SYSTEMS LABORATORY

Cursul #4

Dezvoltarea programelor

There are only two kinds of programming languages: those people always bitch about and those nobody uses.

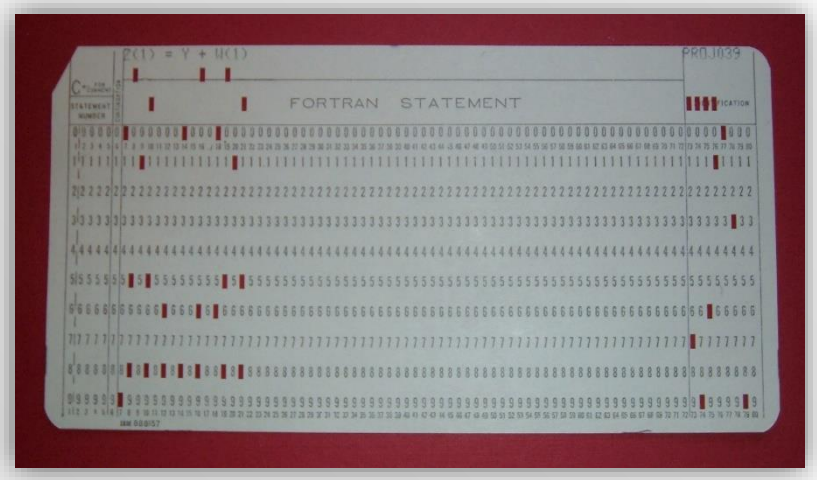
Bjarne Stroustrup

Suport de curs

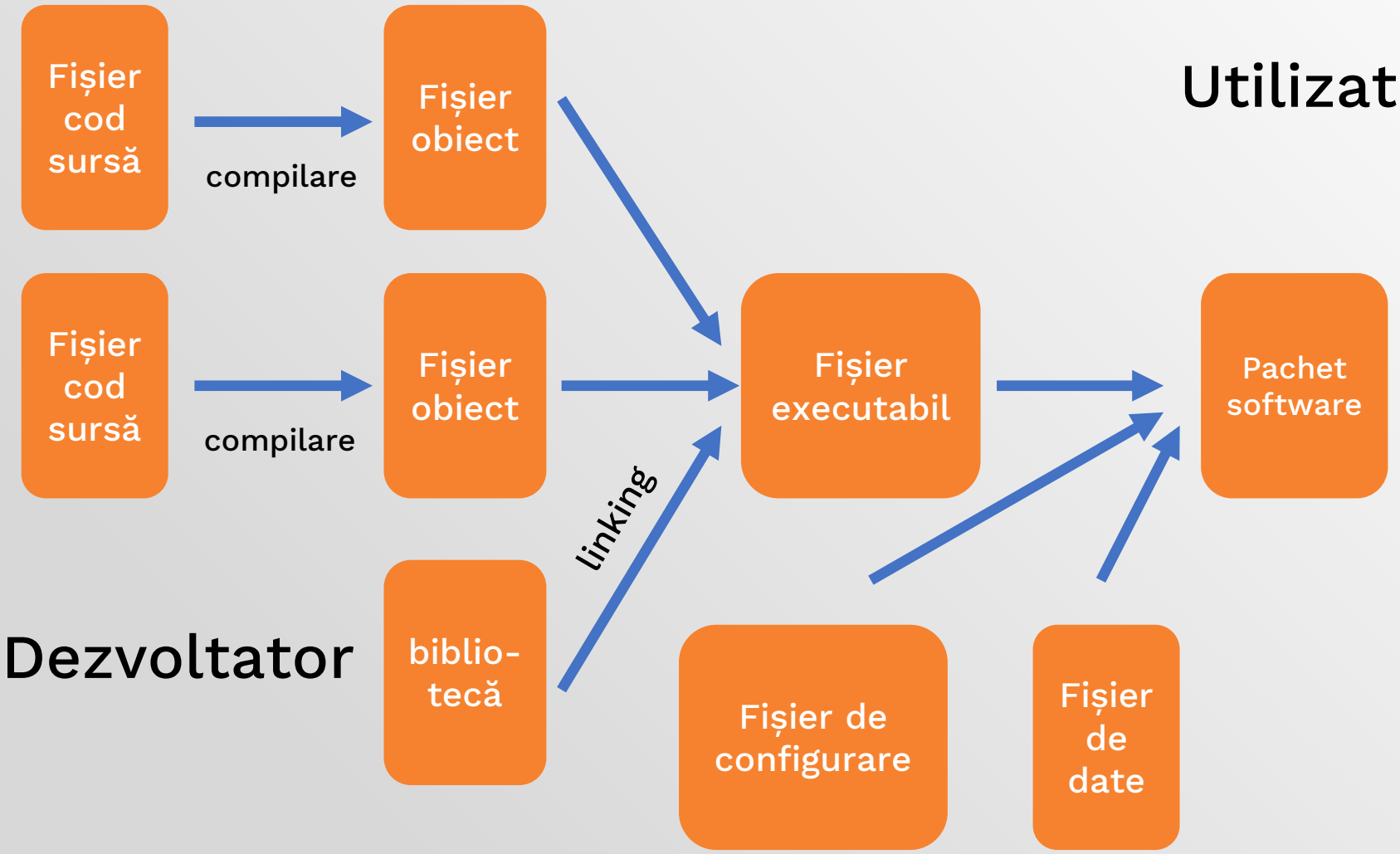
- Capitolul 6 - Dezvoltarea aplicațiilor
  - https://github.com/systems-cs-pub-ro/carte-uso/releases

Evoluția programării

- La început se foloseau benzi magnetice sau perforate pentru a executa operații simple pe mașinile de calcul
- Odată cu evoluția procesoarelor, a apărut limbajul de asamblare
- În prezent, folosim cel mai des limbaje de nivel înalt



Conexiune dezvoltator - utilizator



Pachete software

- Arhive cu fișierele necesare pentru instalarea aplicației
- Fișiere de date, fișiere de configurare, fișiere executabile
- Utilizatorul instalează pachetele software și apoi folosește fișierele din acestea
- dpkg – manager de pachete

```
student@host:~$ dpkg -L tcpdump
[...]
/usr/share/man/man8/tcpdump.8.gz  # pagina de manual
[...]
/usr/sbin/tcpdump                 # executabilul din pachet
```

Operațiile utilizatorului cu aplicațiile

- Instalare
- Dezinstalare
- Parcurgere documentație
- Configurare și personalizare
- Rulare
- Dezvoltatorul trebuie să îi facă viața cât mai ușoară utilizatorului

Fișiere cod sursă

- Conțin instrucțiuni scrise într-un anume limbaj de programare, adesea limbaj de nivel înalt
- Codul citibil al unui limbaj de programare: format text
- Este scris de programator
- **Primul pas în dezvoltarea unui program**
- Aplicația principală necesară: editor de text

Editoare

În mod text:

- Vim
  - Se găsește pe toate sistemele Unix
  - Funcționează în linia de comandă
  - Foarte configurabil
- Nano
- Pico
- Emacs

În interfață grafică:

- Sublime Text
  - Recunoaște limbajele de programare
  - Suportă extensii (ex. Git)
- Atom
  - Asemănător Sublime Text, dezvoltat de GitHub
  - Are sursa deschisă și nu percepe nicio taxă
- Visual Studio Code

Caracteristici editoare

- Indentarea automată a codului
- Evidențierea cuvintelor cheie
- Semnalarea erorilor
- Autocompletion
- Utilitare pentru debugging integrate

Medii integrate de dezvoltare (IDE)

- Asemănătoare cu editoarele de text, acestea au în plus funcționalități avansate, precum faptul că au compilatoare sau interpretoare integrate
- De obicei sunt adaptate pentru un număr redus de limbaje
- Oferă sugestii mai relevante programatorului și metode avansate de a depana programele
- Exemple: Eclipse, CodeBlocks, Microsoft Visual Studio, NetBeans, Xcode

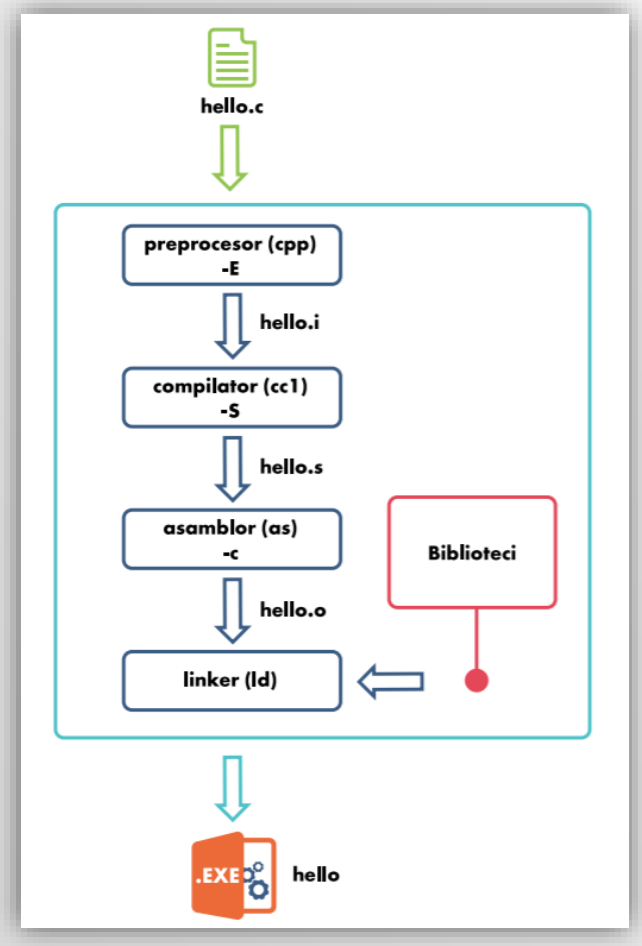
Compilare

- Codul sursă este translatat de un program denumit compilator în cod mașină (machine code), după care poate fi executat
- Exemple compilatoare: GCC – GNU Compiler Collection (pt C/C++,Fortran), MSVC – Microsoft Visual C (pt C/C++)
- Exemple de limbaj de programare compilat: C, C++, Pascal

Etapele compilării

- **Preprocesare:** se înlocuiesc macro-uri, se numerotează fiecare linie de cod; rezultatul e un fișier cu extensia „.i”
- **Compilare:** codul sursă este transformat în cod în limbaj de asamblare; rezultatul e un fișier cu extensia „.s”
- **Asamblare:** codul în limbaj de asamblare este transformat în cod mașină; rezultatul e un fișier cu extensia „.o”
- **Link-editare:** se fac legăturile către fișiere externe care conțin simboluri sau funcții apelate din fișierul sursă; dacă avem o funcție definită într-un fișier și folosită în altul, implementarea funcției trebuie legată de apelul ei

Etapele compilării



Interpretare

- Un program este executat direct din cod sursă prin intermediul unui interpretor
- **Interpretorul** ia fiecare linie, generează codul mașină aferent ei, codul este rulat pe procesor, după care operația este reluată
- Limbajele interpretate nu oferă acces direct la resursele hardware, ceea ce le face mai sigure, dar și mai ineficiente
- Pentru dezvoltarea de aplicații care nu țin de sistemul de operare
- Exemple de interpretoare: GCL – GNU Common Lisp (pt Common Lisp), Python (pt Python), Perl (pt Perl)

Compilare vs interpretare

Compilare

- Viteză mare de execuție – compilarea e un proces separat față de execuția programului
- Codul mașină rezultat în urma compilării poate rula doar pe un anumit tip de procesor
- Limbajele compilate permit distribuirea unui executabil pe alte mașini, fără sa fie nevoie de publicarea codului sursă

Interpretare

- Codul sursă se compilează la fiecare rulare
- Are mai multă portabilitate, deoarece pot fi direct rulate pe orice arhitectură
- Limbajele interpretate presupun că orice mașină care va rula aplicația dezvoltată trebuie să conțină codul sursă ce va fi rulat

Limbaje hibride

- Îmbină elemente de compilare cu elemente de interpretare, pentru a rezulta în aplicații portabile, rapide și sigure
- Transformă codul sursă într-un cd intermediar, care se numește **bytecode**, care va fi apoi interpretat
- Exemple de limbaje hibride: Java, C#, JavaScript

Biblioteci și framework-uri

- Un alt aspect important în dezvoltarea programelor
- **Bibliotecile:** colecții de resurse pe care le integrăm în aplicațiile noastre pentru a ne ușura procesul de dezvoltare
- **Framework:** oferă un schelet pe care aplicația îl urmează pentru a beneficia de proprietățile framework-ului
- Pentru a fi folosite, acestea trebuie să existe pe sistemul pe care lucrăm

De ce programare în C?

- Limbaj foarte cunoscut
- Limbaj foarte puternic
- Limbaj aproape de hardware și de sistemul de operare, ceea ce îl face eficient și rapid
- Aduce un nivel de abstractizare peste limbajul de asamblare, ușurând scrierea programelor, dar presupunând o cunoaștere avansată a funcționării procesorului

De ce nu programare în limbajul C?

- Limbaj relativ greu de învățat
- Limbaj periculos, necesită experiență
- Limbaj mai puțin expresiv
- Durează mult timp să dezvolți o aplicație
- Posibile probleme de portabilitate (Linux, Windows, Mac OS X, Android, etc)

Dezvoltarea aplicațiilor în C

- Cel mai cunoscut compilator este **gcc** (GNU Compiler Collection)
- Pentru folosirea compilatorului este suficient să rulăm comanda gcc cu parametrii aferenți

Compilare de program simplu în C

```
student@host$ ls -F
hello-world.c
student@host$ gcc hello-world.c
student@host$ ls -F
a.out* hello-world.c
student@host$ rm a.out
student@host$ ls -F
hello-world.c
student@host$ gcc -Wall hello-world.c -o hello-world
student@host$ ls -F
hello-world* hello-world.c
student@host$ ./hello-world
Hello, World!
```

Compilare și linking de program simplu în C

```
student@host$ ls -F
hello-world.c
student@host$ gcc -Wall -c hello-world.c
student@host$ ls -F
hello-world.c hello-world.o
student@host$ gcc hello-world.o -o hello-world
student@host$ ls -F
hello-world* hello-world.c hello-world.o
student@host$ ./hello-world
Hello, World!
```

Modularizare și modul

- Do one thing, do one thing well!
- Funcționalitățile diferite intră în fișiere sursă diferite
- Evităm ingrămădirea funcționalităților într-un singur fișier
- Fiecare fișier este numit „modul” (module) sau „unitate de compilare” (compilation unit)

Compilare și linking din surse multiple

```
user@host$ ls -F
debug.h http_reply_once.c sock_util.c sock_util.h util.h
user@host$ gcc -Wall -c sock_util.c
user@host$ gcc -Wall -c http_reply_once.c
user@host$ ls -F
debug.h http_reply_once.o sock_util.h util.h
http_reply_once.c sock_util.c sock_util.o
user@host$ gcc http_reply_once.o sock_util.o -o
http_reply_once
user@host$ ls -F
debug.h http_reply_once.c sock_util.c sock_util.o
http_reply_once* http_reply_once.o sock_util.h util.h
```

Procesul de build

- **Building:** obținerea unui executabil, a unui set de executabile sau a unui pachet software din fișiere cod sursă
- O versiune de pachet construită se mai cheamă și un build
- Pentru programe scrise în C înseamnă compilare, linkind, împachetare

Sisteme de build

- Cele care permit automatizarea procesului de build, deoarece este obositor să rulăm toate comenzile de fiecare dată când aducem o modificare codului sursă
- **make:** folosit foarte mult în lumea Unix/Linux
- **Ant, Maven:** folosit pentru Java
- **Scons:** scris în Python
- **Rake:** folosit pentru Ruby
- altele

Automatizarea procesului de build - make

- Utilitarul de automatizare cel mai folosit pentru aplicațiile C/C++
- Pentru a folosi **make** e suficient să creăm un fișier cu numele **Makefile** în structura programului nostru
- În Makefile se scriu reguli – un fel fișier cu rețete
- La rularea comenzii make, utilitarul găsește acel fișier și execută instrucțiunile descrise
- Formatul unui fișier makefile:

```
Regula: dependinte
<tab> comanda
```

