banch

(Programozás alapjai 2 — házi feladat terv)

Péter Bertalan Zoltán
(QO7CU6)

2018. április 18.

Pontosított specifikáció

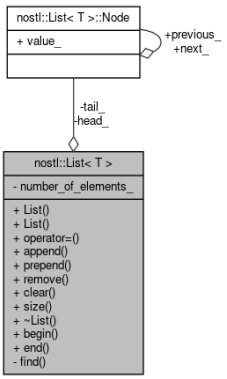
A *banch* valójában egy igen egyszerű szoftver. Mindössze arról szól, hogy van egy `Recipe` osztály, ezeknek a példányait kezeli a program. Ez az osztály egy heterogén tárolóban tárolja a különböző hozzávalókat (`Ingredient`), igazából egy általam implementált halmaz osztályt használ erre (`Set`). Elméletileg egy-egy recept tetszőleges mennyiségű hozzávalót tartalmazhat.

A felhasználónak van lehetősége `Recipe` példányokat létrehozni és ezeket „megtölteni" `Ingredient`-ekkel. Persze az `Ingredient` osztálynak vannak leszármazottai, hogy valóban heterogén kollekcióról beszélhessünk: például vannak alkoholos italok, amiknek van pluszban alkoholtartalom adattagjuk.

nostl

Mivel a feladatban meg volt tiltva, hogy STL tárolókat használjak, ezért kénytelen voltam megírni a sajátjaimat. Szükségem volt `String`-re és `Set`-re, viszont az utóbbihoz kellett írnom valamilyen okosabb tárolót, mint a tömb, ezért csináltam egy `List`-et is. A `Set`-et (és ezért a `List`-et is) úgy írtam meg, hogy tetszőleges típussal működjenek (*template*). Lentebb egy pár diagram (sajnos a Doxygen által generált diagramok nem írják a változók típusait; remélem, ez nem túl nagy probléma)

`nostl::List`

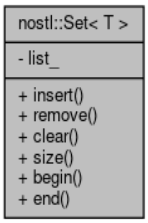


Beszúrok ide egy algoritmust az osztályból, hogy legyen valami:

```
template <typename T>
void List<T>::prepend(T const & val)
{
    Node * new_node = new Node;
    new_node->value_ = val;

    // setting new_node's pointers
    new_node->previous_ = this->head_;
    new_node->next_ = this->head_->next_;

    // setting neighbouring nodes' pointers
    new_node->previous_->next_ = new_node;
    new_node->next_->previous_ = new_node;

    // incrementing counter
    ++this->number_of_elements_;
}
```

(ez az algoritmus felelős azért, hogy a láncolt lista elejére beszúrhassunk valamit)

`nostl::Set`

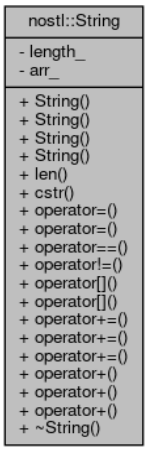


A `Set` osztály, mint látható, tartalmaz egy `List` adattagot, és tulajdonképpen semmit sem csinál, csak egy „wrapper” akörül, annyi bónusszal, hogy nem enged egy már létező elemet újra a halmazba tenni.

Tehát voltaképpen ez a lényege:

```
template <typename T>
void Set<T>::insert(T const & val)
{
    // make sure list/set doesn't contain item yet
    if (this->size() != 0)
    {
        for (typename List<T>::Iterator i = this->list_.begin();
                i != this->list_.end(); ++i)
        {
            if (*i == val)
            {
                return;
            }
        }
    }

    this->list_.append(val);
}
```

`nostl::String`

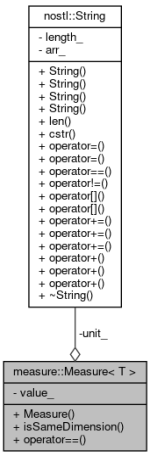


Ezt az osztályt nem szeretném részletezni, mert tulajdonképpen ugyanaz, amit már laboron is megírtunk.

