USMENI DEO ISPITA

RAND

- **Faze u razvoju C++ programa (navesti po redosledu).**
  - Faze u razvoju C++ programa podrazumevaju:
    - **Editovanje** – program kreniran u editoru i zapamćen na disku,
    - **Preprocesiranje** – program procesira programski kod,
    - **Kompajliranje** – Compiler kreira object code i smešta ga na disk,
    - **Linkovanje** – Linker spaja object code fajlove sa bibliotekama I kreira .out fajl (na disk),
    - **Učitavanje** – Loader učitava program u memoriju,
    - **Izvršavanje** – CPU uzima instrukcije iz programa i izvršava ih pamteći međurezultate.
- **Šta je klasa?**
  - ***Klasa*** je korisnički definisani tip koji služi kao shema za kreiranje objekata.
  - Grupa sličnih objekata.
  - Logički entitet.
  - Deklarisana jednom, bez alociranja memorije.
- **Šta je objekat?**
  - ***Objekat*** je instanca klase.
  - To je fizički, realni entitet.
  - Objekat je kreiran onoliko puta koliko je potrebno i pritom mu se alocira memorija.
- **Na koji način se mogu slati poruke objektima?**
  - Funkcije članice se aktiviraju kao odgovori na poruke. One služe za komunikaciju sa objektima.
- **Inline funkcije.**
  - ***Inline funkcije*** su funkcije za koje se ne dešava smeštanje na stack, već ih kompajler još u vreme kompajliranja na mestima gde su pozvane kopira, tako da se smanjuje vreme potrebno za njihovo izvršavanje, ali raste veličina binary executable fajla.
  - Inline funkcije su sve funkcije čija je implementacija unutar definicije klase, odnosno u klasi u kojoj su deklarisane ispred sebe imaju ključnu reč inline, a definicija se nalazi u istom .h fajlu kao i definicija klase.
  - Postoje ograničenja za to koja funkcija može biti inline. Funkcije koje kompajler ignoriše, iako su definisane kao inline su:
    - Funkcije koje sadrže petlje,
    - Rekurzivne funkcije,
    - Funkcije sa goto ili switch strukturom,
    - Funkcije čiji je povratni tip različit od void, a unutar njih nemaju return liniju,
    - Funkcije koje rade sa static promenljivama.
- **Kako kompajler poziva inline funkcije?**
  - Na mestu poziva kompajler kopira telo funkcije i ugrađuje je u izvorišni kod.
- **Specifikatori prava pristupa članovima klase.**

- Postoje 3 specifikatora za pravo prisupa unutar klase, to su:
  - **Public:** članovi klase su dostupni na svim mestima gde objekat klase ima scope.
  - **Protected:** članovi klase su dostupni metodama svoje klase, kao i metodama svih izvedenih klasa iz te klase.
  - **Private:** članovi klase su dostupni metodama klase, kao i metodama i klasama koje su prijatelji te klase.

Ako se u osnovnoj klasi iz nje izvedena klasa proglasi prijateljskom, ona ima pristup svim članovima osnovne klase.

- **Čemu služi typeid?**
  - ***typeid(objekat/tip)*** služi za ispitivanje tipa objekta.
- **Čemu služi static_cast?**
  - ***static_cast<tip>(objekat)*** služi za eksplicitno kastovanje (konvertovanje) jednog tipa objekta u drugi (ovo se uglavnom kombinuje sa ***typeid()***). Najveću primenu ima kada radimo sa objektima osnovne klase koji referenciraju objekte izvedene klase. Pogodno za copy constructor.
- **Čemu dynamic_cast?**
  - ***dynamic_cast<tip*>(objekat*)*** služi za eksplicitno konvertovanje (kastovanje) I najveću primenu ima kada radimo sa pokazivačima osnovne klase koji pokazuju na objekte izvedene klase.

KONSTRUKTORI / DESTRUKTORI

- **Šta je konstruktor?**
  - ***Konstruktor*** je posebna funkcija (subroutine-a) koja služi (biva pozivana) u trenutku kada je potrebno kreirati novi objekat određene klase. Nema povratnu vrednost, a ime je isto kao i ime klase čije objekte kreira. U njegovom telu se uglavnom nalaze naredbe za incijaliaciju atributa objekta.
  - Postoji nekoliko vrsta konstruktora:
    - **Default (podrazumevani)** – konstruktor bez argumenata
    - **Parametrizovani** – konstruktor sa bar jednim argumentom.
    - **Copy** - konstruktor za bar jedan od argumenata ima referencu na objekat klase koji kopira.
    - **Conversion** – služi kompajleru za implicitno (explicit-no) konvertovanje određenih tipova (koji su deo parametara parametrizovanog konstruktora) u objekte date klase.
    - **Move** – konstruktor za argument ima X&& I služi za preuzimanje “preusmeravanje” (pomeranje) resursa/memorije objekta (više o ovome sledeće godine).
- **Šta je destruktor?**
  - ***Destruktor*** je unikatna funkcija članica klase, koja nema povratni tip, ni argumente, njeno ime je definisano znakom ~ iza kog stoji ime klase i služi (poziva se) pri uništavanju, brisanju, dealociranju (memorije) objekata.
- **Da li je moguće preklopiti (overload) :**

A) ***Konstruktor***

- Da, s obzirom da klasa može imati više konstruktora, s tim što će se pozivati onaj kome odgovara lista argumenata za koju je pozvan.

