SISTEME DE OPERARE (SO)

CURS 6

Lect. Univ. Dr. Mihai Stancu

Compilarea programelor

Departamentul de Informatică

- Suport (Introducere în sisteme de operare)
  - Capitolul 11 – Compilare și linking

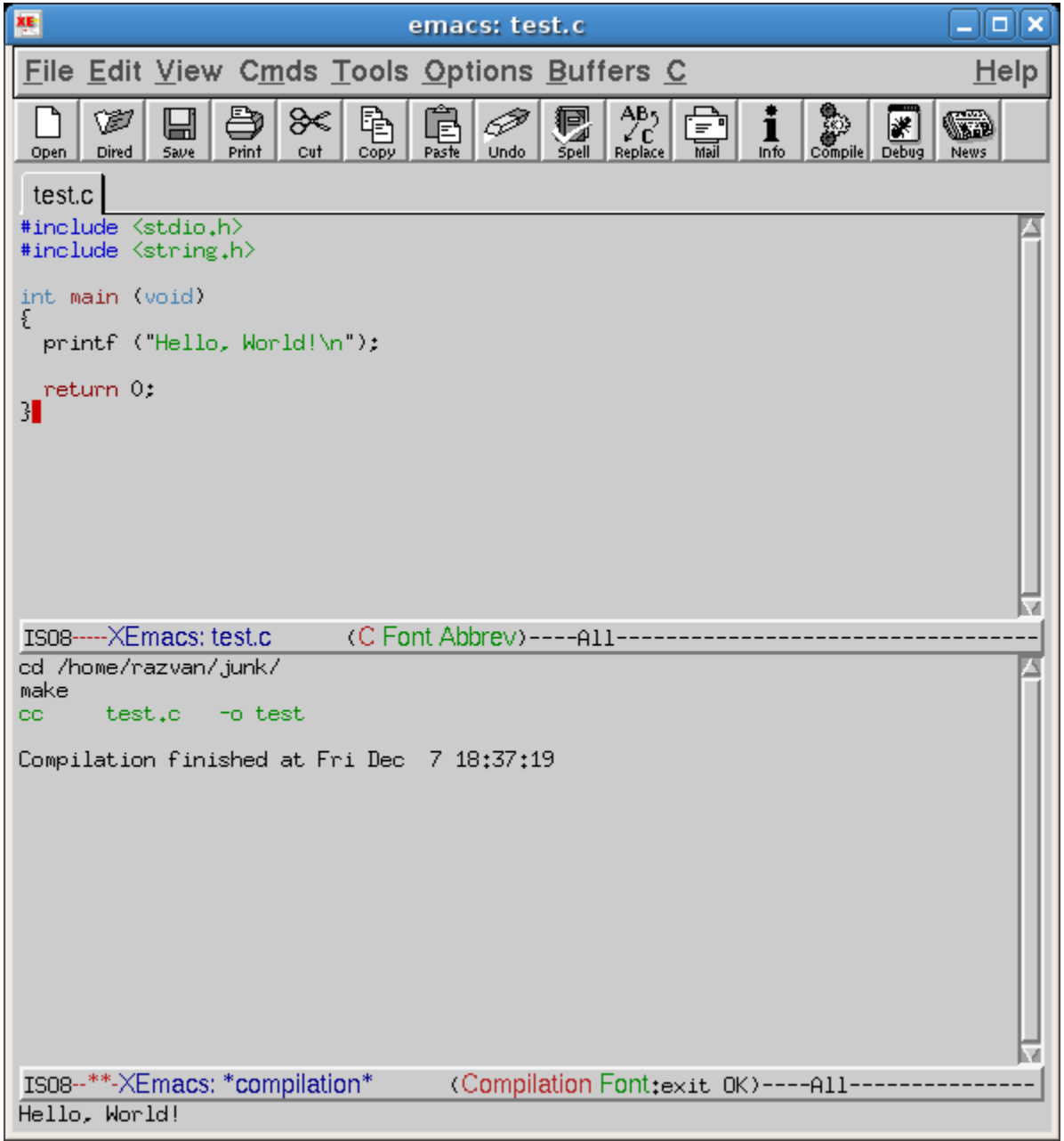
Programe

- Ce este un program?
  - fișier executabil binar (conținutul este cod mașină)
- Care este faza inițială în care se găsește un program?
  - fișier cod sursă – fișier text ce conține implementarea programului într-un limbaj de programare
    - C – app.c, app.h
    - C++ – app.cpp, app.cxx, app.C, app.hpp
    - Java – Application.java
    - Perl, Python, Lisp etc.
  - se mai numește simplu sursă (sursă C, sursă Java, etc.)
  - scriere de cod sursă sau scriere de cod
- la început, programele erau scrise direct în cod mașină (binar) (cartele perforate)

