Wariant 2 2022

Nazwisko

Imię

Numer Indeksu

**Proszę wszystkie poprawienia oraz odpowiedzi wpisywać w kolorze czarnym.**

|  |  |  |
| --- | --- | --- |
| N | Zadanie | Ocena |
| 1 | void main()  {  double a;  int b = 1, c = 2;  //używając union, umieścić w obszarze pamięci, zajmowanym zmienną a, //dwie liczby int b, c;  fun(a);  }  void fun(double aa)  {  //odczytać zmienne b, c, umieszczone w pamięci zmiennej a;  }  #include <iostream>  #include<vector>  #include<string>  #include <algorithm>  using namespace std;  union data1 {  int b[2];  double a;  } ;  void fun(double aa)  {  data1 unia;  unia.a = aa;  //odczytać zmienne b, c, umieszczone w pamięci zmiennej a;  cout << unia.b[0] << endl;  cout << unia.b[1] << endl;  }  void main()  {  double a;  int b = 1, c = 2;  //używając union, umieścić w obszarze pamięci, zajmowanym zmienną a, //dwie liczby int b, c;  data1 unia;  unia.b[0] = b;  unia.b[1] = c;  a = unia.a;  fun(a);  } |  |
| 2 | Ile razy wywołuje się konstruktor, destruktor?  class A  {  public:  A();  A(const A &);  ~A();  };  void fun(A ob)  {  …………..  }  int main()  {  A ob;  fun(A); //chuje zły argument fdałes  return 0;  }  #pragma warning(disable:4996)  #include <iostream>  #include<vector>  #include<string>  #include <algorithm>  #include <typeinfo>  using namespace std;  class A  {  public:  A() { cout << "A" << endl; };  A(const A&){ cout << "A kopi" << endl; };  ~A(){ cout << "destr A" << endl; };  };  void fun(A ob)  {  // …………..  }  int main()  {  A ob;  fun(ob);  return 0;  }  A  A kopi  destr A  destr A |  |
| 3. | Czy podany fragment kodu jest poprawny?  int \* fun()  {  int i = 10;  return &i;  }  int main()  {  int \* j;  j = fun();  cout << \*j << endl;  return 0;  }  **Wyświetli się 10, ale zmienna lokalna jest niszczona po wyjściu z fun, a tu jest zwrócony do niej adres, a tamta pamiec może zostac nadpisana** |  |
| 4. | Kiedy i dla czego stosuję się destruktor wirtualny?  **Destruktora wirtualnego używa się w klasach bazowych (takich, które można odziedziczyć ). Takie destruktory umożliwiają nam wywołanie destruktorów klas ,,dzieci”(potomnych). Mają one zwalniać nam zasobów przydzielonych podczas pracy z obiektem.** |  |
| 5. | class A  {  public:  void f() { cout<< “A\n”; }  };  class C : public A  {  void f() { cout<< “C\n”; }  };  int main()  {  A \*pA;  C ob\_c;  pA = &ob\_c;  pA->f();  return 0;  }  **Odpowiedź: A,a z virtual C** |  |
| 6 | class B  {  public:  virtual void f() { ……. }  };  class D : public B  {  public:  void f() { ………}  };  int main()  {  B ob\_B, \*pB;  D ob\_D;  pB = &ob\_B;  cout<< “do którego typu wskazuje pB ?“ <<**typeid(\*pB).name()**<<endl;  pB = &ob\_D;  cout<< “ do którego typu wskazuje pB ?“ <<**typeid(\*pB).name()**<< endl;  return 0;  }  Wyznaczyć typ obiektu, do którego wskazuje wskaźnik pB, i wyprowadzić nazwę typu na monitor.  #include <iostream>  #include<vector>  #include<string>  #include <algorithm>  #include <typeinfo>  using namespace std;  class B  {  public:  void f() {  //…….  }  };  class D : public B  {  public:  void f() {  // ………  }  };  int main()  {  B ob\_B, \* pB;  D ob\_D;  pB = &ob\_B;  cout << typeid(\*pB).name() << endl;  pB = &ob\_D;  cout << typeid(\*pB).name() << endl;  return 0;  }  B  B  Z virtual w B  B  D |  |
| 7. | class coord  {  double \*cr; //cr[0] – x, cr[1] - y  public:  //Podaj deklaracji metod i operatorów jakie są potrzebny dla poprawnego działania //następującego kodu.  **Coord(double xx, double yy);**  **Coord(const& Coord oth);**  **Operator =(const& Coord oth);**  };  template<classT>voidswap\_arg(T&a, T&b)  {  T c = a;  a = b;  b = c;  }  void main()  {  coordaaa(0, 1), bbb(1, 0);  swap\_arg(aaa, bbb);  cout << aaa << endl;  }  #include <iostream>  #include<vector>  #include<string>  #include <algorithm>  #include <typeinfo>  using namespace std;  class coord  {  double\* cr; //cr[0] – x, cr[1] - y  public:  //Podaj deklaracji metod klasy i operatorów jakie są potrzebny dla poprawnego działania //następującego kodu.  coord(double x, double y){  cr = new double[2];  cr[0] = x;  cr[1] = y;  }  coord(coord& kopia) {  cr = new double[2];  cr[0] = kopia.cr[0];  cr[1] = kopia.cr[1];  };  coord& operator = (coord & rhs){  cr[0] = rhs.cr[0];  cr[1] = rhs.cr[1];  return \*this;  }  friend ostream& operator<< (ostream& lhs, const coord& rhs);  ~coord() {  if (cr) delete[] cr;  }  };  ostream& operator<< (ostream& lhs, const coord& rhs) {  lhs << rhs.cr[0] << " " << rhs.cr[1] << endl;  return lhs;  }  template <class T> void swap\_arg(T& a, T& b)  {  T c = a;  a = b;  b = c;  }  void main()  {  coord aaa(0, 1), bbb(1, 0);  swap\_arg(aaa, bbb);  cout << aaa << endl;  } |  |
| 8. | Mamyhierarchieklas:  class A  {  int a;  public:  A(int aa) : a(aa) {}  ~A() {}  };  class B  {  int b;  public:  B(int bb) : b(bb) {}  ~B() {}  };  class C : public B, public A  {  int c;  public:  //stworzyć konstruktor sparametryzowany dla przekazywania danych zmiennym //a, b, c.  **C(int aa, int bb, int cc) : A(aa), B(bb){**  **C**  **}**  ~C() {}  };  void main()  {  C ob(1, 2, 3);  return;  //w jakiej kolejności wywołują się konstruktory? Destruktory?  }  #pragma warning(disable:4996)  #include <iostream>  #include<vector>  #include<string>  #include <algorithm>  #include <typeinfo>  using namespace std;  class A  {  int a;  public:  A(int aa) : a(aa) { cout << "A" << endl; }  ~A() { cout << "destr A" << endl; }  };  class B  {  int b;  public:  B(int bb) : b(bb) { cout << "B" << endl; }  ~B() { cout << "destr B" << endl; }  };  class C : public B, public A  {  int c;  public:  //stworzyć konstruktor sparametryzowany dla przekazywania danych zmiennym //a, b, c.  C(int aa, int bb, int c) : A(aa), B(bb), c(c) { cout << "C" << endl; };  ~C() { cout << "destr C" << endl; }  };  void main()  {  C ob(1, 2, 3);  return;  //w jakiej kolejności wywołują się konstruktory? Destruktory?  }  B  A  C  destr C  destr A  destr B |  |