

Doina Logofătu (n. 18 martie 1974) este absolventă a Facultății de Informatică, Universitatea „Al.I. Cuza” din Iași. În 1999 a absolvit cursurile de studii aprofundate (masterat), cu lucrarea de disertație *Programarea distribuită în Java RMI, o analiză comparativă*. A predat la Liceul de Informatică „Grigore C. Moisil” Iași, a susținut laboratoare de C/C++ la Facultatea de Informatică Iași, a lucrat ca programator la firme de soft din Iași și München. În prezent, lucrează ca programator în München. Este coautoare a manualului de informatică pentru clasa a XI-a, profilul matematică-informatică (Petriom, 2001), autoare a cărții C++. *Probleme rezolvate și algoritmi* (Polirom, 2001) și a unor articole publicate în revista *GInfo*.

Doina Logofătu

Bazele programării în C Aplicații

© 2006 by Editura POLIROM

www.polirom.ro

Editura POLIROM
Iași, B-dul Carol I nr. 4, P.O. BOX 266, 700506
București, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33, O.P. 37; P.O. BOX 1-728, 030174

Descrierea CIP a Bibliotecii Naționale a României :

LOGOFĂTU, DOINA

Bazele programării în C: aplicații / Doina Logofătu. – Iași : Polirom, 2006
Bibliogr.

ISBN : 973-46-0219-5

004.43 C (075.8)

Printed in ROMANIA

POLIROM
2006

Profesorilor mei de matematică, *Rodica Ungureanu*
(Liceul teoretic „Grigore Ghica”) și *Victor Barnea*
(Școala Generală nr. 7), cu mulțumiri și recunoștință

Cuprins

<i>Cuvânt înainte</i>	11
Evoluția sistemelor de calcul	13
1. Abacul	15
2. Noi instrumente de calcul	15
3. Blaise Pascal	16
4. Gottfried Wilhelm von Leibniz	17
5. Xavier Thomas de Colmar și primul calculator „pentru mase”	18
6. Mașina lui Joseph Jacquard	18
7. Charles Babbage și Lady Ada Lovelace	18
8. Părintele algebrei Boole	20
9. Inventarea mașinii de scris	21
10. Hollerich și Powers – noi concepte importante	21
11. Efectul Edison, electricitate, comutatoare și amplificatoare	21
12. Analizorul diferențial al lui Bush	22
13. Mașina Turing	22
14. Impactul algebrei Boole asupra circuitelor electronice	22
15. Primul computer electronic digital – ABC	23
16. George Stibitz, primul computer electronic digital, acces de la distanță	23
17. Konrad Zuse	24
18. ENIAC – succesul computerelor digitale	24
19. Ideile revoluționare ale lui John von Neumann	25
20. Succesorii lui ENIAC: EDVAC și UNIVAC; caracteristici	26
21. Circuitul integrat – cipul – cipul de siliciu	26
22. Generațiile sistemelor de calcul	27
23. Incursiune în viitor	27
24. Exerciții, întrebări	27

Algoritmi – elemente definitorii	29	
1. Geneza cuvântului <i>algoritm</i>	31	
2. Definiții alternative	31	
3. Exemple	32	
4. De la problemă spre soluție	37	
5. Proprietățile unui algoritm	39	
6. Algoritmica	39	
7. Modelul de calcul RAM	40	
8. Complexitatea algoritmilor	41	
8.1. <i>Notajile</i> Θ , O și Ω	41	
8.2. <i>Optimalitate, reducere, exemple</i>	42	
9. Exemplu comparativ de creștere a lui $O(g(n))$	45	
10. Timpul real necesar unui algoritm (polinomial vs exponential)	45	
11. Clasificarea problemelor (P, NP, NP-complete, NP-hard)	46	
12. Probleme NP-complete	47	
13. Problema satisfiabilității (<i>SAT</i>)	48	
14. Clasa problemelor NP-hard	48	
15. Exerciții și probleme propuse	49	
Limbajul C – prezentare generală	51	
1. Istorici al limbajului	53	
2. Construcții de bază	54	
2.1. <i>Setul de caractere</i>	54	
2.2. <i>Identificatori</i>	54	
2.3. <i>Cuvinte-cheie</i>	54	
2.4. <i>Tipuri de date</i>	55	
2.5. <i>Constanțe</i>	55	
2.6. <i>Variabile</i>	55	
2.7. <i>Comentariile</i>	56	
3. Expresii	57	
3.1. <i>*Operanzi</i>	57	
3.2. <i>Operatori</i>	57	
4. Structura programelor C	58	
4.1. <i>Directive de preprocessare</i>	58	
4.2. <i>Funcții</i>	59	
5. Instrucțiuni	60	
5.1. <i>Instrucțiunea vidă</i>	60	
5.2. <i>Instrucțiunea expresie</i>	60	
5.3. <i>Instrucțiunea compusă</i>	60	
5.4. <i>Instrucțiunea if</i>	61	
5.5. <i>Instrucțiunea while</i>	61	
5.6. <i>Instrucțiunea for</i>	62	
5.7. <i>Instrucțiunea do-while</i>	62	
5.8. <i>Instrucțiunea switch</i>	62	
5.9. <i>Instrucțiunea break</i>	63	
5.10. <i>Instrucțiunea continue</i>	63	
6. Intrări/iesiri standard	63	
6.1. <i>Funcția de scriere cu format</i>	63	
6.2. <i>Funcția de citire cu format</i>	64	
7. Fișiere	64	
7.1. <i>Deschiderea unui fișier</i>	65	
7.2. <i>Închiderea unui fișier</i>	65	
7.3. <i>Funcția de verificare a sfârșitului de fișier</i>	65	
7.4. <i>Funcții de citire/scriere</i>	65	
8. Noi tipuri de date: struct, enum, union	66	
8.1. <i>struct</i>	66	
8.2. <i>enum și typedef</i>	67	
8.2. <i>union</i>	68	
9. Tablouri	69	
9.1. <i>Concept, declarare, initializare</i>	69	
9.2. <i>Tablourile multidimensionale</i>	70	
9.3. <i>Tablourile de caracter</i>	71	
10. Pointeri, sizeof, alocarea dinamică a memoriei	71	
10.1. <i>Noțiune, exemple, metode specifice</i>	71	
10.2. <i>Pointeri folosiți ca argumente în funcții</i>	74	
10.3. <i>Pointeri și tablouri</i>	74	
10.4. <i>Pointeri și structuri</i>	75	
10.5. <i>Greșeli uzuale cu pointeri</i>	76	
11. Siruri de caracter	77	
11.1. <i>Reprezentarea și lucru cu siruri de caractere</i>	77	
11.2. <i>Operații cu siruri de caractere: biblioteca <string.h></i>	77	
12. Probleme rezolvate în C	79	
13. Întrebări, exerciții și probleme propuse	93	
Algoritmi elementari	95	
Problema 1. <i>Divizibilitate prin scăderi</i>	97	
Problema 2. <i>Algoritmul lui Euclid</i>	98	
Problema 3. <i>Test simplu de primalitate</i>	99	
Problema 4. <i>Distanță minimă între puncte</i>	100	
Problema 5. <i>Dimensiunea spațiului de memorie</i>	103	
Problema 6. <i>Numerele lui Fibonacci</i>	104	
Problema 7. <i>Pozitia punctului în cerc</i>	106	
Problema 8. <i>Media aritmetică</i>	107	
Problema 9. <i>Sir recurrent</i>	108	
Problema 10. <i>Funcție sinonimă cu atoi</i>	109	
Problema 11. <i>Informații despre caracter</i>	111	
Problema 12. <i>Palindrom și sumă cifrelor</i>	111	
Problema 13. <i>Radical infinit</i>	113	
Problema 14. <i>Serie cu valoarea π</i>	114	
Problema 15. <i>Instrucțiunea de decizie ?</i>	120	

Problema 16. Perechi speciale	121
Problema 17. Seria Farey	123
Problema 18. Divizori comuni	124
Problema 19. Transformare dintr-o bază în baza 10	125
Problema 20. Testarea formătării afișării numerelor naturale	126
Problema 21. Numere perfecte	127
Problema 22. Numere prietene	128
Problema 23. Suma cuburilor	130
Problema 24. Coduri ASCII	131
Exerciții și probleme propuse (nu sunt permise tablouri)	132
Tablouri, pointeri, pointeri la funcții	135
Problema 1. Element minim într-un tablou unidimensional	137
Problema 2. Bubble Sort	138
Problema 3. Derivata unui polinom	140
Problema 4. Ciurul lui Eratostene	142
Problema 5. Selectarea numerelor ce depășesc un prag	143
Problema 6. Înmulțirea unui polinom cu $(X-a)$	144
Problema 7. Produs a două matrice	145
Problema 8. Conjectura lui Goldbach	148
Problema 9. Elemente vecine cu diferență 1	150
Problema 10. Rezolvarea ecuațiilor de gradul al II-lea	152
Problema 11. Căutarea unui element într-un tablou unidimensional	154
Problema 12. Sortarea coloanelor în matrice	155
Problema 13. Elemente simetrice în tablou pătratic	157
Problema 14. Pointer de parcurgere al unui tablou	159
Problema 15. Adrese în tablouri	160
Problema 16. Interschimbare de octeți	161
Problema 17. Tablou cu pointeri la funcții	163
Problema 18. Puncte-șă	164
Probleme propuse	166
Șiruri de caractere. Operații cu fișiere	171
Problema 1. Copierea unui sir constant (caracter cu caracter, pointer)	173
Problema 2. Utilizarea funcțiilor <code>strcat</code> , <code>strlen</code> , <code>strchr</code> , <code>strrchr</code>	174
Problema 3. Compararea a două siruri (<code>strcmp</code> , <code>stricmp</code>)	175
Problema 4. Primul subșir care începe cu un caracter din alt sir (<code>strpbrk</code>)	177
Problema 5. Găsirea unui subșir într-un sir (<code>strstr</code>)	178
Problema 6. Adunarea a două numere (utilizarea funcțiilor <code>strlen</code> , <code>strcat</code>)	179
Problema 7. Sevență de căutat	183
Problema 8. Păsărește	185
Problema 9. Utilizarea funcțiilor <code>strncpy</code> , <code>strcat</code> , <code>strlen</code>	187
Problema 10. Utilizarea funcției <code>strtok()</code>	189
Problema 11. Concatenarea a două fișiere	190
Problema 12. Argumente în linia de comandă	192
Probleme și exerciții propuse	193

Structuri, unuini, câmpuri de biți	197
Problema 1. Puncte colineare	199
Problema 2. Suma a două fracții	201
Problema 3. Reuniune de intervale	203
Problema 4. Diferența, reuniunea și intersecția a două multimi	204
Problema 5. Apartenența unui punct la disc	207
Problema 6. Test union	209
Problema 7. Câmpuri de biți	212
Probleme propuse	213
Operații pe biți	215
Problema 1. Reprezentare binară	217
Problema 2. Operații elementare pe biți	218
Problema 3. Împachetarea datei	219
Problema 4. Diferite operații folosind biți	221
Problema 5. Numărul biților 1 în reprezentarea în baza 2	224
Problema 6. Ciurul lui Eratostene pe biți	225
Probleme propuse	227
Funcții de timp, numere aleatorii	229
Problema 1. Ziua din săptămână	231
Problema 2. <code>rand()</code> , <code>qsort()</code> și <code>bsearch()</code>	233
Problema 3. Cap și pajură	235
Problema 4. Hârtie-pumn-foarfece	237
Problema 5. Timp simplu	239
Problema 6. Pauză în secunde	240
Problema 7. Data și ora curente	241
Problema 8. Trecut sau viitor	242
Probleme propuse	243
Structuri cu autoreferire	247
Problema 1. Cuvinte în propoziție	249
Problema 2. Cuvinte cu majuscule ordonate crescător	251
Problema 3. Cuvinte cu majuscule ordonate crescător și luate o singură dată	253
Problema 4. Matrice rare	254
Problema 5. Joc de copii I	262
Problema 6. Joc de copii II	265
Problema 7. Tabela de dispersie (Hash Table)	269
Problema 8. Crearea listelor	274
Problema 9. Stiva (Stack)	278
Problema 10. Cărți de joc	282
Problema 11. Parcurgerea arborelui binar	289
Probleme propuse	292
Grafuri, tehnici de programare	295
1. Grafuri	297
Metode de reprezentare	298
Problema 1.1. Parcurgerile BFS și DFS	298

<i>Problema 1.2. Matricea drumurilor</i>	301
2. Recursivitate	303
<i>Problema 2.1. Suma cifrelor unui număr natural</i>	303
<i>Problema 2.2. Numărul 4</i>	304
3. Divide et impera	306
<i>Problema 3.1. Aflarea celui mai mare divizor comun a n numere naturale</i>	306
<i>Problema 3.2. Turnurile din Hanoi</i>	308
4. Greedy	309
<i>Problema 4.1. Problema continuă a rucsacului</i>	310
<i>Problema 4.2. Colorarea hărții</i>	312
5. Backtracking	314
<i>Problema 5.1. Colorarea hărții</i>	314
<i>Problema 5.2. Problema fotografiei</i>	316
6. Programare dinamică	318
<i>Problema 6.1. Determinarea combinărilor</i>	319
<i>Problema 6.2. Cuvinte potrivite</i>	320
7. Probleme propuse	322
Probleme de concurs	327
<i>Problema 1. Cuburi</i>	329
<i>Problema 2. Numere Bangla</i>	331
<i>Problema 3. Să tăiem pizza!</i>	333
<i>Problema 4. Secvență Collatz</i>	335
<i>Problema 5. PI</i>	337
<i>Problema 6. Intersecție de cercuri</i>	339
<i>Problema 7. Sortarea DNA-ului</i>	343
<i>Problema 8. Cutii ascunse</i>	345
<i>Problema 9. Numere fel de fel</i>	349
<i>Problema 10. Big Mod</i>	351
<i>Problema 11. Numere umile</i>	352
<i>Problema 12. Moneda contrafăcută</i>	354
<i>Problema 13. Numere unu</i>	358
<i>Problema 14. Codul secret</i>	360
<i>Problema 15. Împărțirea cărjilor la Bridge</i>	364
<i>Probleme propuse</i>	368
Aplicație : numere mari	377
Aplicație : fractali space-feeling	394
Bibliografie	405

Cuvânt înainte

Primul pas este cel mai greu.

Concepțută ca un manual pentru cei ce vor să înceapă studiul programării, cartea are ca scop introducerea în domeniu și formarea unor abilități de abordare a situațiilor practice, dar suportul teoretic riguros și varietatea temelor prezentate o indică și celor având deja cunoștințe avansate. O recomandăm elevilor, studenților, programatorilor, profesorilor de informatică și tuturor celor ce doresc să își perfecționeze cunoștințele în domeniu, putând fi folosită pentru studiul individual, dar și ca material de lucru la orele de curs.

Considerăm că limbajul C este cel mai indicat limbaj pentru începerea studiului programării, fiind puternic, flexibil, eficient și stând la baza limbajelor orientate pe obiect moderne : C++, Java, C#. Înainte de a lucra la nivelul unui limbaj orientat pe obiect sunt necesare dezvoltarea unei gândiri algoritmice, structurate, și stăpânirea conceptelor de bază (elaborarea algoritmilor, variabile, instrucțiuni, modularitate, alocarea de resurse). Cei care își însușesc aceste aspecte vor vedea ca pe un lucru firesc continuarea cu C++ sau Java, limbaje ce aduc în plus facilități de abstracțizare a datelor și suportă programarea orientată pe obiect.

Cele 125 de probleme complet rezolvate au grade diferite de dificultate. Pe lângă descrierea problemelor este furnizat și un set de date de intrare și ieșire elocvent, astfel încât cititorul să își poată evalua o eventuală soluție proprie. Imediat după enunțul problemei se continuă cu o parte de analiză a acesteia și cu descrierea unei metode de rezolvare, ce se regăsește în *listing*-ul complet al programului aferent. Urmează câteva exerciții înrudite cu problema analizată. O serie de probleme propuse se găsesc și la sfârșitul fiecărui capitol, în total cartea conținând 400 de exerciții și probleme propuse.

De-a lungul a 14 capitole, cititorul este invitat la o călătorie care începe cu prezentarea unor fapte istorice și teoretice, urmărind evoluția sistemelor de calcul de-a lungul timpului, aspecte teoretice și practice privind definirea și elaborarea algoritmilor și prezentarea limbajului C. Drumul anevoiește de la abac până la calculatorul modern, cu puncte-cheie precum mașina de calcul a lui Pascal, cea de jesut a lui Jacquard, mașina analitică a lui Babbage și contribuția Adei Lovelace, efectul Edison, *Complex Number Calculator*, ENIAC, apariția cipului de siliciu, este descris cronologic în primul capitol. Notiunea de algoritm, cu istoricul său, elementele caracteristice, clasificări și exemple

sugestive, este prezentată în cel de al doilea capitol, iar elementele de bază ale limbajului C sunt descrise concis și susținute de exemple sugestive în capitolul al treilea.

Cartea continuă cu șapte capitole ce conțin probleme rezolvate și propuse, care își propun familiarizarea treptată cu limbajul C și cu elaborarea soluțiilor pentru probleme concrete (instrucțiuni simple, siruri de caractere, tablouri, structuri, uniuni, pointeri, operații pe biți, funcții de timp, numere aleatorii).

Ultimele cinci capitole conțin elemente mai avansate (structuri înlántuite, grafuri, *Backtracking*, recursivitate, programare dinamică, probleme de concurs, fractali), unul dintre scopuri fiind și acela de a mări interesul și curiozitatea cititorului în ceea ce privește programarea. Capitolul 10, deosebit de interesant, conține 11 probleme complet rezolvate folosind structuri cu autoreferire (liste simplu și dublu înlántuite, circulare, *Hash Table*, stivă și arbori) și numeroase probleme propuse. Următorul capitol prezintă noțiunea de graf și tehniciile tradiționale de programare (recursivitatea, *Divide et impera*, *Backtracking*, *Greedy*, programarea dinamică), împreună cu probleme clasice rezolvate și propuse. Capitolul 12 conține 15 probleme rezolvate, în general propuse la concursul studentesc ACM, și peste 30 de exerciții și probleme propuse. Se continuă cu realizarea unei aplicații complexe care utilizează liste simplu înlántuite pentru reprezentarea numerelor naturale mari. În scopul calculării radicalului de un ordin dat k dintr-un număr mare, se vor scrie metodele corespunzătoare ce operează cu astfel de numere reprezentate ca liste. Ultimul capitol este o aplicație inedită cu fractali *space-feeling* care folosește, pe lângă alte noțiuni, operațiile pe biți și o rutină grafică.

În încheiere, vă rog pe dumneavoastră, cititorii pasionați, să îmi împărtășiți ideile, completările și sugestiile la adresa : doinabooks@yahoo.com.

Doina Logofătu
München, februarie 2006

Evoluția sistemelor de calcul

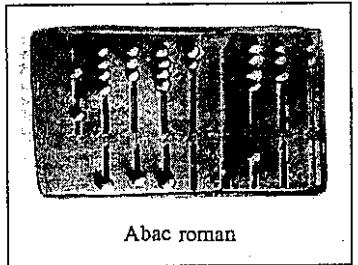
- Abacul – invenție revoluționară
- Noi instrumente de calcul, concepte matematice, personalități multivalente : Napier, Pascal, Fermat, Leibniz
- Xavier Thomas de Colmar și primul calculator pentru mase
- Spre sistemul binar : mașina pentru stofe a lui James Jaquard
- Charles Babbage și Lady Ada Lovelace
- Creatorul algebrei Boole
- Inventarea mașinii de scris
- Hollerich și Powers : noi concepte importante
- Efectul Edison, electricitate, comutatoare, amplificatoare
- Analizorul diferențial al lui Bush, mașina Turing
- Redescoperirea târzie a algebrei Boole
- Primul computer electronic digital : *ABC*
- George Stibitz : calcule de la distanță
- Konrad Zuse – inventatorul computerului
- *ENIAC* sau succesul computerelor digitale
- Ideile revoluționare ale lui John von Neumann
- Succesoriile lui *ENIAC* : *EDVAC* și *UNIVAC*
- Circuitul integrat, cipul, cipul de siliciu
- Incursiune în viitor
- Exerciții, întrebări

*Întotdeauna provoacă unul singur scânteia care împinge
omenirea înainte.*

Igor Sikorsky (1889-1972),
inventatorul elicopterului

1. Abacul

La începuturi se foloseau pentru calcule cele zece degete (de unde și geneza cuvântului *digital - digit*), ceea ce a condus la marea popularitate de-a lungul istoriei a sistemului zecimal de calcul. Mai târziu, în Antichitate, au fost folosite pietricele, pentru a ilustra numărul de obiecte în posesie, metodă eficientă pentru lucrul cu adunări și scăderi. În unele culturi, pietricele au fost înlocuite cu bețișoare, noduri la o sfoară, semne pe o tăblă de argilă etc. Cuvântul *abacus* provine de la cuvântul arab *abaq*, care înseamnă fără și era o tablă portabilă cu șanțuri de nisip, pe care se puteau amplasa pietricele. Ulterior, aceste pietricele au fost înlocuite cu mărgele, șanțurile cu sfuri și tabla cu un cadru, formând ceea ce noi cunoaștem sub numele de *numărătoare*. În Antichitate, abacul a fost o invenție revoluționară, utilizată mii de ani pentru calcule rapide, deși au fost descoperite felurile table cu numere de către greci, romani, mesopotamieni, egipteni. Utilizatorii abacului au devenit extrem de pricepuți în calcule, unele teste arătând că un expert în folosirea abacului poate fi mai rapid decât un calculator mecanic. După descoperirea abacului, apariția altor instrumente de calcul a stagnat pentru mai mult de 2000 de ani; se pare că în Evul Mediu știința și comerțul nu au necesitat calcule mai înaintate.