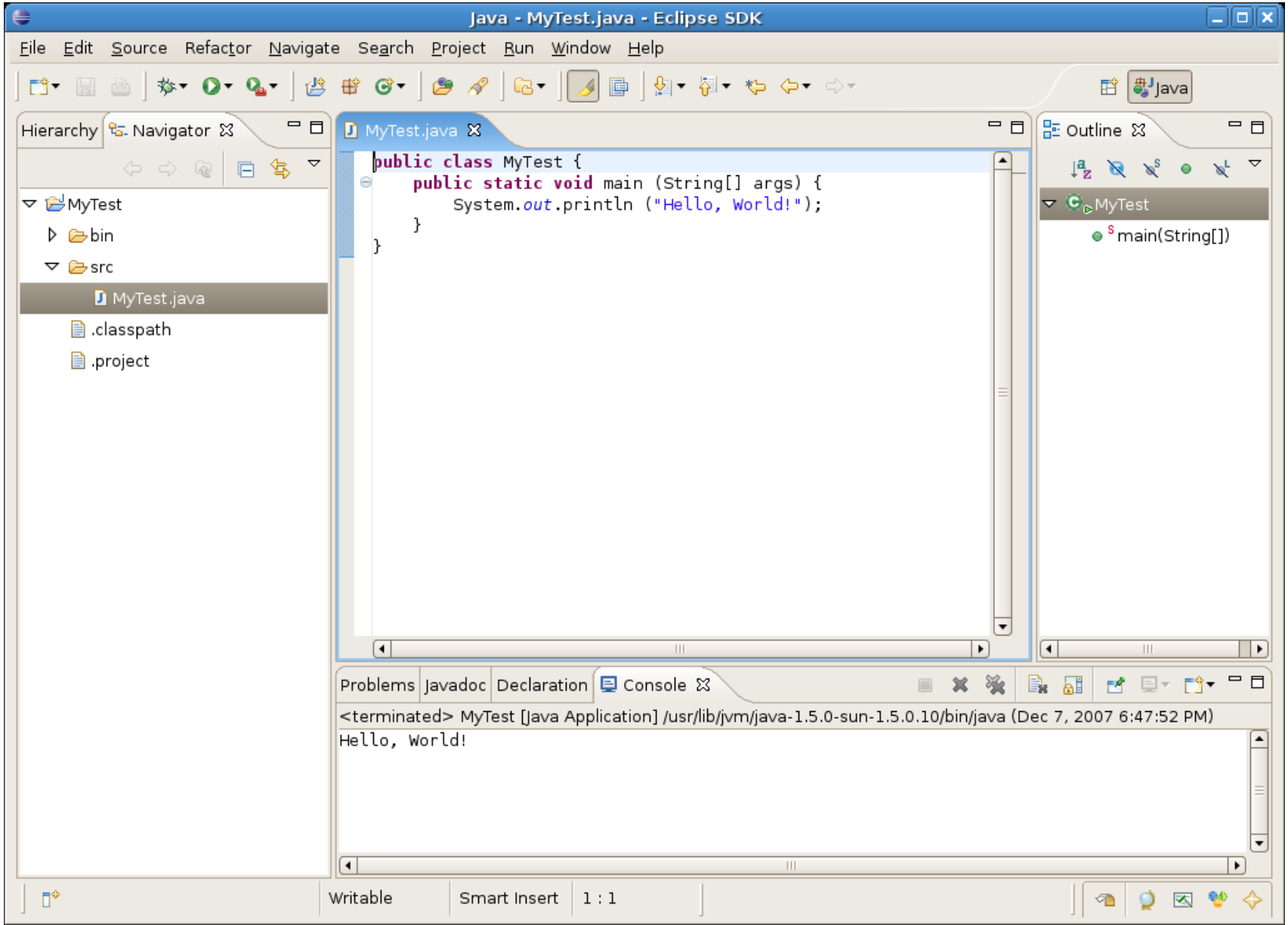
Editoare

- scrierea de fișiere text (în particular cod sursă)
- editoare
  - simple – one tool for the job
    - vim, Emacs, nano, joe
    - notepad++, notepad2, UltraEdit, Crimson Editor
  - integrate (IDE)
    - Visual Studio 2010 (varianta Visual Studio Express Edition este free)
    - Eclipse, NetBeans
    - Emacs, Kdevelop
- facilități
  - syntax higlighting, auto indentation, utilitare pentru debugging integrate
  - code folding, code completion (autocompletion)

EMACS


emacs: test.c
File Edit View Cmds Tools Options Buffers C Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News
test.c
#include <stdio.h>
#include <string.h>
int main (void)
{
printf ("Hello, World!\n");
return 0;
}
ISO8----XEmacs: test.c (C Font Abbrev)----All----------
cd /home/razvan/junk/
make
cc test.c -o test
Compilation finished at Fri Dec 7 18:37:19
ISO8--**-XEmacs: *compilation* (Compilation Font:exit OK)----All----------
Hello, World!

Eclipse


Java - MyTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Hierarchy
Navigator
MyTest
bin
src
MyTest.java
.classpath
.project
MyTest.java
public class MyTest {
public static void main (String[] args) {
System.out.println ("Hello, World!");
}
}
Outline
MyTest
main(String[])
Problems
Javadoc
Declaration
Console
<terminated> MyTest [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0.10/bin/java (Dec 7, 2007 6:47:52 PM)
Hello, World!
Writable
Smart Insert
1 : 1

Compilare si interpretare

- un fișier cod sursă poate fi compilat sau interpretat
- deosebirea între compilare și interpretare
  - compilare: codul sursă este translatat de un program denumit compilator în cod mașină, după care poate fi executat
  - interpretare: un program este executat direct din cod sursă prin intermediul unui interpretor
- multe limbaje au atât compilatoare, cât și interpretoare: C, Lisp
- GCC (GNU Compiler Collection) – compilator de C, C++, Ada, Fortran
- MSVC (Microsoft Visual C) – compilator de C, C++
- GCL (GNU Common Lisp) – interpretor de Common Lisp

Compilare si interpretare – continuare

- limbaje tradițional compilate
  - C, C++, Objective-C, Pascal, Java, Ada
- limbaje tradițional interpretate (limbaje de scripting)
  - Perl, PHP, Python, Lisp, Ruby, shell scripting (bash, csh, ksh)
- avantaje/dezavantaje
  - interpretare
    - limbaje, în general, mai ușor de înțeles de programator
    - debugging facil
    - execuție lentă
  - compilare
    - debugging greoi (folosirea unui depanator – debugger)
    - viteză ridicată de execuție

Compilare

- un compilator este un translator: translatează codul sursă în cod obiect
  - de la un limbaj de nivel înalt în cod mașină
- un compilator va genera de obicei un executabil

Compilare în Windows

```
F:ncode>cl simple.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version
14.00.50727.42 for 80x86
Copyright (C) Microsoft Corporation.
All rights reserved.
simple.c
Microsoft (R) Incremental Linker Version 8.00.50727.42
Copyright (C) Microsoft Corporation.
All rights reserved.
/out:simple.exe
simple.ob
```