B) ***Destruktor***

- Ne, s obzirom da klasa može imati samo jedan destruktor.

- **Koji tipovi podataka mogu da predstavljaju povratni tip konstruktora za kopiranje?**
  - Konstruktor nema povratnu vrednost (nije ni void), tako da ovo važi i za copy constructor.
- **Kog tipa može biti povratna vrednost konstruktora?**
  - Nijednog, jer konstruktor nema povratnu vrednost (ni void).
- **Redosled poziva konstruktora.**
  - **Globalni objekti :**
    - pre bilo koje druge funkcije (uključujući i main).
  - **Automatski lokalni objekti :**
    - kada se definišu objekti (svaki put kada u toku izvršenja dobiju scope).
  - **Statički lokalni objekti :**
    - tačno jedanput, kada se dođe do mesta gde je objekat definisan.

DIGRESIJA :

Konstruktore i destruktore kompajler poziva IMPLICITNO !

Ukoliko klasa ima konstruktor koji uzima samo jednu vrednost kao njegov argument, onda on postaje konstruktor konverzije i može se pozvati, ukoliko je to potrebno programu, kako bi se vrednost koja se konvertuje iskoristila za njegov parametar. OVO JE SA NETA I POKUŠAO SAM DA PREVEDEM SVOJIM REČIMA. NAJLAKŠE JE UZ PRIMER, IAKO GUBI NA

```cpp
#include <iostream>

using namespace std;

class Complex
{
private:
    double real;
    double imag;

public:
    // Default constructor
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    // A method to compare two Complex numbers
    bool operator == (Complex rhs) {
       return (real == rhs.real && imag == rhs.imag)? true : false;
    }
};

int main()
{
    // a Complex object
    Complex com1(3.0, 0.0);

    if (com1 == 3.0)
       cout << "Same";
    else
       cout << "Not Same";
     return 0;
}
```

```cpp
#include <iostream>

using namespace std;

class Complex
{
private:
    double real;
    double imag;

public:
    // Default constructor
    explicit Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    // A method to compare two Complex numbers
    bool operator== (Complex rhs) {
       return (real == rhs.real && imag == rhs.imag)? true : false;
    }
};

int main()
{
    // a Complex object
    Complex com1(3.0, 0.0);

    if (com1 == (Complex)3.0)
       cout << "Same";
    else
       cout << "Not Same";
     return 0;
}
```

GENERALIZACIJI, BUT... Feel free to resize the photos

Konstruktori ispred sebe mogu da sadrže ključnu reč **explicit**, što znači da svaki poziv poziv konstruktora mora biti "naglašen", odnosno gubi se konverziona funkcionalnost.

- **Poziv destruktora objekata.**
  - **Globalni objekti :**
    - Kada se main završi, ili je pozvana exit(<0>) ,
    - **NE POZIVA SE AKO SE PROGRAM ZAVRŠI SA ABORT().**
  - **Automatski lokalni objekti :**
    - Kada objekti izgube scope (prestanu da važe), završi se blok u kome je objekat bio definisan,
    - **NE POZIVA SE UKOLIKO SE PROGRAM ZAVRŠI SA EXIT() ILI ABORT().**
  - **Statički lokalni objekti :**
    - Kada se main završi, ili je pozvana exit(),
    - **NE POZIVA SE KADA SE PROGRAM ZAVRŠI SA ABORT().**
- **U kojim situacijama se poziva konstruktor kopije?**
  - Copy constructor se poziva kada se:
    - Objekat inicijalizuje pomoću operatora ***"="***. Objekat (iste klase), koji se kopira se nalazi s desne strane.
    - Objekat inicijalizuje pomoću ***"()"***. Objekat (iste klase), koji se kopira se nalazi u zagradama.
    - Objekat prenosi kao ***argument funkcije*** (kreira se lokalni automatski objekat).
    - Objekat vraća, kao ***povratna vrednost funkcije*** (kreira se privremeni objekat).
    - Kada kompajler generiše privremeni objekat.
- **Koliko argumenata može imati konstruktor kopije i šta važi za njih?**
  - ***Konstruktor kopije*** za jedan od argumenata ima referencu na objekat koji kopira, odnosno referencu na objekat koji ne može da menja unutar tela (const), dok ostali argumenti, onoliko koliko programer zada, moraju da imaju podrazumevane vrednosti.