Abac roman

2. Noi instrumente de calcul

Începuturile computerelor moderne se conturează abia în secolul al XVII-lea, când personalități ca Descartes, Pascal, Fermat, Leibniz și Napier revoluționează conceptele asupra lumii ale Antichității în domenii precum filosofia, matematica, astronomia. În matematică, progresul este evident și de aceea devine necesară apariția unor instrumente mai sofisticate de calcul. Inventarea logaritmilor în 1614 de către matematicianul scoțian John Napier (1550-1617) stimulează apariția unor suporturi care să execute adunări și

scăderi de logaritmi. Tabelele sale de logaritmi au ajuns repede cunoscute și intens utilizate; în plus, a inventat și o mașină simplă pentru înmulțiri cu o cifră. O serie de alți oameni de știință pasionați au continuat munca lui Napier, inventând diverse instrumente de calcul (germanul Wilhelm Schickard, 1623, englezii Edmund Gunter și William Oughtred, 1650 etc.).

3. Blaise Pascal

Un pas important în evoluția calculatoarelor mecanice a fost introducerea, în 1642, a „rojilor dințate”, de către Blaise Pascal (1623-1662), celebrul filosof, fizician și matematician francez. El a construit pentru tatăl său, care lucra la un birou de colectare a taxelor, un sistem mecanizat cu o serie de cadrane atașate unor roți dințate care aveau înscrise cifrele de la 0 la 9 pe circumferință (acest aparat era utilizat doar pentru adunări și scăderi). Visul



Mașina de calcul a lui Pascal

său era să construiască un aparat cu inteligență mecanică, care să execute toate operațiile posibile. Și pentru că distanța de la adunare la înmulțire nu e aşa de mare, nu va trece mult timp până la descoperirea unei mașini de înmulțit prin adunarea repetată a numerelor. Ca filosof, Blaise Pascal a căutat supraviețuirea sufletului, ca poet și scriitor a scris contemplațiile sale despre „măreția și mizeria umană”, dar cel mai puternic s-a remarcat prin studiile din domeniile fizicii și matematicii. Împreună cu Fermat, este considerat a fi coinventatorul teoriei probabilităților, a descoperit, în fizică, *Legea lui Pascal* (presiunea lichidului într-un container este constantă), în geometrie, teorema hexagonului (când avea 16 ani), triunghiul lui Pascal și aplicabilitatea lui în expresii binomiale și probabilistică. Iată și câteva dintre afirmațiile sale:

Claritatea minții înseamnă și claritatea pasiunii; de aceea, o minte clară și minunată iubește cu ardoare și vede clar ceea ce iubește.
Am descoperit că totă nenorocirea umană vine de la aceea că omul nu e capabil să stea liniștit într-un loc.
De vreme ce nu putem să totul despre toate, ar trebui să știm câte puțin din toate.
Tăcerea acestor spații infinite nu umple de spaimă.
De obicei, suntem mai convinși de cauzele pe care le descoperim singuri decât de cele pe care ni le oferă alții.
Ajungem la adevăr nu doar datorită unei cauze, ci și datorită inimii.

Am făcut această scrisoare mai lungă, pentru că nu am avut timp să o fac mai scurtă. Omul nu găsește nimic mai intolerabil ca faptul de a fi într-o stare de complet repaus, fără pasiuni, fără ocupație, fără diversiune, fără efort. Atunci își simte nulitatea, singurătatea, imperfecțiunea, dependența, lipsa de ajutor și de conființă.

Un fleac ne consolăază pentru că un fleac ne produce necazuri.

Toate cauzele sfârșesc atunci când dispar sentimentele.

Imaginația decide totul.

Râul este mai pur decât izvorul său.

Omul este o trestie, cea mai slabă din natură, dar este o trestie gânditoare.

Spre sfârșitul vieții a suferit serioase probleme de sănătate și s-a concentrat pe scrierea de filozofie religioasă (s-a izolat de lume din cauza puternicelor convingeri religioase). Unul dintre visele sale, acela de a crea o mașină inteligentă, „dinspre suflet, spre folosire mecanică”, avea să se împlinească pas cu pas de-a lungul timpului, prin implicarea altor oameni de știință pasionați.

4. Gottfried Wilhelm von Leibniz

Filosoful și matematicianul german Gottfried Wilhelm von Leibniz (1646-1716) a adus îmbunătățirile necesare mașinii lui Pascal în 1671. *Numărătoarea lui Leibniz* (bazată pe *roata lui Leibniz*) a fost primul calculator în două mișcări proiectat pentru înmulțirea prin adunări repetitive. Leibniz avea o cultură enciclopedică, consolidată devreme în timpul studiilor săle la Universitatea din Leipzig. Interesul său pentru matematică creștea pe măsură ce cinea tot mai multe opere filozofice. A susținut un doctorat în drept la Universitatea de la Nürnberg, a practicat dreptul la Paris pentru a-și putea susține studiile sale de matematică, a ocupat funcții de sfătuitor pe lângă personalități politice ale vremii. În biografia sa din *Encyclopaedia Britannica*, el este descris astfel:

Leibniz era un om de statură mijlocie, gârbovit și crăcănat, la fel de capabil să gândească câteva zile în același scaun ca și să călătorească pe străzile Europei vară și iarnă. Era un muncitor neobosit, un corespondent universal (a avut peste 600 de corespondenți), un patriot și un cosmopolit, un mare om de știință și unul dintre cele mai puternice spirite ale civilizației vestice.

În 1675, Leibniz a descoperit calculul integral și calculul diferențial (independent de Isaac Newton), iar notațiile sale pentru metodele de calcul ale diferențialelor și integralelor au rămas valabile până astăzi. El a publicat foarte mult, atât lucrări filozofice



Gottfried Wilhelm von Leibniz

inovatoare, cât și matematice, iar corespondențele sale aveau ca domenii matematică, logica, istoria, filosofia, dreptul, politica, teologia.

5. Xavier Thomas de Colmar și primul calculator „pentru mase”

Francezul Xavier Thomas de Colmar (1785-1870) a creat *Arithmometrul*, primul calculator produs în masă, ce a continuat să fie vândut timp de 90 de ani (între 1820 și 1878 au fost construite peste 1.500 de astfel de mașini). Au fost folosite cilindrii în treaptă și activatorul digital inventat de Leibniz.

6. Mașina lui Joseph Jacquard

Una dintre invențiile revoluției industriale ce a avut legătură directă cu computerele este făcută în 1801 de către francezul Joseph Jacquard (1752-1834). El a perfectat prima mașină de perforat cartele, cu scopul de a crea modele pentru stofe. Un celebru portret al inventatorului a fost produs cu ajutorul a 24.000 de cartele perforate. Când Jacquard și-a introdus mașina pentru prima dată, a întâmpinat dificultăți în a fi acceptată din cauza „fricii de mașini”. În orașul Lyon, el a fost atacat fizic și mașina i-a fost distrusă. Dar ceea ce a reușit Jaccquard cu acest dispozitiv a fost stabilirea unei comunicări om-mașină cu ajutorul a două cuvinte: gaură și „ne-gaură”, prefăjare a sistemului binar devenit ulterior universal.



Mașina lui Jacquard

7. Charles Babbage și Lady Ada Lovelace

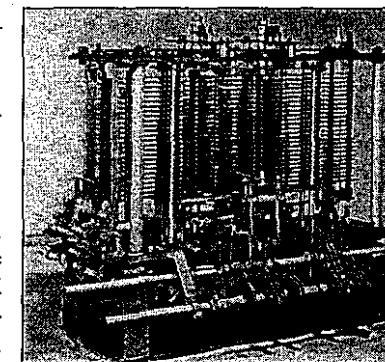
Charles Babbage (1792-1871) nu are faima de a fi construit primul computer din lume, dar a încercat din greu și a fost foarte aproape să o facă. Singura sa greșeală a fost că s-a născut într-un timp când tehnologia de a construi un computer modern nu era aşa de avansată ca și știința proiectării acestuia. Profesor la Cambridge, el a fost un fanatic privind acuratețea matematică.

A proiectat, în 1822, o mașină mecanică automată de calcul, mașina diferențială, care s-a dovedit mult mai greu de construit decât își imaginase; guvernul britanic și-a

retras fondurile după un timp, iar Babbage a încercat să continue construcția din fonduri personale, până când acestea s-au terminat și a fost nevoie să renunțe. Această mașină, deși avea adaptabilitate și aplicabilitate limitate, a reprezentat un progres important. Mai târziu, Babbage a inventat o altă mașină, proiect mult mai ambicios, *mașina analitică*, care includea cinci aspecte cruciale ale viitoarelor calculatoare:

- un suport de intrare;
- o modalitate de stocare a numerelor pentru procesare;
- un procesor sau calculator pentru numere;
- o unitate de control al acțiunilor ce trebuie executate;
- o unitate de ieșire.

Babbage nu a definitivat niciodată construcția, dar structura logică a calculatoarelor actuale vine de la el, una dintre caracteristicile de bază fiind conceptual de „program stocat”, necesar implementării unui compilator. Relativ la prima mașină a lui Babbage, mașina diferențială, ea a fost construită de suedezul Georg Scheutz, care a avut fondurile necesare și a creat o variantă simplificată a acesteia, înțând cont și de *Observațiile* Adei Lovelace. Mașina a obținut medalia de aur la o expoziție de la Paris din 1855. Babbage era prezent în audiență și a fost frustrat și descuprănit. Ulterior, mașina a fost cumpărată de către guvernul englez, același care sistase fondurile de realizare a ei. Foarte interesant este faptul că în 1991, deci la 120 de ani de la moartea sa, oamenii de știință britanici au reconstruit *mașina analitică* a lui Babbage, urmându-i întocmai planurile. Rezultatul a fost o mașină imensă care a funcționat perfect, iar calculele aveau o precizie de 31 de cifre. Aceasta poate fi admirată astăzi la Muzeul Britanic, unde se remarcă acuratețea gândirii lui Babbage, migala și răbdarea cu care a proiectat-o.



Lady Ada Lovelace (1815-1852), fiica Lordului Byron, a devenit implicată în proiectare. Ea l-a ajutat pe Babbage nu doar finanțar, ci și prin importante contribuții matematice cu privire la *mașina analitică*. Ada Lovelace a fost printre puținii din vremea sa care i-a înțeles importanța și potențialul, scriindu-i acestuia: „Putem spune că mașina analitică țese modelele algebrice la fel cum războiul de țesut a lui Jaquard țese flori și frunze”. Ada a fost fascinată de mașinile lui Babbage și a acceptat, în 1843, să traducă din franceză lucrarea *Elemente ale mașinii analitice a lui Charles Babbage* a lui Luigi Frederico Menabrea. Această lucrare și-a triplat volumul datorită notelor și observațiilor personale ale Adei. Babbage, persoană nu tocmai plăcută în particular, a fost impresionat și i-a scris: „pe măsură ce citeam notele dumneavoastră, mi-a crescut și surprinderea cu privire la ele și regret că nu am explorat mai din timp un filon așa de bogat în metalul cel mai nobil”. În iulie 1843, Ada îi scrie lui Babbage, cerându-i părerea: „Aș vrea să pot introduce ceva de genul numerelor lui Bernoulli... ca un exemplu despre cum o funcție trebuie să fie prelucrată de mașină, fără a fi lucrată de mintea și mâna omului”. Rezultatul a fost larg acceptat ca fiind primul program de computer și chiar dacă nu a fost transpus în practică în timpul vieții sale, cu sistemele din zilele noastre programul propus de ea pentru calculul numerelor lui Bernoulli produce rezultate corecte. De exemplu, ea

scrie : *Here follows a repetition of operations 13 to 23* („Urmează o repetiție a operațiilor 13 până la 23”), introducând ideea de *loop*. Babbage însuși a recunoscut că Ada îi înțelege *mașina analitică* chiar mai bine decât el însuși și că era cu mult mai bună decât el în a o explica. Una dintre ideile ei a fost și conectarea mașinii cu muzica, ea scriindu-i lui Babbage în 1843 : „Din nou, [mașina analitică] poate lucra și cu alte obiecte în afară de numere, acolo unde obiectele ale căror relații mutuale fundamentale pot fi exprimate de cele ale științei abstracte a operațiilor, și care pot fi susceptibile la adaptări pentru a acționa cu notația operațiilor mașinii... Presupunând, de exemplu, că relațiile fundamentale ale diferitelor sunete în știința armoniei și a compoziției muzicale ar fi susceptibile unor astfel de expresii și adaptări, mașina ar putea elabora și piese muzicale științifice de orice fel de complexitate și întindere”. Augusta Ada Byron, contesă de Lovelace, este prima femeie programator, ideile sale de *loop* și subrute au devenit bazele programării și este singura femeie căreia i s-a dedicat un limbaj de programare – ADA –, dezvoltat de Departamentul de Apărare al Statelor Unite, primul limbaj standardizat de nivel înalt. Ea a fost un matematician prolific de la o vîrstă fragedă (într-o lume ostilă, în care doamnele nu făceau astfel de lucruri), iar flerul și premonițiile sale privind gândirea logică și matematică erau spectaculoase și inedite. Din păcate, a murit la doar 36 de ani, întrerupându-și prea devreme activitățile creațioare.



8. Părintele algebrei Boole



Englezul George Boole (1815-1864) a introdus ideea de a reprezenta informația doar cu ajutorul celor două valori logice, adevărat și fals, plus concepte și formule de a manipula aceste informații. Algebra Boole devine teoria matematică necesară pentru a opera cu sistemul binar. La vîrstă de 24 de ani, George Boole își publică primul său articol, „Cercetări privind teoria transformărilor analitice”, în *Cambridge Mathematical Journal*, iar în următorii 10 ani publică o serie de articole care largesc limitele matematicii „moderne”. În 1844, s-a concentrat pe utilizarea algebrei combinate și, în același an, a primit o medalie de la Societatea Regală pentru contribuțiile sale privind analiza. Curând, Boole a observat posibilitățile aplicării algebrei sale pentru soluționarea problemelor logice, iar în lucrarea din 1847, *Analiza matematică a logicii*, nu numai că a extins speculațiile lui Leibniz privind legătura dintre logică și matematică, dar chiar a afirmat că logica este mai curând o disciplină matematică decât una filosofică. Concentrându-se pe rafinarea conceptelor sale, a căutat să creeze un limbaj logic care să poată fi manipulat matematic, ajungând la o algebră lingvistică cu trei operatori de bază : AND, OR și NOT. Sistemul lui Boole

(detaliat în *O investigație a legilor gândirii, pe care sunt fondate Teoriile Matematice ale Logicii și Probabilităților*, 1954) se baza pe o concepție binară, procesând două obiecte : atât de cunoștutele acum da/nu (yes/no), adevărat/fals (true/false), on/off, zero/unu (zero/one). Surprinzător, comunitatea academică a vremii a ignorat sau chiar criticat această teorie, doar eminentul om de știință american Charles Sanders Peirce, intuindu-i perspectivele, a realizat potențiala utilizare în circuitele electronice, fără a reuși însă construcția unui aparat teoretic corespunzător. Meritul său este acela de a fi introdus algebra booleană în cursurile de filosofie logică de la universitatea sa. Cîțiva ani mai târziu, studentul la Massachusetts Institute of Technology (MIT) Claude Shannon avea să găsească legătura concretă dintre logica booleană și circuitele electrice.

9. Inventarea mașinii de scris

În 1867, americanii Christopher Latham Sholes, Carlos Glidden și Samuel W. Soule au inventat prima mașină de scris, iar în 1885, Dorr Eugene Felt a creat *comptometrul*, primul calculator care permitea introducerea numerelor prin apăsarea unor taste, conectate cu o serie de metode primitive.

10. Hollerich și Powers – noi concepte importante

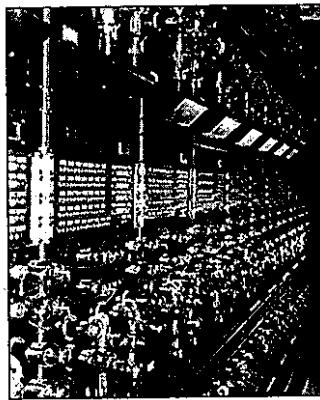
Un semnificativ pas înainte în calculul automat a fost redescoperirea cartelelor perforate, introduse cu succes în 1890 de către Herman Hollerich și James Powers. Ei au proiectat un sistem prin care informații relevante despre o persoană (nume, vîrstă, sex) puteau fi codificate cu ajutorul unor găuri în cartele cu 240 de poziții (20 de rânduri și 12 coloane). Aceste cartele perforate au introdus concepte precum intrare/ieșire (*input/output*) și stocarea memoriei la scară mare. În 1896, Dr. Hollerich a fondat Tabulating Machine Company pentru a promova utilizarea comercială a acestei mașini, urmând ca în 1924, după câteva transformări și redenumiri, compania să se numească International Business Machines (IBM).

11. Efectul Edison, electricitate, comutatoare și amplificatoare

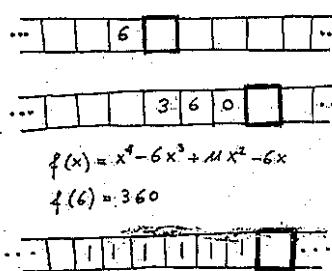
Alte descoperiri importante au fost becul electric și *Efectul Edison*, descoperite în 1879 de către inventatorul american Thomas Alva Edison (1847-1931), detectarea undelor radio și convertirea lor în electricitate de către fizicianul englez John Ambrose Fleming. În 1906, inventatorul american Lee De Forest (1873-1961) a introdus al treilea electrod, având ca rezultat o *triodă*; rolul de amplificator sau comutator și multe transmițătoare radio au fost construite folosind aceste triode. Triodele introduse de De Forest au revoluționat cîmpul radiodifuziunii și au fost destinate pentru a realiza chiar mai mult. Datorită abilității lor de a funcționa drept comutatoare, ele vor avea un impact extraordinar asupra calculului digital.

12. Analizorul diferențial al lui Bush

În 1927, cu ajutorul a doi colegi de la MIT, politologul, inginerul și omul de știință american Vannevar Bush (1890-1954) a proiectat un computer analog pentru rezolvarea ecuațiilor simple și, în 1930, a construit o versiune mai avansată, numit *analizor diferențial*, ce lucra cu motoare electrice și operațiile erau mecanice. În 1935, Bush a proiectat a doua versiune, în care angrenajele se deplasau electromecanic și care includea casete de hârtie pentru a transporta instrucțiunile și a seta angrenajele. În zilele noastre, când un procesor are mărimea unui timbru poștal, este foarte dificil să ne imaginăm asemenea mașini gigantice (a doua mașină a lui Bush avea 100 de tone, 2.000 de lămpi, sute de relee, 150 de motoare, 300 de kilometri de cabluri).



13. Mașina Turing



Matematicianul englez Alan Mathison Turing (1912-1954) a fost unul dintre marii pionieri în domeniu; el a scris în 1937 articolul „Despre numere calculabile folosind aplicarea problemei de decizie”. Una dintre premisele articolului său a fost faptul că unele clase de probleme matematice nu pot avea reprezentări algoritmice și deci nu pot fi rezolvabile cu ajutorul computerelor automate. Turing a inventat propriul model teoretic de computer, numit *mașina Turing*, un calculator ipotetic care prefațează era calculatoarelor programabile. Mașina Turing a fost proiectată să execute operații logice și să poată citi, scrie, sterge simboluri, care erau de obicei 0 și 1, scrise în celule pe o bandă infinită. Mașina Turing citește și execută o secvență de pași, mai fiind cunoscută și ca „mașină cu stări finite”, deoarece pentru fiecare pas în cadrul calculului, următoarea acțiune este aleasă dintr-o listă finită de acțiuni posibile. În acest fel, Alan Turing a aplicat conceptul de algoritm computerelor digitale, iar cercetările relației dintre mașini și natură au dus la apariția inteligenței artificiale.

14. Impactul algebrei Boole asupra circuitelor electronice

În 1937, adică la aproape 75 de ani de la moartea lui Boole, apare un prim articol despre legătura dintre algebra booleană și circuitele bazate pe comutatoare (*switch-uri*), scrisă de către Claude Shannon (1916-2001), student la MIT. El a transferat cele două stări

logice circuitelor electronice și a demonstrat cum conceptele algebrei Boole pot fi utilizate pentru a reprezenta funcțiile comutatoarelor în circuitele electronice, instrumente matematice care au rămas o piatră de bază în proiectarea electronică digitală până în zilele noastre. Teza lui Shannon a avut un impact deosebit, deoarece ulterior atenția comunităților științifice internaționale din domeniu s-a concentrat pe construcția de mașini de calcul bazate pe logica formală.



15. Primul computer electronic digital – ABC

În 1939, Atanasoff și Berry au construit primul computer electronic digital din lume – Atanasoff Berry Computer (ABC). În 1937, profesorul Howard Aiken de la Harvard a început construcția unui sistem pentru calculul automat, creând seria calculatoarelor Mark. Mark I a fost terminat cu ajutorul inginerilor de la IBM, avea dimensiuni gigantice, folosea cartele perforate ca intrare pentru luarea deciziilor de-a lungul unor sisteme electromecanice (adunarea și scăderea luau 0,3 secunde, înmulțirea – mai puțin de 6 secunde și împărțirea – mai puțin de 16 secunde), iar rezultatele erau scrise tot pe cartele perforate. Putea să execute cele cinci operații, dar poseda și subroutines pentru a lucra cu funcții logaritmice și trigonometrice. Această mașină a fost folosită intens de către armata americană în perioada celui de-al doilea război mondial.