Compilare – continuare

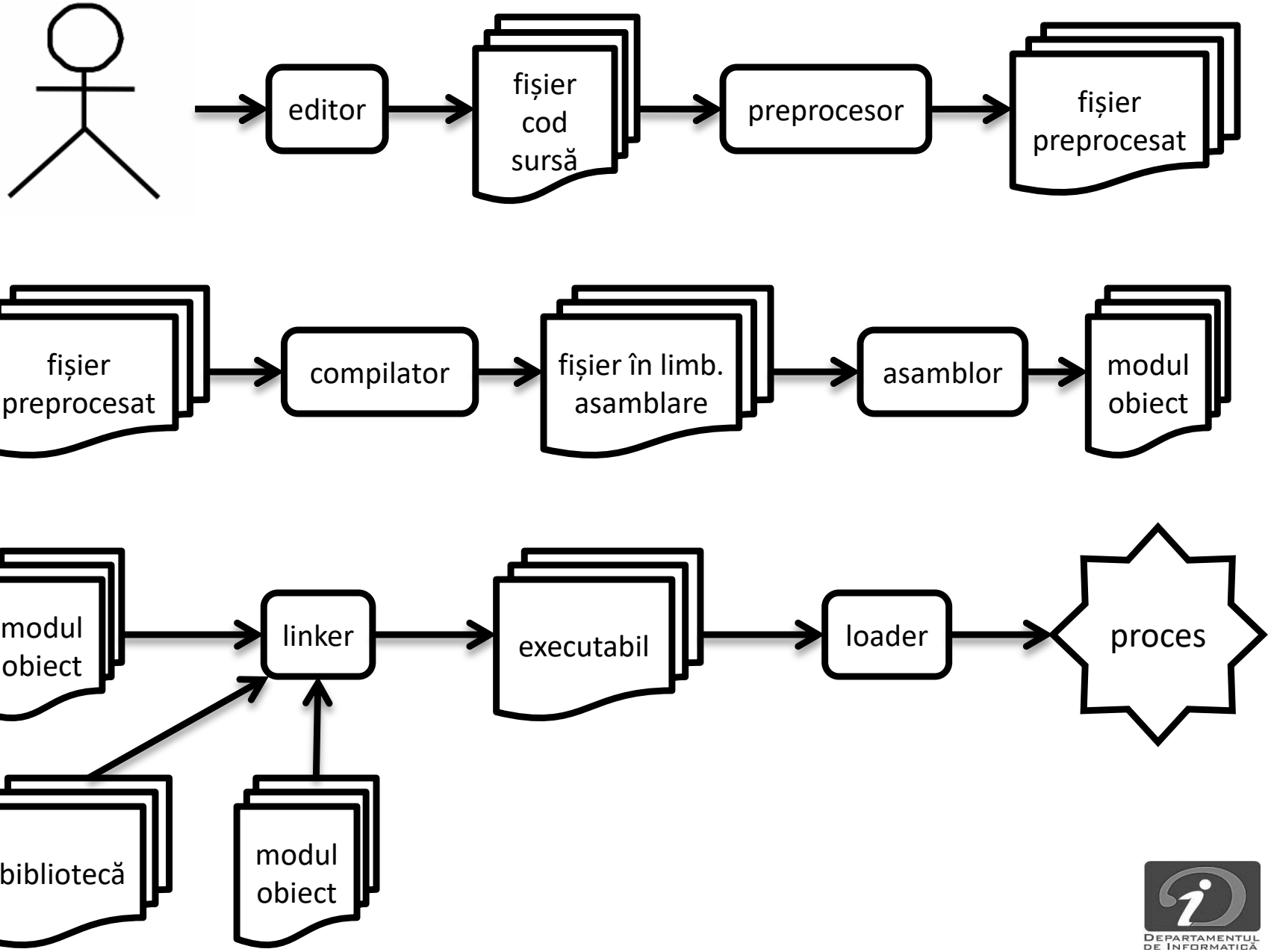
Compilare în Linux

```
alin@anaconda:~/code/hello$ ls
hello.c
alin@anaconda:~/code/hello$ gcc hello.c
alin@anaconda:~/code/hello$ ls
a.out hello.c
alin@anaconda:~/code/hello$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=4b991036ec68778ff5f3c95309f047d4344cd6ef, not
stripped
```

Fazele compilarii

- componentă/utilitar specializat pentru fiecare fază
- preprocesare: `cpp`
  - înlocuirea directivelor de preprocesare (`#include, #define`)
  - nu este specific tuturor limbajelor
- compilare efectivă: `gcc`
  - translatarea codului sursă (preprocesat) în limbaj de asamblare
- asamblare: `as`
  - translaterea fișierului în limbaj de asamblare într-un modul obiect (cod mașină)
- linking: `ld`
  - legarea mai multor module obiect și biblioteci într-un modul de sine stătător (de obicei un fișier executabil)

