Programarea Calculatoarelor
Cursul 1

Introducere
Funcții de intrare/ieșire

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Cadre didactice

- Curs
  - Robert Varga
    - E-mail: robert.varga@cs.utcluj.ro
    - Web page: http://users.utcluj.ro/~robert
- Laborator
  - Alexandru Boțolan
  - Raluca Bozdog
  - Vivian Chiciudean
  - Dan Domnița
  - Radu Drăgan
  - Robert Varga
- Seminar
  - Robert Varga

Prezență și evaluare

- Sunt permise maximum
  - 4 absențe la curs – pentru 1 punct din oficiu la nota examen
  - 1 absență la seminar
  - 0 absențe la laborator
    - Pot fi recuperate 2 gratuit și încă 2 cu plată
- Nota finală la disciplina Programarea Calculatoarelor
  - 3 teme verificate la seminar
  - 50% nota activități practice laborator
    - obligatoriu >= 5
  - 50% nota examen scris în sesiune
    - obligatoriu >= 5
  - Bonusuri pentru note peste 5 pentru sesiunea de iarnă
    - Teste în timpul semestrului
    - Activitate la seminar
    - Concursuri teme

Obiectivele cursului

- Învățarea programării în limbajul C
  - Elemente de programare generale – reutilizabile la alte limbaje
  - Elemente specifice limbajului C
  - Folosirea uneltelor necesare pentru programare
    - editorul şi depanatorul – laborator
- Abilităţi dobândite
  - Proiectarea algoritmilor
  - Implementarea unui algoritm ca un program C
  - Analiza şi înţelegerea algoritmilor şi a codului scris de altcineva
  - Găsirea erorilor de proiectare/programare (bug-uri)

Surse bibliografice

- Carte în limba engleză - pentru teorie
  - Kim N. King - C programming: a Modern Approach
- Probleme
  - laborator PC și teme PC
  - infoarena.ro - probleme rezolvate de 500 utilizatori
  - codeforces.com - div. 3 sau div. 2 A, B
  - projecteuler.net - grad de dificultate < 20%
  - https://leetcode.com/problemset/all/
- Documentație bibliotecă
  - https://en.cppreference.com/w/c standardul C99 sau mai nou

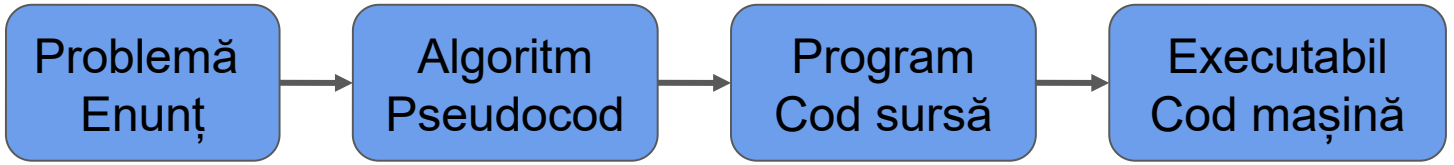
Conținutul cursului

1. Introducere, funcții de intrare/ieșire
2. Tipuri de date, expresii, instrucțiuni de decizie
3. Operatori pe biți, instrucțiuni repetitive
4. Funcții, tablouri

5. Directive de preprocesare
6. Pointeri 1 - Legătură pointeri tablouri
7. Pointeri 2 - Alocare dinamică
8. Șiruri de caractere
9. Recursivitate
10. Structuri, uniuni, enumerări, definiții de tipuri
11. Fișiere
12. Argumente din linia de comandă

Programare

- Programarea este activitatea prin care se obține un program, codificat într-un limbaj de programare, în vederea rezolvării unei probleme
- Este necesară codificarea soluției într-un limbaj intermediar, ușor de înțeles de om și ușor convertit în instrucțiuni executabile de calculator



