МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**ИНТЕЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ**

ОТЧЁТ

По лабораторной работе № \_\_

Выполнил:

Студент группы ИИ-22

Копанчук Евгений Романович

Проверил\_:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Брест – 2021

ЦЕЛЬ РАБОТЫ

1. Знакомство с интегрированной средой (на примере Microsoft Visual Studio). Изучение ГОСТ 19.701 и правил описания графических схем алгоритмов.

2. Знакомство с разработкой консольных приложений в технологии структурной разработки ПС и перенос задач в среду программирования С++ (например, варианты задач 1 и 2) – тестирование навыков по использованию базовых средств языка С.

3. Изучение индивидуальных заданий по разработке классов, изучение правил описания классов в терминах языка UML, описание классов

**Ход работы**

ЗАДАЧА 1 (индивидуальное задание).

Объект автоматизации – предметная область типа “Неарифметический класс”, например, СТУДЕНТ, ГРУППА, ТОВАР, ПРАЙС, КНИГА, РЕЙС, МАРШРУТ и т.п.

Для заданной предметной области следует разработать класс. Он должен в итоговом варианте содержать следующие члены.

1. 2-3 свойства-атрибута (один массивного типа).

2. Три конструктора (по одному каждого типа).

3. Методы, достаточные для установки и получения свойств объекта класса (например, TO\_SET(), TO\_SHOW() и т.п.).

4. Три перегруженных оператора (операторы типа “=”, а также двуместный и одноместный операторы).

ЗАДАЧА 2 (индивидуальное задание).

Объект автоматизации – предметная область типа “Арифметический класс”, например, класс для обработки и хранения трех значений целого типа; класс для обработки и хранения одного значения вещественного и двух значений целого типа; класс для обработки и хранения значения комплексного типа; класс для обработки и хранения значений вектора и т.п.

