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 №8 по дисциплине  
«Основы алгоритмизации и программирования»

Разработка программ с использованием структур

Студенты

«число» год                            Степанов М. Т.

\_\_\_\_\_\_\_\_\_\_\_\_\_ 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 программы для их решения.
3. Выполнить программы.

Задание №1

Создать записи, определяющие положение точки в декартовой и полярной системах координат. По заданному массиву координат точек в декартовой системе получить соответствующий массив координат заданных точек в полярной системе.

Задание №2

Дан массив записей, содержащих дату (число, месяц, год) и время (час, минута, секунда). Упорядочить этот массив данных в порядке возрастания, т.е. от более ранних значений к более поздним.

**Теоретические сведения**

# Структура — это , некое объединение различных переменных (даже с разными типами данных), которому можно присвоить имя. Например, можно объединить данные об объекте Дом: город (в котором дом находится), улица, количество квартир, интернет(проведен или нет) и т.д. в одной структуре. В общем, можно собрать в одну совокупность данные обо всем, что угодно, точнее обо всем, что необходимо конкретному программисту.

Что ещё важно знать:

Объект структуры можно объявить до функции main().

Структуру можно вкладывать в другие структуры.

Функции могут возвращать структуры в результате своей работы.

# Ход работы

Создали проект в среде программирования Visual Studio 2022 «ОАиП ЛР - 8.1».

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

Результат работы программы:

|  |
| --- |
| Enter the number of points: 2  Enter point's name: first  Enter point's dimension (whole number): 2  Enter point's coordinates:  x0 = 4  x1 = 6  Enter point's name: second  Enter point's dimension (whole number): 3  Enter point's coordinates:  x0 = 3  x1 = 5  x2 = 7  Polar points:  first (2-dim):  r = 7.211  phi0 = 0.9828  second (3-dim):  r = 9.11  phi0 = 1.03  phi1 = 0.9505 |

Создали проект в среде программирования Visual Studio 2019 «ОАиП ЛР - 8.2».

Пишем код второй программы (см. листинг А.2)

Результат работы программы:

|  |
| --- |
| Enter the number of dates: 5  Enter the date#1:  Day = 31  Month = 12  Year = 2012  Hour = 23  Minute = 59  Second = 59  Result: 23:59:59 31.12.2012  Enter the date#2:  Day = 11  Month = 9  Year = 2011  Hour = 10  Minute = 1  Second = 23  Result: 10:01:23 11.09.2011  Enter the date#3:  Day = 6  Month = 7  Year = 2013  Hour = 0  Minute = 0  Second = 1  Result: 00:00:01 06.07.2013  Enter the date#4:  Day = 25  Month = 4  Year = 2010  Hour = 18  Minute = 0  Second = 9  Result: 18:00:09 25.04.2010  Enter the date#5:  Day = 11  Month = 11  Year = 2012  Hour = 14  Minute = 38  Second = 50  Result: 14:38:50 11.11.2012  Ordered by increasing:  18:00:09 25.04.2010  10:01:23 11.09.2011  14:38:50 11.11.2012  23:59:59 31.12.2012  00:00:01 06.07.2013 |