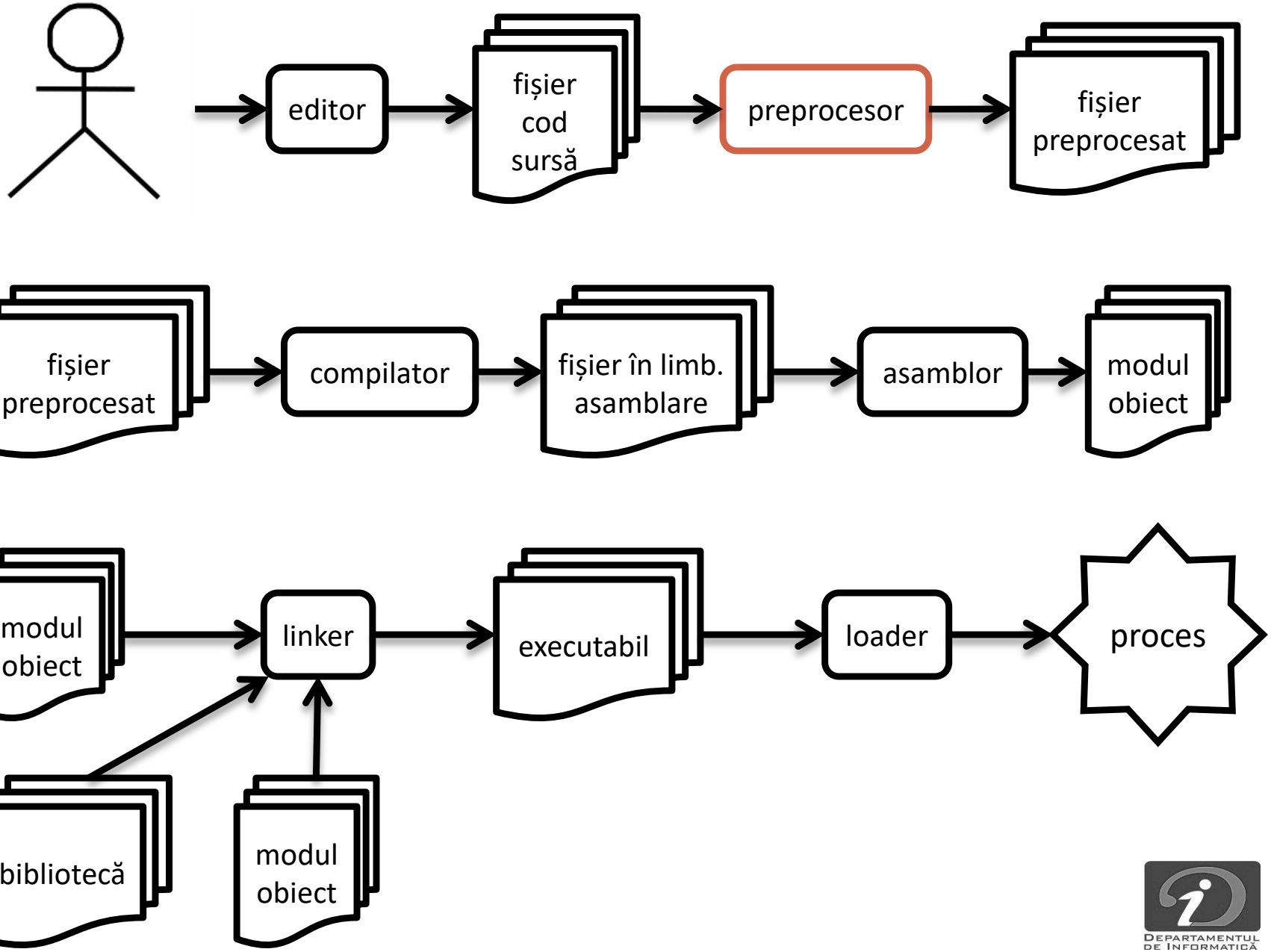
Fazele compilarii



Fisier cod sursa

- componenta vizibilă programatorului
- declarații de variabile, funcții, clase etc.
- sintaxa impusă de limbajul respectiv
- fișier text (într-un limbaj de programare)
  - `memcap.c`
  - `btdownloadheadless.py`
  - `checkpatch.pl`
  - `dispatch.rb`
- De ce i se spune **cod** sursă?
  - pentru că nu este limbaj natural; nu este ușor inteligibil

Fazele compilarii – preprocesare



Preprocesare

- Fișier cod sursă

```
razvan@asgard:~/code$ cat simple.c
#include <stdio.h>

#define ITERATIONS      10
#define INITIAL         1
#define HEADER_STRING   "Number sum: "

int main (void)
{
    int i;
    int sum;

    printf (HEADER_STRING);
    for (i = INITIAL; i < INITIAL + ITERATIONS;
    i++)
            sum += i;

    printf ("%d\n", sum);
    return 0;
}
```

- Fișier preprocesat

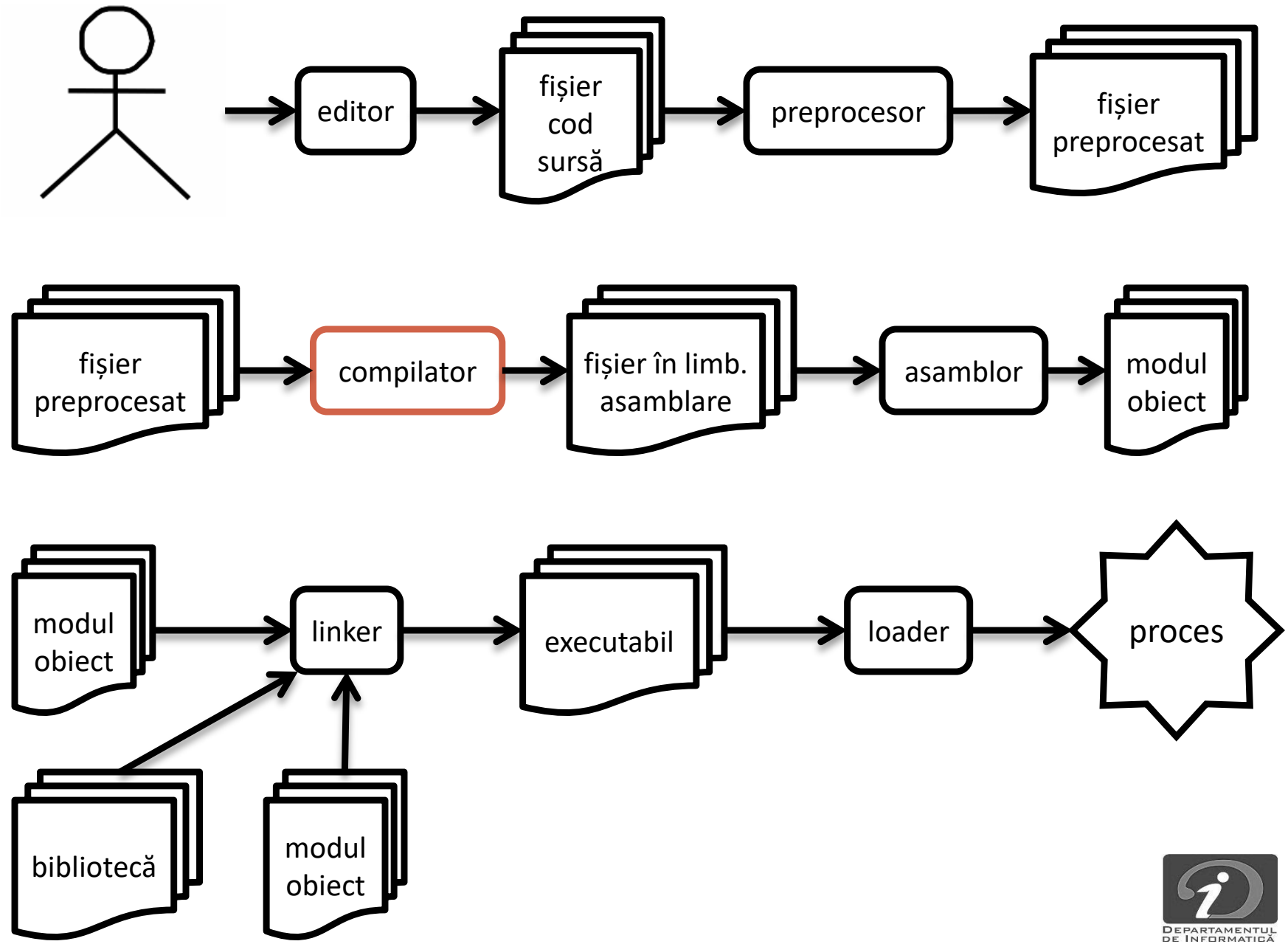
```
razvan@asgard:~/code$ gcc -E -o simple.i simple.c
razvan@asgard:~/code$ cat simple.i
# 1 "simple.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "simple.c"
# 1 "/usr/include/stdio.h" 1 3 4
..............

int main (void)
{
        int i;
        int sum;

        printf ("Number sum: ");
        for (i = 1; i < 1 + 10; i++)
                sum += i;

        printf ("%d\n", sum);
        return 0;
}
```

