Wstęp do programowania obiektowego

Poprawkowe zaliczenie ćwiczeń, Wilno 2023-07-03

Imię: ..... Nazwisko: .....

Zadanie 1. Która z poniższych instrukcji **NIE** wypisze danych w oknie terminala?

(a) `printf("hello world\n");`
(b) `cout >> "hello world" >> endl;`
(c) `cout << "hello world" << endl;`

Zadanie 2. Która instrukcja pobierze wartość z terminala i wstawi do zmiennej całkowitej `x`?

(a) `cin >> x;`
(b) `cin << x;`
(c) `x << cin;`

Zadanie 3. Co to znaczy, że funkcja jest przeciążona?

(a) Funkcja jest przeciążona, gdy zwraca różne typy danych.
(b) Funkcja jest przeciążona, gdy wywołuje samą siebie.
(c) Funkcja jest przeciążona, gdy jej identyfikator użyty jest dwa lub więcej razy w definicjach funkcji różniących się zestawem parametrów, tzn. ilością lub typami argumentów.

Zadanie 4. Co to jest klasa?

(a) Zbiór obiektów określonego typu.
(b) Obiekt określonego typu.
(c) Typ danych posiadający strukturę danych w postaci własności i interfejs złożony z metod.

Zadanie 5. Czym się różni struktura od klasy?

(a) Struktura i klasa są tym samym.
(b) Struktura to szczególny przypadek klasy. W strukturze wszystko domyślnie jest publiczne, w klasie wszystko domyślnie jest prywatne.
(c) Klasa to szczególny przypadek struktury. W strukturze wszystko domyślnie jest prywatne, w klasie wszystko domyślnie jest publiczne.

Zadanie 6. Co to znaczy, że funkcja ma argumenty o wartościach domyślnych?

(a) Argument funkcji ma wartość domyślną, gdy ją zawołamy z mniejszą ilością argumentów i kompilator w miejsce brakujących argumentów wstawi 0.
(b) Funkcja ma argument o wartości domyślnej, gdy w jej nagłówku podamy wartość tego argumentu oddzielając go znakiem `=`. Wołając funkcję możemy pominąć argumenty z wartościami domyślnymi.
(c) W C++ nie można przypisać domyślnej wartości argumentom funkcji.

Zadanie 7. Co to jest konstruktor?

(a) Konstruktor to dowolna metoda, która zwaraca obiekt danej klasy.
(b) Konstruktor to metoda, która nie ma argumentów, albo ma tylko argumenty domyślne.
(c) Konstruktor to metoda składowa klasy wywoływana podczas tworzenia nowego obiektu zaraz po zaalokowaniu pamięci dla tego obiektu. Nazwa konstruktora jest identyczna jak nazwa klasy.

Zadanie 8. Co to jest konstruktor domyślny?

(a) Konstruktor domyślny to pierwszy konstruktor zadeklarowany w klasie.
(b) Konstruktor jest konstruktorem domyślnym tylko wtedy, gdy może być wywołany bez żadnych argumentów. To oznacza, że albo nie ma argumentów wcale, albo wszystkie jego argumenty muszą mieć zadane wartości domyślne.
(c) Konstruktor domyślny to konstruktor tworzony przez kompilator.

Zadanie 9. Ile argumentów może mieć konstruktor domyślny?

(a) Dowolnie wiele.
(b) Zero.
(c) Jeśli ma jakieś argumenty to wszystkie muszą mieć wartości domyślne.

Zadanie 10. Co to jest destruktor?

(a) Destruktor to metoda składowa klasy wywoływana automatycznie podczas niszczenia obiektu, tuż przed zwolnieniem zajmowanej przez niego pamięci. Nazwa destruktora to nazwa klasy poprzedzona znakiem `~`.
(b) Destruktor to dowolna metoda składowa klasy, która zwalnia pamięć zajmowaną przez obiekt.
(c) Destruktor to dowolna metoda składowa klasy, której nazwa zaczyna się od znaku `~`.

Zadanie 11. Ile argumentów może mieć destruktor?

(a) 0
(b) 1
(c) Dowolnie wiele.

Zadanie 12. Co to jest obiekt?

(a) Egzemplarz (instancja) klasy.
(b) Inna nazwa istniejącej klasy.
(c) Adres klasy.

Zadanie 13. Co to jest referencja?

(a) Zmienna wskaźnikowa.
(b) Inna nazwa istniejącego obiektu.
(c) Adres obiektu w pamięci.

Zadanie 14. Co wypisze poniższy program?

```
int k = 1;
int &r = k;
r = 2;
cout << k << endl;
```

(a) 1
(b) 2
(c) Kompilator zgłosi błąd w drugiej linii bo powinno tam być `int *r = &k`.

Zadanie 15. Czym jest zmienna `this`?

(a) To wskaźnik do kopii obiektu utworzonej na stosie podczas przekazywania referencji do tego obiektu.
(b) To wskaźnik do klasy, dostępny w metodach składowych uruchamianych na rzecz tej klasy.
(c) To wskaźnik do obiektu, dostępny w metodach składowych uruchamianych na rzecz tego obiektu.

Zadanie 16. Co wyróżnia metodę od funkcji w programowaniu obiektowym?

(a) Metoda to inna nazwa funkcji.
(b) Metoda to funkcja, która ma dodatkowy, niejawny parametr, którym jest wskaźnik `this` do obiektu na rzecz którego jest wołana.
(c) Metoda to funkcja, której argumentami są obiekty.

Zadanie 17. Który konstruktor zostanie wykonany dla obiektu `sc` w poniższym programie?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(int c);
  SpaceCraft(char c);
};

int main() {
  SpaceCraft sc = 101;
}
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 18. Który konstruktor zostanie wykonany dla obiektu `sc` klasy `SpaceCraft` z programu w zadaniu 17?

```
  SpaceCraft sc;
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 19. Który konstruktor zostanie wykonany dla obiektu `sc` klasy `SpaceCraft` z programu w zadaniu 17?

```
  SpaceCraft sc = 'X';
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 20. Który konstruktor zostanie wykonany dla obiektu `sc1`, a który dla `sc2` w poniższym programie?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(int c);
  explicit SpaceCraft(char c);
};

