Predavanje 8_a

- Pri deklaraciji strukture iza ključne riječi „struct" se prvo navodi ime strukture, a zatim u vitičastim zagradama deklaracije individualnih elemenata od kojih se struktura sastoji.

```
struct Radnik
{ char ime[40];
  char odjeljenje[20];
  int platni_broj;
  double plata;
};
```

atribut

- Ključna riječ struct definira novi tip → strukturni.
- Svaki atribut unutar strukture ponaša se poput individualne promjenljive odgovar. tipa.
- Atributima pristupamo tako što navedemo ime strukture (promjenljive) zatim tačku i na kraju ime atributa → Radnik sekretar, službenik, portir

```
sekretar.plata = 1260;
cin.getline(portir.ime, sizeof portir.ime);
```

- Bilo koja slogovna promjenljiva može se čitava dodijeliti drugoj slogovnoj promjenljivi istog tipa → npr. službenik = portir;
- Podrazumijevano dejstvo ove dodjele je da se svi atributi iz promjenljive portir kopiraju u odgovarajuća polja promjenljive službenik.
- Inicijalizaciju polja (atributa) unutar strukture je najbolje izbjegavati.
- Ukoliko su polja strukture deklarisana specifikacijama „const" i „static" takva polja se moraju inicijalizirati odmah u deklaraciji strukture.
- Strukture se mogu prenositi kao parametri u funkcije i po vrijednosti i po referenci.
- Strukture se mogu vraćati kao rezultat iz fje.
- Kvalifikator „const" se upotrebljava (ispred parametra funkcije) iz dva razloga: da funkcija ne promijeni sadržaj parametra; na taj način je omogućeno da se kao stvarni parametar ne mora upotrijebiti promjenljiva, nego je

mogućе upotrijebiti bilo koji legalan izraz čiji je tip isti kao tip koji vraća funkcija
- Reference na konstantne objekte & mogu vezati na privremene objekte koji nastaju kao rezultat izračunavanja izraza.
- Nizovi (vektori) čiji je svaki element strukturnog tipa ⇒ Radnik radnici[100];
vector<Radnik> radnici(100);
- Svaki element ovakvog niza (vektora) predstavlja strukturu.
- Promjenljive strukturnog tipa se mogu kreirati dinamički pomoću operatora „new"
- *x.y = *(x.y); (*x).y = x->y
- -> - operator indirektnog pristupa.

* Predavanje 8.5 *

- Generičke strukture ⇒ strukture kod kojih neki, a možda i svi tipovi atributa nisu poznati

```
template <typename Tip1, typename Tip2>
struct UredjeniPar
{
    Tip1 prvi_clan;
    Tip2 drugi_clan;
};
```

- Ovdje nisu moguće automatske dedukcije tipova
- pair iz biblioteke utility ⇒ generička struktura praktična za postavljanje pojmova koji se mogu predstaviti kao uređeni par dvije anjedinosti
- make_pair
- Mape su tijesno vezane sa uređenim parovima i u mnogim aspektima se ponašaju poput skupova čiji su elementi uređeni parovi
- Mapama se mogu dodjeljivati inicijalizacione liste

- Razlika između mapa i skupova uređenih parova je što se kod mapa prvo polje pripadnih parova tretira kao ključno polje, dok se drugo polje tretira kao pridružena vrijednost.
- Članovima para se pristupa preko funkcija first i second
- U mapama ne smiju postojati dva para sa istim ključnim poljem.
- Mape zahtijevaju da tip ključa bude tip koji podržava poređenje pomoću operatora "<"
- Parametri generičkih struktura ne moraju biti samo imena tipova, već mogu biti i neke druge stvari, poput cjelobrojnih konstanti.
- Plitko kopiranje => prilikom kopiranja struktura koje sadrže pokazivače iz jedne u drugu se kopiraju samo pokazivači, ali ne i ono na šta oni pokazuju. Plitko kopiranje može nastati i prilikom prenosa po vrijednosti strukturnih promjenljivih kao parem. u funkcije kao i prilikom vraćanja struktura.

