Imię i nazwisko: MAREK GORZECKI

Prowadzący ćwiczenia: M. RAWSKI

WYNIK WYŚWIETLONY TO 2 ALE NIEMA TAKIEJ ODP. !

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
																	2																					

AHJ BSF CZM DYU ESB FCE GYZ HGV IXI JNB

1. Jeśli przeciążamy operator jednoargumentowy poprzez metodę klasy, to metoda ta powinna być:
a) dwuparametrowa b) jednoparametrowa c) bezparametrowa d) trzyparametrowa

2. Jeśli ob jest obiektem zdefiniowanej przez użytkownika klasy Klasa, to aby w normalny sposób zadziałało
`cout << ob << endl;`
należy zdefiniować
a) *metodę* klasy `Klasa ostream& operator<<(ostream&);`
b) *funkcję globalną* `ostream& operator<<(ostream&, const Klasa&);`
c) *funkcję globalną* `ostream& operator<<(ostream&);`
d) *metodę* klasy `Klasa ostream& operator<<(ostream&, const Klasa&);`
e) *metodę* klasy `Klasa ostream& operator<<(const Klasa&);`

3. Jaką literę wydrukuje następujący fragment programu
```
const char* s = "UVWXYZ";
cout << *(&s[3]-2) << endl;
```
a) X b) Y c) V d) W e) U

4. Jeśli użyliśmy dyrektywy '`#include<iostream>`', ale *nie* użyliśmy '`using namespace std;`', to do nazwy cout trzeba się odnosić poprzez:
a) `std.cout` b) `std->cout` c) `std(cout)` d) `std::cout`

5. Funkcja rekur zdefiniowana jest następująco
```
void rekur(const char* nap) {
    if ( *nap != 'D' ) {
        rekur(nap+1);
        cout << *nap;
    }
}
```
ABCD

A B C

!= 'D'

rekur(nap+1)

*nap

Co zostanie wydrukowane po wywołaniu `rekur("ABCD")` ?
a) ABC b) CBA c) D d) ABCD e) DCBA f) ABCDCBA

6. Jeśli konstruktor zawiera listę inicjalizacyjną, to składowe inicjowane są w kolejności:
a) nieprzewidywalnej (zależnej od użytego kompilatora) b) w jakiej pojawiają się na liście argumentów wywołania c) takiej w jakiej występują w deklaracji klasy d) takiej w jakiej występują na liście

7. Następujące operatory mogą być przeciążane zarówno za pomocą funkcji globalnej jak i metody klasy:
a) *, (), ! b) ++, !, [] c) +, ==, % d) *, [] e) ==, =, + f) (), %

8. Zmienne n, pn, rn zdefiniowane są instrukcją '`int n=1, *pn=&n, &rn=n;`' a funkcja fun ma definicję
`void fun(int* t[]) { cout << *t[0] << endl; }`
Które z wywołań: (1) `fun(n);` (2) `fun(pn);` (3) `fun(&pn);` (4) `fun(rn);` powiedzie się:
a) wszystkie b) pierwsze c) żadne z nich d) trzecie e) drugie f) czwarte

9. Po następujących dwóchinstrukcjach: '`int * const p, k = 7; p = &k;`'
a) wartość `*p` wynosi 7 b) wartość `&p` wynosi 7 c) instrukcje te są nieprawidłowe i nie skompilują się d) wartość p wynosi 7

10. W klasie zdefiniowano dwie metody o identycznej nazwie. Jest to:
a) Niemożliwe b) Zawsze możliwe, niezależnie od liczby i typów ich parametrów; wybór odpowiedniej metody determinowany jest składnią ich wywołania c) Możliwe, jeśli tylko typ zwracany jest dla nich inny d) Możliwe, jeśli liczba i typ parametrów tych metod różnią się dostatecznie

11. Jeśli zaalokowana została pamięć na tablicę obiektów pewnej klasy/struktury
`Klasa* k = new Klasa[10];`
to należy ją zwolnić poleceniem:
a) `delete *k;` b) `delete k;` c) `delete [] *k;` d) `delete [] &k;` e) `delete &k;` f) `delete [] k;`

