Програмски језици 1 - скрипта теорије за завршни испит

Никола Марковић (2025)

- ова скрипта је срочена са једним једини циљем – исправљање грешака из осталих скрипта које су у циркулацији
- највећим дијелом је коришћена за личну употребу али како се многи распитују за скрипте, мало сам је додатно форматирао и учинио јавном
- ако сагледамо испите са ранијих година и како су се постављала питања, 99% питања која ће се икада појавити на завршним испитима се налазе овдје, али то свакако није гаранција било каквог типа нити продаја приче да је ово мало текста довољно да се изађе са стопостотном сигурношћу на испит, свакако се предлаже да научимо уздуж и попријеко најосновнија правила C++-а и/или генерално ООП принципа

1. **Шта изведена класа не насљеђује од основне:**

- конструкторе и операторе додјеле (operator=)
- деструктор
- релације пријатељства

2. **Који оператори су већ унапријед дефинисани:**

- оператор додјеле (operator=)
- оператор адресирања (operator&)
- зарез оператор (operator,)

3. **Објаснити set_terminate():**

Према C++ стандарду, уколико се било гдје баци неочекиван изузетак (нпр. изузетак који је бачен и није ухваћен унутар try-catch клаузуле или је бачен унутар функције квалификоване са noexcept), позива се функција std::terminate() која потом подразумијевано позива std::abort() и прекида извршавање програма. Функција std::set_terminate((*f)) нам омогућава да уз аргумент показивача на властиту функцију осигурамо да std::exception(), умјесто std::abort(), позове нашу функцију.

4. **Када се inline захтјев игнорише:**

Неке од ситуација су:

- ако је тијело функције сувише велико
- ако функција садржи петљу
- ако функција садржи статичке промјенљиве
- ако функција ради са I/O
- ако је функција рекурзивна
- ако функција није void типа али нема return наредбу у тијелу
- ако функција садржи switch/goto

5. **Када се позива конструктор копије:**

- приликом иницијализације објекта (експлицитне са заградама или имплицитне са ‘=’)
- приликом преноса аргумента у функцију
- приликом враћања вриједности из функције
- ^ за друга два се наравно подразумијева да шаљемо/враћамо **по вриједности**, тј. НЕ као референцу или показивач

6. **Да ли конструктор и деструктор могу бити виртуелне функције?**

- конструктор не може
- деструктор може; ово се користи код полиморфизма гдје деалокација динамичког објекта основне класе, уколико она садржи виртуелни деструктор, позива деструкторе свих изведених класа у дијелу хијерархије извођења између класе која енкапсулира полиморфни објекат (у примјеру, B* енкапсулира објекат типа D) и најосновније класе датог дијела хијерархије (A у примјеру); нпр. ако имамо:

A ← B ← C ← D ← E ← F

```
B* obj = new D;
delete obj;
```

- редом се позивају деструктори класа: D, C, B, A

У примјеру је за наведено понашање неопходно да само B посједује виртуелни деструктор.

7. **Објаснити спецификаторе override и final:**

- override – наглашава да дата метода (која мора бити virtual-квалификована) надјачава понашање најближе методе “изнад” (у суперкласи) ње са истим потписом у хијерархији извођења
- final – наглашава да се
    1) дата виртуелна метода у даљим извођењима класа не може више надјачати
    2) дата класа не може даље насљеђивати (је посљедња у хијерархији извођења)

8. **Објаснити функцију &get(char* niz, int max):**

Ова функција врло једноставно у низ карактера наведен као први аргумент учитава карактере из тока над којим се позове. Други аргумент ограничава максималан број катактера који се могу учитати.

9. **Које су особине релације пријатељства?**

- не насљеђује се
- није симетрична (ако је A наводи B као пријатеља, не значи да и B наводи A као пријатеља)
- није транзитивна (ако A наводи B као пријатеља, не значи да су класе изведене од B такође пријатељи од A)

10. **Који header file је потребно укључити да би могли да користимо функцију find/sort над неким објектом контејнерске класе vector?**

- треба укључити <algorithm> (#include <algorithm>)
- подразумијева се да је вектор <vector> већ експлицитно укључен или га индиректно укључује неко друго заглавље

11. **Класа A је изведена из класе B, а класа B из класе C. Којим редослиједом треба поређати catch клаузуле чији су типови редом A, B и C?**

