2.generatii de limbaje de programare

Familia limbajelor de programare se poate clasifica în:  
limbaje de nivel coborât, limbaje de nivel înalt şi limbaje de  
nivel mediu.  
Prima categorie cuprinde limbajul cod-maşină şi limbajul  
de asamblare. Ambele sunt specifice tipului de maşinăde  
calcul pe care sunt implementate. Limbajul cod maşină este  
limbajul alcătuit din acele instrucţiuni elementare care sunt  
înţelese şi executate de un anumit tip de calculator. Limbajul  
de asamblare foloseşte în locul codurilor numerice reprezentări  
simbolice, numite şi mnemonice, care uşurează munca de  
programare. Operaţiile limbajului de asamblare sunt operaţii de  
bază ale calculatorului. El nu acceptă structuri de control şi  
4 Programarea în limbajul C  
date structurate, dar permite adresarea simbolică a locaţiilor de  
memorie. Din aceste motive programele în limbaj de asamblare  
sunt lungi şi se scriu anevoios, dar sunt performante din punct  
de vedere al vitezei de execuţie şi al posibilităţilor de acces la  
resursele hardware.  
A doua categorie, cea a limbajelor de nivel înalt, include  
nume binecunoscute: Fortran, Cobol, Basic, Pascal etc.. O  
parte din trăsăturile lor comune se referă la posibilitatea de a  
folosi structuri de control, date structurate, de a elabora cu  
uşurinţă programe portabile (care se pot adapta uşor la  
implementarea pe diverse categorii de sisteme de calcul).  
Limbajele din această categorie pierd însă calitatea esenţială a  
limbajelor de nivel coborât, aceea de a exploata eficient  
resursele maşinii de calcul pe care sunt implementate.  
Categoria limbajelor de nivel mediu îmbină trăsăturile  
principale ale limbajelor de nivel înalt cu cele ale limbajelor de  
nivel coborât. Limbajul C este un limbaj de nivel mediu

6. Tipuri de date fundamentale in c si operatii elementare.structuri de date

În C, ca şi în alte limbaje, datele sunt clasificate în tipuri  
de date. Există tipuri de date fundamentale (numite şi  
predefinite, simple sau de bază) şi tipuri de date derivate.  
Tipurile derivate (tablouri, pointeri, structuri, uniuni, enumerări  
şi orice tip definit de programator) se bazează pe tipurile  
fundamentale.

7.tipuri de date fundamentale in C

Tipurile de date fundamentale din C se împart în cinci  
categorii: char, int, float, double şi void. Primele patru tipuri se  
mai numesc şi tipuri aritmetice şi se referă respectiv la valori  
caracter, întregi, reale în simplă precizie şi reale în dublă  
precizie. Tipul de date void indică absenţa oricărei valori şi  
este utilizat, de exemplu, la descrierea funcţiilor care nu  
returnează nici o valoare

În majoritatea implementărilor însă, tipul char ocupă un octet,  
int ocupă doi octeţi, iar float patru octeţi. Domeniul de valori  
poate fi modificat utilizând modificatorii de tip. Aceştia sunt:  
signed, unsigned, short şi long. Modificatorii signed,  
unsigned, short şi long se pot aplica tipului int, signed şi  
unsigned, tipului char, iar long, tipului double. Efectul aplicării  
modificatorilor signed sau unsigned asupra tipurilor de date  
întregi constă în interpretarea diferită, din punct de vedere al  
semnului, a informaţiei memorate.

Observa***ţ***ii:  
· Pentru tipul întreg de date (char, int, short, long)  
reprezentarea implicită este signed.  
· Specificarea unui modificator fără tip înseamnă  
considerarea implicită a tipului int.  
· În C nu există tipul de date boolean. Din acest motiv  
func*ţ*ionează următoarea convenţie: orice expresie diferită de  
zero are valoarea adevărat, iar dacă e egală cu zero, valoarea  
fals.

