Concepte fundamentale ale limbajelor de programare

Familii de limbaje de programare

Curs 02

conf. dr. ing. Ciprian-Bogdan Chirila

Universitatea Politehnica Timisoara
Departamentul de Calculatoare si Tehnologia Informatiei

February 27, 2023

Cuprins

Cele trei familii de limbaje de programare

- Există câteva criterii pentru clasificarea limbajelor de programare
  - imperative
  - funcționale
  - declarative
- În interiorul fiecărei familii există o diversitate de limbaje
- Au aceleași principii de bază

Limbaje de programare imperative

- Imperative inseamna bazate pe instrucțiuni
- Sunt cele mai răspândite
  - Fortran, Cobol, Basic, Pascal, Ada, Modula-2, C, C++, C#, Java
- Concepția lor este bazată pe arhitectura traditională von Neumann
- Ruleaza pe un calculator format din:
  - memorie (pentru date si instrucțiuni)
  - unitate de comandă
  - unitate de execuție

Limbaje de programare imperative

- Bazat pe două concepte cheie:
  - Secvența
    - Execuția instrucțiunilor pas cu pas
  - Păstrarea unui set modificabil de valori în timpul execuției programului
    - Aceste valori definesc starea sistemului

Limbaje de programare imperative

- Cele trei componente esențiale:
  - Variabilele
    - Componenta majoră în limbajele de programare imperative
    - Celulele de memorie cu nume asignat si cu valori stocate
  - Instrucțiunea de atribuire
    - Memorarea unei valori calculate
  - Iterația
    - Este modul tipic de a face calcule complexe
    - Înseamnă a executa în mod repetat un set de instrucțiuni

Exemplu de program imperativ C
Testarea numerelor prime

```
#include <stdio.h>
#include <math.h>
int prime(unsigned long n)
{
 unsigned long i;
 if(n<=1) return 0;
 for(i=2;i<sqrt(n);i++)
  if(n%i==0) return 0;
 return 1;
}
int main()
{
 unsigned long n;
 printf("N=");
 scanf("%ld", &n);
 if(prime(n)) printf("The number %ld is prime!", n);
 else printf("The number %ld is not prime!",n);
}
```

Cuprins

Limbaje de programare funcționale

- sunt bazate pe concepte matematice de:
  - Funcție
  - Aplicarea funcției
- sunt numite și limbaje aplicative
- sunt libere, independente de conceptul von Neumann ce implică variabile și atribuiri
- exemple de limbaje funcționale
  - LISP, SML, Miranda

Cele 4 componente esențiale ale limbajelor funcționale

- setul predefinit de funcții primitive
- setul formelor funcționale
  - Mecanisme care permit combinarea funcțiilor pentru a crea altele noi
- operația de aplicare
  - Permite aplicarea funcțiilor pe argumente și producerea ca rezultat de valori noi
- setul de date (obiectele)
  - setul de argumente și valori ale funcțiilor

Exemplu in limbajul functional Lisp
Numararea atomilor unei liste

```
(defun count(x)
 (COND ((NULL x) 0)
  ((ATOM x) 1)
   (T (+ count (CAR x))
      (count (CDR x))))))
```

Exemplu in limbajul functional of F#
Generarea numerelor lui Fibonacci

```
let fib n =
    let rec g n f0 f1 =
        match n with
        | 0 -> f0
        | 1 -> f1
        | _ -> g (n - 1) f1 (f0 + f1)
    g n 0 1
```

Cuprins

Limbaje de programare declarative

- În procesul de dezvoltare a unui sistem software
  - În faza de specificare a cerințelor aflăm:
    - **Ce** trebuie să facă un sistem
  - În faza de proiectare și implementare aflăm:
    - **Cum** funcționează acel sistem

Ce este nou/diferit la limbajele de programare declarative?