OPERATORI

- **Šta je operatorska funkcija?**
  - ***Operatorska funkcija*** predstavlja operator za korisničke tipove, gde korisnik sam može da definiše ponašanje pojedinih operatora.
  - Praktično, one omućavaju svođenje pozivanja funkcija na matematičke izraze, koji su mnogo lakši za čitanje.
- **Koji se operatori koriste za pristup članovima klase i kada?**
  - Članovima klase se pristupa pomoću dva operatora:
    - ***Operator { . } – operator tačka***
      - Koristi se kada želimo da pristupimo članovima klase preko objekta ili reference na objekat.
      - Ovaj operator se koristi na sledeći način:
        - ***Obj.Ime ⇔ (&Obj) -> Ime***
          - Ime – predstavlja atribut klase kome želimo pristupiti
          - Obj – Objekat/ref na objekat klase koji sadrži imenovani atribut
    - ***Operator { -> } – operator dereferenciranja pokazivača i pristupa članu***
      - Koristi se kada želimo da pristupimo članovima klase preko pokazivača na objekat te klase.
      - Ovaj operator se koristi na sledeći način:
        - ***Pokazivač -> Ime ⇔ (*Pokazivač).Ime***
          - Ime – član klase kome želimo pristupiti
          - Pokazivač – pokazivač na objekat klase koji sadrži imenovanog člana
    - ***Oprator { :: } – binarni rezolucioni operator***
      - Pri definisanju metoda van definicije klase, odnosno za pristup statičkim članovima klase.
      - Ovaj operator se koristi na sledeći način:
        - ***Ime_klase::Ime_člana_klase***
- **Kako se kreiraju imena operatorskih funkcija?**
  - Operatorske funkcije nose ime ***operator@***, gde je @ bilo koji od ugrađenih operatora u C++ jeziku, osim { **.** , **.*** , **? :** , **sizeof** , **::** } operatora.
- **Ograničenja u preklapanju operatora?**
  - Osnovna pravila:
    - C++ dozvoljava preklapanje operatora (operator overloading), kao što dozvoljava I preklapanje imena funkcija.
    - Ograničenja su sledeća:
      - Ne mogu se preklapati operatori {“.”, “.*”, “?:”, “sizeof”, “::”},
      - Ne mogu da se redefinišu značenja operatora za primitivne (standardne) tipove podataka,
      - Ne mogu da se uvode novi simboli za operatore,
      - Ne mogu da se menjaju karakteristike operatora, koje u ugrađene u jezik:

  - N-arnost
  - Prioritet
  - Asocijativnost

- **Kako treba realizovati funkciju XXX operator@(int a, YYY b)?**
  - Ovu funkciju treba realizovati kao globalnu prijateljsku funkciju klase YYY.
- **Šta treba da vrate operatori koji menjaju levi operand?**
  - Operatori koji menjaju levi operand treba da da vrate ***referencu na levi operand***.
  - NPR: operator=, kao i sve njegove kombinacije +=,-=,*=,…
    - X& operator=(const X& x)
- **Povratna vrednost operatorskih funkcija preinkrement I predekrement?**
  - Operatorske funkcije X& operator++() I X& operator—() očigledno treba da vrate ***referencu na operand koji biva izmenjen*** (*this, ukoliko nije friend).
- **Povratna vrednost operatorskih funkcija postinkrement I postdekrement.**
  - Ova operatorska funkcija treba da vrati ***kopiju objekta date*** klase pre promene vrednosti tog objekta.
    - X operator++(int a),
    - X opertaor—(int a).
- **Koja je namena operatora ::?**
  - **Operator { :: } – binarni rezolucioni operator** suži za:
    - “Vezivanje” članova klase (atriuti ili funkcije) sa imenom klase,
    - Jedinstveno identifikovanje članova jedne klase,
    - Pristupanje statičkih članova klase.

FRIEND

- **Šta predstavlja relacija prijateljstva?**
  - ***Relacija prijateljstva*** predstavlja način regulacije prava pristupa određenim funkcijama ili klasama.
- **Navesti osobine relacije prijateljstva.**
  - Osobine prijateljstva, kao relacije:
    - ***Nije nasleđiva*** (moji potomci ne moraju biti prijatelji mojih prijatelja),
    - ***Nije simetrična*** (one koje ja smatram prijateljima, ne moraju da me smatraju njihovim prijateljem) i
    - ***Nije tranzitivna*** (prijatelj mog prijatelja ne mora da bude i moj prijatelj).

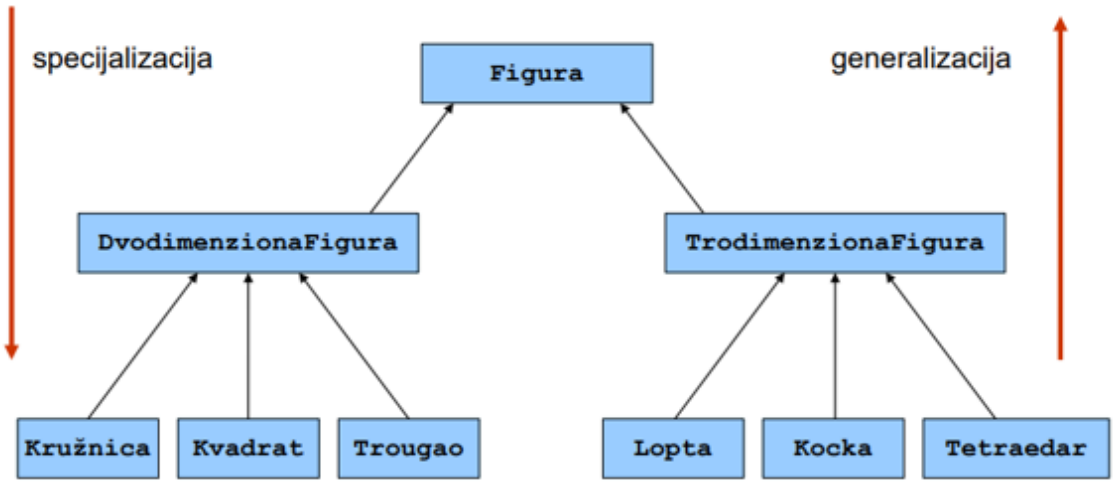
DIGRESIJA:

Prijateljstvo reguliše pravo pristupa, odnosno omogućava prijateljskim funkcijama i klasama da pristupe privatnim članovima one klase koja ih smatram prijateljima.