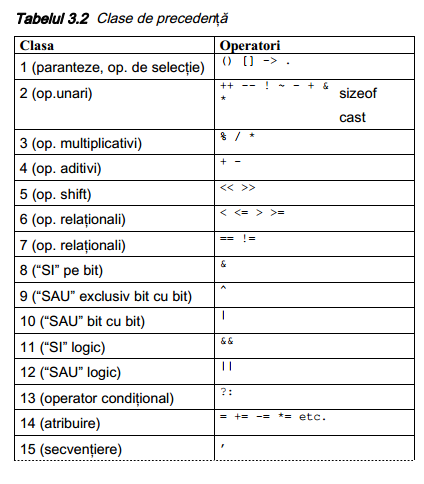
În Capitolul 2 (vezi paragraful Vocabularul limbajului),  
am prezentat o clasificare a constantelor în: întregi, reale,  
caracter, şir. Se pune problema, cărui tip de date îi aparţine o  
constantă numerică? Când constanta este caracter, răspunsul  
este simplu: tipului char. De asemenea, constanta în virgulă  
mobilă (în notaţie uzuală cu punct sau în notaţie ştiinţifică) va  
aparţine tipului double. Pentru celelalte constante numerice  
compilatorul va considera implicit încadrarea în cel mai mic tip  
de date compatibil. De exemplu, 23 este de tip int, 65000 de tipunsigned, 2000002 de tip long int. Încadrarea într-un tip de  
date se poate face şi explicit adăugând constantei unul din  
sufixurile L sau U, dacă e întreagă sau F sau L, dacă e reală.  
Constanta întreagă cu sufixul L este de tip long, iar cu sufixul  
U, de tip unsigned. Constanta reală cu sufixul F are tipul float,  
iar dacă e urmată de sufixul L are tipul long double

8. clasificarea operatorilor in C

Operatorii sunt elemente de bază ale limbajului care  
arată ce operaţii trebuie executate asupra unor operanzi. În C,  
operanzi pot fi constantele, numele de variabile, numele de  
funcţii, expresiile.  
Bogata familie de operatori conferă limbajului C o  
trăsătură aparte.  
Clasificarea operatorilor C se poate face după mai multe  
criterii:  
· după numărul de operanzi prelucraţi (unari, binari,  
ternari);  
· după prioritatea avută în evaluarea expresiilor (clase  
de preceden**ţ**ă);

· după tipul operanzilor (aritmetici, rela**ţ**ionali, logici **ş**i  
la nivel de bit)

9. operatori de atribuire incrementare decrementare

Priorităţile operatorilor impun ordinea de evaluare a  
expresiilor. Ca şi în calculele algebrice obişnuite ordinea de  
evaluare poate fi modificată cu ajutorul parantezelor rotunde.  
Operatorii care au priorităţi egale, aparţin aceleiaşi clase  
de preceden**ţ**ă. Lista operatorilor grupaţi după clase de  
precedenţă este dată în Tabelul 3.2.

Operatorul de atribuire = ocupă un loc aparte în familia  
operatorilor. Cu ajutorul lui putem să atribuim unei variabile o  
anumită valoare. Forma sa generală este:  
v=e;  
unde v este un nume de variabilă, iar e este o expresie. În C,  
membrul stâng şi membrul drept al unei atribuiri se mai  
numesc valoare stângă (lvalue), respectiv valoare dreaptă  
(rvalue). Spre deosebire de alte limbaje (Fortran, Pascal etc.)  
în C, operatorul de atribuire poate apare şi în interiorul unei  
expresii, fapt ce permite o compactare a codului sursă. De  
exemplu, două atribuiri succesive de genul:  
A=pi\*r\*r;  
V=A\*h;  
pot fi scrise compact sub forma:  
V=(A=pi\*r\*r)\*h;

