**Egzamin**

**Podstawy programowania**

**1. Zaznaczanie odpowiedzi:**

zaznaczamy markerem nie kolorem tekstu

Na zielono zaznaczamy odpowiedzi **poprawne**

Na żółto zaznaczamy odpowiedzi **prawdopodobne**

Na czerwono zaznaczamy odpowiedzi **złe**

**2. Wklejanie pytań:**

- Wystarczy zaznaczyć całe pytanie na insperze i przekopiowaćdouble

- Jeśli pytanie się rozjeżdza i nie mamy jak tego naprawić wklejamy screena

- **Jeśli coś popsujemy np usuniemy wszystkie pytania w dokumencie używamy ctrl+z**

**Na zielono podkreślone są prawidłowe odpowiedzi.**

**Na żółto odpowiedzi prawdopodobne.**

[**https://drive.google.com/drive/u/0/folders/1kwJFbiZJECRBFD4z6cCsQhy1wWUmyqKk**](https://drive.google.com/drive/u/0/folders/1kwJFbiZJECRBFD4z6cCsQhy1wWUmyqKk)

**link do drive google z odpowiedziami**

**1.**

TAK/NIE *Liczby ujemne konwertują się do wartości logicznej fałsz*.

TAK/NIE *Konwersja typu char do typu bool musi być jawna*

TAK/NIE *Istnieje niejawna konwersja typu int do typu bool*

**2**.

TAK/NIE *Pętle while zawsze można zastąpić pętlą do while*

TAK/NIE *Pętle do while zawsze można zastąpić pętlą while*

TAK/NIE *Pętle for zawsze można zastąpić pętlą while*

**3.**

TAK/NIE *Stała Liczbowa 7.5F jest typu double (double to liczby zmiennoprzecinkowe)*

TAK/NIE *Stała liczbowa 7 jest typu int ( Int - liczby całkowite integralne)*

TAK/NIE *Stała liczbowa 7. jest typu float (float to również liczby zmiennoprzecinkowe)*

**4.**

TAK/NIE *Istnieje niejawna konwersja referencji nie modyfikującej do modyfikującej.*

TAK/NIE i*stnieje niejawna konwersja referencji jednego typu do referencji innego typu.*

TAK/NIE *Istnieje niejawna konwersja referencji modyfikującej do niemodyfikujące.*

**5.**

TAK/NIE *Zawartość pliku można odczytać za pomocą funkcji, która przyjmuje skojarzony z tym plikiem strumień przez wartość.*

*[Nie można przekazać obiektu pliku przez wartość bo wymagałoby to utworzenia kopii obiektu którego nie można kopiować]*

TAK/NIE *Funkcja przyjmująca strumień typu std::ifstream przez wartość nie kompiluje się.*

TAK/NIE *Odczyt z pliku nie zmienia jego zawartości. Zawartość pliku można więc odczytać za pomocą funkcji przyjmującej skojarzony z tym plikiem strumień przez referencję nie modyfikującą.*

*[Za pośrednictwem stałej referencji nie da się pisać do pliku ani z niego czytać]*

**6.**

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

TAK/NIE *Iteratora stałego nigdy nie można przesunąć na następny element*

TAK/NIE *Iteratora nie modyfikującego nigdy nie można przesunąć na następny elemen*t.

**7.**

TAK/NIE *Każde wyrażenie lambda ma inny typ.*

TAK/NIE, *Zamiast używać słowa kluczowego auto, w definicji nazwanego wyrażenia lambda można samemu podać jego typ.*

*[std::function< bool(int, int) > myFunc = []( int x, int y ){ return x > y; };]*

TAK/NIE *Typ wyrażenia lambda jest opisany szablonem std::function.*

**8**.

TAK/NIE *Jeżeli pomiędzy pierwszym a drugim średnikiem w okrągłym nawiasie pętli for nie umieszczono żadnego wrażenia, to nastąpi błąd kompilacji.*

TAK/NIE *Warunek pętli for jest sprawdzany bezpośrednio po wykonaniu ciała pętli.*

TAK/NIE *Jeżeli warunek pętli for nie jest spełniony, to zamiast wykonania ciała pętli obliczane jest wyrażenie za drugim średnikiem w okrągłym nawiasie pętli, po czym pętla się kończy.*

**9**.

TAK/NIE Przez wartość można przekazać do funkcji zarówno zmienną, stałą, jak i obiekt tymczasowy bez nazwy.