12. Poniższy program
```
#include <iostream>
using namespace std;

struct Klasa {
    int d;
    Klasa(int d) : d(d){ cout << "K" << d; }
                       { cout << "D" << d; }
   ~Klasa()
};

Klasa k1(1);
```

```
int main() {
    Klasa k2(2);
}
Klasa k3(3);
```

wydrukuje
a) K1K2K3D3D2D1 b) K1K2K3D1D2D3 c) K1K3K2D2D3D1 d) program jest błędny

13. Jeśli metoda klasy została zadeklarowana jako stała (`const`) to:
a) nie może zmieniać wartości żadnego przekazanego parametru b) może być wywołana tylko na rzecz obiektów ustalonych (`const`) c) nie może zmieniać składowych żadnego obiektu tej klasy d) wszystkie jej parametry muszą być `const` e) nie może zmieniać żadnej składowej obiektu na rzecz którego została wywołana

14. Zmienne `n`, `pn`, `rn` zdefiniowane są instrukcją '`int n = 1, *pn = &n, &rn = *pn;`', a funkcja `fun` ma prototyp '`int fun(int*);`'. Które z wywołań

```
pn = fun(n); rn = fun(*pn); rn = fun(&n); *pn = fun(*pn); n = fun(*n);
```

może się powieść:
a) żadne z nich b) drugie i czwarte c) pierwsze i piąte d) tylko trzecie e) wszystkie

15. Po wykonaniu

```
double tab1[] = {1,4,7,10};
double tab2[] = {2,5,8,11};
double tab3[] = {3,6,9,12};
double* tab[] = {tab1,tab2,tab3};
cout << *(*(tab+2)+1) << endl;
```

na ekranie wydrukowane zostanie
a) 6 b) fragment nie skompiluje się c) 8 d) przypadkowa liczba

16. Jeśli przeciążamy operator dwuargumentowy poprzez metodę klasy, to metoda ta powinna być:
a) jednoparametrowa b) dwuparametrowa c) bezparametrowa d) trzyparametrowa

17. Jeśli `Klasa` jest nazwą klasy z publicznym konstruktorem domyślnym, to po instrukcji '`Klasa k, *p;`' prawidłowe może być przypisanie
a) `k = new Klasa();` b) `p = new Klasa();` c) `*p = new Klasa();` d) `*k = new Klasa();`

18. Jeśli funkcja `f` ma definicję

```
int f(int x, int y=1, int z=2) {return 2*x-y;}
```

to instrukcja `cout << f(0)-f(1,1)-f(2,2,2);` spowoduje wypisanie liczby
a) -5 b) 7 c) -11 d) -7 e) 11 f) 0 g) 5

19. Spośród następujących stwierdzeń:
(A) funkcja zaprzyjaźniona z daną klasą musi być jej metodą; (B) funkcja zaprzyjaźniona z daną klasą nie jest jej metodą; (C) funkcja może być zaprzyjaźniona tylko z jedną klasą; (D) funkcja może być zaprzyjaźniona z wieloma klasami
prawdziwe są tylko
a) B,C b) B,D c) A,D d) A,C

20. Spośród następujących deklaracji konstruktora kopiującego dla klasy `A`

```
(a) A A(const A*);    (b) A A(const A&);    (c) A(const A*);
(d) A(const A&);      (e) A(A&);            (f) A(const A);
```

akceptowalne mogą być tylko:
a) żaden b) (d) i (f) c) (a), (b) i (c) d) (d) i (e) e) (c) i (f) f) (a) i (b)

21. Jeśli klasa `D` dziedziczy z klasy `B`, a w klasie pochodnej `D` istnieje składowa obiektowa typu `M`, to przy tworzeniu obiektu klasy `D` konstruktory poszczególnych klas będą wywoływane w kolejności
a) M B D b) B M D c) D M B d) M D B e) B D M f) D B M