int main() {
  SpaceCraft sc1 = 'X';
  SpaceCraft sc2 = SpaceCraft('Y');
}
```

(a) Dla `sc1` i dla `sc2` konstruktor `SpaceCraft(char c)`.
(b) Dla `sc1` konstruktor `SpaceCraft(int c)`, a dla `sc2` konstruktor `SpaceCraft(char c)`.
(c) Dla `sc1` konstruktor `SpaceCraft()`, a dla `sc2` konstruktor `SpaceCraft(char c)`.

Zadanie 21. Ile argumentów należy przekazać, gdy przeciążamy operator dwuargumentowy jako składowy wewnątrz klasy?

(a) 0
(b) 1
(c) 2

Zadanie 22. Która deklaracja przeciążenia operatora + jako składowego jest poprawna?

(a) `Fraction operator+(const Fraction &f, const Fraction &g) const;`
(b) `Fraction operator+(const Fraction &f) const;`
(c) `void operator+(const Fraction &f, const Fraction &g);`

Zadanie 23. Która deklaracja przeciążenia operatora + jako globalnego jest poprawna?

(a) `operator+(const Fraction &f, const Fraction &g);`
(b) `Fraction operator+(const Fraction &f);`
(c) `Fraction operator+(const Fraction &f, const Fraction &g);`

Zadanie 24. Czy operator przypisania = można przeciążać poza klasą jako globalny?

(a) Nie. Operator przypisania = generowany jest automatycznie jako składowy w klasie, jeśli sami go nie przeciążymy w tej klasie. Gdybyśmy przeciążyli go jako operator globalny to kompilator miałby do wyboru dwa operatory przypisania = dla danej klasy i powstałaby niejednoznaczność.
(b) Tak. Operator przypisania = tak jak pozostałe operatory przypisania można przeciążać poza klasą.
(c) Nie. Operator przypisania = jest wyjątkowy bo modyfikuje dany obiekt i dlatego nie można przeciążać go poza klasą.

Zadanie 25. Ile argumentów należy przekazać, gdy przeciążamy operator jednoargumentowy poza klasą jako globalny?

(a) 0
(b) 1
(c) 2

Zadanie 26. Na czym polega dziedziczenie w programowaniu obiektowym?

(a) Dziedziczenie polega na tworzeniu bardziej szczegółowych, wyspecjalizowanych klas na podstawie klas bardziej ogólnych, o większym stopniu abstrakcji. Klasy potomne posiadają te same własności i metody co ich rodzice, więc nie trzeba definiować ich całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy.
(b) Dziedziczenie polega na tworzenia bardziej zaawansowanych klas poprzez wykorzystanie kodu dostępnego w postaci bibliotek. Nowe klasy powstałe w ten sposób posiadają więcej funkcjonalności.
(c) Dziedziczenie polega na przekazywaniu danych z klasy potomnej bardziej szczegółowej, wyspecjalizowanej do klasy nadrzędnej bardziej ogólnej, o większym stopniu abstrakcji. Klasy potomne posiadają nowe własności i metody w odróżnieniu do ich rodziców.

Zadanie 27. W jakiej kolejności wołane są destruktory w przypadku dziedziczenia?

(a) Kolejność wołania destruktorów możemy określić w definicji klasy.
(b) Najpierw wołany jest destruktor klasy pochodnej, potem klasy bazowej.
(c) Najpierw wołany jest destruktor klasy bazowej, potem klasy pochodnej.

Zadanie 28. Co klasa potomna dziedziczy po klasie bazowej?

(a) Tylko metody składowe.
(b) Tylko własności składowe.
(c) Wszystkie własności oraz metody poza konstruktorami i destruktorami.

Zadanie 29. Do których składowych własności i metod klasy bazowej mamy dostęp w klasie pochodnej?

(a) Do wszystkich składowych.
(b) Do składowych publicznych (public).
(c) Do składowych publicznych (public) i chronionych (protected).

Zadanie 30. Jaka jest dostępność metody `info()` w klasie Square przy następującej implementacji:

```
class Figure {
  public:
      void info() const;
};

class Square : protected Figure {
  public:
      double area() const;
};
```

(a) Publiczna (public).
(b) Chroniona (protected).
(c) Prywatna (private).

Zadanie 31. Mamy klasę `Polygon` i jej klasę pochodną `Triangle`. Czy możliwe jest poniższe przypisanie?

```
Polygon p;
Triangle t;
p = t;
```

(a) Kompilator zgłosi błąd i kompilacja zostanie przerwana bo typy zmiennych `p` i `t` są różne.
(b) Kompilator zgłosi ostrzeżenie, że typy zmiennych `p` i `t` są różne, ale program się skompiluje.
(c) Tak, program skompiluje się poprawnie.

Zadanie 32. W programie mamy klasę `Point` i jej klasę pochodną `Pixel`. Mamy także funkcję o następującym prototypie:

```
float distance(Point &p1, Point &p2);
```

Czy poniższe wywołanie funkcji `distance()` jest poprawne?

```
Pixel px1(0, 0, 0), px2(1, 1, 255);
cout << "Distance: " << distance(px1, px2) << endl;
```

(a) Tak, ponieważ typ dynamiczny przekazywanych argumentów jest bardziej szczegółowy niż typ statyczny argumentów w prototypie funkcji `distance()`.
(b) Nie, bo typ dynamiczny przekazywanych argumentów jest bardziej ogólny niż typ statyczny argumentów w prototypie funkcji `distance()`.
(c) Nie, bo typy argumentów w prototypie są inne niż typy przekazywanych argumentów podczas wołania funkcji `distance()`.

Zadanie 33. Na co może wskazywać wskaźnik do obiektu klasy X?

(a) Wyłącznie na obiekty klasy X.
(b) Na obiekty klasy X oraz obiekty klas od niej bardziej ogólnych.
(c) Na obiekty klasy X oraz obiekty klas od niej bardziej szczegółowych.

Zadanie 34. Co to jest metoda wirtualna?

(a) Metoda zadeklarowana ze słowem kluczowym `virtual`, zaimplementowana w klasie bazowej, nie można tej metody nadpisać/przesłonić w klasie potomnej.
(b) Metoda wirtualna to każda metoda w klasie polimorficznej.
(c) Metoda zadeklarowana ze słowem kluczowym `virtual`, może nie być zaimplementowana w klasie bazowej, używana do uzyskania polimorfizmu.

Zadanie 35. Jaki mechanizm programowania obiektowego realizują metody wirtualne?

(a) Dziedziczenie.
(b) Polimorfizm.
(c) Hermetyzację.

Zadanie 36. Co wypisze następujący program:

```
class Polygon {
  public:
    float circumference() const { return 0; };
    virtual float area() const { return 1; };
};

class Rectangle: public Polygon {
  public:
    float circumference() const { return 2; };
    float area() const { return 3; }
};

int main () {
  Polygon *p = new Rectangle;
  cout << p->circumference() << " : " << p->area() << endl;
}
```

(a) 0 : 3
(b) 2 : 3
(c) 0 : 1

Zadanie 37. Czy konstruktory w C++ mogą być wirtualne?

(a) Tak.
(b) Nie.
(c) Tylko wtedy gdy cała klasa jest wirtualna.

Zadanie 38. Na czym polega abstrakcja w programowaniu obiektowym?

(a) Abstrakcja polega na możliwie ogólnym podejściu do projektowania klas, wyróżnieniu spośród wielu najbardziej istotnych cech modelowanej rzeczywistości.
(b) Abstrakcja polega na wykonywaniu operacji bez ujawniania sposobu, w jaki ta operacja została zaimplementowana.
(c) Abstrakcja polega na dopisaniu słowa kluczowego abstract przed definicją klasy.

Zadanie 39. Jaki jest typ statyczny i dynamiczny zmiennej `x` w poniższym programie?