- Čvorovi → strukture koje sadrže polja koja su pokazivači na isti strukturni tip.

```
struct Cvor
{ int element;
  Cvor *veza;
};
```

- Fizički raspored čvorova u memoriji nije bitan, već je bitna logička veza između čvorova.
- Mana povezanih lista: nije moguće direktno pristupiti npr. petom čvoru, a da prethodno ne pročitamo prva četiri čvora.
- Do curenja memorije može doći kada koristimo pametne pokazivače pri kružnom referenciranju.
- Rješenje problema: kružne veze moraju biti ručno raskinute prije nego nestane posljednji pametni pokazivač koji pokazuje

na neki objekat unutar „začaranog kruga". Drugi način je uopće ne dozvoliti da dođe do uspostavljanja kružnih veza. Najjednostavniji način sprečavanja uspostavljanja kružnih veza je na neko fiksno mjesto unutar ciklusa umjesto pametnog pokazivača upotrijebiti obični pokazivač.

Predavanje 9_a

- Utoliko razmatramo funkcije kojoj se šalje argument kao parametar govorimo o proceduralnom programiranju.
- Utoliko razmatramo podatak nad kojim se primjenjuje fja govorimo o objektno orijentisanom programiranju.
- Osnovu OOP čine klase (razredi) koje predstavljaju objedinjenu cjelinu koja objedinjuje skupinu podataka, kao i postupke koji se mogu primjenjivati nad tim podacima.
- Klasu dobijamo tako što unutar strukture dodamo i prototipove funkcija koje se mogu primjenjivati nad tim podacima

```
struct Datum
{ int dan, mjesec, godina;
  void Postavi (int d, int m, int g);
};
```

* Predavanje 9_b *

- Statički atributi → atributi koje dijele svi primjerci iste klase. Deklarišu se uz „static" u privatnom dijelu klase, a prostor za njih se „rezerviše" van klase
- Statičke funkcije članice → ne znaju nad kojim su objektom pozvane; mogu pristupati statičkim atributima; mogu pozivati druge statičke funkcije članice.
- Prijateljske funkcije → imaju pravo pristupa privatnim atributima i metodama te klase.
- Prijateljska klasa: friend class B;
Ovo znači da sve metode klase B postaju prijatelji klase A.

* Predavanje 10_a *

- Konstruktori ⇒ definiraju skupinu akcija koje se automatski izvršavaju nad objektom ovog tipa trenutka kada dođe do njegovog stvaranja
- Konstruktori uvijek imaju isto ime kao i klasa kojoj pripadaju i nemaju povratnog tipa

```
class kompleksni
{ double re, im;
  public:
    kompleksni() { re=im=0; }
...
};
```

- Konstruktori se koriste za inicijalizaciju objekata, dok za njihovu uništavanje iznimno treba koristiti druge funkcije članice.
- Konstruktor sa parametrima koji imaju podrazumijevane vrijednosti:

```
kompleksni (double r=0, double i=0)
{ re=r; im=i; }
```

- Ukoliko klasa sadrži makar jedan privatni atribut ili/ili ima definirane konstruktore, tada se prilikom pokušaja inicijalizacije te klase pomoću inicijalizacione liste {} vrijednosti iz liste prosljeđuju u odgovarajuće parametre konstruktora, tako da će inicijalizacija biti uspješna ako i samo ako postoji konstruktor koji može da prihvati odgovarajući tip i broj vrijednosti koje se nalaze u inicijalizacionoj listi.

```
Kompleksni z1 = {1.5, 2};
Datum dat1 = {15, 5, 2013};
Datum dat2 = new Datum {15, 5, 2013};
```

- Konstruktor mora biti deklarisan u javnom dijelu klase.
- Nije preporučljivo deklarisati objekte koji imaju konstruktor u tijelu petlje.

- Pri eksplicitnom pozivu konstruktora, konstruktor klase se poziva kao obična funkcija, a ne kao funkcija članica (funkcija prvo kreira neki betimeni privremeni objekat odgovarajuće klase, zatim ga inicijalizira na uobičajen način i na kraju vraća kao rezultat tako kreiran i inicijaliziran objekat).
- Kad god je neki od atributa neke klase tipa neke druge klase koja ima podrazumijevani konstruktor (bez parametara), konstruktor te klase će automatski za svaki od takvih atributa istoristiti njihov podrazumijevani konstruktor da ih inicijalizira iako takve akcije ne trebamo nigdje specificirati unutar tijela konstruktora.
- Konstruktor kopije (kopirajući konstruktor) → konstruktor sa jednim (ili više) parametrom, koji prima kao parametar postojeći objekat

istog tipa (ili referencu na takav objekat)
Kompleksni (const Kompleksni &z)
{ re = z.re; im = z.im; }
- Podrazumijevani konstruktor kopije obavlja podrazumijevani postupak kopiranja koji se sastoji od kopiranja atributa iz jednog objekta u drugi.
- Konstruktor klase sa jednim parametrom se koristi za automatsku pretvorbu tipa iz tipa koji odgovara parametru konstruktora u tip koji odgovara klasi.
- U koliko iz bilo kojeg razloga želimo da zabranimo implicitno korištenje konstruktora sa jednim parametrom za automatsku pretvorbu tipova, tada ispred deklaracije konstruktora navodimo „explicit"

Predavanje 10_b

- U slučaju da klasa nema podrazumijevani konstruktor, deklaracije nizova čiji su elementi primjerci te klase nisu moguće.

