# Capitolul 3 Date şi tipuri de date

## 3.1 Noţiuni generale

Aşa cum s-a arătat în capitolul 1, calculatorul stochează şi prelucrează date, acestea având diferite semnificaţii şi fiind indispensabile rezolvării oricărui tip de problemă. Există două categorii de date:

- unele care îşi pot modifica valoarea pe parcursul execuţiei unui program, numite variabile;
- altele care îşi păstrează valoarea stabilită iniţial, fără să existe vreo modalitate de a o modifica, numite constante.

Indiferent dacă este vorba de variabile sau de constante, înainte de a putea fi folosite într-un program, acestea trebuie declarate. Variabilelor li se specifică numele, tipul şi opţional valoarea iniţială, iar constantelor numele, tipul şi obligatoriu valoarea. În urma declarării, variabilelor şi constantelor li se alocă spaţiu în memorie.

Numele unei variabile sau al unei constante este alcătuit prin alăturarea unor caractere (litere, cifre, semne speciale), respectând anumite reguli. În acest sens e bine ca un programator să cunoască întregul set de caractere pe care le are la dispoziţie. Acestea pot fi consultate în anexa 1. Atunci când se stabileşte numele unei variabile sau al unei constante trebuie ţinut cont de câteva restricţii [CH96]:

- primul caracter al numelui trebuie să fie neapărat o literă sau o liniuţă de subliniere (de exemplu `nume` sau `_nume`);
- după primul caracter se pot utiliza orice combinaţii de cifre şi litere cu excepţia caracterelor care nu sunt alfanumerice, cum ar fi #, $ sau spaţiu (de exemplu se pot folosi denumiri ca `variabila1`, `cuv2nou`, dar nu sunt acceptate nume de genul `var$`);
- limbajul C este *case sensitive*, adică face diferenţa între litere mari şi mici; atunci când se declară şi se utilizează variabile sau constante, trebuie să se ţină cont şi de acest lucru (de exemplu variabilele `nume1` şi `Nume1` sunt diferite);
- există o serie de cuvinte rezervate care nu se pot utiliza ca nume de variabile sau de constante; lista completă a acestor cuvinte, numite şi cuvinte cheie, se găseşte în anexa 2.

### 3.1.1 Tipuri de date

Tipul unei variabile sau constante este dependent de ceea ce trebuie reprezentat, de aici rezultând felul în care informaţia este stocată în memorie. Orice entitate care se doreşte a fi procesată de calculator trebuie transpusă din forma în care ea este achiziţionată (de la utilizator sau din alte surse) într-o formă fizică definită prin tensiuni electrice. Astfel se ajunge la unitatea de bază pentru reprezentarea informaţiei în memorie, care este bitul. Acesta poate lua valorile logice 0 (dacă tensiunea se află în intervalul 0 – 0.4 volţi) sau 1 (pentru o tensiune în intervalul 2.4 – 5 volţi). Aşadar, un bit [NR01] este informaţia elementară, ireductibilă, caracterizată prin aceea că surprinde una din două posibilităţi care sunt contradictorii şi complementare.

De obicei biţii se grupează în coduri de lungimi care sunt puteri ale lui 2. Codurile uzuale sunt pe 8, 16, 32, 64, 128, 256 biţi. Opt biţi formează un byte sau octet. În tabelul 3.1.1 sunt prezentaţi multiplii bitului şi ai octetului [NR01].

Tabelul 3.1.1 Multiplii bitului şi ai octetului

| **Multiplii bitului** | **Multiplii octetului** |
|---|---|
| 1 Kilobit (Kbit) = $2^{10}$ biţi | 1 Kilooctet (Kilobyte – KB) = $2^{10}$ octeţi |
| 1 Megabit (Mbit) = $2^{20}$ biţi | 1 Megaoctet (Megabyte – MB) = $2^{20}$ octeţi |
| 1 Gigabit (Gbit) = $2^{30}$ biţi | 1 Gigaoctet (Gigabyte – GB) = $2^{30}$ octeţi |
| 1 Terabit (Tbit) = $2^{40}$ biţi | 1 Teraoctet (Terabyte – TB) = $2^{40}$ octeţi |
| 1 Petabit (Pbit) = $2^{50}$ biţi | 1 Petaoctet (Petabyte – PB) = $2^{50}$ octeţi |
| 1 Exabit (Ebit) = $2^{60}$ biţi | 1 Exaoctet (Exabyte – EB) = $2^{60}$ octeţi |
| 1 Zettabit (Zbit) = $2^{70}$ biţi | 1 Zettaoctet (Zettabyte – ZB) = $2^{70}$ octeţi |
| 1 Yottabit (Ybit) = $2^{80}$ biţi | 1 Yottaoctet (Yottabyte – YB) = $2^{80}$ octeţi |

Tipul reprezintă mulţimea valorilor pe care le poate lua o variabilă sau o constantă. Tipurile au un nume şi se împart în două mari categorii:

- tipuri standard – sunt deja definite în limbajul de programare C şi pot fi utilizate ca atare (`int`, `float`, `double`, `char`);
- tipuri definite de utilizator – în funcţie de contextul şi de cerinţele unei probleme, utilizatorul poate defini şi folosi ulterior propriile sale tipuri de date.