- se descrie ce se așteaptă de la un sistem
- nu se definește implementarea acelui sistem
- se specifică doar:
  - proprietățile problemei
  - condițiile problemei
- sistemul software va găsi răspunsuri în mod automat

Limbaje de programare declarative

- efortul și creativitatea sunt concentrate în faza de definire a cerințelor
- sunt limbaje de nivel foarte înalt
- SNOBOL4
- SQL
  - Structures Query Language
  - utilizat pentru interogarea bazelor de date
- Prolog
  - limbaj declarativ si logic
  - condițiile problemelor sunt exprimate prin calcul de predicate

Examplu de program declarativ in LINQ

```
var results =
    from c in SomeCollection
    where c.SomeProperty < 10
    select new {c.SomeProperty, c.OtherProperty};

foreach (var result in results)
{
    Console.WriteLine(result);
}
---
var results =
    SomeCollection
        .Where(c => c.SomeProperty < 10)
        .Select(c => new {c.SomeProperty, c.OtherProperty});

results.ForEach(x => {Console.WriteLine(x.ToString());})
```

Exemplu de program declarativ in Prolog

```
parent(helen,ralph).
parent(peter,ralph).
parent(peter,marry).
parent(ralph,anna).
parent(ralph,dan).
```

```
? - parent(peter,mary).
yes

? - parent(peter,x).
x=ralph

? - parent(peter,x).
x=ralph;
x=mary;
no

? - parent(y,anna), parrent(x,y).
x=helen;
x= peter;
y=ralph;
no
```

Limbajele de programare

- in general nu sunt pure
  - sunt imperative sau funcționale sau declarative
- ML
  - este funcțional cu facilități imperative
- C
  - permite scrierea de programe ce definesc și utilizează funcții în manieră intensivă
- F#
  - este limbaj funcțional cu facilități imperative

Limbajele de programare și calculatoarele

- Limbajele imperative
  - rulează optim pe calculatoarele curente
- Limbajele funcționale și declarative au:
  - baze teoretice solide
  - mecanisme de verificare automată
  - oferă programare la nivel înalt

Domeniile limbajelor de programare funcționale și declarative

- Inteligență artificială
- Procesarea listelor
- Baze de date
- Calcul simbolic
- Procesarea limbajului natural
- Baze de cunoștinte
- Verificarea programelor
- Demonstratoare de teoreme

Cuprins

Programarea secvențială vs. programarea concurentă

- Programele imperative implică
  - acțiuni
  - date
- Acțiunea următoare este inițiată cînd acțiunea curentă s-a terminat
- Programul devine un proces
- Activitatea de programare se numește programare secvențială

Cuprins

Procese paralele vs. procese concurente

- Un proces utilizează resursele calculatorului
  - câte una la un moment dat
- dacă un singur process în sistem utilizează la un moment dat toate resusrsele, rezultă:
  - performanțe slabe de utilizare/exploatare
  - că procesoarele multiple sunt nefolositoare
- mai multe procese în memorie ce utilizează procesorul în regim de împărțire a timpului sunt utile
  - are loc un paralelism logic
  - În mod virtual, procesele sunt executate în mod paralel

Sisteme de operare de multiprogramare

- În memorie sunt mai multe programe
- Are loc o execuție paralelă
- Paralelismul fizic depinde de:
  - Numărul de procesoare/nuclee/nuclee virtuale
  - Tipul de procesoare/nuclee/nuclee virtuale (hyperthreading)
- Windows, Unix, MacOS, OS/2, Android
  - permit gestionarea de procese în mod multiprogramat
  - partajeaza resursele sistemului
  - rezultă o îmbunătățire a utilizării resurselor

Programea în sisteme de operare multiprogramare

- procese ce se execută paralel
- se execută independent
  - ca și cum ar rula pe un sistem monoprogramat
- Conflictele de resurse
  - sunt gestionate de sistemul de operare
  - sunt transparente aplicațiilor scrise de programatori

Procese cu comunicare

- Procesele izolate nu sunt o soluție
- O soluție ar fi procesele multiple
  - asincrone
  - cu schimb de mesaje
  - cu transferuri de date
  - cu partajarea în comun a resurselor sistemului
- Procese concurente
- Câteodată au nevoie de sincronizare

