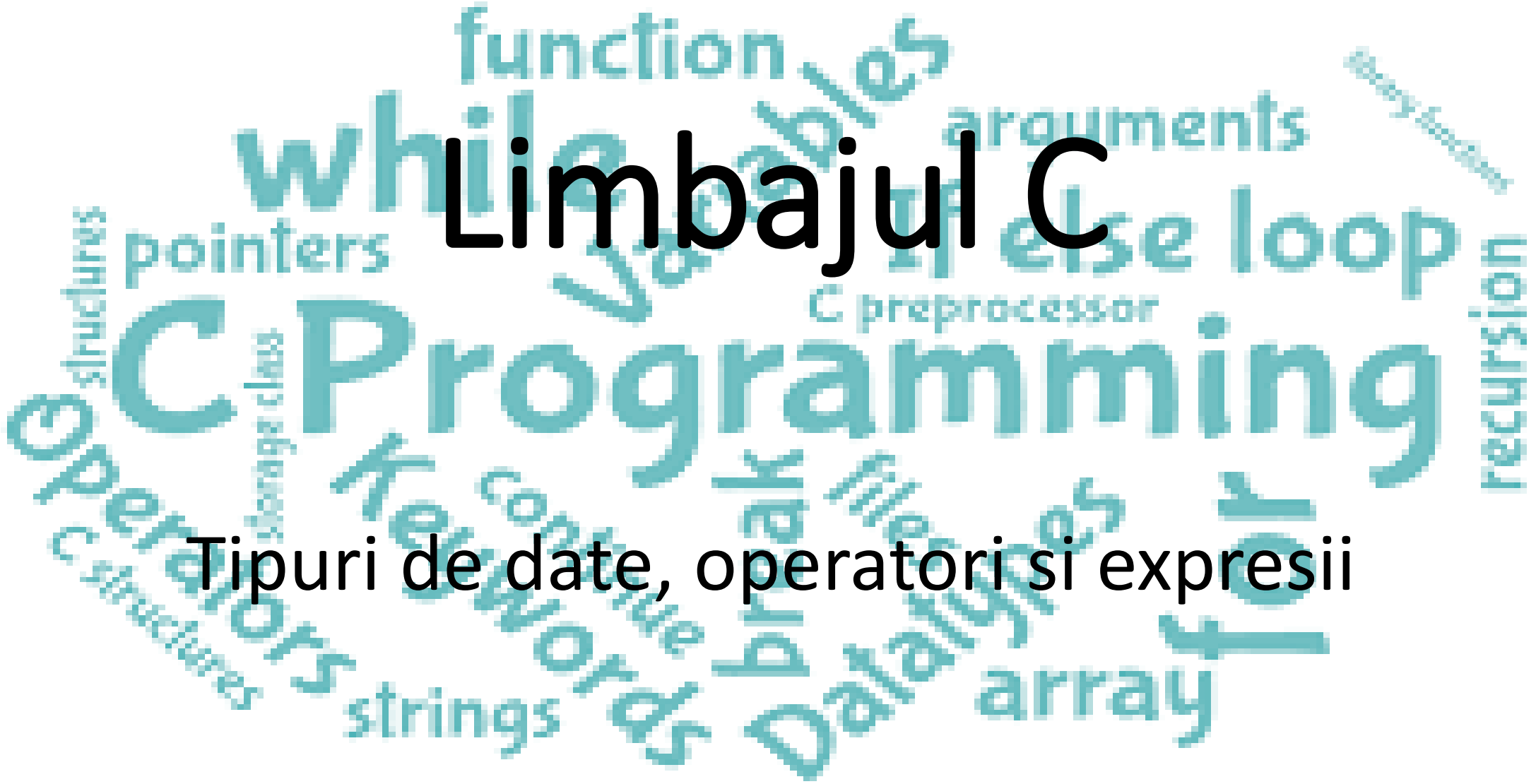

Limbajul C

Tipuri de date, operatori si expresii

Cuprins

- Nume de variabile
- Tipuri de date si dimensiunea lor
- Constante
- Declaratii
- Operatori
  - Aritmetici
  - Relationali
  - Logici pe biti
  - Asignare (atribuire)
  - Incrementare si decrementare
- Conversii de tip
- Expresii conditionale
- Ordinea de evaluare (precedenta operatorilor)