16. George Stibitz, primul computer electronic digital, accesul de la distanță

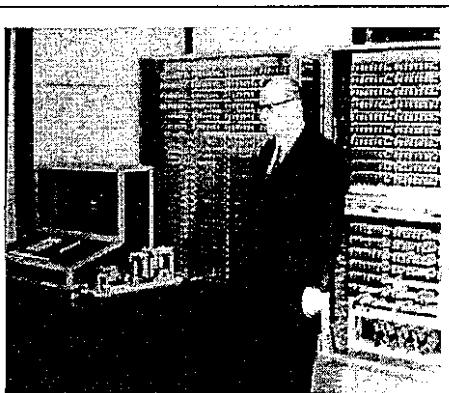
Cam în aceeași perioadă când Claude Shannon lucra la teza sa de masterat despre algebra booleană și circuitele electronice, George Stibitz (1904-1995), cercetător la Bell Labs, a avut aceleași idei. Realizând că logica booleană poate fi utilizată pentru comunicarea dintre releele telefonului electromecanic, el a conectat împreună un conglomerat de relee, baterii, lămpi, sârme și bucați de cositor, pe care le-a meșterit pe masa sa din bucătărie în 1937. Rezultatul a fost un circuit ce controla adunarea binară, pe care l-a incorporat în calculatorul digital *Model K* (de la *kitchen* – bucătărie, în engleză). Doi ani mai târziu, în colaborare cu Samuel Williams, a construit la Bell Labs o mașină care executa toate cele patru operații matematice pentru numere complexe. *Complex Number Calculator* (redenumit mai târziu *Bell Labs Model Relay Computer*)



a fost larg recunoscut ca fiind primul computer electronic digital din lume. În 1940, Stibitz a instalat invenția la sediul central al companiei de la Manhattan, legând de ea trei mașini telex separate din aceeași clădire, permitând astfel folosirea computerului de la

distanță. Nouă luni mai târziu a adăugat al patrulea telex, de astă dată la 400 de kilometri distanță, în New Hampshire. Acolo, în fața unei audiențe sceptice formată din membri ai Societății Matematice Americane, Stibitz a performat calculul de la distanță prin transmisarea și primirea datelor spre/dinspre computer prin intermediul telexului, moment în care s-au modificat decisiv concepțele despre utilizarea computerului.

17. Konrad Zuse



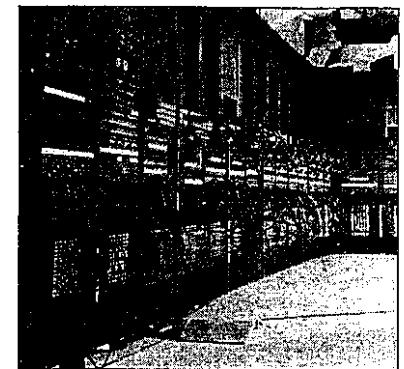
Konrad Zuse în fața lui Z3, 1941

transportat în circumstanțe aventuroase de la Berlin la Göttingen și apoi la Allgäu. Ascunsă într-un grajd, mașina rămâne nedescoperită de către părți și a fost transportată mai târziu la Institutul Federal de Tehnologie de la Zürich. O altă creație a geniuului Zuse este primul limbaj de programare algoritmic, *Plankalkül*, descoperit în 1945-1946. A fost numit de către mulți „inventatorul computerului”. Z3 era controlat de o casetă perforată, folosea role de film uzate, introducerea datelor se făcea cu o tastatură, afișarea – cu un ecran cu lămpi. Întreaga mașină se baza pe tehnologia telelor și necesita 2.600 de relee: 1.400 pentru memorie, 600 pentru unitatea aritmetică, restul pentru felurite circuite de control. Ele erau montate în trei stative, două pentru memorie și unul pentru unitățile aritmetice și de control, fiecare de aproape 2 metri înălțime și 1 metru lungime. Memoria pe 64 de cuvinte era în binar cu reprezentare în virgulă mobilă. Viteza lui Z3 era comparabilă cu a lui Harvard Mark I. Z3 a funcționat din 1939 până la 5 decembrie 1941. A rămas în casa lui Zuse până când a fost distrus de un raid aerian în 1944.

18. ENIAC – succesul computerelor digitale

La începutul celui de-al doilea război mondial crescuse necesitatea unor calculatoare performante. În 1942, John P. Eckert, John W. Mauchly și colaboratori de la Facultatea de Inginerie Electrică a Universității din Pennsylvania au decis să construiască un

computer mai performant. Mașina, cunoscută sub numele de *Electrical Numerical Integrator and Calculator* (ENIAC) a fost terminată în 1946, avea 30 de tone și un volum imens, dar putea să execute 500 de adunări și 300 de înmulțiri pe secundă, adică efectua într-o zi cât ar fi făcut o persoană în 300 de zile. Datele de intrare și ieșire erau codificate pe cartele perforate care stoca doar 20 de numere zecimale. Instrucțiunile executabile ale calculatorului erau încapsulate în unități separate ale sale, ce trebuiau conectate pentru a crea „traseul” fluxului de informații. Conexiunile trebuiau refăcute pentru fiecare calcul, împreună cu inițializarea unor tabele și comutatoare. Această tehnică de „cablare” era greoală, dar ENIAC era programabil și a fost eficient pentru programele cărora le-a fost destinat. ENIAC a fost acceptat ca fiind primul computer digital de succes și a fost folosit în perioada 1946-1955.



19. Ideile revoluționare ale lui John von Neumann



Fascinat de succesul lui ENIAC, matematicianul John von Neumann (1903-1957) a scris în 1945 un studiu abstract al calculului în care arăta că un sistem de calcul trebuie să aibă o structură fizică fixă, simplă și să fie capabil de a executa orice fel de calcule pe baza unui control corespunzător programat, fără să fie nevoie să se modifice unități fizice. El recomanda sistemul binar pentru stocarea datelor și propunea instrucțiuni de control și metode de stocare a acestora. Von Neumann a adus o contribuție importantă despre cum sistemele de calcul rapide trebuie organizate și construite. Aceste idei, numite de obicei „tehnica programelor stocate”, au devenit esențiale pentru generațiile viitoare de sisteme de calcul performante și au fost universal acceptate. Tehnica programelor stocate înglobează multe caracteristici ale proiectării și funcționalității sistemelor de calcul și face posibile operații performante, de exemplu 1.000 de operații pe secundă. El a introdus și ideea grupării instrucțiunilor în subrutine care să fie apelate repetat, depinzând de starea și direcția calculului. Von Neumann și-a folosit ideile pentru a construi un nou tip de instrucțiuni-mașină, numit *transfer condițional al controlului*, care permitea ca o secvență de program să fie oprită și pornită din nou în orice moment și stocarea cumulată a tuturor instrucțiunilor programului împreună cu datele în aceeași unitate de memorie, astfel încât, atunci când este necesar instrucțiunile să se poată schimba aritmetic în același timp cu datele. Ca rezultat al acestor tehnici, calculul și programarea au devenit mult mai rapide, mai flexibile și mai eficiente. Subrutinele des folosite nu trebuiau să fie reprogramate pentru fiecare nou program, ele puteau fi păstrate în „biblioteci” și citite din memorie atunci

când erau necesare. Mai mult decât atât, un program poate fi asamblat pe baza metodelor din biblioteci, iar scopul memoriei este de a reprezenta un loc de asamblare în care toate părțile unui calcul laborios sunt păstrate, manipulate bucătă cu bucătă și combinare pentru a furniza rezultatul final. Atunci când avantajele acestor tehnici s-au cristalizat, ele au devenit o practică standardizată.

20. Succesorii lui ENIAC: EDVAC și UNIVAC. Caracteristici

Prima generație de computere moderne programate electronic ce au adus cu ele avantajele acestor îmbunătățiri a fost construită în jurul anului 1940. Acest grup a inclus sistemele de calcul Random Access Memory (RAM), în care memoria era proiectată astfel încât să ofere timp constant de acces la informație pentru orice parte particulară de date. Mașinile aveau cartele perforate sau casete I/O perforate, capacitatea memoriei era de 1.000 de cuvinte, iar timpul de acces, în jur de 5×10^{-6} secunde. Ele erau fizic mult mai mici decât ENIAC, de exemplu, de mărimea unui pian, foloseau în jur de 2.500 de tuburi de electroni, mult mai puține decât cele necesare pentru ENIAC. Sistemele de calcul cu programe stocate din prima generație aveau nevoie de multă întreținere, atingea o acuratețe a operațiilor de 70-80% și au fost folosite între 8 și 12 ani. Primele sisteme de calcul disponibile pentru uz comercial au fost *Electronic Discrete Variable Automatic Computer (EDVAC)* și *Universal Automatic Computer (UNIVAC)*.

EDVAC a fost construit la Facultatea de Inginerie Electrică a Universității Pennsylvania și a fost livrat pentru instalare laboratorului BRL Computing Laboratory în august 1949. Inițial, a avut câteva erori logice, care au fost rezolvate în 18 luni și mașina a intrat în funcțiune în 1951, fiind apoi utilizată timp de 15-20 de ore pe săptămână pentru a rezolva probleme de matematică. Până în 1961, EDVAC a funcționat în medie 145 de ore pe săptămână din 168. EDVAC a fost primul sistem de calcul cu programe stocate intern și organizat în felul următor :

- Control : această unitate conțineă toate butoanele de operare, comutatoarele de control și un osciloscop necesar pentru operațiile de întreținere.
- Dispecer : decodificarea comenzielor primite de la unitatea de control și de la memoria.
- Memoria rapidă : conținea două unități identice, fiecare cu câte 64 de linii de așteptare.
- Calculator : această unitate efectua operațiile raționale (adunare, scădere, înmulțire și împărțire).
- Cronometru : emitea pulsuri de ceas la intervale de o microsecundă și pulsuri de timp la intervale de 48 de microsecunde.

Chiar și după 10 ani de activitate, EDVAC a funcționat în continuare datorită acurateții și productivității sale, a costului redus de operare, a eficienței, vitezei și flexibilității în rezolvarea unor probleme specifice.

21. Circuitul integrat - cipul - cipul de siliciu

Cipul a fost inventat în 1961 de către doi ingineri americani : Jack St. Clair Kilby și Robert Noyce. Crearea lor a revoluționat tehnologia și a pus bazele miniaturizării, deschizând astfel calea spre apariția computerelor moderne. Înainte de inventarea

cipului, cele mai multe dispozitive electrice funcționau pe bază de lămpi și consumau cantități uriașe de energie electrică. Dezvoltarea tranzistorilor a rezolvat parțial această problemă, dar aceștia trebuiau legați prin fire pe plăci. Kilby și Noyce au găsit soluția aproape simultan, combinând componente separate într-un circuit integrat făcut dintr-un material semiconductor. Jack St. Clair Kilby a fost și inventatorul calculatorului de buzunar. Noyce este fondatorul Intel și, la vremea invenției, lucra în Palo Alto, California. El este cel care a introdus adoptarea siliciului dintre materialele semiconductoare. Din acest motiv, Noyce este creditat ca fiind cel care a pus cuvântul silicon (siliciu) în Silicon Valley, mai fiind numit și „Maiorul din Silicon Valley”. Cipul este considerat invenția ultimilor 50 de ani, în cadrul unui sondaj de opinie realizat la începutul anului 2005 de către CNN, 28.500 de persoane din 120.000 (24%) fiind de această părere. Alte invenții importante : Internetul (23.600 – 20%), computerul personal (20.700 – 17%).



22. Generațiile sistemelor de calcul

În concluzie, de-a lungul evoluțiilor tehnice și științifice, sistemele de calcul au evoluat continuu. Se pot chiar determina generații de sisteme de calcul :

1. Prima generație, cea a tuburilor catodice (*vacuum tubes*) – 1940-1950.
2. A doua generație : tranzistoarele – 1950-1964.
3. A treia generație folosea circuite integrate – 1964-1971.
4. A patra generație utilizează cipuri pentru microprocesoare – 1971-prezent.

23. Incursiune în viitor

Încă într-o copilărie relativă, computerele par să fie perfectibile la infinit. Marșul înspre Computopia este mai degrabă liniar – odată cu fiecare pas înainte apar alți cățiva pași înspre acolo și ne conduc spre o progresie clară în direcția unei mai mari inteligențe și mai mari sofisticări. Ceea ce ar trebui să ne readucă chiar mai des la întrebarea fundamentală : Ne vor putea computerele înlocui ? Numai timpul o va spune...

Carol Iacofano

24. Exerciții, întrebări

1. Care a fost primul instrument de calcul ? Ce fel de variante ale acestuia existau ? Cât a durat până la descoperirea de noi astfel de instrumente ? De ce ?
2. Care a fost aportul lui John Napier în evoluția sistemelor de calcul ?
3. Descrieți câteva dintre preocupările și rezultatele lui Blaise Pascal și ale lui Gottfried Wilhelm Von Leibniz. Ce aveau ei în comun ?

4. Care a fost primul calculator vândut la scară largă? Când a fost introdusă prima dată ideea de sistem binar și comunicarea om-mașină?
5. Care sunt cele cinci aspecte cruciale introduse de Charles Babbage odată cu proiectarea mașinii sale analitice? Care sunt ideile și contribuțiile Adei Lovelace privind această mașină? Cum compară Ada mașina lui Jacquard cu mașina analitică? Ce idei de bază din programare îi sunt atribuite Adei? Ce omagiu i-a adus Departamentul Apărării al Statelor Unite?
6. Care sunt operatorii algebrei Boole? Ce afirmă creatorul George Boole despre locul logicii privind matematica și filosofia? A fost teoria să unanim acceptată de comunitatea științifică a vremii? Când avea să se aducă recunoașterea algebrei Boole și în ce domeniu?
7. Ce aduce nou descoperirea efectului Edison?
8. Care sunt contribuțiile matematicianului englez Alan Mathison Turing?
9. Care a fost primul computer digital, cine sunt constructorii și unde a fost el utilizat? Ce operații putea el să execute? Ce performanțe și volum avea?
10. Care este primul computer digital de succes? Cum funcționa el?
11. Ce aduce nou „tehnica programelor stocate” a lui John von Neumann?
12. Care sunt succesorii lui ENIAC? Ce model de calcul și caracteristici au fost înglobate? Care sunt cele cinci dispozitive din care se compun?
13. Ce a revoluționat tehnologia și a pus bazele miniaturizării, deschizând calea către apariția computerelor moderne?
14. Scrieți, cu ajutorul Internetului, un referat despre corespondența lui Leibniz, concepțele descoperite de Pascal sau activitățile Augustei Ada Lovelace.
15. Scrieți-vă părerea despre viitorul mașinilor, pe marginea următoarelor citate:

„Mașina descurajează omul. Acum că mașina este perfectă, inginerul este un nimeni.”

Ralph Waldo Emerson

„A ține în frâu mașina și a limită arta la meșteșug sunt negarea oportunității.”

Lewis Mumford

„Mă interesează viitorul pentru că îmi voi petrece restul vieții acolo.”

Charles Kettering

„Nu mă gândesc niciodată la viitor. El vine oricum destul de repede.”

Albert Einstein

„Oamenii au devenit instrumente pentru instrumentele lor.”

Henry David Thoreau

„Imperiul viitorului este imperiul minții.”

Winston Churchill

Algoritmi – elemente definitorii

- Etimologia cuvântului *algoritm*
- Definiții alternative
- Șapte exemple de algoritmi (*cmmdc*, ciurul lui *Eratostene*, înmulțirea numerelor întregi, maximul a n elemente, căutarea liniară, căutarea binară, *Tiramisu*)
- Abordarea unei probleme (repräsentare, codificare, transformare, proprietăți ale unui algoritm)
- Algoritmica (descrierea, demonstrarea și studiul complexității)
- Noțiunea de model de calcul
- Modelul de calcul RAM (*Random Access Machine*)
- Complexitatea algoritmilor, exemple
- Notațiile Θ , O și Ω pentru analiza complexității-timp
- Optimalitatea, reducerea algoritmilor
- Clasificarea problemelor
- Probleme NP-complete, exemple
- Problema SAT, de ce este ea importantă
- Probleme NP-hard

*Există trei căi de a rezolva intelligent o problemă :
prima este concentrarea, asta e cea mai nobilă ;
a doua e copierea, asta e cea mai ușoară ; a treia
este prin experiență și e cea mai amară.*

Confucius

Chiar dacă computerele și aplicațiile variate corespunzătoare lor sunt un fenomen exploziv ce a devenit predominant în ultimii 20 de ani, o mare parte din concepțele ce stau la baza acestor instrumente soft datează de foarte mult timp, chiar din Antichitate, și ele s-au dezvoltat de-a lungul timpului.

1. Geneza cuvântului *algoritm*

Cuvântul *algoritm* provine de la numele matematicianului, geografului și astrologului arab *Abu Ja'far Muhammad ibn Musa Al-Khwarizmi* (?780-?850, Bagdad). Cea mai importantă lucrare a sa este *Hisab al-jabr w'al-muqabala*, iar titlul va conduce și la geneza cuvântului *algebra*, utilizat pentru a defini această ramură a matematicii. El a lucrat la Casa Înțelepciunii din Bagdad, sub patronajul direct al califului Al-Mamun, unde traducea manuscrisele științifice grecești și studia algebra, geometria și astrologia. Vechii hinduși, greci, babilonieni, romani, chinezi foloseau noțiunea de algoritm pentru operații aritmetice simple.



2. Definiții alternative

În linii mari, relativ la un algoritm putem afirma următoarele :

- Un algoritm este o mulțime de reguli ce se pot aplica în cadrul procesului de construcție a soluției și pot fi executate fie „de mână”, fie de către o mașină.
- Un algoritm este o secvență de pași care transformă mulțimea datelor de intrare în datele de ieșire, rezultatele dorite.
- Un algoritm este o secvență de operații executate cu date ce trebuie organizate în structuri de date.

- Un algoritm este abstractizarea unui program care trebuie executat pe o mașină fizică (model de calcul).
- Un algoritm pentru o problemă dată este o mulțime de instrucțiuni care garantează găsirea unei soluții corecte pentru orice instanță a problemei, într-un număr finit de pași.

3. Exemple

Exemplul 1. Algoritmul lui Euclid

Euclid din Alexandria (?300-260 i.e.n.) este cel mai proeminent matematician al Antichității, bine cunoscut datorită tratatului său de matematică *Elementele*. Perenitatea operei sale ar trebui să îl impună ca fiind cel mai mare matematician al tuturor timpurilor. Despre viață sa se cunoaște foarte puțin, cu excepția faptului că a predat la Alexandria, în Egipt. Cel mai cunoscut algoritm îi aparține, iar acesta se bazează pe una dintre teoremele sale.

Teoremă

Presupunem că a și b sunt două numere naturale, $a \geq b$, și că dorim aflarea celui mai mare divizor comun al celor două numere. Considerăm $a_1 = a$ și $b_1 = b$ și definim perechile (m_i, r_i) astfel încât $a_i = m_i b_i + r_i$, $0 \leq r_i < b_i$. Mai departe, vom considera $a_{i+1} = b_i$ și $b_{i+1} = r_i$. Atunci există un număr natural k astfel încât $r_k = 0$. Mai mult, dacă $r_k = 0$, $\text{cmmdc}(a, b) = r_{k-1}$.



Pe baza acestei teoreme se deduce algoritmul :

1. Citește a, b numere naturale, $a \geq b > 0$
2. $a_1 \leftarrow a, b_1 \leftarrow b, i \leftarrow 1$
3. Cât_timp ($b_i \neq 0$) execută
 - $a_{i+1} \leftarrow b_i$
 - $b_{i+1} \leftarrow r_i$ (= a_i modulo b_i)
 - $i \leftarrow i + 1$
4. $\text{cmmdc}(a, b) = r_{i-1}$.

$a = 77, b = 294$	$a = 2521, b = 338$
$294 = 3 \cdot 77 + 63$	$2521 = 7 \cdot 338 + 155$
$77 = 1 \cdot 63 + 14$	$338 = 2 \cdot 155 + 28$
$63 = 4 \cdot 14 + 7$	$155 = 5 \cdot 28 + 15$
$14 = 2 \cdot 7 + 0$	$28 = 1 \cdot 15 + 13$
	$15 = 1 \cdot 13 + 2$
	$13 = 6 \cdot 2 + 1$
	$2 = 2 \cdot 1 + 0$

Citiți programul C corespunzător din capitolul 4, „Algoritmi elementari”.

Exemplul 2. Algoritmul clasic de înmulțire a numerelor naturale

Următoarele înmulțiri desfășurate descriu sugestiv acest algoritm, cunoscut încă din clasele primare.

Varianta americană, folosită și la noi, în care fiecare cifră din al doilea număr se consideră de la dreapta la stânga	Varianta engleză, în care cifrele se consideră de la stânga spre dreapta
$ \begin{array}{r} 9\ 8\ 1 \\ 1\ 2\ 3\ 4 \\ \hline 3\ 9\ 2\ 4 \\ 2\ 9\ 4\ 3 \\ 1\ 9\ 6\ 2 \\ 9\ 8\ 1 \\ \hline 1\ 2\ 1\ 0\ 5\ 5\ 4 \end{array} $	$ \begin{array}{r} 9\ 8\ 1 \\ 1\ 2\ 3\ 4 \\ \hline 9\ 8\ 1 \\ 1\ 9\ 6\ 2 \\ 2\ 9\ 4\ 3 \\ 3\ 9\ 2\ 4 \\ \hline 1\ 2\ 1\ 0\ 5\ 5\ 4 \end{array} $

Exemplul 3. Ciurul lui Eratostene

Eratostene (?276-194 i.e.n., Cyrene, acum Libia) a fost primul mare geograf al lumii antice și este considerat fondatorul geografiei fizice și matematice. A lucrat mult timp la celebra bibliotecă din Alexandria, Egipt, și a fost, de asemenea, interesat de geometrie și numere prime. El a măsurat primul circumferință pământului (ce s-a dovedit ulterior corectă) și a creat o hartă a lumii, pe care a desenat linii paralele. În plus, a scris un poem în versuri, *Hermes*, despre fundamentele astronomiei ; a sugerat că anii bisecți se repetă la fiecare 4 ani ; a introdus metoda aflării numerelor prime *Sita lui Eratostene*. În memoria sa, există azi un crater pe lună care îi poartă numele.

Definiție : Un număr prim este un număr mai mare decât 1 și care se divide numai cu 1 și cu el însuși. Un număr natural care se divide cu un alt număr decât 1 și el însuși se numește număr compus. *Sita (ciurul) lui Eratostene* este un algoritm care identifică toate numerele prime mai mici sau egale cu un număr natural n dat, astfel :

1. Se scriu în ordine toate numerele de la 1 la n . (Le vom elimina succesiv pe cele compuse prin marcarea lor. Inițial, toate numerele sunt nemarcate.)
2. Marchează 1 ca număr special (nu este nici prim, nici compus).
3. $k \leftarrow 1$
4. Execută :
 - $m \leftarrow$ primul număr mai mare decât k , care nu a fost marcat (primul va fi 2 !);
 - marchează numerele $2m, 3m, 4m, \dots$ din listă ca fiind compuse (mai întâi se vor marca toți multiplii lui 2, apoi multiplii lui 3 rămași etc.);
 - m este un număr prim, adaugă-l în lista de numere prime ;
 - $k \leftarrow m$.
- Până_când ($k \geq \sqrt{n}$).
5. Adaugă numerele rămase nemarcate în lista numerelor prime.
6. Afisează lista numerelor prime până la n .