Etapele rezolvării unei probleme

- Definirea și analiza problemei
  - Enunțul clar și precis al problemei de rezolvat
  - Specificarea datelor de intrare și limitele lor
- Proiectarea algoritmului
  - Stabilirea metodei de rezolvare pas cu pas
  - Pseudecod, schemă logică sau desen
- Implementarea programului
  - Codificare într-un limbaj de programare
  - Depanarea programului
- Testarea și validarea programului
  - Rulează și se comporta conform așteptărilor
  - Oferă rezultate corecte pentru toate cazurile posibile
- Întreținerea programului și întocmirea documentației

Strategii ajutătoare la rezolvarea problemelor

- Desenarea și vizualizarea
- Calcularea manuală pe exemple mici
  - Ghicim și verificăm regula
- Stabilirea unei legături la probleme și concepte cunoscute
- Transformarea problemei în una mai generală
  - Ignorarea (temporară) a unor constrăngeri
- Transformarea problemei în una mai specifică
  - Adăugarea unor constrângeri care duce la o problemă rezolvabilă
- Studierea cazurilor speciale
- Formulare recursivă prin specificarea relației de recurență
- Folosirea simetriei

Algoritm

- Algoritmul este o succesiune de operații, ordonate în timp, care pornind de la datele de intrare produc într-un timp finit datele de ieșire
- Descris de obicei în pseudocod, independent de un limbaj de programare
- Pseudocodul este un limbaj informal, intermediar, care se află între limbajul natural și un limbaj de programare (formal)
- Trebuie să fie:
  - corect, finit, eficient, economic, clar definit

Algoritm - exemplu

- Fie următorul pseudocod pentru descompunerea unui număr pozitiv *x* în factori primi:

```
citeste x
fie p = 1
cattimp p < x
   cattimp x divizibil cu p
      afiseaza p
      x = x / p
   p = p + 1
```

greșelile discutate la curs

Limbajul de programare C

- Dezvoltat de Dennis Ritchie la Bell Labs între anii 1969-1973
- Originea limbajului de programare C este strâns legată de sistemul de operare Unix
- Utilizat cel mai adesea pentru scrierea programelor eficiente și portabile
- Este un limbaj de programare:
  - imperativ - se bazează pe instrucțiuni/comenzi
  - compilat - se generează cod mașină folosind un compilator
  - de nivel înalt - nu este nevoie să cunoaștem detaliile arhitecturii
  - scurt - are un număr redus de cuvinte cheie

Primul program în C

```c
// primul program in C
#include <stdio.h>
int main(void)
{
  printf("Hello, World!");
  return 0;
}
```

Rând comentat

- conținutul nu influențează programul
- este destinat programatorului
- valabil doar pe un rând

Primul program în C

```c
// primul program in C
#include <stdio.h>
int main(void)
{
  printf("Hello, World!");
  return 0;
}
```

Directivă de preprocesare

- copiază conținutul fișierului
- permite accesul la funcțiile din bibliotecă
- necesară pentru funcțiile de citire și afișare

Primul program în C

```c
// primul program in C
#include <stdio.h>
int main(void)
{
  printf("Hello, World!");
  return 0;
}
```

Antetul funcției principale `main`

- este obligatoriu
- definește
  - tipul returnat,
  - numele funcției și
  - parametrii de intrare
- funcția este rulată la lansarea programului

Primul program în C

```c
// primul program in C
#include <stdio.h>
int main(void)
{
  printf("Hello, World!");
  return 0;
}
```

Acoladele definesc blocul funcției

- instrucțiunile aflate între ele formează corpul funcției main
- este obligatoriu

Primul program în C

```c
// primul program in C
#include <stdio.h>
int main(void)
{
  printf("Hello, World!");
  return 0;
}
```

Apel la funcția de afișare

- tipărește pe ecran mesajul între ghilimele "..."
- instrucțiunea se termină obligatoriu cu ;

Primul program în C

```c
// primul program in C
#include <stdio.h>
int main(void)
{
  printf("Hello, World!");
  return 0;
}
```