Cazuri de sincronizare

- Excluderea mutuală
- Cooperarea

Excluderea mutuală

- când procese multiple folosesc aceleași resurse
- accesul la resursă este permis doar unui process la un moment dat
- cererile de acces trebuiesc secvențializate
- sincronizările se pot baza pe condiții
  - Se întârzie un process până când o condiție devine adevărată
- Resursă critică
  - este o resursă ce poate fi folosită într-un singur process la un moment dat
- Secțiunea critică
  - secțiune de cod ce manipunează resursa critică

Excluderea mutuală

Definitie:

- Excluderea mutuală este o formă de sincronizare pentru procese concurente permitând unui singur process să fie în sectiunea critică la un moment dat
- Construcția de limbaj ce rezolvă această problemă este secțiunea critică
  - concepută de CAR Hoare si P. Brinch Hansen in 1972
  - are ca scop evidențierea textului de program și a variabilelor ce se referă la resursa critică
  - Se adaugă cuvine cheie noi cum ar fi: region, when pentru accesul la resurse
  - Prin adăugarea unei condiții de sincronizare se obtine o regiune critică conditionată

Cooperarea

- Mesaje și date ce sunt schimbate între procese
- Se păstrează o relație de producator-consumator
- Informațiile produse de un proces sunt consumate de altul
- Descrierea proceselor concurente si relatia dinttre ele conduce la programarea concurentă
- Resursele
  - Sunt partajate între procesele autorizate
  - Sunt protejate de procesele neautorizate
- Când este implicat factorul de timp rezultă procese în timp real
- Limbaje de programare pentru procese concurente

Exemplu de program Cuda
Algoritmul PageRank paralel

```
__global__ void pageRankKernel(int n, double dp, int *matrix, int *nwol, double *op
{
 int p = blockIdx.x * blockDim.x + threadIdx.x;
 if(p>=0 && p < n)
 {
  double nprp;
  nprp = dp + 0.15 / n;
  for (int k = 0; ibn[p*n+k] != -1 && k < n; k++)
  {
   int ip = ibn[p*n+k];
   nprp = nprp + 0.85 * opr[ip] * matrix[p*n+ip] / ow[ip];
  }
  npr[p]=nprp;
 }
}
int main()
{
 ...
 // Launch a kernel on the GPU with one thread for each node
 pageRankKernel<<< grid, block >>>(n, dp, dev_matrix, dev_nwol, dev_opr, dev_npr, d
 ...
```

Sisteme distribuite

- sisteme concurente
- sunt cele mai răspândite din cauza Internetului și a rețelelor
- comunicarea este bazată pe transmiterea de mesaje

Cuprins

Limbaje de programare concurente

- au fost dezvoltate in ultimii 40 de ani
- au facilități speciale ce descriu:
  - procese paralele și concurente
  - sincronizarea și comunicarea
- limbajul Edison definit de P. Brinch Hansen in 1980
- descrie programe concurente de dimensiuni mici și medii pentru micro și mini sisteme de calcul

Instrucțiunea "when"

- The processes
  - communicate through common variables
  - synchronize through conditional critical regions

```
when b_1 do instr_list_1
else b_2 do instr_list_2
...
else b_n do instr_list_n
```

Insctucțiunea ”when”

- Variabila comună pentru regiunea critică nu este specificată
- Soluția adoptată în limbajul Edison
  - Excluziunea mutuală în toate regiunile critice
  - Doar o secventă critică este executată la un moment dat
- Astfel rezultă:
  - Implementare simplificată a limbajului
  - Restricții complexe legate de concurența proceselor

Instrucțiunea "when"

- Se execută în două faze:
  - Faza de sincronizare:
    - Procesul este întârziat până când nici un alt proces nu execută faza critică a unei instrucțiuni
  - Faza critică:
    - sunt evaluate expresiile logice b_1, b_2,...,b_n
    - Dacă una dintre ele este adevarată atunci lista de instrucțiuni corespunzătoare este executată
    - Dacă toate condițiile sunt false atunci faza de sincronizare se repetă

Instrucțiunea "cobegin"

```
cobegin
 const_1 do instr_list_1 also
 const_2 do instr_list_2 also
 ...
 const_n do instr_list_n
end
```

Instrucțiunea "cobegin"

- Descrie activități concurente
- Lista de instrucțiuni reprezintă procese ce se execută în paralel
- Procesele încep la instrucțiunea cobegin
- Instrucțiunea cobegin se termină atunci când toate procesele se termină
- Fiecare process are o constantă atașată
- Semantica acestei constante este dependentă de implementarea limbajului de programare
  - Spațiul de memorie necesar fiecărui proces
  - Numărul procesorului pe care să se execute procesul
  - Prioritatea procesului în sistem etc.

