# PARADIGME DE PROGRAMARE

## Curs 7

Transparență referențială. Legare statică / dinamică.

# Transparență referențială – Cuprins

- Efecte laterale
- Transparență referențială

# Efecte laterale

## Efecte laterale ale unei funcții

- **Efectul principal al oricărei funcții este să întoarcă o valoare**
- **Efecte laterale = alte efecte** asupra stării programului (ex: modificarea unor variabile globale) sau asupra „lumii de afară” (ex: scrierea în fișier)

## Funcție pură

- **Aplicată pe aceleași argumente, întoarce mereu aceeași valoare**
- **Nu are efecte laterale**

## Exemplu (C++)

```
i = 7;
```

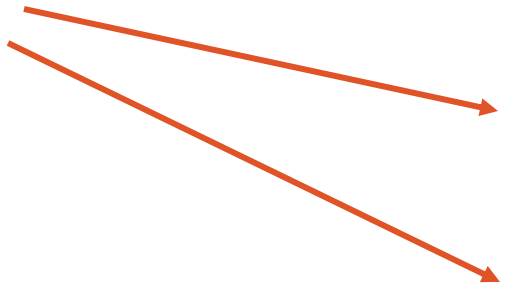
- Expresia întoarce valoarea 7
- Efect lateral: setează variabila i la valoarea 7

# Efecte laterale

Consecințe

- Contează strategia de evaluare

?



```
1.    int x = 0;
2.    int f() {
3.       x = 15;
4.       return x;
5.    }
6.
7.    int main() {
8.       int i = 0;
9.       int a = i-- + ++i;
10.      int b = ++i + i--;
11.      cout << a << " " << b;
12.
13.      int res = x + f();
14.      cout << " " << res << "\n";
```

# Efecte laterale

Consecințe

- Contează strategia de evaluare

Comportament bizar al compilatorului care nu are operația definită

Afișează **0 1** → Contează ordinea de evaluare a argumentelor adunării

Afișează **30** → call by reference
Cu call by value și evaluare stânga-dreapta s-ar afișa **15**

```
1.    int x = 0;
2.    int f() {
3.       x = 15;
4.       return x;
5.    }
6.
7.    int main() {
8.       int i = 0;
9.       int a = i-- + ++i;
10.      int b = ++i + i--;
11.      cout << a << " " << b;
12.
13.      int res = x + f();
14.      cout << " " << res << "\n";
```

# Efecte laterale

Consecințe

- Contează strategia de evaluare
- Scade nivelul de abstractizare
  - Funcția add() nu mai e ca o cutie neagră → implementări diferite au efecte diferite

```
1.   int a = 5, b = 7;
2.   int add() {
3.      while (a > 0) {a--; b++;}
4.      return b;
5.   }
6.   //SAU
7.   int add() {
8.      int s = b;
9.      while (a > 0) {a--; s++;}
10.     return s;
11.  }
12.
13.  int main() {
14.    cout << add() << " " << b;
```

# Efecte laterale

Consecințe

- Contează strategia de evaluare
- Scade nivelul de abstractizare
  - Funcția add() nu mai e ca o cutie neagră → implementări diferite au efecte diferite

Afișează **12 12** → Funcția are efectul lateral al modificării lui b

Afișează **12 7** → Această implementare nu alterează valoarea lui b

```
1.   int a = 5, b = 7;
2.   int add() {
3.      while (a > 0) {a--; b++;}
4.      return b;
5.   }
6.   //SAU
7.   int add() {
8.      int s = b;
9.      while (a > 0) {a--; s++;}
10.     return s;
11.  }
12.
13.  int main() {
14.    cout << add(); cout << b;
```

# Efecte laterale

Consecințe

- Contează strategia de evaluare
- Scade nivelul de abstractizare

- Risc ridicat de bug-uri

# Transparență referențială – Cuprins

- Efecte laterale
- Transparență referențială

# Transparență referențială

## Transparență referențială

- **Există atunci când toate funcțiile/expresiile sunt pure**
- O expresie poate fi înlocuită prin valoarea sa (fără să se piardă nimic)