Nume de variabile

- Numele sunt alcatuite din litere si cifre;
- Primul caracter trebuie sa fie o litera;
- Liniuta de subliniere "_" este considerata litera;
- Literele mari si mici sunt caractere distincte;
- Numai primele opt caractere ale unui nume intern sunt semnificative;

Tipuri de date si dimensiunea lor

- ***char*** - un singur octet, capabil sa memoreze un caracter din setul local de caractere;
- ***int*** - un intreg, reflectand tipic marimea efectiva a intregilor pe calculatorul gazda;
- ***float*** - numar flotant (virgula mobila) in simpla precizie;
- ***double*** - numar flotant (virgula mobila) in dubla precizie;
- Exista un numar de calificatori care pot fi aplicati tipului "int": short, long si unsigned;

Tipuri de date si dimensiunea lor

- Exemple:
  - short int x;
  - long int y;
  - unsigned int z;
- Intentia e ca ***short*** si ***long*** sa aiba lungimi diferite de intregi;

Constante

- Exemple: 123.456e-7, 0.12E3, 123L;
- O constanta intreaga normala care este prea lunga pentru un **int**, este considerata ca fiind o constanta ***long***;
- Exista o notatie speciala pentru constantele octale si hexazecimale:
  - un 0 (zero) la inceputul unei constante ***int*** inseamna octal;
  - un 0x sau 0X la inceputul unei constante ***int*** inseamna hexazecimal.
- De exemplu, numarul zecimal 31 poate fi scris 037 in octal si 0x1f sau 0X1F in hexazecimal;
- Constantele octale si hexazecimale pot fi urmate un L pentru a le face "long”;
- O constanta caracter este un caracter singur scris intre ghilimele simple, de exemplu, 'x’;
- Constantele caracter participa in operatiile numerice la fel ca oricare alte numere;

Constante

- Anumite caractere negrafice pot fi reprezentate drept constante caracter cu ajutorul secventelor escape, de exemplu:
  - \n linie noua
  - \t (tab)
  - \0 (nul)
  - \\ (backspace)
  - \'
- O expresie constanta este o expresie care implica numai constante:

```
#define MAXLINE 1000
char line[MAXLINE+1];
```

Constante

- “I am a string”
- “” /* un sir nul */
- Compilatorul plaseaza automat un caracter nul (\0) la sfarsitul oricarui sir de caractere, astfel ca programele pot determina usor sfarsitul sirului;

```
int strlen(char[]s)  /* returneaza lungimea lui s */
{
     int i;
     i = 0;
     while (s[i] != '\0')
          i = i + 1;
     return(i);
}
```

Declaratii

- In C toate variabilele trebuie declarate inainte de a fi folosite!!!

```
int lower, upper, step;
char c, line[1000];
```

- Sau

```
int lower;
int upper;
int step;
char c;
char line[1000];
```

Declaratii

- Variabilele pot fi initializate in declaratia lor:

```
char backslash = '\\';
int i = 0;
float eps = 1.0e-5;
```

- Variabilele externe sau statice se initializeaza o singura data (implicit se initializeaza cu zero);
- Variabilele automate initializate explicit sunt initializate la fiecare apel al functiei in care sunt continute;

Operatori

- Tipuri de operatori in C:
  - Aritmetici: unari sau binari
  - De atribuire
  - De incrementare/ decrementare
  - Relaţionali
  - Logici
  - Deplasare pe biti
  - Conversia de tip (cast)

Operatori aritmetici

- Operatorii aritmetici binari sint: "+", "-", "*", "/" si operatorul modulo "%";
- Atentie: operatorul % poate fi aplicat numai în expresii care conţin variabile de tip întreg!
- Operatorul % nu poate fi aplicat la float sau double!!!
- x % y produce restul impatirii lui x la y si va fi zero cand impartirea este exacta;
- Operatorii + si - au aceeasi prioritate, care este mai mica decat prioritatea lui *, / si % care la randul ei este mai mica decat prioritatea operatorului unar -;
- Operatorii aritmetici se grupeaza de la stanga la dreapta;