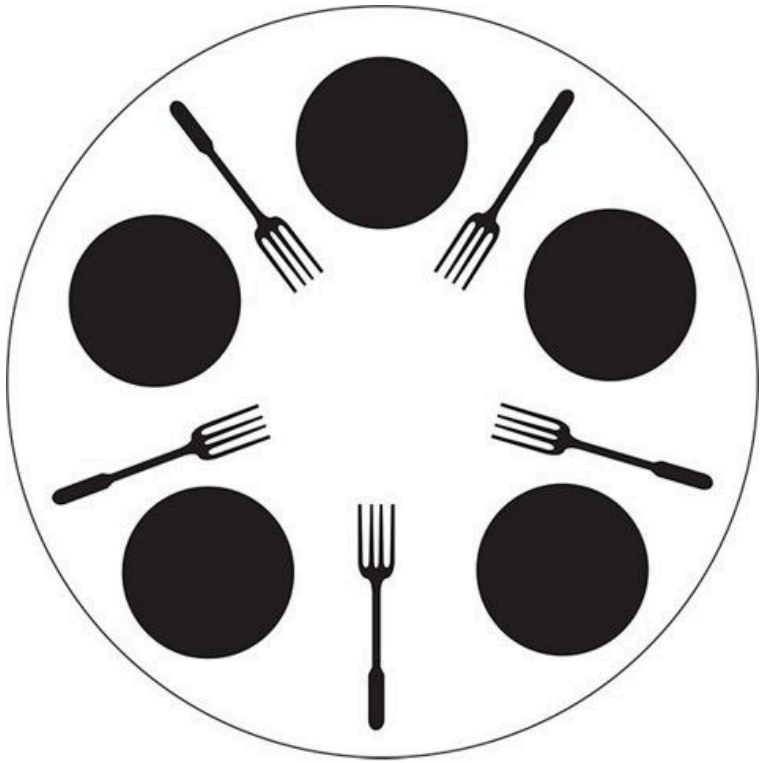
Programele Edison

- Au forma unor proceduri
- Sunt lansate prin activarea instrucțiunilor de procedură
- Sunt formate din mai multe module
- Identificatorii exportați sunt precedati de simbolul *(stea)

Problema filosofilor

- 5 filosofi își petrec timpul mâncând și meditând
- Când unui filosof i se face foame merge în sala de mese, se așează la masă și mănâncă
- Pentru a mânca spaghete din oala el are nevoie de 2 furculițe
- Pe masă sunt doar 5 furculițe
- Există doar o furculiță între două locuri
- Fiecare filosof poate accesa doar furculițele din stânga sau din dreapta sa
- După ce a mâncat (într-un timp finit) filosoful pune inapoi furculițele și părăsește camera

Problema filosofilor



Programul solutie

- Comportamentul filosofilor este modelat de procese concurente
- Furculițele sunt modelate ca și resurse partajate
- Un filosof asteaptă până când ambele furculițe din stânga si dreapta sa sunt libere
- Tabloul “forks” stochează numărul de furculițe disponibile unui filosof.
- Se poate ajunge în situația de înfometare (starvation) când vecinii unui filosof manancă alternativ
- Cei 5 filosofi sunt reprezentati prin activarea procedurii “philosopherlife” din fiecare ramură a instrucțiunii cobegin
- Fiecare ramură lansează în paralel un proces

Programul filosofilor

```
proc philosophers
module
  array tforks[0...4] (int)
  var forks:tforks; i:int;

  proc philoright(i:int):int
  begin
    val philoright:=(i+1) mod 5
  end

  proc philoleft(i:intr):int
  begin
    if i=0 do val philoleft:=4
    else true val philoleft:=i-1
  end
```