```
class Vehicle {
     ...
};
class Truck : public Vehicle {
     ...
};
Vehicle *x = new Truck;
```

Jaki jest typ statyczny i dynamiczny zmiennej `x`?

(a) Typ statyczny: wskaźnik na klasę `Vehicle`. Typ dynamiczny: wskaźnik na klasę `Truck`.
(b) Typ statyczny: wskaźnik na klasę `Truck`. Typ dynamiczny: wskaźnik na klasę `Vehicle`.
(c) Typ statyczny: wskaźnik na klasę `Vehicle`. Typ dynamiczny: wskaźnik na klasę `Vehicle`.

Zadanie 40. Czym wyróżnia się klasa abstrakcyjna?

(a) Zawsze można tworzyć obiekty tej klasy. Metody abstrakcyjne nie będą wówczas dostępne.
(b) Można tworzyć obiekty tej klasy, ale lepiej jest zaimplementować (przeciążając) jej wszystkie metody abstrakcyjne.
(c) Nie można tworzyć obiektów tej klasy. Najpierw należy zaimplementować (przesłaniając) jej wszystkie metody abstrakcyjne tworząc klasę pochodną.

Zadanie 41. Jaka metoda jest czysto wirtualna (pure virtual)?

(a) To metoda poprzedzona specyfikatorem `virtual` i przypisanym 0 w deklaracji, co oznacza, że pomijamy jej implementację.
(b) To metoda poprzedzona specyfikatorem `abstract` i przypisanym 0 w deklaracji, co oznacza, że pomijamy jej implementację.
(c) To metoda w klasie abstrakcyjnej, której deklaracja poprzedzona jest specyfikatorem `abstract`.

Zadanie 42. Jak powstaje klasa abstrakcyjna?

(a) Klasa abstrakcyjna powstaje wówczas, gdy dopiszemy sepcyfikator `virtual` przed `class`.
(b) Klasa abstrakcyjna powstaje wówczas, gdy dopiszemy sepcyfikator `abstract` przed `class`.
(c) Klasa abstrakcyjna powstaje wówczas, gdy choć jedna jej metoda jest czysto wirtualna (pure virtual).

Zadanie 43. Która z poniższych klas jest abstrakcyjna?

(a)
```
class Polygon {
  public:
    Polygon();
    virtual float area() const = 0;
  ...
};
```

(b)
```
class Polygon {
  public:
    Polygon();
    virtual float area() const { return 0; };
  ...
};
```

(c)
```
abstract class Polygon {
  public:
    Polygon();
    virtual float area() const;
  ...
};
```

Zadanie 44. Czy destruktor może być wirtualny?

(a) Destruktor nie może być wirtualny.
(b) Desktruktor może być wirtualny wyłącznie w klasie abstrakcyjnej.
(c) Desktruktor z reguły powinien być wirtualny, aby umożliwić zwolnienie zasobów: dealokację pamięci, zamknięcie otwartych plików itp. gdy wołany jest dla obiektu o typie statycznym bardziej ogólnym.

Zadanie 45. Która instrukcja spowduje zgłoszenie wyjątku?

(a) `terminate "expected non-zero value";`
(b) `throw "expected non-zero value";`
(c) `exception "expected non-zero value";`

Zadanie 46. W jaki sposób możemy obsłużyć (przechwycić, wyłapać) wyjątek?

(a) Deklarując własną funkcję zakończenia programu przez `set_terminate()`
(b) Umieszczając kod w bloku: `try{} catch(type t){}`
(c) Umieszczając kod w bloku: `throw{} catch(type t){}`

Zadanie 47. W jakim celu w `catch(type t)` przekazywany jest parametr?

(a) Jeśli został zgłoszony wyjątek typu/klasy `t` to zostanie wykonany ten właśnie blok instrukcji, a parametr `t` może być w tym bloku użyty.
(b) Parametr `t` zostanie zwrócony podczas zakończenia programu, gdy powstanie wyjątek.
(c) Parametr `t` zostanie przekazany do funkcji `set_terminate()`, gdy powstanie wyjątek.

Zadanie 48. Jaki wyjątek wyłapie `catch(...)`?

(a) Każdy wyjątek.
(b) Każdy wyjątek do tej pory nie wyłapany.
(c) Każdy wyłapany wyjątek.

Zadanie 49. Mamy dwa, jeden za drugim, bloki: `catch(int x){}` oraz `catch(char *s){}`. Który z nich zostanie wykonany, gdy w ich zasięgu zgłosimy `throw "exception"`?

(a) `catch(int x){}`
(b) `catch(char *s){}`
(c) Żaden z nich.

Zadanie 50. Mamy dwie klasy: `A` oraz jej pochodną `B` oraz dwa bloki: `catch(A){}` oraz `catch(B){}` jeden za drugim. Który z tych bloków zostanie wykonany po zgłoszeniu `throw B()`?

(a) `catch(A)`
(b) `catch(B)`
(c) `catch(A)` oraz kolejno `catch(B)`

Wstęp do programowania obiektowego

Egzamin, Wilno 2023-07-03

Imię: ..... Nazwisko: .....

Zadanie 1. Która instrukcja pobierze wartość z terminala i wstawi do zmiennej całkowitej `x`?

(a) `cin >> x;`
(b) `cin << x;`
(c) `x << cin;`

Zadanie 2. Co to znaczy, że funkcja jest przeciążona?

(a) Funkcja jest przeciążona, gdy zwraca różne typy danych.
(b) Funkcja jest przeciążona, gdy wywołuje samą siebie.
(c) Funkcja jest przeciążona, gdy jej identyfikator użyty jest dwa lub więcej razy w definicjach funkcji różniących się zestawem parametrów, tzn. ilością lub typami argumentów.

Zadanie 3. Co to jest wzorzec funkcji?

(a) Wzorce funkcji stosujemy, gdy definiujemy funkcje różniące się tylko typem argumentów, a czynności, które mają wykonać, są identyczne. Na podstawie wzorca funkcji kompilator tworzy odpowiednie warianty funkcji różniące się tylko typami parametrów, zwracanych wartości i ewentualnie zmiennych lokalnych.
(b) Wzorce funkcji stosujemy, gdy definiujemy funkcje różniące się tylko typem zwracanej wartości, a typy parametrów i czynności, które mają wykonać, są identyczne. Na podstawie wzorca funkcji kompilator tworzy odpowiednie warianty funkcji różniące się tylko typami zwracanych wartości.
(c) Wzorce funkcji stosujemy, gdy definiujemy funkcje różniące się tylko typem zmiennych lokalnych, a typy argumentów i czynności, które mają wykonać, są identyczne. Na podstawie wzorca funkcji kompilator tworzy odpowiednie warianty funkcji różniące się tylko typami zmiennych lokalnych.

Zadanie 4. Co to jest klasa?

(a) Zbiór obiektów określonego typu.
(b) Obiekt określonego typu.
(c) Typ danych posiadający strukturę danych w postaci własności i interfejs złożony z metod.

Zadanie 5. Czym się różni struktura od klasy?

(a) Struktura i klasa są tym samym.
(b) Struktura to szczególny przypadek klasy. W strukturze wszystko domyślnie jest publiczne, w klasie wszystko domyślnie jest prywatne.
(c) Klasa to szczególny przypadek struktury. W strukturze wszystko domyślnie jest prywatne, w klasie wszystko domyślnie jest publiczne.

Zadanie 6. Co to znaczy, że funkcja ma argumenty o wartościach domyślnych?

(a) Argument funkcji ma wartość domyślną, gdy ją zawołamy z mniejszą ilością argumentów i kompilator w miejsce brakujących argumentów wstawi 0.
(b) Funkcja ma argument o wartości domyślnej, gdy w jej nagłówku podamy wartość tego argumentu oddzielając go znakiem `=`. Wołając funkcję możemy pominąć argumenty z wartościami domyślnymi.
(c) W C++ nie można przypisać domyślnej wartości argumentom funkcji.

Zadanie 7. Co to jest konstruktor?

(a) Konstruktor to dowolna metoda, która zwaraca obiekt danej klasy.
(b) Konstruktor to metoda, która nie ma argumentów, albo ma tylko argumenty domyślne.
(c) Konstruktor to metoda składowa klasy wywoływana podczas tworzenia nowego obiektu zaraz po zaalokowaniu pamięci dla tego obiektu. Nazwa konstruktora jest identyczna jak nazwa klasy.

Zadanie 8. Co to jest konstruktor domyślny?

(a) Konstruktor domyślny to pierwszy konstruktor zadeklarowany w klasie.
(b) Konstruktor jest konstruktorem domyślnym tylko wtedy, gdy może być wywołany bez żadnych argumentów. To oznacza, że albo nie ma argumentów wcale, albo wszystkie jego argumenty muszą mieć zadane wartości domyślne.
(c) Konstruktor domyślny to konstruktor tworzony przez kompilator.

Zadanie 9. Ile argumentów może mieć konstruktor domyślny?

(a) Dowolnie wiele.
(b) Zero.
(c) Jeśli ma jakieś argumenty to wszystkie muszą mieć wartości domyślne.

Zadanie 10. Co to jest destruktor?

(a) Destruktor to metoda składowa klasy wywoływana automatycznie podczas niszczenia obiektu, tuż przed zwolnieniem zajmowanej przez niego pamięci. Nazwa destruktora to nazwa klasy poprzedzona znakiem `~`.
(b) Destruktor to dowolna metoda składowa klasy, która zwalnia pamięć zajmowaną przez obiekt.
(c) Destruktor to dowolna metoda składowa klasy, której nazwa zaczyna się od znaku `~`.

Zadanie 11. Ile argumentów może mieć destruktor?

(a) 0
(b) 1
(c) Dowolnie wiele.

Zadanie 12. Co to jest obiekt?

(a) Egzemplarz (instancja) klasy.
(b) Inna nazwa istniejącej klasy.
(c) Adres klasy.

Zadanie 13. Co to jest referencja?

(a) Zmienna wskaźnikowa.
(b) Inna nazwa istniejącego obiektu.
(c) Adres obiektu w pamięci.

Zadanie 14. Co wypisze poniższy program?