22. Spośród następujących deklaracji destruktora klasy `A`

```
(a) void ~A(A&);      (b) A A(const A&);    (c) ~A();
(d) A(A*);            (e) ~A(A&);           (f) ~A(const A&);
```

akceptowalne mogą być tylko:
a) (b) i (d) b) (e) c) (c) d) (f) e) (a) i (f) f) (d)

23. Jeśli klasa `D` dziedziczy z klasy `B`, a w klasie pochodnej `D` istnieje składowa obiektowa typu `M`, to przy usuwaniu obiektu klasy `D` destruktory poszczególnych klas będą wywoływane w kolejności
a) D B M b) D M B c) M D B d) B D M e) B M D f) M B D

24. W klasie `Klasa` zadeklarowano konstruktor

```
struct Klasa {
    Klasa(int a, char* b="", int c);
    // ...
};
```

Który z następujących sposobów utworzenia obiektu tej klasy jest prawidłowy:

```
Klasa *k = new Klasa(1,"napis", 2); // (A)
Klasa *k = new Klasa(1,2);          // (B)
Klasa *k = new Klasa(1);            // (C)
Klasa *k = new Klasa("napis", 2);   // (D)
```

a) definicja konstruktora jest nieprawidłowa b) tylko (A) i (B) c) tylko (A) i (C) d) tylko (A) i (D) e) wszystkie f) tylko (B) i (C)

25. Jeśli jest zdefiniowana klasa `AClass` z publicznym konstruktorem domyślnym i publiczną bezparametrową metodą `fun()`, to spośród instrukcji

(a) `(new AClass)->fun();`
(b) `(*(new AClass))->fun();`
(c) `(*(new AClass)).fun();`
(d) `(new AClass).fun();`

prawidłowe są tylko:
a) (a) i (c) b) tylko (b) c) (c) i (d) d) (b) i (d) e) (a) i (b) f) wszystkie g) żadna

26. Jeśli funkcja `F` ma deklarację '`void F(int* n);`', a `k` jest typu `int`, to wywołanie tej funkcji może mieć postać:
a) `F(*k);` b) `F(k&);` c) `F(k*);` d) `F(k);` e) `F(&k);`

27. Spośród następujących stwierdzeń
(A) Konstruktory można przeciążać; (B) Typem zwracanym konstruktora musi być typ definiowany przez klasę, w której konstruktor jest zawarty; (C) W każdej klasie musi istnieć konstruktor domyślny; (D) Konstruktory można przeciążać pod warunkiem, że dostarczymy definicji konstruktora domyślnego;
prawdziwe są tylko
a) A b) A,B c) A,C d) C,D e) C f) B,C

28. Jeśli wzorzec klasy zdefiniowano jako

```
template<typename T, int S>
class Klasa {
    // ...
};
```

i w tej klasie istnieje publiczny konstruktor domyślny, to utworzenie obiektu klasy powstałej poprzez konkretyzację tego wzorca mogłoby mieć postać:

```
Klasa a(int,6);              // (A)
Klasa<int,int> a;            // (B)
Klasa<int,6> a;              // (C)
template Klasa<int, 6> a;    // (D)
Klasa<template int, 6> a;    // (E)
```

a) tylko (B) b) tylko (A) i (E) c) tylko (D) d) tylko (C) e) tylko (A) i (C)

29. Jeśli `obA` i `obB` są obiektami zdefiniowanej przez użytkownika klasy `K`, to aby zadziałało przeciążenie '`obA + obB`' należy zdefiniować
a) *funkcję globalną* `K operator+(const K&);`
b) *metodę* `K operator+(const K&, const K&);` lub *funkcję globalną* `K operator+(const K&);`
c) *metodę* `K operator+(const K&);` lub *funkcję globalną* `K operator+(const K&, const K&);`
d) *metodę* `K operator+(const K&, const K&);`

30. Dla klas abstrakcyjnych (zawierających przynajmniej jedną metodę czysto wirtualną), spośród poniższych stwierdzeń:
(A) Nie wszystkie zadeklarowane w klasie abstrakcyjnej metody muszą być w niej zdefiniowane; (B) W klasie abstrakcyjnej żadna z zadeklarowanych metod nie może mieć implementacji (definicji); (C) Nie można tworzyć obiektów klasy abstrakcyjnej; (D) Obiekty klasy abstrakcyjnej można tworzyć, ale są one wtedy również abstrakcyjne; (E) W klasie abstrakcyjnej wszystkie metody muszą być wirtualne;
prawdziwe są tylko:
a) A,C b) A c) C,E d) C,D e) B,E

