Dinamički životni vek

- Tzv. *curenje memorije* (*memory leak*) je problem koji može nastupiti nekorektnim rukovanjem dinamičkim objektima, tako što se dinamički objekti repetitivno prave, ali se ne uništavaju, jer je programer zaboravio da propisno uništava dinamičke objekte koji više nisu potrebni (a program iznova pravi nove kada su potrebni)
- Nakon dužeg izvršavanja programa, slobodna memorija će biti iscrpljena, pa memorije za nove dinamičke objekte više neće biti, i naredna operacija *new* baciće izuzetak (ili vratiti *null* vrednost); nakon toga, program više neće radi kako se od njega očekuje
- Na primer, sledeći deo koda je banalan, ali i očigledan primer curenja memorije:

```
int i = *new int(0);
```

Problem: ovom objektu ne može se pristupiti, pa se on ne može ni obrisati

- Tipičan uzrok jeste nepažljivo dodeljena odgovornost za brisanje dinamički napravljenih objekata. Na primer, korisniku neke funckije koja pravi nov dinamički objekat i vraća pokazivač (ili referencu) na njega nije jasno da je odgovornost za brisanje tog objekta na njemu (a ne na onom ko ga je koristio), pa će zaboraviti da ga obriše:

```
extern X* getAnX (…);
X* px = getAnX(…);
```

Nije sasvim jasno da li pozivalac ima odgovornost za brisanje objekta na kog ukazuje vraćeni pokazivač

- Uzrok može biti i situacija u kojoj se objekat ne obriše zbog izuzetka:

```
void f () {
  int* p = new int(1);
  g();
  delete p;
}
```

Ako funkcija *g* baci izuzetak, dinamički objekat neće biti obrisan

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(…);
```

- 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