## Exemple

- Toate funcțiile Racket implementate de noi până acum
- `(random)`
- 
```
(define counter 1)
(define (display-and-inc-counter)
   (display counter)
   (set! counter (add1 counter)))
```

# Transparență referențială

## Transparență referențială

- **Există atunci când toate funcțiile/expresiile sunt pure**
- O expresie poate fi înlocuită prin valoarea sa (fără să se piardă nimic)

## Exemple

- Toate funcțiile Racket implementate de noi până acum — transparente referențial
- `(random)` — opacă referențial
- opacă referențial

```
(define counter 1)
(define (display-and-inc-counter)
   (display counter)
   (set! counter (add1 counter)))
```

# Transparența referențială - avantaje

- Analiză formală
  - programe elegante, ușor de verificat matematic
  - e suficient să analizăm codul funcției pentru a ști ce face funcția (nu are efecte în exterior)
- Optimizări de compilator
  - reordonarea/amânarea evaluărilor (expresiile se evaluează oricând la același rezultat)
  - caching (rezultatele deja calculate nu se pot schimba, deci pot fi reutilizate)
  - inlining (înlocuiește apelul cu corpul funcției)
- Paralelizare nativă
  - fire multiple de execuție (funcțiile nu își afectează reciproc execuția)
- Depanare și testare
  - funcții testate izolat (fără a ține cont de starea programului)

# Legare statică / dinamică – Cuprins

- Variabile
- Domeniu de vizibilitate
- Tipuri de legare a variabilelor
- Expresii pentru legare statică
- Expresii pentru legare dinamică

# Variabile

Variabilă = pereche identificator-valoare

Caracteristici

- Domeniu de vizibilitate (zona de program în care valoarea poate fi accesată prin identificator)
- Durată de viață

Observație

- În Racket, tipul este asociat valorilor, nu variabilelor

# Legare statică / dinamică – Cuprins

- Variabile
- Domeniu de vizibilitate
- Tipuri de legare a variabilelor
- Expresii pentru legare statică
- Expresii pentru legare dinamică

# Domeniu de vizibilitate al unei variabile

Domeniu de vizibilitate = mulțimea punctelor din program unde asocierea identificator – valoare este vizibilă (și valoarea se poate accesa prin identificator)

Exemple în Calcul Lambda

**(x λx.(λx.y λy.(x z)))**

**(x λx.(λx.y λy.(x z)))** – acest x nu mai este vizibil nicăieri în această zonă de program

**(x λx.(λx.y λy.(x z)))** – practic: zona din corp în care aparițiile lui x sunt libere

**(x λx.(λx.y λy.(x z)))**

# Domeniu de vizibilitate al unei variabile

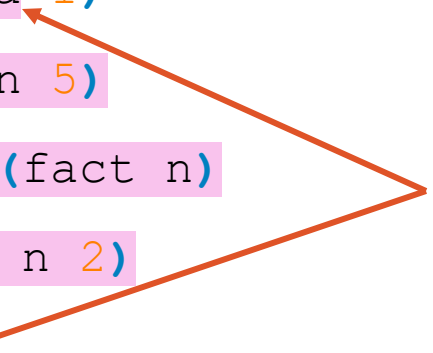
Exemplu în Racket

```
1.  (define a 1)
2.  (define n 5)
3.  (define (fact n)
4.    (if (< n 2)
5.        a
6.        (* n (fact (- n 1)))))
7.  (fact n)
```

# Domeniu de vizibilitate al unei variabile

## Exemplu în Racket

```
1.  (define a 1)
2.  (define n 5)
3.  (define (fact n)
4.    (if (< n 2)
5.        a
6.        (* n (fact (- n 1)))))
7.  (fact n)
```