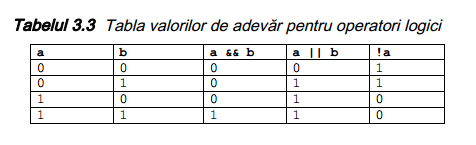
Dacă v e un operand oarecare cazurile uzuale de  
atribuire v=v+1 şi v=v-1 se pot scrie simplificat sub forma v++  
şi respectiv v--. Operatorul ++ se numeşte operator de  
incrementare, iar -- operator de decrementare. Funcţie de  
poziţia lor faţă de operand ei pot fi operatori prefix sau postfix.  
De exemplu:  
x=x+1 se poate scrie x++ sau ++x,  
x=x-1 se poate scrie x-- sau --x.  
Dacă operandul nu apare în cadrul unei expresii, după  
cum se vede din exemplul de mai sus, nu are importanţă dacă  
operatorii sunt prefix sau postfix. Când operandul apare în  
cadrul unei expresii, dacă este precedat de operatorul ++ sau -  
- se execută întâi incrementarea sau decrementarea  
operandului şi apoi este folosit în expresie; dacă este însă urmat de operatorul ++ sau -- incrementarea sau  
decrementarea se va face după folosirea sa în expresie.  
De exemplu,  
expresia y=++x; e echivalentă cu secvenţa x=x+1;  
y=x;  
iar  
expresia y=x++; e echivalentă cu secvenţa y=x;  
x=x+1;  
Să observăm că cele două secvenţe vor produce aceeaşi  
valoare pentru x şi valori diferite pentru y.  
Folosirea operatorilor de incrementare şi decrementare  
este recomandabilă nu doar din raţiuni de simplificare a scrierii  
programelor ci şi datorită faptului că majoritatea  
compilatoarelor C generează coduri obiect foarte rapide în  
astfel de cazuri.

Operatorii **?,[,],\*,&,sizeof,**•**,->** vor fi prezentaţi mai pe  
larg în capitolele unde prezenţa lor este necesară: operatorul  
?: la instruc**ţ**iuni condi**ţ**ionale, parantezele pătrate [ şi ] la  
tipul de date tablou, \*,& şi sizeof la pointeri, • şi -> la tipurile  
de date struct şi union

10. operatori aritmetici ,relationali si logici

Operatorii rela**ţ**ionali din limbajul C sunt: <, >, <=, >=,  
==, !=. Semnificaţia lor rezultă din Tabelul 2.5. În urma  
evaluării unei expresii în care intervin operatori relaţionali  
rezultă valoarea 0 pentru fals şi 1 pentru adevarat. Programul  
de mai jos afişează valorile anunţate în comentariile alăturate:  
Operatorii logici din C sunt:  
! NU logic  
&& SI logic  
|| SAU logic

Modul de acţiune al acestor operatori este prezentat în Tabelul  
3.3.



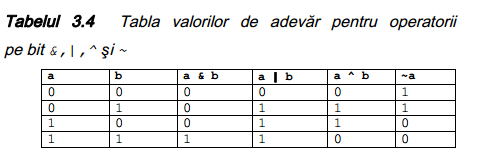
Observa***ţ***ii:  
Din Tabelul 3.2 rezultă că operatorii logici && *ş*i || au o  
prioritate mai mică decât operatorii relaţionali. Din acest motiv:  
expresia a<=b||c>d e echivalentă cu (a<=b)||(c>d)  
expresia a>b&&c<=d e echivalentă cu (a>b)&&(c<=d)  
Concluzia este că în astfel de situaţii prezen*ţ*a  
parantezelor este op*ţ*ională. Există însă cazuri în care  
parantezele sunt absolut necesare; de exemplu, dacă dorim să  
negăm expresia a>3 vom scrie !(a>3) *ş*i nu !a>3. Expresia  
!(a>3) va returna corect valoarea 0 sau 1 func*ţ*ie de mărimea  
lui a, în timp ce expresia !a>3 are totdeauna valoarea 0  
(datorită priorită*ţ*ii mai mari a operatorului ! fa*ţ*ă de > se  
evaluează mai întâi !a care poate fi 0 sau 1; oricare ar fi  
valoarea lui a, rezultă în final valoarea 0).

