Jeżeli klasa dziedziczy po strukturze to dziedziczenie jest domyślnie zabezpieczone

1.

* Jeżeli klasa dziedziczy po strukturze to dziedziczenie jest domyślnie zabezpieczone.   
  TAK / NIE
* Jeżeli klasa dziedziczy po klasie to dziedziczenie jest domyślnie prywatne.  
  TAK / NIE
* Jeżeli struktura dziedziczy po klasie to dziedziczenie jest domyślnie publiczne.  
  TAK / NIE

"class has private inheritance by default”

2.

* Wygenerowany przez kompilator konstruktor kopiujący nie ma listy inicjalizacyjnej. TAK / NIE
* Wygenerowany przez kompilator konstruktor kopiujący inicjalizuje wszystkie składowe klasy.  
  TAK / NIE
* Wygenerowany przez kompilator konstruktor kopiujący nie inicjalizuje żadnych składowych klasy.  
  TAK / NIE

3.

Które z poniższych linii powodują błędy kompilacji?

struct Parent {

virtual void method() = 0; };

struct Child: public Parent {

virtual void method() {}};

* Parent parent;

TAK / NIE

* Parent \*parent = new Parent;  
  TAK / NIE
* Parent \*parent = new Child;  
  TAK / NIE

4.

* Jeśli iterator modyfikujący i nie modyfikujący wskazują na ten sam element wektora, to ich porównanie operatorem == daje w wyniku prawdę.  
  TAK / NIE
* Porównanie dwóch iteratorów operatorem > daje w wyniku prawdę jedynie jeśli iterator po lewej stronie operatora wskazuje na element o większym indeksie niż iterator po prawej stronie.  
  TAK / NIE
* Od iteratora modyfikującego można odjąć niemodyfikujący.  
  TAK / NIE

5. Które z poniższych wycinków kodu powodują błędy kompilacji?

std::string operator <<(std::string &string, char character) {

return string += character; }

* std::string string; string << 'a';

TAK / NIE

* std::string string; string << 'a' << 'b';

TAK / NIE

* std::string string; std::cout << string;

TAK / NIE

6.

* Jeżeli argument funkcji jest zadeklarowany przy pomocy szablonu std::function, to można jako ten argument przekazać zwykłą funkcję lub wyrażenie lambda.

TAK / NIE

* Funkcja przyjmująca jako argument inną funkcję lub wyrażenie lambda to przykład funkcji wyższego rzędu.

TAK / NIE

* Argumentami funkcji wyższego rzędu mogą być tylko takie funkcje lub wyrażenia lambda, które zwracają wynik.

TAK / NIE

7. Które z poniższych linii powodują błędy kompilacji?

template<typename Type>

void function(Type argument) {

argument.size(); }

* function(std::vector<int>());

TAK / NIE

* function(int());

TAK / NIE

* function(std::string());

TAK / NIE

8. Przy tworzeniu obiektu klasy pochodnej:

* Ciało konstruktora klasy bazowej wykonuje się przed ciałem konstruktora klasy pochodnej.

TAK / NIE

* Lista inicjalizacyjna konstruktora klasy bazowej wykonuje się przed ciałem konstruktora klasy pochodnej.

TAK / NIE

* Konstruktor klasy bazowej wykonuje się tylko jeśli jest wywołany na liście inicjalizacyjnej konstruktora klasy pochodnej.

TAK / NIE

9. Które z poniższych pętli powodują błędy kompilacji?

std::set<int> set {6, 3, 7};

* for (int element: set) {std::cout << element; }

TAK / NIE

* for (auto &element: set) {std::cout << element; }

TAK / NIE

* for (int &element: set) {std::cout << element; }

TAK / NIE

10. Konstruktor klasy pochodnej może zainicjalizować zmienną odziedziczoną:

* Tylko wywołując konstruktor klasy bazowej.   
  TAK / NIE
* Bezpośrednio na swojej liście inicjalizacyjnej.   
  TAK / NIE
* W swoim ciele.  
  TAK / NIE

11.