# Domeniu de vizibilitate al unei variabile

Exemplu în Racket

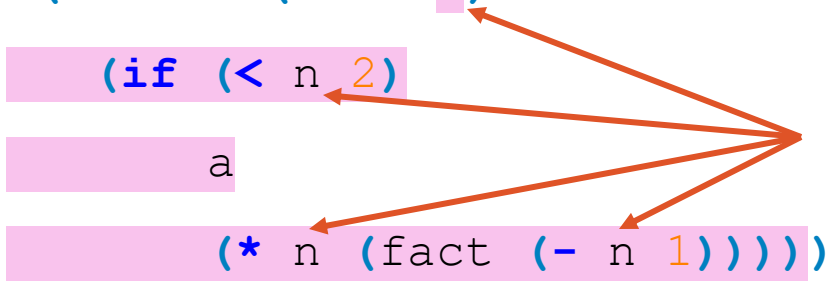
```
1.  (define a 1)
2.  (define n 5)
3.  (define (fact n)
4.    (if (< n 2)
5.        a
6.        (* n (fact (- n 1)))))
7.  (fact n)
```

În corpul funcției fact este vizibilă legarea parametrului n la valoarea pe care este aplicată funcția. Legarea interioară obscurează legarea de la `(define n 5)`.

# Domeniu de vizibilitate al unei variabile

## Exemplu în Racket

```
1.  (define a 1)
2.  (define n 5)
3.  (define (fact n)
4.    (if (< n 2)
5.        a
6.        (* n (fact (- n 1)))))
7.  (fact n)
```



# Legare statică / dinamică – Cuprins

- Variabile
- Domeniu de vizibilitate
- Tipuri de legare a variabilelor
- Expresii pentru legare statică
- Expresii pentru legare dinamică

# Tipuri de legare a variabilelor

Legare = asocierea identificatorului cu valoarea

- Se poate realiza la define, la aplicarea unei funcții pe argumente, la let (vom vedea)
- Modul în care se realizează determină domeniul de vizibilitate al variabilei

Legare statică (lexicală)

**Domeniul de vizibilitate este**

- **Controlat textual, prin construcții specifice limbajului (lambda, let, etc.)**
- **Determinat la compilare (static)**

Legare dinamică

**Domeniul de vizibilitate este**

- **Controlat de timp (se folosește cea mai recentă declarație a variabilei – cel mai recent define)**
- **Determinat la execuție (dinamic)**

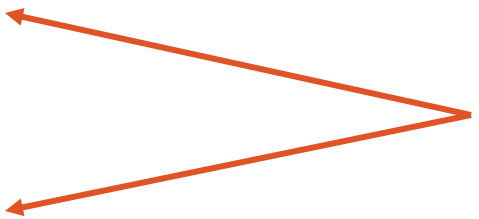
# Tipuri de legare a variabilelor

## Observații

- Calculul Lambda are doar legare statică
- Racket are legare statică (pentru variabilele definite cu define, contează „language”-ul)

## Exemplu de legare dinamică într-o versiune extinsă de Racket (lang PrettyBig)

```
1.  (define (f)
2.    (g 5))
3.
4.  (define (g x)
5.    (* x x))
6.  (f)
7.
8.  (define (g x)
9.    (* x x x))
10. (f)
```



Redefinirea nu este posibilă în lang racket, pentru exemplele de legare dinamică este necesar să schimbăm limbajul în Pretty Big.

# Tipuri de legare a variabilelor

## Observații

- Calculul Lambda are doar legare statică
- Racket are legare statică (pentru variabilele definite cu define, contează „language”-ul)

## Exemplu de legare dinamică într-o versiune extinsă de Racket (lang PrettyBig)

```
1.  (define (f)
2.    (g 5))
3.
4.  (define (g x)
5.    (* x x))
6.  (f)                  ;; 25      ← Când se intră în corpul lui f se evaluează g la definiția cea mai recentă
7.
8.  (define (g x)
9.    (* x x x))
10. (f)                  ;; 125     ← Când se intră în corpul lui f se evaluează g la definiția cea mai recentă
```

# Legare statică / dinamică – Cuprins

- Variabile
- Domeniu de vizibilitate
- Tipuri de legare a variabilelor
- Expresii pentru legare statică
- Expresii pentru legare dinamică