TAK/NIE Jeśli funkcja przyjmuje argument przez wartość, to podczas wywołania funkcji tworzona jest kopia argumentu.

TAK/NIE Jeśli funkcja zwraca wynik przez wartość, to wynikiem wywołania tej funkcji jest obiekt tymczasowy bez nazwy.

**11**.

TAK/NIE Za pośrednictwem iteratora stałego można w pewnych przypadkach zmodyfikować element

TAK/NIE Iteratora stałego nigdy nie można przesunąć na następny element

TAK/NIE Iteratora nie modyfikującego nigdy nie można przesunąć na następny element

**13.**

TAK/NIE Argumenty wywołania programu są przekazywane do funkcji main jako napisy w stylu języka C

TAK/NIE Istnieje niejawna konwersja elementu tablicy argv do liczby całkowitej

TAK/NIE Aby odczytać liczbę całkowitą podaną jako argument wywołania programu należy użyć operatora konwersji do liczby całkowitej.

**15.**

TAK/NIE Wynikiem operatora \*= jest obiekt tymczasowy bez nazwy.

TAK/NIE Wynikiem operatora \*= jest referencja nie modyfikująca.

TAK/NIE Wynikiem operatora \*= jest referencja modyfikująca .

*std::cout << (\*(& (i\*=2)) = 15) << " " << i;*

[Wynik operatora modyfikującego to wartość zmodyfikowanej zmiennej]

**16.**

TAK/NIE Operatorem >> można odczytać znak biały.

TAK/NIE Operatorem >> opuszcza wszystkie znaki białe poprzedzające odczytywaną wartość

TAK/NIE Wciśnięcie *ENTER* zawsze kończy odczyt danych operatorem >>

**17.**

TAK/NIE Funkcja przyjmująca, argumentem przez referencję nie modyfikującą, działa na kopii argumentu przekazanego w wywołaniu

TAK/NIE Funkcja przyjmująca argument przez wartość działa na kopii argumentu przekazanego w wywołaniu

TAK/NIE Aby funkcja mogła zmodyfikować zmienną przekazaną jako argument wywołania, musi przyjmować ten argument przez referencje

**18.**

TAK/NIE Łańcuch typu std::string może przechowywać tekst w kodowaniu UTF-8.

TAK/NIE Ostatnim elementem łańcucha typu std::string jest znak zerowy

TAK/NIE Elementy łańcucha typu std::string są typu char

**19.**

TAK/NIE Liczba 3.2e05 ma pięć cyfr znaczących. // 320000.0 -> 320000

TAK/NIE Liczba 3.210 ma cztery cyfry znaczące.

TAK/NIE Liczba 3.21 ma trzy cyfry znaczące.

https://encyklopedia.pwn.pl/haslo/cyfry-znaczace;3888601.html

**20.**

Które z poniższych linii powodują błędy kompilacji?

| TAK/NIE [](int e) {e++; }(0);  TAK/NIE [](int &e) {e++; }(0);  TAK/NIE [&e] {e++; }(0); |
| --- |

**21.**

Które z poniższych linii drukują nierozdzielone cyfry?

TAK/NIE std::cout << std::setw(2) << std::right << 1 << 2 << std::endl;

TAK/NIE std::cout << std::right << 1 << std::setw(2) << 2 << std::endl;

TAK/NIE std::cout << std::setw(2) << 1 << std::right << 2 << std::endl;

**22.**

Dla których definicji wektora poniższy wycinek kodu wpisuje liczbę 5?

int index = 0;

int result = std::count\_if(vector.begin(), vector.end(),

[&](int element) {return index++ == element; });

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

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

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

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

**23.**

Które z poniższych funkcji powodują błędy kompilacji?

TAK/NIE int function(int &r) {return r++; }

TAK/NIE int &function(int &r) {returreturnn ++r; }

TAK/NIE int &function(int &r) {return r++; }

**24.**

Które z poniższych pętli poprawnie wypisują wszystkie elementy wektora?

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

**25.**

Które z poniższych wycinków kodu dają wydruk 123?

TAK/NIE int i = 1; do {std::cout << i; ++i; } while(i <= 3);

TAK/NIE for (int i = 1; i <= 3; ++i) {std::cout << i; }

TAK/NIE int i = 1; while (i <= 3) {std::cout << i; ++i;}

**26.**

TAK/NIE Rozmiar łacucha typu *std::string* to liczba znaków z wyłączeniem znaków białych

TAK/NIE Pusty łańcuch typu *std::string* to taki, który zawiera tylko znaki białe.