Programul filosofilor

```
*proc get(philo:int)
begin
  when forks[philo] = 2 do
    forks[philoright(philo)]:=forks[philoright(philo)]-1;
    forks[philoleft(philo)]:=forks[philoleft(philo)]-1;
  end
end
*proc put(philo:int)
begin
  when true do
    forks[philoright(philo)]:=forks[philoright(philo)]+1;
    forks[philoleft(philo)]:=forks[philolefttt(philo)]+1;
  end
end
```

Programul filosofilor

```
begin
  i:=0
  while i<5
    forks[i]:=2
    i:=i+1;
  end
end
```

Programul filosofilor

```
proc philosoperlife(i:int)
begin
  while true do
    -think-
    get(i);
    -eat-
    put(i);
  end
end
```

Programul filosofilor

```
begin
  cobegin
    1 do philosopherlife(0) also
    2 do philosopherlife(1) also
    3 do philosopherlife(2) also
    4 do philosopherlife(3) also
    5 do philosopherlife(4) also
  end
end
```

Cuprins

Programarea sistemelor distribuite

- Sistemul distribuit
  - este un set de calculatoare capabile de schimb de informații
  - calculatoarele sunt numite noduri
  - ele pot fi programate să rezolve probleme bazându-se pe:
    - concurență
    - paralelism

Probleme de algoritmică tipice

- Sincronizarea bazată pe o condiție
- Difuzarea de mesaje la toate nodurile
- Selecția de procese pentru îndeplinirea de acțiuni speciale
- Detecția terminării
  - Un nod ce execută o acțiune de calcul trebuie să poată fi capabil să își detecteze momentul de încheiere

Probleme de algoritmică tipice

- Excluziunea mutuală
  - Utilizarea resurselor prin excluziune mutuală
  - Fișiere, imprimante, etc
- Detecția și prevenirea blocajelor (deadlock)
- Gestiunea sistemelor de fișiere distribuite
- Un limbaj de programare pentru sisteme distribuite trebuie să aibă toate aceste facilități: Java
- exemplu: un sistem de mesaje (chat)

Modelul client-server

- Procesele programului server
  - Gestionează resursele
- Procesele programului client
  - Accesează resursele gestionate de servere
- Mesajele sunt limitate la maxim o linie
- Serverul
  - Trebuie să fie primul pornit
  - A fost dezvoltat în unitatea de compilare Server.java
- Clientul
  - Trimite o cerere
  - Asteaptă un răspuns
  - Trimite comanda STOP
  - A fost dezvoltat în unitatea de compilare Client.java

Client.java

```
import java.net.*; import java.io.*;
class Client
{
  public static void main(String[] args) throws IOException
  {
    Socket cs=null;
    BufferedReader is=null; DataOutputStream os=null;
    try
    {
      cs=new Socket("localhost",5678);
      is=new BufferedReader(new InputStreamReader(cs.getInputStream()));
      os=new DataOutputStream(cs.getOutputStream());
    }
    catch(UnknownHostException e)
    {
      System.out.println("No such host");
    }
```

Client.java

```
    BufferedReader stdin=
      new BufferedReader(new InputStreamReader(System.in));
    String line;
    for(;;)
    {
      line=stdin.readLine()+"\n";
      os.writeBytes(line);
      System.out.println("Transmission:\t"+line);
      if(line.equals("STOP\n")) break;
      line=is.readLine();
      System.out.println("Receiving:\t"+line);
    }
    System.out.println("READY");
    cs.close(); is.close(); os.close();
  }
}
```

Server.java

```
import java.net.*;
import java.io.*;
class Server
{
  public static void main(String[] args) throws IOException
  {
    ServerSocket ss=null; Socket cs=null;
    BufferedReader is=null; DataOutputStream os=null;
    try
    {
      ss=new ServerSocket(5678);
      System.out.println("The server is running!");
      cs=ss.accept();
      is=new BufferedReader(new InputStreamReader(cs.getInputStream()));
      os=new DataOutputStream(cs.getOutputStream());
```