Principalele tipuri de date standard care pot fi folosite în limbajul de programare C (numite şi tipuri primare) sunt prezentate în tabelul 3.1.2. Alături de numele şi semnificaţia tipului, apare numărul de biţi pe care se face reprezentarea, precum şi domeniul în care pot lua valori datele definite utilizând acel tip. Un caz mai special este tipul întreg, a cărui reprezentare este dependentă de sistem.

Tabelul 3.1.2 Principalele tipuri de date ale limbajului C

| **Denumirea tipului** | **Semnificaţia tipului** | **Dimensiunea** | **Domeniul de valori** |
|---|---|---|---|
| *char* | caracter | 8 biţi | -128 – 127 |
| *int* | întreg | 16 sau 32 biţi | -32.768 – 32.767 sau<br>-2.147.483.648 – 2.147.483.647 |
| *float* | real (virgulă flotantă) | 32 biţi | 3.4 e-38 – 3.4 e+38 |
| *double* | real (virgulă flotantă, dublă precizie) | 64 biţi | 1.7 e-308 – 1.7 e+308 |

Alături de tipurile de date, apar deseori aşa-numiţii modificatori, prin intermediul cărora se poate specifica lungimea spaţiului de memorie alocat unui element. Aceştia sunt:

- *short* – scurt;
- *long* – lung;

- *signed* – cu semn;
- *unsigned* – fără semn.

Tabelul 3.1.3 conţine tipurile de date din C, împreună cu modificatorii posibili (au rezultat astfel tipurile derivate) [HS98], [DL06].

Tabelul 3.1.3 Tipurile de date şi modificatorii limbajului C

| **Denumirea tipului** | **Semnificaţia tipului** | **Dimen-siunea** | **Domeniul de valori** |
|---|---|---|---|
| *unsigned char* | caracter fără semn | 8 biţi | 0 – 255 |
| *char* sau *signed char* | caracter cu semn | 8 biţi | -128 – 127 |
| *unsigned int* | întreg fără semn | 16 sau 32 biţi | 0 – 65.535 sau<br>0 – 4.294.967.295 |
| *int* sau *signed int* | întreg cu semn | 16 sau 32 biţi | -32.768 – 32.767 sau<br>-2.147.483.648 – 2.147.483.647 |
| *unsigned short int* | întreg scurt fără semn | 8 biţi | 0 – 255 |
| *short int* sau *signed short int* | întreg scurt cu semn | 8 biţi | -128 – 127 |
| *unsigned long int* | întreg lung fără semn | 32 biţi | 0 – 4.294.967.295 |
| *long int* sau *signed long int* | întreg lung cu semn | 32 biţi | -2.147.483.648 – 2.147.483.647 |
| *float* | real (virgulă | 32 biţi | 3,4 e-38 – 3,4 e+38 |

| | | | |
|---|---|---|---|
| | flotantă) | | |
| *double* | real (virgulă flotantă, dublă precizie) | 64 biţi | 1,7 e-308 – 1,7 e+308 |
| *long double* | real lung (virgulă flotantă, dublă precizie) | 80 biţi | 3.4 e-4932 – 3.4 e+4932 |

Numele tipurilor de date şi ai modificatorilor reprezintă cuvinte cheie în limbajul de programare C, prin urmare ele sunt rezervate doar acestui scop. Nu se pot declara variabile sau constante care să poarte aceste nume şi de asemenea nu se pot crea noi tipuri de date sau funcţii care să fie denumite astfel. Lista cuvintelor cheie din C poate fi consultată în anexa 2.

### 3.1.2 Variabile

La declararea unei variabile, aşa cum s-a arătat, se va specifica tipul informaţiei care va fi reţinută în acea variabilă, precum şi numele ei. Forma generală a unei instrucţiuni prin intermediul căreia se declară variabile este:

```
tip var1,var2, …, varn;
```

În declaraţia de mai sus, `var1`, `var2`, …, `varn` sunt numele unor variabile, iar **`tip`** este denumirea unuia dintre tipurile prezentate în tabelul 3.1.3 sau un tip definit de utilizator. Variabilele sunt separate prin virgulă, iar declaraţia trebuie să se încheie cu punct şi virgulă.

Exemple:

```
int numar1, numar2, suma;
```

```
double media;
```

La declarare, există posibilitatea ca unei variabile să i se dea şi o valoare iniţială, care poate fi modificată pe parcursul execuţiei programului. De exemplu:

```
int contor=0;
```

Astfel se declară o variabilă de tip întreg, căreia i se dă numele `contor` şi care are valoarea iniţială zero.

