1. Według standardu języka C++ rozmiar automatycznej tablicy:

* Może być zadany tylko liczbową stałą dosłowną (np. 5, 100 itp.)
  + Prawda / Tak
  + Fałsz / Nie
* Może być zadany stałą nawet gdy jej wartość nie jest znana podczas kompilacji
  + Prawda / Tak
  + Fałsz / Nie
* Może być zadany zmienną o ile jej wartość jest znana podczas kompilacji
  + Prawda / Tak
  + Fałsz / Nie

Nauka polskiego o 2 rano: pozdro pimpusie

Pójdę do sklepu o ile nie będzie padać

~(Pójdę do sklepu o ile nie będzie padać) =

polski[nie pójdę do sklepu nawet jeśli nie będzie padać]

~(Jeśli nie będzie padać => pójdę do sklepu) =

(nie będzie padać ^ ~pójdę do sklepu) =

nie będzie padać ^ nie pójdę do sklepu

Jej wartość jest znana => może być zadana zmienną

~(Jej wartość jest znana => może być zadana zmienną) =

Jej wartość jest znana ^ ~może być zadana zmienną =

Jej wartość jest znana ^ nie może być zadana zmienną

Nie może być zadana zmienną nawet jeśli jej wartość jest znana

Jej wartość jest znana <=> może być zadana zmienną

~(Jej wartość jest znana => może być zadana zmienną) v ~(może być zadana zmienną => Jej wartość jest znana) = [Jej wartość jest znana ^ ~może być zadana zmienną] v [może być zadana zmienną ^ ~Jej wartość jest znana] =

[Jej wartość jest znana i nie może być zadana zmienną] lub

[Jej wartość nie jest znana i może być zadana zmienną] =

[Jej wartość jest znana i nie jest zadana zmienną] lub

[Jej wartość nie jest znana i jest zadana zmienną] =

[Jej wartość jest znana i nie może być zadana zmienną] lub

[Jej wartość nie jest znana i może być zadana zmienną] =

[nie może być zadana zmienną, której wartość jest znana] lub

[może być zadana zmienną, której wartość nie jest znana] =

0 lub 1 = 1

Zaprzeczenie jest prawdą -> zdanie jest fałszem

>>tak wykładowcę pokurlisiowaje

>>tak ja też nie chcę tu być

Powyżej test z polskiego, ponieważ jest to implikacja I nie musi być znana w trakcie kompilacji

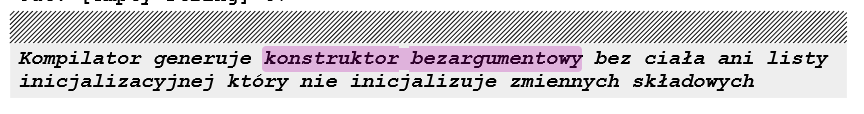
Jeśli podczas kompilacji nie znana -> nie może być

Jeśli podczas kompilacji znana -> może być



2.

* Wygenerowany przez kompilator konstruktor bezargumentowy nie inicjalizuje żadnych składowych klasy.
  + Prawda / Tak
  + Fałsz / Nie
* Wygenerowany przez kompilator konstruktor bezargumentowy nie ma listy inicjalizacyjnej.
  + Prawda / Tak
  + Fałsz / Nie
* Wygenerowany przez kompilator konstruktor bezargumentowy inicjalizuje wszystkie składowe klasy wartościami domyślnymi.
  + Prawda / Tak
  + Fałsz / Nie



3. Które z poniższych pętli poprawnie drukują wszystkie 10 elementów dynamicznie zaalokowaniej tablicy table?

int \*table = (int\*)std::malloc(10 \* sizeof(int));

* for (int index = 0; index < sizeof(table); std::cout << table[index++]);
  + Prawda / Tak
  + Fałsz / Nie (wypisuje 8)
* for (int index = 0; index < table[index]; std::cout << table[index++]);
  + Prawda / Tak
  + Fałsz / Nie (wypisuje 1)
* for (int index = 0; index < sizeof(table) / sizeof(int); std::cout << table[index++]);
  + Prawda / Tak
  + Fałsz / Nie (wypisuje 2)

4. Które z poniżysz linii powodują błędy kompilacji?

double variable;

* char \*pointer = nullptr;
  + Prawda / Tak
  + Fałsz / Nie
* double \*pointer = &variable;
  + Prawda / Tak
  + Fałsz / Nie
* int \*pointer;
  + Prawda / Tak
  + Fałsz / Nie

