 Результаты 6](#_Toc81760274)

[Приложение А. Исходный текст 7](#_Toc81760275)

[Приложение Б. Ответы на вопросы 9](#_Toc81760276)

# Постановка задачи

Цель работы: научиться писать программы с использованием структур на языке C++ в среде разработки Visual Studio 2022

Задачи:

1. Создать проект в Visual Studio.
2. Написать программу с использованием структур.
3. Выполнить программу.

Задание №1

В аптечном складе хранятся лекарства. В специальной ведомости содержаться сведения о лекарствах: наименование препарата, его количество, цена и допустимый срок хранения в месяцах. Определить, сколько стоят самый дорогой и самый дешевый препараты; сколько наименований препаратов хранится на складе, какой препарат имеет срок хранения более трех месяцев и сколько стоят все препараты, хранящиеся на складе.

**Теоретические сведения**

Структура в языке C++ представляет собой производный тип данных, который представляет какую-то определенную сущность, также как и класс. Для определения структуры применяется ключевое слово struct, а сам формат определения выглядит следующим образом:

|  |  |
| --- | --- |
|  | struct имя\_структуры  {      компоненты\_структуры  }; |

После имени структуры в фигурных скобках помещаются *компоненты\_структуры*, которые представляют набор описаний объектов и функций, которые составляют структуру.

После определения структуры мы можем ее использовать. Мы можем определить объект структуры - по сути обычную переменную, которая будет представлять выше созданный тип. Также после создания переменной структуры можно обращаться к ее элементам - получать их значения или, наоборот, присваивать им новые значения. Для обращения к элементам структуры используется операция "точка":

|  |  |
| --- | --- |
|  | имя\_переменной\_структуры.имя\_элемента |

Также мы можем инициализировать структуру, присвоив ее переменным значения с помощью синтаксиса инициализации. Инициализация структур аналогична инициализации массивов: в фигурных скобках передаются значения для элементов структуры по порядку.

Лямбда-выражение в программировании — специальный синтаксис для определения функциональных объектов, заимствованный из λ-исчисления. Применяется как правило для объявления анонимных функций по месту их использования, и обычно допускает замыкание на лексический контекст, в котором это выражение использовано.

Один из вариантов объявления лямбда-выражения в C++:

[](параметры){тело\_функции}

Указатель на функцию – одно из средств ЯП C++. Указатель на функцию хранит адрес функции. По сути указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции. Самым распространенным указателем на функцию является её имя. С помощью имени функции можно вызвать её и получить результат её работы.

Но также указатель на функцию мы можем определить в виде отдельной переменной с помощью следующего синтаксиса:

тип\_возвращаемого\_значения(\*имя\_указателя)(параметры)

Указатель на функцию может быть параметром другой функции. Параметр функции типа указатель на функцию может принимать в качестве значения лямбда-выражение.

# Ход работы

Создаём проект в среде программирования Visual Studio 2022 «ОАиП ЛР - 11».

Пишем код первой программы (см. листинг А.1)

Результат работы программы:

|  |
| --- |
| Medicine storage:  Perflufradil:  Price 149.99  Count 23  Expiration time (months) 3  Gadonabacasan:  Price 389.99  Count 1  Expiration time (months) 1  Arilotilate:  Price 99.98  Count 3.255  Expiration time (months) 1  Guanikefagatran:  Price 999.99  Count 20  Expiration time (months) 4  Calciavinspirone:  Price 490  Count 38  Expiration time (months) 18  Activated carbon:  Price 39.99  Count 160  Expiration time (months) 80  Somidralazine:  Price 249  Count 16  Expiration time (months) 2  Estramesine:  Price 379  Count 18  Expiration time (months) 2  Vingrelithiazide:  Price 599  Count 20  Expiration time (months) 3  Pegprosticaine:  Price 599  Count 30  Expiration time (months) 1  -------------------------------  The most expensive medicine: Guanikefagatran Price 999.99  The cheapest medicine: Activated carbon with price 39.99  Total medicines: 10  Medicine with more than 3 months expiration time: Guanikefagatran  Total price of all medicines: 89939.4 |

# Результаты

Мы научились писать программы с использованием структур на языке C++ в среде разработки Visual Studio 2022.

