Programarea calculatoarelor și limbaje de programare 1

Laborator 2
Operatori, expresii

Dr. Ing. Liviu-Daniel ȘTEFAN
liviu_daniel.stefan@upb.ro

Dr. Ing. Mihai DOGARIU
mihai.dogariu@upb.ro

Prof. Dr. Ing. Bogdan IONESCU
bogdan.ionescu@upb.ro

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Universitatea POLITEHNICA din Bucureşti

16-20 Octombrie, 2023

Cuprins

Operatori aritmetici disponibili în C

Semantică

Operatorii sunt grupați în patru clase de operatori: *aritmetici*, *relaționali*, *logici*, și *de acțiune pe biți*. Seplimentar, limbajul C definește operatori specifici anumitor sarcini.

Tabel: Operatorii aritmetici disponibili în C

Operator	Acțiune
-	Scădere, de asemenea și minus unar
+	Adunare
*	Înmulțire
/	Împărțire
%	Modul
- -	Decrementare cu 1 a valorii operandului
++	Incrementare cu 1 a valorii operandului

Operatori aritmetici disponibili în C

- Operatorii aritmetici pot fi aplicați oricăror tipuri de date permise în limbajul C/C++.
  - Atunci când aplicați operatorul / unui întreg sau caracter, rezultatul va fi tot un întreg;
  - Operatorul % returnează restul unei împărțiri de întregi. Nu se poate folosi pentru valori reale;
  - Atât operatorul de incrementare (++) cât și cel de decrementare (- -) pot să fie plasați înainte (prefix) sau după (postfix) variabila operată. Atunci când operatorul ++ / - - precede variabila, operația se realizează înaintea utilizării valorii în expresii. Atunci când operatorul ++ / - - urmează variabilei, valoarea va fi utilizată în expresie și apoi se va efectua operația.

Operatori aritmetici disponibili în C

Următorul exemplu declară două variabile de tip întreg `var1` și `var2`. Inițializează variabila `var1` cu valoarea 2, și variabila `var2` cu valoarea 5. Apoi exemplifică utilizarea operatorilor minus unar (linia 8), împărțire (linia 9), modulo (linia 10), imcrementare prefix (linia 11) și respectiv, incrementare postfix (linia 12).

```
1  #include<stdio.h>
2
3  int main(){
4
5      int var1, var2;
6      var1 = 2;
7      var2 = 5;
8
9      printf("%d\n", -var1); // aplică operatorul minus unar și afișează -2
10     printf("%d\n", var2 / var1); // afișează 2
11     printf("%d\n", var2 % var1); //afișează 1
12     printf("%d\n", ++var1); // incrementează valoarea cu 1 și apoi afișează 3
13     printf("%d\n", var2++); //afișează 5 apoi incrementeaza valoarea cu 1
14     return 0;
15 }
```

Operatori relaționali disponibili în C

Reprezentarea valorilor adevărat și fals

Limbajul C/C++ interpretează orice valoare diferită de 0 ca adevărat, iar valoarea 0 ca fals.

- Operatorii relaționali se referă la relațiile pe care un operand poate să le aibă cu un alt operand. Operatorii relaționali funcționează pe conceptul de adevărat și fals.

Tabel: Operatori relaționali disponibili în C

Operator	Acțiune
>	Mai mare decât
>=	Mai mare sau egal
<	Mai mic decât
<=	Mai mic sau egal
==	Egal
!=	Diferit

Operatori logici disponibili în C

- Operatorii logici se referă la modul cum sunt corelate relațiile pe care un operand poate să le aibă cu un alt operand. Operatorii logici funcționează pe conceptul de adevărat și fals.

