

Lecture 1 - Introduction in declarative programming.

Recursion

Official web site: www.cs.ubbcluj.ro/~hfpop/pfl

Contents

References

Programming and programming languages

Recursion

Examples of recursion

References

Czibula, G., Pop, H.F., Elemente avansate de programare în Lisp și Prolog. Aplicații în Inteligența Artificială., Ed. Albastră, Cluj-Napoca, 2012

Programming and programming languages

➤ LANGUAGES

- **Procedural (imperative) - high level languages**
 - Fortran, Cobol, Algol, Pascal, C, ...
 - program - sequence of instructions
 - the assignment statement, control structures - for the control of sequential execution, branching and cycling
 - the role of the programmer - “what” and “how”
 1. to describe what is to be calculated
 2. to organize the calculation
 3. to organize memory management
 - !!! it is argued that the assignment instruction is dangerous in high-level languages, just as the GO TO instruction was considered dangerous for structured programming in the '68s.
- **Declarative (descriptive, applied) - very high level languages**
 - based on expressions
 - expressive, easy to understand (have a simple basis), extensible

- programs can be seen as descriptions that state information about values, rather than instructions to determine values or effects.
 - they give up instructions
 1. thus they protect users from making too many mistakes
 2. they are generated from mathematical principles - analysis, design, specification, implementation, abstraction and reasoning (deductions of consequences and properties) become more and more formal activities.
 - the role of the programmer - “what” (not “how”)
 - two classes of declarative languages
 1. **functional languages** (eg Lisp, ML, Scheme, Haskell, Erlang)
 - focus on values of data described by expressions (built through applications of functions and definitions of functions), with automatic evaluation of expressions
 2. **logical languages** (e.g. Prolog, Datalog, Parlog),
 - focus on logical assertions that describe the relationships between data values and automatic derivations of answers to questions, starting from these assertions.
 - applications in Artificial Intelligence – automated proofs, natural language processing and speech understanding, expert systems, machine learning, intelligent agents, etc.
- Multiparadigm languages: **F#, Python, Scala** (imperative, functional, object oriented)
 - Interactions between declarative and imperative languages - declarative languages that provide interfaces with imperative languages (eg C, Java): SWI Prolog, GNU Prolog, etc.
 - **Logtalk** – integrates logic and object-oriented programming
 - Logic programming in **Python**:
 - **Karen**
 - **SymPy** – library for symbolic computations

Recursion

- general mechanism to elaborate programs
- recursion arose from practical necessities (direct transcription of recursive mathematical formulas; see Ackermann's function)
- recursion is the mechanism by which a subprogram (function, procedure) calls itself
 - two types of recursion: **direct** or **indirect**
- **!!! Result**
 - any calculable function can therefore be expressed and programmed) in terms of recursive functions
- two things to consider in describing a recursive algorithm: **the recursive rule** and **the termination condition**
- **advantage** of recursion: source text that is extremely short and very clear.

- **disadvantage** of recursion: filling the stack segment if the number of recursive calls, respectively of the formal and local parameters of the recursive subprograms is high enough.
 - declarative languages have specific mechanisms to optimize the recursion (see the mechanism of tail recursion in Prolog).

Examples of recursion

Remarks

- a list is a sequence of items ($l_1 l_2 \dots l_n$)
- the empty list (with 0 elements) is denoted by \emptyset
- adding an item to a list is denoted by \oplus

1. Create list (1,2,3, ... n)

a) directly recursive

$$createLista(n) = \begin{cases} \emptyset & \text{daca } n = 0 \\ createLista(n-1) \oplus n & \text{altfel} \end{cases}$$

b) using a recursive auxiliary function to create the sublist (i, i + 1, ..., n)

// create the list consisting of the elements i, i + 1, ..., n

Recursive mathematical model

$$create(i, n) = \begin{cases} \emptyset & \text{daca } i > n \\ i \oplus create(i+1, n) & \text{altfel} \end{cases}$$

// create the list consisting of elements 1, 2, ..., n

$$createLista(n) = create(1, n)$$

Pseudocode

Data representation : singly linked list with dynamic allocation of nodes.