### 3.1.3 Constante

Atunci când într-un program se foloseşte în mod repetat o anumită valoare, este bine ca acea valoare să fie declarată drept constantă. I se va atribui un nume şi va putea fi folosită pe tot parcursul programului prin intermediul acelui nume. E la fel ca în matematică unde pentru calcularea ariei unui cerc de exemplu, se va folosi constanta π în locul valorii 3.14159265…

Utilizarea constantelor într-un program are o serie de avantaje [CH96]. Cele mai importante sunt legate de faptul că datorită constantelor codul devine simplu de înţeles şi uşor de modificat. Dacă într-un program se foloseşte constanta π, este destul de clar pentru toată lumea că ea reprezintă raportul dintre circumferinţa unui cerc şi diametrul său în geometria euclidiană. Dacă însă este utilizată valoarea 3.1415, lucrurile nu mai sunt atât de evidente. În ceea ce priveşte modificarea valorilor constante folosite într-un program, totul e foarte simplu: modificare se face doar în locul unde constanta este declarată şi se reflectă oriunde este utilizată. În cazul în care nu se declară o constantă, ci se foloseşte valoarea ei pe tot parcursul programului, atunci codul este mai greu de modificat pentru că trebuie căutate şi înlocuite toate apariţiile acelei valori.

Înainte de a putea fi utilizate, constantele trebuie declarate. Acest lucru se realizează prin cuvântul cheie **`const`**. Forma generală a instrucţiunii prin care se declară o constantă este:

```
const tip nume_constanta = valoare;
```

unde:

- **`tip`** este tipul de dată al constantei (de exemplu `int`, `char`, `float` etc.);
- `nume_constanta` este denumirea cu care constanta va fi utilizată în program;
- `valoare` reprezintă valoarea pe care o va avea constanta.

În următorul exemplu se declară o constantă în virgulă mobilă care are numele `pi` şi valoarea `3.1415`:

```
const float pi = 3.1415;
```

Pe lângă constantele pe care utilizatorul şi le defineşte, în limbajul de programare C pot fi folosite şi o serie de constante simbolice care sunt predefinite. De exemplu există o constantă cu numele `M_PI` şi valoarea `3.14159265`….

### 3.1.4 Operatori

Variabilele şi constantele pot fi incluse în diverse operaţii pentru a obţine noi valori. În acest sens se vor utiliza operatori, cu ajutorul cărora variabilele şi constantele vor forma expresii. În funcţie de felul operaţiei în care sunt implicaţi, operatorii se împart în mai multe categorii. Cei care sunt utilizaţi mai frecvent vor fi prezentaţi în continuare [CH96], [DL06], [GP00].

- **operatori relaţionali** – rezultatul aplicării lor este `1` în cazul în care comparaţia este adevărată şi `0` în caz contrar:

  < mai mic (`a<b`);

  <= mai mic sau egal (`a<=b`);

  > mai mare (`a>b`);

  >= mai mare sau egal (`a>=b`);

  == egal (`a==b`);

  != diferit (`a!=b`).

- **operatori aditivi**:

  + plus (adunare: `a=b+c`);

  − minus (scădere: `a=b-c`).

- **operatori multiplicativi**:

  * înmulţire (`a=b*c`);

  / împărţire – în cazul împărţirii a două numere întregi, oferă ca rezultat câtul; dacă cel puţin unul dintre operanzi este număr real, rezultatul este real (`a=b/c`);

  % modulo – operator care furnizează drept rezultat restul împărţirii a două numere întregi (`a=b%c`); nu poate fi folosit cu operanzi reali.

- **operatori compuşi** – se folosesc pentru scrierea mai rapidă a unor expresii:

  += adunare al cărei rezultat se salvează în primul operand (de exemplu: `x+=y` este echivalent cu `x=x+y`);

`-=` scădere al cărei rezultat se salvează în primul operand (de exemplu: `x-=y` este echivalent cu `x=x-y`);

`*=` înmulţire al cărei rezultat se salvează în primul operand (de exemplu: `x*=y` este echivalent cu `x=x*y`);

`/=` împărţire al cărei rezultat se salvează în primul operand (de exemplu: `x/=y` este echivalent cu `x=x/y`);

`%=` operaţia modulo, al cărei rezultat se salvează în primul operand (de exemplu: `x%=y` este echivalent cu `x=x%y`).

- **operatori logici**:

  `!` operatorul de negare (`p=!q`);

  `&&` ŞI logic, numit şi conjuncţie logică (`p=p1&&p2`);

  `||` SAU logic, numit şi disjuncţie logică (`p=p1||p2`).

- **alţi operatori**:

  = operatorul de atribuire – copiază valoarea din dreapta sa în variabila din stânga sa; se formează astfel o expresie, obţinându-se o nouă valoare (de exemplu: `x=7;` sau `i=i+2;`);

  `++` operatorul de incrementare – este un operator unar (adică are nevoie doar de un element), se aplică doar asupra variabilelor de tip întreg şi are ca efect mărirea cu o unitate a valorii variabilei respective (`x++` sau `++x` sunt echivalente cu `x=x+1`). Operatorul poate apărea înainte sau după numele variabilei; diferenţa e că variabila îşi schimbă valoarea înainte, respectiv după utilizare;

  `--` operatorul de decrementare – micşorează cu o unitate valoarea variabilei asupra căreia este aplicat (`x--` sau `--x` sunt echivalente cu `x=x-1`) şi are aceleaşi caracteristici ca operatorul de incrementare;