Tabel: Operatori logici

Operator	Acțiune
&&	AND (ȘI)
||	OR (SAU)
!	NOT (NEGAT)

- ☞ Rețineți că toate expresiile relaționale și logice au ca rezultat 0 sau 1;
- ☞ Puteți combina mai mulți operatori relaționali și/sau logici într-o expresie;

Operatori logici disponibili în C

Tabel: Tabela de adevărat și fals pentru operatorii logici

p	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

- Pentru a fi adevărată condiția realizată cu operatorul ȘI logic, trebuie ca ambele condiții să fie evaluate ca adevarate. Dacă vreuna dintre ele este falsă, condiția rezultată este evaluată ca falsă;
- Pentru a fi adevărată condiția realizată cu operatorul SAU logic, trebuie ca numai una dintre condiții să fie evaluată ca adevărată. Dacă vreauna dintre ele este adevărată (sau amândouă sunt adevărate), condiția rezultată este evaluată ca adevărată. Dacă ambele condiții sunt evaluate ca false, atunci condiția rezultată este falsă.

Exemplu utilizare operatori logici

Următorul program exemplifică utilizarea operatorilor relaționali și logici.

```
1  #include<stdio.h>
2
3  int main(){
4      int var1, var2;
5      var1 = 2;
6      var2 = 5;
7
8      printf("%d\n", var1 > var2); // afișează 0 - fals
9      printf("%d\n", var1 < var2); // afișează 1 - adevărat
10     printf("%d\n", var1 >= var2); // afișează 0 - fals
11     printf("%d\n", var1 <= var2); // afișează 1 - adevărat
12     printf("%d\n", var1 == var2); // afișează 0 - fals
13     printf("%d\n", var1 && var2); // afișează 1 - adevărat
14     printf("%d\n", var1 || var2); // afișează 1 - adevărat
15     printf("%d\n", !var1); // afișează 0 - fals
16     printf("%d\n", var1 && !var2 || var1 <= var2); // afișează 1 - adevărat
17     return 0;
18 }
```

Operatori de acțiune pe biți

- Operațiile asupra biților se referă la testare, inițializare sau deplasare a biților existenți într-un octet sau într-un identificator asociat tipurilor de date `char` și `int` și variațiilor valide.
  - ☞ Nu pot fi folosite asupra tipurilor `float`, `double`, `long`, `double`, `void` sau asupra altor tipuri mai complexe.

Tabel: Operatori pentru biți

Operator	Acțiune
&	AND (ȘI)
|	OR (SAU)
^	OR exclusiv (SAU exclusiv - XOR)
~	NOT (Complement față de 1)
>>	Deplasare la dreapta
<<	Deplasare la stânga

Operatori de acțiune pe biți

Sintaxă

```
variabila << sau >> numar_de_poziție_ale_bitilor
```

- Operatorii de deplasare pentru biți >> și <<, deplasează toți biții dintr-o variabilă la dreapta sau la stânga, după cum se specifică. Deoarece biții sunt mutați către un capăt, la celălalt capăt se adaugă zerouri. În cazul unui întreg negativ, o deplasare la dreapta va determina introducerea unui 1, astfel încât bitul de semn se va păstra;
- Operatorii de deplasare pentru biți pot, de asemenea, să înmulțească și să împartă întregi. O deplasre la dreapta împarte efectiv un număr cu 2, iar o deplasare la stânga îl înmulțește cu 2;
- Operatorii pentru biți determină o valoare arbitrară în concordanță cu operația respectivă, diferit de operatorii relaționali sau logici care determină întotdeauna un rezultat de 0 sau 1;

Operatori de acțiune pe biți

- Operatorii pentru biți `AND`, `OR` și `NOT` au aceleași tabele de adevăr ca și echivalentele lor logice, doar că lucrează bit cu bit. `XOR` este adevărat dacă numai unul dintre termeni este adevărat, altfel, el este fals.

