Limbaje de Programare
Curs 1 – Introducere

Dr. Casandra Holotescu

Universitatea Politehnica Timișoara

Ce discutăm azi...

1. Despre Curs
2. Programarea Calculatoarelor
3. Limbajul C
4. Functii
5. Variabile
6. Operatii de I/O

Limbaje de Programare

Curs: 10 săptămâni $\times$ 3 h
Laborator: 14 săptămâni $\times$ 2 h

Evaluare: distribuită

Ce înseamnă asta?

- fără examen final în sesiune
- notă examen = media a 3 teste date pe parcurs

Consultații: la cerere

Principii de bază

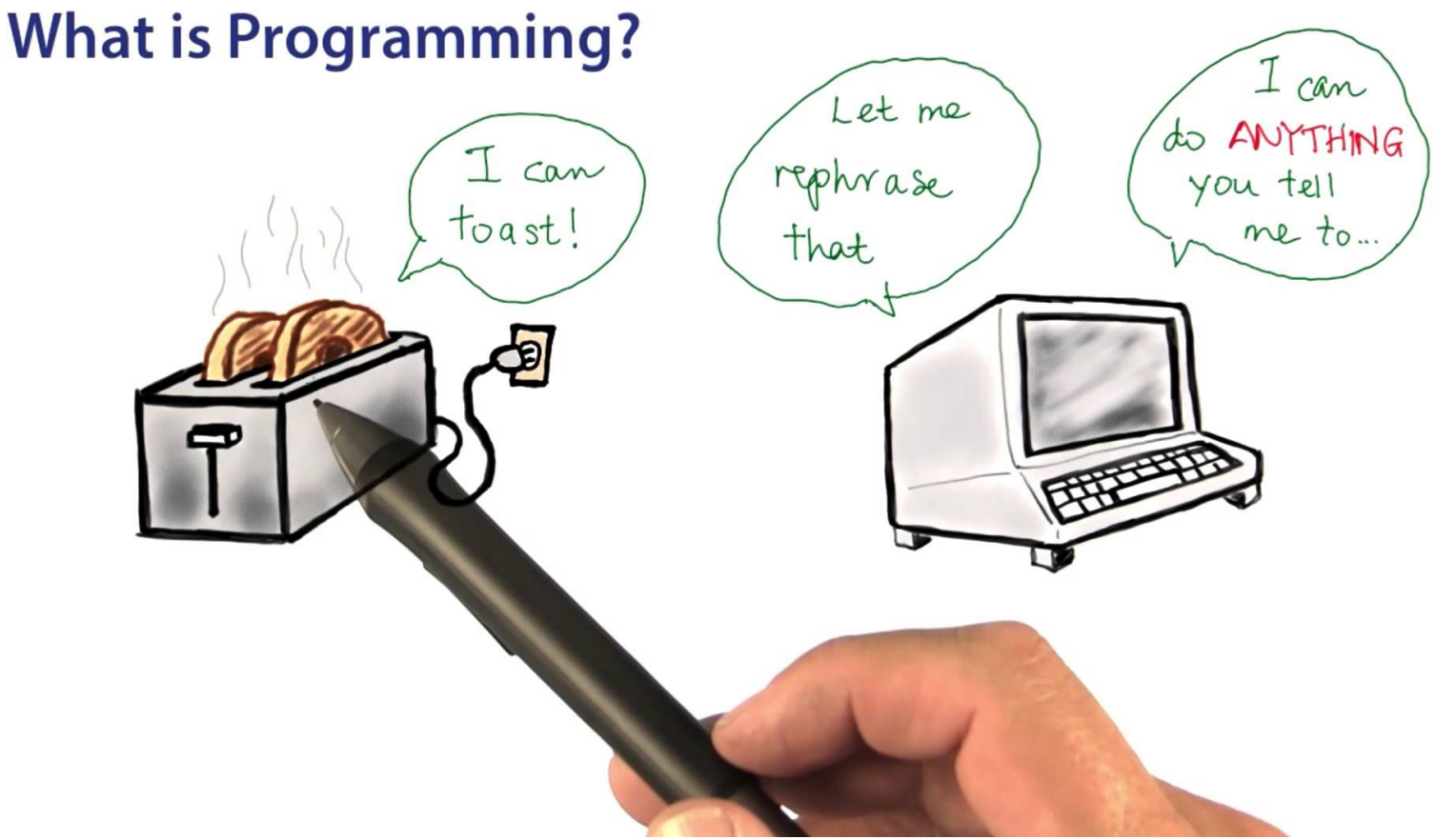
Nu e OK [și poate avea consecințe grave] să:

- copiați
- prezentați munca altuia drept a voastră

E OK să:

- oferiți un feedback onest
- întrebați când ceva nu e clar
- învățați împreună

Programarea Calculatoarelor...