# Результаты

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

# Приложение А. Исходный текст

Листинг А.1

|  |
| --- |
| main.cpp:  #include "Points.h"  #include <iostream>  #include <stdexcept>  int main()  {  std::cout << "Enter the number of points: ";  size\_t numberOfPoints = 0;  std::cin >> numberOfPoints;  CartesianPoint\* points = new (std::nothrow) CartesianPoint[numberOfPoints];  if (points == nullptr)  {  std::cout << std::endl << "Failed memory allocation" << std::endl;  return -1;  }  try  {  for (size\_t i = 0; i < numberOfPoints; ++i)  {  points[i] = CartesianPoint::Scan();  }  std::cout << std::endl << "Polar points:" << std::endl;  for (size\_t i = 0; i < numberOfPoints; ++i)  {  points[i].ToPolarPoint().Print();  }  }  catch(std::exception& e)  {  std::cout << std::endl << e.what() << std::endl;  return -1;  }  delete[] points;  return 0;  }  Points.cpp:  #include "Points.h"  #include "MathUtil.h"  #include <iostream>  #include <iomanip>  #include <stdexcept>  const size\_t CartesianPoint::MIN\_DIMENSION = 1;  const size\_t CartesianPoint::MAX\_DIMENSION = 1024;  const std::string CartesianPoint::NAME\_PLACEHOLDER = "untitled";  const std::string CartesianPoint::DIMENSION\_OUT\_OF\_RANGE\_MESSAGE = "dimension was out of range";  const std::string CartesianPoint::DIMENSION\_MISMATCH\_MESSAGE = "dimension and number of components must be the same";  const std::string CartesianPoint::INDEX\_OUT\_OF\_RANGE\_MESSAGE = "the index was out of range";  CartesianPoint::CartesianPoint(size\_t dimension, const std::vector<float>& x, const std::string& name)  {  if (dimension < MIN\_DIMENSION || dimension > MAX\_DIMENSION)  {  throw std::out\_of\_range(DIMENSION\_OUT\_OF\_RANGE\_MESSAGE);  }  if (dimension != x.size())  {  throw std::logic\_error(DIMENSION\_MISMATCH\_MESSAGE);  }  m\_dimension = dimension;  m\_x = x;  m\_name = name;  }  CartesianPoint::CartesianPoint(const std::vector<float>& x, const std::string& name) : CartesianPoint(x.size(), x, name) {}  CartesianPoint::CartesianPoint(const std::vector<float>& x) : CartesianPoint(x.size(), x, NAME\_PLACEHOLDER) {}  CartesianPoint::CartesianPoint() : CartesianPoint(1, { 0.f }, NAME\_PLACEHOLDER) {}  const std::string& CartesianPoint::GetNamePlaceholder()  {  return NAME\_PLACEHOLDER;  }  const std::string& CartesianPoint::GetName()  {  return m\_name;  }  size\_t CartesianPoint::GetDimension()  {  return m\_dimension;  }  float CartesianPoint::GetXAt(size\_t index)  {  if (index >= m\_x.size() || index < 0)  {  throw std::out\_of\_range(INDEX\_OUT\_OF\_RANGE\_MESSAGE);  }  return m\_x[index];  }  void CartesianPoint::SetXAt(size\_t index, float value)  {  if (index >= m\_x.size() || index < 0)  {  throw std::out\_of\_range(INDEX\_OUT\_OF\_RANGE\_MESSAGE);  }  m\_x[index] = value;  }  void CartesianPoint::Print()  {  const int FLOAT\_PRECISION = 4;  std::cout << std::endl << m\_name << " (" << m\_dimension << "-dim):" << std::endl;  for (size\_t i = 0; i < m\_x.size(); ++i)  {  std::cout << " x" << i << " = "  << std::setprecision(FLOAT\_PRECISION)  << m\_x[i] << std::endl;  }  }  CartesianPoint CartesianPoint::Scan()  {  std::cout << std::endl << "Enter point's name: ";  std::string name;  std::cin >> name;  std::cout << "Enter point's dimension (whole number): ";  size\_t n = 0;  std::cin >> n;  if (n < MIN\_DIMENSION || n > MAX\_DIMENSION)  {  throw std::out\_of\_range(DIMENSION\_OUT\_OF\_RANGE\_MESSAGE);  }  std::cout << "Enter point's coordinates:" << std::endl;  std::vector<float> x(n);  for (size\_t i = 0; i < n; ++i)  {  std::cout << " x" << i << " = ";  std::cin >> x[i];  }  CartesianPoint point(n, x, name);  return point;  }  PolarPoint CartesianPoint::ToPolarPoint()  {  float r = 0.f;  auto n = m\_dimension - 1;  std::vector<float> phi(n);  for (int i = 0; i <= n; ++i)  {  r += static\_cast<float>(pow(m\_x[i], 2));  }  r = sqrt(r);  if (n > 0)  {  for (int i = 0; i < n - 1; ++i)  {  float sumOfSquares = 0.f;  for (int j = i; j < n; ++j)  {  sumOfSquares += static\_cast<float>(pow(m\_x[j], 2));  }  phi[i] = acos(m\_x[i] / sqrt(sumOfSquares));  }  float cos = static\_cast<float>(m\_x[n - 1] / sqrt(pow(m\_x[n], 2) + pow(m\_x[n - 1], 2)));  phi[n - 1] = GetAngle(Sign(m\_x[n]), cos);  }  PolarPoint result(r, phi, m\_name);  return result;  }  const size\_t PolarPoint::MIN\_DIMENSION = 1;  const size\_t PolarPoint::MAX\_DIMENSION = 1024;  const std::string PolarPoint::NAME\_PLACEHOLDER = "untitled";  const std::string PolarPoint::DIMENSION\_OUT\_OF\_RANGE\_MESSAGE = "dimension was out of range";  const std::string PolarPoint::DIMENSION\_MISMATCH\_MESSAGE = "number of angles must be one less then dimension";  const std::string PolarPoint::INDEX\_OUT\_OF\_RANGE\_MESSAGE = "the index was out of range";  PolarPoint::PolarPoint(size\_t dimension, float r, const std::vector<float>& phi, const std::string& name)  {  if (dimension < MIN\_DIMENSION || dimension > MAX\_DIMENSION)  {  throw std::out\_of\_range(DIMENSION\_OUT\_OF\_RANGE\_MESSAGE);  }  if (dimension - 1 != phi.size())  {  throw std::logic\_error(DIMENSION\_MISMATCH\_MESSAGE);  }  m\_dimension = dimension;  m\_r = r;  m\_phi = phi;  m\_name = name;  }  PolarPoint::PolarPoint(float r, const std::vector<float>& phi, const std::string& name) : PolarPoint(phi.size() + 1, r, phi, name) {}  PolarPoint::PolarPoint(float r, const std::vector<float>& phi) : PolarPoint(phi.size() + 1, r, phi, NAME\_PLACEHOLDER) {}  void PolarPoint::Print()  {  const int FLOAT\_PRECISION = 4;  std::cout << std::endl << m\_name << " (" << m\_dimension << "-dim):" << std::endl;  std::cout << " r = " << std::setprecision(FLOAT\_PRECISION) << m\_r << std::endl;  for (size\_t i = 0; i < m\_phi.size(); ++i)  {  std::cout << " phi" << i << " = "  << std::setprecision(FLOAT\_PRECISION)  << m\_phi[i] << std::endl;  }  }  Points.h:  #pragma once  #include <string>  #include <vector>  class PolarPoint  {  private:  static const size\_t MIN\_DIMENSION;  static const size\_t MAX\_DIMENSION;  static const std::string NAME\_PLACEHOLDER;  static const std::string DIMENSION\_OUT\_OF\_RANGE\_MESSAGE;  static const std::string DIMENSION\_MISMATCH\_MESSAGE;  static const std::string INDEX\_OUT\_OF\_RANGE\_MESSAGE;  size\_t m\_dimension;  float m\_r;  std::vector<float> m\_phi;  std::string m\_name;  public:  PolarPoint(size\_t dimension, float r, const std::vector<float>& phi, const std::string& name);  PolarPoint(float r, const std::vector<float>& phi, const std::string& name);  PolarPoint(float r, const std::vector<float>& phi);  void Print();  };  class CartesianPoint  {  private:  static const size\_t MIN\_DIMENSION;  static const size\_t MAX\_DIMENSION;  static const std::string NAME\_PLACEHOLDER;  static const std::string DIMENSION\_OUT\_OF\_RANGE\_MESSAGE;  static const std::string DIMENSION\_MISMATCH\_MESSAGE;  static const std::string INDEX\_OUT\_OF\_RANGE\_MESSAGE;  size\_t m\_dimension;  std::vector<float> m\_x;  std::string m\_name;  public:  CartesianPoint(size\_t dimension, const std::vector<float>& x, const std::string& name);  CartesianPoint(const std::vector<float>& x, const std::string& name);  CartesianPoint(const std::vector<float>& x);  CartesianPoint();  const std::string& GetNamePlaceholder();  const std::string& GetName();  size\_t GetDimension();  float GetXAt(size\_t index);  void SetXAt(size\_t index, float value);  void Print();  static CartesianPoint Scan();  PolarPoint ToPolarPoint();  };  MathUtil.cpp:  #define \_USE\_MATH\_DEFINES  #include "MathUtil.h"  #include <cmath>  #include <stdexcept>  int Sign(float number)  {  if (std::isnan(number))  {  throw std::invalid\_argument("recieved NaN value");  }  const float eps = 0.00000001f;  float diff = fabs(number);  if (diff <= eps)  {  return 0;  }  else if (number > 0)  {  return 1;  }  else  {  return -1;  }  }  float GetAngle(int sign, float cos)  {  float arccos = acos(cos);  return (sign >= 0 ? arccos : static\_cast<float>(M\_PI \* 2 - arccos));  }  MathUtil.h:  #pragma once  int Sign(float number);  float GetAngle(int sign, float cos); |
|  |