  `&` operatorul de adresare – indică adresa de memorie a unei variabile şi este plasat în faţa numelui variabilei (`&x`);

  `?:` operatorul condiţional: `a?e1:e2` – dacă `a` e diferit de zero (sau adevărat), se execută expresia `e1`; dacă `a` e zero (sau fals) se execută expresia `e2`;

  `()` apel de funcţie – se realizează prin numele funcţiei, urmat eventual de o listă de argumente (de exemplu: `afisare(a,b)`);

`(tip)` conversie de tip – operator folosit pentru a schimba tipul unei variabile (de exemplu, dacă se consideră variabila `a` de tip real, prin executarea instrucţiunii `(int)a`, se obţine o valoare întreagă);

`sizeof()` operator unar care calculează dimensiunea unui tip de dată, adică numărul de octeți necesari pentru a reprezenta o variabilă de acel tip (de exemplu, `sizeof(int)`).

Ordinea în care se execută operaţiile (denumită şi precedenţa operatorilor) este cea cunoscută din matematică. În tabelul 3.1.4 se regăsesc operatorii prezentaţi în acest paragraf în ordinea precedenţei pe care o au, începând cu cei mai puternici şi încheind cu cei mai slabi [BK03], [HS98].

Tabelul 3.1.4 Precedenţa operatorilor

| **Categoria** | **Operatorii** |
|---|---|
| unari | `++`, `--`, `(tip)` , `sizeof()` |
| multiplicativi | `*`, `/`, `%` |
| aditivi | `+`, `-` |
| relaţionali | `<`, `<=`, `>`, `>=` |
| de egalitate | `==`, `!=` |
| ŞI logic | `&&` |
| SAU logic | `\|\|` |
| condiţional | `?:` |
| de atribuire | `=`, `+=`, `-=`, `*=`, `/=`, `%=` |

Pentru a evita greşelile care pot apărea din cauza unor neînţelegeri legate de precedenţa operatorilor, e indicat a se folosi paranteze atunci când sunt create expresii mai ample. De exemplu, dacă se doreşte calcularea expresiei $x = \frac{7+y}{2z}$, e

indicat să se scrie $x = (7 + y)/(2 * z)$. Această expresie este diferită de $x = (7 + y)/2 * z$ care calculează de fapt $x = \frac{7+y}{2} \cdot z$.

Precedenţa operatorilor are un rol foarte important în evaluarea expresiilor, iar cunoaşterea ordinii în care operatorii sunt aplicaţi este obligatorie. De exemplu expresia `(1||1&&0)` are valoarea `1` deoarece operatorul `&&` este aplicat înaintea operatorului `||`[PT12]. Dacă ar fi invers, valoarea expresiei s-ar schimba.

### 3.1.5 Expresii

Într-un limbaj de programare expresiile reprezintă, la fel ca în matematică, operaţii care se fac asupra unor operanzi (variabile, constante sau alte expresii) utilizând operatori. În acest fel se obţine o valoare nouă, al cărei tip este dependent de tipul operanzilor. Forma generală a unei expresii este următoarea:

E = expresie_simplă [operator_relaţional altă_expresie_simplă]

Încadrarea unor entităţi în paranteze drepte indică faptul că acele entităţi au caracter opţional. Aşadar, o expresie este formată din una sau mai multe expresii simple conectate prin operatori relaţionali. La rândul său, o expresie simplă conţine termeni legaţi prin operatori aditivi. Un termen are mai mulţi factori uniţi prin operatori multiplicativi. În fine, un factor poate fi: o constantă, o variabilă (reprezentată de valoarea sa), adresa unei variabile (operatorul de adresare urmat de numele variabilei), o variabilă negată, un apel de funcţie, o transformare de tip sau o altă expresie simplă încadrată în paranteze rotunde.

În funcţie de rezultatul pe care îl produc, expresiile pot fi împărţite în trei categorii [CH96]:

- **expresii matematice**

Acestea au întotdeauna ca rezultat un număr şi sunt alcătuite din valori conectate prin intermediul unor simboluri reprezentând operaţii matematice. Pentru a specifica ordinea efectuării operaţiilor se pot folosi paranteze.

De exemplu:

expresia `3+4*2` va avea ca rezultat valoarea `11`;

în schimb `(3+4)*2` este `14`.

- **expresii de tip text**

Rezultatul acestora este un şir de caractere, adică o secvenţă de caractere care poate include cifre, litere, semne de punctuaţie şi alte simboluri (de exemplu `"Dennis M. RITCHIE"`). Din păcate limbajul C nu dispune de operatori pentru prelucrarea şirurilor de caractere. În locul operatorilor, în expresiile de tip text se folosesc funcţii specifice, care vor fi prezentate în capitolul dedicat şirurilor de caractere.

- **expresii logice**

Pot avea ca rezultat valoarea `1` sau valoarea `0`, cu semnificaţia `adevărat`, respectiv `fals` şi sunt create cu ajutorul operatorilor relaţionali. O expresie logică este un fel de întrebare la care răspunsul este „da” sau „nu”.

