Декларативно програмиране в процедурните езици

Уводна лекция

Владислав Ненчев

Софийски Университет “Св. Климент Охридски”
Факултет по Математика и Информатика
Катедра по Математическа Логика и Приложенията ѝ

14 Октомври 2016

Валидация на входен текст I

```
bool ValidateProcedural(string input)
{
  if(input.Length == 0)
    return false;
  if(input[0] == '0' && input.Length > 1)
    return false;

  for(int i =0; i < input.Length; i++)
  {
    if(((int) input[i] < 48) || ((int) input[i] > 57))
      return false;
  }

  return true;
}
```

Валидация на входен текст I

```
bool ValidateProcedural(string input)
{
  if(input.Length == 0)
    return false;
  if(input[0] == '0' && input.Length > 1)
    return false;

  for(int i =0; i < input.Length; i++)
  {
    if(((int) input[i] < 48) || ((int) input[i] > 57))
      return false;
  }

  return true;
}
```

```
bool ValidateRegex(string input) =>
  Regex.IsMatch(input, @"^0\$|^[1-9][0-9]*$");
```

Валидация на входен текст II

```
static bool CheckSymbol(char symbol)
{
  return
    ((int) symbol < 65 || (int) symbol > 90) &&
    ((int) symbol < 97 || (int) symbol > 122) &&
    ((int) symbol < 48 || (int) symbol > 57) &&
    symbol != '-' && symbol != '_';
}

bool CheckTokens(String[] tokens, int minCount = 1)
{
  if(tokens.Length < minCount) return true;
  foreach(var token in tokens)
  {
    if(token.Length == 0) return true;
    for(int i =0; i < token.Length; i++)
      if(CheckSymbol(token[i])) return true;
  }
  return false;
}
```

Валидация на входен текст II (продължение)

```
bool ValidateProcedural(string input)
{
  if(input.Length == 0) return false;

  String[] parts = input.Split('@');
  if(parts.Length != 2) return false;

  if(CheckTokens(parts[0].Split('.')))
    return false;
  if(CheckTokens(parts[1].Split('.'), 2))
    return false;

  return true;
}
```

Валидация на входен текст II (продължение)

```
bool ValidateProcedural(string input)
{
  if(input.Length == 0) return false;

  String[] parts = input.Split('@');
  if(parts.Length != 2) return false;

  if(CheckTokens(parts[0].Split('.')))
    return false;
  if(CheckTokens(parts[1].Split('.'), 2))
    return false;

  return true;
}
```

```
bool ValidateRegex(string input) =>
  Regex.IsMatch(input,
    @"^[_\-\w]+(\.[_\-\w]+)*@[_\-\w]+(\.[_\-\w]+)+$");
```

Разпознаване на свойство на граф I

```
bool GraphPropertyProcedural()
{
  foreach(var x in vertices)
    foreach(var y in vertices)
      foreach(var z in vertices)
        if(edges.Contains(Edge(x, y)) &&
            edges.Contains(Edge(y, z)) &&
            !edges.Contains(Edge(x, z))) return false;
  return true;
}
```

Разпознаване на свойство на граф I

```
bool GraphPropertyProcedural()
{
  foreach(var x in vertices)
    foreach(var y in vertices)
      foreach(var z in vertices)
        if(edges.Contains(Edge(x, y)) &&
            edges.Contains(Edge(y, z)) &&
            !edges.Contains(Edge(x, z))) return false;
  return true;
}
```

```
bool GraphPropertyLogicStyle() =>
  ! vertices.Any(x => vertices.Any(y =>
    vertices.Any(z =>
      edges.Contains(Edge(x, y)) &&
          edges.Contains(Edge(y, z)) &&
          !edges.Contains(Edge(x, z)))));
```

Разпознаване на свойство на граф II

```
bool GraphPropertyProcedural()
{
  foreach(var x in vertices)
  {
    bool found = false;

    foreach(var y in vertices)
    {
      if(AreConnected(x, y))
      {
        bool isYCentral = true;
        foreach(var z in vertices)
        {
          if(!edges.Contains(Edge(y, z)))
          {
            isYCentral = false;
            break;
          }
        }
```