Гледавши дату хијерархију A→B→C, клаузуле треба поредати редом за A, па за B, па коначно за C, јер ако би редослијед био обрнут, C би као општа наткласа ухватила сваки изузетак једног од ова три типа и остале двије клаузуле би представљале недокучив код. Укратко, редамо их од најспецијализованијег случаја ка најгенералнијем. Примјер правилног руковања:

```
try { … }
    catch (const A& exception) {...} catch (..B&..) {...} catch (..C&..) {...}
```

12. **Како треба да изгледа други/трећи аргумент конструктора копије/помјерања?**

Очекује се да конструктори копије и помјерања раде када су позвани са **једним** стандардим аргументом (const T& / T&&). Као резултат, очекује се да се, ако се додају додатни аргументи, они имају подразумијевану вриједност како не би било потребно да буду експлицитно наведени. На примјер:

```
T(const T& other) { … }                ✅
T(const T& other, int n = 6) { … }     ✅
T(const T& other, int n) { … }         ❌
```

13. Шта треба урадити да изведена класа посједује само један подобјекат индиректне основне класе?

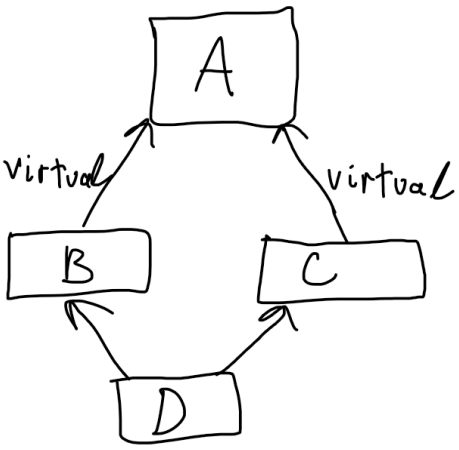
Треба подкласе које ће бити вишеструко насљеђење од стране неке треће класе извести виртуелно из њима основне. На примјер:

```
class A { … };

class B : virtual public A { … };

class C : virtual public A { … };

class D : public B, public C { …
};
```



Без виртуелног извођења, класа D би наслиједила двије инстанце класе A (једну преко B и једну преко C), док овако насљеђује једну.

14. Који је подразумијевани начин извођења класе:

- класе се подразумијевано (без навођења) изводе приватно:
  class A : private B {...} ⇔ class A: B {...}

15. Које особине оператора не могу да се мијењају:

- n-арност (нпр. operator+ је увијек бинаран (захтијева два операнда))
- приоритет (нпр. код израза &a++, оператор инкрементовања ‘++’ ће увијек бити евалуиран прије оператора адресе ‘&’, без обзира да ли је оператор подразумијеван или преклопљен)
- асоцијативност (нпр. A = B ће увијек да буде “промјенљивој A се додијељује вриједност промјенљиве B”, а не обрнуто)

16. Ког типа ће бити изузетак који је креиран само позивом ‘throw’ без додатног израза иза ‘throw’:

- ово је дозвољено искључиво унутар catch блокова и бациће изузетак истог типа који нас је и довео у тај блок

17. Набројати и објаснити врсте специјализације шаблона:

Узимамо за примјер да имамо класу:
template <typename T, size_t N> class Array {...}

- дјелимична специјализација – само неки од параметара шаблона су специјализовани, нпр:
  template<size_t N> Array<double, N> {...}
  template <typename T> Array<T, 5> {...}

- потпуна специјализација – сваки параметар шаблона је специјализован, нпр:
  template<> Array<double, 5> {...}

18. Које врсте специјализације су могуће код генеричких функција?

- само потпуна специјализација

19. Како се дефинише виртуелна статичка чланица класе:

- никако, јер није могуће
- виртуелне методе се вежу за објекте, док статичке за читаве класе, па имамо контрадикцију

20. Како се креирају имена операторских функција:

- <return type> operator@(<args…>)
- @ се наравно мијења са неким од преклопљивих оператора уграђених у C++

21. Које је дејство методе seekg?

- позиционирање унутар датотеке како би се читање/писање започело од наведене позиције умјесто од почетка

22. Из којих функција се не могу изазвати изузеци?

- default конструктора, конструктора копије и оператора додјеле копијом
- default деструктора
- static-квалификованих
- const-квалификованих

23. Шта су итератори?