Citiți programele corespunzătoare din capitolele 5, „Tablouri, pointeri, pointeri la funcții”, și 8, „Operații pe biți”.

eliminarea multiplilor lui 2	eliminarea multiplilor lui 3
(2) 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	(2) (3) 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
eliminarea multiplilor lui 5	numerele prime mai mici decât 49
(2) (3) 4 (5) 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	(2) (3) 4 (5) 6 (7) 8 9 10 11 12 (13) 14 15 16 (17) 18 (19) 20 21 22 (23) 24 25 26 27 28 (29) 30 31 32 33 34 35 36 (37) 38 39 40 41 42 (43) 44 45 46 (47) 48

Exemplul 4. Determinarea maximului a n elemente

Presupunem că ne dorim determinarea maximului din cadrul unei mulțimi date de numere. Algoritmul de determinare poate să fie următorul:

1. Citește numerele a_1, a_2, \dots, a_n
2. $\max \leftarrow a_1$
3. $i \leftarrow 2$
4. Cât_timp ($i \leq n$) execută
 - 4.1. Dacă ($\max < a_i$) $\max \leftarrow a_i$
 - 4.2. $i \leftarrow i + 1$
5. Sfârșit_cât_timp
5. Scrie \max

Observăm cum fiecare element din sirul de numere este comparat succesiv cu \max (maximul curent); \max va fi, de fapt, maximul dintre elementele subșirului a_1, \dots, a_i . Dacă maximul găsit până la poziția $i-1$ este mai mic decât elementul a_i curent, atunci acesta se reactualizează.

Exemplul 5. Căutarea liniară

Presupunând că se dă ca date de intrare un sir finit de numere a_1, a_2, \dots, a_n și un număr x ; să se determine prima poziție a numărului x în sir, dacă acesta există; altfel, să se returneze valoarea 0. Sirul numerelor date se va parcurge secvențial, cercetându-se dacă x este egal cu vreun element. Algoritmul are următoarea formă:

1. Citește x, a_1, a_2, \dots, a_n
2. $poz \leftarrow 0, i \leftarrow 1$
3. Cât_timp ($i \leq n$) execută
 - 3.1. Dacă ($x = a_i$) execută
 - 3.2. $poz \leftarrow i$
 - 3.3. $i \leftarrow n + 1$

- 3.4. Sfârșit_dacă
- 3.5. $i \leftarrow i + 1$
- Sfârșit_cât_timp
4. Scrie poz

Vom compara succesiv elementele din sir cu x . În momentul când găsim un număr a_i care este egal cu x , decidem că prima poziție la care se află x în sir este i (3.2) și vom întrerupe cercetarea celorlalte elemente (3.3), pentru că problema a fost complet rezolvată. Dacă s-ar dori determinarea *ultimului* element care este egal cu x , atunci vom sterge din algoritm doar linia 3.3, restul rămânând neschimbat. Încercați simularea algoritmului cu numerele 5, 7, 2, 1, 3, 7, 1, 6 și $x = 1$, prin scrierea succesivă a tuturor variabilelor folosite.

Exemplul 6. Căutarea binară

Presupunând că se dă ca date de intrare un sir finit ordonat crescător de elemente a_1, a_2, \dots, a_n și un element x , să se determine prima poziție lui x în sir, dacă acesta există, altfel să se returneze valoarea 0.

Informația că sirul de elemente date este deja ordonat ne conduce la determinarea unui algoritm prin micșorarea succesivă a intervalului de căutat (de exemplu, dacă ultimul element a_n este mai mic decât x , ne oprim deja după o primă comparație și afirmăm că x nu se află în sir). Vom vedea mai târziu de ce acest algoritm este în practică mai eficient decât unul în care s-ar efectua căutarea liniară, adică cercetarea tuturor elementelor.

Algoritmul de căutare binară:

1. Citește x, a_1, a_2, \dots, a_n
2. $i \leftarrow 1, j \leftarrow n$
3. Cât_timp ($i < j$) execută
 - 3.1. $m \leftarrow (i + j) / 2$
 - 3.2. Dacă ($x > a_m$), atunci $i \leftarrow m + 1$
 - 3.3. Altfel $j \leftarrow m$
4. Dacă ($x = a_i$), atunci $poz \leftarrow i$
5. Altfel $poz \leftarrow 0$
6. Afisează poz

Ilustrăm în continuare aplicarea algoritmului pentru sirul ordonat de 17 litere

a c d f g h j l m o p r s u v x z

în care ne propunem să căutăm litera j.

Pasul 3.2 din algoritm ne indică faptul că dacă numărul x căutat se situează la dreapta elementului din mijlocul intervalului cercetat (m este mijlocul intervalului $[i, j]$), atunci el va trebui căutat în intervalul $[m+1, j]$; altfel, se va efectua căutarea în intervalul $[i, m]$, capetele i , respectiv j modificându-se corespunzător.

$a_1, \dots, a_n = a c d f g h j l m o p r s u v x z$	$a c d f g h j l m o p r s u v x z$
$x = j$	j
$i \leftarrow 1, j \leftarrow 17$	
$a c d f g h j l m o p r s u v x z$	$a c d f g h j l m o p r s u v x z$
$i = 1, j = 17, m \leftarrow 9$ (3.2 sau 3.3?)	$i = 1, j = 9$ (pas 3.3), $m \leftarrow 5$ (3.2 sau 3.3?)
$a c d f g h j l m o p r s u v x z$	$a c d f g h j l m o p r s u v x z$
$i = 5, j = 9$ (pas 3.2), $m \leftarrow 7$ (3.2 sau 3.3?)	$i = 7, j = 9$ (pas 3.2), $m \leftarrow 8$ (3.2 sau 3.3?)
$a c d f g h j l m o p r s u v x z$	$a c d f g h j l m o p r s u v x z$
$i = 7, j = 8$ (pas 3.3), $m \leftarrow 7$ (3.2 sau 3.3?)	$i = 7, j = 7$ Stop

Exemplul 7. Algoritmul de preparare a prăjiturii Tiramisu

Ingrediente :



1 pachet de pișcoturi (cele pentru şampanie sau *Löffelbiskuits*), 500 g mascarpone (brânză specială italiană sau smântână mai groasă), 4 ouă, 150 g zahăr, 1 pahar mic (10 ml) de lichior Amaretto, 1 pachețel de pudră de gelatină, 3 ml de cacao, 1 cană de cafea îndulcită.

Rețetă (algoritm) :

- Start preparare.
- Răsturnați cafeaua într-o farfurie de supă.
- Înmulțiați pișcoturile în cafea, așezându-le câte puțin pe fiecare în farfurie cu cafea (ele trebuie să fie pe jumătate umede).
- Așezați pișcoturile unul lângă altul în tavă, astfel încât acestea să „tapeteze” tava.
- Separati ouăle.
- Amestecați gălbenușul, zahărul și lichiorul Amaretto cu mixerul până se obține o cremă uniformă.
- Adăugați în crema astfel obținută mascarponele împreună cu gelatina și amestecați în continuare.
- Bateți cu mixerul separat și albușurile până când spuma obținută devine fixă (prin întoarcerea bolului, aceasta nu cade).
- Răsturnați albușurile bătute în crema de gălbenușuri și amestecați totul cu mixerul.
- Acoperiți stratul de biscuiți uniform cu o jumătate din cremă. Peste stratul de cremă așezați încă un strat de biscuiți (înmulțiti în cafea) și peste acest strat distribuiți uniform restul de cremă.
- Așezați tava în frigider pentru cel puțin 3 ore (cel mai bine peste noapte).
- Înainte de servire se presără cacao pudră.
- Sfârșitul preparării. Poftă bună !

La prima vedere pare să nu existe nici o legătură între bucătărie și algoritmică, dar dacă încercăm să facem o paralelă, vom observa că lucrurile nu stau deloc aşa. Rețeta de preparare este de fapt un algoritm care, pe baza unor date de intrare (ingrediente), produce datele de ieșire cerute (prăjitura), folosind amestecuri intermediere („structuri de date”). Pașii se execută succesiv și ordinea de execuție a unor pași specifici este foarte importantă, în timp ce o serie de instrucțiuni (pași) se pot inter schimba. De exemplu, nu putem executa pasul 6 înaintea pasului 5, dar putem, în schimb, executa pasul 7 înaintea pasului 5. Observăm cum ingrediente sunt combinate și transformate pentru a ajunge la îndeplinirea obiectivului final, desertul. Se poate descrie și un alt algoritm care va conduce la același rezultat (de exemplu, dacă biscuiții sunt îmbibați succesiv cu cafea cu ajutorul unei lingurițe, după ce au fost așezăți în tavă). Ca un exercițiu de imaginație, putem chiar să ne gândim și la construirea unei mașini care să prepare automat Tiramisu.

4. De la problemă spre soluție

Problemele din sfera informaticii actuale, indiferent dacă sunt de dimensiuni reduse sau adevărate produse-gigant, au la bază o serie de idei și caracteristici comune. Comportarea internă în cadrul ciclului de viață al unui program se bazează întotdeauna pe două elemente primordiale, reprezentarea și transformarea :

- *reprezentarea* se referă la o codificare concretă a informației (cum ar fi biți, numere, cuvinte, înregistrări);
- *transformarea* se referă la modificarea succesivă a stării prin derularea unor pași (rețetă, program, algoritm) în scopul găsirii rezultatelor așteptate.

Codificarea simbolică a informației (*reprezentarea*) furnizează o modalitate de comunicare între lumea reală a problemei și lumea imaginară a computerului. Se va crea astfel o „mapare”, funcție cu ajutorul căreia entitățile reprezentative din cadrul problemei de rezolvat sunt codificate prin valori înțelese de calculator. Această etapă de codificare este foarte importantă, deoarece de ea depind corectitudinea soluției, complexitatea structurilor alese și a algoritmului.

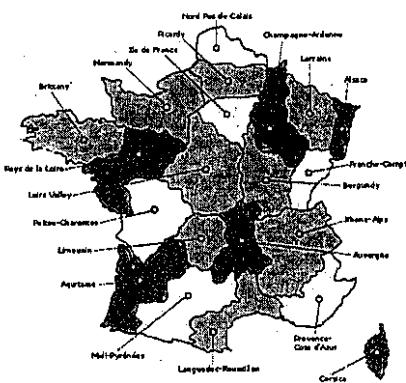
Rezolvarea unei probleme de informatică presupune construirea a trei procese ce se întrepătrund și se execută secvențial :

- găsirea unei reprezentări simbolice a informației, astfel încât să poată fi manipulate ulterior ;
- formularea algoritmului care, utilizând aceste reprezentări, va conduce la rezultatele dorite ;
- construirea cadrului specific care permite ca acești algoritmi să fie implementați într-o manieră corespunzătoare (de exemplu, scrierea codului într-un limbaj de programare).

Exemplu 1. Problema colorării hărții (Map Colouring Problem)

Dată fiind o hartă cu țări, ne propunem să găsim o modalitate de a colora fiecare țară astfel încât oricare două țări vecine să fie colorate diferit (culorile se vor alege dintr-o paletă dată de culori).

De exemplu, harta Franței va trebui desenată astfel încât oricare două regiuni vecine să fie colorate diferit folosind culorile roșu, galben, verde și albastru. Problema apare în



instrumente matematice, prelucrabile cu ajutorul computerului. Soluția ar trebui să fie un vector de „culori” (numere) $S = \{c_1, c_2, \dots, c_n\}$, $c_i \in \{1, \dots, m\}$, cu proprietatea că dacă $A(i, j) = 1$ atunci $c_i \neq c_j$. Odată stabilită această reprezentare simbolică a informației reale, va trebui să trecem la elaborarea unui algoritm care, pe baza matricei A și a numărului de culori m , să ne furnizeze o soluție S . De fapt, reprezentarea problemei este un graf neorientat cu matricea de adiacență A , în care va trebui să colorăm vârfurile, astfel încât oricare două vârfuri vecine să fie colorate diferit.

 8 regiuni, 3 culori <i>Se cere determinarea unei colorări a regiunilor utilizând cele trei culori, astfel încât oricare două regiuni vecine să fie colorate diferit.</i>	
$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$	1. Construim graful atașat hărții și matricea de adiacență în care $A(i, j) = 1$, dacă și numai dacă $i \neq j$ și regiunile i, j sunt vecine. 2. Observăm că matricea de adiacență este simetrică: $A(i, j) = A(j, i)$, $\forall i, j \in \{1, \dots, n\}$. 3. Culorile posibile vor fi c_1, \dots, c_m . 4. Soluția va trebui să fie de forma $S = \{c_1, c_2, \dots, c_n\}$, $c_i \in \{1, \dots, m\}$, cu semnificația că c_i este culoarea atașată vârfului i al grafului.

Exemplul 2. Varianta a problemei orarului (Schedule Problem)

Concret, presupunem că dorim să distribuim sălii, profesorii, grupele de studenți pentru a realiza orarul unei facultăți. Codificarea informației practice este mult mai dificilă în acest caz decât cea din exemplul anterior, această „mapare” nu se mai produce atât de natural și accesibil. Va trebui să codificăm cu numere de la 1 la n_p profesorii, cu numere de la 1 la n_s , toate grupele de studenți (plus fiecare student și apartenența sa la grupe), la fel locațiile (săli, amfiteatre, laboratoare etc.), cursurile posibile. Vor trebui stabilite diverse funcții (caracteristici) care trebuie să fie satisfăcute de o eventuală soluție (fiecare curs are o durată și se poate desfășura doar în anumite săli, o grupă are doar anumite cursuri, depinzând de an și specializare, un profesor predă doar anumite cursuri cu anumite grupe etc.) și acestea trebuie, de asemenea, accesibili codificate. Vor trebui definite și o serie de restricții, astfel încât o posibilă soluție să poată fi verificată matematic (un profesor sau un student nu poate fi în două locuri simultan, într-o anumită sală nu se pot suprapune activități diferite etc.). Numai când definirea în termeni matematici a acestor entități, caracteristici și forma soluției sunt complet realizate se poate trece la elaborarea unui algoritm pentru a găsi posibilele soluții.

5. Proprietățile unui algoritm

- Trebuie să posede date de intrare (*input data*).
- Trebuie să posede date de ieșire (*output data*).
- *Determinismul* (la executarea oricărui pas, trebuie să cunoaștem successorul acestuia).
- *Corectitudinea* datelor de ieșire (a rezultatelor) în raport cu datele de intrare.
- *Finitudinea* (pentru orice set de date de intrare posibile ale problemei, soluția este furnizată într-un număr finit de pași).
- *Eficiența* (furnizarea soluției trebuie să fie realizabilă prin consumul eficient al resurselor timp, spațiu etc.).
- *Generalitatea* (algoritmul este aplicabil unei clase de probleme, cele ale căror date de intrare satisfac anumite condiții).

6. Algoritmica

Algoritmica este ramura informaticii care se ocupă cu proiectarea și analizarea algoritmilor destinați a fi implementați cu ajutorul computerului.

1. Prin *proiectare (design)* înțelegem două etape:
 - descrierea algoritmului utilizând un pseudolimbaj (schemă logică, pseudocod, descriere cuprinzătoare sugestivă pe o foaie de hârtie etc.);
 - demonstrarea corectitudinii, adică a faptului că indiferent de datele de intrare posibile introduse, rezultatul satisfac cerința problemei.
2. Analiza algoritmului se referă la evaluarea performanței acestuia, adică a timpului necesar pentru a determina soluția, dar și a altor aspecte, cum ar fi spațiul de memorie folosit și optimalitatea.

Începutul trebuie să fie alegerea sau definirea unui model de calcul, adică a unui formalism matematic care descrie interacția dintre componentele sistemului și furnizează abstractizări utile pentru concepe precum timp sau concurență. De obicei, modelul folosit este *RAM* (*Random Access Machine*), dar există și alte modele de calcul precum: *PRAM*, *Mașini Turing*, automate finite, circuite booleene, automate celulare etc. Odată modelul definit, algoritmul poate fi descris folosind un limbaj simplu, asemănător ca sintaxă, de exemplu Basic, Pascal, C, C++, Java etc.

7. Modelul de calcul RAM

Proiectarea unui algoritm independent de mașină depinde de un computer ipotetic numit *Random Access Machine* (*RAM*). Potrivit acestui model, computerul ipotetic satisfac:

- Fiecare operație „simplă” (+, -, *, /, %, =, if/dacă, call/apeleză) consumă o unitate de timp.
- Ciclurile și metodele (funcții, proceduri) nu sunt considerate operații simple. Acestea sunt compuse din mai multe operații de un singur pas. Nu ar avea sens să atribuim metodei *sortare*, de exemplu, o unitate de timp, când sortarea unui milion de elemente va consuma mult mai mult timp decât sortarea a 10 elemente. Timpul necesar pentru a executa un ciclu sau de a executa o metodă (un subprogram) depinde de numărul de iterații, respectiv de natura metodei.
- Fiecare acces la memorie se execută într-o unitate de timp și avem la dispoziție atâta memorie câtă este necesară. Modelul RAM nu face diferență dacă anumite date sunt pe disc sau în *cache* (memoria volatilă), ceea ce simplifică analiza algoritmului.

Conform cu modelul RAM, se va calcula numărul de pași (unități de timp) necesari pentru instanța unei probleme. Presupunând că RAM execută un anumit număr de pași pe secundă, se poate ușor determina timpul real necesar. Una dintre nemulțumiri care poate fi faptul că modelul este mult prea simplu, că unele presupuneri sunt prea generale pentru a fi aplicate în practică. De exemplu, pentru înmulțirea a două numere este nevoie de obicei de mai mult timp decât pentru adunarea lor, ceea ce contrazice prima presupunere a modelului. Timpul de acces la memorie diferă de obicei relativ la locul de stocare a datelor, în *cache* (memoria volatilă) sau pe disc, ceea ce contrazice cea de a treia presupunere. Dar chiar și cu aceste neajunsuri, modelul RAM este un instrument excelent pentru a înțelege cum se comportă un algoritm în realitate, el reflectând comportarea computerului, dar rămânând destul de simplu pentru a putea fi manipulat și înțeles.

Fiecare model are un interval-limită în care este util. Să considerăm ca exemplu *modelul pământului plat*. Se poate afirma că acesta este un model incorrect, pentru că este dovedit faptul că pământul este rotund. Dar pentru a construi fundația unei case, modelul pământului plat este util și poate fi utilizat ca atare. Este mult mai ușor de lucrat practic cu un astfel de model decât cu un model sferic. Tot astfel se întâmplă și cu modelul de calcul RAM. El este de fapt o abstractizare care, în general, devine foarte utilă pentru a explica în practică performanța unui algoritm, într-un mod independent de mașină.



8. Complexitatea algoritmilor

Prin *complexitatea* unui algoritm înțelegem de fapt *costul*, măsurat cu ajutorul unor anumiti parametri (timp de execuție, memoria necesară, numărul anumitor operații etc.). Pentru a calcula complexitatea unui algoritm, avem nevoie să decidem care sunt acești parametri și să găsim o funcție de determinare a costului corespunzătoare.

Dacă fiind că resursele de memorie sunt, în practică, foarte mari, deducem că timpul de execuție al unui algoritm este parametrul de bază. Complexitatea-timp poate fi calculată în funcție de numărul de operații elementare (unități de timp) necesare atunci când datele de intrare au o anumită dimensiune n . Cum am văzut mai sus, la descrierea modelului RAM, aceste operații de bază pot fi comparații (număr foarte mare, de exemplu, în cazul unui algoritm de căutare), asignări (de exemplu, în cazul unui algoritm de sortare), adunări, înmulțiri, diviziuni, modulo, apeluri de metodă etc.

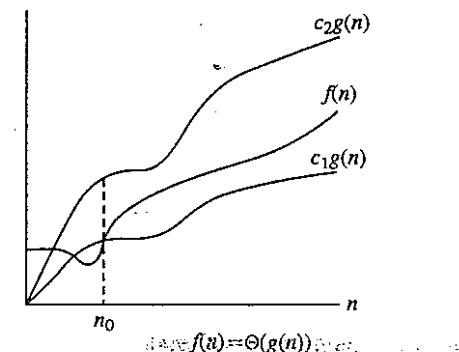


8.1. Notațiile Θ , O și Ω

În practică sunt folosite o serie de notații utile pentru analiza performanței și a complexității unui algoritm. Acestea mărginesc valorile unei funcții f date cu ajutorul unor constante și a altrei funcții. Prezentăm în continuare cele trei notații cunoscute în acest sens.

8.1.1. Notația Θ (categorie constantă - same order)

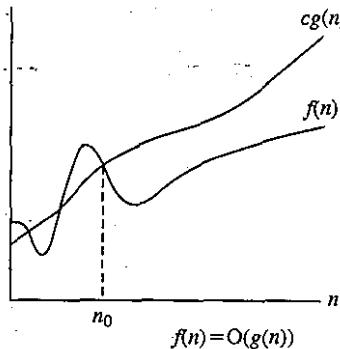
Spunem că $f(n) = \Theta(g(n))$ dacă există constantele pozitive n_0 , c_1 și c_2 , astfel încât pentru toate numerele naturale n de la dreapta lui n_0 , valoarea lui $f(n)$ se află întotdeauna între $c_1 \cdot g(n)$ și $c_2 \cdot g(n)$ inclusiv.



Notația Θ : $\exists n_0, c_1, c_2$, astfel încât $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$, $\forall n \geq n_0$.

