Итератор

доц. д-р Нора Ангелова

Итератор

- Поведенчески шаблон за дизайн, който предоставя начин за последователен достъп до елементите на дадена структура, без да е нужна информация за вътрешното представяне на структурата
- Представлява обект, който може да сочи елемент в даден контейнер

Итератор

```
template <typename T>
class LListIterator;
// дефиниция на клас LList

// всички операции са O(1)
template <typename T>
class LListIterator {
private:
  typedef ListElement<T> LE;
  LE *ptr;
public:

  typedef LListIterator<T> I;
  friend class LList<T>;

  // конструктор по указател
  LListIterator(LE* _ptr = nullptr) : ptr(_ptr) {}

  // няма нужда от голяма четворка!
```

Итератор

```
// следваща позиция
I next() const {
  // считаме, че итераторът е валиден
  // if (!valid())
  //  return *this;

  return I(ptr->next);
}

// предишна позиция
I prev() const;

// достъп до елемент с право на промяна
T& get() const {
  // допускаме, че итераторът е валиден
  // if (!valid())
  //  return error;
  return ptr->data;
}
```

Итератор

```
// достъп до елемент без право на промяна
T const& getConst() const;

// проверка за валидност
bool valid() const {
  return ptr != nullptr;
}

// сравнение на два итератора
bool operator==(I const& it) const {
  return ptr == it.ptr;
}

bool operator!=(I const& it) const {
  return !(*this == it);
}
```

Итератор

```
// допълнения

// *it <-> it.get()
T& operator*() const {
  return get();
}

// it++ <-> it = it.next(), връща старата стойност на it
I operator++(int) {
  I prev = *this;
  ++(*this);
  return prev;
}

// ++it <-> it = it.next(), връща новата стойност на it
I& operator++() {
  // ptr = ptr->next;
  // return *this;
  return *this = next();
}
```

Итератор

```
  // it <-> it.valid()
  operator bool() const {
    return valid();
  }
};
```

Итератор

```
template <typename T>
class LList {
public:
  typedef LListIterator<T> I;

private:

  typedef ListElement<T> LE;

  LE *front, *back;

  // O(n) по време, O(1) по памет
  void copy(LList<T> const& l) {
    for(I it = l.front; it; ++it)
      insertToEnd(*it);
  }

// …
};
```

Край