De exemplu:

`5>1` este o expresie adevărată, deci va avea ca rezultat valoarea `1`;

`7==0` este falsă, aşadar rezultatul evaluării expresiei este `0`;

`a>=b` poate fi `0` sau `1`, în funcţie de valorile celor două variabile luate în considerare.

Rezultatele expresiilor logice, precum şi elemente care pot fi interpretate logic, pot fi combinate cu ajutorul operatorilor logici, obţinându-se astfel condiţii complexe. Pentru aceasta trebuie în primul rând cunoscut modul în care acţionează operatorii logici. În tabelul 3.1.5 se consideră două variabile logice, `a` şi `b`, şi se aplică operatorii AND (ŞI), OR (SAU) şi NOT (negare), evidenţiindu-se toate rezultatele posibile.

Tabelul 3.1.5 Aplicarea operatorilor logici

| **a** | **b** | **!a** | **a&&b** | **a\|\|b** |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |

De exemplu:

expresia `(7>=5)&&(0!=10)` conţine două expresii adevărate, legate prin operatorul logic ŞI; expresia finală este adevărată, deci va avea ca rezultat valoarea `1`;

expresia `(7==0)&&(3<4)` este formată dintr-o expresie falsă (prima) şi una adevărată (a doua), legate prin ŞI logic; rezultatul este fals, adică `0`;

expresia `(7==0)||(3<4)` conţine aceleaşi două expresii ca precedenta (una adevărată şi una falsă), dar legate prin operatorul SAU logic; prin urmare se obţine valoarea `1` pentru că este suficient ca cel puţin una din expresii să fie adevărată pentru ca expresia finală să fie adevărată, aşa cum rezultă şi din tabelul 3.1.5;

expresia `!(7==0)` este adevărată deoarece se neagă o expresie falsă.

## 3.2 Descrierea tipurilor de date

Următoarele paragrafe prezintă tipurile de date standard care au fost descrise succint anterior. De asemenea sunt aduşi în discuţie operatorii care pot fi aplicaţi fiecărui tip. La sfârşitul paragrafelor sunt prezentate exemple de utilizare a respectivelor tipuri de dată.

### 3.2.1 Tipul întreg

În C există mai multe tipuri prin intermediul cărora se pot reprezenta numere întregi, fiecare dintre aceste tipuri fiind un subset al mulţimii numerelor întregi (Z) din matematică. Descrise succint în tabelul 3.1.3, tipurile întregi vor fi detaliate în cele ce urmează.

***unsigned int*** este utilizat pentru numere întregi fără semn. Un astfel de număr se reprezintă pe 16 biţi şi poate lua valori în intervalul 0 – 65.535. În C există definită o constantă prin intermediul căreia se poate determina care este limita maximă a acestui interval. Este vorba de constanta `UINT_MAX` a cărei valoare este 65.535. Acest tip poate fi reprezentat și pe 32 de biți, putând lua valori în intervalul 0 – 4.294.967.295.

***int*** sau ***signed int*** sunt tipurile prin care se reprezintă numere întregi cu semn. Pentru aceasta sunt necesari tot 16 biţi, domeniul în care pot lua valori

astfel de numere fiind -32.768 – 32.767. Constantele C care delimitează acest domeniu sunt `INT_MIN`, având valoarea -32.768, respectiv `INT_MAX`, cu valoarea 32.767. Așa cum s-a specificat și în tabelul 3.1.3, pot exista și reprezentări pe 32 de biți ale acestor tipuri, cu valori între -2.147.483.648 și 2.147.483.647.

***unsigned short int*** este tipul utilizat pentru numere întregi scurte, fără semn. Pe cei 8 biţi dedicaţi se pot reprezenta numere în intervalul 0 – 255. Pentru a accesa într-un program limita superioară a acestui interval se poate folosi constanta `USHRT_MAX` care are valoarea 255.

***short int*** sau ***signed short int*** definesc numere întregi scurte, cu semn. Aceste tipuri se reprezintă pe 8 biţi, iar numerele iau valori în domeniul -128 – 127. Limitele acestui domeniu pot fi aflate prin constantele `SHRT_MIN`, care are valoarea -128, respectiv `SHRT_MAX`, cu valoarea 127.

***unsigned long int*** este un tip de data utilizat pentru numere întregi mari şi pozitive. Pe cei 32 de biţi aflaţi la dispoziţie se pot reprezenta numere în intervalul 0 – 4.294.967.295. Constanta `ULONG_MAX` reţine limita superioară a acestui interval.

***long int*** sau ***signed long int*** se folosesc pentru numere întregi de dimensiuni mari, pozitive sau negative. Reprezentarea se face pe 32 de biţi, iar numerele se găsesc în domeniul -2.147.483.648 – 2.147.483.647. Pentru a accesa limitele acestui domeniu se pot folosi constantele `LONG_MIN` şi `LONG_MAX`.