Для заданной предметной области следует разработать класс аналогично ЗАДАЧЕ 2

**Программная реализация:**

|  |  |
| --- | --- |
| #pragma once  #include <iostream>  using std::cout;  using std::endl;  using std::string;  using std::ostream;  class User  {  private:  string name;  double balance;  int options[3];  public:  User();  User(string name);  User(string name, double balance, int\* options);  User(const User& user);  void TO\_SET\_NAME(string name);  void TO\_SET\_BALANCE(double balance);  void TO\_SET\_OPTIONS(int\* options);  string TO\_GET\_NAME();  double TO\_GET\_BALANCE();  int\* TO\_GET\_OPTIONS();  void TO\_SHOW\_NAME();  void TO\_SHOW\_BALANCE();  void TO\_SHOW\_OPTIONS();  User operator = (const User& other);  bool operator == (const User& other);  User operator+ (const User& other);  }; | User.h |
| #include "User.h"  User::User() {  name = "guest";  balance = 0;  options[0] = 90;  options[1] = 60;  options[2] = 90;  }  User::User(string name) {  this->name = name;  balance = 0;  options[0] = 90;  options[1] = 60;  options[2] = 90;  }  User::User(string name, double balance, int\* options) {  this->name = name;  this->balance = balance;  this->options[0] = options[0];  this->options[1] = options[1];  this->options[2] = options[2];  }  User::User(const User& user) {  this->name = user.name;  this->balance = user.balance;  this->options[0] = user.options[0];  this->options[1] = user.options[1];  this->options[2] = user.options[2];  }  void User::TO\_SET\_NAME(string name) {  this->name = name;  }  void User::TO\_SET\_BALANCE(double balance) {  this->balance = balance;  }  void User::TO\_SET\_OPTIONS(int\* options) {  this->options[0] = options[0];  this->options[1] = options[1];  this->options[2] = options[2];  }  string User::TO\_GET\_NAME() { return name; }  double User::TO\_GET\_BALANCE() { return balance; }  int\* User::TO\_GET\_OPTIONS() { return options; }  void User::TO\_SHOW\_NAME() { cout << name << endl; }  void User::TO\_SHOW\_BALANCE() { cout << balance << endl; }  void User::TO\_SHOW\_OPTIONS() { cout << options[0] << ' ' << options[1] << ' ' << options[2] << endl; }  User User::operator = (const User& other) {  this->name = other.name;  this->balance = other.balance;  this->options[0] = other.options[0];  this->options[1] = other.options[1];  this->options[2] = other.options[2];  return \*this;  }  bool User::operator == (const User& other) {  if (this->name != other.name ||  this->balance != other.balance ||  this->options[0] != other.options[0] ||  this->options[1] != other.options[1] ||  this->options[2] != other.options[2])  return false;  return true;  }  User User::operator + (const User& other) {  this->options[0] = (this->options[0] + other.options[0]) / 2;  this->options[1] = (this->options[1] + other.options[1]) / 2;  this->options[2] = (this->options[2] + other.options[2]) / 2;  return \*this;  } | **User.cpp** |
| #pragma once  #include <iostream>  #include <ctime>  using std::cout;  using std::endl;  using std::string;  using std::ostream;  class Sum  {  private:  int a;  int b;  int c;  public:  Sum();  Sum(int a, int b, int c);  Sum(const Sum& sum);  void TO\_SET\_A(int a);  void TO\_SET\_B(int b);  void TO\_SET\_C(int c);  int TO\_GET\_A();  int TO\_GET\_B();  int TO\_GET\_C();  void TO\_SHOW\_A();  void TO\_SHOW\_B();  void TO\_SHOW\_C();  void TO\_SHOW\_SUM();  bool TO\_COMPARE();  bool TO\_COMPARE(char fst, char snd);  bool TO\_COMPARE(int fst, int snd);  Sum operator = (const Sum& other);  bool operator == (const Sum& other);  Sum operator+ (const Sum& other);  }; | **Sum.h** |
| #include "Sum.h"  Sum::Sum() : a(0), b(0) {}  Sum::Sum(int a, int b, int c) : a(a), b(b), c(c) {}  Sum::Sum(const Sum& sum) : a(sum.a), b(sum.b) {}  void Sum::TO\_SET\_A(int a) { this->a = a; }  void Sum::TO\_SET\_B(int b) { this->b = b; }  void Sum::TO\_SET\_C(int c) { this->c = c; }  int Sum::TO\_GET\_A() { return a; }  int Sum::TO\_GET\_B() { return b; }  int Sum::TO\_GET\_C() { return c; }  void Sum::TO\_SHOW\_A() { cout << a << endl; }  void Sum::TO\_SHOW\_B() { cout << b << endl; }  void Sum::TO\_SHOW\_C() { cout << c << endl; }  void Sum::TO\_SHOW\_SUM() { cout << a + b << endl; }  bool Sum::TO\_COMPARE() { return a == b && b == c; }  bool Sum::TO\_COMPARE(char fst, char snd) {  if (fst == 'a' && snd == 'b') return a == b;  if (fst == 'a' && snd == 'c') return a == c;  if (fst == 'b' && snd == 'c') return b == c;  }  bool Sum::TO\_COMPARE(int fst, int snd) {  if (fst == 1 && snd == 2) return a == b;  if (fst == 1 && snd == 3) return a == c;  if (fst == 2 && snd == 3) return b == c;  }  Sum Sum::operator = (const Sum& other) { this->a = other.a; this->b = other.b; return \*this; }  bool Sum::operator == (const Sum& other) { if (this->a + this->b == other.a + other.b) return 1; return 0; }  Sum Sum::operator + (const | **Sum.cpp** |

**Пример работы некоторых функций:**

|  |  |
| --- | --- |
| #include <iostream>  #include "User.h"  #include "Sum.h"  int main() { // User  User guest;  guest.TO\_SHOW\_NAME();  guest.TO\_SHOW\_BALANCE();  guest.TO\_SHOW\_OPTIONS();  string name = guest.TO\_GET\_NAME();  double balance = guest.TO\_GET\_BALANCE();  int\* options = guest.TO\_GET\_OPTIONS();  cout << name << endl <<  balance << endl <<  options[0] << ' ' <<  options[1] << ' ' <<  options[2] << endl;  int\* admin\_opt = new int[3];  admin\_opt[0] = 100;  admin\_opt[1] = 70;  admin\_opt[2] = 75;  User admin("admin", 10.23, admin\_opt);  admin = admin + guest;  admin.TO\_SHOW\_NAME();  admin.TO\_SHOW\_BALANCE();  admin.TO\_SHOW\_OPTIONS();  cout << (admin == guest) << endl;  admin = guest;  cout << (admin == guest) << endl;  // Sum  cout << '\n';  Sum c;  c.TO\_SHOW\_A();  c.TO\_SHOW\_B();  int a = c.TO\_GET\_A();  int b = c.TO\_GET\_B();  cout << a << ' ' << b << endl;  a += 11; b += 21;  Sum d(a, b);  d = d + c;  d.TO\_SHOW\_A();  d.TO\_SHOW\_B();  cout << (c == d) << endl;  c = d;  cout << (c == d) << endl;  } |  |

**Вывод:** Сегодня я научился создавать простые классы с набором свойств и методов, разобрался с понятиями конструктор, конструктор копирования, научился работать с простейшей перегрузкой оператора. Попробовал разработать свой собственный арифметический (Sum) и неарифметический (User) классы.