Tabel: Exemplu de folosire a operatilor de execuție pe biți

unsigned char x	**x după executarea fiecărei instrucțiuni**	**valoarea lui x**
x = 7;	0 0 0 0 0 1 1 1	7
x = x <<1;	0 0 0 0 1 1 1 0	14
x = x <<3;	0 1 1 1 0 0 0 0	112
x = x <<2;	1 1 0 0 0 0 0 0	192
x = x >>1;	0 1 1 0 0 0 0 0	96
x = x >>2;	0 0 0 1 1 0 0 0	24

Alți operatori

- operatorul de atribuire (=). Poate fi folosit în cadrul oricărei expresii valide;
  - ☞ Operatorul de atribuire definește doi termeni: lvalue - orice obiect care poate să apară în partea stângă a instrucțiunii de atribuire, și rvalue - se referă la expresiile din membrul drept al atribuirii.
- operatorul ternar de condiționare (`?:`). Înlocuiește instrucțiuni de forma dacă-atunci-altfel;
  - ☞ Operatorul `?:` are forma `exp1 ? exp2 : ex3;` și lucrează astfel: se evaluează `exp1`. Dacă rezultatul evaluării este adevărat (1) se evaluează `exp2`, și valoarea ei devine valoarea returnată de operator. Dacă rezultatul evaluării expresiei1 este fals (0), se evaluează `exp3`, și valoarea sa devine valoarea returnată de operator.

Alți operatori

- operatorii `&` și `*`. Operatorul de referențiere pentru pointeri `&` este un operator unar care returnează adresa din memorie a unui element. Operatorul de dereferențiere pentru poiteri `*` este un operator unar și returnează valoarea din variabila localizată la adresa specificată;
- operatorul `sizeof`. Operatorul unar `sizeof` returnează lungimea în octeți a variabilei sau a specificatorului de tip dintre paranteze care îi urmează;
  - ☞ Parantezele sunt opționale pentru numele de variabile.
- operatorul cast. Operator unar care forțează o expresie să devină de un anumit tip de date, având forma `(tip) expresie`;

Alți operatori

- operatorii punct `(.)`, și săgeată (`->`). Acești operatori se referă la elemente individuale ale structurilor și uniunilor. Structurile și uniunile sunt tipuri de date agregate. Operatorul punct este utilizat atunci când se face referire la structuri sau uniuni efective. Operatorul săgeată este folosit atunci când se face referire la un pointer catre o structură sau o unuiune;
- operatorul virgulă (`,`). Operatorul virgulă stabilește o succesiune de operații, iar expresia din dreapta stabilește valoarea întregii expresii separate prin virgulă. Exemplu: `x = (y = 1, y + 1);` atribuie variabilei `y` valoarea 1, adună cu 1 variabila `y`, și în final, atribuie lui `x` valoarea variabilei `y`, 2;
- operatorii `()`, `[]`. Parantezele rotunde sunt operatori care măresc prioritatea operațiilor din interiorul lor. Parantezele pătrare se folosesc pentru a specifica indicii matricelor.

Conversii automate în expresii

Semantică

O expresie este o combinație validă formată din operatori, constante și variabile.

Semantică

Conversia automată în expresii, numită și promovarea tipului este procedura prin care compilatorul face conversia tuturor elementelor asupra cărora se operează în tipul celui mai mare.

- ☞ Atunci când intr-o expresie sunt amestecate constante și variabile de diferite tipuri, ele sunt convertite în același tip. Mai intâi, toate valorile de tip char și short int sunt automat evaluate ca int (promovare la întreg). O dată încheiat acest proces, toate celelalte conversii sunt efectuate operație cu operație.