# Expresii pentru legare statică

- lambda
- let
- let*
- letrec
- named let

# Construcția `lambda`

```
( (lambda (var1 var2 … varm)
    expr1
    expr2
    …
    exprn)  arg1 arg2 … argm)
```

La aplicarea λ-expresiei pe argumente

- Se evaluează în ordine aleatoare argumentele $arg_1$, $arg_2$, … $arg_m$ (evaluare aplicativă)
- Se realizează legările $var_k \leftarrow valoare(arg_k)$
- Domeniul de vizibilitate al variabilei $var_k$ este corpul lui lambda (exceptând aparițiile legate ale lui $var_k$ în corp)
- Se evaluează în ordine expresiile din corpul funcției ($expr_1$, $expr_2$, … $expr_n$)
- Rezultatul este valoarea lui $expr_n$ (valorile lui $expr_1$, $expr_2$, … $expr_{n-1}$ se pierd)

# Construcția `let`

```
(let ((var1 e1) (var2 e2) … (varm em))
  expr1
  expr2
  …
  exprn)
```

La evaluare

- Se evaluează exact ca
  `((lambda (var1 var2 … varm) expr1 expr2 … exprn) e1 e2 … em)`
- Domeniul de vizibilitate al variabilei $var_k$ este corpul lui let (exceptând aparițiile legate ale lui $var_k$ în corp)
- Rezultatul este valoarea lui $expr_n$ (valorile lui $expr_1$, $expr_2$, … $expr_{n-1}$ se pierd)

# Construcția `let*`

```
(let* ((var1 e1) (var2 e2) (var3 e3) … (varm em))
  expr1
  expr2
  …
  exprn)
```

La evaluare

- Se realizează în ordine (stânga→dreapta) legările $var_k \leftarrow valoare(e_k)$
- Domeniul de vizibilitate al variabilei $var_k$ este restul textual (restul legărilor + corp) al lui let* (exceptând aparițiile legate ale lui $var_k$ în acest rest)
- Se evaluează în ordine expresiile din corpul funcției ($expr_1$, $expr_2$, … $expr_n$)
- Rezultatul este valoarea lui $expr_n$ (valorile lui $expr_1$, $expr_2$, … $expr_{n-1}$ se pierd)

# Construcția **letrec**

```
(letrec ((var1 e1) (var2 e2) (var3 e3) … (varm em))
  expr1
  expr2
  …
  exprn)
```

La evaluare

- Se realizează în ordine (stânga→dreapta) legările $var_k \leftarrow valoare(e_k)$
- Domeniul de vizibilitate al variabilei $var_k$ este întregul letrec (celelalte legări + corp) (exceptând aparițiile legate ale lui $var_k$ în această zonă), dar variabila trebuie să fi fost deja definită atunci când valoarea ei este solicitată într-o altă zonă din letrec
- Se evaluează în ordine expresiile din corpul funcției ($expr_1$, $expr_2$, … $expr_n$)
- Rezultatul este valoarea lui $expr_n$ (valorile lui $expr_1$, $expr_2$, … $expr_{n-1}$ se pierd)

# Construcția „named **let**”

```
(let nume ((var1 e1) (var2 e2) … (varm em))
  …
  (nume arg1 arg2 … argm)
  …)
```

## Semnificație

- Se creează o funcție recursivă (care va fi invocată în corpul named let-ului prin `nume`) cu parametrii $var_1$, $var_2$, … $var_m$, și se și aplică funcția pe argumentele $e_1$, $e_2$, … $e_m$
- Domeniul de vizibilitate al variabilei $var_k$ este corpul named let-ului (exceptând aparițiile legate ale lui $var_k$ în corp)
- Ca și la celelalte forme de let, rezultatul este valoarea ultimei expresii evaluate în corp

# Comparație între variantele de `let`

| Variantă | Tipar | Utilizări |
|---|---|---|
| `let` | variabile independente | constante locale |
| `let*` | variabile dependente (secvențial) | calcule care folosesc pașii anteriori |
| `letrec` | variabile recursive (auto/mutual) | funcții locale sau funcții mutual recursive |
| `let <name>` | funcție recursivă | iterație în context local<br>(ex: pentru a implementa recursivitate pe coadă) |

Regulă generală

- Când contextul interior folosește un nume existent într-un context exterior, legarea din contextul interior „câștigă” (o obscurează pe cealaltă)