31. Jeśli funkcja ma jeden parametr z wartością domyślną, to musi to być:
a) parametr pierwszy b) taka sytuacja jest niemożliwa c) parametr ostatni d) parametr jedyny

32. Przeanalizuj następujący program:

```
#include <iostream>
using namespace std;

class X {
public:
    virtual void f1() {cout << "f1X ";}
    void f2()         {cout << "f2X ";}
};

class Y : public X {
public:
    void f1() {cout << "f1Y ";}
    void f2() {cout << "f2Y ";}
};

int main() {
    X x, *px = &x, *pxy = new Y;
    px->f1();  px->f2();  pxy->f1();  pxy->f2();
}
```

Co zostanie wypisane po jego uruchomieniu?

a) f1X f2X f1Y f2Y **b)** f1X f2Y f1X f2Y **c)** f1X f2X f1X f2Y **d)** f1X f2X f1X f2X **e)** f1X f2Y f1Y f2Y **f)** f1X f2X f1Y f2X

33. Zakładając, że klasa `B` została zdefiniowana wcześniej, następująca definicja klasy `A`

```
struct A {
    A  a;   \\ (1)
    A* pa;  \\ (2)
    B  b;   \\ (3)
};
```

mogłaby być prawidłowa po skreśleniu linii

a) może być prawidłowa bez skreśleń **b)** 1 **c)** 2 **d)** 3

34. Jeśli klasa `B` dziedziczy publicznie z klasy `A`, a w `A` nie ma konstruktora domyślnego, to

a) `B` nie może mieć konstruktora domyślnego **b)** taka sytuacja jest niemożliwa (program się nie skompiluje) **c)** każdy konstruktor `B` musi mieć listę inicjalizacyjną **d)** co najmniej jeden konstruktor `B` musi mieć listę inicjalizacyjną **e)** `B` musi mieć konstruktor domyślny

35. Funkcja `fun` zdefiniowana jest następująco:

```
int fun(int a, int* pb, int& c) {
    --a; --*pb; --c;
    return a + *pb + c;
}
```

Co wydrukuje następujący fragment programu:

```
int a = 0, b = 1, c = 2, x = fun(a,&b,c);
cout  << a + b + c + x << endl;
```

a) 4 **b)** 1 **c)** -1 **d)** -2 **e)** 2 **f)** 0

36. Spośród następujących stwierdzeń

(A) Jeśli w klasie zdefiniowano konstruktor, to zawsze trzeba też zdefiniować destruktor; (B) Jeśli w klasie zdefiniowano destruktor, to zawsze trzeba też zdefiniować konstruktor; (C) Każdemu przeciążonemu konstruktorowi musi odpowiadać przeciążony destruktor; (D) Jeśli w klasie zdefiniowano destruktor, to musi on być funkcją bezparametrową (E) Jeśli w klasie zdefiniowano konstruktor, to musi on być funkcją bezparametrową

prawdziwe są tylko

a) A,B **b)** C **c)** A **d)** D,E **e)** D **f)** A,B,C

37. Po instrukcji

```
int tab[] = {1,2,3,4,5}, *p = tab+1;
```

wartość wyrażenia `p[2]` wynosi

a) 2 **b)** 3 **c)** 4 **d)** instrukcja jest nielegalna **e)** wartość ta jest nieokreślona

38. Spośród poniższych stwierdzeń:

(A) Konstruktor może być wirtualny, ale destruktor nie; (B) Konstruktor nie może być wirtualny, ale destruktor może; (C) Ani konstruktor ani destruktor nie może być wirtualny; (D) Zarówno konstruktor jak i destruktor może być wirtualny;

prawdziwe są tylko:

a) A,C **b)** A **c)** B,D **d)** B