Васильев Е.О.

***Лабораторная работа* №*8***

## ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОТНОШЕНИЯ

**множественного наследования**

ЦЕЛЬ РАБОТЫ:

Изучить принципы и механизмы множественного наследования, правила доступа к базовым классам; приобрести практические навыки работы с базовыми и производными классами при множественном наследовании.

**Задания:**

1. Используя предыдущую программу, создайте новый производный класс с применением множественного наследования.

2. Проверьте работоспособность АТД и производных классов на тестовом наборе данных.

**Код:**

|  |
| --- |
| //#include "pch.h"  #include <iostream>  #include <string>  using namespace std;  void PrintLine() {  cout << "-================================-\n";  cout << "\n";  }  template <class T>  class SoundRecord  {  private:  char\* name;  T kolvo;  T price;  char\* power;  public:  SoundRecord();  SoundRecord(char\* a, T b, T c, char\* d);  SoundRecord(const SoundRecord& a);  ~SoundRecord();  SoundRecord Sum(const SoundRecord& a);  /\*void copy(const SoundRecord &a);\*/  bool Compair(SoundRecord& a);  void destroy();  void Setname(char\* a);  void Getname();  void Setkolvo(T a);  int Getkolvo();  void SetPrice(T a);  void GetPrice();  void Setpr(char\* a);  void Getpr();  SoundRecord& operator+(const SoundRecord& a);  bool operator>(const SoundRecord& r);  SoundRecord& operator= (const SoundRecord& r);  bool operator==(const SoundRecord& a);  };  template <class T> SoundRecord<T>::SoundRecord()  {  name = new char[11];  power = new char[11];  strcpy(name, "none");  strcpy(power, "none");  price = 0;  kolvo = 0;  }  template <class T> SoundRecord<T>::SoundRecord(char\* a, T b, T c, char\* d)  {  name = a;  kolvo = b;  price = c;  power = d;  }  template <class T> SoundRecord<T>::SoundRecord(const SoundRecord<T>& a)  {// констр копирования  // тело конструктора  name = new char[strlen(a.name) + 1];  power = new char[strlen(a.power) + 1];  strcpy(name, a.name);  strcpy(power, a.power);  price = a.price;  kolvo = a.kolvo;  }  template <class T>SoundRecord<T>::~SoundRecord()  {  //delete[] name;  //delete[] pr;  cout << "Memory has been succesfully cleaned" << endl;  };  template <class T>void SoundRecord<T>::Getname() {  cout.width(5);  cout << "name: " << name << "\n";  }  template <class T>void SoundRecord<T>::Setkolvo(T a) {  kolvo = a;  }  template <class T>int SoundRecord<T>::Getkolvo() {  cout.width(5);  cout << "released: " << kolvo << "\n";  return kolvo;  }  template <class T>void SoundRecord<T>::Setname(char\* a) {  name = new char[strlen(a) + 1];  strcpy(name, a);  }  template <class T>void SoundRecord<T>::SetPrice(T a) {  price = a;  }  template <class T>void SoundRecord<T>::GetPrice() {  cout.width(5);  cout << "price: " << price << "\n";  }  template <class T>void SoundRecord<T>::Setpr(char\* a) {  power = new char[strlen(a) + 1];  strcpy(power, a);  }  template <class T>void SoundRecord<T>::Getpr() {  cout.width(5);  cout << "processor: " << power << "\n";  }  template <class T>SoundRecord<T>& SoundRecord<T>::operator+(const SoundRecord<T>& a)  {  SoundRecord v;  v.name = new char[strlen(a.name) + strlen(name) + 2];  v.power = new char[strlen(a.power) + strlen(power) + 2];  strcpy(v.name, name);  strcat(v.name, ",");  strcat(v.name, a.name);  strcpy(v.power, power);  strcat(v.power, ",");  strcat(v.power, a.power);  v.price = price + a.price;  v.kolvo = kolvo + a.kolvo;  v.Getname();  v.Getpr();  v.GetPrice();  v.Getkolvo();  return v;  }  template <class T>bool SoundRecord<T>::operator>(const SoundRecord<T>& a)  {  return ((!strcmp(name, a.name)) &&  price > a.price &&  kolvo > a.kolvo);  }  template <class T>SoundRecord<T>& SoundRecord<T>::operator=(const