Operatori relationali si logici

- Operatorii relationali sunt: >, >=, <, <=, ==, !=
- Operatorii relationali au prioritatea mai mica decat cei aritmetici;
- Exemplu: i < lim-1 se evalueaza ca i < (lim-1)
- Operatorii logici: && (AND) || (OR)
- Expresiile care contin operatorii logici sunt evaluate de la stanga la dreapta si evaluarea se opreste in clipa in care se cunoaste valoarea de adevarat sau fals a rezultatului;

```
for (i=0; i<lim-1 && (c = getchar()) != '\n' && c != EOF;++i)
            s[i] = c;
```

Operatori relationali si logici

- Prioritatea lui && este mai mare decat cea a lui || si amandoua sunt mai mici decat cele ale operatorilor relationali si de egalitate;

```
i<lim-1 && (c = getchar()) != '\n' && c != EOF
```

- Deoarece prioritatea lui != este mai mare decat cea a asignarii, este nevoie de paranteze in:

```
(c = getchar()) != '\n'
```

Operatori relationali si logici

- Operatorul unar de negatie "!" converteste un operand nonzero sau adevarat in zero si un operand zero sau fals in 1.
- O utilizare obisnuita a lui ! este in constructii de tipul

```
if (!inword)
```

- Care este echivalent cu:

```
if (inword == 0)
```

Operatori de incrementare si decrementare

- Operatorul de incrementare ++ aduna 1 la operandul sau;
- Operatorul de decrementare -- scade 1;
- Ei pot fi folositi atat ca operatori prefix (++n) cat si ca operatori sufix (n++);
- Deosebirea: ++n il incrementeaza pe n inainte de a-i folosi valoarea, n++ il incrementeaza pe n dupa ce a fost folosita valoarea lui;
- Exemplu: daca n este 5, atunci:

```
x = n++;
```

il face pe x egal cu 5, dar

```
x = ++n;
```

il face pe x egal cu 6

Operatori de incrementare si decrementare

- Operatorii de incrementare si decrementare se pot aplica numai variabilelor;
- O expresie de tipul x = (i+j)++ este incorecta!!!
- Exemple:

```
void squeeze (char[] s, int c)  /* sterge toate aparitiile lui c din s */
{
      int i, j;
      for (i = j = 0; s[i] != '\0'; i++)
            if (s[i] != c)
                  s[j++] = s[i];
      s[j] = '\0';
}
```

Operatori de incrementare si decrementare

```
if (c == '\n') {
     s[i]=c;
     ++i;
}
```

- Echivalent cu

```
if (c == '\n')
     s[i++] = c;
```

Operatori de incrementare si decrementare

- Functia strcat(s, t) concateneaza sirul t la sfarsitul sirului s (strcat presupune ca exista suficient spatiu in s pentru a pastra combinatia)

```
void strcat(char s[],t[]) /* concateneaza pe t la sfarsitul lui s */
{
      int i, j;
      i = j = 0;
      while (s[i] != '\0')  /* gaseste sfirsitul lui s */
            i++;
      while ((s[i++] = t[j++]) != '\0') /* copiaza pe t */
            ;
}
```

Operatori logici si de deplasare pe biti

- Limbajul C ofera un numar de operatori pentru manipularea bitilor; acestia nu se pot aplica variabilelor de tipul **float** sau **double**.

```
 &      SI bit cu bit
 |      SAU inclusiv bit cu bit
 ^      SAU exclusiv bit cu bit
<<      deplasare stanga
>>      deplasare dreapta
 ~      complement fata de 1 (unar)
```

Operatori logici si de deplasare pe biti

- Exemple:

```
n = n | 0177;
```

pune pe 0 toti bitii lui n, mai putin ultimii 7 biti!

- Operatorul SAU bit cu bit "|" este folosit de regula pentru a seta pe 1 anumiti biti:

```
x = x | MASK;
```

seteaza pe 1 in x bitii care sunt sunt egali cu 1 in MASK!

- ATENTIE: daca x este 1 si y este 2, atunci x & y este zero dar x && y este 1!!!

Operatori logici si de deplasare pe biti

- Operatorii de deplasare << si >> realizeaza deplasari la stanga si la dreapta pentru operandul lor din stanga, cu numarul de pozitii dat de operandul din dreapta lor;
- Exemple:

```
x << 2
```

deplaseaza la stanga pe x cu doua pozitii, umpland locurile libere cu zero; aceasta este echivalent cu inmultirea cu 4!

- Deplasand la dreapta o cantitate **unsigned**, bitii vacanti se umplu cu zero;
- Deplasand la dreapta o cantitate cu semn, bitii vacanti se umplu cu semnul ("deplasarea aritmetica") pe anumite calculatoare, si cu 0 ("deplasare logica") pe altele;

Operatori logici si de deplasare pe biti

- Operatorul unar ~ da complementul fata de 1 al unui intreg – adica el converteste fiecare bit de 1 in 0 si vicevesa;
- Exemplu:

```
x & ~077
```

seteaza ultimii 6 biti ai lui x pe 0!

- Exercitiu: scrieti o functie ***wordlength***() care calculeaza lungimea unui cuvant de pe calculatorul gazda, adica numarul de biti dintr-un ***int***. Functia trebuie sa fie portabila in sensul ca acelasi cod sursa sa lucreze pe orice calculator;

Operatori si expresii de asignare/atribuire

- Operatorul de atribuire este folosit pentru a atribui unei variabile valoarea unei constante sau a unei expresii.
- Exemplu:

```
int a, b, c;
a =10;
b = 9;
c = a –b;
```

- Operatorul de atribuire are prioritatea cea mai mică faţă de ceilalţi operatori.
- Limbajul C permite alcătuirea de instrucţiuni complexe de forma:
  x1 =x2=x3=constantă;
- Operatorul de atribuire care contribuie la alcătuirea unei expresii se asociază de la dreapta la stânga. Pentru evaluarea corectă a expresiei este necesar ca în prealabil valoarea expresiei din partea dreaptă a iniţializării să fie cunoscută.

Operatori si expresii de asignare/atribuire

- Expresii de tipul:

```
i = i + 2;
```

in care membrul stang este repetat in membrul drept pot fi scrise intr-o forma condensata:

```
i += 2;
```

folosind operatorul de asignare +=

- Generalizare: majoritatea operatorilor binari (operatori ca +, care au un operand stang si un operand drept) au un operator de asignare corespunzator "op=", unde op este unul din:

```
+    -    *    /    %    <<    >>    &    ^    |
```

Operatori si expresii de asignare

- Daca e1 si e2 sunt doua expresii, atunci:

```
e1 op= e2
```

- este echivalent cu

```
e1 = (e1) op (e2)
```

- Atentie :

```
x *= y + 1
```

inseamna de fapt

```
x = x * (y + 1)
```

si nu

```
x = x * y + 1
```

Conversii de tip

- Cand intr-o expresie apar operanzi de mai multe tipuri, ei se convertesc intr-un tip comun, dupa anumite reguli;
- Regula de baza este ca “*tipul de data mai mic se converteste in tipul de data mai mare*”;
- In primul rand, ***char*** si ***int*** pot fi amestecati in expresiile aritmetice: orice ***char*** este convertit automat intr-un ***int***.
- Exemplu de conversie intre ***char*** si ***int*** il constituie functia lower care transforma literele mari din setul de caractere ASCII in litere mici;

Conversii de tip

```
char lower(int c)  /* conversie ASCII litere mari in
litere mici */
{
if (c >= 'A' && c <= 'Z')
     return(c + 'a' - 'A');
else
     return(c)
}
```

Conversii de tip

- Atentie: limbajul nu specifica daca o variabila de tip ***char*** este o cantitate cu semn sau fara semn;
- Cand un ***char*** este convertit intr-un ***int***, poate el produce un intreg negativ?
- Raspunsul variaza de la calculator la calculator, reflectand diferentele arhitecturale;
- Pe anumite calculatoare un ***char*** al carui cel mai din stanga bit este 1 va fi convertit intr-un intreg negativ ("extensie de semn”);
- Pe altele, un ***char*** este convertit intr-un ***int*** prin adaugarea de zerouri in partea stanga si astfel el este intotdeauna pozitiv;

Conversii de tip

- Cea mai comuna aparitie a acestei situatii este cand pentru EOF se foloseste -1. Sa consideram codul:

```
char c;
     c = getchar();
     if (c == EOF)
          ...
```

- Pe un calculator care nu face extensie de semn, c este intodeauna pozitiv deoarece el este un ***char***, dar totusi EOF este negativ.
- In consecinta testul esueaza intodeauna. Pentru a evita aceasta, trebuie sa avem grija atunci cand folosim ***int*** in loc de ***char*** pentru orice variabila care primeste o valoare returnata de getchar.