Exemplu fișier Makefile

```
build: utils.o hello.o help.o
	gcc utils.o help.o
hello.o -o hello

all:
	gcc simple_hello.c -o
simple

utils.o: utils.c
	gcc -c utils.o

hello.o: hello.c
	gcc -c hello.c

help.o: help.c
	gcc -c help.c

clean:
	rm -f *.o hello
```

- Regula reprezintă numele unei instrucțiuni
- La simpla rulare a comenzii make, prima regulă din fișier este cea care va fi executată
- `make <nume_regula>`: va executa comanda aferentă regulii cu numele respectiv
- Dependințele sunt fișiere sau reguli necesare pentru a rula o regulă

Folosire simplă make

```
all: hello-world

hello-world: hello-world.c
	gcc -Wall hello-world.c -o hello-world

clean:
	rm -f hello-world hello-world.o
```

```
student@host$ ls -F
hello-world.c
student@host$ make
gcc -Wall hello-world.c -o hello-world
student@host$ ls -F
hello-world* hello-world.c
```

Folosire elegantă a Make pentru surse multiple

```
.PHONY: all clean

All: http_reply_once

http_reply_once: http_reply_once.o sock_util.o
            gcc http_reply_once.o sock_util.o -o
http_reply_once

http_reply_once.o: http_reply_once.c util.h
debug.h sock_util.h
            gcc -Wall -c http_reply_once.c

sock_util.o: sock_util.c util.h debug.h sock
util.h
            gcc -Wall -c sock util.c

clean:
            rm -f http_reply_once
http_reply_once.o sock_util.o
            rm -f *
```

```
student@host$ ls -F
Makefile debug.h http_reply_once.c sock_util.c
sock_util.h util.h
student@host$ make
gcc -Wall -c http_reply_once.c
gcc -Wall -c sock_util.c
gcc http_reply_once.o sock_util.o -o
http_reply_once
student@host$ ls -F
Makefile http_reply_once* http_reply_once.o
sock_util.h util.h
debug.h http_reply_once.c sock_util.c sock_util.o
```

Depanarea programelor

- O mare parte din timpul destinat dezvoltării aplicațiilor îl dedicăm depanării
- Se întâmplă de multe ori ca programul rulat să arunce o eroare sau sa nu obținem rezultatul dorit
- Erorile pot să varieze de la un simbol uitat până la accesări ilegale de memorie sau erori în logică
- Unul din cele mai cunoscute utilitare este **gdb**
- Cea mai simplă metodă de depanare e să afișăm mesaje pe parcursul execuției, dar nu este deloc eficientă

Sisteme de management și versionare

- Avem nevoie de ele pentru a ne ușura lucrul în echipă la proiecte complexe, pentru a partaja cod intr-un mod eficient si sigur
- **Git:**
  - Sistem de management și versionare a codului sursă care permite partajarea unui proiect în cadrul echipei
  - Proiectul este stocat într-un repository
  - Fiecare utilizator lucrează la o versiune proprie a proiectului, pe care apoi o urcă online și este automat integrată în proiect

Licențe pentru programe

- Oferă informații despre dreptul de folosire și distributie a programului
- Când publicăm un program pe care l-am dezvoltat, este important să definim scopul aplicației și să îi atribuim o licență corespunzătoare
- De asemenea, este foarte important să verificăm licența fiecărui pachet extern pe care îl integrăm în aplicația noastră
- Licența unui modul extern poate dicta licența pe care trebuie să o atribuim programului nostru
- Cele mai cunoscute licențe software: GNU GPL, GNU LGPL, MIT

Resurse utile

- http://www.oualline.com/style/index.html
- http://www.gnu.org/software/make/
- http://www.gnu.org/software/libc/manual/
- http://git-scm.com/
- http://gitimmersion.com/
- http://www.moolenaar.net/habits.html

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 exhaustive a analizei sintactice, semantice și 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 și Darwin (Mac OS X)
- un engine peste care rulează componente dedicate: memcheck (implicit), cachegrind, callgrind, helgrind

Cuvinte cheie

- Cod sursă
- Cod mașină
- Editor
- IDE
- Pachet Software
- Fișier executabil
- Fișier Obiect
- Compilare
- Linking
- Limbajul C
- gcc
- modularizare
- Sistem de build
- make
- Makefile
- Git