8.1.2. Notația O (mărginire superioară – upper bound)

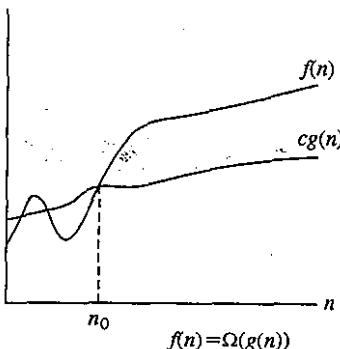
Spunem că $f(n) = O(g(n))$ dacă există o constantă pozitivă n_0 astfel încât pentru toate numerele n de la dreapta lui n_0 , valoarea lui $f(n)$ se află întotdeauna sub $c \cdot g(n)$.



Notația O : $\exists n_0, c$, astfel încât $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$.

8.1.3. Notația Ω (marginie inferioară – lower bound)

Spunem că $f(n) = \Omega(g(n))$ dacă există constantele pozitive n_0 și c astfel încât pentru toate numerele n mai mari decât n_0 , valoarea lui $f(n)$ este mai mare decât $c \cdot g(n)$.



Notația Ω : $\exists n_0, c$, astfel încât $f(n) \geq cg(n)$, $\forall n \geq n_0$.

8.2. Optimalitate, reducere, exemple

Complexitatea unui algoritm este deci o funcție $g(n)$ care limitează superior numărul de operații (la execuție) necesare pentru dimensiunea n a problemei. Există două interpretări ale limitei superioare:

- complexitatea în cazul cel mai defavorabil (*worst-case complexity*): timpul de execuție pentru orice dimensiune dată va fi mai mic decât limita superioară, exceptând câteva dimensiuni ale problemei, unde este atins un maxim;
- timpul de execuție pentru orice dimensiune dată va fi media numărului de operații pentru toate instanțele posibile ale problemei.

Deoarece este dificil să se estimeze o comportare statistică ce depinde de dimensiunea intrării, de cele mai multe ori este folosită prima interpretare, cea a cazului celui mai defavorabil. De cele mai multe ori, complexitatea lui $f(n)$ este aproximată de familia sa $O(g(n))$, unde $g(n)$ este una dintre funcțiile: n (complexitate liniară), $\log(n)$ (complexitate logaritmică), n^a , $a \geq 2$ (complexitate polinomială), a^n (complexitate exponențială), $n!$ (complexitate factorială). O altă definiție sugestivă echivalentă este cea folosind limitele din analiza matematică: dacă limita pentru n tinde la infinit din $f(n)/g(n)$, există și este finită, atunci $f(n)$ are ordinul $O(g(n))$:

Exemplu :

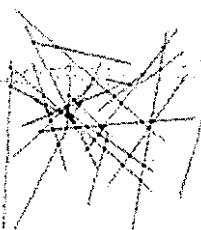
- $\lim (\sqrt{n}/n) = 0 \rightarrow \sqrt{n} = O(n)$
- $\lim (n/\sqrt{n}) = \text{infinit} \rightarrow n \text{ nu este } O(\sqrt{n})$
- $\lim (n/2n) = \frac{1}{2} \rightarrow n = O(2n)$
- $\lim (2n/n) = 2 \rightarrow 2n = O(n)$

Pentru n suficient de mare au loc inegalitățile:

$\log(n) < n < n \cdot \log(n) < n^2 < n^3 < 2^n$, ceea ce implică $O(\log(n)) < O(n) < O(n \log(n)) < O(n^2) < O(n^3) < O(2^n)$.

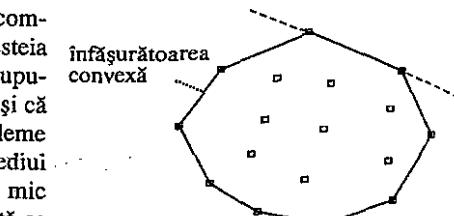
Exemplu :

$$10n + 5 \log(n) + 8 \in O(n), 8n \in O(n^2), 65 \in O(1), n^{1000} \in O(2^n).$$



Odată determinată complexitatea unui algoritm, apare problema stabilirii dacă acest algoritm este **optimal**. Un algoritm pentru o problemă dată este optimal atunci când complexitatea sa atinge o limită inferioară a tuturor algoritmilor care rezolvă această problemă. De exemplu, orice algoritm care rezolvă problema „intersecției a n segmente” va executa cel puțin n^2 operații în cazul cel mai nefavorabil și există un algoritm cu această complexitate, deci vom afirma că problema are o complexitate $\Omega(n^2)$.

Reducerea este o altă tehnică de a estima complexitatea unei probleme prin transformarea acesteia într-o problemă echivalentă. De exemplu, presupunem că știm limita inferioară a unei probleme A și că ne dorim estimarea limitei inferioare a unei probleme B. Dacă putem transforma A în B prin intermediul unui pas de transformare al cărui cost este mai mic decât rezolvarea lui A, atunci B are aceeași limită ca și A. Ca un exemplu, putem considera problema



determinării înfășurătorii convexe, care se reduce la sortarea punctelor date (complexitate: $\Theta(n \log(n))$).

Exemplu:

$$\Theta(n) + \Theta(\log(n)) = \Theta(n) \quad (\log(n) < n), \quad \Theta(n!) + \Theta(n^3) = \Theta(n!) \\ (n^3 < n!)$$

Exemplul 1 de analiză a complexității. Considerăm următorul algoritm:

1. Citește n
2. $diff \leftarrow 0$, $sum \leftarrow 0$, $k \leftarrow 0$
3. Cât_timp ($k < n$) execută


```
        sum   ← sum + 1
        diff ← diff + 1
        k    ← k + 1
```

 Sfărșit_cât_timp
4. $k \leftarrow 0$
5. Cât_timp ($k < 3n$) execută


```
        sum   ← sum - 1
        k    ← k + 1
```

 Sfărșit_cât_timp
6. Scrie sum , $diff$

Analiza complexității:

Pasul 2 are nevoie de 3 unități de bază. Ciclul de la pasul 3 va executa $2n$ adunări, n scăderi și $3n$ atribuiri, deci în total $6n$ unități de timp.

Pasul 4 are nevoie de o unitate. Pasul 5 face $2 \cdot 3n$ atribuiri, $3n$ scăderi și $3n$ adunări, deci în total $12n$ unități de bază.

$$\text{Total: } 3 + 6n + 1 + 12n = 18n + 4$$

Aceasta înseamnă $\Theta(n)$.

Exemplul 2. Analiza complexității și a eficienței. Vom face o analiză comparativă a trei algoritmi care rezolvă aceeași problemă: calcului sumei $1 + 2 + \dots + n$. În acest scop, putem scrie următorii algoritmi :

Analiza eficienței algoritmilor

Algoritm A	Algoritm B	Algoritm C
<pre>sum ← 0 pentru (i ← 1, n) sum ← sum + i</pre>	<pre>sum ← .0 pentru (i ← 1, n) pentru (j ← 1, i) sum ← sum + 1</pre>	<pre>sum ← n*(n+1) / 2</pre>
$n+1$ asignări, n atribuiri	$1+n(n+1)/2$ asignări $n(n+1)/2$ atribuiri	1 adunare, 1 înmulțire, 1 diviziune, 1 atribuire
Total: $2n + 1$	Total: $n^2 + n + 1$	Total: 4

Deducem că cel mai performant algoritm este C (cunoștințele de matematică sunt importante în conceperea algoritmilor!), fiind și algoritmul optimal pentru afilarea acestei sume, iar cel mai ineficient este B.

9. Exemplu comparativ de creștere a lui $O(g(n))$

Rata comparativă de creștere a funcției $O(g(n))$

n	$\log(\log n)$	$\log n$	$(\log n)^2$	N	$n \log n$	n^2	2^n	$n!$
10	2	3	11	10	33	10^2	10^3	10^5
10^2	3	7	44	100	664	10^4	10^{30}	10^{94}
10^3	3	10	99	1.000	9.966	10^6	10^{301}	$10^{1.435}$
10^4	4	13	177	10.000	132.877	10^8	$10^{3.010}$	$10^{19.335}$
10^5	4	17	276	100.000	1.660.964	10^{10}	$10^{30.103}$	$10^{243.338}$
10^6	4	20	397	1.000.000	19.931.569	10^{12}	$10^{301.030}$	$10^{2.933.369}$

Alte complexități:

1. Se poate demonstra că *algoritmul lui Euclid* are complexitatea $O((\log a)(\log b))$.
2. Complexitatea pentru *căutarea liniară* este $O(n)$, deoarece în cel mai nefavorabil caz trebuie să parcuregem toată secvența de numere, deci complexitatea este liniară.
3. Pentru *căutarea binară*, complexitatea este $O(\log n)$; în cazul cel mai nefavorabil avem nevoie de $(1 + \log n)$ operații pentru a găsi elementul x .
4. Pentru *problema colorării hărților*, unde avem un graf cu n vârfuri și trebuie atribuite m culori, în cazul cel mai nefavorabil vor trebui generate toate combinațiile (căutare exhaustivă), care sunt m^n (numărul funcțiilor definite pe o mulțime cu n elemente într-o mulțime cu m elemente), deci complexitatea este exponențială: $O(m^n)$.

10. Timpul real necesar unui algoritm (polinomial vs exponențial)

Următoare tabelă arată cum se modifică timpul real necesar sistemului de calcul atunci când trebuie executată $T(n)$ pași simpli pe un computer obișnuit cu 1 bilion pași/secundă. O microsecundă este o milionime dintr-o secundă, iar o milisecundă este o miile dintr-o secundă.

Analiza comparativă a timpului real necesar (algoritm polinomial versus algoritm exponențial)

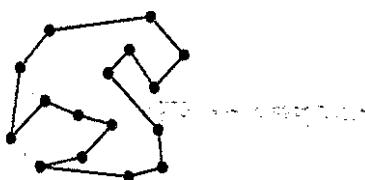
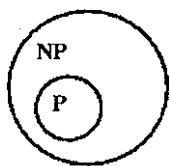
n	$T(n) = n$	$T(n) = n \lg(n)$	$T(n) = n^2$	$T(n) = n^3$	$T(n) = 2^n$
5	0,005 microsec	0,01 microsec	0,03 microsec	0,13 microsec	0,03 microsec
10	0,1 microsec	0,03 microsec	0,1 microsec	1 microsec	1 microsec
20	0,02 microsec	0,09 microsec	0,4 microsec	8 microsec	1 milisec
50	0,05 microsec	0,28 microsec	2,5 microsec	125 microsec	13 zile
100	0,1 microsec	0,66 microsec	10 microsec	1 milisec	4×10^{13} ani

Remarcăm că pentru $n \geq 50$, timpul de calcul pentru $T(n) = 2^n$ devine prea mare pentru a fi și practic. Chiar dacă se mărește de un million de ori viteza sistemului, pentru $n = 100$ tot vor fi nevoie de 40.000.000 de ani pentru a afla răspunsul.

11. Clasificarea problemelor (P, NP, NP-complete, NP-hard)

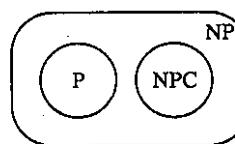
Pe lângă determinarea complexității-timp, ne putem referi și la clasificarea problemelor relativ la faptul dacă ele sunt „greu” sau „ușor” de rezolvat (accesibile sau inaccesibile). Această schemă de clasificare include clasele P și NP, iar „NP-complete” și „NP-hard” sunt subclase ale clasei NP. În vederea stabilirii unor caracteristici formale relativ la mulțimea problemelor posibile, a fost mai întâi definită clasa *problemelor de decizie*. Această clasă conține toate problemele a căror soluție poate să fie „Da” sau „Nu”. De exemplu, problema deciziei conexității unui graf unidirecțional (adică dacă între oricare două vîrfuri ale grafului există un drum). Datele de intrare sunt vîrfurile și muchiile grafului G (elemente care determină graful), iar întrebarea se pune astfel „Este graful G conex?”. De obicei, cele mai multe probleme de optimizare nu sunt probleme de decizie, dar se pot transforma ușor în probleme de decizie. De exemplu, una dintre versiunile problemei comis-voiajorului (*TSP – Travelling Salesman Problem*) are ca date de intrare graful G cu costurile muchiilor și un număr K. Întrebarea asociată este: „G conține un tur de lungime mai mică sau egală cu K?”. De obicei, o problemă de optimizare nu este mult mai greu de rezolvat decât problema de decizie corespunzătoare.

Clasa P se definește ca fiind mulțimea problemelor de decizie pentru care există un algoritm în timp polinomial de rezolvare, adică acele probleme care sunt considerate formal „ușoare” (accesibile). Mai largă clasă de probleme NP conține clasa P și se referă la probleme cu algoritmi de rezolvare „nedeterministic-polinomiali”. Clasa NP conține acele probleme de decizie care au următoarea proprietate: pentru orice instanță „da” a problemei rezultatul se poate verifica folosind un algoritm de tip polinomial. De exemplu, dacă considerăm problema: „Sunt grafurile G și G' izomorfe?”, „instanța da” a acestei probleme este o instanță în care grafurile G și G' sunt izomorfe. O demonstrație a acestei instanțe va fi un izomorfism (o mapare 1-1) f de la G la G' . Demonstrația răspunsului presupune, de fapt, verificarea faptului că f este într-adevăr un izomorfism de la G la G' , adică pentru orice pereche de muchii $\{u, v\}$ din G , $\{f(u), f(v)\}$ este muchie în G' și G' nu are alte muchii în afara de cele verificate. Această demonstrație se poate executa în timp polinomial relativ la dimensiunea grafurilor G și G' , deci această problemă aparține clasei NP. Un alt exemplu este și problema comis-voiajorului prezentată anterior în varianta de decizie, unde instanța este un graf G și un număr $K = 100$. Dacă răspunsul este „da”, atunci există o listă de muchii de lungime cel mult 100 care este un circuit; această listă acționează ca un „certificat” și se poate demonstra în timp polinomial că este o soluție validă și suficient pentru a decide că răspunsul este „da”. Nu există simetrii între cele două variante „da” și „nu”. Dacă pentru a ne convinge că răspunsul este da e suficient să verificăm în timp polinomial o variantă de circuit, în cazul răspunsului „nu” totul devine mult mai complicat (ar trebui să verificăm toate variantele posibile, ceea ce este extrem de costisitor,



practic imposibil). De fapt, prin schimbarea rolurilor jucate de „da” și „nu” se obține o clasă de probleme numită Co-NP (de exemplu, pentru problema comis-voiajorului: „toate circuitele posibile ale comis-voiajorului în G au lungimea mai mare decât K ?”). Există și multe probleme care par să se situeze în afara claselor NP și Co-NP pentru că ele nu posedă o demonstrație în timp polinomial („certificat”). Un exemplu poate fi problema cu datele de intrare graful G și numerele K și L , cu întrebarea: „Este numărul circuitelor distincte ale comis-voiajorului de lungime cel mult K egal cu L ?“.

12. Probleme NP-complete



Până acum nu s-a găsit un algoritm polinomial pentru rezolvarea problemei comis-voiajorului. Pe de altă parte, nimeni nu a fost capabil să demonstreze că un astfel de algoritm nu există. Deci se poate afirma că problema comis-voiajorului, ca și multe alte probleme, este „intractabilă” (inaccesibilă). Se poate demonstra că varianta de decizie a problemei comis-voiajorului, ca și alte multe probleme din clasa NP, este una dintre cele mai *grele* probleme NP, în sensul: dacă există un algoritm în timp polinomial pentru *una* dintre aceste probleme, atunci există un algoritm polinomial pentru *toate* problemele din NP. Observăm că aceasta este o afirmație foarte puternică, de vreme ce NP conține un număr imens de probleme care par să fie extrem de dificil de rezolvat, atât teoretic, cât și practic! Problemele din NP cu acestă proprietate se numesc NP-complete.

Alte exemple de probleme NP-complete:

- problema determinării dacă două grafuri date sunt izomorfe;
- problema colorării grafurilor (*Graph Coloring Problem*): date fiind un graf G și un număr natural $k \geq 1$, există o k -colorare a lui G ? k -colorare înseamnă posibilitatea de a se eticheta vîrfurile grafului folosind culorile $1, \dots, k$, astfel încât oricare două vîrfuri vecine să fie colorate diferit;
- problema orarului (*Schedule Problem*): decizia existenței unui orar pentru un set de obiective, resurse și caracteristici ale unor activități;
- problema împachetării (*Bin Packing Problem*): date fiind o mulțime de obiecte cu dimensiunile lor și o mulțime de lăzi, decizia dacă obiectele pot fi împachetate în lăzi;
- problema partilionării (*Partition Problem*) unei mulțimi de numere întregi în două submulțimi, astfel încât suma numerelor din primul grup este egală cu suma numerelor din al doilea grup;
- problema acoperirii mulțimilor (*Set Cover Problem*): date fiind o mulțime S , o colecție de submulțimi ale lui S și un număr natural k , să se decidă dacă există posibilitatea de a scrie pe S ca reuniune de k sau mai puține submulțimi din colecție;
- problema rucsacului (*Knapsack Problem*): date fiind un rucsac de dimensiune s , o mulțime de obiecte cu dimensiunile și greutățile lor și un număr întreg v , să se decidă dacă există o mulțime de obiecte pentru care suma dimensiunilor nu depășește s și suma greutăților este v sau mai mare.

13. Problema satisfiabilității (SAT)

Clasa NP și noțiunea de probleme „complete” din NP au fost prima dată introduse de un articol al lui Cook (1971). În acel articol, el a demonstrat că o problemă particulară de decizie din logică, problema satisfiabilității (SAT), este NP-completă, dar și că orice altă problemă din NP poate fi codificată ca fiind un caz special al problemei SAT. Odată rezolvată o problemă NP-completă, orice problemă NP-completă devine rezolvabilă, prin furnizarea unei transformări polinomiale între cele două. Putem deci afirma că SAT este una dintre cele mai importante probleme din teoria complexității.

Considerându-se o formulă logică în forma normală conjunctivă (CNF), să se decidă dacă există o atribuire a variabilelor astfel încât evaluarea formulei să aibă valoarea „adevărat”. O formulă logică este în formă normală conjunctivă (CNF) dacă are forma:

$(y_{11} \vee y_{12} \vee \dots \vee y_{1n_1}) \wedge (y_{21} \vee y_{22} \vee \dots \vee y_{2n_2}) \wedge \dots \wedge (y_{m1} \vee y_{m2} \vee \dots \vee y_{mn_m})$, unde y_{ij} sunt valori booleene din mulțimea de variabile $\{x_1, x_2, \dots, x_n, \bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$.

Exemplu : Presupunem următoarea formulă în CNF:

$$F = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_3)$$

Problema este de a decide dacă există o atribuire a variabilelor x_1, x_2, x_3 astfel încât valoarea de adevară a lui F corespunzătoare ei să fie 1. Observăm că în acest caz răspunsul este da, existând chiar două astfel de atribuiriri.

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

14. Clasa problemelor NP-hard

Problemele din clasa NP-hard sunt cel puțin la fel de grele ca și orice problemă din NP. În particular, problemele de optimizare pentru care problema de decizie corespunzătoare este NP-completă (de exemplu, problema comis-voiajorului) sunt NP-hard, deoarece versiunea de optimizare este cel puțin tot atât de dificilă ca și versiunea de decizie. Am văzut și mai sus că timpul real necesar găsirii unei soluții prin căutare exhaustivă, de

exemplu, este inacceptabil, de aceea găsirea unei soluții optime este imposibilă. În ultimii ani, au apărut domenii ale informaticii care încearcă găsirea unor soluții „cât mai optime”: algoritmii evolutivi, inteligența artificială, sistemele fuzzy, toate inspirate de natură.

15. Exerciții și probleme propuse

1. Scrieți un referat despre viața și contribuțiile lui Eratostene (*Eratostenes*) sau Euclid, folosind resurse Internet.
2. De unde provine cuvântul *algoritm*? Dați câteva definiții alternative.
3. Scrieți formal *algoritmul lui Euclid* și transformarea variabilelor pentru a afla cel mai mare divizor comun al numerelor 2.322 și 654.
4. Scrieți formal algoritmul *Sita lui Eratostene* și aplicați-l practic pentru a afla numerele prime mai mici decât 30.
5. Scrieți formal și algoritmul de înmulțire a numerelor întregi.
6. Construiți formal algoritmul de determinare simultană a minimului și maximului dintr-o secvență de elemente. Care este complexitatea acestuia?
7. Scrieți algoritmii de căutare liniară, respectiv binară. Ce complexități au? Care este mai performant? Cum se explică acest lucru?
8. De ce putem afirma că o rețetă culinară este tot un algoritm?
9. Ce sunt *reprzentarea și transformarea* în cadrul ciclului de viață al unui program? Cum se codifică *problema colorării hărților*?
10. Care sunt proprietățile unui algoritm?
11. Ce înțelegem prin *proiectarea*, respectiv *analiza* algoritmului?
12. Ce se înțelege prin model de calcul? Dați exemple de modele de calcul. Descrieți proprietățile și avantajele modelului RAM.
13. Ce este complexitatea unui algoritm? Descrieți formal și grafic notațiile Θ , O și Ω .
14. Ce înțelegem prin *reducerea*, respectiv *optimalitatea* unui algoritm?
15. Cum se clasifică problemele relativ la complexitate? Dați exemple de probleme NP-complete.
16. Descrieți problema SAT. Dați și un exemplu sugestiv. De ce este ea considerată cea mai importantă problemă NP-completă?
17. Care este clasa problemelor NP-hard?
18. Care dintre următoarele afirmații sunt adevărate?
 - A. $n^2 \in O(n^3)$; B. $n^3 \in O(n^2)$; C. $2^{n+1} \in O(2^n)$; D. $(n+1)! \in O(n!)$.
19. Determinați ordinul următorilor algoritmi:

<pre> pentru i ← 1, n executa pentru j ← 1, 5 executa (operatie elementara) pentru i ← 1, n executa pentru j ← 1, 6 executa pentru k ← 1, n executa (operatie elementara) </pre>	<pre> pentru i ← 1, n executa pentru j ← 1, i+1 executa (operatie elementara) pentru i ← 1, n executa pentru j ← 1, i+1 executa pentru k ← 1, n executa (operatie elementara) </pre>

Limbajul C – prezentare generală

- Scurt istoric al limbajului C
- Construcții de bază: setul de caractere, identificatori, cuvinte-cheie, tipuri de date, constante, variabile, comentarii
- Expresii: operanți și operatori
- Structura programelor C: directive de preprocessare, funcții
- Instrucțiuni: vidă, expresie, compusă, if, while, for, do-while, switch, break, continue
- Intrări/ieșiri standard: funcții de scriere și citire cu format
- Fișiere: deschiderea și închiderea, verificarea sfârșitului de fișier, funcții de citire scriere
- **struct, enum, union, typedef**
- Tablouri: structură, exemple
- Pointeri: importanță, legătura cu tablourile, structurile, argumente în funcții, greșeli uzuale, indicații
- Siruri de caractere: construcție, exemple, legătura cu tablourile, metode din biblioteca **string.h**
- 11 probleme sugestive complet rezolvate, cu explicații detaliate la nivel de limbaj C (afișarea textului, afișarea/citirea valorilor simple, interschimbarea conținutului variabilelor, suma cifrelor, calculator electronic, testul numărului prim, minim/maxim într-un tablou, analiza unui sir de caractere, afișarea conținutului unui fișier, o problemă cu o matrice de pointeri, numere aleatorii)

Cine învață o nouă limbă câștigă un nou suflet.