Conversii de tip

- Conversie de tip automata: expresiile relationale de tipul i > j si expresiile logice conectate prin && si || se definesc a avea valoarea 1 pentru adevar si 0 pentru fals. Astfel, o asignare:

```
isdigit = c >= '0' && c <= '9';
```

pune pe isdigit pe 1 daca c este o cifra si pe 0 daca nu (In partea de test a lui if, while, for "adevarat" inseamna "nonzero").

- Conversiile aritmetice implicite: intr-o operatie binara, tipul "inferior" este promovat la tipul "superior" inainte de executia operatiei;

Conversii de tip

- ***char*** si ***short*** se convertesc in ***int*** iar ***float*** este convertit in ***double***;
- Daca un operand este ***double***, celalalt este convertit in ***double*** iar rezultatul este ***double***;
- Altfel, daca un operand este ***long***, celalalt este convertit in ***long*** iar rezultatul este ***long***;
- Altfel, daca un operand este ***unsigned***, celalalt este convertit in ***unsigned***, iar rezultatul este ***unsigned***;
- Altfel, operanzii trebuie sa fie ***int***, iar rezultatul este ***int***;

Conversii de tip

- Conversiile se fac si in asignari: valoarea partii drepte este convertita la tipul din stanga, care este tipul rezultatului;
- Un caracter este convertit intr-un ***int*** fie cu extensie de semn, fie nu. Operatia inversa, ***int*** in ***char***: bitii de ordin superior in exces sunt eliminati. Astfel, in:

```
int i;
char c;
i = c;
c = i;
```

valoarea lui c este neschimbata!

Conversii de tip

- Daca x este ***float*** iar i este ***int***, atunci:

```
x = i;
```

- si

```
i = x;
```

provoaca amandoua conversii;

- ***float*** in ***int*** provoaca trunchierea oricarei parti fractionare;
- ***double*** este convertit in ***float*** prin rotunjire;
- Intregii lungi sunt convertiti in scurti sau in ***char*** prin pierderea bitilor de ordin superior in exces;

Conversii de tip

- Conversia explicita de tip poate fi fortata in orice expresie cu o constructie numita ***cast***:

```
(nume de tip) expresie
```

- Semnificatia precisa a unui ***cast*** este de fapt ca si daca o expresie ar fi asignata la o variabila de tipul specificat, care este apoi folosita in locul intregii constructii;

```
sqrt((double) n)
```

Expresii conditionale

```
if (a < b)
    m = a;
else
    m = b;
```

- Calculeaza maximul dintre a si b si il memoreaza in m;
- Acelasi lucru se poate realiza astfel:

```
z = (a > b) ? a : b;
```

Expresii conditionale

- In expresia:

```
e1 ? e2 : e3
```

- expresia e1 se evalueaza prima. Daca ea este nonzero (adevarata) atunci se evalueaza expresia e2 si aceasta este valoarea expresiei conditionale. Altfel, se evalueaza e3 si aceasta este valoarea expresiei.
- Daca e2 si e3 sunt expresii de tipuri diferite, tipul rezultatului se determina dupa regulile de conversie;
- Daca f este un float si n este un int, atunci expresia

```
(n > 0) ? f : n
```

este de tipul double, indiferent daca n este pozitiv sau nu;

Expresii conditionale

- Ce realizeaza urmatoarea secventa de cod?

```
for (i = 0; i < N; i++)
   printf("%6d%c", a[i], (i % 10 == 9 || i == N-1) ? '\n' : ' ');
```

Precedenta operatorilor

operator	ordine de evaluare
() [] -> .	de la stanga la dreapta
! ~ ++ -- - (tip) * & sizeof	de la dreapta la stanga
* / %	de la stanga la dreapta
+ -	de la stanga la dreapta
<< >>	de la stanga la dreapta
< <= > >=	de la stanga la dreapta
== !=	de la stanga la dreapta
&	de la stanga la dreapta
^	de la stanga la dreapta
|	de la stanga la dreapta
&&	de la stanga la dreapta
||	de la stanga la dreapta
? :	de la dreapta la stanga
= += -= etc	de la dreapta la stanga
,	de la stanga la dreapta