Operatori *new* i *delete*

- Sada se ova pomoćna klasa za alokaciju može koristiti na sledeći način:

```
class X {
public:

  …

  void* operator new (size_t) {
    void* addr = storage.alloc();
    if (!addr) throw std::bad_alloc;
    else return addr;
  }
  void operator delete (void* addr) { storage.free(addr); }
private:
 static Storage<X,2000> storage;
};
Storage<X,2000> X::storage;
```

- Nedostatak ovog rešenja jeste to što se unapred, statički mora zadati veličina skladišta; potrebnu veličinu je često teško odrediti, jer zavisi od dinamičke prirode programa, a čak i ako se odredi, može biti premala (da ne može da zadovolji trenutne potrebe, iako slobodne memorije i dalje ima) ili prevelika (pa se memorija nepotrebno rezerviše i drži zauzetom za skladište, iako nije potrebna za objekte). Ova veličina može se zadati i dinamički, ali pri njegovoj inicijalizaciji (unaprediti ovo rešenje na taj način)
- Postoje i naprednija rešenja koja nemaju ovo ograničenje; zadatak: osmisliti i implementirati neko takvo rešenje

Operatori *new* i *delete*

- Evo još jedne ideje elegantnog rešenja čiji je način korišćenja skoro isti, a koje nema problem fragmentacije niti potrebe za dimenzionisanjem skladišta; proučiti njegovo delovanje:

```
template <class T>
class RecycleBin {
public:

  void* alloc () {
    Slot* p=head;
    if (p) head=p->next; else p = new Slot;
    return p;
  }

  void  free (void* addr) { head = new (addr) Slot(head); }

private:

  struct Slot {

    Slot (Slot* nxt=nullptr) : next(nxt) {}

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

  };

  Slot* head = nullptr;

};
```