5. Które z poniższych wycinków kodu powodują błędy kompilacji?

struct Object {

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

* Object object1, object2; object1.operator +(object2);
  + Prawda / Tak
  + Fałsz / Nie
* Object object1, object2; operator +(object, object2);
  + Prawda / Tak
  + Fałsz / Nie
* Object() + Object();
  + Prawda / Tak
  + Fałsz / Nie

6. Które z poniższych linii powodują błędy kompilacji?

const int variable = 0;

* int \*pointer = &variable;
  + Prawda / Tak
  + Fałsz / Nie
* const int \*pointer = &variable;
  + Prawda / Tak
  + Fałsz / Nie
* const int \*pointer;
  + Prawda / Tak
  + Fałsz / Nie

7. Które z poniższych instrukcji mogą powodować błędy wykonania?

int variable;

int \*pointer;

* std::free(pointer);
  + Prawda / Tak (powoduje build error w vs, ale w pozostały powoduje runtime error)
  + Fałsz / Nie
* std::free(&variable);
  + Prawda / Tak
  + Fałsz / Nie
* std::free(nullptr);
  + Prawda / Tak
  + Fałsz / Nie

8. Które z poniższych wycinków kodu powodują błędy kompilacji?

struct Object {

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

* Object object1; object1 = Object();
  + Prawda / Tak
  + Fałsz / Nie
* Object object1, object2; object1 = object2;
  + Prawda / Tak
  + Fałsz / Nie
* Object object1, object2, object3; object1 = object2 = object3;
  + Prawda / Tak
  + Fałsz / Nie

9. Które z poniższych linii powodują błędy kompilacji?

struct Object {

static void method() {}};

* Object::method();
  + Prawda / Tak
  + Fałsz / Nie
* Object().method();
  + Prawda / Tak
  + Fałsz / Nie
* Object()->method();
  + Prawda / Tak
  + Fałsz / Nie

10. Które z poniższych linii powodują błędy kompilacji?

* int table[5] = {};
  + Prawda / Tak
  + Fałsz / Nie
* int table[];
  + Prawda / Tak
  + Fałsz / Nie
* int table[] = {5};
  + Prawda / Tak
  + Fałsz / Nie

11. Które z poniższych linii drukują napis 00?

struct Object {

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

* Object object1, object2; object2 = Object();
  + Prawda / Tak
  + Fałsz / Nie 000
* Object object1, object2;
  + Prawda / Tak
  + Fałsz / Nie
* Object object1, object2 = object1;
  + Prawda / Tak
  + Fałsz / Nie 0

12. Które z poniższych linii powodują błędy kompilacji?

struct Structure{

int member1;

const int member2; };

* Structure structure {};
  + Prawda / Tak
  + Fałsz / Nie
* Structure structure;
  + Prawda / Tak
  + Fałsz / Nie
* Structure structure {0};
  + Prawda / Tak
  + Fałsz / Nie

13.

* Różnica między adresem danego elementu tablicy a adresem jej pierwszego elementu to indeks danego elementu w tablicy
  + Prawda / Tak
  + Fałsz / Nie
* Różnica końcowego i początkowego wskaźnika tablicy to indeks pierwszego elementu za tablicą
  + Prawda / Tak
  + Fałsz / Nie
* Różnica końcowego i początkowego wskaźnika tablicy to liczba elementów tablicy
  + Prawda / Tak
  + Fałsz / Nie

14. Które z poniższych pętli poprawnie wypisują wszystkie elementy tablicy table?

int table[5];

* for (int \**pointer = table; pointer < table + 5; std::cout << ++*\*pointer);
  + Prawda
  + Fałsz
* for (int \**pointer = table; pointer < table 5; std::cout <<* \*pointer++)
  + Prawda
  + Fałsz
* ***for (int \*pointer = table; pointer < table + 5; std::cout << \*pointer++) ←- dopisane!!!***
  + Prawda / Tak
  + Fałsz / Nie
* for (int \**pointer = table; pointer < table + 5; std::cout << \*++*pointer)
  + Prawda / Tak
  + Fałsz / Nie

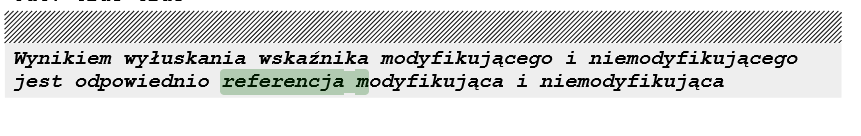
zależności od interpretacji, przesunięcie z tablicy, nie wypisuje wszystkich z tablicy, ale wartości przy kompilacji w 2 próbach były takie same (mimo, że adresy w pamięci NIE są)