Итератори су објекти којима се итерира (обилази) кроз елементе дате колекције. За разлику од нпр. низова који су секвенцијални у меморији, структуре попут мапа нису и итерација кроз обоје се C-астим приступом не може објединити. У C++-у итератори користе преклапање одређених оператора попут инкремента да обједине тј. апстрахују итерацију кроз апсолутно сваки тип структуре у једну цјелину која је за програмера наизглед увијек иста и константна.

24. Шта је апстрактна класа?

Апстрактна класа је класа која садржи барем једну чисто виртуелну (апстрактну) методу, односно методу без тијела означену са '= 0'. Не можемо инстанцирати објекте апстрактне класе, већ је искључиво користимо као шаблон за извођењекоји изведене класе морају да прате.
Класу код које је свака метода апстрактна називамо **интерфејс**.

25. Карактеристике јавног/приватног/заштићеног извођења:

Приватно извођење карактерише сљедећа конверзија спецификатора видљивости чланица:

Основна класа	Јавно (public) изведена поткласа	Заштићено (protected) изведена поткласа	Приватно (private) изведена поткласа
public	public	protected	private
protected	protected	protected	private
private	//	//	//

У суштини, приватни чланови основне класе су увијек неприступачни, док се остали "спуштају" на ниво извођења уколико су раније били већи од њега.

26. Која је намјена функције istream& get(char& znak)?

Функција, када се позове над неким током, просто уписује један знак из тог тока у промјенљиву znak прослијеђену као атрибут. Разлог што враћа референцу на себе је да се може уланчавати, нпр: std::cin.get(...).get(...).get(...). Вишак уноса се игнорише.

27. Навести редослијед позива конструктора при креирању објекта изведене класе:

- врло једноставно се позива конструктор највише класе надоле док се не дође до тражене класе у хијерархији, нпр:

A←B←C←D

C obj; // poziva se konstruktor A, pa B, pa C

28. Навести оператор за приступ члановима класе:

- тачка оператор (.)
- стрелица оператор (->) (за динамичке објекте)
- бинарни резолуциони оператор (::) (за статичке чланове)

29. Објаснити намјену функције set_unexpected():

Ова функција, слично као и раније наведена set_terminate(), узима показивач на функцију коју потом позива када се **успјешно ухвати изузетак у try блоку** али он се не поклапа са очекиваним изузетком наведеним у catch изразу.

30. Објаснити дејство функције istream& read(char* niz, int broj):

Ова функција врло једноставно чита *broj* карактера из тока над којим се позове и пише их у *niz* наведен као први аргумент. Стандардно, враћа референцу на себе да омогући уланчавање.

31. Да ли деструктор може бити виртуелна функција?

- може
- раније смо навели и зашто и како се користи

32. Ако лијеви операнд операторске функције треба да буде стандардног типа, како се мора дефинисати ова функција?

Лијеви операнд је у случају операторске функције дефинисане на нивоу класе (као метода) увијек инстанца класе (а други операнд је први аргумент), док се у случају глобалних функција наводи као први аргумент (а други операнд је други аргумент). Ако радимо са

стандардним типовима тј. било којим типом који није класа коју смо сами дефинисали, закључујемо да операторска функција **мора** бити глобална. Такође, најчешће мора бити и пријатељска функција да приступи приватним атрибутима класе.

33. Који оператори не могу да се преклапају?

- тернарни оператор – ?:
- зарез оператор – ,
- тачка оператор – .
- показивачки тачка оператор – .*
- бинарни резолуциони оператор – ::
- *sizeof*
- *typeid*

34. Редослијед позива конструктора и деструктора за аутоматске локалне објекте:

Већ смо навели да се код насљеђивања конструктори позивају редом од основне класе ка изведеној. За деструкторе важи обрнуто.

Конструктори се позивају када се дати објекат декларише, а деструктори се позивају када изађе из опсега (scope-a).

35. Који су услови за активирање виртуелног механизма?

За активирање виртуелног механизма услов је да се објектима приступа преко показивача. Наравно, подразумијева се да су тражене функције квалификоване као виртуелне.

36. Ако је met() метода класе PK која је пријатељска класи A, којим атрибутима класе A може да се приступи у функцији met()?

- све једном

37. Која врста асоцијативности важи за сложене (composite) операторе додјеле:

- сложеним операторима додјеле се називају оператори који врше и операцију и додјелу истовремено (нпр. *=, +=...)
- важи RTL (right-to-left) асоцијативност

38. Ког је типа повратна вриједност постинкремент оператора?