Które z poniższych linii powodują błędy kompilacji?

struct Object {

Object(int a) {}};

* Object object();

TAK / NIE

* Object object;

TAK / NIE

* Object object(0);

TAK / NIE

12. Co wypisuje poniższy kod?

std::set<int, std::function<bool(int, int)>>

set({2, 1, 3}, [](int element1, int element2){return element2 < element1; });

for (auto iterator = set.cbegin(); iterator != set.cend(); ++iterator) {

std::cout << \*iterator; }

* 321  
  TAK / NIE
* 213  
  TAK / NIE
* 123  
  TAK / NIE

13. Co wypisuje poniższy kod?

std::map<int, int> map {{7, 1}, {5, 3}, {3, 8}};

std::cout << map['3'];

std::cout << map.find(7)->first;

std::cout << map.size();

* 873  
  TAK / NIE
* 073  
  TAK / NIE
* 074  
  TAK / NIE

14. Które z poniższyh wycinków kodu powodują błędy kompilacji?

struct Object {

void operator [](int n) {}};

* Object object; object[0];

TAK / NIE

* const Object object; object[0];

TAK / NIE

* Object()[0];

TAK / NIE

@15.

* W niektórych sytuacjach do składowej odziedziczonej klasa pochodna może się odwołać bez kwalifikatora zakresu.

TAK / NIE

* Do składowej odziedziczonej klasa pochodna może się odwołać tylko przy użyciu kwalifikatora zakresu.

TAK / NIE

* Do składowej odziedziczonej klasa pochodna zawsze może się odwołać przy użyciu kwalifikatora zakresu.

TAK / NIE

16.

Co wypisuje poniższy program

int counter = 0;

struct Parent {

~Parent() {++counter; }};

struct Child: public Parent {

virtual ~Child() {--counter; }};

int main() {

Parent \*parent = new Parent, \*child = new Child;

delete parent, child;

std::cout << counter; }

* 2

TAK / NIE

* 1

TAK / NIE

* 0

TAK / NIE

@18.

* Różnica między końcowym a początkowym iteratorem wektora to indeks hipotetycznego elementu za ostatnim elementem wektora.   
  TAK / NIE
* Różnica między iteratorami dwóch elementów wektora jest równa różnicy między ich indeksami.   
  TAK / NIE
* Różnica między iteratorem elementu a iteratorem początkowym wektora to indeks tego elementu.  
  TAK / NIE jeśli drugie to i trzecie iguess

@19.

* Każde wyrażenie lambda ma inny typ.  
  TAK / NIE
* Zamiast używać słowa kluczowego auto, w definicji nazwanego wyrażenia lambda można samemu podać jego typ. (function<OutType<InType...>>???)  
  TAK / NIE
* Typ wyrażenia lambda jest opisany szablonem std::function.  
  TAK / NIE

20. Które z poniższych linii powodują błędy kompilacji?

struct Object {

static int method();

static int variable1;

int variable2; };

* int Object::method() {return variable2; }   
  TAK / NIE
* int Object::method() {return Object::method(); }   
  TAK / NIE
* int Object::method() {return variable1; }  
  TAK / NIE

@21. Które z poniższych pętli poprawnie wypisują wszystkie elementy wektora?

* for (auto i = vector.begin(); i < vector.end(); std::cout << \*++i);  
  TAK / NIE
* for (auto i = vector.begin(); i < vector.end(); std::cout << ++\*i);  
  TAK / NIE
* for (auto i = vector.begin(); i < vector.end(); std::cout << \*i++);  
  TAK / NIE

@22.

* W pojemniku typu std::map wartości mogą się powtarzać.   
  TAK / NIE
* W pojemniku typu std::map klucze mogą się powtarzać.   
  TAK / NIE
* W pojemniku typu std::map para o zadanym kluczu i wartości może wystąpić najwyżej raz.  
  TAK / NIE

@23.

Które z poniższych definicji klas powodują błędy kompilacji?

* class Object {Object &object; };

TAK / NIE

* class Object {Object \*object; };

TAK / NIE

* class Object {Object object; };

TAK / NIE

@24.

* Wygenerowany przez kompilator konstruktor bezargumentowy inicjalizuje wszystkie składowe klasy wartościami domyślnymi.

TAK / NIE

* Wygenerowany przez kompilator konstruktor bezargumentowy nie ma listy inicjalizacyjnej.

TAK / NIE

* Wygenerowany przez kompilator konstruktor bezargumentowy nie inicjalizuje żadnych składowych klasy.

TAK / NIE

@25.

* Na podstawie indeksu można wyznaczyć iterator elementu wektora.  
  TAK / NIE wektor.begin() + indeks == iterator elementu wektora? x
* Istnieje niejawna konwersja iteratora modyfikującego do niemodyfikującego.

TAK / NIE

* Istnieje konwersja indeksu do iteratora elementu w wektorze.