TAK/NIE W odniesieniu do łańcucha typu *std::string* metoda ***size*** może dać inny wynik niż operator *sizeof.*

**28.**

TAK/NIE Aby wyrażenie lambda mogło korzystać ze zmiennej lokalnej zdefiniowanej w swoim ciele, to musi ją przechwycić.

TAK/NIE Wyrażenie lambda może przechwycić inne wyrażenie lambda.

TAK/NIE Jeżeli wyrażenie lambda przechwytuje zmienną lokalną przez wartość, to działa na jej kopii i może tę kopię modyfikować.

**29. [approved magda]**

TAK/NIE Obliczane są zawsze oba operandy alternatywy i koniunkcji logicznej

TAK/NIE Jeżeli pierwszy operand alternatywy logicznej jest prawdą, to drugi nie jest obliczany.

TAK/NIE Jeżeli pierwszy operand koniunkcji logicznej jest prawdą, to drugi nie jest obliczany.

**31.**

TAK/NIE Każda funkcja musi kończyć się instrukcją *return.*

TAK/NIE Instrukcja *return* może znajdować się w pętli.

TAK/NIE W funkcji może znajdować się najwyżej jedna instrukcja *return*

**33.**

Które z poniższych definicji tworzą wektor o elementach 2 i 7?

TAK/NIE std::vector<int> vector = {2, 7};

TAK/NIE std::vector<int> vector {2, 7};

TAK/NIE std::vector<int> vector(2, 7);

**34.**

Które z poniższych wycinków kodu wypisują liczbę 2?

int i = 0;

TAK/NIE i++ || ++i; std::cout << i;

TAK/NIE (i = 1) && (i = 2); std::cout << i;

TAK/NIE (i = 1) || (i = 2); std::cout << i;

**35.**

Które z poniższych pętli poprawnie zliczają zawarte w pliku znaki nie białe?

std::ifstream file("file.txt");

int count = 0;

TAK/NIE for (char character; file >> character; ++count);

TAK/NIE for (char character; file.good(); ++count) {file >> character; }

TAK/NIE for (char character; file.get(character); ++count);

**36.**

**++i**

**int i = 5;**

**int a = ++i;**

TAK/NIE Wynikiem operatora preinkrementacji jest referencja nie modyfikująca.

TAK/NIE Wynikiem operatora preinkrementacji jest obiekt tymczasowy bez nazwy.

TAK/NIE Wynikiem operatora preinkrementacji jest referencja modyfikująca.

***yes I do, cursed shit too***  
 int i = 5;  
 ++i;  
 int a = ++i;  
 std::cout << (\*(& ++i) = 15) << " " << i;

**37.**

TAK/NIE Jeżeli pierwszy operand koniunkcji logicznej jest prawda to drugi jest obliczany

TAK/NIE Obliczane są zawsze oba operandy alternatywy i koniunkcji logicznej.

TAK/NIE Jeżeli pierwsze operand alternatywy logicznej jest prawdą to drugi jest obliczany.

**38.**

TAK/NIE Od iteratora modyfikującego można odjąć niemodyfikujący.

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 Jeśli iterator modyfikujący i niemodyfikujący wskazują na ten sam element wektora, to ich porównanie operatorem == daje w wyniku prawdę.

**39.**

TAK/NIE Wszystkie funkcje o tej samej nazwie muszą zwracać wynik tego samego typu.

TAK/NIE W programie mogą istnieć funkcje o tej samej nazwie i argumentach, ale różniące się typem wyniku.

TAK/NIE Funkcje przeciążone to funkcje o tej samej nazwie, ale różniące się liczbą lub typem argumentów.

**40.**

TAK/NIE Wyrażenie std::rand() / RAND\_MAX zwraca liczby całkowite z przedziału od zera do jednego włącznie.

TAK/NIE Wyrażenie std::rand() % RAND\_MAX zwraca liczby całkowite z przedziału od zera do RAND\_MAX włącznie.

TAK/NIE Wyrażenie std::rand() zwraca liczby całkowite z przedziału od zera do RAND\_MAX włącznie

**41.**

Dla których definicji wektora poniższa linia drukuje cyfrę 1?

std::cout << vector[1][1];

TAK/NIE std::vector<std::vector<char>> vector(3, std::vector<char> {2, 1});

TAK/NIE std::vector<std::vector<int>> vector(3, std::vector<int> {2, 1});

TAK/NIE std::vector<std::string> vector(3, std::string {2, 1});

**42.**

TAK/NIE Wynikiem operatora postinkrementacji jest referencja niemodyfikująca.