SoundRecord<T>& r)  {  price = r.price;  kolvo = r.kolvo;  power = r.power;  name = r.name;  return \*this;  }  template <class T>bool SoundRecord<T>::operator==(const SoundRecord<T>& a)  {  return ((!strcmp(name, a.name)) &&  price == a.price &&  kolvo == a.kolvo);  }  template <class T>void SoundRecord<T>::destroy() {  delete[] name;  delete[] power;  cout << "Memory has been succesfully cleaned" << endl;  };  template <class T>SoundRecord<T> SoundRecord<T>::Sum(const SoundRecord<T>& a)  {  SoundRecord v;  v.name = new char[strlen(a.name) + strlen(name) + 2];  v.power = new char[strlen(a.power) + strlen(power) + 2];  strcpy(v.name, name);  strcat(v.name, ",");  strcat(v.name, a.name);  strcpy(v.power, power);  strcat(v.power, ",");  strcat(v.power, a.power);  v.price = price + a.price;  v.kolvo = kolvo + a.kolvo;  v.Getname();  v.Getpr();  v.GetPrice();  v.Getkolvo();  return v;  }  template <class T>bool SoundRecord<T>::Compair(SoundRecord<T>& a)  {  return ((!strcmp(name, a.name)) &&  price == a.price &&  kolvo == a.kolvo);  }  template <class T, class T2>  class SoundRecord1  {  private:  T2\* name;  T kolvo;  T price;  T2\* power;  public:  SoundRecord1();  SoundRecord1(T2\* a, T b, T c, T2\* d);  SoundRecord1(const SoundRecord1& a);  ~SoundRecord1();  SoundRecord1 Sum(const SoundRecord1& a);  /\*void copy(const SoundRecord &a);\*/  bool Compair(SoundRecord1& a);  void destroy();  void Setname(T2\* a);  void Getname();  void Setkolvo(T a);  int Getkolvo();  void SetPrice(T a);  void GetPrice();  void Setpr(T2\* a);  void Getpr();  SoundRecord1& operator+(const SoundRecord1& a);  bool operator>(const SoundRecord1& r);  SoundRecord1& operator= (const SoundRecord1& r);  bool operator==(const SoundRecord1& a);  };  template <class T, class T2> SoundRecord1<T, T2>::SoundRecord1()  {  name = new char[11];  power = new char[11];  strcpy(name, "none");  strcpy(power, "none");  price = 0;  kolvo = 0;  }  template <class T, class T2> SoundRecord1<T, T2>::SoundRecord1<T, T2>(T2\* a, T b, T c, T2\* d)  {  name = a;  kolvo = b;  price = c;  power = d;  }  template <class T, class T2> SoundRecord1<T, T2>::SoundRecord1<T, T2>(const SoundRecord1<T, T2>& a)  {// констр копирования  // тело конструктора  name = new char[strlen(a.name) + 1];  power = new char[strlen(a.power) + 1];  strcpy(name, a.name);  strcpy(power, a.power);  price = a.price;  kolvo = a.kolvo;  }  template <class T, class T2> SoundRecord1<T, T2>::~SoundRecord1()  {  //delete[] name;  //delete[] pr;  cout << "Memory has been succesfully cleaned" << endl;  };  template <class T, class T2 >void SoundRecord1<T, T2>::Getname() {  cout.width(5);  cout << "name: " << name << "\n";  }  template <class T, class T2 >void SoundRecord1<T, T2>::Setkolvo(T a) {  kolvo = a;  }  template <class T, class T2 >int SoundRecord1<T, T2>::Getkolvo() {  cout.width(5);  cout << "released: " << kolvo << "\n";  return kolvo;  }  template <class T, class T2 >void SoundRecord1<T, T2>::Setname(T2\* a) {  name = new char[strlen(a) + 1];  strcpy(name, a);  }  template <class T, class T2 >void SoundRecord1<T, T2>::SetPrice(T a) {  price = a;  }  template <class T, class T2 >void SoundRecord1<T, T2>::GetPrice() {  cout.width(5);  cout << "price: " << price << "\n";  }  template <class T, class T2 >void SoundRecord1<T, T2>::Setpr(T2\* a) {  power = new char[strlen(a) + 1];  strcpy(power, a);  }  template <class T, class T2 >void SoundRecord1<T, T2>::Getpr() {  cout.width(5);  cout << "processor: " << power << "\n";  }  template <class T, class T2 >SoundRecord1<T, T2>& SoundRecord1<T, T2>::operator+(const SoundRecord1<T, T2>& a)  {  SoundRecord1 v;  v.name = new char[strlen(a.name) + strlen(name) + 2];  