Dacă într-o expresie formată din operanzi lega*ţ*i prin  
operatorul || , iar valoarea primului operand este 1, valoarea  
expresiei este 1 (vezi Tabelul *3.2* ) *ş*i ceilal*ţ*i operanzi nu se mai  
evaluează. Dacă într-o expresie formată din operanzi lega*ţ*i  
prin operatorul &&, primul operand ia valoarea 0, valoarea  
expresiei este 0 iar ceilal*ţ*i operanzi nu se mai evaluează.  
Deoarece prezen*ţ*a parantezelor nu reduce viteza de  
execu*ţ*ie a expresiilor, ele pot fi folosite, alături de spa*ţ*ii, la  
cre*ş*terea gradului de lizibilitate a unui program. Iată două  
expresii echivalente:  
a=c<d||d>=c&&a<3;  
a = (c<d) || (d>=c) && (a<3);  
Este evident că cea de a doua expresie este mai u*ş*or de citit.

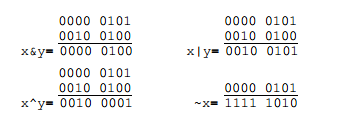
11.operatori la nivel de bit

Operatorii la nivel de bit din C permit programatorului să  
manevreze biţii aparţinând unor valori de tip char sau int (ei nu  
se aplică tipurilor de date float, double sau long double).  
Aceasta este una din acele facilităţi care apropie limbajul C de  
nivelul limbajelor de asamblare.

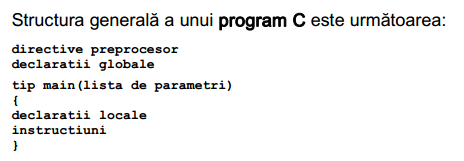
Operatorii la nivel de bit sunt:  
& SI pe bit  
| SAU pe bit  
^ SAU exclusiv pe bit  
~ NU pe bit (complement faţă de 1)  
>> deplasare dreapta (shift dreapta)  
<< deplasare stânga (shift stânga)  
Modul de acţiune al primilor 4 operatori pe bit e rezumat  
în Tabelul 3.4.

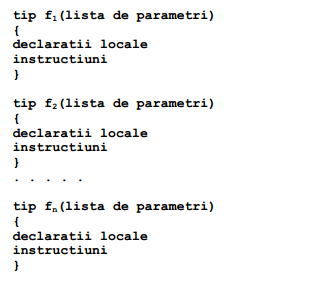


Se observă că a^b ia valoarea 1 numai dacă biţii a şi b  
sunt diferiţi.  
Asemănarea dintre Tabelul 3.3 şi Tabelul 3.4 nu trebuie  
să ne inducă în eroare. Operatorii logici tratează valorile 0 şi 1  
ale operanzilor ca pe valori logice, iar operatorii pe bit tratează  
operanzii ca succesiuni de biţi.  
Să considerăm două valori x=5 şi y=36 şi reprezentările  
lor în baza 2, adică x=0000 0101 şi y=0010 0100. Avem:

  
Observaţii:  
· Operatorul *&* reprezintă o modalitate de a elimina un bit (de  
a-l face egal cu 0) sau de a reţine biţii care ne interesează  
(operaţie de mascare).  
· În operaţiile de transmitere a datelor, bitul superior (cel mai  
din stânga) al unui octet este considerat de cele mai multe ori  
bit de paritate. Printr-o operaţie de setare la 1 a acestui bit sau  
de anulare a sa, se poate obţine un număr par sau impar de  
biţi în octetul respectiv funcţie de convenţia de paritate  
acceptată (paritatea poate fi pară sau impară). Utilizarea  
parităţii, permite verificarea corectitudinii octetului transmis