TAK/NIE Wynikiem operatora postinkrementacji jest referencja modyfikująca.

TAK/NIE Wynikiem operatora postinkrementacji jest obiekt tymczasowy bez nazwy.

**43.**

TAK/NIE Operator indeksowania wektora zawsze zwraca referencje elementu

TAK/NIE Operator indeksowania wektora zawsze zwraca obiekt tymczasowy bez nazwy.

TAK/NIE W przypadku wektora stałego operator indeksowania zwraca referencje niemodyfikującą.

**44.**

TAK/NIE Wyłuskanie iteratora końcowego wektora zawsze jest błędem.

TAK/NIE Wyłuskanie iteratora początkowego wektora nigdy nie jest błędem.

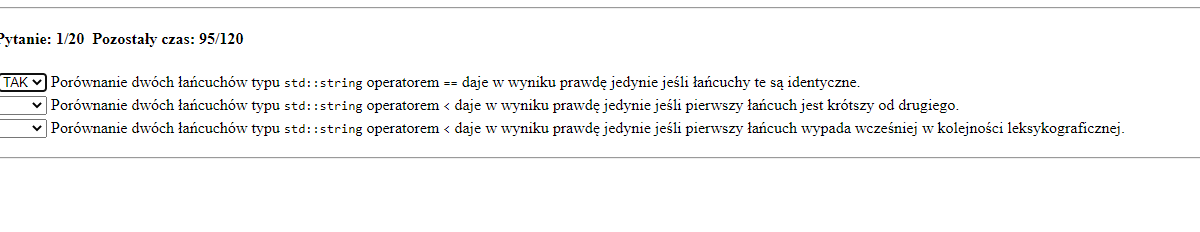
TAK/NIE Jeżeli iterator początkowy wycinka jest równy jego iteratorowi końcowemu, to wyłuskanie tych iteratorów zawsze jest błędem.

**45.**

TAK/NIE W kodzie ASCII istnieje specjalny znak końca pliku i każdy plik tekstowy kończy się tym znakiem.

TAK/NIE W terminalu systemu Linux koniec pliku symuluje się wciskając Ctrl-C.

TAK/NIE W konsoli systemu Windows koniec pliku symuluje się wciskając Ctrl-Z, a potem Enter.



**Do zapytania:**iterator stały vs. iterator niemodyfikujący

**Iterator stały = const**

**Niemodyfikujący = const it; cbegin()**

W przypadku wektora stałego operatora indeksowania zwraca referencję nie modyfikującą

Const []

Const vector<int> v;

Const int = vector[i]

Wyrażenie lambda może przechwycić inne wyrażenie lambda

Lambda przechwytuje lambdy

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

Tak - mutable

Argumenty wywołania programu są przekazywane do funkcji main jako napisy w stylu języka C

Mozna deklarowac swoje zmienne lokalne

Aby wyrażenie lambda mogło korzystać ze zmiennej lokalnej zdefiniowanej w swoim ciele, to musi ją przechwycić - kiedy może zdefiniować w ciele zmienną?

Można obiekt tymczasowy bez nazwy - wewnątrz funkcji przekazać

Przez wartość można przekazać do funkcji zarówno zmienną, stałą, jak i obiekt tymczasowy bez nazwy

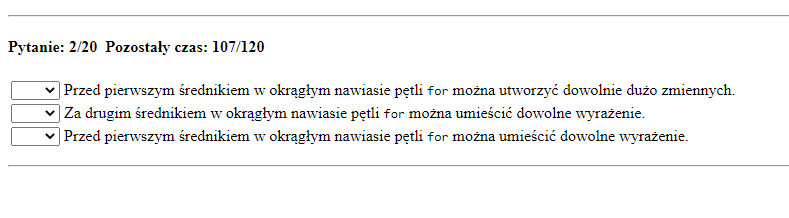
**Które z poniższych wycinków kodu wypisują przy pierwszym wykonaniu liczbę 1?**

**int &function() {**

**static int variable = 0;**

**return variable; }**

| **TAK NIE** | **++function(); std::cout << function();** |
| --- | --- |
| **TAK NIE** | **std::cout << (++function())++;** |
| **TAK NIE** | **function()++; std::cout << function();** |