Fazele compilarii – compilare



Compilare

- ține cont de sintaxa și semantica programului
  - sintaxă: programul respectă un set de reguli (o anumită gramatică)
    - operatorul '+' poate avea doi operanzi sau poate avea unul singur
  - semantică: instrucțiunile din program trebuie să aibă sens
    - se adună un număr cu un alt număr, nu un număr cu un șir de caractere
- se obține fișier in limbaj de asamblare
- program incorect din punct de vedere sintactic sau semantic –> erori la compilare:

Erori de compilare

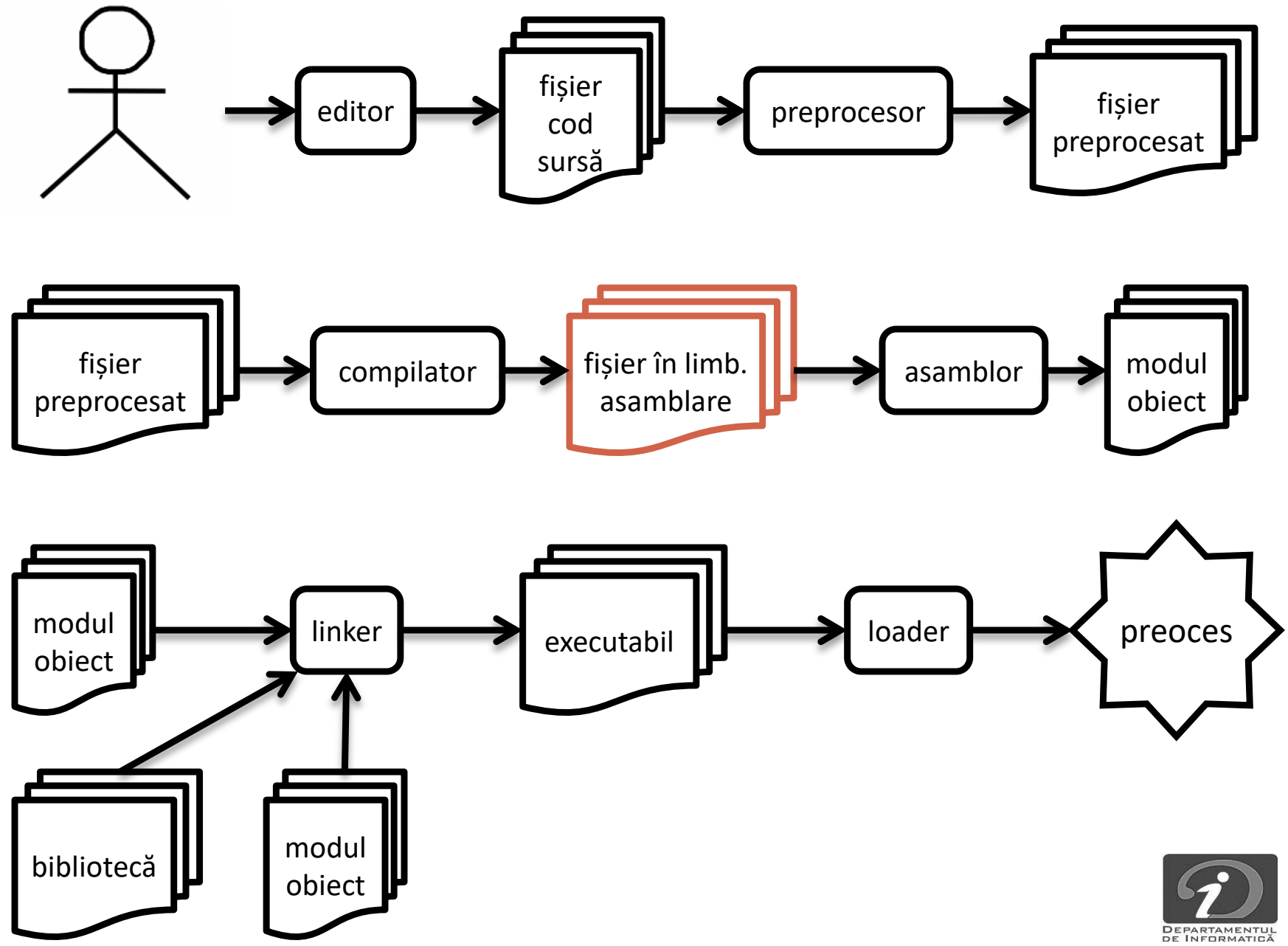
```
alin@asgard:~/code$ gcc main.c
main.c: In function 'main':
main.c:6: error: 'i' undeclared (first use in this function)
main.c:6: error: (Each undeclared identifier is reported only once
main.c:6: error: for each function it appears in.)
```

Compilare – continuare

```
#include <stdio.h>
#define ITERATIONS 10
#define INITIAL 1
#define HEADER_STRING "Number sum:"
int main (void)
{
    int i;
    int sum;
    sum = 0;
    printf (HEADER_STRING);
    for (i = INITIAL; i < INITIAL +
        ITERATIONS; i++)
        sum += i;
    printf ("%dnn", sum);
    return 0;
}
```

```
alin@asgard:~/code$ gcc -S simple.c
alin@asgard:~/code$ cat simple.s
    .file "simple.c"
    .section .rodata
.LC0:
    .string "Number sum: "
.LC1:
    .string "%dnn"
    .text
.globl main
    .type main, @function
main:
    pushl %ebp
    movl %esp, % ebp
    subl $24, %esp
    andl $-16, %esp
    ...
    movl $0, -8(%ebp)
    movl %,LC0, (%esp)
    call printf
    ...
```

Fazele compilarii – limbaj de asamblare



Limbaj de asamblare

- formă de reprezentare a instrucțiunilor procesorului
- folosește mnemonici (movl, cmpl, addl)
- movl $0, %eax ; stocheaza valoarea 0 in eax
- extensia .asm (Windows) sau .s (Unix)
- o instrucțiune în limbaj de asamblare
  - operatori (add, mul, cmp, mov)
  - operanzi puși la dispoziție de procesor (registre, locații de memorie)