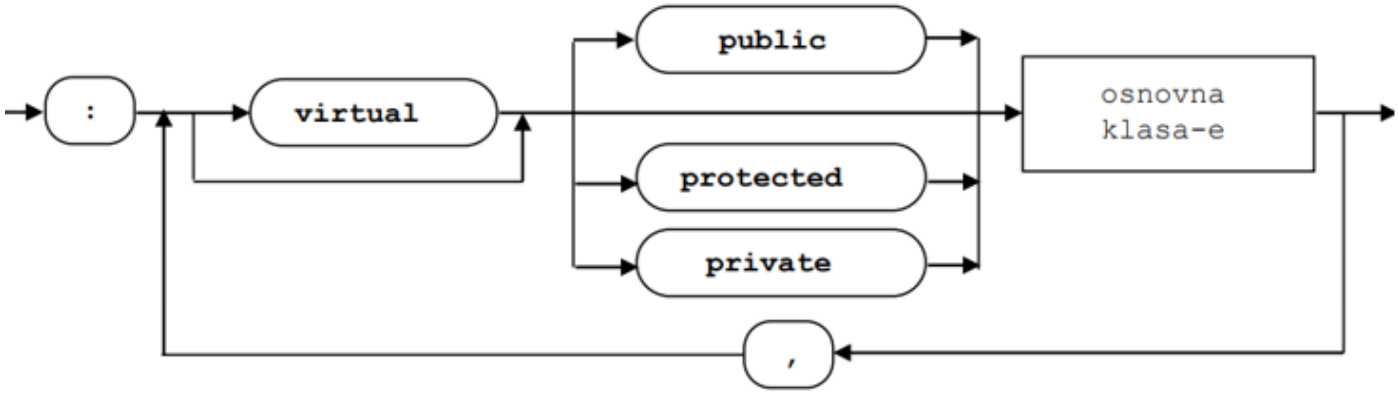
- **Kako prijateljska funkcija pristupa objektu klase kojoj je prijatelj pomoću pokazivača this?**
  - Nikako. Pokazivač "this" se koristi unutar metoda članica klase. S obzirom da prijateljska funkcija nije član klase, u njoj se ne koristi ovaj pokazivač.
- **U kom delu klase se definišu prijateljske funkcije?**
  - Nevažno je u kom delu se navode deklaracije, jer prijateljske funkcije nisu članice klase kojoj su prijatelj, već su ili članice neke druge klase, ili su globalne funkcije.
- **Koliko maksimalno parametara može imati globalna prijateljska funkcija?**
  - S obzirom da svaka globalna funkcija može biti prijateljska funkcija neke klase, ne postoji nikakva posebna restrikcija vezana za argumente te funkcije. S tim u vezi, ukoliko klasa nema statičke članove, ili globalna prijateljska funkcija nema za jedan od argumenata tip klase, onda se gubi njena funkcionalnost.

IZVOĐENJE

- **Kako se definiše izvedena klasa?**
  - ***Izvedena klasa*** se definiše tako što se između identifikatora klase i tela definicije klase navede { : } i imena klasa koje nasleđuje. Dijagram je dole
- **Koji proces je suprotan procesu specijalizacije?**
  - Generalno govoreći, proces koji je suprotan procesu specijalizacije je **generalizacija**, odnosno nalaženje istih karakteristika za veću grupu nečega. U našem slučaju, to podrazumeva nalaženje istih atributa i metoda za veći broj klasa i objedinjavanje svih njih u jednu roditeljsku klasu, iz koje dalje izvodimo specijalizovanije klase.



- **Nacrtati i objasniti sintaksni dijagram za definisanje izvedene klase.**



  - Dakle, nakon { : } mogu se definisati određeni identifikatori koji menjaju način na koji izvedena klasa nasleđuje osnovnu klasu.
    - **Virtual** : podobjekti ove izvedene klase će se u daljem izvođenju deliti sa svim podobjektima klasa, koje su nasledile ovako izvedenu klasu (indirektno osnovnu).
    - **Public** : prava pristupa se preslikavaju na izvedenu klasu na sledeći način:
      - Public -> public
      - Protected -> protected
      - Private -> nepristupačni
    - **Protected** : prava pristupa se preslikavaju na izvedenu klasu na sledeći način:
      - public-> protected,
      - protected-> protected,

        - private -> nepristupačni
    - **Private** : prava pristupa se preslikavaju na izvedenu klasu na sledeći način:
        - public-> private,
        - protected-> private,
        - private -> nepristupačni
    - **Default** : podrazumeva **Private**
- Ukoliko klasa nasleđuje više klasa, stavili bismo { , } i nastavili da na isti način navodimo i ostale klase koje nasleđuje. Sve te klase su osnovne klase izvedenoj klasi.