Server.java

```
      BufferedReader stdin=
        new BufferedReader(new InputStreamReader(System.in));
      String line;
      for(;;)
      {
        line=is.readLine();
        System.out.println("Receiving:\t"+line);
        if(line.equals("STOP")) break;
        line=stdin.readLine()+"\n";
        os.writeBytes(line);
      }
    }
    finally
    {
      cs.close(); ss.close();
      is.close(); os.close();
    }
  }
}
```

Conceptul de soclu (socket)

- O adresă de IP identifică un calculator în internet
- Un număr de port identifică un program rulând pe un calculator:
  - 21 server file transfer protocol ftp
  - 22 secure shell ssh
  - 80 server web cu protocolul http
  - 443 sever web cu protocolul securizat https
  - etc.

Conceptul de soclu (socket)

- O combinație dintre o adresă de IP si un port este un punct final dintr-o cale de comunicații
- Două aplicații ce comunică trebuie să se poată găsi una pe alta în Internet
- În mod normal clientul trebuie să găsească serverul
- Serverul trebuie să aibă o adresă bine definită și cunoscută de clienți

Conceptul de soclu (socket)

- Clientul se conectează la server prin instanțierea unui obiect de tip conexiune soclu (Socket)
- Primul mesaj al clientului către server conține soclul clientului
- Serverul transmite adresa sa de soclu către client în primul mesaj răspuns
- Transmisia de date este facută prin fluxurile de date de intrare și ieșire ale soclurilor
- Fluxurile de date pot fi accesate prin metodele din clasa Socket
  - getInputStream()
  - getOutputStream()

Cuprins

Scurta istorie a dezvoltării limbajelor de programare

- Primul limbaj de programare de nivel înalt a fost creat în anii 1950
- În acea perioadă se punea problema eficienței
- Fortran
  - Creat de un grup de lucru de la IMB condus de John Bachus în 1954
- Algol 60
  - Creat între anii 1958-1960
  - Avea următoarele facilități:
    - blocuri structurate
    - proceduri recursive

Scurta istorie a dezvoltării limbajelor de programare

- Finanțat de departamentul apărării din America in 1959
- Pentru aplicații economice
- Lucru cu fișiere
- Facilități de descriere a datelor
  - record
  - struct
- Utilizat și în ziua de azi într-o versiune evoluată

Scurta istorie a dezvoltării limbajelor de programare
La finalul anilor 50 și începuturile anilor 60

- Au apărut limbajele de programare funcționale
  - Lisp
    - Creat de John McCarthy la MIT în anul 1955
    - Limbaj de programare principal în domeniul inteligenței artificiale
  - APL
    - Creat de Iverson la firma IBM în anul 1962
- Au apărut limbajele de programare declarative
  - Snobol
    - Creat la Bell Laboratories în anul 1964

Scurta istorie a dezvoltării limbajelor de programare
La mijlocul anilor 60

- Există o mare diversitate de limbaje de programare
- Proiectul IBM
  - s-a încercat concentrarea tuturor conceptelor într-un singur limbaj de programare
  - s-a încercat înlocuirea tuturor celorlalte limbaje de programare
  - astfel a rezultat limbajul PL/I în anul 1964
    - Succesul a fost unul limitat
    - Complex
    - Greoi de utilizat

Scurta istorie a dezvoltării limbajelor de programare

În anii 60

- Algol 68 1968
  - Cu concepte perfect ortogonale
  - A fost definit utilizând metode formale
- Simula 67 1967
  - Limbaj de programare de simulare
  - Prezintă facilități de simulare
  - De origine norvegiană
  - Creat de Ole-Johan Dahl și Kristen Nygaard
  - Introduce următoarele concepte:
    - Modularizare, descriere de date abstracte
    - Obiect, clasă, moștenire, subclasă, proceduri virtuale
    - Simularea de evenimente discrete, garbage collection