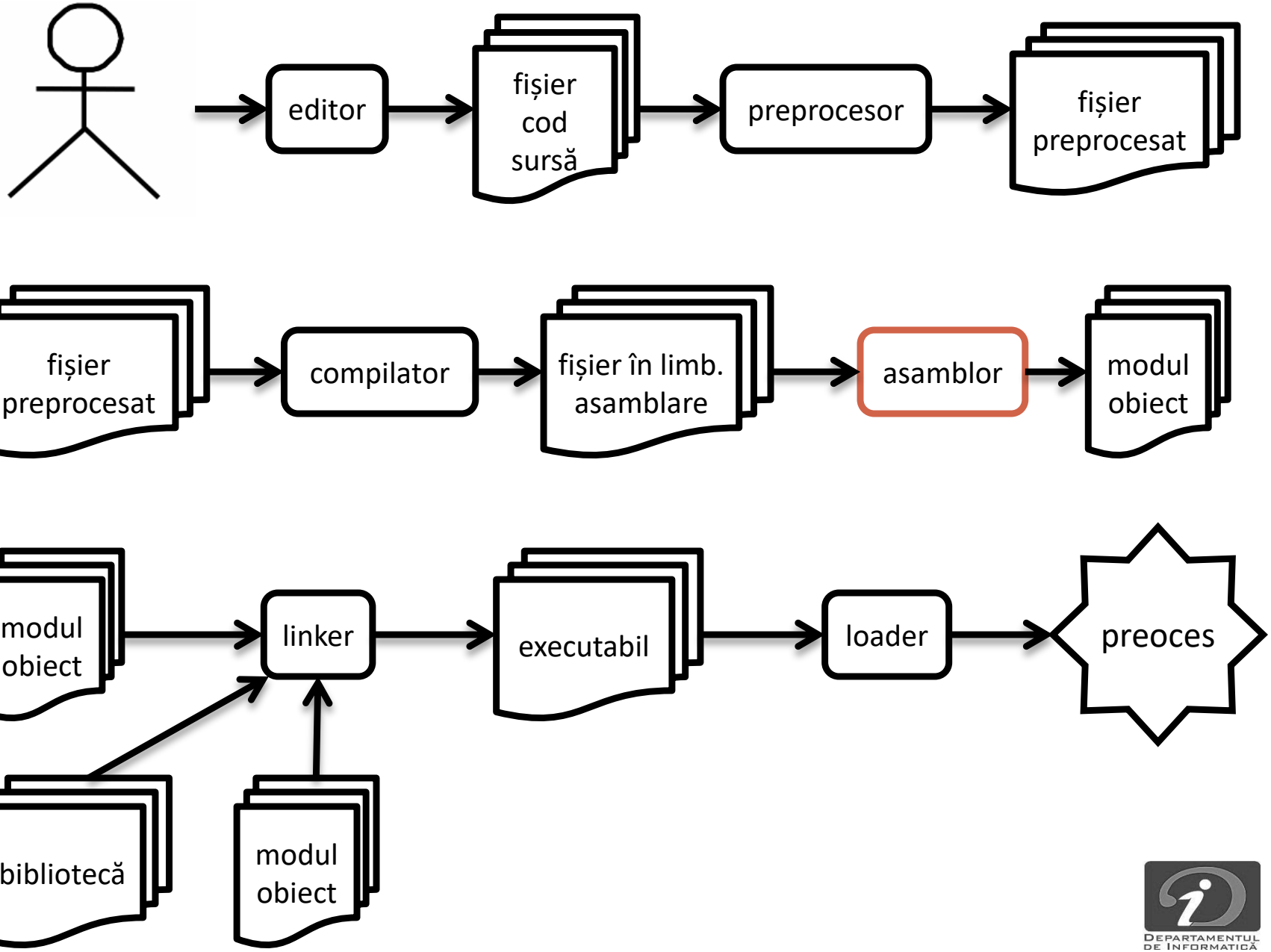
Folosire limbaj de asamblare

- construirea unor module "pure": totul în assembly
- integrare în programe C – inline assembly

```
__asm__ ("movl %0,r9\n\tmovl %1,r10\n\tcall _foo"
: /* no outputs */
: "g" (from), "g" (to)
: "r9", "r10");
```

- limbaj depedent de arhitectura
  - greu extensibil
  - dificil de menținut
- Când folosim limbajul de asamblare?
  - limbajul de nivel înalt nu oferă un suport pentru anumite instrucțiuni ale procesorului
  - anumite părți din cod trebuie optimizate "de mână"

Fazele compilarii – asamblor



Asamblare și dezasamblare

```
alin@asgard:~/code$ as -o simple.o simple.s
alin@asgard:~/code$ objdump --disassemble simple.o
simple.o:
file format elf32-i386
Disassembly of section .text:
00000000 <main>:
0: 55 push %ebp
1: 89 e5 mov %esp,%ebp
3: 83 ec 18 sub $0x18,%esp
6: 83 e4 f0 and $0xfffffff0,%esp
9: b8 00 00 00 00 mov $0x0,%eax
e: 29 c4 sub %eax,%esp
10: c7 45 f8 00 00 00 00 movl $0x0,0xfffffff8(%ebp)
17: c7 04 24 00 00 00 00 movl $0x0,(%esp)
1e: e8 fc ff ff ff call 1f <main+0x1f>
23: c7 45 fc 01 00 00 00 movl $0x1,0xfffffffc(%ebp)
...
4f: e8 fc ff ff ff call 50 <main+0x50>
54: b8 00 00 00 00 mov $0x0,%eax
59: c9 leave
5a: c3 ret
```

Asamblare – continuare

- translatarea fișierului din limbaj de asamblare în fișier cod obiect
- instrucțiunile în limbaj de asamblare se traduc una câte una în echivalentul lor binar (cod mașină)
- asamblor