TAK / NIE

26.

* Jeżeli trzecim argumentem wywołania algorytmu std::for\_each jest funkcja lub wyrażenie lambda przyjmujące referencję niemodyfikującą, to pierwszymi dwoma argumentami muszą być iteratory niemodyfikujące.

TAK / NIE

* Jeżeli pierwszymi dwoma argumentami wywołania algorytmu std::for\_each są iteratory modyfikujące, to trzecim musi być funkcja lub wyrażenie lambda przyjmujące referencję modyfikującą.

TAK / NIE

* Jeżeli trzecim argumentem wywołania algorytmu std::for\_each jest funkcja lub wyrażenie lambda przyjmujące referencję modyfikującą, to pierwszymi dwoma argumentami muszą być iteratory modyfikujące.

TAK / NIE

27. Które z poniższych linii powodują błędy kompilacji?

struct Object {

static void method() {}};

* Object::method();

TAK / NIE

* Object()->method();

TAK / NIE

* Object().method();

TAK / NIE

28. Co wypisuje poniższy program?

struct Parent {

int method1() {return 1; }

virtual int method2() {return 2; }};

struct Child: public Parent {

int method1() {return 3; }

int method2() {return 4; }};

int main() {

Parent \*parent = new Child;

std::cout << parent->method1() << parent->method2(); }

* 34

TAK / NIE

* 12

TAK / NIE

* 14

TAK / NIE

29. Które z poniższych wycinków kodu powodują błędy kompilacji?

struct Object {

void operator =(Object &object) {}};

* Object object1, object2, object3; object1 = object2 = object3;  
  TAK / NIE
* Object object1; object1 = Object();  
  TAK / NIE
* Object object1, object2; object1 = object2;  
  TAK / NIE

30. Co wypisuje poniższy program?

int counter = 0;

struct Object {

~Object() {++counter; }};

int main() {

{ Object array[10];

array[0].~Object(); }

std::cout << counter; }

* 10  
  TAK / NIE
* 1   
  TAK / NIE
* 11  
  TAK / NIE

@31.

* Wyrażenie lambda może przechwycić zmienną lokalną przez wartość lub referencję.  
  TAK / NIE
* Wyrażenie lambda może swobodnie korzystać ze zmiennych globalnych.  
  TAK / NIE
* Domyślnie wyrażenie lambda może korzystać ze zmiennych lokalnych zdefiniowanych w tym samym zakresie, co wyrażenie.  
  TAK / NIE chuj wie co znaczy domyślnie, jak wrzuci się [=] albo [&] to tak, jeśli się wrzuci [] to nie

32. Co wypisuje poniższy program?

template <typename Iterator>

void print(Iterator begin, Iterator end) {

for (Iterator iterator = begin; iterator != end; ++iterator) {

std::cout << \*iterator; }}

int main() {

int array[] = {2, 5, 7};

print(array, array + 3); }

* 257  
  TAK / NIE
* co innego   
  TAK / NIE
* 752  
  TAK / NIE

33.

* Klasa pochodna ma dostęp do składowych zabezpieczonych klasy bazowej.  
  TAK / NIE
* Klasa bazowa ma dostęp do składowych publicznych klasy pochodnej.  
  TAK / NIE
* Klasa pochodna ma dostęp do składowych prywatnych klasy bazowej.  
  TAK / NIE

34. Istnieje niejawna konwersja:

* Referencji klasy bazowej do referencji klasy pochodnej.  
  TAK / NIE
* Referencji klasy pochodnej do referencji klasy bazowej.  
  TAK / NIE
* Wskaźnika klasy pochodnej do wskaźnika klasy bazowej.  
  TAK / NIE

35. Co wypisuje poniższy program?

struct Object {

int method() {return 1; }

int method() const {return 2; }};

int main() {

Object object;

std::cout << object.method() << Object().method(); }

* co innego  
  TAK / NIE
* 11   
  TAK / NIE
* 12  
  TAK / NIE

36.

* Za pośrednictwem iteratora modyfikującego można modyfikować wartości pojemnika typu std::map.  
  TAK / NIE
* Pojemnik typu std::map nie ma iteratorów modyfikujących.  
  TAK / NIE
* Za pośrednictwem iteratora modyfikującego można modyfikować klucze pojemnika typu std::map.  
  TAK / NIE

37. Które z poniższych wycinków kodu mogą powodować naruszenie ochrony pamięci?

* int \*pointer = new int; delete pointer; delete pointer;  
  TAK / NIE
* int \*pointer = nullptr; delete pointer;  
  TAK / NIE
* int \*pointer = new int[10]; delete pointer;  
  TAK / NIE // possible memory leak, not a memory protection violatio

38.