v.power = new char[strlen(a.power) + strlen(power) + 2];  strcpy(v.name, name);  strcat(v.name, ",");  strcat(v.name, a.name);  strcpy(v.power, power);  strcat(v.power, ",");  strcat(v.power, a.power);  v.price = price + a.price;  v.kolvo = kolvo + a.kolvo;  v.Getname();  v.Getpr();  v.GetPrice();  v.Getkolvo();  return v;  }  template <class T, class T2 >bool SoundRecord1<T, T2>::operator>(const SoundRecord1<T, T2>& a)  {  return ((!strcmp(name, a.name)) &&  price > a.price &&  kolvo > a.kolvo);  }  template <class T, class T2 >SoundRecord1<T, T2>& SoundRecord1<T, T2>::operator=(const SoundRecord1<T, T2>& r)  {  price = r.price;  kolvo = r.kolvo;  power = r.power;  name = r.name;  return \*this;  }  template <class T, class T2 >bool SoundRecord1<T, T2>::operator==(const SoundRecord1<T, T2>& a)  {  return ((!strcmp(name, a.name)) &&  price == a.price &&  kolvo == a.kolvo);  }  template <class T, class T2 >void SoundRecord1<T, T2>::destroy() {  delete[] name;  delete[] power;  cout << "Memory has been succesfully cleaned" << endl;  };  template <class T, class T2 >SoundRecord1<T, T2> SoundRecord1<T, T2>::Sum(const SoundRecord1<T, T2>& a)  {  SoundRecord v;  v.name = new char[strlen(a.name) + strlen(name) + 2];  v.power = new char[strlen(a.power) + strlen(power) + 2];  strcpy(v.name, name);  strcat(v.name, ",");  strcat(v.name, a.name);  strcpy(v.power, power);  strcat(v.power, ",");  strcat(v.power, a.power);  v.price = price + a.price;  v.kolvo = kolvo + a.kolvo;  v.Getname();  v.Getpr();  v.GetPrice();  v.Getkolvo();  return v;  }  template <class T, class T2 >bool SoundRecord1<T, T2>::Compair(SoundRecord1<T, T2>& a)  {  return ((!strcmp(name, a.name)) &&  price == a.price &&  kolvo == a.kolvo);  }  //ОДИНОЧНОЕ НАСЛЕДОВАНИЕ  class Lamp : public SoundRecord<char>  {  public:  void SetMark(char\* a);  void GetMark();  void SetAvailability(bool i);  void GetAvailability();  Lamp() {  Mark = new char[11];  strcpy(Mark, "none");  }  private:  char\* Mark;  bool Availability;  };  void Lamp::SetMark(char\* a) {  Mark = new char[strlen(a) + 1];  strcpy(Mark, a);  }  void Lamp::GetMark()  {  cout << "Наименование: " << Mark << "\n";  }  void Lamp::GetAvailability()  {  if (Availability) {  cout << "Имеется в наличии: да\n";  }  else {  cout << "Имеется в наличии: нет\n";  }  }  void Lamp::SetAvailability(bool i)  {  if (i)  {  Availability = 1;  }  else  {  Availability = 0;  }  }  //МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ  class Frequency : public SoundRecord<char>  {  public:  Frequency() {  chastota = 400;  }  void GetZgib() {  cout << "Частота = " << chastota << " Hz\n";  }  void SetZgib(int n) {  chastota = n;  }  private:  int chastota;  };  class speakers : public Frequency, public Lamp  {  public:  void GetHead() {  cout << "Размер динамиков = " << Head << "дюйма \n";  }  void SetHead(float n) {  Head = n;  }  private:  float Head;  };  //speakers::speakers()  //{  //  //}  int main()  {  setlocale(LC\_ALL, "Russian");  PrintLine();  Lamp i6;  cout << "проверка работы методов унаследованных от базового класса\n";  cout << "\n";  i6.Setname((char\*)"VOX");  i6.Getname();  PrintLine();  Lamp i7;  i7 = i6;  i7.Getname();  PrintLine();  cout << "проверка работы методов производного класса\n";  cout << "\n";  i6.SetMark((char\*)"Marshall");  i6.GetMark();  i6.SetAvailability(1);  i6.GetAvailability();  i7.SetAvailability(0);  i7.GetAvailability();  PrintLine();  SoundRecord<int> G5;  cout << "проверка работы множественного наследования\n";  cout << "\n";  speakers f3;  f3.GetZgib();  f3.GetAvailability();  f3.GetMark();  f3.SetZgib(800);  f3.GetZgib();  f3.SetHead(12.3);  f3.GetHead();  PrintLine();  } |

**Тестирование:**