* emplace_back → ekvivalent push_back-u samo što prosljeđuje konstruktoru svoje parametre, te je efikasnost veća
* emplace_front → push_front
* emplace → insert

- Konstruktor neke klase može koristiti konstruktore drugih klasa za inicijalizaciju svojih atributa
- Za tu svrhu, nakon zatvorene zagrade u deklaraciji parametara konstruktora stavlja se dvotačka iza koje slijedi konstruktorska inicijalizacijska lista

```
class Student
{ char ime_i_prezime[50];
  const int indeks;
  Datum datum_rodjenja;
public:
  Student (const char i_i_p[], int
indeks, int d, int m, int g):
datum_rodjenja(d, m, g)
{ strcpy (ime_i_prezime, i_i_p);
  Student::indeks=indeks;
}
...
};
```

- Atribut se mora inicijalizirati u konstrukt. inicij. listi kada atribut predstavlja referencu na neki drugi objekat.
- Kada konstruo konstr. inicij. liste svi atributi promenuti u listi se inicijalizuju onim redom kako su deklarisani unutar klase.

* Predavanje 11_a *

- Ukoliko postoje dvije metode istog imena i istih parametara od kojih je jedna označena kao konstantna, a druga nije, tada se konstantna verzija poziva samo ukoliko se primjeni nad konstantnim objektom.
- Destruktori predstavljaju skupinu akcija koje se automatski izvršavaju prilikom uništavanja objekta, tačnije kada objekat prestaje postojati.
- Zadatak destruktora je da oslobodi dodatne resurse koje je objekat zauzeo tokom svog života, recimo prilikom poziva konstruktora ili neke metode koja vrši zauzimanje dodatnih resursa.
- Destruktori ne mogu imati parametre, a ime im je isto kao ime klase u kojoj se nalaze, samo sa prefiksom „~".

- Destruktori se automatski pozivaju samo nad objektima koji su zaista stvoreni.
- Bacanje izuzetka iz konstruktora je najbolje izbjeći.
- Destruktore nikada ne treba pozivati eksplicitno kao obične funkcije članice.
- Sekvencijski konstruktor (konstruktor iz inicijalizacijske liste) prima objekat tipa "initializer_list" kao parametar.

```
Vektor Vd (initializer_list<double> lista);
...
```

- Ovaj konstruktor će biti pozvan kad god se za inicijalizaciju objekata upotrijebi inicijalizacijska lista čiji su svi elementi tipa "double" ili se mogu automatski konvertovati u tip "double".
- Svaka klasa koja posjeduje vlastiti destruktor (tj. onaj koji nije automatski generisan), obavezno mora posjedovati i vlastiti konstruktor kopije i vlastiti operator dodjele

- Konstruktor kopije → specijalan slučaj konstruktora sa jednim parametrom, čiji je formalni parametar referenca na konstantni objekat klase kojoj pripada.
- Kompajleri imaju pravo (ali ne i obavezu) da izbjegnu pozivanje konstruktora kopije kad god je to moguće → elizija konstruktora kopije.
- Pomjerajući konstruktor kopije → njegov parametar je r-vrijednosna referenca na objekat tipa klase u kojoj se definira, tako da on koristi samo kopiranje r-vrijednosti (r-vrijednost imaju samo privremeni objekti)
- Kopirajući operator dodjele → formalni parametar je referenca na konstantni objekat pripadne klase, tip rezultata je referenca na objekat pripadne klase.

* Predavanje 11_b *

- Bilo kojoj funkciji čiji je formalni parametar funkcija (pokazivač na funkciju) možemo kao stvarni parametar proslijediti običnu funkciju ili statičku funkciju članicu, ali ne i nestatičku funkciju članicu.
- Zabrana kopiranja primjeraka klase → Umjesto implementacije kopirajućeg konstruktora kopije pišemo "=delete"
- Zabrana dodjele → isto ↑
- Zabrane „rade" samo na automatskim konstruktorima kopije / operatorima dodjele
- Onim se onemogućava vraćanje primjeraka klase kao rezultata funkcije.