Навигация в теста



Мартин Попов

1 2 3 4 5 6 7 8 9
10 11

Показване по един въпрос на страница

Край на прегледа

| | |
|---|---|
| **Започнат на** | сряда, 7 юли 2021, 11:16 |
| **Състояние** | Завършен |
| **Приключен на** | сряда, 7 юли 2021, 12:12 |
| **Изминало време** | 55 мин. 56 сек. |
| **Точки** | 12,50/20,00 |
| **Оценка** | **6,25** от 10,00 (**63%**) |

Въпрос **1**

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Нека са дадени следните дефиниции:

```
class something {
public:
   explicit something(int)
   {}
};

void f(int)
{}

void g(something)
{}
```

Кой от следните изрази ще се компилира?

- `g(5);`
- `g(something(5));` ✔
- `something obj = 10;`
- `f(something(5));`

Your answer is correct.

Правилният отговор е:

```
g(something(5));
```

Въпрос 2

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Нека са дадени следните дефиниции:

```
class Test {
};

void by_ref(Test& param) {}
void by_cref(const Test& param) {}
void by_value(Test param) {}
void by_cvalue(const Test param) {}

int main()
{
   by_ref(Test());
   by_cref(Test());
   by_value(Test());
   by_cvalue(Test());
}
```

Във функцията `main` се съдържат поредица от обръщенията към функциите, при които им подаваме обект от тип `Test`. За всяко от тях посочете дали е коректно или ще предизвика грешка.

`by_ref(Test());` грешка ✔

`by_cref(Test());` коректно ✔

`by_value(Test());` коректно ✔

`by_cvalue(Test());` коректно ✔

Your answer is correct.

Правилният отговор е: `by_ref(Test());` → грешка, `by_cref(Test());` → коректно, `by_value(Test());` → коректно, `by_cvalue(Test());` → коректно

Въпрос 3

Неправилен отговор

0,00 от максимално 1,00 точки

Отбелязване на въпроса

Какъв вид копиране извършва автоматично генерираният оператор за присвояване в дадения по-долу фрагмент?

```
class Internal {
   int a;
   Internal& operator=(const Internal& other)
   {
      a = other.a;
      return *this;
   }
};

class External {
   Internal obj;
   int b;
};

int main()
{
   External x, y;
   x = y;
}
```

- Задължително е да се използва покомпонентно копиране с оператора за присвояване на всеки от членовете.
- Компилаторът не може да генерира оператор за присвояване.
- Компилаторът може да използва тривиално копиране (например с `memcpy` или `memmove`)

Your answer is incorrect.

Операторът за присвояване на `Internal` е `private`.

Правилният отговор е: Компилаторът не може да генерира оператор за присвояване.

Въпрос 4

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Каква версия на оператора за присвояване ще генерира компилаторът за дадения по-долу фрагмент?

```
class Test {
   int var = 0;
};

int main()
{
   Test a, b;
   a = b;
   return 0;
}
```

- Test& operator=(const Test &) ✔
- Test& operator=(Test &)
- Test& operator=(Test)
- Компилаторът няма да генерира оператор за присвояване, защото променливата var е private.

Your answer is correct.

Правилният отговор е: Test& operator=(const Test &)

Въпрос 5

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Безопасно ли е да копираме данни между два обекта, от произволен клас, с функцията memmove?

Изберете едно:

- Истина
- Лъжа ✔

Правилният отговор е "Неистина"

Въпрос 6

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Възможно ли е в един клас да се дефинират няколко различни копиращи конструктора?

Изберете едно:

- (•) Истина ✔
- ( ) Лъжа

Правилният отговор е "Истина"

Въпрос 7

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Кои от следните оператори можем да предефинираме?

Всеки верен отговор увеличава точките за въпроса, а всеки погрешен ги намалява. В скоби до операторите са посочени техните имена.

- [x] `&&` (and) ✔
- [x] `()` (function call) ✔
- [ ] `sizeof`
- [ ] `::` (scope resolution)

Your answer is correct.

Правилните отговори са: `()` (function call), `&&` (and)

Въпрос: **8**

Правилен отговор

1,00 от максимално 1,00 точки

Отбелязване на въпроса

Посочете вярно ли е следното твърдение.

"В C++, когато предефинираме оператор, можем да променим броя на неговите параметри."

Изберете едно:

- Истина
- Лъжа ✔

Правилният отговор е "Неистина"

Въпрос: **9**

Отговорен

1,50 от максимално 4,00 точки

Отбелязване на въпроса

Обяснете следните неща свързани с виртуалното наследяване в C++:

1. Обяснете какво представлява виртуалното наследяване.
2. Защо ни е нужно то (какъв проблем решава)?
3. В какъв ред работят конструкторите когато имаме виртуално наследяване.
4. Каква особеност има по отношение на викането на конструктор на виртуално наследен клас от неговите наследници.

Дайте примери.