Conversii automate în expresii

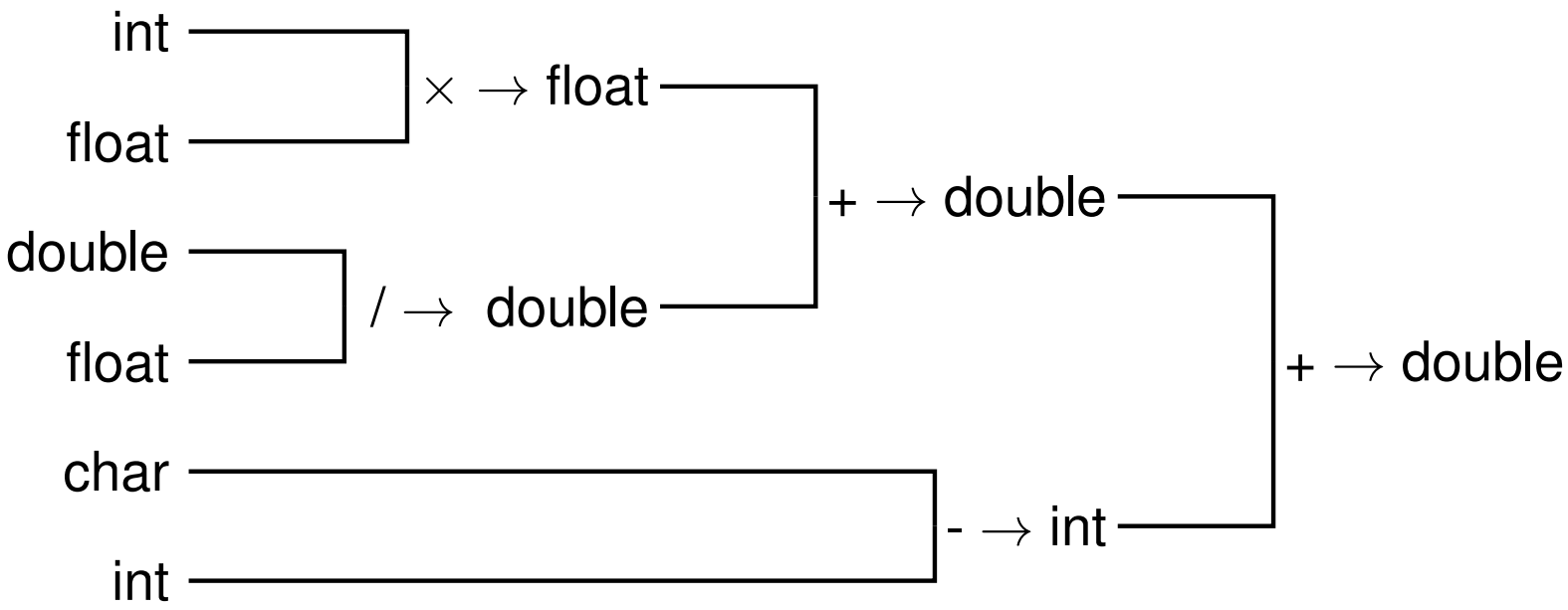

Figura: Exemplu de conversie automată a tipurilor pentru expresia `(int * float) + (double / float) - (char + int)`

Expresii compuse

Semantică

Expresiile compuse sunt variațiuni ale instrucțiunilor de atribuire, uneori numite prescurtări în C. Acestea constau dintr-un operator binar și operatorul de atribuire.

- Scopul lor este să efectueze operația operatorului binar pe ambii operanzi și stochează rezultatul acelei operații în operandul din stânga.

Tabel: Expresii compuse disponibile în C

Operator	Lvalue	Rvalue
+= sau -=	Aritmetic	Aritmetic
+= sau -=	Pointer	Întreg
*=, /= și %=	Aritmetic	Aritmetic
<<=, >>=, &=, ^=, și |=	Întreg	Întreg

Precedența operatorilor în C

Tabel: Precedența și asociativitatea operatorilor C (de la cea mai mare (Clasa 1) la cea mai mică (Clasa 15) precedență).

