Operatori *new* i *delete*

- Analogno, dealokatorska funkcija *delete* može se preklopiti za neku klasu *T* kao statička funkcija članica te klase (čak i ako se ne deklariše kao *static*, uvek je implicitno statička):

```
void T::operator delete (void*)
```

```
void T::operator delete [] (void*)
```

- Ove funkcije mogu da se preklope u još nekim dostupnim oblicima (npr. onim kojim primaju veličinu prostora koji se dealocira)
- Ove funkcije ne treba da pozivaju destruktore eksplicitno; destruktor se uvek poziva kao korak u izvršavanju izraza *delete*
- Ove funkcije imaju zadatak da memorijski prostor na zadatoj adresi proglase slobodnim i ništa više
- Upravo zato i primaju pokazivač tipa *void** koji sadrži adresu prostora koji treba dealocirati (a ne pokazivač na objekat klase, jer je objekat već uništen)
- Ako klasa ima ovakvu funkciju, ona će biti pozivana u izvršavanju drugog koraka izraza *delete* kada se dealociraju objekti, odnosno nizovi objekata te klase

Operatori *new* i *delete*

- Jedna ideja za alokaciju prostora za objekte klase *X* koja nema problem fragmentacije, jer objekte smešta u niz slotova veličine tipa *X*, pri čemu slobodne slotove ulančava u listu (pa su operacije alokacije i dealokacije kompleksonsti *O*(1)):

```
template <class T, int size>
class Storage {
public:

  Storage () : head(slots) { slots[size-1].next = nullptr; }

  void* alloc () { Slot* p=head; if (p) head=p->next; return p?p->slot:nullptr; }
  void  free (void* addr) { head = new (addr) Slot(head); }

private:

  struct Slot {

    Slot () : next(this+1) {}
    Slot (Slot* nxt) : next(nxt) {}

    union {
      Slot* next;
      char slot[sizeof(T)];
    };

  };

  Slot* head;
  Slot slots[size];

};
```