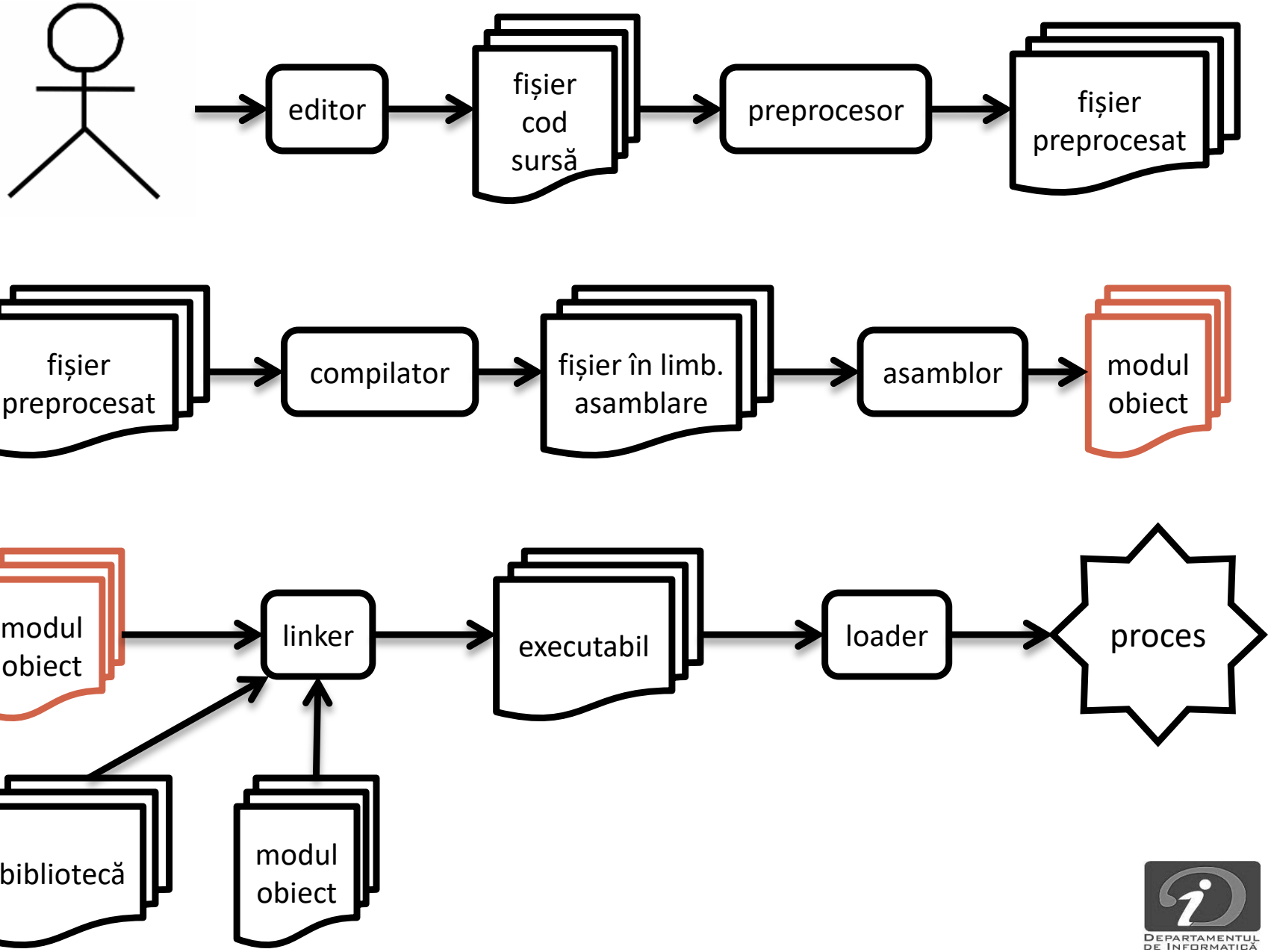
Exemplu rulare `as`

```
alin@anaconda:~/code$ as -o simple.o simple.s
```

Exemplu rulare `cl` (MSVC)

```
F:\code>cl /Fasimple.asm simple.c
```

Fazele compilarii – modul obiect



Fisiere cod obiect

- extensia .obj (Windows) sau .o (Unix)
- conțin, codificate, variabilele (zona de date) și instrucțiunile programului (zona de cod)

objdump – investigarea unui modul obiect

```
alin@asgard:~/code$ objdump -s simple.o
simple.o: file format elf32-i386
Contents of section .text:
0000 5589e583 ec1883e4 f0b80000 000029c4 U.............).
0010 c745f800 000000c7 04240000 0000e8fc .E.......$......
[...]
Contents of section .rodata:
0000 4e756d62 65722073 756d3a20 0025640a Number sum: .%d.
[...]
```

nm – identficarea simbolurilor dintr-un modul obiect

```
alin@asgard:~/code$ nm simple.o
00000000 T main
U print
```