15. Dla których definicji tablicy table standard języka C++ gwarantuje, że wyrażenie sizeof(table) ma wartość 4?

* bool table[4];
  + Prawda / Tak
  + Fałsz / Nie
* char table[4];
  + Prawda / Tak
  + Fałsz / Nie
* int table[1];
  + Prawda / Tak
  + Fałsz / Nie

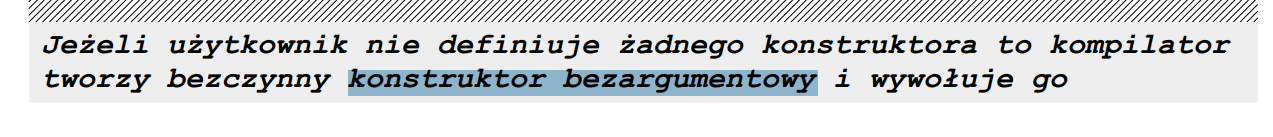
16.

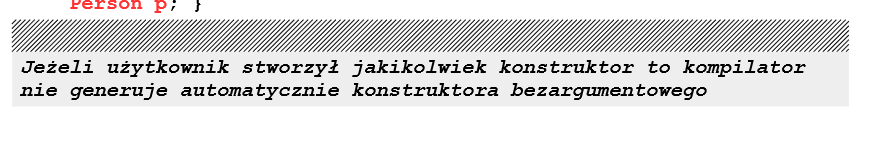
* Wyłuskanie wskaźnika stałego zawsze daje referencje niemodyfikującą
  + Prawda / Tak
  + Fałsz / Nie
* Wyłuskanie dowolnego wskaźnika zawsze daje jakiś rodzaj referencji
  + Prawda / Tak
  + Fałsz / Nie
* Wyłuskanie wskaźnika modyfikującego zawsze daje referencje modyfikującą
  + Prawda / Tak
  + Fałsz / Nie

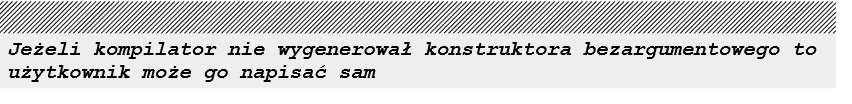
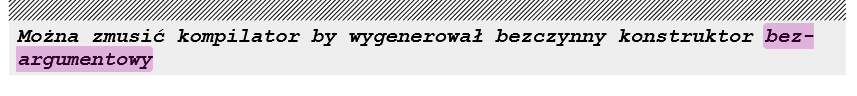


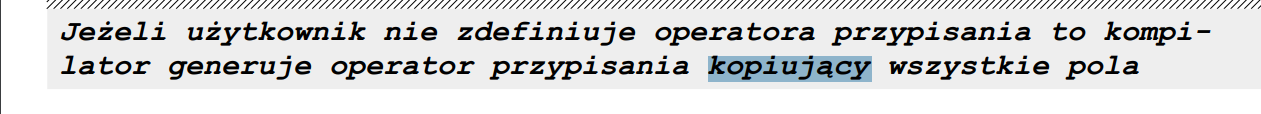
17.

* Kompilator generuje konstruktor bezargumentowy jedynie jeśli programista nie zadeklarował w klasie żadnego konstruktora.
  + Prawda / Tak
  + Fałsz / Nie
* Kompilator generuje konstruktor bezargumentowy jedynie jeśli programista nie zadeklarował w klasie konstruktora domyślnego.
  + Prawda / Tak
  + Fałsz / Nie
* Kompilator generuje konstruktor bezargumentowy jedynie jeśli programista nie zadeklarował w klasie konstruktora kopiującego
  + Prawda / Tak
  + Fałsz / Nie









18.