Instrucțiune de return

- programul returnează `0` dacă s-a terminat fără erori

Compilarea programelor C

- Editarea codului sursă
  - Salvarea fișierului scris cu extensia .c
- Preprocesarea
  - Efectuarea directivelor de preprocesare
  - Includerea fișierelor header (cu extensia .h) corespunzătoare bibliotecilor folosite
  - Ca un editor – modifică și adaugă la codul sursă
- Compilarea
  - Verificarea sintaxei
  - Transformare în cod obiect (fișier cu extensia .o sau .obj)
- Editarea legăturilor (link-editarea)
  - Combinarea codului obiect cu alte coduri obiect (al bibliotecilor asociate fișierelor header)
  - Transformarea adreselor simbolice în adrese reale
  - Se obține fișierul executabil cu extensia .exe

Variabile

- O **variabilă** reprezintă o valoare care se poate modifica
  - se folosește la stocarea și manipularea datelor
- În limbajul C fiecare variabilă trebuie declarată
- Declarația stabilește numele și tipul variabilei
- Exemple:

```
char c;
int i, j, k;
float temp_celsius, temp_fahrenheit;
```

tipul variabilei

numele variabilei - începe cu literă, nu conține spațiu

Inițializarea și atribuirea

- Variabilele se pot **inițializa** în momentul declarării lor
- Este recomandat, altfel ele vor avea valori necunoscute (aleatoare, neinițializate)
- Exemple:

```
char c = 'a';
int i = 1, j = -2, k = 3;
float x = 1.5, y = -.4;
```

- După inițializare putem schimba valoarea stocată prin **atribuire**
- Exemple:

```
char c;
c = 'a';
int i = 1;
i = 2;
```

Tipuri de date - introducere

- Tipul unei date determină operațiile posibile și domeniul ei
- La declarația variabilelor trebuie să stabilim tipul corect
- Tipuri importante:
  - `char` - caracter - o singură literă, cifră, simbol
  - `int` - întreg - un număr întreg
  - `float` - real - un număr zecimal (real)
  - `void` - vid - lipsa unui tip
  - `char*` - șir de caractere

Funcții de intrare/ieșire

- Limbajul C nu are instrucțiuni pentru citire/scriere
- Pentru a citi sau a afișa date trebuie să folosim funcțiile din biblioteca standard de intrare/ieșire `stdio.h`
- Există două funcții principale care permit citirea/afișarea formatată
  - scanf - citire formatată - vine de la eng. scan formatted
    - citește în mod formatat de la tastatură = fișierul de intrare standard - `stdin`
  - printf - afișare formatată - vine de la eng. print formatted
    - afișează pe ecran = fișierul standard de ieșire - `stdout`
- Mai există un fișier atașat terminalului standard
  - Ieșirea standard pentru erori - `stderr`

Funcția printf

```
int printf(const char* format, ...);
```

- Funcția afișează pe ecran conform specificatorilor de format din șirul de caractere `format` expresiile care urmează (0 sau mai multe)
- Parametrii
  - `const char* format` - reprezintă șirul afișat pe ecran care poate conține 0 sau mai mulți specificatori de format
  - … - sunt 0 sau mai multe expresii
- Valoare returnată
  - returnează numărul de caractere scrise cu succes

Funcția printf - exemple simple

```
//Exemplu 1
printf("hello");
```

- Afișează șirul trimis
- Nu avem nici un specificator de format

```
//Exemplu 2
int x = 7;
printf("%d", x);
```

- Afișează variabila de tip `int` interpretată ca un întreg în baza 10 (d = decimal)
- Avem un specificator de format și o expresie, care acum este o variabilă

```
//Exemplu 3
printf("%c", 65);
```

- Afișează numărul 65 interpretat ca un caracter (c = character)
- Avem un specificator de format și o expresie, care acum este un număr (literal)

Specificatorul de format

- Specificatorul de format are forma generală:

```
% indicator dimensiune precizie modificator conversie
```

- Obligatoriu este semnul % și conversie (tipul)
- Numărul de specificatori trebuie să coincidă cu numărul expresiilor

Specificatorul de format

Specificator format	Tip de date	Explicații
%c	char	scrie un singur caracter conform codului ASCII
%d (%i)	int	scrie un singur număr întreg mic în baza 10
%lld	long long int	scrie un singur număr întreg mare în baza 10
%f	float, double	scrie un singur număr real (cu zecimale) în baza 10
%s	char*	scrie un șir de caractere

ce înseamnă număr mic și mare vom vedea exact în cursul următor

Specificatorul de format

Specificator format	Tip de date	Explicații
%o	unsigned int	scrie un singur număr întreg fără semn în baza 8
%x %X	unsigned int	scrie un singur număr întreg fără semn în baza 16
%u	unsigned int	scrie un singur număr întreg fără semn în baza 10
%e %E	float, double	scrie un singur număr real (cu zecimale) în baza 10 în notație științifică mantisă/exponent
%g	float, double	scrie un singur număr real (cu zecimale) în baza 10 în notație scurtă
%a %A	float, double	scrie un singur număr real (cu zecimale) în baza 10 în notație hexazecimală fracționară
%p	void*	scrie adresa la care arată un singur pointer ca un număr hexazecimal
%n	int*	stochează în variabilă numărul de caractere afișate până momentul prezent
%%	-	scrie caracterul special %

Specificatorul de format - indicator

```
% indicator dimensiune precizie modificator conversie
```

- indicator (opțional)
  - modifică alinierea datelor

-	Scrie valoarea în cadrul câmpului aliniată la stânga (în locul alinierii implicite la dreapta)
+	Scrie semnul plus dacă valoarea este pozitivă
#	Scrie 0 ca și prim caracter pentru o valoare reprezentată în octal și respectiv 0x pentru o valoare reprezentată în hexazecimal. Pentru valori reale indică faptul că punctul zecimal va fi prezent chiar dacă nu urmează nicio cifra zecimală după acesta
0	Completează câmpul cu caractere zero în locul spațiilor. Indicatorul este ignorat dacă este specificat și indicatorul -

Specificatorul de format - dimensiune, precizie

```
% indicator dimensiune precizie modificator conversie
```

- dimensiune (opțional)
  - este un număr întreg sau *
    - dacă este * folosește valoarea precedentă stocată
  - reprezintă dimensiunea minimă a câmpului în care se face scrierea
  - în caz de depășire se tipăresc toate caracterele necesare
  - implicit se aliniază la dreapta
- precizie (opțional)
  - este un număr de forma .x sau .*
    - dacă este .* folosește valoarea precedentă stocată
  - specifică numărul de cifre după virgula zecimală care vor fi afișate

Specificatorul de format - modificator

```
% indicator dimensiune precizie modificator conversie
```

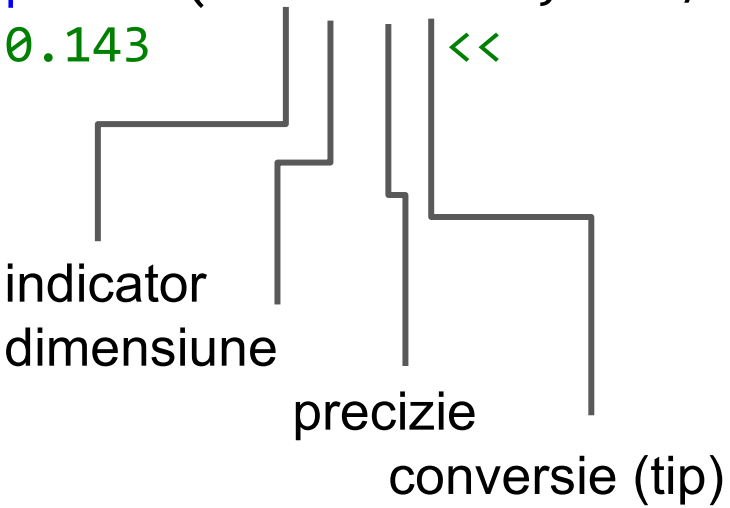
- modifică mărimea datelor
  - sunt literele l, h, L