Листинг А.2

|  |
| --- |
| main.cpp:  #include <iostream>  #include <string>  #include <stdexcept>  #include "DateTime.h"  template<typename T>  void Swap(T& a, T& b)  {  T c = a;  a = b;  b = c;  }  void SortDates(DateTime\* dates, size\_t numberOfDates)  {  for (size\_t i = 0; i < numberOfDates - 1; ++i)  {  bool isSorted = true;  for (size\_t j = 0; j < numberOfDates - i - 1; ++j)  {  if (dates[j].IsGreaterThan(dates[j + 1]))  {  Swap<DateTime>(dates[j], dates[j + 1]);  isSorted = false;  }  }  if (isSorted)  {  break;  }  }  }  int main()  {  try  {  std::cout << "Enter the number of dates: ";  size\_t numberOfDates = 0;  std::cin >> numberOfDates;  DateTime\* dates = new DateTime[numberOfDates];  for (size\_t i = 0; i < numberOfDates; ++i)  {  dates[i] = DateTime::Scan('#' + std::to\_string(i + 1));  std::cout << "Result: " << dates[i].ToString() << std::endl;  }  SortDates(dates, numberOfDates);  std::cout << std::endl << "Ordered by increasing:" << std::endl;  for (size\_t i = 0; i < numberOfDates; ++i)  {  std::cout << std::endl << dates[i].ToString() << std::endl;  }  }  catch(std::exception& e)  {  std::cout << "Error: " << e.what() << std::endl;  return -1;  }  return 0;  }  DateTime.cpp:  #include "DateTime.h"  #include "StringUtil.h"  #include <iostream>  #include <iomanip>  #include <cmath>  #include <vector>  #include <stdexcept>  const int DateTime::MIN\_DAY = 1;  const int DateTime::MAX\_DAYS[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  const int DateTime::MIN\_MONTH = 1;  const int DateTime::MAX\_MONTH = 12;  const int DateTime::MIN\_YEAR = 1;  const int DateTime::MAX\_YEAR = 9999;  const int DateTime::MIN\_HOUR = 0;  const int DateTime::MAX\_HOUR = 23;  const int DateTime::MIN\_MINUTE = 0;  const int DateTime::MAX\_MINUTE = 59;  const int DateTime::MIN\_SECOND = 0;  const int DateTime::MAX\_SECOND = 59;  const std::string DateTime::INPUT\_OUT\_OF\_RANGE\_MEASSGE = "The input was out of range";  DateTime::DateTime(int day, int month, int year, int hour, int minute, int second)  {  if (month < MIN\_MONTH || month > MAX\_MONTH  || day < MIN\_DAY || day > MAX\_DAYS[month - 1]  || year < MIN\_YEAR || year > MAX\_YEAR  || hour < MIN\_HOUR || hour > MAX\_HOUR  || minute < MIN\_MINUTE || minute > MAX\_MINUTE  || second < MIN\_SECOND || second > MAX\_SECOND)  {  throw std::out\_of\_range(INPUT\_OUT\_OF\_RANGE\_MEASSGE);  }  m\_day = day;  m\_month = month;  m\_year = year;  m\_hour = hour;  m\_minute = minute;  m\_second = second;  }  DateTime::DateTime() : DateTime(1, 1, 1, 0, 0, 0) {}  int DateTime::GetDay()  {  return m\_day;  }  int DateTime::GetMonth()  {  return m\_month;  }  int DateTime::GetYear()  {  return m\_year;  }  int DateTime::GetHour()  {  return m\_hour;  }  int DateTime::GetMinute()  {  return m\_minute;  }  int DateTime::GetSecond()  {  return m\_second;  }  bool DateTime::IsGreaterThan(DateTime& other)  {  if (m\_year != other.GetYear())  {  return m\_year > other.GetYear();  }  else if (m\_month != other.GetMonth())  {  return m\_month > other.GetMonth();  }  else if (m\_day != other.GetDay())  {  return m\_day > other.GetDay();  }  else if (m\_hour != other.GetHour())  {  return m\_hour > other.GetHour();  }  