Toate constantele care specifică limitele domeniilor de reprezentare pentru aceste tipuri de date sunt definite în biblioteca `limits.h`. Detalii despre utilizarea bibliotecilor limbajului C vor fi date în capitolul dedicat funcţiilor de bibliotecă.

Atunci când se scriu programe, variabilele trebuie definite în mod corespunzător. De exemplu, dacă se doreşte reprezentarea vârstei unei persoane, aceasta poate fi o variabilă de tip `unsigned short int` pentru că vârsta este totdeauna pozitivă şi, de asemenea, este un număr scurt. Dacă se lucrează însă cu o forţă, prin definirea variabilei trebuie să se asigure faptul că acesteia i se pot da atât valori pozitive, cât şi negative şi, de asemenea, domeniul de valori trebuie să fie suficient de mare (`int` de exemplu).

Datele de tip întreg pot fi implicate în diverse operaţii. Astfel, se pot realiza operaţii cu caracter relaţional (<, <=, >, >=, ==, !=), operaţii aritmetice (+, −,

*, /, %) sau operaţii logice (&&, ||). De asemenea, variabilele întregi pot fi incrementate (++) sau decrementate(--).

**Utilizarea tipului întreg**

➢ În cele ce urmează se vor declara, utilizând tipul întreg, trei variabile (x, a şi b) şi o constantă (lambda). Constanta are, evident, o valoare iniţială (în cazul acesta 3). Variabilele nu sunt iniţializate la declarare; a şi b primesc valori ulterior (2, respectiv -7), iar valoarea lui x este calculată prin intermediul unei expresii. Rezultatul acestei expresii este -15.

```
const unsigned int lambda = 3;
int x, a, b;
a = 2; b = -7;
x = lambda * (a + b);
```

➢ Variabilele de tip întreg sunt frecvent utilizate împreună cu operatorii de incrementare şi decrementare. Aceşti operatori pot apărea înaintea variabilei sau după ea, diferenţa fiind evidenţiată în cele două secvenţe de cod care urmează. Se declară două variabile de tip întreg a şi x. Cea de a doua variabilă, x, primeşte valoarea 7. Diferenţa apare în momentul în care variabilei a i se atribuie valoarea lui x incrementată după atribuire (în primul caz) şi înainte de atribuire (în al doilea caz).

```
int a, x;
x = 7;
a = x++;
```

În urma rulării acestor trei linii de cod rezultatele sunt a=7 şi x=8 pentru că incrementarea lui x s-a realizat după operaţia de atribuire.

```
int a, x;
x = 7;
a = ++x;
```

Deoarece incrementarea lui x se face înaintea operaţiei de atribuire, în acest al doilea caz rezultatele sunt a=8 şi x=8.

## 3.2.2 Tipul real

Numerele reale (mulţimea R din matematică) pot fi introduse într-un program C prin intermediul tipurilor float, double şi long double descrise în tabelul 3.1.3. Aceste tipuri oferă posibilitatea de a utiliza numere reale, de diferite precizii şi dimensiuni, ajungând până la valoarea 3.4 e+4932.

**`float`** este un tip de dată prin intermediul căruia se reprezintă, pe 32 de biţi, numere reale cu o precizie de 6 zecimale. Dacă e nevoie de o acurateţe mai mare, atunci numerele se pot declara **`double`** (cu reprezentare pe 64 de biţi), iar pentru a extinde precizia se poate utiliza tipul **`long double`** (care are o reprezentare pe 80 de biţi).

Variabilele de tip real se pot introduce în expresii alături de operatori relaţionali (<, <=, >, >=, ==, !=) şi aritmetici (+, -, *, /). În cazul operaţiilor cu caracter relaţional, rezultatul este o valoare care poate fi interpretată ca adevărată sau falsă (de exemplu `7.14 < 17.2` este o expresie adevărată, iar `3.5 == 7.8` este falsă).

**Utilizarea tipului real**

➢ În secvenţa următoare de instrucţiuni se declară, şi se iniţializează totodată, două variabile de tip `float` care vor reţine preţul, respectiv cantitatea dintr-un anumit produs vândut într-un interval de timp. O a treia variabilă, de acelaşi tip, este declarată, iar apoi valoarea ei se calculează ca produs al primelor două, reprezentând suma totală încasată.

```
float pret = 4.5, cantitate = 125.75, total;
total = pret * cantitate;
```

### 3.2.3 Tipul caracter

Această categorie de date constituie mulţimea tuturor caracterelor posibile. Ele se pot reprezenta pe 8 biţi, cu sau fără semn, prin intermediul a două tipuri de date (***`unsigned char`*** şi ***`char`*** sau ***`signed char`***) ale căror proprietăţi sunt descrise în tabelul 3.1.3.

Pe cei 8 biţi disponibili se pot codifica 256 de caractere. Toate caracterele sunt convertite la numere întregi conform codului ASCII (*American Standard Code for Information Interchange*). În anexa 1 se regăsesc câteva dintre aceste caractere, împreună cu codurile ASCII asociate.