- **Nacrtati tabelu prava pristupa elementima izvedene klase u zavisnosti od načina izvođenja.**

Pristup osnovnoj klasi	Pristupačnost u privatno izvedenoj klasi	Pristupačnost u zaštićeno izvedenoj klasi	Pristupačnost u javno izvedenoj klasi
Nepristupačan	Nepristupačan	nepristupačan	nepristupačan
Privatni	Nepristupačan	nepristupačan	nepristupačan
Zaštićeni	Privatni	zaštiđeni	zaštićeni
Javni	Privatni	zaštićeni	javni

- **Koji je podrazumevani način izvođenja klase?**
    - Podrazumevani način izvođenja klase je ***nevirtuelni, private način izvođenja***.
- **Šta ne nasleđuje izvedena klasa?**
    - ***Izvedena klasa ne nasleđuje konstruktore, destruktor i operator= (dodele) iz osnovne klase***, uz to ona nema ni pravo pristupa private atributima. (Oni su nasleđeni, samo ne može da im se pristupi, osim ako izvedena klasa nije proglašena friend class-om u osnovnoj klasi).
- **Šta nasleđuje izvedena klasa?**
    - Izvedena klasa nasleđuje sve atribute i metode, sem gore navedenih.
- **Navesti redosled poziva konstruktora pri kreiranju objekta izvedene klase ?**
    - Poziv konstruktora klase je "rekurzivan" (ovo ne preporučujem da napišete, jer je to moj način shvatanja) i izvršava se sledećim redosledom:
        - ***Poziv konstruktora osnovne klase,***
        - ***Poziv konstruktora atributa izvedene klase (po redosledu deklarisanja),***
        - ***Izvršavanje tela odgovarajućeg konstruktora izvedene klase.***
- **Navesti poziv destruktora pri uništavanju objekta izvedene klase.**
    - Takođe "rekurzivan":
        - ***Izvršavanje tela destruktora izvedene klase,***
        - ***Poziv destruktora atributa izvedene klase,***
        - ***Poziv destruktora osnovne klase.***
- **Koji objekti se mogu dodeliti objektu osnovne klase?**
    - Objektu osnovne klase se može dodeliti:
        - ***Objekat osnovne klase,***
        - ***Objekat direktno izvedene klase,***
        - ***Objekat indirektno izvedene klase.***

- **Šta će biti odštampano na izlazu kada se izvrši main?**

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

class A
{
public:
    A();
    ~A();
    A(int n) { m_i = n; }
protected:
    static int m_i;
};

int A::m_i = 1;
A::A() { m_i = 2; }
A::~A() { cout << "A destruktor:\n" << m_i << endl; --m_i; }

class B
    :public A
{
public:
    B(int n)
        :m_a1(m_i + 1), m_a2(n - 1) {}
    ~B() { cout << "B desturktor:\n" << m_i << endl; }
private:
    A m_a1;
    int m_a2;
};