**li**

| **TAK NIE** | **Jeżeli uruchamiając program w linii poleceń podano tylko nazwę egzekutabli, to przekazany do funkcji main argument argc ma wartość jeden.** |
| --- | --- |
| **TAK NIE** | **Jeżeli uruchamiając program w linii poleceń za nazwą egzakutabli napisano ala ma kota, to przekazany do funkcji main argument argc ma wartość trzy.** |
| **TAK NIE** | **Jeżeli uruchamiając program w linii poleceń za nazwą egzakutabli napisano "ala ma kota", to przekazany do funkcji main argument argc ma wartość dwa.** **Pytanie: 2/20 Pozostały czas: 59/120**  | **TAK NIE** | **Jeżeli argument funkcji jest zadeklarowany przy pomocy szablonu std::function, to można jako ten argument przekazać zwykłą funkcję lub wyrażenie lambda.** | | --- | --- | | **TAK NIE** | **Argumentami funkcji wyższego rzędu mogą być tylko takie funkcje lub wyrażenia lambda, które zwracają wynik.** | | **TAK NIE** | **Funkcja przyjmująca jako argument inną funkcję lub wyrażenie lambda to przykład funkcji wyższego rzędu.** | |

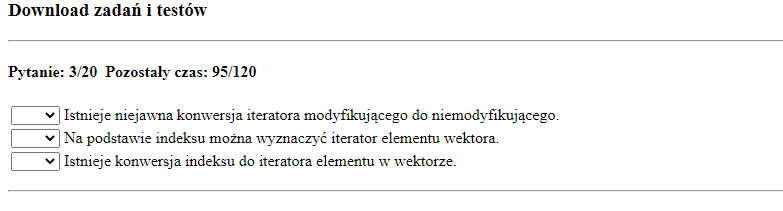
#### **Pytanie: 3/20 Pozostały czas: 115/120**

|  |
| --- |

| **Domyślnie wyrażenie lambda może korzystać ze zmiennych lokalnych zdefiniowanych w tym samym zakresie, co wyrażenie.** |
| --- |
|  |

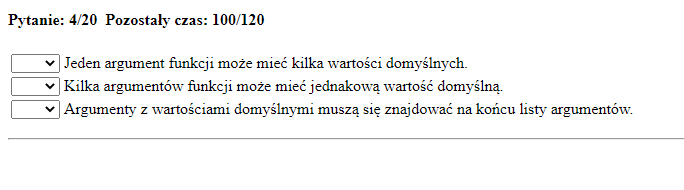
| **Wyrażenie lambda może przechwycić zmienną lokalną przez wartość lub referencję.** |
| --- |
|  |

**Wyrażenie lambda może swobodnie korzystać ze zmiennych globalnych.**

**  
  
Jeżeli umieszczona w ciele pętli definicja zmiennej jest połączona z inicjalizacją, to inicjalizacja ta wykonuje się tylko w pierwszej iteracji. TAK/NIE**

**Jeżeli umieszczona w ciele pętli definicja zmiennej jest połączona z inicjalizacją, to inicjalizacja ta wykonuje się w każdej iteracji. TAK/NIE**

**Jeżeli umieszczona w ciele pętli definicja zmiennej typu wbudowanego nie jest połączona z inicjalizacją, to zmienna ta ma przypadkową wartość początkową. TAK/NIE**

****

#### 

#### 

#### 

#### 

#### 

#### **Pytanie: 5/20 Pozostały czas: 50/120**

**Które z poniższych wywołań funkcji powodują błędy kompilacji?**

**void function(int a, int b = 0, int c = 0) {}**

**void function(int a, int b) {}**

| **TAK NIE** | **function(1, 2);** |
| --- | --- |
| **TAK NIE** | **function(1, 2, 3);** |
| **TAK NIE** | **function(1);** |

