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...
  }
}
```

Za pametne pokazivače preklopljeni su operatori *, -> i =

Ovde, po prestanku života pametnog pokazivača *p*, i dinamički objekat klase *X* se uništava implicitno

Životni vek vezan za nit

- Varijable koje bi inače bile statičke po životnom veku (lokalne, u prostoru imena ili u oblasti klase) mogu biti deklarisane kao *thread_local*; tada imaju životni vek vezan za nit
- Ovakve varijable nastaju pri kreiranju svake nove niti i nestaju kada se nit završi
- Svaka nit ima svoju instancu ove varijable, nezavisnu od ostalih niti; svako obraćanje toj varijabli odnosi se na onu instancu koja pripada niti u čijem kontekstu se izvršava taj pristup. Na primer:

```
thread_local int i=0;
void f (int ii) {
  i = ii;
}

void tf (int id) {
  f(id);
  std::cout<<++i<<std::endl;
}

int main () {
  i = 10;
  std::thread t1(tf,1);
  std::thread t2(tf,2);
  std::thread t3(tf,3);

  std::cout<<i<<std::endl;

}
```