Разпознаване на свойство на граф II (продължение)

```
            if(isYCentral)
            {
               found = true;
               break;
            }
         }
      }

      if(!found) return false;
   }

   return true;
}
```

Разпознаване на свойство на граф II (продължение)

```
            if(isYCentral)
            {
              found = true;
              break;
            }
          }
        }

        if(!found) return false;
      }

      return true;
}
```

```
bool AllAccessCentralVertex() =>
  vertices.All(x => vertices.Any(y =>
    AreConnected(x, y) &&
    vertices.All(z => edges.Contains(Edge(y, z)))));
```

Йерархия на субекти

```
enum PartyType  { Person, Company }
abstract class Party
{
  public Integer PartyId() {  ... }
  abstract public PartyType Type();
  public String PartyName() { ... }
  public String PartyDescription()  { ... }
}

class Person extends Party
{
  public PartyType Type()
    { return PartyType.Person;  }
}

class Company extends Party
{
  public PartyType Type()
    { return PartyType.Company; }
}
```

Извличане на данни по идентификатори

```
String ReplaceIdsProcedural(String ids,
  ArrayList<Party> parties)
{
  ArrayList<Person> persons = new ArrayList<Person>();
  String[] parts = ids.split(",");
  for(String part : parts)
  {
    Integer id = Integer.parseInt(part);
    for(Party party : parties)
    {
      if(party.PartyId() == id &&
        party.Type() == PartyType.Person)
      {
        persons.add((Person) party);
        break;
      }
    }
  }
```

Извличане на данни по идентификатори (продължение)

```
  Collections.sort(persons, new PartyComparator());

  StringBuilder stringBuilder = new StringBuilder();
  boolean first = true;
  for(Person person : persons)
  {
    if(!first)  stringBuilder.append("\n");
    stringBuilder.append(person.PartyName() + " - " +
        person.PartyDescription());
    first = false;
  }

  return stringBuilder.toString();
}
```

Извличане на данни по идентификатори (продължение)

```
String ReplaceIdsPipeline(String ids,
  ArrayList<Party> parties)
{
  return
    Stream.of(ids.split(",")).
    map(part -> Integer.parseInt(part)).
    map(id -> parties.stream().filter(party ->
        party.PartyId() == id).findFirst().get()).
    filter(party -> party.Type() == PartyType.Person).
    sorted(new PartyComparator()).
    map(person -> person.PartyName() + " - " +
        person.PartyDescription()).
    collect(Collectors.joining("\n"));
}
```

Минимални прости числа

```
def IsPrimeProcedural(number):
  if number < 2: return False
  for lesser in range(2, number - 1):
    if number % lesser == 0: return False
  return True

def GetNextPrime(start):
  while not IsPrimeProcedural(start): start += 1
  return start

def VerifySumOf3Cubes(prime):
  for lesser1 in range(prime - 1):
    if IsPrimeProcedural(lesser1):
      for lesser2 in range(prime - 1):
        if IsPrimeProcedural(lesser2):
          for lesser3 in range(prime - 1):
            if IsPrimeProcedural(lesser3):
              if prime == math.pow(lesser1, 3) +
                  math.pow(lesser2, 3) +
                  math.pow(lesser3, 3): return True
  return False
```

Минимални прости числа (продължение)

```
minimalSumOf3Cubes = 2
while not VerifySumOf3Cubes(minimalSumOf3Cubes):
  minimalSumOf3Cubes =
      GetNextPrime(minimalSumOf3Cubes + 1)
```