#### **Pytanie: 6/20 Pozostały czas: 36/120**

**Które z poniższych linii drukują napis alamakota?**

**std::string s1 = "ala", s2 = "ma", s3 = "kota";**

|  |
| --- |

| **TAK std::cout << (s1 + s2 += s3) << std::endl;** |
| --- |
|  |

| **TAK std::cout << (s1 += s2 + s3) << std::endl;** |
| --- |
|  |

**NIE std::cout << (s1 += s2 += s3) << std::endl;**

#### **Pytanie: 6/20 Pozostały czas: 40/120**

**Które z poniższych pętli dają wydruk 2?**

| **TAK NIE** | **for (int i = 0; ++i < 2; ++i) {std::cout << i++; }** |
| --- | --- |
| **TAK NIE** | **for (int i = 0; ++i < 2; i++) {std::cout << ++i; }** |
| **TAK NIE** | **for (int i = 0; i++ < 2; ++i) {std::cout << ++i; }** **Pytanie: 7/20 Pozostały czas: 84/120**  | **TAK NIE** | **Obiekt tymczasowy bez nazwy jest niedmodyfikowalny.** | | --- | --- | | **TAK NIE** | **Można utworzyć modyfikującą referencję obiektu tymczasowego bez nazwy.** | | **TAK NIE** | **Utworzenie niemodyfikującej referencji obiektu tymczasowego w pewnych przypadkach przedłuża istnienie tego obiektu.** |  **Pytanie: 8/20 Pozostały czas: 108/120**  | **TAK NIE** | **Typ każdej zmiennej musi być znany podczas kompilacji.** | | --- | --- | | **TAK NIE** | **Typ każdej zmiennej musi być podany przez programistę.** | | **TK** | **Typ zmiennej może się zmieniać podczas wykonania programu.** |  |  | | --- |  | **Porównanie dwóch łańcuchów typu std::string operatorem == daje w wyniku prawdę jedynie jeśli łańcuchy te są identyczne.** | | --- | | **tak** |  | **Porównanie dwóch łańcuchów typu std::string operatorem < daje w wyniku prawdę jedynie jeśli pierwszy łańcuch wypada wcześniej w kolejności leksykograficznej.** | | --- | | **?** |   **Porównanie dwóch łańcuchów typu std::string operatorem < daje w wyniku prawdę jedynie jeśli pierwszy łańcuch jest krótszy od drugiego.** |
|  | **?**  Trzeci argument funkcji std::getline określa, czy znak końca linii ma zostać dołączony do wczytanego łańcucha.  Dwuargumentowa funkcja std::getline wczytuje znak końca linii, ale nie dołącza go do wczytanego łańcucha.  Dwuargumentowa funkcja std::getline wczytuje znak końca linii i dołącza go do wczytanego łańcucha.  Argumentem operatora sizeof może być stała dosłowna.  Argumentem operatora sizeof może być nazwa typu.  Argumentem operatora sizeof może być nazwa zmiennej.  Mianipulator std::setw wpływa tylko na najbliższy wydruk.  W pewnych przypadkach manipulator std::setprecision ustawia dokładną liczbę cyfr po przecinku.  W pewnych przypadkach manipulator std::setprecision ustawia minimalną liczbę cyfr znaczących. **Pytanie: 15/20 Pozostały czas: 40/120** Co wypisuje poniższy wycinek kodu?  std::vector<int> vector {1, 2};  for (int i = 1; i < vector.size(); i \*= 2) {  vector.push\_back(vector[i]); }  for (int i = 1; i < vector.size(); i \*= 2) {  std::cout << vector[i]; }   |  | | --- |  | NIE cztery cyfry | | --- | |  |  | TAK dwie cyfry | | --- | |  |   NIE trzy cyfry |
|  | **Słownik:**  **Wyłuskanie iteratora końcowego - \*x.end()  dereferencja - dostanie się do wartości**  **przeciążenie = overloading**  **Operator indeksowania = []**  **referencja niemodyfikująca = const (typ danych) &**  **referencja modyfikująca = (typ danych) &**  **postinkrementacja = i++**  **preinkrementacjia = ++i**  **operand alternatywy = a || b ...**  **koniukcji = a && b**  **pierwszy operand alternatywy = jeśli (a || b) to a**  **iterator stały vs niemodyfikujący || to to samo (?)**    **W przypadku wektora stałego operatora indeksowania zwraca referencję nie modyfikującą ???**  **??**  **obiekt tymczasowy bez nazwy**  **napisy w stylu języka C**  **Aby wyrażenie lambda mogło korzystać ze zmiennej lokalnej zdefiniowanej w swoim ciele, to musi ją przechwycić - kiedy może zdefiniować w ciele zmienną?**  **czy lambda może przechwytywać lambdy?**  **Za pośrednictwem iteratora stałego można w pewnych przypadkach zmodyfikować element.**  **? swap** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |