Unelte de programare

Responsabili:

- Laura Vasilescu (2015) [mailto:laura.vasilescu@cti.pub.ro]

Obiective

În urma parcurgerii acestui laborator studentul va fi capabil să:

- înțeleagă ce înseamnă noțiunile de program, limbaj de programare, compilare/interpretare
- scrie și compileze programe simple în limbajul C
- utilizeze utilitarul Make pentru a compila automat programele scrise
- utilizeze funcțiile din limbaj de citire și scriere a datelor

Introducere

Calculatorul este un instrument care stochează, procesează și redă date sub forma unei secvențe de zero și unu (un sistem binar [https://ro.wikipedia.org/wiki/Sistem_binar], despre care veți învăța mai multe semstrul acesta la Proiectare Logică și Introducere în Informatică), fiind utilizat pentru rezolvarea unor probleme folosind acele date. Un **program** este o înșiruire de instrucțiuni pe care calculatorul le va urma *întocmai* în vederea îndeplinirii unei sarcini. Când calculatorul urmează instrucțiunile dintr-un program, spunem că *execută* programul.

Deoarece noi nu putem folosi direct sistemul binar pentru a spune unui calculator ce să facă, utilizăm un **limbaj de programare**, similar limbii vorbite (are o sintaxă, punctuație bine definite și cuvinte-cheie preluate în general din limba engleză) și deci ușor de asimilat. Acesta poate fi "tradus" prin compilare sau interpretare în *cod-mașină*, limbajul înțeles de calculator și exprimat, în general, în binar.

Compilarea (specifică limbajelor precum Java, Pascal, C - pe care îl veți folosi anul acesta la Programarea Calculatoarelor și Structuri de Date) constă în traducerea propriu-zisă de către un compilator din limbajul de programare ales în cod-mașină, care va fi executat direct de calculator. Interpretarea (Python, Ruby, JavaScript) implică faptul că programul nu va fi rulat direct de calculator, ci de un alt program, *interpretor*, care este deja compilat în cod-mașină.

Pentru acasă

- *Puteți gândi programul ca pe o rețetă care îți spune cum să faci un sendviș cu unt de arahide. În acest model, voi sunteți calculatorul, programul este rețeta, iar limbajul de programare este, desigur, limba în care este scrisă rețeta.*
- *Încercați să formulați o secvență de instrucțiuni pentru a spune cuiva cum să facă un sendviș cu unt de arahide, fără să omiteți vreun pas sau să îi puneți în ordinea greșită.*
- *A fost ușor? V-ați amintit toți pașii? Poate ați uitat să menționați că trebuie folosit un cuțit și cine urmează instrucțiunile are mâinile pline de unt de arahide [https://www.youtube.com/watch?v=cDA3_5982h8]! Sigur vă veți gândi că un om nu este atât de inapt. Dar un calculator chiar este! El va face numai și numai ce îi spui să facă. Acest lucru poate fi frustrant la început, dar într-un fel este o ușurare: dacă faci totul corect, știi exact ce va face calculatorul, pentru că i-ai spus tu. (sursa [https://en.wikiversity.org/wiki/Introduction_to_Programming/About_Programming])*

C este un limbaj de programare structurată menit să simplifice scrierea programelor apropiate de masină. A fost creat de către Dennis Ritchie (care este și unul dintre creatorii sistemului Unix) în perioada 1968-1973 și a fost dezvoltat în strânsă legatură cu sistemul de operare Unix (cu care vă veți familiariza semestrul acesta prin materia Utilizarea Sistemelor de Operare), care a fost rescris în întregime în C.

Utilizarea limbajului s-a extins cu trecerea timpului de la sisteme de operare și aplicaţii de sistem la aplicaţii generale.

Deşi în prezent, pentru dezvoltarea aplicaţiilor complexe au fost create limbaje de nivel mai înalt (Java, C#, Python), C este în continuare foarte folosit la scrierea sistemelor de operare şi a aplicațiilor de performanţă mare sau dimensiune mică (în lumea dispozitivelor embedded). Nucleele sistemelor Windows şi Linux sunt scrise în C.

Să începem prin a analiza un program simplu scris în limbajul C, al cărui scop este să afișeze în linia de comandă mesajul "Hello, World!":

```
#include <stdio.h>

int main()
{
    //Hello from my first program!
    printf("Hello, World!\n");
    return 0;
}
```

Puteți testa codul aici [http://tpcg.io/2yCLtg]!

Prima linie reprezintă o instrucțiune de preprocesare care spune programului să includă header-ul standard input output (stdio.h) din biblioteca C, permițând utilizarea funcțiilor de bază precum *printf, scanf, getc, puts* etc.

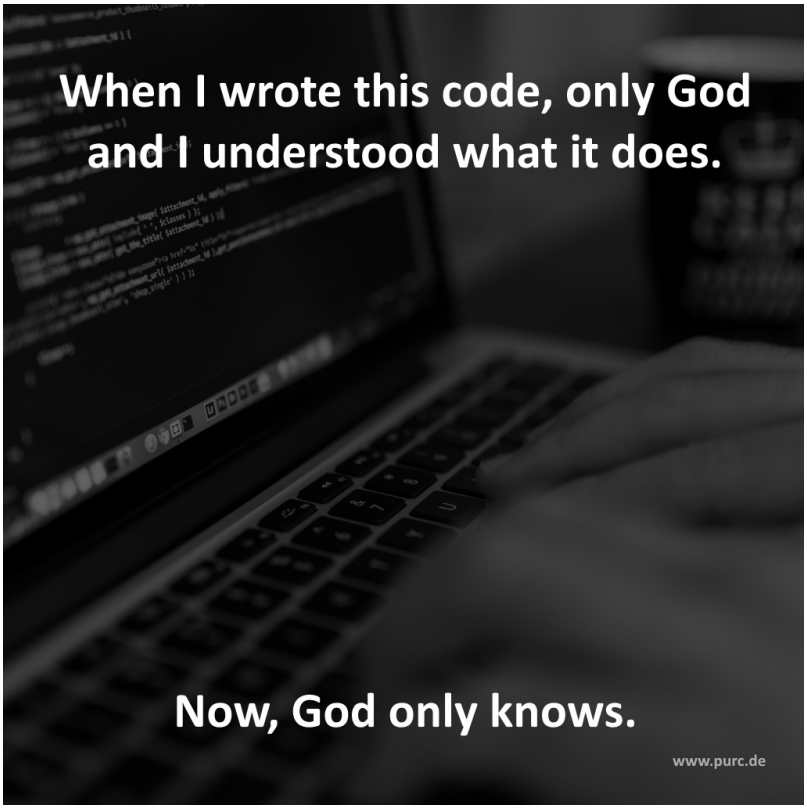
Declarația int main() indică începutul funcției principale, de la care începe execuția în orice program C. Conținutul acestei funcții este încadrat de acolade {}. Conform convenției [https://www.kernel.org/doc/html/v4.10/process/coding-style.html], main fiind o funcție, cele două acolade se află singure pe câte o linie. Pentru alte comenzi (if, while, do, for etc), prima acoladă se va pune pe linia comenzii.

După fiecare comandă este obligatoriu caracterul punct și virgulă *;* pentru a indica sfârșitul comenzii. Din această cauză orice comandă în C se poate scrie pe oricâte rânduri, dar acest lucru nu este recomandat decât dacă linia respectivă ar fi extrem de lungă (peste 80 de caractere).

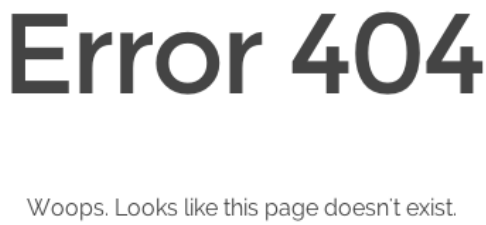
Așa cum este important să scrii frumos de mână pentru ca ceilalți care citesc să înțeleagă, este important să scrii și cod frumos ca să fie ușor de înțeles. Există niște reguli de bază numite **coding style** pe care le găsiți pe scurt aici [https://ocw.cs.pub.ro/courses/programare/coding-style].

Textul care urmează după cele două slash-uri reprezintă un comentariu dedicat exclusiv omului, pe care compilatorul nu îl citește. Comentariile sunt în general folosite pentru a explica/clarifica ce fac anumite părți din cod persoanei care citește codul (inclusiv celui care l-a scris!). Există două moduri de a scrie comentarii:

```
//Everything on this line is a comment.
This text is not a comment. It would cause errors when the compiler tries to translate it.
/* Comments like this can be written
   on multiple lines */
```



Ultima linie din funcția main, *return 0*; indică sfârșitul cu succes a programului în C și este obligatorie. Orice program are un exit code la sfârșitul rulării, care indică dacă programul a fost executat cu succes (acest cod este, în general, zero) sau nu (caz în care va fi returnată o altă valoare care semnifică un anumit cod de eroare). În cazul nostru, 0 este *exit* code.



Unelte folosite

Compilatorul GCC

În cadrul laboratorului și pentru testarea temelor de casă se va folosi compilatorul GCC. GCC este unul dintre primele pachete software dezvoltate în cadrul Proiectului GNU (`GNU's Not Unix`) de către Free Software Foundation. Deși GCC se traducea inițial prin `GNU C Compiler`, acesta a devenit între timp un compilator multifrontend, multi-backend, având suport pentru o serie largă de limbaje, ca C, C++, Objective-C, Ada, Java, etc, astfel că denumirea curentă a devenit `GNU Compiler Collection`. În cadrul cursului de Programare [http://ocw.cs.pub.ro/programare] ne vom referi totuşi numai la partea de C din suita de compilatoare.

Compilatorul GCC rulează pe o gamă largă de echipamente hardware (procesoare din familia: i386, alpha, vax, m68k, sparc, HPPA, arm, MIPS, PowerPC, etc.) și de sisteme de operare (GNU/Linux, DOS, Windows 9x/NT/2000, Solaris, Tru64, VMS, Ultrix, Aix), fiind la ora actuală cel mai folosit compilator.

Compilatorul GCC se apelează din linia de comandă, folosind diferite opțiuni, în funcție de rezultatul care se dorește (specificarea de căi suplimentare de căutare a bibliotecilor/fișierelor antet, link-area unor biblioteci specifice, opțiuni de optimizare, controlul stagiilor de compilare, al avertisementelor, etc.).

Utilizare GCC

Vom folosi pentru exemplificare un program simplu care tipărește la ieșirea standard un șir de caractere.

hello.c

```
#include <stdio.h>

int main() {
  printf("Hello from your first program!\n");
  return 0;
}
```

Pentru compilarea programului se va lansa comanda (în linia de comandă):

```
gcc hello.c
```

presupunând că fișierul sursă se numește hello.c.

Pe un sistem de operare Linux, compilarea default va genera un executabil cu numele a.out. Pentru rularea acestuia, trebuie executată comanda:

```
./a.out
```

Pentru un control mai fin al comportării compilatorului, sunt prezentate în tabelul următor cele mai folosite opţiuni (pentru lista completă studiaţi pagina de manual pentru GCC - man gcc):

Opțiune	Efect
-o nume_fișier	Numele fișierului de ieşire va fi nume_fişier. În cazul în care această opțiune nu este setată, se va folosi numele implicit (pentru fișiere executabile: a.out - pentru Linux).
-I cale_către_fișiere_antet	Caută fișiere antet și în calea specificată.
-L cale_către_biblioteci	Caută fișiere bibliotecă și în calea specificată.
-l nume_bibliotecă	Link-editează biblioteca nume_bibliotecă. **Atenție!!!** nume_bibliotecă nu este întotdeauna același cu numele fișierului antet prin care se include această bibliotecă. Spre exemplu, pentru includerea bibliotecii de funcții matematice, fișierul antet este math.h, iar biblioteca este m.
-W tip_warning	Afișează tipurile de avertismente specificate (Pentru mai multe detalii man gcc sau gcc –help). Cel mai folosit tip este all. Este indicat ca la compilarea cu -Wall să nu apară nici un fel de avertismente.
-c	Compilează și asamblează, dar nu link-editează. Generează fișiere obiect, cu extensia .o.
-S	Se oprește după faza de compilare, fără să asambleze. Rezultă cod assembler în fișiere cu extensia .s.

Despre etapele compilării puteți să citiți mai multe aici [http://elf.cs.pub.ro/so/wiki/laboratoare/laborator-01#fazele-compilarii].

Exemplu

```
gcc -o tema1 tema1.c -lm -Wall
```

Comanda de mai sus are ca efect compilarea și link-editarea fișierului `tema1.c`, cu includerea bibliotecii matematice, afișând toate avertismentele. Fișierul de ieșire se va numi `tema1`.

Atenție!!! Dacă folosiți opțiunea **-o**, nu adăugați imediat după fișierele sursă. Acest lucru ar avea ca efect suprascrierea acestora și pierderea întregului conținut.

Utilitarul Make

Utilitarul `make` determină automat care sunt părțile unui proiect care trebuie recompilate ca urmare a operării unor modificări și declanşează comenzile necesare pentru recompilarea lor. Pentru a putea utiliza `make`, este necesar un fișier de tip `makefile` (numit de obicei `Makefile` sau `makefile`) care descrie relațiile de dependenţă între diferitele fișiere din care se compune programul şi care specifică regulile de actualizare pentru fiecare fişier în parte.

În mod normal, într-un program, fişierul executabil este actualizat (recompilat) pe baza fișierelor-obiect, care la rândul lor sunt obținute prin compilarea fișierelor sursă. Totuși, acest utilitar poate fi folosit pentru orice proiect care conţine dependenţe şi cu orice compilator/utilitar care poate rula în linia de comandă. Odată creat fișierul `makefile`, de fiecare dată când apare vreo modificare în fișierele sursă, este suficient să rulăm utilitarul `make` pentru ca toate recompilările necesare să fie efectuate. Programul `make` utilizează fișierul Makefile ca bază de date şi pe baza timpilor ultimei modificări a fișierelor din Makefile decide care sunt fișierele care trebuie actualizate. Pentru fiecare din aceste fișiere, sunt executate comenzile precizate in Makefile. În continuare prezentăm un exemplu simplu.

```
# Declaratiile de variabile
CC = gcc
CFLAGS = -Wall -lm
SRC = radical.c
EXE = radical

# Regula de compilare
all:
	$(CC) -o $(EXE) $(SRC) $(CFLAGS)

# Regulile de "curatenie" (se folosesc pentru stergerea fisierelor intermediare si/sau rezultate)
.PHONY : clean
clean :
	rm -f $(EXE) *~
```

Atenție!!! Este obligatorie folosirea tab-urilor (nu spații!). Mai multe informații puteți găsiti pe pagina proiectului [http://www.gnu.org/software/make].

Editoare

Pentru editarea surselor se poate folosi orice editor de text.

Exemple:

- interfața în mod text
  - Vim [http://www.vim.org]
  - Emacs [http://www.gnu.org/software/emacs]
  - nano [http://www.nano-editor.org]
- interfață grafică
  - gedit [http://projects.gnome.org/gedit]
  - Kate [http://kate-editor.org]
- IDE
  - CLion [https://www.jetbrains.com/clion] (gratuit pentru studenți)
  - Code::Blocks [http://www.codeblocks.org]

Interacțiunea program-utilizator

Majoritatea algoritmilor presupun introducerea unor date de intrare și calcularea unor rezultate. În cazul programelor de consolă (cele scrise la laborator), datele sunt introduse de la tastatură și afișate pe ecran (alte variante sunt folosirea fișierelor sau preluarea datelor de la un hardware periferic).

Programul dat ca exemplu mai sus folosește funcția de afișare `printf`. Această funcție realizează transferul și conversia de reprezentare a valorii întregi / reale in șir de caractere sub controlul unui format (specificat ca un șir de caractere):

```
printf("format", expr_1, expr_2, ..., expr_n);
```

unde `expr_i` este o expresie care se evaluează la unul din tipurile fundamentale ale limbajului. Este necesar ca pentru fiecare expresie să existe un specificator de format, şi viceversa.

În caz contrar, compilatorul va returna o eroare (în afara cazului în care formatul este obtinut la rulare). Sintaxa unui descriptor de format este:

```
% [ - ] [ Lung ] [ .frac ] [ h|l|L ] descriptor
```

Semnificația câmpurilor din descriptor este descrisă în tabelul următor:

Câmp	Descriere
-	Indică o aliniere la stânga în câmpul de lungime `Lung` (implicit alinierea se face la dreapta).
Lung	Dacă expresia conține mai puțin de `Lung` caractere, ea este precedată de spații sau zerouri, dacă `Lung` începe printr-un zero. Dacă expresia conține mai mult de `Lung` caractere, câmpul de afișare este extins. În absența lui `Lung`, expresia va fi afișată cu atâtea caractere câte conține.
frac	Indică numărul de cifre după virgulă (precizia) cu care se face afișarea.
l	Marchează un `long int`, în timp ce pentru reali `l` determină afișarea unei valori double.
h	Marchează un `short int`.
L	Precede unul din descriptorii `f`, `e`, `E`, `g`, `G` pentru afișarea unei valori de tip `long double`.

Tabelul următor prezintă descriptorii și conversiile care au loc:

Descriptor	Descriere
d	Întreg cu semn în baza 10.
u	Întreg fără semn în baza 10.
o	Întreg fără semn în baza 8.
x sau **X**	Întreg fără semn în baza 16. Se folosesc literele a, b, c, d, e, f mici, respectiv mari.
c	Caracter.
s	Șir de caractere.
f	Real zecimal de forma [-]xxx.yyyyyy (implicit 6 cifre după virgulă)
e sau **E**	Real zecimal în notație exponențială. Se folosește e mic, respectiv E mare.
g	La fel ca și e, E și f dar afișarea se face cu număr minim de cifre zecimale.