Минимални прости числа (продължение)

```
minimalSumOf3Cubes = 2
while not VerifySumOf3Cubes(minimalSumOf3Cubes):
  minimalSumOf3Cubes =
      GetNextPrime(minimalSumOf3Cubes + 1)
```

```
def NaturalNumbers():
  number = 0
  while True:
    yield number
    number += 1

def IsPrimeDeclarative(number):
  return number > 1 and
    all(not number % lesser == 0 for lesser
      in range(2, number - 1))

def PrimeNumbers():
  return (number for number in NaturalNumbers()
    if IsPrimeDeclarative(number))
```

Минимални прости числа (продължение)

```
def First(iterator, predicate):
  return next(element for element in iterator()
    if predicate(element))

minimalSumOf3Cubes =
  First(PrimeNumbers,
    lambda prime:
      any(prime == math.pow(lesser1, 3) +
          math.pow(lesser2, 3) + math.pow(lesser3, 3)
      for lesser1 in range(prime - 1)
        if IsPrimeDeclarative(lesser1)
      for lesser2 in range(prime - 1)
        if IsPrimeDeclarative(lesser2)
      for lesser3 in range(prime - 1)
        if IsPrimeDeclarative(lesser3)))
```

Декларативно програмиране: Защо?

Декларативно програмиране?!

За какво ни трябва това???

Декларативно програмиране: Какво?

Максимално съответствие между описанието (декларацията) на решението на задачата и неговата програмна реализация:

Декларативно програмиране: Какво?

Максимално съответствие между описанието (декларацията) на решението на задачата и неговата програмна реализация:

"For all vertices x there exists a vertex y such that x and y are connected and for every vertex z there is an edge from y to z."

Декларативно програмиране: Какво?

Максимално съответствие между описанието (декларацията) на решението на задачата и неговата програмна реализация:

"For all vertices x there exists a vertex y such that x and y are connected and for every vertex z there is an edge from y to z."

$$\forall x \exists y (\mathit{AreConnected}(x, y) \,\&\, \forall z (\langle y, z \rangle \in \mathit{edges}))$$

Декларативно програмиране: Какво?

Максимално съответствие между описанието (декларацията) на решението на задачата и неговата програмна реализация:

"For all vertices x there exists a vertex y such that x and y are connected and for every vertex z there is an edge from y to z."

$$\forall x \exists y (AreConnected(x, y) \,\&\, \forall z (\langle y, z \rangle \in edges))$$

```
bool AllAccessCentralVertex() =>
  vertices.All(x => vertices.Any(y =>
    AreConnected(x, y) &&
    vertices.All(z => edges.Contains(Edge(y, z)))));
```

Декларативно програмиране: Как?

- Използване на регулярни изрази.

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).
- Минимизиране на разклоненията в логиката (в най-добрия случай - писане на код без условни оператори).

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).
- Минимизиране на разклоненията в логиката (в най-добрия случай - писане на код без условни оператори).
- Използване на ламбда изрази.

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).
- Минимизиране на разклоненията в логиката (в най-добрия случай - писане на код без условни оператори).
- Използване на ламбда изрази.
- Използване на техниката на поточната линия.

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).
- Минимизиране на разклоненията в логиката (в най-добрия случай - писане на код без условни оператори).
- Използване на ламбда изрази.
- Използване на техниката на поточната линия.
- Използване на формули.

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).
- Минимизиране на разклоненията в логиката (в най-добрия случай - писане на код без условни оператори).
- Използване на ламбда изрази.
- Използване на техниката на поточната линия.
- Използване на формули.
- “Математическо” програмиране (преобразувания на формули, булеви функции, зависимости между множества; формулиране на решения в дадена област; познания по изчислимост и разрешимост ...).

Декларативно програмиране: Как?

- Използване на регулярни изрази.
- Използване на операции върху множества и списъци (минимално използване на цикли).
- Минимизиране на разклоненията в логиката (в най-добрия случай - писане на код без условни оператори).
- Използване на ламбда изрази.
- Използване на техниката на поточната линия.
- Използване на формули.
- “Математическо” програмиране (преобразувания на формули, булеви функции, зависимости между множества; формулиране на решения в дадена област; познания по изчислимост и разрешимост ...).
- Ефективност!!!