Descriere operator și clasa de precedență	Operator
Clasa 1, asociativitate de la stânga la dreapta	
Selecție membru (obiect sau pointer)	. sau ->
Indice matrice	[]
Apel funcție	()
Incrementare postfix	++
Decrementare postfix	- -
Clasa 2, asociativitate de la dreapta la stânga	
Dimensiune obiect sau tip	sizeof
Incrementare prefix	++
Decrementare prefix	- -
Complement față de 1	$\sim$
NOT logic	!
Minus unar	-
Plus unar	+
Operatorul de referențiere	&
Operatorul de dereferențiere	*
Cast	(tip)

Precedența operatorilor în C

Descriere operator și clasa de precedență	Operator
Clasa 3, asociativitate de la stânga la dreapta	
Pointer la membru	.* sau ->*
Clasa 4, asociativitate de la stânga la dreapta	
Înmulțire	*
Împărțire	/
Modulo	%
Clasa 5, asociativitate de la stânga la dreapta	
Adunare	+
Scădere	-
Clasa 6, asociativitate de la stânga la dreapta	
Deplasare pe biți la dreapta	>>
Deplasare pe biți la stânga	<<
Clasa 7, asociativitate de la stânga la dreapta	
Mai mic decât	<
Mai mare decât	>
Mai mic sau egal	<=
Mai mare sau egal	>=
Clasa 8, asociativitate de la stânga la dreapta	
Egalitate	==
Inegalitate	!=

Precedența operatorilor în C

Descriere operator și clasa de precedență	Operator
Clasa 9, asociativitate de la stânga la dreapta	
ȘI pe biți	&
Clasa 10, asociativitate de la stânga la dreapta	
SAU exclusiv pe biți (XOR)	^
Clasa 11, asociativitate de la stânga la dreapta	
SAU pe biți	|
Clasa 12, asociativitate de la stânga la dreapta	
ȘI logic	&&
Clasa 13, asociativitate de la stânga la dreapta	
SAU logic	||
Clasa 14, asociativitate de la dreapta la stânga	
Operatorul ternar condițional	?:
Atribuire	=
Atribuire compusă prin înmulțire	*=
Atribuire compusă prin împărțire	/=
Atribuire compusă prin modulo	%=
Atribuire compusă prin adunare	+=
Atribuire compusă prin scădere	-=
Atribuire compusă prin deplasare pe biți la stânga	<<=
Atribuire compusă prin deplasare pe biți la dreapta	>>=

Precedența operatorilor în C

Atribuire compusă prin ȘI pe biți	&=
Descriere operator și clasa de precedență	Operator
Atribuire compusă prin SAU pe biți	|=
Atribuire compusă prin SAU exclusiv pe biți (XOR)	^=
Clasa 15, asociativitate de la stânga la dreapta	
Virgula	,

- Prioritatea operatorului specifică ordinea operațiilor în expresiile care conțin mai mult de un operator. Asociativitatea operatorului specifică dacă, într-o expresie care conține mai mulți operatori cu aceeași precedență, un operand este grupat cu cel din stânga sau cu cel din dreapta lui.
- Operatorii sunt listați în ordinea descrescătoare a priorității. Dacă mai mulți operatori apar într-un grup, aceștia au prioritate egală;
- Ordinea operațiunilor nu este definită de standardele ANSI. Compilatorul este liber să evalueze astfel de expresii în orice ordine, dacă compilatorul poate garanta un rezultat consecvent.

Exemple expresii în C

```c
#include<stdio.h>

int main(){
    int a = 3, b = 6, c = 10, d = 0xAAAA, e = 0x5555;
    a += b;
    b %= a;
    c >>= 1;
    d |= e;
    printf("a += b este %d\n", a); //afișează 9
    printf("b %= a este %d\n", b); // afișează 6
    printf("c >>= 1 este %d\n", c); // afișează 5
    printf("d |= e este 0x%X\n", d); //afisează 0xFFFF

    // afișează 4 pentru ANSIC17, x64.
    printf("Dim. în octeți a lui a este: %d\n", sizeof(a));
    // afișează 1
    printf("a\? 0 : 1 returnează %d\n", a? 1 : 0);
    //afișează 9.000000
    printf("%f\n", float(a));
    // afișează 1 (aceeași precedență, asociativitate de la stânga la dreapta)
    printf("Expresia a == 2 != b returnează %d\n", a == 2 != b);
    return 0;
}
```