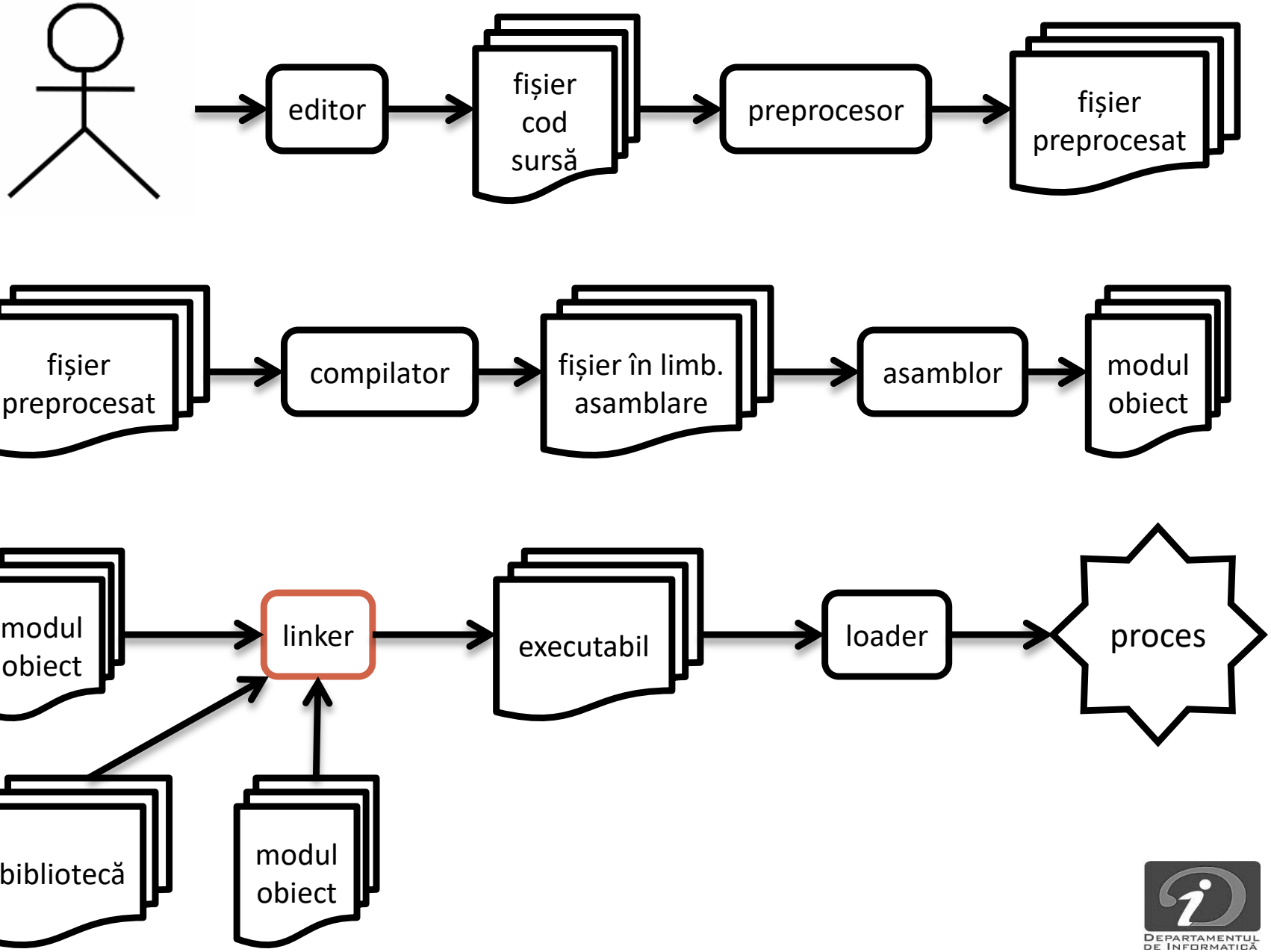
Biblioteci

- bibliotecă = library ≠ librărie
- fișier ce conține o colecție de funcții des utilizate de alte programe
  - de fapt, o colecție de module obiect (fișiere binare)
- nu sunt independente (precum executabilele)
  - module ce pot ajuta la crearea unui executabil
- în prezent, o parte importantă din codul programelor se regăsește în biblioteci
  - libc (biblioteca standard C), libpurple (Pidgin), libpng
  - /lib/*, /usr/lib/*

Biblioteci – continuare

- biblioteci statice (*static libraries*)
  - funcțiile de bibliotecă apelate de un program sunt atașate codului executabil la linking
  - codul executabil se mărește
  - eventuale modificări ale bibliotecii din sistemul de operare nu afectează în nici un fel programul
- biblioteci partajate (*shared object/dynamic libraries*)
  - la linking se creează referințe către funcțiile apelate fără a fi incluse efectiv în executabil
  - biblioteca va fi încărcată în momentul lansării în execuție a programului (*load time*) sau în timp ce rulează (*run time*)
  - cod executabil minim
  - mare parte din codul bibliotecii este partajat între procese
  - este nevoie de recompilare dacă modificăm biblioteca
  - extensia `.so` în Linux și `.dll` în Windows

Fazele compilarii – linker



Linking

- mai multe module obiect (inclusiv biblioteci) sunt grupate într-un modul de sine stătător (de obicei executabil)
- la legare/linking se rezolvă referințele
  - un modul apelează o funcție (sau o variabilă) dintr-un alt modul
  - în urma compilării se creează o referință către acea funcție (fără a se ști în ce modul se găsește)
  - linker-ul rezolvă aceste referințe, găsind funcțiile (variabilele) apelate din alte module
  - dacă se apelează o funcție dintr-un modul ce nu se regăsește la linking, apare eroare la linking

Crearea de biblioteci

- fișierul add.c implementează funcția add, iar sub.c implementează funcția sub
- biblioteca va conține cele două module

Crearea și legarea unei biblioteci statice

```
alin@asgard:~/code$ gcc -Wall -c add.c
alin@asgard:~/code$ gcc -Wall -c sub.c
alin@asgard:~/code$ ar cr libsimple.a add.o sub.o
alin@asgard:~/code$ ar t libsimple.a
add.o
sub.o
alin@asgard:~/code$ gcc -Wall -L. -o lib_main lib_main.c -lsimple
```

Crearea și legarea unei biblioteci partajate

```
alin@asgard:~/code$ gcc -Wall -fPIC -c add.c
alin@asgard:~/code$ gcc -Wall -fPIC -c sub.c
alin@asgard:~/code$ gcc -fPIC -shared -o libsimple.so add.o sub.c
alin@asgard:~/code$ export LD_LIBRARY_PATH=.
alin@asgard:~/code$ gcc -Wall -L. -o lib_main lib_main.c -lsimple
```

Compilers: Principles, Techniques and Tools

- The Dragon Book
- Aho, Sethi, Ullman
- 2nd Edition, 2006
- cartea de bază pentru toate cursurile de compilatoare din universități
- expunere exhaustivă a analizei sintactice, semnatice și a parserelor

Guido van Rossum

- inventatorul limbajului de programare Python
- Benevolent Dictator for Life (BDFL) pentru Python
- a activat la Google în perioada 2005-2012
- din 2013 lucrează la Dropbox

Google

- google.com – cel mai folosit site din lume
- fondată de Larry Page și Sergey Brin
- lansată în 1998
- inițial: search engine + advertising
- aplicații web
- Android
- Chrome
- YouTube

VALGRIND

- http://valgrind.org
- detectarea de probleme la rulare (runtime)
- în principal folosit pentru probleme de lucru cu memoria
- Linux, Darwin (Mac OS X) și Solaris
- un engine peste care rulează componente dedicate:

  memcheck (implicit), cachegrind, callgrind, helgrind

Cuvinte cheie

- program
- cod sursă
- editor
- IDE
- compilator
- interpretor
- preprocesare
- compilare
- limbaj de asamblare
- asamblare
- cod obiect
- bibliotecă
- linker
- executabil
- `cpp, gcc, gas, ld`
- `objdump, nm`

Resurse utile

- http://en.wikipedia.org/wiki/List_of_text_editors
- http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments
- http://www.microsoft.com/express/
- http://en.wikipedia.org/wiki/Source_code
- http://www.oualline.com/style/index.html
- http://en.wikipedia.org/wiki/Library_(computing)