**Utilizarea tipului caracter**

➢ Pentru a exemplifica felul în care se poate lucra cu caracterele într-un program C, se declară şi se iniţializează câteva caractere.

```
char c1 = 'A';
char c2 = '7';
```

Aceste caractere pot fi supuse unor operaţii standard:

```
char c3;
c3 = c1 + c2;
```

În urma executării operaţiei de adunare, variabila `c3` va avea valoarea `'x'` deoarece se adună de fapt codurile ASCII asociate variabilelor `c1` şi `c2`.

➢ Asupra caracterelor se pot aplica şi operatori relaţionali. Astfel, `'D'<'b'` e o evaluare al cărei rezultat este adevărat, iar `'8'<'6'` e fals. Asta pentru că se compară codurile ASCII asociate caracterelor respective.

### 3.2.4 Tipul logic

În limbajul C nu există un tip de dată standard dedicat informaţiilor cu caracter logic, aşa cum există tipul boolean în limbaje precum Pascal sau Java. Pentru a reprezenta valoarea de adevăr a unei expresii sau orice element cu caracter logic în limbajul de programare C se vor folosi valori numerice. Astfel, zero înseamnă fals şi orice altceva înseamnă adevărat.

Asupra elementelor de tip logic se pot aplica operatori relaţionali (==, <, >, <=, >=, !=), rezultatul fiind tot de tip logic. De asemenea, operatorii logici (şi, sau, negaţie) se pot aplica asupra datelor cu caracter logic.

**Utilizarea tipului logic**

➢ Se consideră o variabilă `x`, de tip întreg. La un moment dat, pe parcursul execuţiei unui program, trebuie verificat dacă `x` este un număr par. Un număr este par dacă el este divizibil cu 2, adică dacă restul împărţirii la 2 este 0. Pentru a verifica acest lucru se foloseşte operatorul `%` (*modulo*). Expresia `x%2==0` va avea valoarea `adevărat` dacă şi numai dacă `x` este par. Rezultatul acestei expresii este aşadar o valoare logică.

➢ Fie două variabile `x` şi `y` de tip întreg. Se doreşte scrierea unei expresii care să aibă valoarea `adevărat` dacă şi numai dacă cel puţin una dintre cele două variabile are valoare strict pozitivă. Respectiva expresie va arăta astfel: `(x>0)||(y>0)`.

➢ Variabila `nota` de tip `float` înregistrează nota obţinută de un student la un anumit examen. Pentru a verifica dacă acea notă se află în intervalul (8; 10] se va scrie expresia `(nota>8)&&(nota<=10)`.

### 3.2.5 Tipul void

Acesta nu este chiar un tip de dată sau poate fi considerat un tip de dată vid, care nu are un domeniu de valori. Din punct de vedere sintactic el e situat acolo unde, în mod normal, este aşteptat un tip de dată. `Void` este folosit pentru a specifica faptul că o funcţie nu returnează nimic (de exemplu `void funtie1(int x)`). De asemenea el poate apărea şi ca unic argument în prototipul unei funcţii arătând astfel că funcţia nu primeşte niciun argument (de exemplu `float functie2(void)`).

### 3.2.6 Conversia de tip

Există situaţii în care e nevoie ca tipul unei entităţi să fie schimbat (de obicei pentru a beneficia de avantajele oferite de anumite proprietăţi ale tipurilor). Acest lucru se poate realiza prin intermediul unei conversii de tip. Există două astfel de conversii – implicită şi explicită [BK03], [HS98].

**Conversia implicită** se realizează în mod automat de către compilator şi este necesară atunci când într-o expresie apar date de tipuri diferite. Această conversie are două direcţii.

Pe de o parte conversia presupune modificarea unui tip de dată într-unul care îl include pe cel iniţial, aşa cum se întâmplă în exemplul următor. Variabila `b`, de tip `int` este convertită la `float`, iar rezultatul obţinut este `x = 9.47`.

```
float a = 2.47, x;
int b = 7;
x = a + b;
```

În acest caz, la evaluarea expresiei s-a căutat tipul cel mai cuprinzător. Toate elementele expresiei au fost convertite la acest tip. Ordinea tipurilor de date, începând cu cel mai cuprinzător este: `double`, `float`, `long`, `int`, `char` [CH96].

Pe de altă parte, dacă este vorba de o instrucţiune de atribuire, datele sursă vor fi forţate la tipul destinaţiei [CH96], chiar dacă acest lucru duce la trunchierea informaţiei. Exemplul precedent a fost modificat, considerându-se rezultatul `x` de tip întreg. Drept consecinţă, toate datele din partea dreaptă a semnului egal sunt forţate să devină variabile întregi, rezultatul fiind `x = 9`.

```
float a = 2.47;
int b = 7, x;
```

```
x = a + b;
```

**Conversia explicită** este introdusă clar în cadrul programului prin intermediul unui modificator de tip. Acesta este un operator unar care apare sub forma unui prefix la o variabilă sau la o expresie şi prin care se forţează transformarea informaţiei respective la tipul specificat. Forma generală de realizare a conversiei explicite este:

```
(tip)expresie;
```