else if (m\_minute != other.GetMinute())  {  return m\_minute > other.GetMinute();  }  else if(m\_second != other.GetSecond())  {  return m\_second > other.GetSecond();  }  else  {  return false;  }  }  std::string DateTime::ToString()  {  const std::string YEAR\_TEMPLATE = "0000";  const std::string MONTH\_TEMPLATE = "00";  const std::string DAY\_TEMPLATE = "00";  const std::string HOUR\_TEMPLATE = "00";  const std::string MINUTE\_TEMPLATE = "00";  const std::string SECOND\_TEMPLATE = "00";  return RightInsert(HOUR\_TEMPLATE, m\_hour)  + ':' + RightInsert(MINUTE\_TEMPLATE, m\_minute)  + ':' + RightInsert(SECOND\_TEMPLATE, m\_second)  + ' ' + RightInsert(DAY\_TEMPLATE, m\_day)  + '.' + RightInsert(MONTH\_TEMPLATE, m\_month)  + '.' + RightInsert(YEAR\_TEMPLATE, m\_year);  }  void DateTime::Print()  {  std::cout << this->ToString();  }  int DateTime::ScanComponent(const std::string& message)  {  std::cout << message;  int value = 0;  std::cin >> value;  return value;  }  DateTime DateTime::Scan(const std::string& header)  {  std::cout << std::endl << "Enter the date" + header + ':' << std::endl;  int day = ScanComponent("Day = ");  int month = ScanComponent("Month = ");  int year = ScanComponent("Year = ");  int hour = ScanComponent("Hour = ");  int minute = ScanComponent("Minute = ");  int second = ScanComponent("Second = ");  DateTime result(day, month, year, hour, minute, second);  return result;  }  DateTime DateTime::Scan()  {  return DateTime::Scan("");  }  DateTime.h:  #pragma once  #include <string>  class DateTime  {  private:  static const int MIN\_DAY;  static const int MAX\_DAYS[];  static const int MIN\_MONTH;  static const int MAX\_MONTH;  static const int MIN\_YEAR;  static const int MAX\_YEAR;  static const int MIN\_HOUR;  static const int MAX\_HOUR;  static const int MIN\_MINUTE;  static const int MAX\_MINUTE;  static const int MIN\_SECOND;  static const int MAX\_SECOND;  static const std::string INPUT\_OUT\_OF\_RANGE\_MEASSGE;  int m\_day;  int m\_month;  int m\_year;  int m\_hour;  int m\_minute;  int m\_second;  static int ScanComponent(const std::string& message);  public:  DateTime(int day, int month, int year, int hour, int minute, int second);  DateTime();  int GetDay();  int GetMonth();  int GetYear();  int GetHour();  int GetMinute();  int GetSecond();  bool IsGreaterThan(DateTime& other);  std::string ToString();  void Print();  static DateTime Scan();  static DateTime Scan(const std::string& header);  };  StringUtil.cpp:  #include "StringUtil.h"  std::string RightInsert(const std::string& base, const std::string& insertable)  {  std::string result = base;  int j = insertable.length() - 1;  for (int i = base.length() - 1; i >= 0 && j >= 0; --i)  {  result[i] = insertable[j];  --j;  }  return result;  }  std::string RightInsert(const std::string& base, int insertable)  {  return RightInsert(base, std::to\_string(insertable));  }  StringUtil.h:  #pragma once  #include <string>  std::string RightInsert(const std::string& base, const std::string& insertable);  std::string RightInsert(const std::string& base, int insertable); |
|  |

# Приложение Б. Ответы на вопросы

1. *Вопрос?*

Ответ.

1. *Вопрос?*

Ответ.

1. *Вопрос?*

Ответ.