Specificator format	Tip de date	Descriere
%hd	short int	numere mici
%hu	short unsigned int	numere mici fără semn
%lld	long long int	numere mari
%llu	long long unsigned int	numere mari fără semn
%Ld	long double	numere reale mari

ce înseamnă mic și mare vom vedea exact în cursul următor

Funcția printf - exemple complexe

```
printf("%-13.3f<<", 1.0/7);
0.143        <<
```



- Afișează cifrele numărului real 1/7 și apoi caracterele <<
- Se aliniază la stânga
- Folosește un câmp de lățime 13 pentru 1/7
- Afișează doar 3 cifre după virgula zecimală

```
long long int x = 100000;
x = x * x;
printf("%+*lldxx", 20, x);
       +10000000000xx
```

- Construiește numărul (10^5)^2 în variabila x
- Afișează cifrele numărului întreg stocat în x și apoi caracterele xx
- Alocă câmp de dimensiune 20 citit prin *
- afișează + în fața numărului

Funcția printf - caractere speciale

- Caracterul \ (backslash) are un rol special în șiruri
  - el introduce o secvență specială - eng. escape sequence
  - tabelul următor prezintă doar câteva cele mai des folosite

Secvența specială	**Caracter printat**
\n	linie nouă
\b	backspace
\t	tab
\a	alert
\"	ghilimele
\\	backslash

Funcția scanf

```
int scanf(const char* format, ...);
```

- Funcția citește de la tastatură conform specificatorilor de format din șirul de caractere `format` și stochează valorile la adresele care urmează
- Parametrii
  - `const char* format` - reprezintă 0 sau mai mulți specificatori de format care indică tipul datelor citite
  - … - sunt 0 sau mai multe adrese de variabile unde se stochează valorile citite
- Valoare returnată
  - returnează numărul de valori citite corect sau EOF (-1) în cazul în care apare o eroare de citire (din orice cauză)

Funcția scanf - exemple simple

```
//Exemplu 1
int x;
scanf("%d", &x);
```

- Citește un număr întreg în baza 10 și stochează în variabila x
- Variabila x a fost neinițializată înainte de citire

```
//Exemplu 2
char c = 'a';
scanf("%c", &c);
```

- Citește un singur caracter și stochează în variabila c
- Valoarea de dinaintea apelului din c este suprascrisă

```
//Exemplu 3
char s[100];
scanf("%s", s);
```

- Declară șirul de caractere s și citește un șir în s
- Șirul nu trebuie fie de lungime mai mare ca 99

Specificatori de format pentru scanf

- Folosește aceiași specificatori de format
  - excepție, tipul double se citește cu %lf
- Dimensiunea specificatorului determină câte caractere se citesc
  - de exemplu, %1d citește doar o singură cifră
- Citirea se oprește la primul caracter care nu se potrivește cu specificatorul de format
  - de exemplu, se așteaptă o cifră și utilizatorul introduce o literă
- Dacă apar alte caractere în șir acestea trebuie să se potrivească cu cele introduse de utilizator în momentul scanării
  - Caracterul spațiu se potrivește cu spațiu simplu, rând nou, tab sau alte caractere albe (eng. whitespace)

Specificatorul de format - tipuri fundamentale scanf

Specificator format	Tip de date	Explicații
%c	char	citește un singur caracter conform codului ASCII
%d (%i)	int	citește un singur număr întreg în baza 10
%lld	long long int	citește un singur număr întreg mare* în baza 10
%f	float	citește un singur număr real cu precizie simplă în baza 10
%lf	double	citește un singur număr real cu precizie dublă în baza 10
%s	char*	citește un șir de caractere