# Legare statică / dinamică – Cuprins

- Variabile
- Domeniu de vizibilitate
- Tipuri de legare a variabilelor
- Expresii pentru legare statică
- Expresii pentru legare dinamică

# Construcția **`define`**

…

```
(define var expr)
```

…

La evaluare

- Se evaluează `expr`
- Se realizează legarea `var` ← `valoare(expr)`
- Domeniul de vizibilitate al variabilei `var` este întregul program
  - exceptând zonele obscurate de alte legări statice ale lui `var`
  - cu condiția ca la momentul referirii să nu existe o definire mai recentă a lui `var`

# Construcția `define` – Consecințe

Avantaje

- Definiri în ordine aleatoare (singura condiție este ca definirea să preceadă apelarea)
- Funcții mutual recursive (aceeași condiție ca mai sus)
- Programare în timp real (o redefinire este o versiune nouă cu care sistemul continuă să ruleze, fără ca utilizatorul să observe vreo întrerupere)

Dezavantaje (vezi exemplul de legare dinamică)

- Se pierde transparența referențială (când este permisă redefinirea, comportamentul unei funcții se poate schimba în funcție de cea mai recentă redefinire a componentelor sale)

# `define`, `let` și `set!`

Diferența este în primul rând la nivel intențional.

**`define`**

- Intenționează să lege pentru totdeauna un identificator la o valoare
- Nu e legal în toate zonele de program (nu se poate afla în mijlocul corpului unei funcții, dar îl poate precede, pentru a defini valori/funcții ajutătoare)

**`let`**

- Intenționează să creeze un context local pentru anumite variabile
- Nu realizează atribuire, la let variabila nu se modifică ci se naște

**`set!`**

- Intenționează să schimbe valoarea unei variabile (contrar principiilor paradigmei funcționale)
- Se poate folosi oriunde în program (risc sporit de bug-uri)

# Rezumat

Efect lateral

Funcție pură

Transparență referențială

Domeniu de vizibilitate

Legare statică

Legare dinamică

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură

Transparență referențială

Domeniu de vizibilitate

Legare statică

Legare dinamică

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială

Domeniu de vizibilitate

Legare statică

Legare dinamică

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială: toate funcțiile/expresiile sunt pure

Domeniu de vizibilitate

Legare statică

Legare dinamică

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială: toate funcțiile/expresiile sunt pure

Domeniu de vizibilitate: mulțimea punctelor din program în care variabila e vizibilă

Legare statică

Legare dinamică

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială: toate funcțiile/expresiile sunt pure

Domeniu de vizibilitate: mulțimea punctelor din program în care variabila e vizibilă

Legare statică: domeniu de vizibilitate controlat textual, determinat la compilare

Legare dinamică

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială: toate funcțiile/expresiile sunt pure

Domeniu de vizibilitate: mulțimea punctelor din program în care variabila e vizibilă

Legare statică: domeniu de vizibilitate controlat textual, determinat la compilare

Legare dinamică: domeniu de vizibilitate controlat de timp, determinat la execuție

Expresii de legare statică

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială: toate funcțiile/expresiile sunt pure

Domeniu de vizibilitate: mulțimea punctelor din program în care variabila e vizibilă

Legare statică: domeniu de vizibilitate controlat textual, determinat la compilare

Legare dinamică: domeniu de vizibilitate controlat de timp, determinat la execuție

Expresii de legare statică: lambda, let, let*, letrec, named let

Expresii de legare dinamică

# Rezumat

Efect lateral: alt efect al unei funcții în afară de efectul de a întoarce o valoare

Funcție pură: aplicată pe aceleași argumente întoarce aceeași valoare; fără efecte laterale

Transparență referențială: toate funcțiile/expresiile sunt pure

Domeniu de vizibilitate: mulțimea punctelor din program în care variabila e vizibilă

Legare statică: domeniu de vizibilitate controlat textual, determinat la compilare

Legare dinamică: domeniu de vizibilitate controlat de timp, determinat la execuție

Expresii de legare statică: lambda, let, let*, letrec, named let

Expresii de legare dinamică: define – când permitem redefinirea