# Приложение А. Исходный текст

Листинг А.1

|  |
| --- |
| main.cpp:  #include "Medicine.h"  #include "QueryingGeneric.h"  #include <iostream>  void PrintMedicine(Medicine& medicine)  {  std::cout << medicine.GetTitle() << ":\nPrice "  << medicine.GetPrice() << "\nCount "  << medicine.GetCount() << "\nExpiration time (months) "  << medicine.GetExpirationTime() << '\n' << std::endl;  }  int main()  {  size\_t count = 10;  Medicine\* storage = new Medicine[count]  {  Medicine("Perflufradil", 149.99f, 23.f, 3),  Medicine("Gadonabacasan", 389.99f, 1.f, 1),  Medicine("Arilotilate", 99.98f, 3.255f, 1),  Medicine("Guanikefagatran", 999.99f, 20.f, 4),  Medicine("Calciavinspirone", 490.f, 38.f, 18),  Medicine("Activated carbon", 39.99f, 160.f, 80),  Medicine("Somidralazine", 249.f, 16.f, 2),  Medicine("Estramesine", 379.f, 18.f, 2),  Medicine("Vingrelithiazide", 599.f, 20.f, 3),  Medicine("Pegprosticaine", 599.f, 30.f, 1)  };  std::cout << "Medicine storage:\n\n";  for (size\_t i = 0; i < count; ++i)  {  PrintMedicine(storage[i]);  }  float (\*priceSelector) (Medicine&) = [](Medicine& m) { return m.GetPrice(); };  Medicine& mostExpensive = qry::GetMax<Medicine, float>(  storage,  count,  priceSelector);  std::cout << "-------------------------------\n\nThe most expensive medicine: "  << mostExpensive.GetTitle() << " Price "  << mostExpensive.GetPrice() << std::endl;  Medicine& cheapest = qry::GetMin<Medicine, float>(  storage,  count,  priceSelector);  std::cout << "\nThe cheapest medicine: " << cheapest.GetTitle()  << " with price " << cheapest.GetPrice() << std::endl;  std::cout << "\nTotal medicines: " << count << std::endl;  Medicine more3MonthsExpiration;  if (qry::TryFind<Medicine>(  storage,  count,  [](Medicine& m) { return m.GetExpirationTime() > 3;},  more3MonthsExpiration))  {  std::cout << "\nMedicine with more than 3 months expiration time: "  << more3MonthsExpiration.GetTitle();  }  else  {  std::cout << "\nCould not find medicine with more than 3 months expiration time";  }  std::cout << std::endl;  float (\*totalPriceSelector) (Medicine&)  = [](Medicine& m) { return m.GetCount() \* m.GetPrice(); };  std::cout << "\nTotal price of all medicines: "  << qry::Sum<Medicine, float>(storage, count, totalPriceSelector)  << std::endl;  delete[] storage;  return 0;  }  Medicine.h:  #pragma once  #include <string>  struct Medicine  {  private:  std::string m\_title;  float m\_price;  float m\_count;  int m\_expirationTime;  public:  static const std::string NAME\_PLACEHOLDER;  static const Medicine& NULL\_OBJECT;  Medicine(  const std::string& title,  float price,  float count,  int expirationTime);  Medicine();  std::string GetTitle();  float GetPrice();  float GetCount();  int GetExpirationTime();  };  Medicine.cpp:  #include "Medicine.h"  const std::string Medicine::NAME\_PLACEHOLDER = "strange pills";  const Medicine& Medicine::NULL\_OBJECT = Medicine();  Medicine::Medicine(  const std::string& title,  float price,  float count,  int expirationTime)  {  m\_title = title;  m\_price = price;  m\_count = count;  m\_expirationTime = expirationTime;  }  Medicine::Medicine() : Medicine(NAME\_PLACEHOLDER, 0.f, 0.f, 0) {}  std::string Medicine::GetTitle()  {  return m\_title;  }  float Medicine::GetPrice()  {  return m\_price;  }  float Medicine::GetCount()  {  return m\_count;  }  int Medicine::GetExpirationTime()  {  return m\_expirationTime;  }  QueryingGeneric.h:  #pragma once  #include <type\_traits>  namespace qry  {  template<typename TSource, typename TSelector>  TSource& GetLastSatisfyingPredicate(  TSource\* source,  size\_t count,  bool(\*predicate) (TSelector, TSelector),  TSelector(\*selector) (TSource&))  {  size\_t index = 0;  for (size\_t i = 1; i < count; ++i)  {  if (predicate(selector(source[i]), selector(source[index])))  {  index = i;  }  }  return source[index];  }  template<typename TSource, typename TSelector>  TSource& GetMax(  TSource\* source,  size\_t count,  TSelector(\*selector) (TSource&))  {  return GetLastSatisfyingPredicate<TSource, TSelector>(  source,  count,  [](TSelector a, TSelector b) { return a > b; },  selector);  }  template<typename TSource, typename TSelector>  TSource& GetMin(  TSource\* source,  size\_t count,  TSelector(\*selector) (TSource&))  {  return GetLastSatisfyingPredicate<TSource, TSelector>(  source,  count,  [](TSelector a, TSelector b) { return a < b; },  selector);  }  template<typename TSource>  bool TryFind(  TSource\* source,  size\_t count,  bool (\*match) (TSource&),  TSource& result)  {  for (size\_t i = 0; i < count; ++i)  {  if (match(source[i]))  {  result = source[i];  return true;  }  }  return false;  }  template<typename TSource, typename TResult, typename TSelector>  TResult Aggregate(  TSource\* source,  size\_t count,  void(\*operation) (TResult&, TSelector),  TSelector(\*selector) (TSource&))  {  TResult result = TResult();  for (size\_t i = 0; i < count; ++i)  {  operation(result, selector(source[i]));  }  return result;  }  template<typename TSource, typename TResult = int>  TResult Sum(  TSource\* source,  size\_t count,  TResult(\*selector) (TSource&))  {  static\_assert(  std::is\_arithmetic\_v<TResult>,  "Result type must be arithmetic");  return Aggregate<TSource, TResult, TResult>  (source,  count,  [](TResult& sum, TResult term) { sum += term; },  selector);  }  } // namespace qry |
|  |

# Приложение Б. Ответы на вопросы

1. *Вопрос?*

Ответ.

1. *Вопрос?*

Ответ.

1. *Вопрос?*

Ответ.