Scurta istorie a dezvoltării limbajelor de programare

În anii 70

- Pascal
  - creat în anul 1977
  - de N. Wirth
  - expresiv
  - simplu
- ML
  - Creat în anul 1973
  - Universitatea din Edinburgh
  - Limbaj de programare funcțional
  - Puternic tipizat

Scurta istorie a dezvoltării limbajelor de programare

În anii 70

- C 1974
  - Este unul dintre cele mai răstândite limbaje de programare
  - Creat de Dennis Ritchie la Bell Labs
  - Implementarea portabilă pengtru sistemul de operare Unix
  - Programele C au portabilitate bună

Scurta istorie a dezvoltării limbajelor de programare

În anii 70

- Tipurile de date abstracte
- Verificarea programelor
- Tratarea excepțiilor
- Programarea concurentă

Scurta istorie a dezvoltării limbajelor de programare

În anii 70

- Mesa (Terax, 1974)
- Concurrent Pascal (Hansen, 1975)
- CLU (Liskov, MIT 1974)
- Modula 2 (Wirth, 1977)
- Ada (DoD, 1979)
- Prolog (Colmeraurer, 1972)
  - Programare logică
  - Inteligență artificială

Scurta istorie a dezvoltării limbajelor de programare

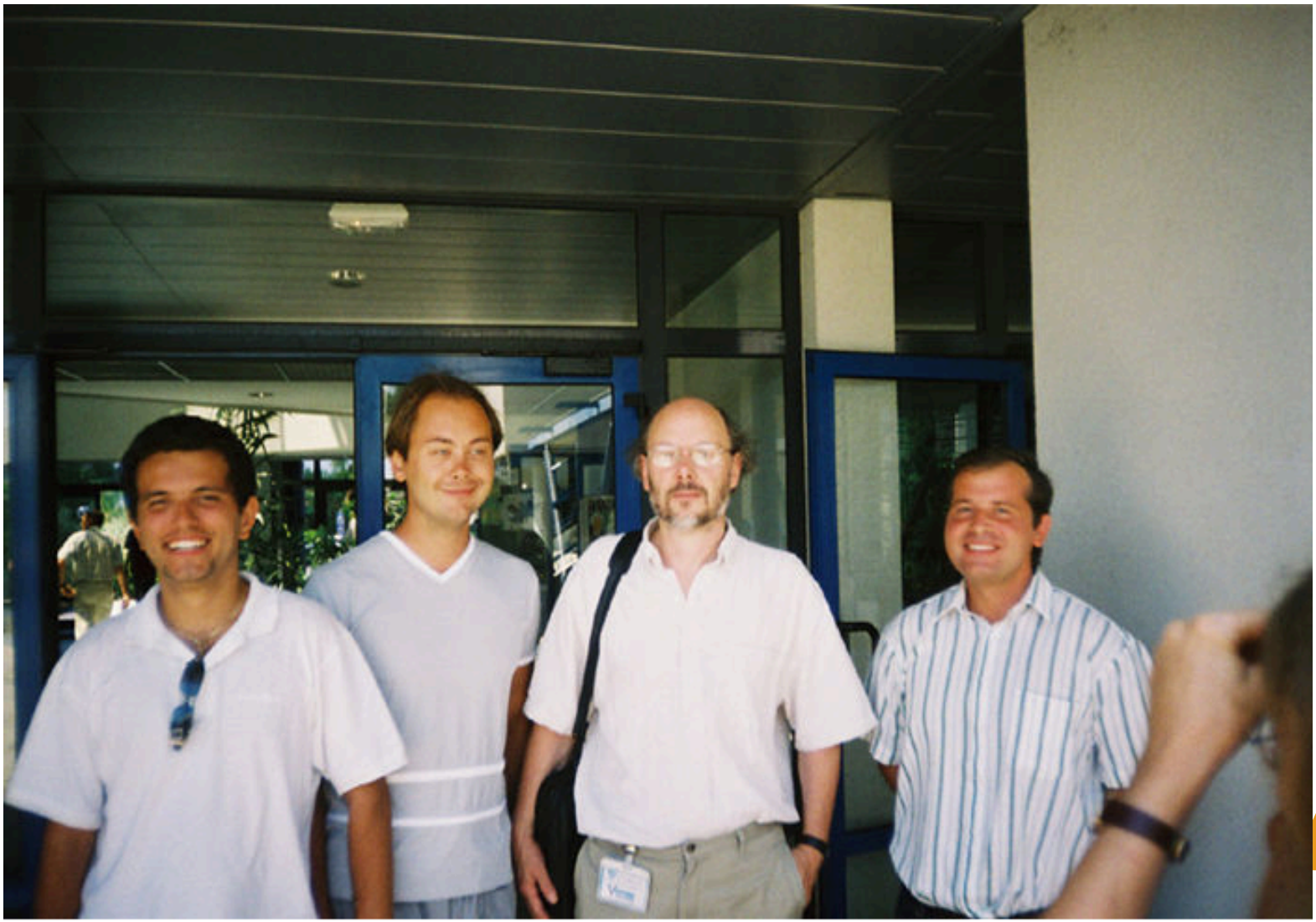
În anii 80

- Common Lisp 1984
  - Utilizat și consolidat
- Standard ML
  - SML, Milner, Edinburgh, 1985
- Miranda
  - Turner Kent, 1985
- Haskell
  - Hudak, 1988

Scurta istorie a dezvoltării limbajelor de programare

- Limbaje de programare orientate-obiect
- SmallTalk
  - Limbaj de programare și mediu de dezvoltare integrate
  - Dezvoltat de Xerox la finalul anilor 70
- C++
  - creat de Bjarne Stroustrup la Bell Labs în anul 1988
  - C cu concepte de programare orientată obiect
  - Utilizat pe scară largă în ziua de azi
  - Standarde
    - 1998 C++98 ISO/IEC 14882:2003
    - 2011 C++11 C++0x
    - 2014 C++14 C++1y
    - 2017 C++17 C++1z
    - 2020 C++20
    - 2023 C++23

Seminar cu Bjarne Stroustrup la INRIA, Sophia Antipolis, Franța, iulie 2003



Scurta istorie a dezvoltării limbajelor de programare

- Python