Funcția scanf - exemple complexe

```
//Exemplu 1
int a, b;
scanf("%d/%d", &a, &b);
```

- Citește o fracție de forma a/b
- Utilizatorul trebuie să introducă un număr urmat de caracterul / și un alt număr, fără spațiu între ele
- De exemplu, 1/2

```
//Exemplu 2
scanf("%d / %d", &a, &b);
```

- În acest caz pot exista 0 sau mai multe spații, sau chiar rând nou, între numere și semnul /
- Exemple, 1/2, 1 / 2, 1/ 2

```
//Exemplu 3
char s[100];
scanf("%[^\n]", s);
```

- Folosește un specificator de format bazat pe expresie regex care citește un șir de caractere
- Se oprește de citit șirul doar la întâlnirea caracterului linie nouă \n
- Cu %s se oprește și la spațiu simplu

Diferențe printf/scanf

- Pentru scanf trebuie sa includem & în fața fiecărei variabile
  - acesta obține adresa de memorie a variabilei
    - mai multe detalii la pointeri
  - omiterea operatorului & duce la erori la rulare
  - excepție fac șirurile de caractere care nu trebuie precedate de &
    - mai multe detalii la pointeri
- Șirul format de la scanf *de obicei* conține doar specificatori de format, fără spații sau alte caractere
- Specificatorul de format pentru scanf folosit pentru tipul double este `%lf` iar pentru printf este `%f`

Funcția puts

```
int puts(const char* s);
```

- Scrie șirul de caractere `s` urmat de caracterul linie nouă (\n)
- Parametrii
  - `const char* s` - șirul de caractere care se tipărește
- Valoare returnată
  - Funcția returnează o valoare nenegativă în caz de succes sau EOF (-1) în caz de eșec
- Este utilă la afișarea mesajelor simple

Funcția gets

```
char* gets(char* s);
```

- Citește un șir de caractere de la intrarea standard (stdin) până la întâlnirea caracterului linie nouă (`'\n'`) și îl memorează în șirul de caractere primit ca și argument
  - Caracterul newline este automat exclus din șirul memorat
- Parametrii
  - `char* s` - reprezintă adresa unde se va face citirea
- Valoare returnată
  - returnează argumentul citit în caz de succes, iar în caz de eroare funcția returnează un pointer NULL
- Este utilă pentru citirea șirurilor care conțin spații
- Funcția trebuie apelată cu grijă întrucât nu are nicio protecție împotriva citirii unui șir de caractere de dimensiune mai mare decât cea alocată
  - Se recomandă în locul acesteia utilizarea funcției `fgets` care să citească din fișierul de intrare standard `stdin`

Funcțiile de conversie sscanf/sprintf

- Pentru convertirea șirurilor de caractere în alte tipuri se recomandă folosirea funcției sscanf
- Pentru conversia inversă se recomandă funcția sprintf
- Mai multe detalii în cursul cu stringuri

Greșeli des întâlnite

```
//Exemplu 1
char c;
scanf("%c", c);
```

- Am uitat operatorul & la scanf

```
//Exemplu 2
int a = 42;
printf("%f", a);
```

- Specificator de format nepotrivit

```
//Exemplu 3
printf("%d");
```

- Numărul de expresii nu coincide cu numărul specificatorilor

Exemplu program complet scanf/printf

```c
#include <stdio.h>

int main(){
    char nume[100];
    puts("Cum te cheama?");
    gets(nume);
    printf("Hello, %s!\n", nume);

    int varsta;
    puts("Varsta ta?");
    scanf("%d", &varsta);
    printf("Varsta in baza 8 este: %o\n", varsta);

    return 0;
}
```

- Se citește un șir de caractere care poate conține și spații
- Se afișează un mesaj care include șirul citit

- Se citește vârsta ca un număr în baza 10
- Se folosește specificatorul de format pentru a afișa numărul în baza 8