1. Виртуалното наследяване накратко представлява обекти, които наследяват виртуално някакъв друг обект. (по принцип имаме два обекта, които наследяват виртуално един по-общ)

2. Идеята е следната:

Имаме клас Person и класове Worker и Father, също и клас Son. Ако наследявахме всичко стандартно, то Father и Worker наследяват Person и Son наследява Father и Worker, понеже интуитивно става въпрос за един и същ Person. Но ако направим така ще излезе, че може да имаме различни Person. Този проблем е известен като диамантения проблем, понеже ако се начертае на схема това всичкото, прилича на диамант. За да решим това, трябва да наследим виртуално -> virtual public Person, понеже така ще има един общ обект Person за Father и Worker.

3. Редът е стандартен, първо се създава обект от тип Person (при виртуално наследяване се вика веднъж, иначе ще се извика 2 пъти), после Worker и Father и накрая Son

4.

```
class Person{};

class Worker: virtual public Person{};

class Father: virtual public Person{};

class Son: public Worker, public Father{};
```

Коментар:

Въпрос: **10**

Отговорен

0,00 от максимално 4,00 точки

Отбелязване на въпроса

Обяснете полиморфичното клониране на обект в C++:

1. Какво представлява то и защо ни е нужно?
2. Обяснете как работи.
3. Дайте пример.

Нека имаме:

```
class A{
public:
f(){}
};
class B: public A{
public:
f(){}
};

B b;
b.f();
A& a = b;
a.f();
```

При тази ситуация, имаме едни и същи функции f в класовете A и B, но при стандартното наследяване, изпълнението на функциите се определя повреме на компилация, тоест ще е ясно, че за b ще се извика функцията за класа B, а за a - тази за класа A. Тук идва това, което ще реши този проблем. Ако направим функциите f виртуално:

```
virtual f(){}
```

Тогава при всяко следващо наследяване ще можем да предефинираме функцията за съответния клас. Съответно в този случай викането на функциите се определя по време на изпълнение. Тоест за b - функцията на B, но и за a от тип A ще се извика функцията на B понеже обекта, който размножаваме (това е b) е използван. В такъв случай ако имаме и клас C

```
class C: public A{};
```

Можем да направим масив от елементи от тип A

```
A array[10]
```

и да го напълним с наследението му елементи, като ги представяме като по-главния клас A.

В такива ситуации обаче трябва да направим и деструкторите виртуални, понеже, когато преобразуваме в различни класове, то работим само с част от класа, тоест ако имаме:

```
B b;
A* a = &b;
```

Тогава когато се унищожи a, то се унищожава само частта на A, а частта на B си остава, което може да доведе до memory leak.

Въпрос **11**

Отговорен

4,00 от максимално 4,00 точки

Отбелязване на въпроса

Обяснете следните неща свързани с жизнения цикъл на обектите в C++:

1. Нека имаме клас X, който наследява други класове, а също и има членове, които са обекти на други класове. В какъв ред работят конструкторите и деструкторите при създаване на обект от тип X?
2. Илюстрирайте отговора си с пример.
3. Какво представляват rule of 0 и rule of 3? Обяснете ги накратко.

1. Редът на конструкторите е същият, като редицата на наследяване. Например X наследява клас Y -> първо ще се извика конструктора за Y, който е "по-големият" клас, а след това този на X. За членовете на X ситуацията е същата. Първо се влиза в конструктора на X и след това, дали с initializer list или по друг начин се викат съответните конструктори на член-данните на X. Редът на деструкторите е обратен на този на конструкторите. Ето същият пример, първо ще се извика деструкторът на член-данните, след това ще се унищожи класът X и след това класът Y.

2. Ето един пример:

```
class Y{

};

class Z{

};

class X: public Y{

Z var;

};

X object; //първо се създава обект Y, после се създава обект X и след това се създава обект Z, който е вътре в X
```

3. Rule of 0 - това е когато създадем клас, например:

```
class X{
int a = 2;
public:
X(){}
};
```

И можем да добавяме функционалност, но имаме само default конструктор за да създаваме обекти от този клас, или може и някакъв друг инициализиращ конструктор, но нямаме нужда да се грижим и за паметта, която използваме.

Ако използваме rule of 3, например:

```
class Y{
std::vector<int*> intArray;
public:
Y(){} //constructor
Y& copy(const Y& other){} //copy constructor
Y& operator=(const Y& other){} //copy assignment operator
~Y(){} //destructor
};
```

Тук използваме копиращ конструктор, копиращ оператор и деструктор, понеже работим с динамична памет. Като копираме елементи от тип Y, не можем да оставим компилатора само да генерира тези функции, понеже той ги генерира тривиално, а на нас ни трябва експлицитно да покажем как трябва да става копирането между два обекта от този тип. Съответно деструктора трябва да освобождава заделената от нас памет, за да нямаме memory leaks.

Коментар: