Arhitectura Sistemelor de Calcul

Lect. Dr. Șotropa Diana
diana.sotropa@ubbcluj.ro

Facultatea de Matematică și Informatică
Universitatea Babeș-Bolyai

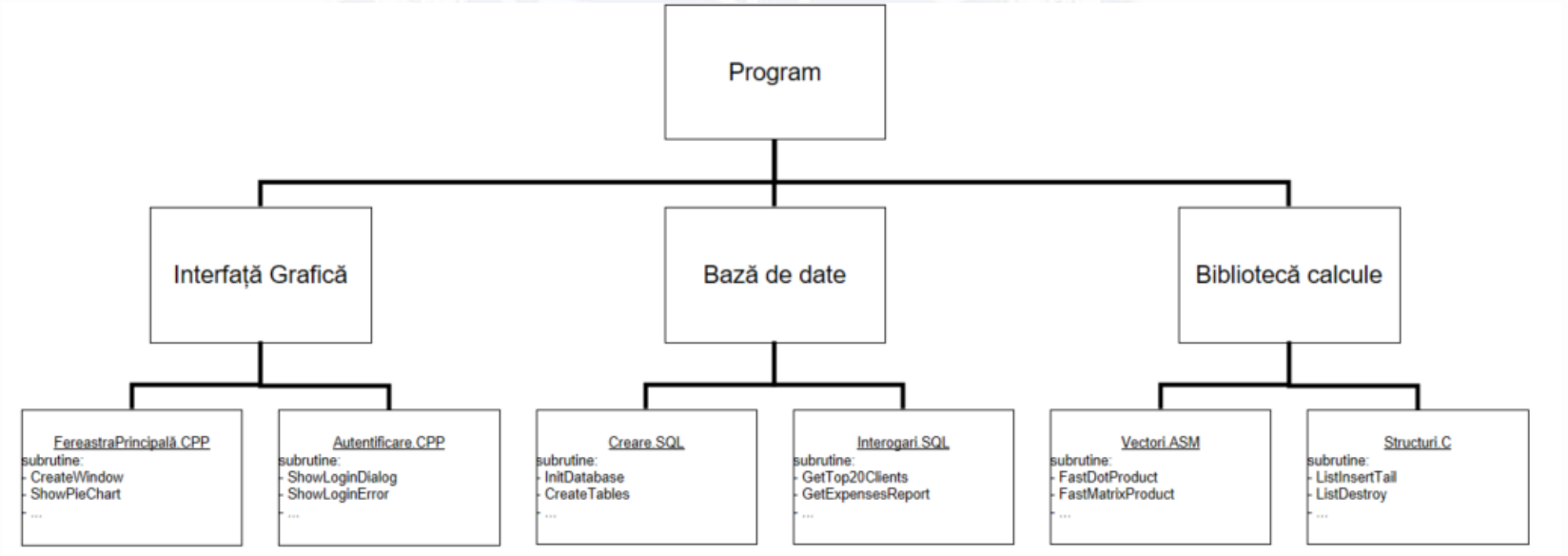
Programarea multimodul

Material creat de către Lect. Dr. Vancea Alexandru împreună cu Marius Vanța
Bitdefender 2017

Arhitecturi modulare

Programare modulară

- Cum împărțim problema în sub-probleme?
  - Modularizare
    - Program -> unități logice
    - Cod (al unităților) -> fișiere distincte
    - Fișiere -> subrutine



Arhitecturi modulare

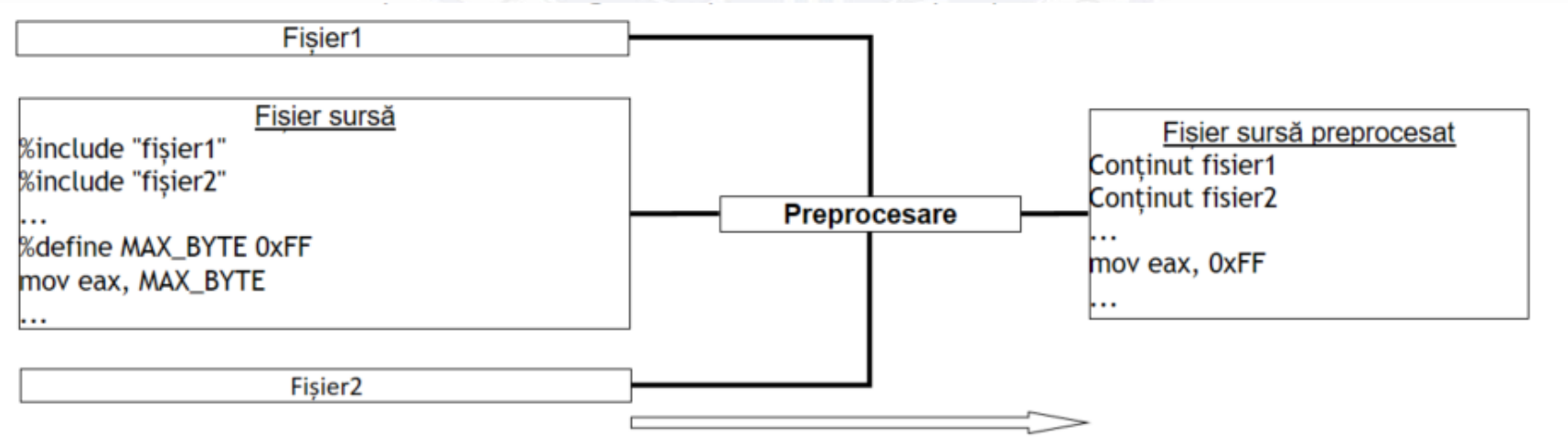
Programare modulară

- Pentru care sub-probleme există deja rezolvări disponibile?
  - Reutilizare:
    - Fișiere sursă
      - Refolosire cod și date din asamblare
        - Directiva `%include`
          (NU este programare multi modul, deoarece la compilare ajunge DOAR UN SINGUR MODUL obținut prin concatenarea textuală a fișierelor incluse)
    - Fișiere binare
      - Refolosire cod și date din asamblare
      - Refolosire cod și date din limbaje de nivel înalt
      - Biblioteci
  - Existența de fișiere binare separate implică ASAMBLARE / COMPILARE SEPARATĂ

Tehnici și instrumente

Includerea statică la compilare / asamblare: directiva %include

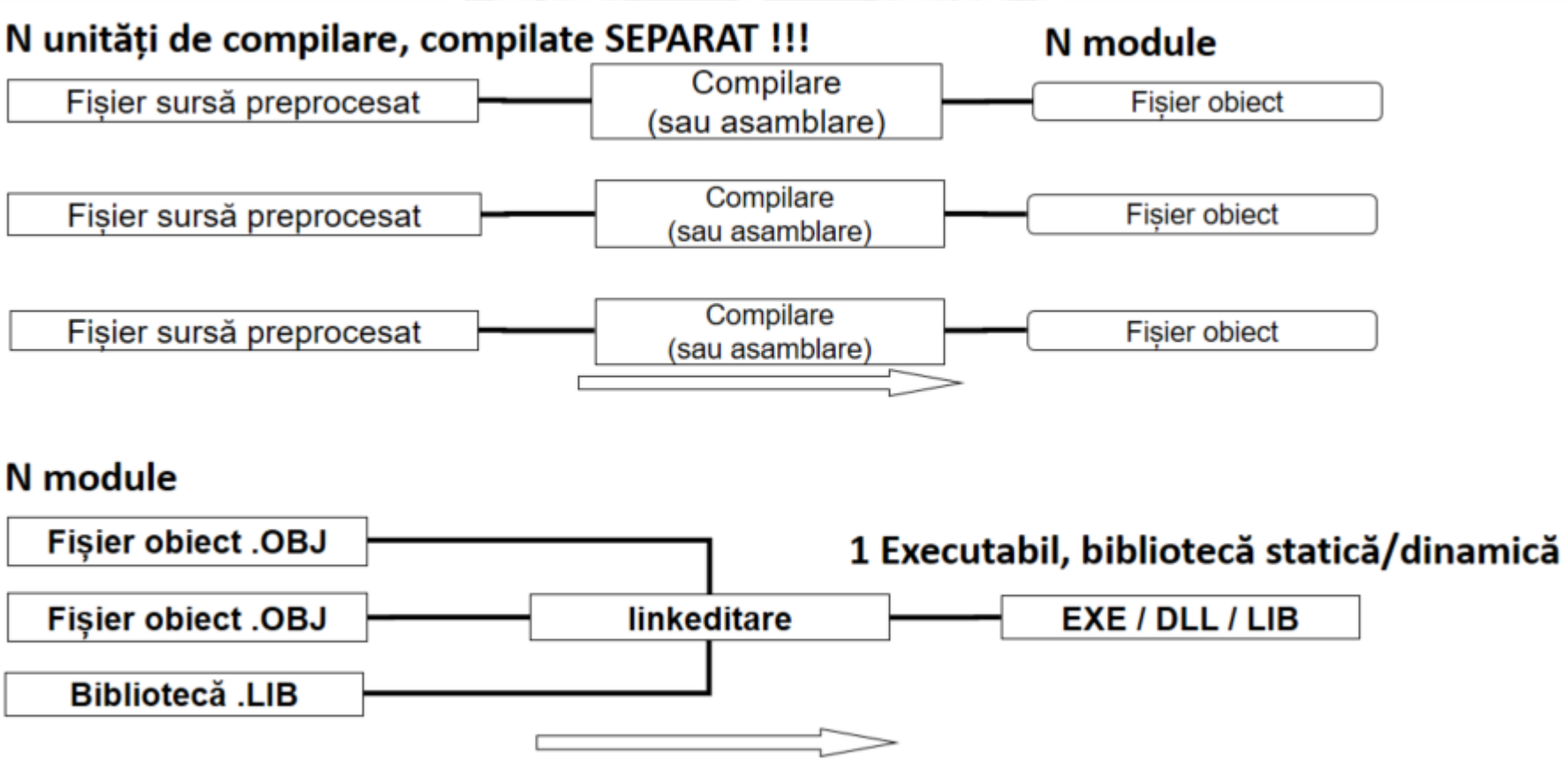
- Specifică limbajului (dar are echivalent și în alte limbaje)
- Modularizare: permite doar divizarea codului scris în acel limbaj
  - NU este programare multimodul, deoarece aceasta necesită COMPILARE SEPARATĂ
- Reutilizare: expune codul sursă
- Periculos și problematic:
  - Mecanism de preprocesor -> concatenare textuală a fișierelor
  - Expune cu vizibilitate globală toate denumirile -> conflicte (redefiniții / redeclarări)
  - Include fișierul în întregime - și ce se folosește, și ce nu



Tehnici și instrumente

Legarea statică la linkeditare:

- Pas realizat de către un linkeditor după asamblare / compilare



Tehnici și instrumente

Legarea statică la linkeditare – sumar responsabilități

- **Preprocesor: text => text**
  - Efectuează prelucrări asupra textului sursă, rezultând un text sursă intermediar
  - Se poate imagina ca fiind o componentă a compilatorului sau asamblorului
  - Poate lipsi, multe limbaje nu au un preprocesor
- **Asamblor: instrucțiuni (text) => codificare binară (fișier obiect)**
  - Codifică instrucțiunile și datele (variabilele) din textul sursă preprocesat și construiește un fișier obiect ce conține cod mașină și valori de variabile alături de informații despre conținut (denumiri de variabile, subrutine, informații despre tipul și vizibilitatea acestora, etc.)

Tehnici și instrumente

Legarea statică la linkeditare – sumar responsabilități

- **Compilator: instrucțiuni (text) => codificare binară (fișier obiect)**
  - Identifică secvențe de instrucțiuni de procesor prin care se pot obține funcționalitățile descrise în textul sursă, iar apoi, precum un asamblor, generează un fișier obiect ce conține codificarea binară a acestora și a variabilelor din program
  - Asamblarea este un caz special de compilare, unde instrucțiunile de procesor sunt gata oferite direct în textul programului și ca atare nu necesită să fie alese de către compilator
- **Linkeditor: fișiere obiect => bibliotecă sau program**
  - Construiește rezultatul final, adică un program (.exe) sau bibliotecă (.dll sau .lib) în care leagă împreună (include) codul și datele binare prezente în fișierele obiect
  - Nu este interesat în ce compilatoare sau ce limbaje au fost folosite! Legarea necesită doar ca fișierele de intrare să respecte formatul standard al fișierelor obiect!

Tehnici și instrumente

Exemplu folosire `%include`

```
; fișierul constante.inc

; gardă dublă-includere
%ifndef _CONSTANTE_INC_  ; la prima includere _CONSTANTE_INC nu este definit
%define _CONSTANTE_INC_  ; definim _CONSTANTE_INC_ -> condiție falsă la
                           viitoare includeri

; recomandat ca astfel de fișiere (incluse de către altele) să conțină (doar) declarații

MAX_BYTE equ 0xFF
MAX_WORD equ 0xFFFF
MAX_DWORD equ 0xFFFFFFFF
MAX_QWORD equ 0xFFFFFFFFFFFFFFFF
%endif ; _CONSTANTE_INC_
```

Tehnici și instrumente

Exemplu folosire `%include`

- impachetare EAX într-un BYTE / WORD / DWORD, conform magnitudinii valorii acestuia

```
;fișierul program.asm
%include "constante.inc"
CMP EAX, MAX_BYTE
JA .nu_incape_in_octet                  ; incape valoarea din EAX într-un BYTE?
.incape_in_octet:
      MOV [rezultat_octet], AL          ; dacă da, salvăm AL în rezultat octet
      JMP .gata
.nu_incape_in_octet:
      CMP EAX, MAX_WORD
      JA .nu_incape_in_cuvant           ; altfel verificăm dacă ajunge un WORD
.incape_in_cuvant:
      MOV [rezultat_word], AX           ; dacă da, salvăm AX în rezultat_word
      JMP .gata
.nu_incape_in_cuvant:
      MOV [rezultat_dword], EAX         ; dacă nu ajunge un WORD, salvăm întreg EAX
.gata:
.
```

Tehnici și instrumente

Legarea statică la linkeditare – cerințele nasm

- Resursele sunt partajate de comun acord
- Export prin `global nume1, nume2, ...`
  - Ofer disponibilitate oricărui fișier ar fi interesat
- Import prin `extern nume1, nume2, ...`
  - Solicit acces, indiferent din ce fișier va fi oferită resursa
- Solicitare fără disponibilitate = eroare!
  - Nu se pot importa decât resurse ce sunt exportate undeva
- Însă disponibilitate fără solicitare este caz permis. De ce?
  - Răspuns: chiar dacă nici un modul din program nu solicită / folosește, poate se va utiliza într-o versiune viitoare sau de către alt program
- Limbajele de programare de nivel mai înalt oferă și ele la rândul lor construcții sintactice cu rol echivalent!
  - Exemplu: În limbajul C
    - Disponibilitatea este automată / implicită, putându-se însă opta pentru a bloca accesul prin folosirea cuvântului cheie **static**
    - Solicitarea de acces se face (tot) prin intermediul cuvântului cheie **extern**

Tehnici și instrumente

Legarea statică la linkeditare

- Permite unirea mai multor **module binare** (fișiere obiect sau biblioteci statice) într-un singur fișier
  - Intrări: oricâte fișiere obiect (**.obj**) și/sau biblioteci statice (**.lib**)
    - Atenție! Nu toate fișierele .LIB sunt biblioteci statice!
  - Ieșire: .EXE sau .LIB sau .DLL (Dynamic-Link Library)
- Multimodul: oricâte fișiere pot fi asamblate / compilate separat și linkeditate împreună
  - Pas realizat de linkeditor după compilare / asamblare -> **nu depinde de limbaj!**
- Reutilizare:
  - În formă binară – nu expune codul sursă!
  - Permite inter-operabilitate între limbaje diferite!
- Alte avantaje și dezavantaje:
  - Editorul de legături poate identifica și elimina resurse neutilizate sau efectua alte optimizări
  - Dimensiune mare a programului: programul înglobează resursele externe reutilizate
  - Dimensiune mare a programelor: bibliotecile populare duplicate în multe programe
- NASM: directivele **global (mecanism export)** și **extern (mecanism import)**
  - Global nume – oferirea posibilității de utilizare din exterior a acestei resurse date prin nume
  - Extern nume – solicitare de acces la resursa specificată; necesită să fie publică

Tehnici și instrumente

- Modularizarea codului în asamblare
  - Procedură – secvență de instrucțiuni care să poată fi apelată din alte zone ale programului și care după terminarea ei returnează controlul programului apelant
  - Este necesar ca la apelul unei proceduri să se salveze adresa de revenire în stiva de execuție
  - Revenirea din procedură este de fapt o instrucșiune de salt la adresa de revenire

```
call eticheta

ret [n]
```

Tehnici și instrumente

Legarea statică la linkeditare - cerințele NASM

- Folosirea în practică a directivelor global și extern

```
; FIȘIER1.ASM
global Var1, Subrutina2
extern Var3, Subrutina3

Subrutina1:

      ...
      Apel (Subrutina3)
      ...
      Operatii (Var3)
      ...
Subrutina2:
      ...
Var1 dd ...

Var2 db ...
```

```
; FIȘIER2.ASM
extern Var1, Subrutina2
global Var3, Subrutina3

Subrutina3:

      ...
      Apel (Subrutina2)
      ...
      Operatii (Var1)
      ...
Subrutina1:
      ...
Var2 dd ...

Var3 db ...
```

Tehnici și instrumente

- Exemplu program multimodul nasm + nasm
  - Pașii necesari construirii programului executabil final
    - Se asamblează fișierul main.asm
      `nasm.exe -fobj main.asm`
    - Se asamblează fișierul sub.asm
      `nasm.exe -fobj sub.asm`
    - Se editează legăturile dintre cele două module
      `alink.exe main.obj sub.obj -oPE -entry start -subsys console`
  - Observație: cele două module pot fi asamblate în orice ordine! Abia în timpul linkeditării este necesar ca simbolurile referite să aibă toate implementare disponibilă în unul dintre fișierele obiect oferite linkeditorului
  - Linkeditarea, în mod evident, este posibilă doar după asamblare / compilare

Tehnici și instrumente

Legarea statică la linkeditare - cerințele NASM

- Folosirea în practică a directivelor global și extern

```
; FIȘIER1.ASM
global Var1, Subrutina2
extern Var3, Subrutina3

Subrutina1:
        ...
        Apel (Subrutina3)
        ...
        Operatii (Var3)
        ...
Subrutina2:
        ...
Var1 dd ...

Var2 db ...
```

```
; FIȘIER2.ASM
extern Var1, Subrutina2
global Var3, Subrutina3

Subrutina3:
        ...
        Apel (Subrutina2)
        ...
        Operatii (Var1)
        ...
Subrutina1:
        ...
Var2 dd ...

Var3 db ...
```

Pot fi refolosite denumirile, cât timp nu sunt globale!

Tehnici și instrumente

Legarea statică la linkeditare - cerințele NASM

- MODALITĂȚI DE TRANSMITERE A ARGUMENTELOR ȘI DE OBȚINERE A REZULTATULUI
  - Argumente: registru, Rezultat: registru
  - Argumente: stivă, Rezultat: registru
  - Argumente: stivă, Rezultat: stivă

Tehnici și instrumente

Exemplu program multimodul nasm + nasm

```
; MODULUL MAIN.ASM
global SirFinal
extern Concatenare

import printf msvcrt.dll
import exit msvcrt.dll
extern printf, exit
global start


segment code use32 public code class='code'
  start:
        mov eax, Sir1
        mov ebx, Sir2
        call Concatenare
        push dword SirFinal
        call [printf]
        add esp, 1*4
        push dword 0
        call [exit]

segment data use32

  Sir1 db 'Buna ',0
  Sir2 db 'dimineata!',0
  SirFinal resb 1000 ; spațiu pentru rezultat
```

```
; MODULUL SUB.ASM
extern SirFinal
global Concatenare

segment code use32 public code class='code'
; eax = adresa primului șir
; ebx = adresa șirului secund

Concatenare:
  mov edi, SirFinal ; destinație = sirFinal
  mov esi, eax      ; sursa = primul șir

.sir1Loop:

  lodsb             ; luăm octetul următor
  test al,al        ; este terminatorul de șir (=0)?
  jz .sir2          ; dacă da trecem la șirul al doilea
  stosb             ; altfel copiem în destinație
  jmp .sir1Loop     ; și continuăm până la nul

.sir2:

  mov esi, ebx      ; sursa = șirul al doilea

.sir2Loop:

  lodsb             ; același proces pentru noul șir
  test al,al
  jz.gata
  stosb
  jmp .sir2Loop

.gata:

  stosb             ; adăugăm terminatorul de șir din al
  ret
```

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
UNIVERSITATEA BABEȘ-BOLYAI

Str. Mihail Kogălniceanu nr. 1
Cluj-Napoca, Cluj, România

www.cs.ubbcluj.ro