* Można pobrać adres każdego obiektu tymczasowego bez nazwy.
  + Prawda / Tak
  + Fałsz / Nie
* Można pobrać adres każdej nazwanej zmiennej
  + Prawda / Tak
  + Fałsz / Nie
* Można pobrać adres każdej nazwanej stałej
  + Prawda / Tak
  + Fałsz / Nie

19. Dla których definicji klasy poniższa funkcja powoduje błędy kompilacji?

int main() {

Object().variable; }

* struct Object {int variable; };
  + Prawda / Tak
  + Fałsz / Nie
* struct Object {public: int variable; };
  + Prawda / Tak
  + Fałsz / Nie
* class Object {int variable; };
  + Prawda / Tak
  + Fałsz / Nie

20. Które z poniższych wycinków kodu powoduje błędy kompilacji?

int table[5];

* const int \*pointer = table; ++\*pointer;
  + Prawda powoduje - Tak “increment of read-only location”
  + Fałsz / Nie
* int \*const pointer = table; ++pointer;
  + Prawda / Tak - powoduje “increment of read-only variable”
  + Fałsz / Nie
* int const \*pointer = table; ++pointer;
  + Prawda / Tak
  + Fałsz / Nie - nie powoduje błędu

21.

* Każdy wskaźnik stały jest modyfikujący
  + Prawda / Tak
  + Fałsz / Nie
* Wskaźnik jest modyfikujący jedynie jeśli jest stały
  + Prawda / Tak
  + Fałsz / Nie

Int \*ptr1 = &tbl; // nie stały, modyfikujący

* Każdy wskaźnik modyfikujący jest stały.
  + Prawda / Tak
  + Fałsz / Nie

Int \*ptr1 = &tbl; // nie stały, modyfikujący

Int \* const ptr2 = &tbl; // stały, modyfikujący

22. Które z poniższych wycinków kodu powodują błędy kompilacji?

struct Object {

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

* Object()[0];
  + Prawda / Tak
  + Fałsz / Nie
* const Object object; object[0];
  + Prawda / Tak
  + Fałsz / Nie
* Object object; object[0];
  + Prawda / Tak
  + Fałsz / Nie

23. 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(); }

* 12;
  + Prawda / Tak
  + Fałsz / Nie
* 11;
  + Prawda / Tak
  + Fałsz / Nie
* Co innego;
  + Prawda / Tak
  + Fałsz / Nie

24.Które z poniższych linii powodują błędy kompilacji?

struct Object {

static int method();

static int variable1;

int variable2; };

* int Object::method() {return variable2; }
  + Prawda / Tak
  + Fałsz / Nie
* int Object::method() {return variable1; }
  + Prawda / Tak
  + Fałsz / Nie
* int Object::method() {return Object::method(); }
  + Prawda / Tak
  + Fałsz / Nie

25. Które z poniższych wycinków kodu powodują błędy kompilacji?

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

return string += character; }

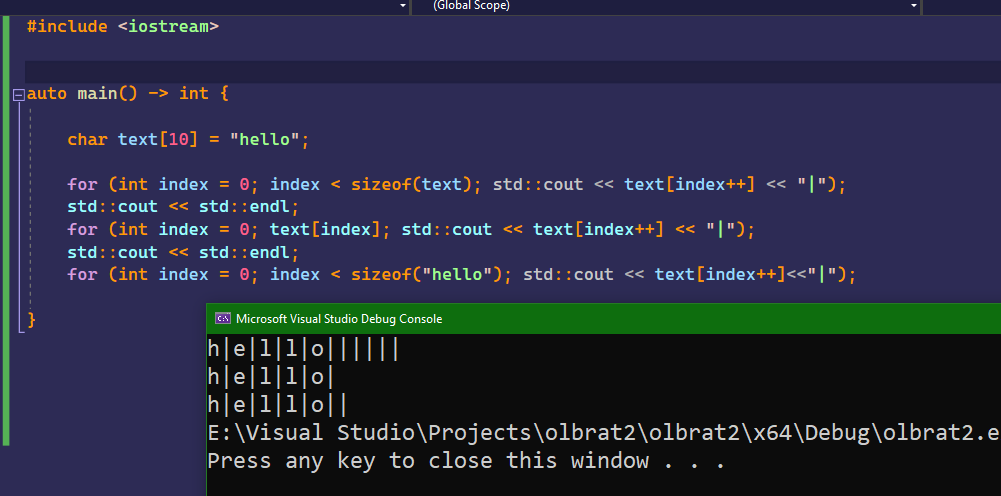
* std::string string; std::cout << string;
  + Prawda / Tak
  + Fałsz / Nie
* std::string string; string << 'a';
  + Prawda / Tak
  + Fałsz / Nie
* std::string string; string << 'a' << 'b';
  + Prawda / Tak
  + Fałsz / Nie

26. Która z poniższych pętli poprawnie wypisuje napis zawarty w tablicy text?

char text[10] = "hello";