Utilizarea unei astfel de conversii în cadrul unui program este foarte simplă. De exemplu, se consideră o variabilă `a` de tip `int` care memorează codul ASCII al unui caracter. O variabilă `x` de tip `char` va reţine caracterul asociat codului ASCII specificat, transformând conţinutul variabilei `a` din `int` în `char`.

```
int a=65;
char x;
x=(char)a;
```

## 3.3 Probleme propuse

A. Specificaţi unde este greşeala.

1. ```
   const int alfa=2;
   int beta=3;
   alfa=beta+10;
   ```
2. ```
   int a=7, double=2, x;
   x=double*a;
   ```
3. ```
   unsigned int a=22, b=35;
   x=a+b;
   ```
4. ```
   unsigned int x=12, y=-15, q;
   q=x-y;
   ```
5. ```
   float a=2, b=-3;
   x=-a/b;
   float x, y;
   y=2*x+a-b;
   ```
6. ```
   int x=1, y=2, z=3, e;
   e=x+2y-z;
   ```

7. `float x=7.5, y;`
   `y=x%2;`
8. `float 1_nota, 2_nota, final_nota;`
   `final_nota=(2*1_nota+2_nota)/final_nota;`

B. Fiind date definiţiile `int a=3, b=4;`, evaluaţi următoarele expresii şi menţionaţi care este rezultatul.

9. `E=a/5+1;`
10. `E=(a++)-(a%2);`
11. `E=(a++)-(--b);`
12. `E=(-a)-(--a);`
13. `E=(a==b);`
14. `E=(a+b)%4;`
15. `E=(((--b)+a)++)%2;`

C. Alegeţi răspunsul corect (unul singur).

16. Care dintre următoarele tipuri nu există în limbajul de programare C?

    □ a) `int`; □ b) `float`; □ c) `double`; □ d) `char`; □ e) toate răspunsurile sunt greşite.

17. Care dintre următoarele declaraţii de variabile este greşită?

    □ a) `float nota_unu;` □ b) `float nota unu;` □ c) `float notaunu;` □ d) toate declaraţiile sunt corecte; □ e) toate declaraţiile sunt greşite.

18. Ce valoare are expresia `E=((2+4)*3-1)*(3-5)`?

    □ a) -34; □ b) 0; □ c) -24; □ d) 24; □ e) expresia conţine o greşeală.

19. Ce valoare va avea variabila `x` după executarea următoarei secvenţe de instrucţiuni?

    ```
    int a=65, x;
    x=a/2;
    ```

    □ a) 32.5; □ b) 1; □ c) 32; □ d) 65; □ e) toate răspunsurile sunt greşite.

20. Ce valoare are expresia `E = 2*(7-(9+5)+4*(21-19)`?

□ a) 8; □ b) -6; □ c) -12; □ d) 2; □ e) expresia conţine o greşeală.

21. Ce valoare va avea variabila `m` după executarea următoarelor linii de cod?

```
double a=7, m;
m=a%2;
```

□ a) 3.5; □ b) 1; □ c) 14; □ d) 49; □ e) atribuirea `m=a%2` este imposibilă.

22. Care dintre următoarele expresii este adevărată dacă `x=1` şi `y=3`?

□ a) `(x>=0)&&(y<3);` □ b) `(x>=0)||(y<3);` □ c) `((x==0)||(y!=3))&&(x!=y);` □ d) `(x==y)||(y<=0);` □ e) toate expresiile sunt false.

23. Care dintre următoarele declaraţii de variabile este corectă?

□ a) `int n1#;` □ b) `int #n1;` □ c) `int n1_#;` □ d) `int #_n1;` □ e) toate declaraţiile sunt greşite.

24. Care dintre următoarele cuvinte nu reprezintă nume de modificatori în limbajul de programare C?

□ a) `short`; □ b) `long`; □ c) `signed`; □ d) `unsigned`; □ e) toate răspunsurile sunt greşite.

25. Care dintre următoarele tipuri nu există în limbajul de programare C?

□ a) `int`; □ b) `float`; □ c) `boolean`; □ d) `char`; □ e) toate răspunsurile sunt greşite.

26. Câţi biţi formează un octet?

□ a) 1; □ b) 2; □ c) 4; □ d) 8; □ e) un octet nu este alcătuit din biţi.

27. Ce valoare va avea variabila `x` după executarea următoarelor instrucţiuni?

```
float a=3.1415;
int x=0;
x=(int)a;
```

□ a) 3; □ b) 3.14; □ c) 3.1415; □ d) 0; □ e) atribuirea `x=(int)a` este imposibilă.

28. Care dintre următoarele expresii este falsă dacă `a=1`, `b=-3` şi `c=7`?

□ a) `!((a+b+c)<0);` □ b) `!((a<b)||(b>c));` □ c) `((a+b)>c)||((a!=0)&&(b!=0));` □ d) `!(a>b);` □ e) toate expresiile sunt adevărate.

29. Care dintre următoarele tipuri nu există în limbajul de programare C?

□ a) `int`; □ b) `float`; □ c) `string`; □ d) `double`; □ e) toate răspunsurile sunt greşite.