12.structura generala a unui program in C

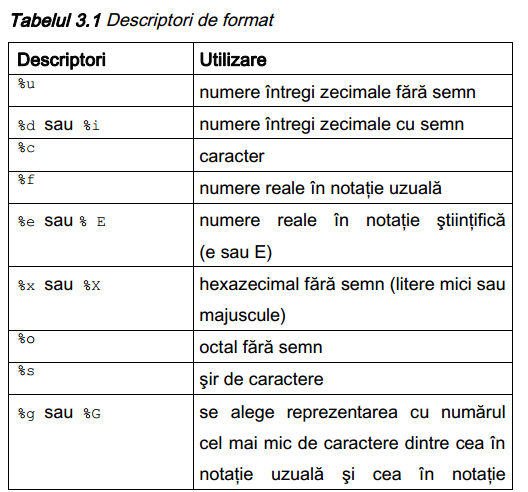


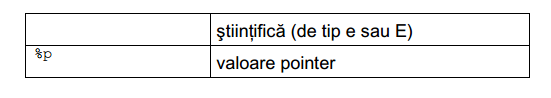


21. functii specific de intrare/ iesire in C. descriptori de format

Prezentăm în continuare funcţiile folosite frecvent pentru  
transferul de date de la tastatură în memoria calculatorului  
(func**ţ**ii de intrare) şi din memoria calculatorului pe ecran  
(func**ţ**ii de ie**ş**ire); cu această ocazie introducem şi descriptorii  
de format cei mai folosiţi. Deosebim trei categorii de funcţii de  
intrare/ieşire pentru consolă:  
· funcţii generale de intrare/ieşire (scanf() şi printf());  
· funcţii speciale de intrare/ieşire;  
· funcţii pentru citirea şi scrierea caracterelor;  
· funcţii pentru citirea şi scrierea şirurilor de caractere

FUNCŢIILE PRINTF() ŞI SCANF()  
Aceste funcţii reprezintă echivalentele pentru consolă a  
funcţiilor de intrare/ieşire pentru fişiere, fprintf() şi fscanf(),  
funcţii care vor fi prezentate în detaliu în Capitolul 10.  
Forma generală a funcţiei de afişare printf() este:  
int printf(sir\_format,lista\_de\_argumente);  
unde:  
· sir\_format poate conţine: mesaje pentru utilizator,  
secvente escape şi descriptori de format pentru valorile  
care se afişează;  
· lista\_de\_argumente reprezintă variabile sau expresii al  
căror conţinut se va afişa.  
Funcţia întoarce numărul de caractere scrise efectiv sau  
o valoare negativă în caz de insucces.  
Precizări:  
· Descriptorii de format servesc la efectuarea conversiilor  
dintre reprezentarea externă şi internă (la citire) *ş*i între  
reprezentarea internă şi externă (la scriere); formatul extern  
presupune succesiuni de caractere, iar cel intern succesiuni de  
cifre binare;  
· Atât la scriere cât *ş*i la citire, descriptorii de format sunt puşi  
în corespondenţă de la stânga spre dreapta cu elementele listei de argumente. Argumentul trebuie să fie compatibil cu  
tipul anunţat în descriptorul de format corespunzător;  
· Dacă numărul de argumente este mai mic decât numărul  
descriptorilor de format, datele de ieşire sunt nedefinite; dacă  
numărul argumentelor este mai mare decât numărul  
descriptorilor de format, argumentele în plus sunt ignorate.  
În Tabelul 3.1 prezentăm lista celor mai utilizaţi  
descriptori folosiţi de funcţia printf() şi semnificaţia lor.