NodeLSI

e: TElement // useful information of node

urm: ^NodeLSI // address the following node is stored

LSI

prim: ^NodeLSI // address of the first node in the list

Function createNodeLSI(e)

{pre: e: TElement }

{post: return a ^NodeLSI having e as useful information }

{ allocates a storing space for a NodeLSI }

{p: ^NodLSI }

allocate(p)

[p].e \leftarrow e

[p].urm \leftarrow NIL

{ result returned by the function }

createNodeLSI \leftarrow p

EndFunction

Function create(i, n)

{post: return a ^NodLSI, pointer towards the head of the linked list formed by }

{ elements i, i+1, ..., n }

If i > n **then**

create \leftarrow NIL

else

{ allocate a storage space for a NodeLSI with usefun information e }

q \leftarrow **createNodeLSI**(i)

{ create the link between node q and the head of the linked list formed }

{ by elements i+1, ..., n }

[q].urm \leftarrow **create**(i+1, n)

create \leftarrow q

EndIf

EndFunction

Function createList(n)

{post: return a ^NodeLSI, pointer towards the head of the linked list formed by }

{ elements 1, 2,..., n }

createList \leftarrow **create**(1, n)

EndFunction

2. Given a natural number n, calculate the sum $1 + 2 + 3 + \dots + n$.

a) directly recursive

$$suma(n) = \begin{cases} 0 & \text{daca } n = 0 \\ n + suma(n - 1) & \text{altfel} \end{cases}$$

b) using a recursive auxiliary function for calculating the sum $i + (i + 1) + \dots + n$

$$suma_aux(n, i) = \begin{cases} 0 & \text{daca } i > n \\ i + suma(n, i + 1) & \text{altfel} \end{cases}$$

$$suma(n) = suma_aux(n, 0)$$

3. Add an item at the end of a list.

// build the list (l1, l2,..., ln, e)

$$adaug(e, l_1 l_2 \dots l_n) = \begin{cases} (e) & \text{daca } l \text{ e vida} \\ l_1 \oplus adaug(e, l_2 \dots l_n) & \text{altfel} \end{cases}$$

4. Search for an element in a list.

$$apare(E, l_1 l_2 \dots l_n) = \begin{cases} fals & \text{daca } l \text{ e vida} \\ adevarat & \text{daca } l_1 = E \\ apare(E, l_2 \dots l_n) & \text{altfel} \end{cases}$$

5. Count the number of occurrences of an item in the list.

$$nrap(E, l_1 l_2 \dots l_n) = \begin{cases} 0 & \text{daca } l \text{ e vida} \\ 1 + nrap(E, l_2 \dots l_n) & \text{daca } l_1 = E \\ nrap(E, l_2 \dots l_n) & \text{altfel} \end{cases}$$

6. Check if a numeric list is set.

$$eMultime(l_1 l_2 \dots l_n) = \begin{cases} adevarat & \text{daca } l \text{ e vida} \\ fals & \text{daca } l_1 \in (l_2 \dots l_n) \\ eMultime(l_2 \dots l_n) & \text{altfel} \end{cases}$$

7. Transform a numeric list into a set.

$$multime(l_1 l_2 \dots l_n) = \begin{cases} \phi & \text{daca } l \text{ e vida} \\ multime(l_2 \dots l_n) & \text{daca } l_1 \in (l_2 \dots l_n) \\ l_1 \oplus multime(l_2 \dots l_n) & \text{altfel} \end{cases}$$

8. Return the inverse of a list.

$$invers(l_1 l_2 \dots l_n) = \begin{cases} \phi & \text{daca } l \text{ e vida} \\ invers(l_2 \dots l_n) \oplus l_1 & \text{altfel} \end{cases}$$

9. Remove all occurrences of an item from a list.

$$stergera(E, l_1 l_2 \dots l_n) = \begin{cases} \phi & \text{daca } l \text{ e vida} \\ l_1 \oplus stergera(E, l_2 \dots l_n) & \text{daca } l_1 \neq E \\ stergera(E, l_2 \dots l_n) & \text{altfel} \end{cases}$$

10. Return the k-th element of a list ($k \geq 1$).

$$element(l_1 l_2 \dots l_n, k) = \begin{cases} \phi & \text{daca } l \text{ e vida} \\ l_1 & \text{daca } k = 1 \\ element(l_2, \dots, l_n, k-1) & \text{altfel} \end{cases}$$

11. Return the difference between two sets represented as lists.

$$diferenta(l_1 l_2 \dots l_n, p_1 p_2 \dots p_m) = \begin{cases} \phi & \text{daca } l \text{ e vida} \\ diferenta(l_2 \dots l_n, p_1 p_2 \dots p_m) & \text{daca } l_1 \in (p_1 p_2 \dots p_m) \\ l_1 \oplus diferenta(l_2 \dots l_n, p_1 p_2 \dots p_m) & \text{altfel} \end{cases}$$

Homework

1. Verify whether a natural number is prime.
2. Calculate the sum of the first k elements in a numeric list ($l_1 l_2 \dots l_n$)
3. Remove the first k even numbers from a numeric list.