int main()
{
    {B b(6); }
    cout << endl;
    return 0;
}
```

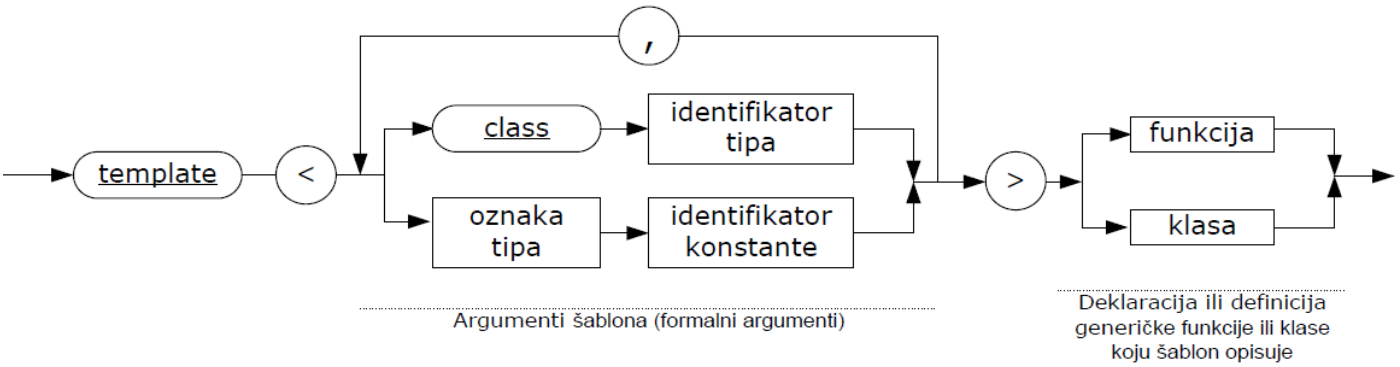
Na konzoli će biti odštampano:
B destruktor:
3
A destruktor: // zbog atributa m_a1
3
A destruktor: // zbog nasleđivanja klase A
2

VIRTUELNI MEHANIZAM / POLIMORFIZAM

- **Kako se difiniše virtuelna statička članica klase ?**
  - Ne može se definisati, jer virtuelna funkcija ne može biti statička.
- **Kada se aktivira virtuelni mehanizam?**
  - Virtuelni mehanizam se aktivira kada su zadovoljeni uslovi:
    1. ***Funkcija je u osnovnoj klasi definisana sa ključnom reči virtual.***
    2. ***Pokazivaču na objekat osnovne klase je dodeljena adresa izvedene klase referenci na objekat osnovne klase je dodeljen objekat izvedene klase***
    3. ***Objekat (2.) je pozvao funkciju definisanu (1.)***
- **Na koliko načina može da se definiše virtuelni konstruktor?**
  - ***Konstruktor ne može biti virtuelan***.
- **Polimorfizam.**
  - ***Polimorfizam*** (je mehanizam koji) omogućava da jedan interfejs može da služi više različitim entitetima, odnosno da iza jednog simbola može da se krije više različitih tipova.
- **Pomoću kojih operatora se može pristupiti objektima apstraktnih klasa?**
  - ***Ne mogu se kreirati objekti apstraktnih klasa, već one služe kako bi se nasleđivale***.
- **Šta je čista virtuelna funkcija?**
  - ***Čista virtuelna funkcija (je virtuelna funkcija, koja) umesto tela funkcije sadrži =0***.
- **Šta je apstraktna klasa?**
  - Apstraktna klasa (je klasa, koja) bar za jedan od svojih metoda sadrži čistu virtuelnu funkciju.

GENERICS

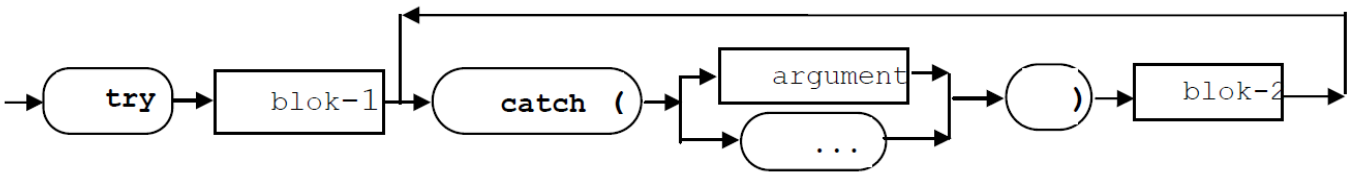
- **Šta je generik (šablon)?**
  - ***Generik*** predstavlja mehanizam koji omogućava da parametrizujemo tipove za funkcije i klase. To je jedan od načina sprečavanja dupliranja koda, kao i generalizacije.
- **Definicija šablona, dijagram i objašnjenje.**



  - Unutar dijamanta (<>) stoje (formalni) argumenti šablona:
    - **ID tipa** – koriste se unutar generika na svim mestima gde se očekuju tipovi.
    - **ID konstante** – koriste se kao simpoličke konstante unutar generika svuda gde se očekuju konstante.
    - **Oznaka tipa** – mogu biti standardni, ili korisnički tipovi.
- **Načini za generisanje funkcija iz šablonskih funkcija.**
  - Funkcije na osnovu zadatog šablona mogu da se generišu na dva način:
    - **Automatski** – kad se naiđe na poziv generičke funkcije sa stvarnim argumentima, koji mogu da se uklope u šablon, bez konverzije,
    - **Na zahtev programera** – nadođenjem deklaracije (prototipa) za datu generičku funkciju sa željenim argumentima.
- **Kako se vrši lokalna deklaracija šablonske klase?**
  - ***Deklaracija šablona može biti samo globalna***.
- **Kakva je priroda generičkog mehanizma?**
  - Mehanizam je ***statički*** – instance šablona se prave tekstualnom zamenom u vreme prevođenja.
- **Šta mogu biti fomralni argumenti šablonske klase?**
  - Formalni argumenti šablonske klase mogu biti :
    - ***Tipovi (typename T),***
    - ***Prosti objekti (konkretizovani konstantnim izrazima).***

IZUZECI

- **Šta je izuzetak?**
  - ***Izuzetak*** je događaj koji treba posebno obraditi izvan osnovnog toka programa.
- **Iz kojih funkcija se mogu izazvati izuzeci?**
  - Izuzeci se mogu izazvati iz bilo kog bloka, direktno i indirektno pozvane funkcije koji/a se nalazi unutar ***try{}*** bloka.
  - Izazivanje/prijavljivanje/bacanje izuzezatak se vrši naredbom ***THROW izraz***, gde je ***izraz*** neki tip objekta na osnovu kog se bira koji hendler će ga obraditi (glupo rečeno, ali bmk).
  - Ukoliko se bačen izuzetak ne uhvati, poziva se ***terminate().***
- **Kako se navodi spisak tipova izuzetaka koje funkcija izaziva?**
  - Spisak tipova izuzetaka koje funkcija može da izazove se navodi nakon argumenata funkcije kao:
    - ***Throw (lista tipova)***
  - Ukoliko se ovo ne napiše podrazumeva se da funkcija može da izazove bilo koji izuzetak.
  - Ukoliko se navede noexcept, funkcija ne bi trebalo da izazove izuzetak (ukoliko se to ipak desi, poziva se unexpected()).
- **Nacrtati i objasniti sintaksni dijagram za mehanizam za obradu izuzetaka.**



Blok1 – ***try blok*** u kome se proizvode (bacaju) izuzeci.

Blok2 – blok u kome se obrađuje određeni (ili neodređeni "…") izuzetak.

Argument – sastoji se od tipa i oznake izuzetka koji obrađuje. Kažemo da je hendler (catch blok) tipa T, kada obrađuje izuzetke tipa T, odnsono kada mu je argument T.

" … " – univerzalni hendler – aktivira se ukoliko ne postoji hendler sa adekvatnim tipom izuzetka.

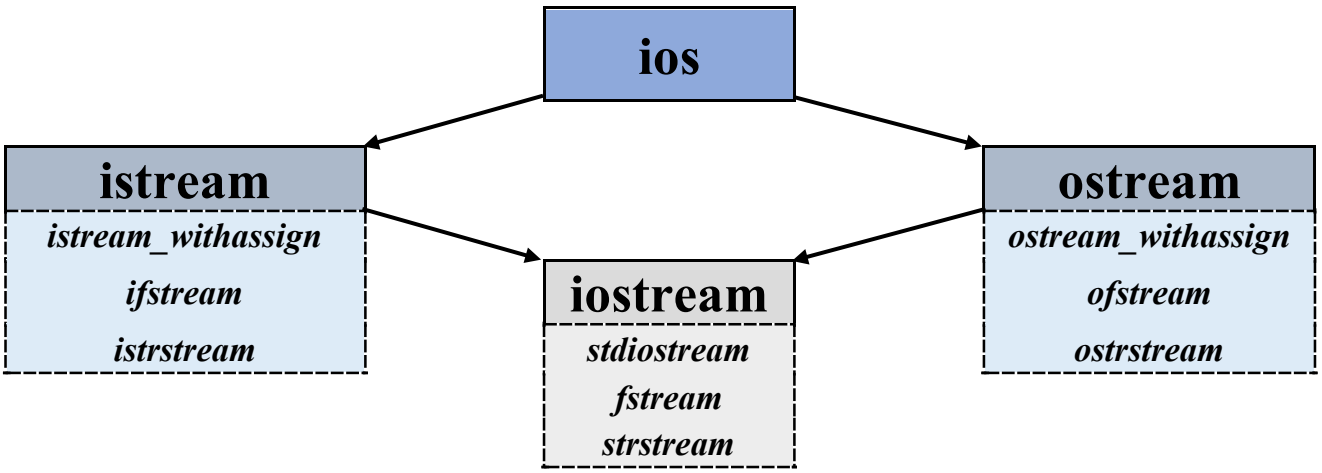
Nakon obrade izuzetka, nastavlja se sa daljim izvršenjem programa (ne vraća se u try blok).

- **Pravila pri navođenju hendlera:**
  - **Hendleri izvedenog tipa, idu pre hendlera osnovnog tipa.**
  - **Univerzalni hendler (…) ide na kraju.**
- **Šta podrazumeva predaja kontrole hendleru?**
  - Podrazumeva definitivno napuštanje bloka, što znači uništavanje svih lokalnih objekata u try bloku i njemu ugnježdenim try blokovima.
- **Kada hendler tipa B može prihvatiti izuzetak tipa D?**
  - Kada su B I D isti tipovi,
  - Kada je D izveden iz B i
  - Kada su B i D pokazivački tipovi i D može da se standardnom konverzijom konvertuje u B.

- **Kada se poziva void terminate()?**
  - ***Ako se za izuzetak ne pronađe hendler koji može da ga prihvati,***
  - ***Kada se detektuje poremećeni stack poziva i***
  - ***Kada se u destruktoru, u toku odmotavanja steka, postavi izuzetak.***
- **Koju funkciju podrazumevano poziva void terminate() i kako to promeniti?**
  - Podrazumevano poziva ***abort()*** funkciju, koja vraća kontrolu OS-u.
  - Ovo se može promeniti uz pomoć funkcije ***set_terminate(druga_funkcija).***
    - ***(*old_funtion) set_terminate (*new_function);***
    - ***void new_function ()*** (obevezno void i bez argumenata!) treba da poziva ***exit(int)*** funkciju, za povratak u OS.
    - Pokušaj povratka sa ***return _;*** poziva abort(), zato mora biti void.
- **Šta se dogodi, ako funkcija izazove izuzetak koga nema na listi izuzetaka koje baca?**
  - Poziva se funkcija ***void unexpected(),*** koja podrazumevano poziva ***void terminate().***
- **Kako promeniti podrazumevanu funkciju koju poziva void unexpected()?**
  - Uz pomoć funkcije ****old_function set_unexpected(*new_function)***
  - ***void new_function()*** mora biti definisana sa void, bez argumenata, (ne sme sadržati return liniju, jer se tada poziva abort()) I treba da vraća kontrolu OS-u pozivom exit(int).

TOKOVI / IO

- **Šta je tok ?**
  - ***Tok*** je logički koncept, koji predstavlja sekvencijalni ulaz ili izlaz znakova na neki uređaj ili datoteku.
- **Hijerarhija klasa za realizaciju I/O operacija.**



- **Od čega zavisi koliko se maksimalno može kreirati standardnih tokova ?**
  - Standardni tokovi su globalni statički objekti, ne mogu se kreirati od strane programera i ima ih 4 :
    - **Cin** – glavni (std) ulaz tipa istream (iz istream_withassign). Prestavlja tastatutu, ukoliko se drugačije ne precizira.
    - **Cout** – glavni (std) izlaz tipa ostream (iz ostream_withassign). Predstavlja ekran, koristi se za ispisivanje podataka koji čine rezultate izvršavanja programa.
    - **Cerr** – std izlaz za poruke tipa ostream. Koristi se za ispisivanje grešaka (error).
    - **Clog** – std izlaz za beleške tipa ostream. Koristi se za ispisivanje evidencije (log).
- **Koliko maksimalno cin objekata je moguće kreirati u main() funkciji?**
  - Nijedan, cin je globalni statički objekat.
- **Koja je namena klase strstream?**
  - ***Strstream služi za rad sa nizovima*** (eng. Strings), tj. Tokovima u operativnoj memoriji.
    - ***Istrstream*** – uzima podatke iz tokova
    - ***Ostrstream*** – upisuje podatke u tokove
    - ***Strstream*** – uzima i upisuje podatke iz/u tokove
- **Koliko maksimalno blanko znakova može sadržati ulazni podatak ako se za ulaz koristi operator >>?**
  - ***Operator>>*** koji se koristi za uzimanje podataka sa std ulaza (tastature) pamti znakove do prvog blanko znaka, ili novog reda (“ “, “\n”).