Citirea cu format se realizează cu ajutorul funcției `scanf()` astfel:

```
scanf("format", &var_1, &var_2, ..., &var_n);
```

care citește valorile de la intrarea standard în formatul precizat și le depune în variabilele `var_i`, returnând numarul de valori citite.

Atenție!!! Funcția `scanf` primește adresele variabilelor în care are loc citirea. Pentru tipuri fundamentale și/sau structuri, aceasta se obține folosind operatorul de adresă - &.

Sintaxa descriptorului de format în acest caz este:

```
% [*] [ Lung ] [ l ] descriptor
```

Semnificația campurilor din descriptor este descrisă în tabelul următor:

Câmp	Descriere
*****	Indică faptul că valoarea citită nu se atribuie unei variabile. (valoarea citită poate fi folosită pentru specificarea lungimii câmpului)
Lung	Indică lungimea câmpului din care se face citirea. În cazul în care e nespecificat, citirea are loc până la primul caracter care nu face parte din număr, sau până la '\n' (linie nouă/enter).
d	Întreg în baza 10.
o	Întreg în baza 8.
x	Întreg în baza 16.
f	Real.
c	Caracter.
s	Șir de caractere.
L	Indică un întreg long sau un real double.
h	Indică un întreg short.

Pentru scrierea și citirea unui singur caracter, biblioteca `stdio.h` mai definește și funcțiile `getchar()` și `putchar()`:

- `getchar()` are ca efect citirea cu ecou a unui caracter de la terminalul standard. Caracterele introduse de la tastatură sunt puse într-o zonă tampon, până la acționarea tastei ENTER, moment în care în zona tampon se introduce caracterul rând nou. Fiecare apel `getchar()` preia următorul caracter din zona tampon.
- `putchar()` afișează caracterul având codul ASCII egal cu valoarea expresiei parametru.

Nota: `getchar()` și `putchar()` nu sunt de fapt funcții, ci niște macroinstrucțiuni definite în `stdio.h`

Pentru citirea și scrierea unei linii biblioteca `stdio.h` definește funcțiile `gets()` și `puts()`:

- `gets(zona)` - introduce de la terminalul standard un șir de caractere terminat prin acționarea tastei ENTER. Funcția are ca parametru adresa zonei de memorie în care se introduc caracterele citite. Funcția returnează adresa de început a zonei de memorie; la întalnirea sfarșitului de fișier (CTRL+Z) funcția returnează NULL.
- `puts(zona)` - afișează la terminalul standard șirul de caractere din zona dată ca parametru, până la caracterul `null` (`\0`), care va fi înlocuit prin caracterul linie nouă. Funcția returnează codul ultimului caracter din șirul de caractere afișate sau `-1` în caz de eroare.

Exerciții laborator CB/CD

Urmați indicațiile împreună cu asistenul de la laborator.

Probleme

1. Compilați programul din laborator (hello.c [http://ocw.cs.pub.ro/courses/_export/code/programare/laboratoare/lab01?codeblock=0]) utilizând `gcc`.
   - folosiți un fișier de tip `makefile`
   - executabilul se va numi `hello`

2. Într-un director care conține fișierul `hello.c` și **nu** conține niciun fișier de tip `makefile` rulați comanda:

```
make hello
```

   Ce observați?

3. Se citește de la tastatură un număr natural în baza 10. Să se afișeze în bazele 8, 10 și 16.
4. Se citesc de la tastatură două numere reale. Să se afișeze suma, diferenta și media lor cu precizie de 5 zecimale exacte.
5. Să se calculeze (folosind formule matematice; nu instrucțiuni repetitive) și să se afișeze sumele (`n` se va citi de la tastatura):
   - $$S_1 = \sum_{k=0}^{n} k$$
   - $$S_2 = \sum_{k=0}^{n} k^2$$
6. Să se determine minimul și maximul a două numere folosind funcția matematiă `fabs`.
   - afișați rezultatul cu două zecimale
   - **Atenție!** Trebuie să includeți antetul `math.h` și să compilați cu opțiunea `-lm`
7. Se citesc de la tastatură două numere reale. Să se afișeze `EQ` dacă cele două numere sunt egale cu precizie de 4 zecimale; în caz contrar se va afișa mesajul `NOT EQ`.

Extra

- Cheatsheet [https://github.com/cs-pub-ro/ComputerProgramming/blob/master/Laboratories/Lab1/cheatsheet.pdf]