

FUNKCIONALNO PROGRAMIRANJE

Funkcionalno programiranje

- Programiranje se zasniva na
 - Definisanju funkcije
 - pridruživanje imenu f-je izraza kojim se izračunava vrednost f-je
 - Primeni funkcije nad argumentima
- Program je niz definicija i poziva funkcija

λ račun – Ideja funkcionalnog programiranja

- λ izraz u matematici je parametrizovani izraz (ili neimenovana funkcija) oblika:

λ (parameters) expression

- Primer λ izraza:

$$\lambda (x) x * x * x$$

- Primena λ izraza na određenim stvarnim parametrima znači zameniti parametre stvarnim i izračunati vrednost izraza:

$$(\lambda (x) x * x * x)(5) \rightarrow 125$$

Osobine funkcionalnog programiranja

- Ne postoji eksplicitno zadavanje redosleda izračunavanja
 - program definiše samo izraz koji je rešenje nekog problema
- Nepostojanje naredbi
 - umesto njih postoje izrazi
 - Uslovni izrazi se koriste umesto naredbi grananja
 - Rekurzivni pozivi se koriste umesto programskih petlji
- Nepostojanje sporednih efekata
 - Vrednost izraza za iste vrednosti ulaznih parametara je uvek ista bez obzira u kojoj se okolini izračunava
- Nepromenljivost podataka
- Promenljive u programu nisu neophodne
 - Potpuno funkcionalni jezici ne podržavaju rad sa promenljivama kao memorijskim lokacijama gde se čuva neka vrednost

Osobine funkcionalnog programiranja

- Funkcije višeg reda
 - Funkcije čiji su parametri i/ili rezultati druge funkcije
- Slaganje (kompozicija) funkcija
 - U matematici kompozicija funkcija je definisana na sledeći način:
$$(f \circ g)(x) = f(g(x))$$
 - U funkcionalnom programiranju se programi grade kao kompozicije funkcija.

Elementi funkcionalnog programskog jezika

- Strukture podataka
 - Atomi
 - Liste
- Skup elementarnih (ugrađenih) funkcija
- Format za definisanje novih funkcija
- Format za primenu (poziv) funkcije

Primena funkcionalne programske paradigme

- Funkcionalni programski jezici:
 - Lisp i njegovi dijalekti:
 - Pure (original) Lisp
 - Interlisp, MacLisp, Emacs Lisp
 - Common Lisp
 - Scheme
 - ML
 - Miranda
 - Haskell
 - FP
- Programski jezici koji podržavaju funkcionalnu paradigmu:
 - Scala,
 - Ruby,
 - JavaScript,
 - Python...

ELEMENTI FUNKCIONALNIH JEZIKA KROZ PROGRAMSKI JEZIK LISP

Strukture podataka

- Atomi – elementarini podaci mogu biti:
 - Numerički - number
 - Stringovi (nizovi karaktera između dvostrukih navodnika)
 - Logički (**t** – true, **NIL** – false)
- Strukturni tipovi:
 - Par – dva podatka elementarnog ili strukturnog tipa napisan između malih zagrada
 - Lista – tri ili više podataka elementarnog ili strukturnog tipa napisan između malih zagrada
 - Liste se smatraju glavnim tipom podataka, otuda i ime jezila **LISt Processing**
 - Primeri listi:
 - (5 3 4),
 - (1 (1 2) 2),
 - (A “A” (A “A”))

Poziv funkcije

- Poziv funkcije je lista u kojoj je prvi element ime funkcije, a ostali parametri)
- Primer poziva funkcije:
`(f a b c)`

Aritmetičke ugrađene funkcije

- Prefiksni zapis aritmetičkih uzraza:

`(+ arg1 arg2 ... argn)`

`(- arg1 arg2 ... argn)`

`(* arg1 arg2 ... argn)`

`(/ arg1 arg2 ... argn)`

`(mod arg1 arg2)`

`(sin arg1)`

`...`

Relacione ugrađene funkcije

(op_code arg1 arg2)

op_code $\in \{<, <=, >, >=, eq, equal\}$

(eq arg1 arg2) – tačno ako oba argumenta ukazuju na isti podatak u memoriji

(equal arg1 arg2) – tačno ako oba argumenta imaju istu vrednost

(null arg) – tačno ako je argument prazna lista

(atom arg) – tačno ako je argument atom

(number arg) – tačno ako je argument numeričkog tipa

Logičke ugrađene funkcije

`(or arg1 arg2 ... argn)`

`(and arg1 arg2 ... argn)`

`(not arg1)`

Ugrađene funkcije za rad sa stringovima

- Poređenje:

`(op_code str1 str2)`

`op_code` ∈ {string-equal, string-not-equal, string-lessp, string-not-lessp, string-greaterp, string-not-greaterp}

- Konverzija malih u velika slova i obrnuto:

`(op_code str)`

`op_code` ∈ {string-upcase, string-downcase, string-capitalize}

- Dužina stringa:

`(length str)`

Ugrađene funkcije za rad sa stringovima

- Nadovezivanje stringova:
`(concatenate str1 str2 ... strn)`
- Izdvajanje podstringa:
`(subseq str start_position end_position)`
`(subseq str start_position)`
- Izdvajanje karaktera:
`(char str position)`
- ...