- **Kada je potrebno napraviti ulazni tok?**
  - Ukoliko se koristi cin objekat, onda nije potrebno pravljenje ulaznog toka.
  - Ulazni tok se kreira kada se koristi:
    - ***Tok iz datoteke (file stream)***

- ***Tok iz niza (strstream)***

- **Konstruktori ulaznih tokova datoteka.**
  - Postoji tri načina da se napravi ulazni tok za neku datoteku:
    - ***Korišćenjem konstruktora bez argumenata,***
    - ***Korišćenjem konstruktora sa argumentima (ime/lokacija datoteke, <odgovarajući parametri>),***
    - ***Navođenjem deskriptora datoteke.***
- **Konstruktor bez argumenata za ulazni tok datoteke.**
  - Korišćenjem konstruktora bez argumenata kreira se objekat klase ifstream, a zatim se poziva funkcija open, koja otvara navedenu datoteku:
    - Ifstream f;
    - f.open("imeDatoteke.txt", iosmod);
- **Konstruktor sa argumentima za ulazni tok datoteke.**
  - Ovaj konstruktor od argumenata sadrzi lokaciju odgovarajuće datoteke i modeFlag, koji može da se izostavi (podrazumeva se ios::in).
    - Ifstream f("imeDatoteke.txt",iosmode);
- **Konstruktor za ulazni tok navođenjem deskripcije.**
  - Navođenje deskriptora datoteke vrši se samo za već otvorene datoteke. Standard je:
    - Int desk=_open("datot.txt",dosmode);
    - Ifstream f(desk);
- **Vrednosti iosmode I dosmode parametara :**
  - **Ios :: app – upisivanje na kraj datoteke (append)**
  - **Ios :: ate – pozicioniranje na kraj datoteke posle otvaranja**
  - **Ios :: in – otvara ulaznu datoteku**
  - **Ios :: out – otvara izlaznu datoteku**
  - **Ios :: nocreate – otvara datoteku, ukoliko već postoji**
  - **Ios :: noreplace – otvara datoteku, ukoliko ne postoji**
  - **Ios :: trunc – otvara datoteku i briše stari sadržaj**
  - **Ios :: binary – binarna datoteka**
- **Koje je značenje vrednosti ios::trunc parametra dosmode ?**
  - Ios :: trunc – otvara datoteku i briše stari sadržaj
- **Šta radi funkcija seekg(int val)?**
  - Za objekte ifstream.
  - Podešava indeks od koga će se vršiti čitanje iz datoteke.
  - U ovom slučaju će se čitanje vršiti od indeksa val, odnosno od val+1. karaktera.
- **Šta radi funkcija seekp(int val)?**
  - Za objekte ofstream.
  - Podešava indeks od kog će se vršiti upisivanje u datoteku.
  - U ovom slučaju će se upisivanje vršiti od indeksa val, odnosno od val+1. karaktera.
- **Šta radi funkcija tellg()?**
  - Za objekte ifstream.
  - Vraća indeks do kog se stiglo upisom u fajl. (Pozicija kursora).
- **Šta radi funkcija tellp()?**

- Za objekte ofstream.
- Vraća indeks do kog se stiglo sa upisom u fajl. (Pozicija kursora).

- **Šta je namespace?**
  - Namena ***prostora imena*** (***namespace***) je grupisanje globalnih imena u velikim programskim sistemima u dosege. Svaki prostor imena čini zaseban doseg, tzv. ***Prostorni doseg*** (***namespace scope***). Ako se delovi programa stave u različite prostore imena, ne može doći do konflikta pri korišćenju istih imena u različitim prostorima.

  Primer:

```
namespace Alfa
{ int a = 55; }
namespace Beta
{ double a = 99.99; }
int p = Alfa::a;
double q = Beta::a;
```

- **Koliko maksimalno neimenovanih namespace-a može biti definisano u jednom fajlu?**
  - Maksimalno možemo imati jedan neimenovani (podrazumevani) namespace, zato je pametnije koristiti ***ime_namespace_a::član***, umesto direktive ***using namespace ime_namespace_a***.
- **Šta predstavljaju instance klase iterator ?**
  - Instance klase iterator predstavljaju pozicije elemenata u različitim kontejnerskim klasama, kojima su pridružene.
- **Kako se pristupa elementima kontejnerskih klasa definisanih u std?**
  - Na dva načina:
    1. ***Uz pomoć direktive using namespace std; koja nam omogućava da funkcije iz ovih klasa pozivamo samo imenom***
    2. ***Uz pomoć binarnog rezolucionog operatora { :: } kojim vezujemo ime funkcije sa bibliotekom (namespace-om) kome pripada. (ovo je sigurniji način i treba da se praktikuje, kada se radi sa više namespace-a).***