```
int k = 1;
int &r = k;
r = 2;
cout << k << endl;
```

(a) 1
(b) 2
(c) Kompilator zgłosi błąd w drugiej linii bo powinno tam być `int *r = &k`.

Zadanie 15. Który konstruktor zostanie wykonany dla obiektu `sc` w poniższym programie?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(int c);
  SpaceCraft(char c);
};

int main() {
  SpaceCraft sc = 101;
}
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 16. Który konstruktor zostanie wykonany dla obiektu `sc` klasy `SpaceCraft` z programu w zadaniu 15?

```
  SpaceCraft sc;
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 17. Który konstruktor zostanie wykonany dla obiektu `sc` klasy `SpaceCraft` z programu w zadaniu 15?

```
  SpaceCraft sc = 'X';
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 18. Który konstruktor zostanie wykonany dla obiektu `sc1`, a który dla `sc2` w poniższym programie?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(int c);
  explicit SpaceCraft(char c);
};

int main() {
  SpaceCraft sc1 = 'X';
  SpaceCraft sc2 = SpaceCraft('Y');
}
```

(a) Dla `sc1` i dla `sc2` konstruktor `SpaceCraft(char c)`.
(b) Dla `sc1` konstruktor `SpaceCraft(int c)`, a dla `sc2` konstruktor `SpaceCraft(char c)`.
(c) Dla `sc1` konstruktor `SpaceCraft()`, a dla `sc2` konstruktor `SpaceCraft(char c)`.

Zadanie 19. Czym jest zmienna `this`?

(a) To wskaźnik do kopii obiektu utworzonej na stosie podczas przekazywania referencji do tego obiektu.
(b) To wskaźnik do klasy, dostępny w metodach składowych uruchamianych na rzecz tej klasy.
(c) To wskaźnik do obiektu, dostępny w metodach składowych uruchamianych na rzecz tego obiektu.

Zadanie 20. Co wyróżnia metodę od funkcji w programowaniu obiektowym?

(a) Metoda to inna nazwa funkcji.
(b) Metoda to funkcja, która ma dodatkowy, niejawny parametr, którym jest wskaźnik `this` do obiektu na rzecz którego jest wołana.
(c) Metoda to funkcja, której argumentami są obiekty.

Zadanie 21. Przy wywołaniu funkcji `foo()` czy `bar()` zostanie wykonany konstruktor kopiujący?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(const SpaceCraft &sc);
};

int foo(const SpaceCraft sc);
int bar(const SpaceCraft &sc);

int main() {
  SpaceCraft sc;
  cout << foo(sc) << " : " << bar(sc) << endl;
}
```

(a) `foo()` bo przekazywany jest parametr przez wartość, więc na stosie tworzona jest jego kopia.
(b) `bar()` bo przekazywany jest parametr przez referencję, więc na stosie tworzona jest jego kopia.
(c) Zarówno przy `foo()` jak i `bar()` bo przekazywany jest jako parametr obiekt klasy `SpaceCraft`.

Zadanie 22. Co oznacza słowo kluczowe `const` w poniższym przykładzie?

```
class SpaceCraft {
public:
  SpaceCraft();
  int getCounter(char *str) const;
};
```

(a) Wszystkie parametry metody `getCounter()` są stałe i nie mogą być zmieniane.
(b) Metoda `getCounter()` jest stała i nie może zmienić składowych własności obiektu, na rzecz którego została wywołana, ale może zmieniać składowe własności innych obiektów
(c) Metoda `getCounter()` jest stała i nie może zmieniać składowych własności żadnych obiektów.

Zadanie 23. Ile argumentów należy przekazać, gdy przeciążamy operator dwuargumentowy jako składowy wewnątrz klasy?

(a) 0
(b) 1
(c) 2

Zadanie 24. Która deklaracja przeciążenia operatora + jako składowego jest poprawna?

(a) `Fraction operator+(const Fraction &f, const Fraction &g) const;`
(b) `Fraction operator+(const Fraction &f) const;`
(c) `void operator+(const Fraction &f, const Fraction &g);`

Zadanie 25. Która deklaracja przeciążenia operatora + jako globalnego jest poprawna?

(a) `operator+(const Fraction &f, const Fraction &g);`
(b) `Fraction operator+(const Fraction &f);`
(c) `Fraction operator+(const Fraction &f, const Fraction &g);`

Zadanie 26. Czy operator przypisania = można przeciążać poza klasą jako globalny?

(a) Nie. Operator przypisania = generowany jest automatycznie jako składowy w klasie, jeśli sami go nie przeciążymy w tej klasie. Gdybyśmy przeciążyli go jako operator globalny to kompilator miałby do wyboru dwa operatory przypisania = dla danej klasy i powstałaby niejednoznaczność.
(b) Tak. Operator przypisania = tak jak pozostałe operatory przypisania można przeciążać poza klasą.
(c) Nie. Operator przypisania = jest wyjątkowy bo modyfikuje dany obiekt i dlatego nie można przeciążać go poza klasą.

Zadanie 27. Ile argumentów należy przekazać, gdy przeciążamy operator jednoargumentowy poza klasą jako globalny?

(a) 0
(b) 1
(c) 2

Zadanie 28. Na czym polega dziedziczenie w programowaniu obiektowym?

(a) Dziedziczenie polega na tworzeniu bardziej szczegółowych, wyspecjalizowanych klas na podstawie klas bardziej ogólnych, o większym stopniu abstrakcji. Klasy potomne posiadają te same własności i metody co ich rodzice, więc nie trzeba definiować ich całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy.
(b) Dziedziczenie polega na tworzenia bardziej zaawansowanych klas poprzez wykorzystanie kodu dostępnego w postaci bibliotek. Nowe klasy powstałe w ten sposób posiadają więcej funkcjonalności.
(c) Dziedziczenie polega na przekazywaniu danych z klasy potomnej bardziej szczegółowej, wyspecjalizowanej do klasy nadrzędnej bardziej ogólnej, o większym stopniu abstrakcji. Klasy potomne posiadają nowe własności i metody w odróżnieniu do ich rodziców.

Zadanie 29. Co klasa potomna dziedziczy po klasie bazowej?

(a) Tylko metody składowe.
(b) Tylko własności składowe.
(c) Wszystkie własności oraz metody poza konstruktorami i destruktorami.

Zadanie 30. Do których składowych własności i metod klasy bazowej mamy dostęp w klasie pochodnej?

(a) Do wszystkich składowych.
(b) Do składowych publicznych (public).
(c) Do składowych publicznych (public) i chronionych (protected).

Zadanie 31. Jaka jest dostępność metody `info()` w klasie Square przy następującej implementacji:

```
class Figure {
  public:
      void info() const;
};

class Square : protected Figure {
  public:
      double area() const;
};
```

(a) Publiczna (public).
(b) Chroniona (protected).
(c) Prywatna (private).

Zadanie 32. W jakiej kolejności wołane są konstruktory w przypadku dziedziczenia?

(a) Kolejność wołania konstruktorów możemy określić w definicji klasy.
(b) Najpierw wołany jest konstruktor klasy pochodnej, potem bazowej.
(c) Najpierw wołany jest konstruktor klasy bazowej, potem pochodnej.

Zadanie 33. Mamy klasę `Polygon` i jej klasę pochodną `Triangle`. Czy możliwe jest poniższe przypisanie?

```
Polygon p;
Triangle t;
p = t;
```

(a) Kompilator zgłosi błąd i kompilacja zostanie przerwana bo typy zmiennych `p` i `t` są różne.
(b) Kompilator zgłosi ostrzeżenie, że typy zmiennych `p` i `t` są różne, ale program się skompiluje.
(c) Tak, program skompiluje się poprawnie.

Zadanie 34. W programie mamy klasę `Point` i jej klasę pochodną `Pixel`. Mamy także funkcję o następującym prototypie:

```
float distance(Point &p1, Point &p2);
```

Czy poniższe wywołanie funkcji `distance()` jest poprawne?

```
Pixel px1(0, 0, 0), px2(1, 1, 255);
cout << "Distance: " << distance(px1, px2) << endl;
```

(a) Tak, ponieważ typ dynamiczny przekazywanych argumentów jest bardziej szczegółowy niż typ statyczny argumentów w prototypie funkcji `distance()`.
(b) Nie, bo typ dynamiczny przekazywanych argumentów jest bardziej ogólny niż typ statyczny argumentów w prototypie funkcji `distance()`.
(c) Nie, bo typy argumentów w prototypie są inne niż typy przekazywanych argumentów podczas wołania funkcji `distance()`.

Zadanie 35. Na co może wskazywać wskaźnik do obiektu klasy X?

(a) Wyłącznie na obiekty klasy X.
(b) Na obiekty klasy X oraz obiekty klas od niej bardziej ogólnych.
(c) Na obiekty klasy X oraz obiekty klas od niej bardziej szczegółowych.

Zadanie 36. Co to jest metoda wirtualna?

(a) Metoda zadeklarowana ze słowem kluczowym `virtual`, zaimplementowana w klasie bazowej, nie można tej metody nadpisać/przesłonić w klasie potomnej.
(b) Metoda wirtualna to każda metoda w klasie polimorficznej.
(c) Metoda zadeklarowana ze słowem kluczowym `virtual`, może nie być zaimplementowana w klasie bazowej, używana do uzyskania polimorfizmu.

Zadanie 37. Jaki mechanizm programowania obiektowego realizują metody wirtualne?

(a) Dziedziczenie.
(b) Polimorfizm.
(c) Hermetyzację.

Zadanie 38. Czy konstruktory w C++ mogą być wirtualne?

(a) Tak.
(b) Nie.
(c) Tylko wtedy gdy cała klasa jest wirtualna.

Zadanie 39. Co wypisze następujący program:

```
class Polygon {
  public:
    float circumference() const { return 0; };
    virtual float area() const { return 1; };
};

class Rectangle: public Polygon {
  public:
    float circumference() const { return 2; };
    float area() const { return 3; }
};

int main () {
  Polygon *p = new Rectangle;
  cout << p->circumference() << " : " << p->area() << endl;
}
```

(a) 0 : 3
(b) 2 : 3
(c) 0 : 1

Zadanie 40. Jaki jest typ statyczny i dynamiczny zmiennej `x` w poniższym programie?

```
class Vehicle {
     ...
};
class Truck : public Vehicle {
     ...
};
Vehicle *x = new Truck;
```

Jaki jest typ statyczny i dynamiczny zmiennej `x`?

(a) Typ statyczny: wskaźnik na klasę `Vehicle`. Typ dynamiczny: wskaźnik na klasę `Truck`.
(b) Typ statyczny: wskaźnik na klasę `Truck`. Typ dynamiczny: wskaźnik na klasę `Vehicle`.
(c) Typ statyczny: wskaźnik na klasę `Vehicle`. Typ dynamiczny: wskaźnik na klasę `Vehicle`.

Zadanie 41. Czym wyróżnia się klasa abstrakcyjna?

(a) Zawsze można tworzyć obiekty tej klasy. Metody abstrakcyjne nie będą wówczas dostępne.
(b) Można tworzyć obiekty tej klasy, ale lepiej jest zaimplementować (przeciążając) jej wszystkie metody abstrakcyjne.
(c) Nie można tworzyć obiektów tej klasy. Najpierw należy zaimplementować (przesłaniając) jej wszystkie metody abstrakcyjne tworząc klasę pochodną.

Zadanie 42. Jak powstaje klasa abstrakcyjna?

(a) Klasa abstrakcyjna powstaje wówczas, gdy dopiszemy sepcyfikator `virtual` przed `class`.
(b) Klasa abstrakcyjna powstaje wówczas, gdy dopiszemy sepcyfikator `abstract` przed `class`.
(c) Klasa abstrakcyjna powstaje wówczas, gdy choć jedna jej metoda jest czysto wirtualna (pure virtual).

Zadanie 43. Czy destruktor może być wirtualny?

(a) Destruktor nie może być wirtualny.
(b) Desktruktor może być wirtualny wyłącznie w klasie abstrakcyjnej.
(c) Desktruktor z reguły powinien być wirtualny, aby umożliwić zwolnienie zasobów: dealokację pamięci, zamknięcie otwartych plików itp. gdy wołany jest dla obiektu o typie statycznym bardziej ogólnym.

Zadanie 44. Która z poniższych klas jest abstrakcyjna?

(a)
```
class Polygon {
  public:
    Polygon();
    virtual float area() const = 0;
  ...
};
```

(b)
```
class Polygon {
  public:
    Polygon();
    virtual float area() const { return 0; };
  ...
};
```

(c)
```
abstract class Polygon {
  public:
    Polygon();
    virtual float area() const;
  ...
};
```

Zadanie 45. Która instrukcja spowduje zgłoszenie wyjątku?

(a) `terminate "expected non-zero value";`
(b) `throw "expected non-zero value";`
(c) `exception "expected non-zero value";`

Zadanie 46. W jaki sposób możemy obsłużyć (przechwycić, wyłapać) wyjątek?

(a) Deklarując własną funkcję zakończenia programu przez `set_terminate()`
(b) Umieszczając kod w bloku: `try{} catch(type t){}`
(c) Umieszczając kod w bloku: `throw{} catch(type t){}`

Zadanie 47. W jakim celu w `catch(type t)` przekazywany jest parametr?

(a) Jeśli został zgłoszony wyjątek typu/klasy `t` to zostanie wykonany ten właśnie blok instrukcji, a parametr `t` może być w tym bloku użyty.
(b) Parametr `t` zostanie zwrócony podczas zakończenia programu, gdy powstanie wyjątek.
(c) Parametr `t` zostanie przekazany do funkcji `set_terminate()`, gdy powstanie wyjątek.

Zadanie 48. Jaki wyjątek wyłapie `catch(...)`?

(a) Każdy wyjątek.
(b) Każdy wyjątek do tej pory nie wyłapany.
(c) Każdy wyłapany wyjątek.

Zadanie 49. Mamy dwa, jeden za drugim, bloki: `catch(int x){}` oraz `catch(char *s){}`. Który z nich zostanie wykonany, gdy w ich zasięgu zgłosimy `throw "exception"`?

(a) `catch(int x){}`
(b) `catch(char *s){}`
(c) Żaden z nich.

Zadanie 50. Mamy dwie klasy: `A` oraz jej pochodną `B` oraz dwa bloki: `catch(A){}` oraz `catch(B){}` jeden za drugim. Który z tych bloków zostanie wykonany po zgłoszeniu `throw B()`?

(a) `catch(A)`
(b) `catch(B)`
(c) `catch(A)` oraz kolejno `catch(B)`

Wstęp do programowania obiektowego

Egzamin poprawkowy, Wilno 2023-07-17

Imię: ..... Nazwisko: .....

Zadanie 1. Która instrukcja pobierze wartość z terminala i wstawi do zmiennej całkowitej `x`?

(a) `x << cin;`
(b) `cin >> x;`
(c) `cin << x;`

Zadanie 2. Co to znaczy, że funkcja jest przeciążona?

(a) Funkcja jest przeciążona, gdy wywołuje samą siebie.
(b) Funkcja jest przeciążona, gdy zwraca różne typy danych.
(c) Funkcja jest przeciążona, gdy jej identyfikator użyty jest dwa lub więcej razy w definicjach funkcji różniących się zestawem parametrów, tzn. ilością lub typami argumentów.

Zadanie 3. Co to jest wzorzec funkcji?

(a) Wzorce funkcji stosujemy, gdy definiujemy funkcje różniące się tylko typem argumentów, a czynności, które mają wykonać, są identyczne. Na podstawie wzorca funkcji kompilator tworzy odpowiednie warianty funkcji różniące się tylko typami parametrów, zwracanych wartości i ewentualnie zmiennych lokalnych.
(b) Wzorce funkcji stosujemy, gdy definiujemy funkcje różniące się tylko typem zwracanej wartości, a typy parametrów i czynności, które mają wykonać, są identyczne. Na podstawie wzorca funkcji kompilator tworzy odpowiednie warianty funkcji różniące się tylko typami zwracanych wartości.
(c) Wzorce funkcji stosujemy, gdy definiujemy funkcje różniące się tylko typem zmiennych lokalnych, a typy argumentów i czynności, które mają wykonać, są identyczne. Na podstawie wzorca funkcji kompilator tworzy odpowiednie warianty funkcji różniące się tylko typami zmiennych lokalnych.

Zadanie 4. Co to jest klasa?

(a) Zbiór obiektów określonego typu.
(b) Obiekt określonego typu.
(c) Typ danych posiadający strukturę danych w postaci własności i interfejs złożony z metod.

Zadanie 5. Czym się różni struktura od klasy?

(a) Struktura i klasa są tym samym.
(b) Struktura to szczególny przypadek klasy. W strukturze wszystko domyślnie jest publiczne, w klasie wszystko domyślnie jest prywatne.
(c) Klasa to szczególny przypadek struktury. W strukturze wszystko domyślnie jest prywatne, w klasie wszystko domyślnie jest publiczne.

Zadanie 6. Co to znaczy, że funkcja ma argumenty o wartościach domyślnych?

(a) Argument funkcji ma wartość domyślną, gdy ją zawołamy z mniejszą ilością argumentów i kompilator w miejsce brakujących argumentów wstawi 0.
(b) Funkcja ma argument o wartości domyślnej, gdy w jej nagłówku podamy wartość tego argumentu oddzielając go znakiem `=`. Wołając funkcję możemy pominąć argumenty z wartościami domyślnymi.
(c) W C++ nie można przypisać domyślnej wartości argumentom funkcji.

Zadanie 7. Co to jest konstruktor?

(a) Konstruktor to dowolna metoda, która zwaraca obiekt danej klasy.
(b) Konstruktor to metoda, która nie ma argumentów, albo ma tylko argumenty domyślne.
(c) Konstruktor to metoda składowa klasy wywoływana podczas tworzenia nowego obiektu zaraz po zaalokowaniu pamięci dla tego obiektu. Nazwa konstruktora jest identyczna jak nazwa klasy.

Zadanie 8. Co to jest konstruktor domyślny?

(a) Konstruktor domyślny to pierwszy konstruktor zadeklarowany w klasie.
(b) Konstruktor jest konstruktorem domyślnym tylko wtedy, gdy może być wywołany bez żadnych argumentów. To oznacza, że albo nie ma argumentów wcale, albo wszystkie jego argumenty muszą mieć zadane wartości domyślne.
(c) Konstruktor domyślny to konstruktor tworzony przez kompilator.

Zadanie 9. Ile argumentów może mieć konstruktor domyślny?

(a) Dowolnie wiele.
(b) Zero.
(c) Jeśli ma jakieś argumenty to wszystkie muszą mieć wartości domyślne.

Zadanie 10. Co to jest destruktor?

(a) Destruktor to metoda składowa klasy wywoływana automatycznie podczas niszczenia obiektu, tuż przed zwolnieniem zajmowanej przez niego pamięci. Nazwa destruktora to nazwa klasy poprzedzona znakiem `~`.
(b) Destruktor to dowolna metoda składowa klasy, która zwalnia pamięć zajmowaną przez obiekt.
(c) Destruktor to dowolna metoda składowa klasy, której nazwa zaczyna się od znaku `~`.

Zadanie 11. Ile argumentów może mieć destruktor?

(a) Dowolnie wiele.
(b) Zero.
(c) Jeśli ma jakieś argumenty to wszystkie muszą mieć wartości domyślne.

Zadanie 12. Co to jest obiekt?

(a) To samo co klasa.
(b) Egzemplarz (instancja) klasy.
(c) Adres klasy umieszczonej w pamięci.

Zadanie 13. Co to jest referencja?

(a) Zmienna wskaźnikowa.
(b) Inna nazwa istniejącego obiektu.
(c) Adres klasy w pamięci.

Zadanie 14. Co wypisze poniższy program?

```
int k = 1;
int &r = k;
r = 2;
cout << k << endl;
```

(a) 1
(b) 2
(c) Kompilator zgłosi błąd w drugiej linii bo powinno tam być `int *r = &k`.

Zadanie 15. Który konstruktor zostanie wykonany dla obiektu `sc` w poniższym programie?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(int c);
  SpaceCraft(char c);
};

int main() {
  SpaceCraft sc = 101;
}
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 16. Który konstruktor zostanie wykonany dla obiektu `sc` klasy `SpaceCraft` z programu w zadaniu 15?

```
  SpaceCraft sc;
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 17. Który konstruktor zostanie wykonany dla obiektu `sc` klasy `SpaceCraft` z programu w zadaniu 15?

```
  SpaceCraft sc = 'X';
```

(a) `SpaceCraft();`
(b) `SpaceCraft(int c);`
(c) `SpaceCraft(char c);`

Zadanie 18. Który konstruktor zostanie wykonany dla obiektu `sc1`, a który dla `sc2` w poniższym programie?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(int c);
  explicit SpaceCraft(char c);
};

int main() {
  SpaceCraft sc1 = 'X';
  SpaceCraft sc2 = SpaceCraft('Y');
}
```

(a) Dla `sc1` i dla `sc2` konstruktor `SpaceCraft(char c)`.
(b) Dla `sc1` konstruktor `SpaceCraft(int c)`, a dla `sc2` konstruktor `SpaceCraft(char c)`.
(c) Dla `sc1` konstruktor `SpaceCraft()`, a dla `sc2` konstruktor `SpaceCraft(char c)`.

Zadanie 19. Czym jest zmienna `this`?

(a) To wskaźnik do kopii obiektu utworzonej na stosie podczas przekazywania referencji do tego obiektu.
(b) To wskaźnik do klasy, dostępny w metodach składowych uruchamianych na rzecz tej klasy.
(c) To wskaźnik do obiektu, dostępny w metodach składowych uruchamianych na rzecz tego obiektu.

Zadanie 20. Co wyróżnia metodę od funkcji w programowaniu obiektowym?

(a) Metoda to inna nazwa funkcji.
(b) Metoda to funkcja, która ma dodatkowy, niejawny parametr, którym jest wskaźnik `this` do obiektu na rzecz którego jest wołana.
(c) Metoda to funkcja, której argumentami są obiekty.

Zadanie 21. Przy wywołaniu funkcji `foo()` czy `bar()` zostanie wykonany konstruktor kopiujący?

```
class SpaceCraft {
public:
  SpaceCraft();
  SpaceCraft(const SpaceCraft &sc);
};

int foo(const SpaceCraft sc);
int bar(const SpaceCraft &sc);

int main() {
  SpaceCraft sc;
  cout << foo(sc) << " : " << bar(sc) << endl;
}
```

(a) `foo()` bo przekazywany jest parametr przez wartość, więc na stosie tworzona jest jego kopia.
(b) `bar()` bo przekazywany jest parametr przez referencję, więc na stosie tworzona jest jego kopia.
(c) Zarówno przy `foo()` jak i `bar()` bo przekazywany jest jako parametr obiekt klasy `SpaceCraft`.

Zadanie 22. Co oznacza słowo kluczowe `const` w poniższym przykładzie?

```
class SpaceCraft {
public:
  SpaceCraft();
  int getCounter(char *str) const;
};
```

(a) Wszystkie parametry metody `getCounter()` są stałe i nie mogą być zmieniane.
(b) Metoda `getCounter()` jest stała i nie może zmienić składowych własności obiektu, na rzecz którego została wywołana, ale może zmieniać składowe własności innych obiektów
(c) Metoda `getCounter()` jest stała i nie może zmieniać składowych własności żadnych obiektów.

Zadanie 23. Ile argumentów należy przekazać, gdy przeciążamy operator dwuargumentowy jako składowy wewnątrz klasy?

(a) 0
(b) 1
(c) 2

Zadanie 24. Która deklaracja przeciążenia operatora + jako składowego jest poprawna?

(a) `Fraction operator+(const Fraction &f, const Fraction &g) const;`
(b) `Fraction operator+(const Fraction &f) const;`
(c) `void operator+(const Fraction &f, const Fraction &g);`

Zadanie 25. Która deklaracja przeciążenia operatora + jako globalnego jest poprawna?

(a) `operator+(const Fraction &f, const Fraction &g);`
(b) `Fraction operator+(const Fraction &f);`
(c) `Fraction operator+(const Fraction &f, const Fraction &g);`

Zadanie 26. Czy operator przypisania = można przeciążać poza klasą jako globalny?

(a) Nie. Operator przypisania = generowany jest automatycznie jako składowy w klasie, jeśli sami go nie przeciążymy w tej klasie. Gdybyśmy przeciążyli go jako operator globalny to kompilator miałby do wyboru dwa operatory przypisania = dla danej klasy i powstałaby niejednoznaczność.
(b) Tak. Operator przypisania = tak jak pozostałe operatory przypisania można przeciążać poza klasą.
(c) Nie. Operator przypisania = jest wyjątkowy bo modyfikuje dany obiekt i dlatego nie można przeciążać go poza klasą.

Zadanie 27. Ile argumentów należy przekazać, gdy przeciążamy operator jednoargumentowy poza klasą jako globalny?

(a) 0
(b) 1
(c) 2

Zadanie 28. Na czym polega dziedziczenie w programowaniu obiektowym?

(a) Dziedziczenie polega na tworzeniu bardziej szczegółowych, wyspecjalizowanych klas na podstawie klas bardziej ogólnych, o większym stopniu abstrakcji. Klasy potomne posiadają te same własności i metody co ich rodzice, więc nie trzeba definiować ich całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy.
(b) Dziedziczenie polega na tworzenia bardziej zaawansowanych klas poprzez wykorzystanie kodu dostępnego w postaci bibliotek. Nowe klasy powstałe w ten sposób posiadają więcej funkcjonalności.
(c) Dziedziczenie polega na przekazywaniu danych z klasy potomnej bardziej szczegółowej, wyspecjalizowanej do klasy nadrzędnej bardziej ogólnej, o większym stopniu abstrakcji. Klasy potomne posiadają nowe własności i metody w odróżnieniu do ich rodziców.

Zadanie 29. Co klasa potomna dziedziczy po klasie bazowej?

(a) Tylko metody składowe.
(b) Tylko własności składowe.
(c) Wszystkie własności oraz metody poza konstruktorami i destruktorami.

Zadanie 30. Do których składowych własności i metod klasy bazowej mamy dostęp w klasie pochodnej?

(a) Do wszystkich składowych.
(b) Do składowych publicznych (public).
(c) Do składowych publicznych (public) i chronionych (protected).

Zadanie 31. Jaka jest dostępność metody `info()` w klasie Square przy następującej implementacji:

```
class Figure {
  public:
      void info() const;
};

class Square : protected Figure {
  public:
      double area() const;
};
```

(a) Publiczna (public).
(b) Chroniona (protected).
(c) Prywatna (private).

Zadanie 32. W jakiej kolejności wołane są konstruktory w przypadku dziedziczenia?

(a) Kolejność wołania konstruktorów możemy określić w definicji klasy.
(b) Najpierw wołany jest konstruktor klasy pochodnej, potem bazowej.
(c) Najpierw wołany jest konstruktor klasy bazowej, potem pochodnej.

Zadanie 33. Mamy klasę `Polygon` i jej klasę pochodną `Triangle`. Czy możliwe jest poniższe przypisanie?

```
Polygon p;
Triangle t;
p = t;
```

(a) Kompilator zgłosi błąd i kompilacja zostanie przerwana bo typy zmiennych `p` i `t` są różne.
(b) Kompilator zgłosi ostrzeżenie, że typy zmiennych `p` i `t` są różne, ale program się skompiluje.
(c) Tak, program skompiluje się poprawnie.

Zadanie 34. W programie mamy klasę `Point` i jej klasę pochodną `Pixel`. Mamy także funkcję o następującym prototypie:

```
float distance(Point &p1, Point &p2);
```

Czy poniższe wywołanie funkcji `distance()` jest poprawne?

```
Pixel px1(0, 0, 0), px2(1, 1, 255);
cout << "Distance: " << distance(px1, px2) << endl;
```

(a) Tak, ponieważ typ dynamiczny przekazywanych argumentów jest bardziej szczegółowy niż typ statyczny argumentów w prototypie funkcji `distance()`.
(b) Nie, bo typ dynamiczny przekazywanych argumentów jest bardziej ogólny niż typ statyczny argumentów w prototypie funkcji `distance()`.
(c) Nie, bo typy argumentów w prototypie są inne niż typy przekazywanych argumentów podczas wołania funkcji `distance()`.

Zadanie 35. Na co może wskazywać wskaźnik do obiektu klasy X?

(a) Wyłącznie na obiekty klasy X.
(b) Na obiekty klasy X oraz obiekty klas od niej bardziej ogólnych.
(c) Na obiekty klasy X oraz obiekty klas od niej bardziej szczegółowych.

Zadanie 36. Co to jest metoda wirtualna?

(a) Metoda zadeklarowana ze słowem kluczowym `virtual`, zaimplementowana w klasie bazowej, nie można tej metody nadpisać/przesłonić w klasie potomnej.
(b) Metoda wirtualna to każda metoda w klasie polimorficznej.
(c) Metoda zadeklarowana ze słowem kluczowym `virtual`, może nie być zaimplementowana w klasie bazowej, używana do uzyskania polimorfizmu.

Zadanie 37. Jaki mechanizm programowania obiektowego realizują metody wirtualne?

(a) Dziedziczenie.
(b) Polimorfizm.
(c) Hermetyzację.

Zadanie 38. Czy konstruktory w C++ mogą być wirtualne?

(a) Tak.
(b) Nie.
(c) Tylko wtedy gdy cała klasa jest wirtualna.

Zadanie 39. Co wypisze następujący program:

```
class Polygon {
  public:
    float circumference() const { return 0; };
    virtual float area() const { return 1; };
};

class Rectangle: public Polygon {
  public:
    float circumference() const { return 2; };
    float area() const { return 3; }
};

int main () {
  Polygon *p = new Rectangle;
  cout << p->circumference() << " : " << p->area() << endl;
}
```

(a) 0 : 3
(b) 2 : 3
(c) 0 : 1

Zadanie 40. Jaki jest typ statyczny i dynamiczny zmiennej `x` w poniższym programie?

```
class Vehicle {
     ...
};
class Truck : public Vehicle {
     ...
};
Vehicle *x = new Truck;
```

Jaki jest typ statyczny i dynamiczny zmiennej `x`?

(a) Typ statyczny: wskaźnik na klasę `Vehicle`. Typ dynamiczny: wskaźnik na klasę `Truck`.
(b) Typ statyczny: wskaźnik na klasę `Truck`. Typ dynamiczny: wskaźnik na klasę `Vehicle`.
(c) Typ statyczny: wskaźnik na klasę `Vehicle`. Typ dynamiczny: wskaźnik na klasę `Vehicle`.

Zadanie 41. Czym wyróżnia się klasa abstrakcyjna?

(a) Zawsze można tworzyć obiekty tej klasy. Metody abstrakcyjne nie będą wówczas dostępne.
(b) Można tworzyć obiekty tej klasy, ale lepiej jest zaimplementować (przeciążając) jej wszystkie metody abstrakcyjne.
(c) Nie można tworzyć obiektów tej klasy. Najpierw należy zaimplementować (przesłaniając) jej wszystkie metody abstrakcyjne tworząc klasę pochodną.

Zadanie 42. Jak powstaje klasa abstrakcyjna?

(a) Klasa abstrakcyjna powstaje wówczas, gdy dopiszemy sepcyfikator `virtual` przed `class`.
(b) Klasa abstrakcyjna powstaje wówczas, gdy dopiszemy sepcyfikator `abstract` przed `class`.
(c) Klasa abstrakcyjna powstaje wówczas, gdy choć jedna jej metoda jest czysto wirtualna (pure virtual).

Zadanie 43. Czy destruktor może być wirtualny?

(a) Destruktor nie może być wirtualny.
(b) Desktruktor może być wirtualny wyłącznie w klasie abstrakcyjnej.
(c) Desktruktor z reguły powinien być wirtualny, aby umożliwić zwolnienie zasobów: dealokację pamięci, zamknięcie otwartych plików itp. gdy wołany jest dla obiektu o typie statycznym bardziej ogólnym.

Zadanie 44. Która z poniższych klas jest abstrakcyjna?

(a)
```
class Polygon {
  public:
    Polygon();
    virtual float area() const = 0;
  ...
};
```

(b)
```
class Polygon {
  public:
    Polygon();
    virtual float area() const { return 0; };
  ...
};
```

(c)
```
abstract class Polygon {
  public:
    Polygon();
    virtual float area() const;
  ...
};
```

Zadanie 45. Która instrukcja spowduje zgłoszenie wyjątku?

(a) `terminate "expected non-zero value";`
(b) `throw "expected non-zero value";`
(c) `exception "expected non-zero value";`

Zadanie 46. W jaki sposób możemy obsłużyć (przechwycić, wyłapać) wyjątek?

(a) Deklarując własną funkcję zakończenia programu przez `set_terminate()`
(b) Umieszczając kod w bloku: `try{} catch(type t){}`
(c) Umieszczając kod w bloku: `throw{} catch(type t){}`

Zadanie 47. W jakim celu w `catch(type t)` przekazywany jest parametr?

(a) Jeśli został zgłoszony wyjątek typu/klasy `t` to zostanie wykonany ten właśnie blok instrukcji, a parametr `t` może być w tym bloku użyty.
(b) Parametr `t` zostanie zwrócony podczas zakończenia programu, gdy powstanie wyjątek.
(c) Parametr `t` zostanie przekazany do funkcji `set_terminate()`, gdy powstanie wyjątek.

Zadanie 48. Jaki wyjątek wyłapie `catch(...)`?

(a) Każdy wyjątek.
(b) Każdy wyjątek do tej pory nie wyłapany.
(c) Każdy wyłapany wyjątek.

Zadanie 49. Mamy dwa, jeden za drugim, bloki: `catch(int x){}` oraz `catch(char *s){}`. Który z nich zostanie wykonany, gdy w ich zasięgu zgłosimy `throw "exception"`?

(a) `catch(int x){}`
(b) `catch(char *s){}`
(c) Żaden z nich.

Zadanie 50. Mamy dwie klasy: `A` oraz jej pochodną `B` oraz dwa bloki: `catch(A){}` oraz `catch(B){}` jeden za drugim. Który z tych bloków zostanie wykonany po zgłoszeniu `throw B()`?

(a) `catch(A)`
(b) `catch(B)`
(c) `catch(A)` oraz kolejno `catch(B)`