Ugrađene funkcije za rad sa parovima i listama

- Kreiranje para/liste:
`(cons arg1 arg2)`
`(list arg1 arg2 ... argn)`
- Izdvajanje prvog elementa para/liste:
`(car arg)`
- Izdvajanje drugog elementa para ili podliste bez prvog elementa:
`(cdr arg)`

Ugrađene funkcije za rad sa listama

- Nadovezivanje listi:
(append arg1 arg2 ... argn)
- Izdvajanje podliste koja sadrži samo poslednji element:
(last arg)
- Ispitivanje da li lista (arg2) sadrži dati element (arg1):
(member arg1 arg2)
- Kreiranje liste sa obrnutim redosledom elemenata:
(reverse arg)

Ugrađene funkcije višeg reda

- mapping funkcije:

- `(mapcar fn arg1 arg2 ... argn)` – primenjuje zadatu funkciju nad odgovarajućim elementima listi *arg1-argn* vraća listu rezultata (ukoliko liste *arg1-argn* nisu iste dužine, dužina rezultujuće liste odgovara dužini najkraće od njih)

- Primer:

`(mapcar '+ '(1 2 3) '(4 5 6) '(7 8)) → (12, 15)`

- `(maplist fn arg1 arg2 ... argn)` – primenjuje zadatu funkciju nad odgovarajućim elementima listi *arg1-argn*, zatim nad njihovim podlistama *cdr(arg1)-cdr(argn)*, pa na podlistama *cdr(cdr(arg1))-cdr(cdr(argn))*, ... Rezultat je lista listi.

- ...

- reduce funkcija:

- `(reduce fn list)` – primenjuje zadatu funkciju nad prva dva elementa liste, zatim nad rezultatom i sledećim elementom,...

- Primer:

- `(reduce '+ '(1 2 3)) → 6`

Uslovni izrazi

C naredba	Lisp izraz
<code>if (cond-expr) s1 else s1</code>	<code>(if (cond-expr) expr1 expr2)</code>
<code>if (cond-expr1) s1 else if (cond-expr2) s2 ... else if (cond-exprN) sN else sN+1</code>	<code>(cond (cond-expr1 expr1) (cond-expr2 expr2) ... (cond-exprN exprN) (t exprN+1))</code>

Definisanje sopstvanih funkcija u lispu

`(defun name (parameters) expression)`

- Primer:

```
(defun min (a b) (if (< a b) a b))
```

Primer realizacije iteracije

C

```
//faktorijel  
for(int i=2, fact=1; i<n; i++)  
    fact *= i;
```

```
//suma elemenata niza  
for(int i=0, s=0; i<N; i++)  
    s += a[i];
```

Lisp

```
(defun fact n  
  (if (<= n 1) 1 (* n (fact(- n 1)))))
```

```
(defun sum l  
  (if (equal (list-length l) 1)  
      (car list)  
      (+ (car list) (sum (cdr list))))))
```

Elementi funkcionalnog programiranja u JavaScript-u

JavaScript

- Skript jezik nastao za programiranje klijentske strane web aplikacija.
- Danas se ravnopravno koristi i za kreiranje klijentske i za kreiranje serverske strane
- U osnovi objektno-orijentisani jezik
- Podržava i funkcionalno programiranje

Pojam „čistih“ funkcija

- Funkcija ne menja vrednosti svojih parametara
- Vrednost funkcije zavisi samo od vrednosti svojih parametara (sredina u kojoj se izvršava ne utiče na vrednost funkcije)
- Funkcija ne može da menja ništa u svojoj okolini

Funkcije u JavaScriptu

- Definicija:

```
function name (parameters) { body }
```

- Primer:

```
function Product (a, b) { return a * b; }
```

- Funkcija može biti dodeljena promenljivoj:

- ES5

```
var x = function (a, b) { return a * b; }
```

- ES6

```
const x = (a, b) => a * b;
```

- Funkcija može biti parametar druge funkcije
- Funkcija može biti rezultat druge funkcije

Funkcije višeg reda i JavaScript-u

- `Array.map(fn)` – Zadatu funkciju primenjuje nad svakim elementom niza.
- Parametri funkcije *fn*:
 - Tekuci clan – obavezno
 - Tekuci indeks – opciono
 - Niz – opciono
- Primer:
 - ES5:

```
var niz = [1, 2, 3];  
var noviNiz = niz.map( function(clan) { return 2*clan; } );
```
 - ES6:

```
var niz = [1, 2, 3];  
var noviNiz = niz.map( x => 2*x );
```

Funkcije višeg reda i JavaScript-u

- `Array.filter(fn)` – Kreira novi niz od elemenata polaznog niza koji zadovoljavaju zadatu test funkciju.
- Parametri funkcije *fn* isti kao kod *map* funkcije.
- Primer:
 - ES6:

```
var niz = [1, 2, 3];  
var noviNiz = niz.filter( x => x % 2 == 1 );
```

Funkcije višeg reda i JavaScript-u

- `Array.reduce(fn)` – Zadatu funkciju primenjuje nad svakim elementom niza i kreira jednu rezultujuću vrednost.
- Parametri funkcije `fn`:
 - Akumulator – inicijalna vrednost rezultata - obavezno
 - Tekuci element – obavezno
 - Tekuci indeks – opciono
 - Niz – opciono
- Primer:
 - ES5:

```
var niz = [1, 2, 3];  
var sum = niz.reduce( (acc, x) => acc + x );
```

Kompozicija funkcija u JavaScriptu

- Kompozicija $f(g(x))$ se realizuje na sledeći način:
`x.g().f()`