---

measure (`measure::Measure`)

Ez egy kis mini-header fájl tulajdonképpen, ami csak arra hivatott, hogy értelmesen lehessen kezelni a programban a mennyiségeket.




```
1  template <typename T>
2  class Measure {
3
4      static_assert(std::is_arithmetic<T>::value, "Type must be numeric!"
       );
5
6  public:
7      Measure(nostl::String const unit, T const & value)
8      {
9          this->unit_ = unit;
```

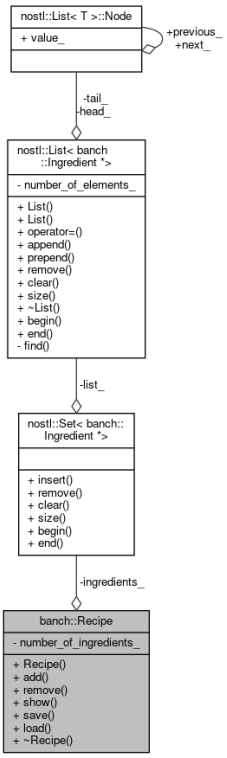
```
10          this->value_ = value;
11      }
12
13      bool isSameDimension(Measure const & comparison) const
14      {
15          return (this->unit_ == comparison.unit_);
16      }
17
18      bool operator==(Measure const & rhs) const
19      {
20          return (isSameDimension(rhs) && (this->value_ == rhs.value_));
21      }
22
23      template <typename F>
24      friend std::ostream & operator<<(std::ostream &, Measure<F> const);
25
26      // TODO more operators may be needed
27
28
29  private:
30      nostl::String unit_;
31      T value_;
32  }; // class Measure
```

---

banch

Végülis itt van a lényeg, itt van a fő projekt. Irónikusan errő tudok a legkevesebbet mondani, mert a nagy munka az STL tárolók újraírása volt, maga a recept nyilvántartó program megírása már nem egy olyan bonyolult dolog.

Így néz ki tehát a `Recipe` osztály diagramja dicső valójában (egyelőre):



hogy a diagramon miért nem látszik, hogy az osztálynak van egy `ingredients_` *adattagja, amiben a hozzávalók vannak, nem tudom …*

Tesztelés

A szoftvert modulokra bontottam és a Google Test keretrendszerrel tesztelem folymatosan. Nagyjából így néz ki egy ilyen teszt:

```
TEST(ListTest, Removal)
{
    // create new List with a few elements like {3, 30, 3, 30}
    nostl::List<int> foo;
    foo.append(3);
    foo.append(30);
    foo.append(3);
    foo.append(30);
    EXPECT_EQ(4, foo.size());

    // remove an element
    foo.remove(30); // --> {3, 3, 30}
    nostl::List<int>::Iterator i = foo.begin();
    EXPECT_EQ(3, *(i++));
    EXPECT_EQ(3, *(i++));
    EXPECT_EQ(30, *i);
    EXPECT_EQ(3, foo.size());

    // remove two more just for fun
    foo.remove(30); // --> {3, 3}
```

```
21      EXPECT_EQ(2, foo.size()); // FIXME this fails
22      foo.remove(3); // --> {3}
23      i = foo.begin();
24      EXPECT_EQ(3, *i);
25      EXPECT_EQ(1, foo.size());
26  }
```

További megjegyzés

A `Recipe` osztály tagfüggvényei még megírásra várnak, de látszik, miket tud majd: hozzávalót bevenni, kivenni, azokat listázni, illetve magát fájlba írni. Ezek voltak az elvárások a specifikációban.

Megjegyzem, hogy mindazon felül, ami ebben a dokumentumban szerepel, még szükség lesz egyéb osztályokra is valószínűleg. De szerintem ez a terv leírja a program lényegét, a terv célját teljesíti.