Dla których definicji klasy poniższa funkcja powoduje błędy kompilacji?

int main() {

Object().variable; }

* struct Object {public: int variable; };  
  TAK / NIE
* class Object {int variable; };  
  TAK / NIE
* struct Object {int variable; };  
  TAK / NIE

@39.

* Jeżeli iterator początkowy wycinka jest równy jego iteratorowi końcowemu, to wyłuskanie tych iteratorów zawsze jest błędem.  
  TAK / NIE
* Wyłuskanie iteratora początkowego wektora nigdy nie jest błędem. // wektor pusty  
  TAK / NIE
* Wyłuskanie iteratora końcowego wektora zawsze jest błędem.  
  TAK / NIE

@40.

* W pojemniku typu std::multimap wartości mogą się powtarzać.  
  TAK / NIE
* W pojemniku typu std::multimap klucze mogą się powtarzać.  
  TAK / NIE
* W pojemniku typu std::multimap para o zadanym kluczu i wartości może wystąpić najwyżej raz.  
  TAK / NIE

41. Które z poniższych linii drukują napis 00?

struct Object {

Object() {std::cout << "0"; }};

* Object object1, object2 = object1;  
  TAK / NIE
* Object object1, object2;  
  TAK / NIE
* Object object1, object2 = Object();  
  TAK / NIE

42. Które z poniższych wycinków powodują błędy kompilacji?

class Parent {};

class Child: public Parent {};

* Parent array[10]; array[0] = Parent();

TAK / NIE

* Parent array[10]; array[0] = Child();

TAK / NIE

* Parent \*array[10]; array[0] = new Child();

TAK / NIE

@43.

* Iteratora stałego nigdy nie można przesunąć na następny element.

TAK / NIE

* Iteratora niemodyfikującego nigdy nie można przesunąć na następny element.

TAK / NIE

* Za pośrednictwem iteratora stałego można w pewnych przypadkach zmodyfikować element.

TAK / NIE

@44.

* Kompilator generuje konstruktor bezargumentowy jedynie jeśli programista nie zadeklarował w klasie konstruktora domyślnego.

TAK / NIE

* Kompilator generuje konstruktor bezargumentowy jedynie jeśli programista nie zadeklarował w klasie żadnego konstruktora.

TAK / NIE

* Kompilator generuje konstruktor bezargumentowy jedynie jeśli programista nie zadeklarował w klasie konstruktora kopiującego.

TAK / NIE

45.

Które z poniższych wycinków kodu powodują błędy kompilacji?

struct Object {

void operator +(const Object &object) const {}};

* Object() + Object();

TAK / NIE

* Object object1, object2; operator +(object1, object2);

TAK / NIE

* Object object1, object2; object1.operator +(object2);  
  TAK / NIE

@46. Jeżeli w klasie bazowej i pochodnej są zdefiniowane składowe o jednakowych nazwach, to klasa pochodna może się posługiwać:

* Tylko składową zdefiniowaną w klasie bazowej.  
  TAK / NIE
* Obiema składowymi.  
  TAK / NIE
* Tylko składową zdefiniowaną w klasie pochodnej.  
  TAK / NIE

@47.

* Pojemnik typu std::set<int> nia ma iteratorów typu std::set<int>::iterator.  
  TAK / NIE
* Pojemnik typu std::set<int> ma iteratory typu std::set<int>::iterator ale są one niemodyfikujące.  
  TAK / NIE
* Pojemnik typu std::set nie ma iteratorów modyfikujących.  
  TAK / NIE

48. Dla których definicji wektora poniższy wycinek kodu wypisuje liczbę 5?

int index = 0;

int result = std::count\_if(vector.begin(), vector.end(),  
 [&](int element) {return index++ == element; });

std::cout << result << std::endl;

* std::vector<int> vector {9, 4, 2, 3, 0, 1, 6, 7, 8, 5};  
  TAK / NIE
* std::vector<int> vector {5, 1, 8, 3, 4, 7, 6, 2, 0, 9};  
  TAK / NIE
* std::vector<int> vector {0, 3, 2, 6, 7, 5, 1, 4, 8, 9};  
  TAK / NIE