* for (int index = 0; index < sizeof(text); std::cout << text[index++]);
  + Prawda / Tak
  + Fałsz / Nie
* for (int index = 0; text[index]; std::cout << text[index++]);
  + Prawda / Tak
  + Fałsz / Nie
* for (int index = 0; index < sizeof("hello"); std::cout << text[index++]);
  + Prawda / Tak
  + Fałsz / Nie

Spacje, dyskusyjne zadanie



26. Które z poniższych linii powodują błędy kompilacji?

struct Object {

Object(int a) {}};

* Object object();
  + Prawda / Tak
  + Fałsz / Nie
* Object object(0);
  + Prawda / Tak
  + Fałsz / Nie
* Object object;
  + Prawda / Tak
  + Fałsz / Nie

26. Co wypisuje poniższy program?

int counter = 0;

struct Object {

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

int main() {

{ Object array[10];

array[0].~Object(); }

std::cout << counter; }

* 1
  + Prawda / Tak
  + Fałsz / Nie
* 10
  + Prawda / Tak
  + Fałsz / Nie
* 11
  + Prawda / Tak
  + Fałsz / Nie

26. Które z poniższych linii powodują błędy kompilacji?

* void \*pointer = std::malloc(10 \* sizeof(int));
  + Prawda / Tak
  + Fałsz / Nie
* int \*pointer = std::malloc(10 \* sizeof(int));
  + Prawda / Tak
  + Fałsz / Nie
* int \*pointer = (int\*)std::malloc(10);
  + Prawda / Tak
  + Fałsz / Nie

26. Dla których deklaracji tablicy table poniższa pętla wypisuje 7 3 8 0 0?

for (int index = 0; index < 5; ++index) {

std::cout << table[index] << " "; }

* int table[10] = {7, 3, 8};
  + Prawda / Tak
  + Fałsz / Nie
* int table[5] = {7, 3, 8, 0};
  + Prawda / Tak
  + Fałsz / Nie
* int table[] = {7, 3, 8, 0, 0};
  + Prawda / Tak
  + Fałsz / Nie

#### 

27. Które z poniższych funkcji powodują błędy kompilacji?

struct Object { Object() = default; Object(const Object &) = delete; };

* Object function() {return Object(); };
  + Prawda / Tak
  + Fałsz / Nie
* Object &function() {return Object(); }
  + Prawda / Tak
  + Fałsz / Nie
* Object &function(const Object &object) {return object; };
  + Prawda / Tak
  + Fałsz / Nietable[]

28. Które z poniższych warunków są spełnione?  
int table[] = {7, 4, 3, 5, 4};

int \*pointer = table + 2;

* \*(table + 1) == \*pointer + 1
  + Prawda / Tak
  + Fałsz / Nie
* table + 1 == pointer - 1
  + Prawda / Tak
  + Fałsz / Nie
* table[1] == pointer[2]
  + Prawda / Tak
  + Fałsz / Nie

29. Które z poniższych wyrażeń powodują błędy kompilacji?

int\* a, b;

* \*(a + b)
  + Prawda / Tak
  + Fałsz / Nie
* \*a + \*b
  + Prawda / Tak
  + Fałsz / Nie
* a - &b
  + Prawda / Tak
  + Fałsz / Nie

30. Które z poniższych linii powodują błędy kompilacji?

char variable;

* int \*pointer = &variable;
  + Prawda / Tak
  + Fałsz / Nie
* double \*pointer = (double\*)&variable;
  + Prawda / Tak
  + Fałsz / Nie
* char \*pointer = (int\*)&variable;
  + Prawda / Tak
  + Fałsz / Nie

31.

* Suma wskaźnika i liczby całkowitej jest wskaźnikiem.
  + Prawda / Tak
  + Fałsz / Nie
* Różnica dwóch wskaźników jest liczbą całkowitą.
  + Prawda / Tak
  + Fałsz / Nie
* Różnica dwóch liczb całkowitych jest wskaźnikiem
  + Prawda / Tak
  + Fałsz / Nie

32.

* Inkrementacja wskaźnika zawsze zwiększa zawarty w nim adres o jeden bajt.
  + Prawda / Tak
  + Fałsz / Nie
* Inkrementacja wskaźnika zwiększa wskazywaną zmienną o jeden.
  + Prawda / Tak
  + Fałsz / Nie
* Inkrementacja zmiennej wskaźnikowej zwiększa jej adres.
  + Prawda / Tak
  + Fałsz / Nie

<https://docs.google.com/document/d/1ReYlc48Cq2P3P0mN98EJJAuoRfe08oL9_lTQB6lJ7EM/edit>