Descriptorul %x are ca efect afişarea cifrelor hexazecimale  
A,B,C,D,E,F cu literă mică; dacă se foloseşte %X se afişează  
cu litere mari.  
Dacă se foloseşte %e litera ”e” din notaţia ştiinţifică apare  
ca e, iar dacă se foloseşte %E, apare ca majusculă (litera E).  
Valorile de tip long int se afişează utilizând  
%ld,%li,%lu,%lo sau %lx.  
Valorile de tip short int se afişează utilizând  
%hd,%hi,%hu,%ho sau %hx.  
Pentru a afişa valori double se va alege una din  
variantele: %lf,%le,%lE,%lg,%lG, iar pentru valori long double  
una din variantele: %Lf,%Le,%LE,%Lg,%LG

Precizări:  
· În marea lor majoritate descriptorii de format folosiţi la  
funcţia scanf() sunt identici cu cei de la funcţia printf(); practic,  
din tabelul prezentat anterior obţinem o listă validă pentru  
scanf(), îndepărtând %E,%X,%G. Folosi*ţ*i cu scanf(), descriptorii  
%f,%e,%g sunt echivalenţi.  
· Lista de argumente este citită de la stânga la dreapta *ş*i  
asociată în această ordine cu lista de descriptori. Fiecare  
descriptor arată funcţiei scanf() tipul valorii care se va citi:  
întreg, real, *ş*ir, pointer etc.. Să observăm că aceste valori sunt  
transferate variabilelor v1,v2,...vn prin intermediul adreselor  
&v1,&v2,...&vn.

22.functii special de intrare/iesire in C pentru caractere si siruri de caractere

Standardul ANSI C prevede două funcţii simetrice pentru  
transferul caracterelor la nivelul consolei: funcţiile getchar() şi  
putchar(). Ele îşi au prototipurile în ”stdio.h”.  
Funcţia pentru citire getchar() are forma generală  
int getchar(void);  
şi întoarce următorul caracter care va fi citit. Dacă s-a atins  
sfărşitul şirului sau se produce o eroare se întoarce EOF.  
Deşi nu aparţin standardului ANSI C, totuşi, funcţiile  
getch(), getche(), putch() sunt incluse frecvent în biblioteca  
standard a compilatoarelor compatibile DOS. Prototipurile  
acestor funcţii sunt în fişierul header ”conio.h”.

Spre deosebire de funcţia getchar() unde caracterul  
tastat este citit numai dacă se apasă în continuare tasta Enter,  
funcţiile getch() şi getche() preiau caracterul imediat după ce a  
fost tastat (fără a mai apăsa Enter). De asemenea, funcţia  
getch() preia caracterul de la tastatură fără a-l afişa pe ecran,  
în timp ce getche() afişează pe ecran caracterul citit (citire cu  
ecou).  
Funcţia pentru afişare putchar() are forma generală  
int putchar(int ch);  
unde ch este caracterul care se afişează.  
Funcţia întoarce în caz de succes caracterul scris, iar în  
caz contrar EOF.  
Funcţia putch() este echivalentă funcţional cu putchar() şi  
are forma generală  
int putch(int ch);  
unde ch este caracterul care se afişează.  
Ea este de asemenea o funcţie nestandard (nu e definită de  
standardul ANSI C) frecvent utilizată.

FUNCŢII SPECIALE PENTRU CITIREA/SCRIEREA  
ŞIRURILOR DE CARACTERE LA NIVELUL CONSOLEI  
Funcţii speciale pentru citirea/scrierea şirurilor de  
caractere la nivelul consolei sunt gets() şi puts(). Ambele  
funcţii îşi au prototipurile în fişierul header ”stdio.h”. Aceste  
prototipuri sunt prezentate în Capitolul 12 al lucrării. Deoarece  
în construcţia acestor prototipuri intervine noţiunea de pointer,  
prezentată în detaliu în Capitolul 6, ne limităm aici la a spune  
că prin apelul  
gets(sir\_destinatie);  
se citeşte un şir de la tastatură în sir\_destinatie, iar apelul  
puts(sir); are ca efect afişarea şirului sir pe ecran.  
Dăm ca exemplu secvenţa de program:  
. . . . . . .  
gets(x);  
. . . . . . .  
printf(”\n Sirul citit este =”);  
puts(x);  
. . . . . . .  
Un program complet care utilizează funcţiile gets() şi  
puts() este prezentat în Capitolul 5 (vezi paragraful Func**ţ**ii  
pentru prelucrarea **ş**irurilor de caractere)