  - developed in late 80's by Guido van Rossum in Netherlands
  - successor of ABC programming language
  - first released in 1991
  - Python 2.0 released in 2000, Python 3.0 released in 2008
  - multiparadigm: object-oriented, structured, support for functional programming, aspect-oriented programming (meta-programming, meta-objects)
- Object Oberon
  - Zurich, 1989
- Eiffel
  - Bertrand Meyer, 1988
- Java
  - Sun Microsystem Inc., 1995, acum detinut de Oracle
  - Limbaj de programare orientat obiecte
  - interactiv
  - Cu grafică și animație
  - Pentru aplicații Internet

Scurta istorie a dezvoltării limbajelor de programare: Java

- tehnologie pentru mediul industrial
  - editie micro - carduri, dispozitive
  - editie standard - aplicatii desktop , interactivitate, animatie
  - editie enterprise - aplicatii distribuite in Internet (SpringBoot)
  - editie mobile - aplicatii mobile
- in 2022, Java version 19
- Anti C++
  - Nu are aritmetica pointerilor
  - Eliberarea memoriei nu se face manual
  - Nu are moștenire multiplă între clase
- alte limbaje de programare orientate obiect
  - Object Pascal(Delphi, Borland 1995-2000)
  - CLOS (Common Lisp Object System)
  - OCAML (ML orientat pe obiecte)

Scurta istorie a dezvoltării limbajelor de programare

- Microsoft C#
  - de uz general, orientat obiect, suporta internationalizare
  - verificare de tipuri la string-uri, verificarea limitelor la tablouri
  - versiunea Alpha a apărut în anul 2000
  - dezvoltat de o echipă de la firma Microsoft condusă de by Andres Hejlsberg
  - derivat din C, C++ and Java
  - portabilitatea a fost preluată de la Java
  - poate fi combinat cu alte limbaje de programare: F#, C++, LINQ
  - proiectat pentru constructia de componente instalabile in medii distribuite
  - se integreaza complet cu sistemul de operare MS Windows
- Java vs. C#
  - doar timpul va decide

Bibliografie

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.