Program = o secvență de instrucțiuni, care se execută într-o anumită ordine, pentru a îndeplini un scop bine precizat

Ce face un program?

- citește date de intrare
- le prelucrează printr-o succesiune de calcule matematice
- scrie rezultatele la ieșire (ecran, imprimantă, etc.)

Primul programator: Augusta Ada King, contesă de Lovelace, (fiica Lordului Byron), 1815–1852
— a scris programe pentru *Motorul Analitic* al matematicianului Charles Babbage (un fel de computer *mecanic*, in sistem decimal)

Terminologie

- *cuvinte cheie* = cuvinte cu un înteles predefinit, care nu poate fi schimbat pentru un limbaj de programare
- *identificator* = nume ales de programator pentru o variabilă, funcție, etc.
- *constante*: numerice (1, 2.5, 3.14, etc.), caractere, șiruri
- *funcții*: unități de bază ale programului, definesc calculele necesare pentru prelucrarea datelor de intrare

Limbajul C

Dezvoltat în 1972, la AT&T Bell Laboratories de Dennis Ritchie.
Cartea de referință:
– Brian Kernighan, Dennis Ritchie: *The C Programming Language* (1978)

standard: ANSI C, 1988 (American National Standards Institute)
versiunea curentă: C99 (standard ISO 9899)

Primul program C

```
int main(void){
    return 0;
}
```

Cel mai mic program – nu face nimic.

Functia **main** este **obligatorie** pentru orice program C, iar execuția programului constă în apelarea ei.

În C, după fiecare instrucțiune e **obligatoriu ;** (punct și virgulă)

void – vid, funcția nu are parametri

Cf. standardului, *main* returnează un cod către sistemul de operare (eroare dacă $\neq 0$, succes daca $= 0$)

Ce face un program?

Un program prelucrează datele de intrare prin calcule matematice...

calcule = funcții

Putem:

- folosi funcții existente
- defini funcții noi
- compune funcții existente sau definite de noi
- alege ordinea în care dorim să le aplicăm

Funcții C vs funcții matematice

$f : \mathbb{Z} \to \mathbb{Z}$
$f(x) = 2x + 1$

va fi in C:

```
int f(int x){
    return 2*x+1;
}
```

Definiția unei funcții este formată din:

- antetul functiei:

```
int f(int x)
```

- corpul funcției:

```
{ return 2*x+1; }
```

Definiția unei funcții

```
int f(int x){
    return 2*x+1;
}
```

Antetul funcției:
tip nume-functie (tip nume-param, tip nume-param, ..)

- tipul returnat – aici *int* (întreg)
- numele funcției – aici *f*
- parametrii funcției, tip si nume – aici *int x*

Corpul funcției:

- o listă de instrucțiuni între { și }

Funcție cu parametri reali

$medie : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$$medie(x, y) = \frac{x + y}{2}$$

va fi in C:

```
double medie (double x, double y){
    return (x+y)/2;
}
```

Obs.: *double* este unul dintre tipurile folosite în C pentru reprezentarea numerelor reale

Tipuri in C

Tip = o mulțime de valori, alături de un set de operații permise de aceste valori

Numere întregi:

- $\mathbb{N}$ și $\mathbb{Z}$ sunt mulțimi infinite
- spațiul de stocare este finit $\Rightarrow$ numai submulțimi finite ale $\mathbb{N}$ și $\mathbb{Z}$ pot fi reprezentate în programare
- tipuri întregi: short, int, long
- pentru numere naturale (fără semn): *unsigned* short, *unsigned* int, *unsigned* long

Tipuri in C

Numere reale:

- $\mathbb{R}$ este o mulțime densă (între oricare 2 numere reale există alt număr real)
- în programare, nu putem reprezenta decât numere reale cu o precizie finită
- tipuri reale: float (simplă precizie), double (dublă precizie)

Unele operații sunt diferite pentru întregi și reali:

- împărțirea între numere întregi e împărțire cu rest: $3/2 = 1$ (întregi), dar $3.0/2 = 1.5$ (reali)

Alte tipuri: caracter, șiruri de caracter, adresă, structură, etc. (vom vedea mai târziu)

Apel de funcție

O funcție definită poate fi apelată oriunde în interiorul altei funcții.

Valoarea returnată de funcție poate fi folosită în alte expresii.

Apelul are loc ca în matematică:
nume-functie(parametru1, parametru2, ...)

Exemplu de apel:

```
f(2)
```

și un apel utilizat într-o expresie (presupunem ca *f* e deja definită):

```
int h(int x, int y){
    return f(x) + f(1+y) + f(2);
}
```

Câțiva operatori aritmetici

Operator	Semnificație	Exemplu expresie
+	adunare	1 + 2
-	scadere	5 - 3
*	înmulțire	3 * 4
/	împărțire	8 / 4
%	rest împărțire	5 % 3

Obs. : % se poate folosi doar pt. numere întregi

Comentarii

```
/* Acesta este un comentariu pe
   mai multe randuri, oricate */

 int main(void){
  return 0;
  //acesta e un comentariu pana la sf liniei
 }
```

Comentariile sunt secvente in limbaj natural, care explică întelesul unor instrucțiuni, fara niciun efect asupra executiei programului.

Programele C pot conține comentarii ori între /* și */, ori pornind de la // și pâna la sfârșitul liniei.

Declarare de variabile

Uneori avem nevoie să reținem în memorie valori de un anumit tip.

Ce presupune asta?

- rezervarea unei zone de memorie de dimensiune potrivită
- asocierea acesteia cu un identificator (nume) pentru a putea accesa valoarea memorată

O variabilă este declarată astfel:
tip nume-variabila;

```
int x;
double c = 5.3;
// valoarea initiala a lui c va fi 5.3
```

Obs.: niciun identificator (nume de variabila, de funcție, etc.) nu poate fi folosit înainte de a fi declarat!!

Domeniu de vizibilitate

Domeniul de vizibilitate al unui identificator
= partea din program unde poate fi utilizat

Parametrii și variabilele declarate în interiorul unei funcții:

- au ca domeniu de vizibilitate corpul funcției
- au durata de memorare automată (create la apelul funcției, distruse la încheierea apelului)

Variabilele declarate în exteriorul oricărei funcții:

- vizibilitate globală & durată de memorare statică
- nu se recomandă în practică

Atribuirea unei valori

Scopul atribuirii este de a modifica valoarea memorată într-o variabilă.

Sintaxa:
nume-variabila = expresie;

```
int a = 2, b=3, c;
c = a + b;
```

Observație: = reprezintă în acest caz operatorul de atribuire, nu cel de egalitate matematică!

Tipărirea

```
#include <stdio.h>

int main(void){
        printf("hello, world!\n");

        return 0;
}
```

- *printf* – funcție standard C pt tiparire
- "hello, world! \n" – constantă șir de caractere, între ghilimele, unde '\n' – caracter linie nouă

prima linie = directivă de preprocesare, include fișierul stdio.h cu declarațiile funcțiilor standard de intrare / ieșire

Tipărirea de numere

Tipărirea unui număr întreg:

```
printf("%d", 9);   //obs. %d !!
```

Tipărirea unui număr real:

```
printf("%f", 4.55); //obs %f !!
```

Pentru a tipări valoarea unei expresii, printf ia două argumente:

- un șir de caractere (specificator de format): %d pentru întregi, %f pentru reali
- expresia, de tip compatibil cu cel indicat prin specificatorul de format

Câte ceva despre citirea de la tastatură

```
#include <stdio.h>

int main(void){
  int k;
  scanf("%d", &k);

  return 0;
}
```

Atentie: in cazul lui *scanf* fiecare variabila trebuie prefixată de simbolul &, ceea ce inseamna adresa acelei variabile.

Se folosesc aceiași specificatori de format ca la *printf* (discutam mai mult intr-un curs viitor).