Dinamički životni vek

- Pažljivim projektovanjem programa mora se unapred odrediti odgovornost za uništavanje dinamičkih objekata koji se kreiraju. Dobro imenovanje funkcija i njihovo dokumentovanje pomaže u sprečavanju ovog problema. Na primer:

```
X* px = createAnX(...);
```

Ovaj naziv funkcije sugeriše pozivaocu da obrati pažnju na odgovornost za uništavanje objekta koji je napravljen u ovoj funkciji

- Postoje i razvojni alati koji nadziru izvršavanje programa i mogu da daju izveštaj o dianamičkim objektima koji su kreirani, a nisu uništeni tokom izvršavanja programa
- U nekim drugim programskim jezicima (npr. Java), dinamički objekat uništava se implicitno, kada poslednja referenca koja na objekat ukazuje prestane da živi (pošto se tada objektu svakako više ne može pristupiti, jer se objektima u tim jezicima ne može pristupiti drugačije nego preko referenci)
- Ovo implicitno brisanje naziva se "skupljanje đubreta" (*garabage collection*) i obavlja ga poseban deo izvršnog okruženja jezika (*garbage collector*) povremeno, po nahođenju izvršnog okruženja
- Ovakav pristup značajno smanjuje pojavu curenja memorije, ali ne može da je potpuno spreči, jer mogu postojati trajne reference koje ukazuju na objekat i tako ne dozvoljavaju njegovo brisanje, iako on zapravo nije neophodan; na primer, kada grupa objekata međusobno ciklično ukazuju referencama jedan na drugog
- Postoje i sofisticiranije metode otkrivanja ovakvih pojava, ali one ne mogu nikada biti potpuno delotvorne, pa problem curenja memorije i dalje ostaje kao pretnja. Osim toga, na sličan način mogu "curiti" bilo koji drugi resursi koji se dinamički alociraju (recimo niti), a ne uništavaju kada je potrebno

Dinamički životni vek

- Sličan pristup za smanjenje problema curenja memorije postoji i na jeziku C++, kao pouzdanije rešenje ovog problema kroz korišćenje tzv. *pametnih pokazivača* (*smart pointer*): pokazivača za koje se vodi evidencija o broju onih koji ukazuju na isti objekat i za koje je obezbeđeno implicitno uništavanje dinamičkog objekta kada poslednji od njih prestane da ukazuje na taj objekat
- U standarnoj biblioteci jezika C++ postoji više šablonskih klasa za pametne pokazivače:
  - *std::unique_ptr*: pametni pokazivač koji je jedini "vlasnik" svog objekta; kada ovakav pokazivač koji ukazuje na dati objekat prestane da ukazuje na taj objekat (zbog kraja životnog veka ili zato što je promenio vrednost dodelom druge vrednosti), i dinamički objekat na koga on ukazuje se implicitno briše
  - *std::shared_ptr*: pametni pokazivač koji je deljeni "vlasnik" svog objekta; kada poslednji pokazivač koji ukazuje na dati objekat prestane da ukazuje na taj objekat (zbog kraja životnog veka ili zato što je promenio vrednost dodelom druge vrednosti), i dinamički objekat na koga on ukazuje se implicitno briše
  - *std::weak_ptr*: pametni pokazivač koji je tzv. "slaba referenca" (*weak reference*) na objekat; objekat može biti uništen i ako na njega ukazuju samo slabi pokazivači, ali se preko ovog pokazivača može bezbedno pristupiti objektu, pri čemu se on tada mora konvertovati u *shared_ptr*; ovakvi pokazivači mogu se koristiti i za raskidanje cikličnih referenciranja objekata pomoću pametnih pokazivača
- Na primer:

```
{
  std::shared_ptr<X> p = new X;
  {
    std::shared_ptr<X> q = p;
    ...q->...  ...*p...
  }
}
```