Примјер постинкремент оператора: n++

- повратна вриједност је тренутна вриједност операнда; нпр. ако је n=6, n++ ће вратити 6 али ће потом инкрементирати вриједност n у меморији на 7
- ово омогућава да урадимо нешто попут Array[n++], гдје ћемо прво приступити низу по индексу 6, а тек онда инкрементирати n на 7
- аналогно, прединкремент враћа већ инкрементирану вр.
- све исто важи и за декремент само што се вриједност умањује

39. Редослијед позива конструктора класе C изведене из класе A и класе B:

- прво се позивају конструктори наткласа редослиједом којим су наведени при извођењу, а тек потом конструктор класе C, нпр:

```
class A { ... };
class B { ... };
class C : public A, public B { ... };
```

^ У овом случају се позивају конструктори A, B па C.

```
class A { ... };
class B { ... };
class C : public B, public A { ... };
```

^ У овом случају се позивају конструктори B, A па C.

40. Шта могу бити параметри шаблонске класе?

Могу бити:

- цјелобројни типови (int, size_t…)
- енумератори
- референце на објекте или типове
- показивачи на функције и друге објекте

- показивачи на чланове
- други шаблони (композиција шаблона)

41. Објаснити дејство оператора noexcept:

Квалификовање неке методе са овим оператором служи као неки вид обећања компајлеру да дата метода неће да баца изузетке ни под којим условима, при чему компајлер може да изврши неке додатне оптимизације. Уколико се ово прекрши (функција ипак баци изузетак), одмах се позива std::terminate() и изузетак неће пратити стандардан процес руковања изузецима који се огледа у одмотавању стека, чишћењу и потенцијалном хватању изузетка у try-catch блоку, већ ће нагло прекинути извршавање.

42. Који је максималан број стандардних токова који се могу креирати у функцији чланице класе?

- под стандардним токовима се подразумијевају cin, cout, cerr, clog токови; за разлику од неких других токова (нпр. std::ifstream) се не могу “креирати” или експлицитно отварати/затварати, већ њима програм рукује аутоматски по потреби
- као такви могу се користити неограничен број пута
- осталих токова се може креирати неограничен број

43. Која врста асоцијативности важи за оператор %=

- као што је раније наведено за композитне операторе генерално међу које спада и овај, вриједи RTL

44. Која врста асоцијативности важи за оператор >>?

- LTR

45. Шта се добија примјеном explicit испред конструктора?

- постоји објекат Object b;
- без ове кључне ријечи, позив конструктора са једним параметром (нпр. конструктора копије) се може извршити:
    - имплицитно (Object a = b;)
    - експлицитно (Object a(b))

- уз додатак ове кључне ријечи уз дати конструктор, компајлер ће да сигнализира грешку при коришћењу имплицитног позива попут овог изнад

46. У којим случајевима хендлер типа A може да прихвати изузетак типа B?

- A и B су истог типа
- A је базна класа за B која је из ње изведена (ово омогућава 1. конверзију из B у A и 2. полиморфно понашање)

47. Анонимни простор имена (namespace):

Анонимним просторим имена називамо namespace блок који је декларисан без идентификатора (имена), попут:

namespace { … }, за разлику од стандардног namespace <ime> { ... }

Карактерише га то да има досег само на нивоу дате датотеке, тј. да се његови идентификатори не могу користити ван ње. Идентификатори унутар њега из очигледних разлога такође не могу да се поклапају са глобалним идентификаторима. Уз то, користи се без бинарног резолуционог оператора (::).

48. Који услов треба да задовољи позив функције за коју не постоји одговарајућа дефиниција да би се генерисала одговарајућа функција на основу шаблона?

Када преводилац наиђе на позив функције за коју не постоји дефиниција, а постоји одговарајућа генеричка функција, он генерише одговарајућу дефиницију функције на основу шаблона и наведених шаблонских параметара.

49. Како треба реализовати функцију T1 operator@ (T2, T3)

Ја најискреније не знам шта је тачан одговор овдје. 99% сам сигуран да треба редизајнирати потпис функције јер најчешће нема смисла и може се интерпретирати на безброј начина. Лично, ако добијем ово питање, навешћу да је неправилно дефинисана функција и промијенити је у неку са униформалним типовима нпр.
T1 operator@(T1, T1) { … } и онда нешто урадити са овим.