Juan Ramón Jiménez

Limbajul C este recunoscut ca fiind unul dintre cele mai flexibile, rapide, robuste și puternice limbaje, ce permite programatorului o interacțiune eficientă cu diferite platforme și sisteme de operare. Robustetea vine atât de la bibliotecile sale, cât și de la elementele de limbaj. Pe de altă parte, C permite manipularea la nivel de bit și interpretarea flexibilă a conținutului memoriei și a variabilelor folosind pointeri și asociații de tipuri. Programatorul are foarte puține restricții comparativ cu alte limbaje, deci libertate, dar poartă și răspunderea pentru deciziile sale. În mare parte, această libertate vine de la neverificarea tuturor tipurilor la compilare, avantaj puternic pentru programatorii experimentați, dar dezavantaj periculos pentru cei începători.

1. Iстория языка C

Основа языка C была заложена в 1972 году в Bell Laboratories Деннисом Ричи и Кеном Томпсоном. Многие принципы и идеи языка были заимствованы из предшествующих языков: ALGOL, BCPL, CPL, Algol. Краткая история языка в этом смысле:

- ALGOL (конец 1950-х годов, один из первых структурированных языков);
- CPL (*Combined Programming Language*);
- BCPL (Martin Richards);
- B (1970, Ken Thompson, Bell Labs);
- C (Dennis Ritchie, Bell Labs).

Ричи и Томпсон, вместе с коллегой Брайаном Кернигем, разработали языок UNIX, используя компилятор C Ричи. С тех пор язык C стал основой для языков объектного типа C++ и Java.

2. Construcții de bază

2.1. Setul de caractere

În cadrul programelor C se folosesc setul de caractere al codului ASCII. Acestea sunt codificate prin întregi din intervalul [1, 127]. Un astfel de întreg poate fi păstrat în binar pe 8 biți (un octet). Mulțimea caracterelor poate fi împărțită în trei grupe: caractere negrafice (exceptând caracterul DEL, au codurile ASCII mai mici decât 32), spațiu, caractere grafice (acestea sunt literele mari, literalele mici și caracterele speciale și au codurile ASCII mai mari decât 32).

De exemplu, literalele mari au codurile ASCII în intervalul [65, 90], valoarea 65 fiind atribuită literiei A, iar valoarea 90, literiei Z. Literalele mici au codurile ASCII în intervalul [97, 122], iar cifrele 0-9 se codifică prin numere de la 48 la 57.

Codul ASCII cu valoarea 0 corespunde caracterului NULL, care este un caracter impropriu, nu poate fi generat de la tastatură și nu poate fi tipărit. Acest caracter este utilizat pentru a termina un sir arbitrar de caractere.

Caracterele negrafice au diferite funcții: de exemplu, caracterul cu codul ASCII 10 deplasează cursorul pe prima coloană a liniei următoare; acesta se numește caracter de rând nou (newline) și poate fi introdus de la tastatură apăsând tasta ENTER (în C, acest caracter se mai notează și cu \n).

2.2. Identificatori

Un *identificator* (nume) este o succesiune de litere și, eventual, cifre, care începe în mod obligatoriu cu o literă. Prin *litere*, în acest caz înțelegem literalele mici și mari ale alfabetului și caracterul de subliniere (_). Literalele mici se consideră distincte de cele mari (de exemplu, variabilele max, Max și MAX se consideră diferite, spre deosebire de limbajul Pascal). Identificatorii sunt folosiți pentru a denumi variabile, constante, tipuri de date, funcții.

2.3. Cuvinte-cheie

Cuvintele-cheie (rezervate) sunt cuvinte care au un înțeles special pentru compilatorul C: sunt nume rezervate instrucțiunilor, tipurilor fundamentale și sintaxei pentru definirea funcțiilor și a tipurilor de date. În limbajul ANSI C există 32 de cuvinte rezervate:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Cuvintele-cheie (rezervate) se scriu întotdeauna cu litere mici.

2.4. Tipuri de date

Prin *tip de date* se înțelege o mulțime peste care se definesc următoarele aspecte:

- dimensiunea zonei de memorie asociată unui element;
- timpul de viață asociat datei;
- mulțimea operațiilor prin care valorile tipului pot fi modificate sau prelucrate și semnificația acestor operații;
- operatorii utilizați și restricții asupra acestora.

Tipurile de date pot fi predefinite (tipuri de bază) sau definite de utilizator (derivate). Tipurile fundamentale și dimensiunile lor:

Tipul	Descrierea	Număr de octeți
char, unsigned char, signed char	caracter reprezentat prin cod ASCII	1
short, unsigned short	întreg binar reprezentat prin complement față de 2	2
int, unsigned int		2
long, unsigned long		4
float	număr reprezentat în virgulă flotantă în simplă precizie	4
double	număr reprezentat în virgulă flotantă în dublă precizie	8
long double		10

Datele de tip char au valori în intervalul [0, 255] sau [-128, 127]. Datele de tip int (întregi cu semn) aparțin intervalului [-32768, 32767]. Datele de tip unsigned int (întregi fără semn) aparțin intervalului [0, 65535]. Datele de tip long (întregi cu semn în dublă precizie) aparțin intervalului [-2³¹, 2³¹-1]. Datele de tip unsigned long aparțin intervalului [0, 2³²-1]. O dată flotantă în simplă precizie de tip float, diferită de 0, are valoarea absolută în intervalul [3.4*10⁻³⁸, 3.4*10³⁸]. O dată flotantă în dublă precizie de tip double, diferită de 0, are valoarea absolută în intervalul [1.7*10⁻³⁰⁸, 1.7*10³⁰⁸], iar una de tip long double, în intervalul [3.4*10⁻⁴⁹³², 1.1*10⁴⁹³²].

2.5. Constante

Sunt valori fixe reprezentând caractere, siruri de caractere, numere întregi sau raționale.

2.6. Variabile

Prin *variabilă* înțelegem o dată a cărei valoare se poate schimba pe parcursul execuției programului. Unei variabile i se atribuie patru entități: *nume* (cu ajutorul căruia va putea fi referită pe parcursul execuției programului), *valoare* (la un moment dat), *tip* (valorile

pe care le poate avea variabila la momente diferite trebuie să aparțină același tip) și adresă în memorie. Corespondența dintre numele și tipul unei variabile se realizează cu ajutorul unei declarații.

Declarații de variabile simple

O declarație de variabile simple de același tip are forma:

```
<nume_tip> lista_de_identificatori;
```

Prin lista_de_identificatori se înțelege o succesiune de nume de variabile separate prin virgulă:

Exemplu :

```
int a, b, c, i, j;
char ch;
float x, y;
```

Declarația de tablou

Un tablou reprezintă un tip structurat care ocupă o zonă de memorie continuu și conține elemente de același tip.

Declarația unui singur tablou are forma:

```
<nume_tip> <nume_tablou>[dim_1][dim_2]...[dim_n];
```

Dacă se declară mai multe tablouri de același tip, se pot scrie separate prin virgulă, ca în exemplele de mai jos.

Exemplu :

```
int a[100], b[20][10];
char m[100];
```

Pentru tabloul a[] se vor putea referi direct elementele a[0], a[1], ..., a[99]. În cazul declarației generale de mai sus indicii elementelor tabloului pot lua valori astfel: primul – între 0 și dim_1-1, al doilea între 0 și dim_2-1 etc. Numele unui tablou poate fi utilizat în diverse construcții și are ca valoare adresa primului element al tabloului.

2.7. Comentariile

În limbajul C, comentariile încep cu secvența de caractere /* și se termină cu secvența */. Acestea sunt ignorate de compilator și au rolul de a explica anumite părți de program.

3. Expresii

O expresie este formată dintr-un operand sau mai mulți operanzi legați prin operatori. O expresie are o valoare și un tip și pot fi folosite parantezele pentru a impune o anumită ordine a operațiilor.

3.1. Operanzi

Un operand poate fi: o constantă, o constantă simbolică, numele unei variabile simple, numele unei structuri, numele unui tip, numele unui tablou, numele unei funcții, referirea la un element al unui tablou, apelul unei funcții, referirea la elementul unei structuri, o expresie inclusă între paranteze rotunde.

3.2. Operatori

Există trei tipuri de operatori. O expresie unară este formată dintr-un operator unar care precedă un operand sau cuvântul-cheie sizeof urmat de o expresie. O expresie poate fi, de asemenea, și denumirea unei variabile sau o expresie cast (conversie explicită). O expresie binară este compusă din doi operanzi asupra căror se aplică un operator binar.

Limbajul C conține următorii *operatori unari*:

Simbol	Semnificație
- ~ !	negația pentru numere întregi, negația pe biți, negația logică
* &	indirectarea și obținerea adresei unei variabile
sizeof	operatorul ce returnează dimensiunea unui tip
+	operatorul plus unar
++ -- ..	operatorii unari de incrementare și decrementare (pot fi pre și post)

Operatorii binari, care se aplică de la stânga la dreapta:

Simbol	Semnificație
* / %	operatorii multiplicativi (înmulțit, cât și rest)
+ -	operatorii aditivi (plus, minus)
<< >>	operatorii de deplasare pe biți
< > <= >= == !=	operatorii relaționali (mai mic, mai mare, mai mic sau egal, mai mare sau egal, egal, diferit)
& ^	operatorii pe biți (ȘI, SAU, SAU EXCLUSIV)
&&	operatorii logici (ȘI, SAU)
,	operatorul virgulă (de execuție secvențială)

Operatorul de atribuire în forma sa simplă se notează prin caracterul „=” și are forma: v = expresie (această expresie se numește *expresie de atribuire*). Operatorul de atribuire are prioritatea cea mai mică față de operatorii descriși până acum. Deoarece operatorii de atribuire se asociază de la dreapta la stânga, se poate scrie și $v_n = v_{n-1} =$

$\dots = v_1 = \text{expresie}$. În evaluarea expresiei se va aplica regula conversiilor implicate: dacă expresia din dreapta semnului egal are un tip diferit de cel al variabilei v , atunci valoarea ei se convertește la tipul variabilei și apoi se face atribuirea. Pentru a realiza operația de atribuire, în afara semnului „=” se mai poate folosi una dintre construcțiile $op=$, unde op poate fi: % / * - + & ^ | << >> ($v op= \text{expresie}$ este echivalentă cu $v = v op \text{ expresie}$).

Operatorul condițional. Se utilizează în evaluări de expresii care reprezintă alternative. Are formatul: $e1?e2:e3$, unde $e1$, $e2$ și $e3$ sunt expresii. Modul de evaluare a acestei expresii este următorul: se evaluatează expresia $e1$; dacă aceasta este diferită de zero valoarea expresiei condiționale este $e2$; dacă valoarea lui $e1$ este 0, valoarea expresiei condiționale este $e3$.

Operatorul virgulă. Se folosește pentru a grupa mai multe expresii în una singură și are cea mai mică prioritate. Cu ajutorul acestui operator construim expresii de forma: $e1, e2, \dots, en$ și valoarea întregii expresii este valoarea lui en (după ce s-au evaluat toate expresiile succesiv).

4. Structura programelor C

Orice program C are următoarea structură:

```
directive de preprocessare
declaratii globale
functii
```

4.1. Directive de preprocessare

Un program C poate fi prelucrat înainte de a fi compilat. Acest tip de prelucrare se numește *preprocessare*. Preprocesorul este apelat automat atunci când se începe compilarea, și relativ simplu și, de obicei, execută substituții de text. Prin intermediul lui se pot realiza: incluzări de text, definiții și apeluri de macrouri simple, compilare condiționată.

Includerea unui fișier-sursă. Are una dintre formele:

```
#include "specificator_de_fisier"
#include <specificator_de_fisier>
```

În primul caz, fișierul specificat este căutat în directorul curent și apoi în directoarele standard pentru `include`; în al doilea caz, se caută numai în directoarele standard. **ACTIONE:** textul fișierului specificat este inclus în program.

Constante simbolice. O constantă simbolică are forma:

```
#define nume succesiune_de_caractere
```

ACTIONE: nume este înlocuit din acest loc până la sfârșit în codul-sursă cu succesiune_de_caractere.

Exemplu:

```
#define MAX 5000
/* MAX se substituie peste tot în continuare cu 5000*/
```

Biblioteci. Limbajul C conține multe funcții pentru prelucrarea datelor; acestea sunt grupate în biblioteci. Dintre cele mai importante biblioteci, precizăm:

- stdio.h și io.h, pentru citire/scriere;
- stdlib.h și math.h, pentru prelucrări numerice;
- ctype.h, pentru prelucrarea sau verificarea caracterelor;
- mem.h și string.h, pentru siruri de caractere și zone de memorie;
- alloc.h, malloc.h și stdlib.h, pentru alocarea memoriei;
- conio.h, pentru interfață cu consola;
- graphics.h, pentru interfață grafică;
- dos.h, pentru interfață cu sistemul de operare DOS.

Declaratiile globale permit utilizarea unor tipuri de date și variabile definite de utilizator în orice punct din interiorul programului.

4.2. Funcții

În limbajul C există două tipuri de funcții: funcții care *returnează o valoare* și funcții care *nu returnează o valoare*. Structura unei funcții este următoarea:

```
tip_returnat nume_functie (lista parametrilor formali)
declaratii de parametri
{
    declaratii de variabile locale
    instructiuni separate prin ;
}
```

Funcția principală main conține instrucțiunile care se execută. Aceasta poate returna o valoare și poate avea și parametri.

Observații:

- parantezele rotunde trebuie să fie prezente chiar și atunci când lista parametrilor formali este vidă;
- partea de început a funcției până la acolada deschisă se numește *antetul funcției*;
- partea funcției inclusă între acolade, inclusiv acoladele, se numește *corpu funcției* (acoladele sunt obligatorii);
- atunci când nu există parametri formali se poate să nu scriem nimic sau se poate scrie cuvântul void (tipul vid), varianta din urmă fiind recomandată;
- când funcția nu returnează nimic se poate scrie void în loc de tipul returnat; dacă nu se scrie nimic, atunci tipul returnat va fi implicit tipul int;

- funcția main poate fi folosită și cu oricare dintre anteturile de mai jos:

```
void main(void)
void main()

int main(void)
int main()

main(void)
main()
```

5. Instrucțiuni

Vom defini în continuare instrucțiunile C caracteristice programării structurate: *structura secvențială, structura alternativă, structura repetitivă*.

5.1. Instrucțiunea vidă

Se reduce la caracterul „;” și nu are nici un efect.

5.2. Instrucțiunea expresie

Se obține scriind caracterul „;” după o expresie (de atribuire, apelul unei funcții, instrucțiuni expresie).

Exemple:

```
i++;           // postincrementare: i se marestă cu 1
max = a[i];    // expresie de atribuire
c = getchar(); // citirea unui caracter de la tastatura
--k;           // predecrementare: k se micșorează cu 1
```

5.3. Instrucțiunea compusă

Este o succesiune de declarații urmate de instrucțiuni, incluse între acolade. Declarațiile sau instrucțiunile (chiar și ambele) pot lipsi. Formatul acestei instrucțiuni:

```
{           declaratii
            instructiuni
}
```

5.4. Instrucțiunea if

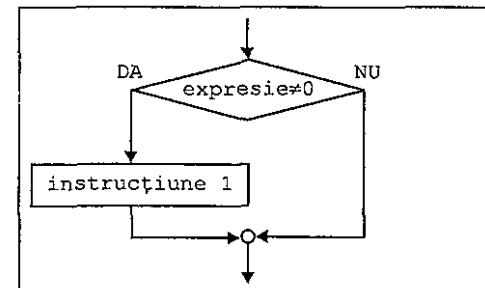
Instrucțiunea if ne permite ramificarea în funcție de valoarea unei expresii.

Format 1:

```
if(expresie) instructiune1;
```

Efect:

1. Se evaluatează expresia dintre paranteze.
2. Dacă valoarea expresiei este diferită de zero, atunci se execută instrucțiune 1; altfel, se trece la următoarea instrucțiune.

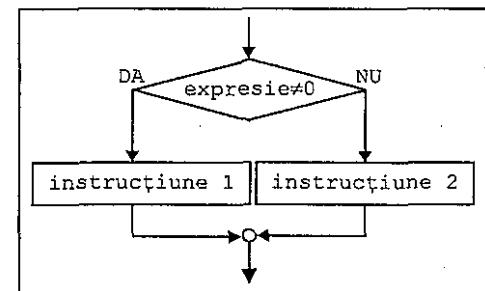


Format 2:

```
if(expresie) instructiune1;
else instructiune2;
```

Efect:

1. Se evaluatează expresia din paranteză.
2. Dacă valoarea expresiei este diferită de zero, atunci se execută instrucțiunea 1; altfel, se execută instrucțiunea 2.
3. Se trece la instrucțiunea următoare.



5.5. Instrucțiunea while

Are următorul format:

```
while(expresie) instructiune;
```

Această instrucțiune mai poartă denumirea de *instrucțiune repetitivă cu test inițial și număr necunoscut de pași*. Ea se execută astfel:

1. Se evaluatează expresia dintre paranteze.
2. Dacă valoarea expresiei este diferită de zero, se execută instrucțiunea și se trece la pasul 1.
3. Dacă valoarea expresiei este 0, se trece la următoarea instrucțiune.

5.6. Instrucțiunea *for*

Se mai numește *instrucțiune repetitivă cu număr cunoscut de pași*. Formatul instrucțiunii este :

```
| for(e1; e2; e3) instructiune;
```

unde e1, e2 și e3 sunt expresii.

for(e1; e2; e3) este *antetul ciclului*. Instrucțiunea care se execută repetat este *corpușul ciclului*. e1 reprezintă partea de inițializare a ciclului, e3 – partea de reinicializare a sa, iar e2 reprezintă condiția de continuare a ciclului.

5.7. Instrucțiunea *do-while*

Se mai numește *instrucțiune repetitivă cu test final*. Are următorul format :

```
| do instructiune while (expresie);
```

Pașii de execuție pentru această instrucțiune sunt :

1. Se execută instructiune.
2. Se evaluatează expresie.
3. Dacă valoarea expresiei este diferită de 0, atunci se reia pasul 1; altfel (valoarea expresiei este 0), se trece la instrucțiunea următoare.

5.8. Instrucțiunea *switch*

Permite realizarea structurii selective. Este o generalizare care poate fi înlocuită cu structuri alternative if imbricate. Formatul instrucțiunii este :

```
switch(expresie)
  case c1: sir1_de_instructiuni; |break;
  case c2: sir2_de_instructiuni; |break;
  ...
  case cn: sirn_de_instructiuni; |break;
|default: sir_de_instructiuni; |
```

Pașii de execuție a instrucțiunii sunt :

1. Se evaluatează expresia dintre paranteze.
2. Se compară pe rând valoarea expresiei cu valorile c₁, c₂, ..., c_n.
3. Dacă valoarea expresiei coincide cu valoarea c_k, atunci se execută secvența de instrucțiuni definită prin sir_k_de_instructiuni; dacă nu coincide cu nici una, atunci se execută sir_de_instructiuni.

Alternativa default nu este obligatorie, la fel și instrucțiunile break. Dacă, de exemplu, nu există instrucțiune break după sir_k_de_instructiuni, atunci se vor

executa și următoarele șiruri de instrucțiuni, până la terminare sau până la întâlnirea unei instrucțiuni break.

5.9. Instrucțiunea *break*

Formatul instrucțiunii este :

```
| break;
```

Această instrucțiune se utilizează pentru a ieși dintr-o instrucțiune switch sau pentru a ieși dintr-o instrucțiune ciclică.

5.10. Instrucțiunea *continue*

Formatul instrucțiunii este :

```
| continue;
```

Efect :

- în cadrul ciclurilor while și do-while, se realizează direct evaluarea expresiei care decide asupra continuării ciclului ;
- în ciclul for, ea realizează saltul la pasul de reinicializare.

6. Intrări/ieșiri standard

Limbajul C nu dispune de instrucțiuni pentru operațiile de intrare/ieșire. Acestea se realizează prin intermediu unor funcții predefinite, care sunt implementate într-o formă compatibilă pe majoritatea sistemelor de calcul.

6.1. Funcția de scriere cu format

Pentru scrierea cu format a datelor se folosește funcția printf. Scrierea se face în fișierul standard de ieșire și formatul este :

```
| int printf(const char *format [, lista_expresii ]);
```

Efect :

1. Acceptă o serie de argumente de tip expresie pe care, după ce le evaluatează, le transformă în șiruri de caractere conform formatului specificat.
2. Scrie aceste șiruri în fișierul standard de ieșire.

Dacă numărul de argumente specificate în format nu corespunde cu numărul de argumente din lista de expresii, atunci apar rezultate neașteptate, care pot avea efecte dăunătoare, în special dacă numărul de argumente este mai mic decât numărul de specifikatori de format. Rezultatul furnizat de funcție, în caz de succes, este numărul de octeți scriși, iar în caz de eroare, valoarea întoarsă este EOF.

Specificatorii de format folosiți de funcțiile `printf` și `scanf` sunt:

Tipuri de formate	
Pentru numere întregi de tip int	
i	întreg în baza 8, 10 sau 16, în funcție de primul sau primele două caractere
d	întreg în baza 10
o	întreg în baza 8, nu este necesară scrierea cifrei 0 la începutul numărului
x	întreg în baza 16, nu este necesară scrierea secvenței 0x la începutul numărului
u	întreg fără semn
Pentru caractere	
s	șir de caractere
c	un singur caracter
Pentru numere reale de tip float	
e E	număr real de forma iiiii.zzzzzz, unde numărul de cifre zecimale z este dat de precizie (implicit 6)
f	număr real de forma i.zzzzzz, unde numărul de cifre zecimale este dat de precizie (implicit 6). Se observă că pentru partea întreagă este folosită doar o cifră, spre deosebire de formatele e și E, unde pot fi folosite până la 4 cifre.
g G	număr real care suprimă caracterele terminale ce nu influențează valoarea, adică cifrele de 0 de la sfârșit și punctul zecimal, dacă numărul are partea fractionară 0

6.2. Funcția de citire cu format

Pentru citirea cu format a datelor se folosește funcția `scanf`. Aceasta are sintaxa:

```
int scanf(const char *format [,lista_adrese_variabile]);
```

unde `format` este o constantă de tip `șir de caractere` specificând formatul în care vor fi cîștite caracterele de la intrare, iar `lista_adrese_variabile` este formată din adresele variabilelor separate prin virgulă. Efectul apelului este următorul:

- citește o secvență de câmpuri de intrare, caracter cu caracter, până la terminarea introducerii câmpurilor și apăsarea tastei `ENTER`;
- formează fiecare câmp conform formatului specificat de argumentul `char *format`;
- valorile astfel construite se memorează la adresele variabilelor specificate ca argumente.

7. Fișiere

Funcțiile de intrare/ieșire se bazază pe conceptul de flux (`stream`). În C, unui flux î se poate asocia orice: un fișier pe disc, un terminal, o unitate de CD-ROM, imprimanta, modemul, placa de sunet.

7.1. Deschiderea unui fișier

Se realizează cu ajutorul funcției `fopen`, cu sintaxa:

```
FILE *fopen(const char *nume_fisier,  
            const char *mod_deschidere);
```

Modurile de deschidere pot fi: `r` (deschidere numai pentru citire), `w` (deschidere numai pentru scriere), `a` (deschidere numai pentru adăugare), `b` (deschidere numai în mod binar) și `t` (deschidere în mod text, opțiune implicită). Dacă dorim să combinăm două sau mai multe moduri de deschidere, putem utiliza `+`.

7.2. Închiderea unui fișier

Se realizează cu ajutorul funcției `fclose`, cu sintaxa:

```
int fclose(FILE *stream)  
care returnează 0 în caz de succes.
```

Închiderea tuturor fișierelor deschise se realizează cu ajutorul funcției `fcloseall`, cu sintaxa:

```
int fcloseall( void )
```

7.3. Funcția de verificare a sfârșitului de fișier

Este `feof`, are sintaxa:

```
int feof( FILE *stream )
```

și returnează 0 dacă poziția pe care ne aflăm în fișier nu este la sfârșitul acestuia și o valoare diferită de zero dacă poziția actuală indică sfârșitul de fișier.

7.4. Funcții de citire/scriere

Funcțiile de citire pot fi, în funcție de tipul datelor citite, de mai multe tipuri. Există funcții de citire pentru: caractere, șiruri de caractere, cu format, pentru înregistrări.

- Funcții de citire/scriere pentru caractere:

Pentru citirea unui caracter se folosesc funcțiile cu sintaxa:

```
int fgetc( FILE *stream );  
int getc( FILE *stream );
```

Dacă citirea a avut succes, se returnează valoarea caracterului citit, altfel se returnează EOF.

Pentru scrierea unui caracter se folosesc funcțiile cu sintaxa:

```
int fputc(int c, FILE *stream);
int putc(int c, FILE *stream);
```

În caz de scriere reușită, ambele întorc caracterul care s-a scris; în caz de eroare, ambele întorc caracterul EOF.

- Funcții de citire/scriere pentru șiruri de caractere:

Funcția fgets citește un șir de caractere dintr-un fișier și are sintaxa:

```
int fgets(char *s, int n, FILE *stream)
```

unde s reprezintă buffer-ul în care se stochează șirul citit și n indică numărul maxim de caractere care se vor citi.

Pentru scrierea unui șir de caractere într-un fișier se poate folosi funcția fputs, cu sintaxa:

```
int fputs(const char *s, FILE *f)
```

În caz de eroare, se întoarce valoarea EOF.

- Funcții de citire/scriere cu format:

Sintaxa funcției fscanf pentru citirea variabilelor dintr-un fișier de tip stream este:

```
int fscanf(FILE *stream,
           const char *format[adresa_var1, ...])
```

Această funcție citește o secvență de câmpuri de intrare caracter cu caracter, formatează fiecare câmp conform formatului specificat corespunzător, iar câmpul format este transmis la adresa variabilei specificate.

Pentru scrierea cu format se utilizează funcția fprintf, cu sintaxa:

```
int fprintf (FILE *stream, const char *format [, argument, ...]);
```

și acceptă o serie de argumente de tip expresie pe care le formatează conform specificării din șirul format; scrie datele în fișierul cerut.

8. Noi tipuri de date: struct, enum, union

8.1. struct

O structură este o metodă de a grupa mai multe date împreună, de exemplu, numele și adresa unei persoane. Elementele unei structuri sunt accesate utilizând operatorul „.”.

Exemple :

tipul struct complex declarare, utilizare	tipul struct pers tipul struct angajat
<pre>struct complex{ double x; double y; } ... struct complex z; ... z.x = 5.45; z.y = -7.89;</pre>	<pre>struct pers{ char nume[35]; int ani; }; struct angajat{ struct pers p; long salariu; int vechime; } ... struct angajat a; a.p.nume = "Maia Ionescu"; a.p.anii = 34; a.salariu = 6000000; a.vechime = 3;</pre>

Observăm, în al doilea exemplu, că o structură poate fi și membră a altei structuri.

8.2. enum și typedef

Să considerăm următorul program sugestiv :

```
#include <stdio.h>

enum culori { ROSU, GALBEN, VERDE, ALBASTRU,
              VIOLET, NUMAR_CULORI };

typedef enum culori TCuloare;

int main(void)
{
    TCuloare cer, padure;
    printf("In enum sunt %d culori\n", NUMAR_CULORI);
    cer = ALBASTRU;
    padure = VERDE;
    printf("cer = %d\n", (int)cer);
    printf("padure = %d\n", (int)padure);
    return 0;
}
```

Efectul programului :

In enum sunt 5 culori

```
cer = 3
padure = 2
```

Se definește tipul enumerare **culori** cu numele culorilor fiind cuvinte constante care vor avea atribuite valori numerice, începând, în cazul nostru, cu 0. Aceste constante vor avea următoarele valori :

cuvânt constant	valoare numerică
ROSU	0
GALBEN	1
VERDE	2
ALBASTRU	3
VIOLET	4
NUMAR_CULORI	5

În mod natural, diferitele tipuri definite de utilizator pot fi combinate, astfel încât putem avea, de exemplu, tablouri de structuri (sau invers), tablouri ale căror elemente sunt enumerări etc. Pentru a evita declarații de genul **struct node x, y, z** sau **enum Boolean s, b;**, limbajul C furnizează mecanismul de definire a numelor de tipuri cu ajutorul lui **typedef**. Acesta crează tipuri sinonime cu cele denumite și care pot fi utilizate în declarații de variabile, măring claritatea programului. Este sugestivă definirea în programul anterior a tipului **TCuloare** și utilizarea sa în programul principal în locul tipului **enum culori**.

8.2. union

C furnizează posibilitatea de a stoca tipuri diferite în aceeași locație de memorie. O variabilă de tip **union** (care implementează această capacitate) este o variabilă care poate, la momente diferite, să memoreze valori de tipuri diferite. Pentru a se asigura că stiva nu este **cerută**, dimensiunea acestei locații de memorie este determinată de membrul cu cea mai mare lungime. Sintaxa pentru uniuni este foarte asemănătoare cu cea pentru structuri. De exemplu, putem avea următoarea declarație :

```
union *TUniune {
    int ival;
    float fval;
    char cval;
}
...
union TUniune x,y,z;
...
x.ival = 27;
y.fval = 32.23;
z.cval = 'X';
```

Nu există nici o modalitate de a determina tipul datei curente stocate într-o uniunie. Revine sarcina programatorului să țină cont de ceea ce se află într-o variabilă de tip

uniune la un anumit moment. O modalitate folosită este de a considera o variabilă care să indice tipul curent de fiecare dată când se setează valoare unei uniuni (de exemplu, o variabilă de tip **enum**). Dereferențierea și aplicarea operatorului de selecție a unui membru (->) funcționează și în cadrul uniunilor. De exemplu, următoarea sintaxă este legală :

```
union TUniune *uptr;
...
uptr->ival=77;
```

Sau un alt exemplu :

```
enum boolean { false, true };
enum boolean x;
...
x = true;
```

9. Tablouri

9.1. Concept, declarare, initializare

Mai multe valori de același tip de date pot fi stocate într-un tablou (*array, vector*). Pentru a accesa un anumit element din tablou, este necesar numele tabloului și indexul elementului. Indexul este adăugat numelui variabilei într-o paranță pătrată. Dacă o variabilă este de tip tablou, atunci acest lucru este definit în declarația variabilei, iar aici trebuie specificată și dimensiunea acestuia. În exemplul de mai jos :

```
int valori[12];
```

unde este declarat un tablou cu numere întregi de dimensiune 12. Indexul începe totdeauna de la 0 și, în cazul nostru, ultimul element are indexul 11 (dimensiunea tabloului minus 1!). Pentru index sunt permise numai valori întregi. Iată un alt exemplu în care un tablou este inițializat cu sumele elementelor de la 1 la indexul curent plus 1 :

```
int suma[100], i;
suma[0]=1;
for (i=1;i<100;i++){
    suma[i]=(i+1)+suma[i-1];
}
```

Indexul **i** ia valori de la 1 la 99. O valoare 0 pentru **i** va avea ca rezultat valoarea -1 ca index în partea dreaptă a operatorului de atribuire. Unele compilaatoare sesizează această eroare, dar altele vor avea ca rezultat blocarea programului la execuție. Observăm, de asemenea, că se pot defini tablouri și variabile simple în aceeași linie de declarație.

Pentru inițializarea unui tablou se vor specifica valorile inițiale între acolade, ca în exemplul următoare:

```
int prime[100] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541 };

float temp[1024] = { 5.0F, 2.3F };
double vals[] = { 1.23, 2.34, 3.45 };
```

În acest exemplu, `prime` este inițializat cu valorile primelor 100 de numere prime. Primele două elemente ale vectorului `temp` (`temp[0]` și `temp[1]`) sunt inițializate cu valorile `5.0F`, respectiv `2.3F`. Elementele rămase în `temp` vor fi inițializate cu `0.0F`. Remarcăm utilizarea caracterului `F` pentru a indica faptul că aceste elemente sunt de tip `float` și nu `double`. Tabloul `vals[]` conține trei numere de tip `double`. Observăm că lungimea tabloului nu trebuie specificată, limbajul C este destul de intelligent pentru a o deduce.

9.2. Tablourile multidimensionale

Ele sunt declarate și referite în C folosind mai multe perechi de paranteze pătrate. De exemplu:

```
int table[2][3][4];
```

`table[]` este un tablou de dimensiuni $2 \times 3 \times 4$, iar cel mai din dreapta element se schimbă cel mai rapid de-a lungul memoriei (primul element este `table[0][0][0]`, al doilea este `table[0][0][1]`, iar ultimul este `table[1][2][3]`).

Inițializarea tablourilor multidimensionale se realizează în mod natural, de exemplu:

```
int mat[3][4] = { { 0, 1, 2, 3}, { 3, 4, 5, 6}, { 6, 7, 8, 9} };
```

unde se inițializează un tablou cu 3 linii și 4 coloane. Ar fi corectă și o inițializare de forma:

```
int mat[3][4] = { { 0, 1, 2}, { 3, 4, 5}, { 6, 7, 8} };
```

cu mențiunea că ultima coloană va fi inițializată automat cu elemente 0.

9.3. Tablourile de caractere

Aceste tablouri conțin câte un caracter (element de tip `char`) în fiecare celulă. În C, manipularea sirurilor ca tablouri de caractere și operațiile cu siruri (concatenare, căutare, comparare etc.) sunt efectuate prin apelul unor funcții din biblioteca `string.h` (de exemplu, `strcat`, `strchr`, `strcmp` etc.). Pentru a lucra cu funcții de manipulare a sirurilor, sfârșitul sirului trebuie să fie caracterul NUL (`\0`). Tablourile de caractere pot fi inițializate astfel:

```
char str[] = {'a', 'b', 'c'};
char prompt[] = "introduceti un numar";
```

În acest exemplu, `str` are 3 caractere și lungime de 3 octeți, iar `prompt` are 21 de caractere (cu unul mai mult decât numărul caracterelor folosite), ultimul fiind folosit pentru a stoca `\0` ca fiind terminatorul de sir.

10. Pointeri, `sizeof`, alocarea dinamică a memoriei

Dacă există un lucru care face distincție certă între limbajul C și alte limbi de programare, atunci acesta este utilizarea *pointerilor*.

10.1. Noțiune, exemple, metode specifice

Un *pointer* este o variabilă care conține adresa unei locații de memorie.

Presupunând că `p` este un pointer, atunci `*p` este valoarea de la adresa `p`. Presupunând că `i` este adresat de pointerul `p`, atunci `i` și `*p` sunt același lucru și semnifică o valoare, iar `p` și `&i` sunt același lucru și semnifică o adresă. Pointerii sunt declarați prin specificarea tipului spre care ei pointează, de exemplu:

```
int *p;
```

unde `p` este definit ca un pointer la `int`, de unde deducem că `*p` este un `int`, conform cu declarația. Trebuie remarcat că, prin această declarație, compilatorul va aloca spațiul necesar pentru adresă (pe cele mai multe compilatoare este de obicei 4 octeți), *dar nu și pentru variabila spre care el pointează!* Aceasta este un aspect foarte important și deseori

nu este luat în considerare, producându-se erori semnificative. Înainte de utilizarea lui `*p` într-o expresie, trebuie să ne asigurăm că `p` este adresa unui `int` valid. Acestui `int` îl trebuie alocat spațiu, static, automat sau prin alocare dinamică a memoriei la execuție (folosind, de exemplu, funcția `malloc()`).

Iată un exemplu sugestiv pentru utilizarea pointerelor:

```
#include <stdio.h>
#include <conio.h>
void main (void) {
    int m = 0, k = 2;
    int *p;
    char msg[] = "salutare lume!";
    char *cp;

    p = &m;           /* p pointeaza acum la m */
    *p = 1;           /* m este acum egal cu 1 */
    k = *p;           /* k este acum egal cu 1 */
    cp = msg;         /* cp pointeaza la primul caract din msg */
    *cp = 'S';        /* schimb in majuscula a lui 's' in msg */
    cp = &msg[9];     /* cp pointeaza la 'l' */
    *cp = 'L';        /* schimbarea in majuscula a lui 'l' */

    printf ("m = %d, k = %d\nmsg = \"%s\"\n", m, k, msg);
    getch();
}
```

Efectul pe ecran al programului:

```
m = 1, k = 1
msg = "Salutare Lume!"
```

Remarcăm utilizarea lui `msg` fără index, acesta fiind considerat un pointer la primul caracter din tablou. De fapt, numele unui tablou urmat de un index este echivalent cu un pointer urmat de deplasare, ca în exemplul sugestiv de mai jos:

```
#include <stdio.h>
#include <conio.h>

void main (void) {
    char msg[] = "salutare lume!";
    char *cp;

    cp = msg;
    cp[0]      = 'S';
    *(msg+9)   = 'L';
```

```
printf ("%s\n", msg);
printf ("%s\n", &msg[0]);
printf ("%s\n", cp);
printf ("%s\n", &cp[0]);
getch();
}
```

Efectul pe ecran:

```
Salutare Lume!
Salutare Lume!
Salutare Lume!
Salutare Lume!
```

În plus, `cp` este o variabilă și poate fi schimbată, în timp ce `msg` este constant și nu se poate afla ca atare la stânga unei atribuiri.

Pointerii sunt un instrument foarte puternic, deoarece ei permit alocarea și dealocarea segmentelor de memorie în mod dinamic, în timpul execuției. Dimensiunea unui tablou, de exemplu, trebuie definită la compilare pentru că atunci când este necesar un spațiu mare pot apărea probleme. Pointerii sunt similari cu tablourile, dar prin intermediul apelului `malloc(n)` se returneză o secvență de memorie de dimensiune `n` octeți, iar pentru eliberarea acesteia se folosește `free(p)`, unde `p` este un pointer care a adresat această memorie. În exemplul de mai jos, se alocă memorie pentru 2.000 de numere întregi, care apoi sunt inițializate cu numerele de la 1 la 2.000, iar în final memoria este eliberată.

```
int *p,i;
p = malloc( 2000*sizeof(int) );
for(i=0;i<2000;i++) {
    *(p+i)=i+1;
}
...
free (p);
```

Funcția `sizeof` returnează dimensiunea memoriei necesare pentru un tip de bază, în cazul nostru `int`. De obicei, un întreg are nevoie de 4 octeți. De exemplu, `malloc(2000)` va rezerva spațiu doar pentru 2.000 de octeți, destul pentru a memora 500 de numere întregi, de aceea avem nevoie să apelăm `malloc()` cu parametrul `(2000*sizeof(int))`. Incrementând pointerul `p` cu 1, acesta nu avansează cu un octet, ci cu dimensiunea tipului de bază al acestuia. Dacă, în acest caz, un întreg are 4 octeți, atunci `p+1` înaintează cu 4 octeți la dreapta poziției unde pointează `p`. Operatorul de incrementare este aplicat înaintea dereferențierii, căci operația `*p+1` ar fi permisă doar la dreapta instrucțiunii de atribuire și ea este valoarea dc la adresa `p` la care se adaugă 1. Funcția `free(p)` returnează `void` și execută eliberarea memoriei alocate de `p`.

10.2. Pointeri folosiți ca argumente în funcții

Am văzut deja că funcțiile C nu pot modifica argumentele lor. Dar ele pot modifica valorile spre care pointează, prin considerarea drept parametri a unor pointeri care vor executa un *apel prin referință*. De exemplu, metoda de mai jos, care interschimbă valorile a două variabile:

```
void interschimba(int *pi, int *pj) {
    int temp;

    temp = *pi;
    *pi = *pj;
    *pj = temp;
}
```

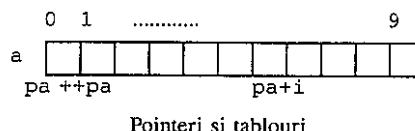
Dacă vom elimina asterixurile din toată metoda, atunci ea va fi compilată, dar efectul nu va fi cel așteptat (valorile se interschimbă doar local, iar în exterior, prin apelul funcției nu se va întâmpla nimic).

10.3. Pointeri și tablouri

Pointerii și tablourile sunt două noțiuni strâns legate în C, deoarece elementele unui tablou sunt amplasate în locații consecutive din memorie. Să considerăm următoarea secvență:

```
int a[10], x;
int *pa;

pa = &a[0]; /* pa adreseaza pe a[0] */
x = *pa; /* x = continutul lui pa (a[0], în acest caz) */
```



Atunci:

- $pa+i$ este adresa lui $a[i]$, mai poate fi scris ca $a+i$ sau $\&a[i]$;
- $pa[i]$ este $*(pa+i)$ sau $a[i]$ sau $*(\&a[i])$.

Deși au multe în comun, pointerul și tabloul sunt entități diferite:

- un pointer este o variabilă, putem scrie: $pa = a$ și $pa++$;
- un tablou nu este o variabilă: $a = pa$ și $a++$ sunt ilegale!

Important: compilatorul C nu verifică limitele unui tablou atunci când acesta este manipulat cu ajutorul pointerilor, de aceea aceste operații trebuie să fie foarte atent implementate, altfel există pericolul suprascrierii memoriei.

Pentru folosirea tablourilor multidimensionale utilizând pointeri, trebuie să știm că acestea se pot transforma în tablouri unidimensionale, pe principiu că fiecare element din tablou este tot un tablou, până când ajungem la elemente de bază.

10.4. Pointeri și structuri

Puteam avea atât elemente de tip pointer la structură, cât și câmpuri în structură care pointează la ea însăși (liste înlănuite sau structuri cu autoreferire).

Exemplul 1. Pointeri simpli la structuri

```
struct COORD {float x,y,z;} pt;
struct COORD *pt_ptr;

...
pt_ptr = &pt; /* asignare adresa lui pt pentru pt_ptr*/
```

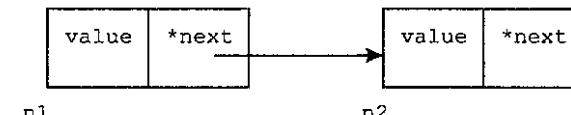
Operatorul „ $->$ ” ne permite accesul la membrii structurii astfel:

```
pt_ptr->x = 1.0;
pt_ptr->y = pt_ptr->y - 3.0;
```

Exemplul 2. Structuri cu autoreferire (liste înlănuite)

```
typedef struct {
    int value;
    ELEMENT *next;
} ELEMENT;
```

ELEMENT n1, n2;	ELEMENT n1, n2;
n1.next = &n2;	n1->next = n2;
n1.next	n1->next
- este un pointer (are tipul ELEMENT*)	- are tipul ELEMENT
- i se atribuie adresa lui n2	- i se atribuie n2



Legătura dintre două noduri

10.5. Greșeli uzuale cu pointeri

10.5.1. Neatribuirea unui pointer înainte de a-l folosi

Prin nealocarea memoriei înainte de folosirea unui pointer efectele pot fi neașteptate, pentru că nu vom obține eroare la compilare. Considerăm exemplul:

<code>int *x; *x = 100;</code>	Prin compilarea unui program simplu cu aceste două instrucțiuni nu obținem eroare la compilare, dar putem obține eroare în timpul execuției, de exemplu: <code>First-chance exception in test_pointer.exe: 0xC0000005: Access Violation.</code>
------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Acest lucru se întâmplă deoarece `x` nu are atașată o locație fizică în memorie, va avea eventual una aleatorie deja ocupată. Două soluții ar fi alocarea de memorie, folosind, de exemplu, `malloc`, sau utilizarea sa prin intermediul unei variabile deja alocate:

<code>x obține o locație proprie de memorie</code>	<code>x</code> va avea adresa lui <code>y</code> , și prin apelul <code>*x = 100;</code> se va modifica iremediabil valoarea lui <code>y</code> !
<code>int *x; x = (int*) malloc(sizeof(int)); *x = 100;</code>	<code>int *x, y; x = &y; *x = 100;</code>

10.5.2. Indirectare ilegală

Presupunem că folosim funcția standard de bibliotecă `malloc()` pentru a aloca memorie dinamic (în timpul execuției). Această funcție returnează un pointer la un bloc de memorie dacă cererea s-a executat cu success și pointerul `NULL` în caz contrar. Sintaxă:

```
char *malloc() metoda standard in biblioteca <stdlib.h>
```

Presupunem următoarea secvență:

```
char *p;  
*p = (char*) malloc(100); /*cerere de 100 de octeti din  
memorie*/  
*p = 'y';
```

Greșeala de mai sus este că în a doua linie atribuirea trebuie făcută lui `p` și nu lui `*p`, deoarece valoarea de întoarcere a lui `malloc()` trebuie să fie de tip `char*`. Următoarea problemă, mai subtilă, apare atunci când nu este suficientă memorie disponibilă și `p` va fi `NULL`, deci nu vom putea executa atribuirea din a treia linie. Un bun program C trebuie să facă această verificare, de exemplu:

```
char *p;  
p = (char*) malloc(100); /*cerere 100 octeti din memorie*/
```

```
if( p == NULL )  
    printf( "Eroare la alocarea memoriei!" );  
    exit(1);  
}  
*p = 'y';
```

11. Siruri de caractere

11.1. Reprezentarea și lucrul cu siruri de caractere

Un *sir constant*, cum ar fi "Eu sunt un sir", este un tablou unidimensional de caractere. Acesta este reprezentat intern în C prin intermediul caracterelor ASCII din sir „E”, „u”, spațiu, „s”, ..., „r”, și este terminat prin caracterul „\0”, astfel încât să se poată depista sfârșitul acestuia. Sirurile constante sunt adesea utilizate pentru afișarea pe ecran sau scrierea de mesaje în fișiere, de exemplu utilizând funcția `printf`:

```
printf("Salutare lume!\n");  
printf("Valoarea lui x este: %f\n", x);
```

Sirurile constante pot fi asociate unor variabile. C furnizează variabile de tip `char`, care conțin întotdeauna un caracter (1 octet). Un sir de caractere este stocat într-un vector cu elemente de tip `char`. Niciodată nu trebuie uitat că sirurile de caractere trebuie să se termine cu caracterul „\0”, deci este necesară *o locație în plus* în tablou!

Exemplu. Sirul de caractere "Acesta este un sir!" se poate reprezenta astfel:

A	c	e	s	t	a		e	s	t	e		u	n		s	i	r	!	\0
---	---	---	---	---	---	--	---	---	---	---	--	---	---	--	---	---	---	---	----

11.2. Operații cu siruri de caractere: biblioteca <string.h>

Limbajul C nu furnizează operatori care manipulează siruri de caractere, acestea sunt manipulate folosind fie pointeri, fie rutine speciale din biblioteca standard `string.h`.

Cele mai utilizate sunt (considerăm `s`, `s1`, `s2` de tipul `char*`, `c` de tip `char`, `n` de tip `size_t`) cele din tabelul următor:

Sintaxă	Efect
<code>char *strcat(s1, s2)</code>	adaugă sirul <code>s2</code> la sfârșitul sirului <code>s1</code>
<code>char *strchr(s, c)</code>	returnează prima apariție a unui caracter <code>c</code> în sirul parametrul <code>s</code> și returnează un pointer la acesta sau pointerul <code>NULL</code> dacă <code>c</code> nu este găsit
<code>int strcmp(s1, s2)</code>	compară sirurile <code>s1</code> și <code>s2</code> (ordinea lexicografică); returnează o valoare negativă dacă <code>s1</code> este mai mic decât <code>s2</code> , 0 dacă sunt egale și pozitivă dacă <code>s1</code> este mai mare decât <code>s2</code> (<i>case sensitive</i>)

Sintaxă	Efect
int strcmp(s1, s2)	compară sirurile s1 și s2 (ordinea lexicografică); returnează o valoare negativă dacă s1 este mai mic decât s2, 0 dacă sunt egale și pozitivă dacă s1 este mai mare decât s2 (<i>case insensitive</i>)
char *strcpy(s1, s2)	copiește sirul s2 în sirul s1
size_t strlen(s)	returnează lungimea sirului s (fără ultimul caracter \n)
char *strncat(s1, s2, n)	adaugă maximum n caractere din sirul s2 la sfârșitul sirului s1, caracterele de după terminatorul nul \n nu sunt copiate
int strncmp(s1, s2, n)	compară maximum n caractere din sirurile s1 și s2 (ordinea lexicografică); returnează o valoare negativă dacă s1 este mai mic decât s2, 0 dacă sirurile sunt egale; pozitivă dacă s1 este mai mare decât s2 (<i>case sensitive</i>)
int strnicmp(s1, s2, n)	compară maximum n caractere din sirurile s1 și s2 (ordinea lexicografică); returnează o valoare negativă dacă s1 este mai mic decât s2, 0 dacă sirurile sunt egale; pozitivă dacă s1 este mai mare decât s2 (<i>case insensitive</i>)
char *strncpy(s1, s2, n)	copiește maximum n caractere din sirul s2 în sirul s1
char *strrchr(s, c)	returnează ultima instanță a caracterului c în sirul s, un pointer la acesta sau pointerul NULL, dacă c nu este găsit

Alte funcții utile pentru manipularea sirurilor de caractere (considerăm s, s1, s2 de tipul char*, c de tip char, n de tip size_t):

Sintaxă	Efect
size_t strcspn(s1, s2)	returnează indexul primului caracter din s1 care apare în s2 sau lungimea lui s1 dacă nici o potrivire nu este posibilă
char *strerror(n)	returnează un sir de caractere corespunzător cu codul de eroare n
char *strupbck(s1, s2) *	returnează un pointer la primul caracter din s1 care se potrivește cu un caracter din s2 sau pointerul NULL, dacă nici unul nu este găsit
size_t strspn(s1, s2)	returnează un pointer la caracterul din s1 care este egal cu un caracter din s2 și 0 dacă nici o potrivire nu este găsită
char *strstr(s1, s2)	returnează un pointer la prima apariție a lui s2 în s1 și pointerul NULL altfel
char *strtok(s1, s2)	returnează un pointer la un semn (<i>token</i>) din s1 delimitat de caracterele din s2, adică va căuta acel subșir din s1 delimitat de caractere din s2

12. Probleme rezolvate în C

12.1. Tipăriți pe ecran textul: Bine ai venit în lumea C!

Program

```
#include <stdio.h>

void main()
{
    printf("Bine ai venit în lumea C!");
}
```

Analiza programului

- prima linie indică includerea bibliotecii stdio.h, care conține funcția printf;
- funcția main() returnează void, deci nu conține nici o instrucție return;
- dacă uităm să scriem linia #include <stdio.h> atunci, la compilare, se va returna eroarea: 'printf': undeclared identifier.

Efect

Bine ai venit în lumea C!

O altă modalitate de a rezolva problema poate fi definirea unei constante simbolice, astfel:

```
#include <stdio.h>
#define MESAJ "Bine ai venit în lumea C!"
void main()
{
    printf(MESAJ);
}
```

În cadrul etapei de preprocesare se va înlocui peste tot în program MESAJ cu "Bine ai venit în lumea C!", rezultând programul prezentat anterior.

12.2. Scrieți un program care citește de la tastatură numele unei persoane și afișează pe ecran o urare de bun venit în lumea C, folosind acest nume.

Program

```
#include <stdio.h>
void main()
{
    char nume[10];
    printf("Numele dumneavoastră: ");
    scanf("%s", nume);
    printf("Bine ai venit %s în lumea C!", nume);
}
```

Analiza programului

Prima linie este o directivă de preprocessare, aceasta va include în program headerul stdio.h, care conține prototipurile (anteturile) funcțiilor scanf și printf. Apoi urmează funcția principală main, care nu returnează nimic și în corpul său conține:

- declarație de sir de caractere (nume) de dimensiune maximum 10;
- afișarea pe ecran a sirului "Numele dumneavoastră:";
- citirea numelui (se va citi în variabila nume, care este de fapt adresa primului element al tabloului de caractere nume);
- ultima instrucție este apelul funcției printf, care afișează pe ecran un mesaj corespunzător (când se întâlnește specificatorul de format %s se va înlocui cu prima variabilă de după virgulă și apoi sirul obținut se va afișa pe ecran).

Efect

```
Numele dumneavoastră: Doina
Bine ai venit Doina în lumea C!
```

12.3. Pentru două variabile date de tip întreg, interschimbați conținutul lor.

Program

```
#include <stdio.h>
main()
{
    int a, b, aux;
    printf("Introduceti cele doua variabile (a, b):");
    scanf("%d%d", &a, &b);
    aux = a;
    a = b;
    b = aux;
    printf("Dupa interschimbare: a = %d, b = %d", a, b);
    return 0;
}
```

Analiza programului

- pe prima linie se include headerul stdio.h pentru a putea utiliza funcțiile de intrare/iesire printf și scanf;
- funcția main nu are specificat tipul returnat, deci se va considera că acest tip este int;
- pe prima linie în corpul funcției principale se declară trei variabile întregi: a, b și aux;
- folosind funcția printf se afișează sirul "Introduceti...";
- folosind funcția scanf, se citesc de la tastură cele două numere, care au formatul corespunzător tipului int: %d; observăm că parametrii de după virgulă în funcție sunt adresele variabilelor (se folosește operatorul de adresă & înaintea numelui);
- urmează cele trei instrucțiuni clasice de atribuire folosind „metoda celor trei pahare”;

- se afișează rezultatul, folosind un apel de funcție printf; observăm că în sirul de formatare avem două sevențe %d, care se vor înlocui cu valorile din variabilele a, respectiv b, apoi se vor afișa;
- urmează o instrucție return, deoarece, cum am specificat mai sus, tipul returnat de funcția principală este int (lipsa acestei instrucțiuni va duce la apariția la compilare a avertismentului 'main' : no return value).

Efect

```
Introduceti valorile celor două variabile (a, b): 45 - 23
Dupa interschimbare: a = -23 b = 45
```

Analizați și următoarea metodă de rezolvare (care nu folosește nici o variabilă suplimentară):

```
#include <stdio.h>
main()
{
    int a, b;
    printf("Introduceti cele doua variabile (a, b):");
    scanf("%d%d", &a, &b);
    a = a - b;
    b = a + b;
    a = b - a;
    printf("Dupa interschimbare: a = %d, b = %d", a, b);
    return 0;
}
```

12.4. Calculați suma cifrelor unui număr natural dat cu maximum 9 cifre.

Program

```
#include <stdio.h>
void main(void)
{
    unsigned long n, m;
    short s = 0;
    printf("N = "); scanf("%ld", &n);
    m = n;
    while(m)
    {
        s += m % 10;
        m /= 10;
    }
    printf("Suma cifrelor lui %ld este %d.", n, s);
}
```

Analiza programului

- corpul funcției main conține pe primele linii declarații de variabile: pe prima linie declarăm variabilele m și n de tip întreg fără semn pe 4 octeți, pe a doua linie variabila s, în care vom calcula suma cifrelor lui n (observăm inițializarea acesteia de la declarare cu 0);
- citirea numărului N: observăm formatul folosit (%ld);
- i se atribuie lui m variabila n (pentru a nu pierde valoarea n);
- urmează o instrucție while care conține două instrucții de atribuire compusă (în prima se adaugă la s ultima cifră a lui m, în a doua se trunchiază m de ultima cifră); aceste instrucții se repetă până când m are valoarea 0;
- se afișează rezultatul folosind tot o funcție printf.

Efect

```
N = 987654321
Suma cifrelor lui 987654321 este 45.
```

Observație

Analizați și următoarea formă de rezolvare, folosind o instrucție do-while:

```
#include <stdio.h>
void main(void)
{
    unsigned long n, m;
    short s = 0;
    printf(" N = "); scanf("%ld", &n);
    m = n;
    do
    {
        s += m % 10;
        m /= 10;
    }
    while(m);
    printf("Suma cifrelor lui %ld este %d.", n, s);
}
```

12.5. Scrieți un program care simulează comportarea unui calculator electronic pentru numere întregi: se introduc două numere întregi și o operație care poate fi +, -, *, /, %, reprezentând adunarea, scăderea, înmulțirea, cîtul și restul.

Program

```
#include <stdio.h>

void main()
{
```

```
int a, b, rez;
char op;
short ok = 1;
scanf("%d %d %c", &a, &b, &op);
switch( op )
{
    case '+': rez = a + b; break;
    case '-': rez = a - b; break;
    case '*': rez = a * b; break;
    case '/': if(b) rez = a/b;
                else{printf("Al doilea operand nul la impartire!");
                      return;}
                break;
    case '%': if(b) rez = a%b;
                else{printf("Al doilea operand nul la impartire!");
                      return;}
                break;
    default: ok = 0;
}
if(ok) printf("%d %c %d = %d", a, op, b, rez);
else printf("Operator invalid!");
```

Analiza programului

- se declară numerele întregi a, b și rez de tip int, reprezentând operanții și rezultatul, caracterul op reprezentând operația care se va executa și variabila de tip short, care va rămâne 1 dacă operația se termină cu succes, altfel i se va atribui 0;
- vom scrie în continuare o instrucție de selecție cu variante, în funcție de operatorul citit; dacă acesta este +, - sau *, se va calcula în rez valoarea corespunzătoare și se va ieși din instrucție; dacă este / sau %, atunci se testează dacă al doilea operand nu este 0; dacă nu, atunci se depune în rez valoarea corespunzătoare, dacă este 0 se va scrie un mesaj și se va ieși din program. Dacă operatorul este altul decât cei indicați mai sus, atunci i se va atribui lui ok valoarea 0;
- dacă ok a rămas 1, atunci se va scrie operația efectuată cu tot cu rezultat, altfel se va scrie un mesaj.

Efect

Varianta 1	Varianta 2
9 0 / Al doilea operand nul la impartire!	3 8 * 3 * 8 = 24

Observație

Încercați să eliberați din program instrucțiunile break și rulați-l apoi pe mai multe exemple. Ce constatați?

12.6. Testați dacă un număr natural dat este prim. (Prin număr prim înțelegem orice număr natural care se împarte doar la 1 și la el însuși; prin definiție, 1 nu este număr prim.)

Program

```
#include <stdio.h>
#include <math.h>

int prim(unsigned long n)
{
    unsigned long i;
    if(n<=1) return 0;
    for(i=2; i*i<=n; i++)
        if(n % i == 0) return 0;
    return 1;
}

main()
{
    unsigned long n;
    printf("N = ");
    scanf("%ld", &n);
    if( prim(n) )printf("Numarul %ld este prim!", n);
    else printf("Numarul %ld nu este prim!", n);
    return 0;
}
```

Analiza programului

- pe primele două linii ale programului se includ headerele stdio.h și math.h, primul pentru funcțiile printf și scanf, al doilea pentru funcția sqrt (radicalul utilizat în funcția prim);
- urmează declararea și implementarea funcției prim, care are ca parametru un întreg fără semn pe 4 octeți și returnează 0 dacă numărul respectiv nu este prim și 1 dacă numărul este prim. Dacă n este mai mic sau egal cu 1, se părăsește funcția cu valoarea 0; dacă nu, vom parcurge cu ajutorul unei instrucțiuni for toate numerele naturale cuprinse între 2 și \sqrt{n} ; dacă n se divide cu vreunul dintre ele (restul împărțirii lui n la i este 0), atunci se părăsește funcția cu valoarea 0 (n nu este prim). Dacă s-a terminat instrucțiunea for și nu am părăsit funcția înseamnă că numărul n este prim, deci se va returna valoarea 1;
- urmează funcția principală main, în care declarăm un număr întreg N de tip unsigned long, apoi apelăm funcția prim prin intermediu unei instrucțiuni if.

Efect

Varianta 1	Varianta 2
N = 79 Numărul 79 este prim!	N = 3453537 Numărul 3453537 nu este prim!

Observație

Este indicat ca în cadrul unei funcții să nu se folosească mai multe instrucțiuni return; în acest caz, funcția prim se poate scrie:

```
int prim(unsigned long n)
{
    unsigned long i;
    int p = 1;
    if(n<=1)p = 0;
    for(i=2; *i<=n; i++)
        if(n % i == 0) p = 0;
    return p;
}
```

12.7. Dat fiind un tablou unidimensional cu numere întregi, determinați minimul și maximul din acest tablou.

Program

```
#include <stdio.h>

int min(int x, int y)
{
    return x < y ? x : y;
}

int max(int x, int y)
{
    return x > y ? x : y;
}

void main()
{
    int a[100];
    int n, i;
    int MIN, MAX;
    printf("Dimensiunea tabloului: "); scanf("%d", &n);
    printf("Introduceti elementele: ");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
    MIN = a[0];
    MAX = a[0];
    for(i=1; i<n; i++)
    {
        if(a[i] < MIN) MIN = a[i];
        if(a[i] > MAX) MAX = a[i];
    }
    printf("Numarul minim este %d\n", MIN);
    printf("Numarul maxim este %d\n", MAX);
}
```

```

MIN = MAX = a[0];
for(i=1; i<n; i++)
{
    MIN = min(MIN, a[i]);
    MAX = max(MAX, a[i]);
}
printf("Minimul din tablou este %d!\n", MIN);
printf("Maximul din tablou este %d!", MAX);
}

```

Analiza programului

- funcțiile min și max calculează minimul și maximul dintre două numere întregi; dacă valoarea expresiei din fața semnului „?” este nenulă, valoarea returnată va fi cea de după semnul „?”, altfel valoarea returnată va fi cea de după semnul „:”;
- urmează declararea unui tablou unidimensional a cu maximum 100 de elemente, a două variabile întregi n și i (dimensiunea tabloului, respectiv o variabilă-contor necesară pentru parcurgere) și a două variabile MIN și MAX reprezentând valorile minimă, respectiv maximă ale tabloului;
- urmează citirea tabloului (mai întâi dimensiunea și apoi elementele); observăm că elementele tabloului se referă cu a[i], unde i parurge sirul 0, 1, ..., n-1;
- în loc de &a[i] se poate folosi și expresia a+i, întrucât aceasta este adresa elementului al i-lea;
- urmează o instrucție de atribuire și, întrucât atribuirea acționează de la dreapta la stânga, se poate scrie în acest fel (MIN și MAX se inițializează cu valoarea a[0]);
- apoi, cu ajutorul instrucției for, se parcurg elementele de la 1 la n-1 și se reactualizează valorile MIN și MAX;
- afișarea numerelor obținute se realizează cu ajutorul a două apeluri ale funcției printf. Observăm că în primul apel, la sfârșitul sirului de formatare, apare secvența \n; această secvență este caracterul newline și are ca efect trećerea la o linie nouă.

Efect

Varianta 1	Varianta 2
Dimensiunea tabloului: 7	Dimensiunea tabloului:1
Introduceti elementele:	Introduceti elementele: 23
45 -90 12 67 13 0 -3	Minimul din tablou este 23!
Minimul din tablou este -90!	Maximul din tablou este 23!
Maximul din tablou este 67!	

12.8. De la tastatură se introduce un sir de caractere care poate să conțină litere și cifre. Să se numere câte cifre, litere mari și litere mici sunt în sir, să se transforme toate literele mici în litere mari și să se afișeze sirul astfel obținut. Să se verifice dacă acest sir este palindrom.

Program

```

#include <stdio.h>
#include <ctype.h>
#include <string.h>

void main()
{
    char s[200];
    int nc = 0, nl = 0, nL = 0;
    short palindrom = 1;
    int i, n;
    printf("Sirul: ");
    scanf("%s", s);
    n = strlen(s);
    for(i=0; i<n; i++)
    {
        if( isdigit(s[i]) ) nc++;
        if( islower(s[i]) ) nl++;
        if( isupper(s[i]) ) nL++;
        s[i] = toupper(s[i]);
    }
    for(i=0; i<n/2; i++)
        if(s[i] != s[n-1-i]) {palindrom=0; break;}
    printf("Numarul de litere mari este %d!\n", nL);
    printf("Numarul de litere mici este %d!\n", nl);
    printf("Numarul de cifre este %d\n", nc);
    printf("Sirul cu litere mari este %s\n", s);
    if(palindrom)printf("Sirul este palindrom!");
    else printf("Sirul nu este palindrom!");
}

```

Analiza programului

- primele trei linii includ headerele stdio.h pentru funcțiile de intrare/iesire, ctype.h pentru funcțiile isdigit, islower, isupper, toupper și string.h pentru apelul funcției strlen (lungimea sirului);
- se declară sirul cu maximum 200 de caractere s, variabilele întregi nc, nl și nL, care se inițializează cu 0 și vor conține numerele de caractere cifră, literă mică, respectiv literă mare din sir, variabila palindrom, care se inițializează cu 1 (dacă vom constata că două caractere simetrice în sir diferă, i se va atribui 0);
- după citirea sirului urmează o instrucție de atribuire în care lui n i se atrbuie lungimea lui s;
- cu ajutorul unei instrucții for se va parcurge sirul (de la 0 la n-1); vom testa dacă elementul curent al sirului este literă mare, mică sau cifră și se va reactualiza variabila corespunzătoare prin incrementare, apoi se va transforma caracterul i în literă mare corespunzătoare prin apelul funcției toupper din headerul ctype.h;

- urmează o instrucțiune `for` în cadrul căreia vom compara elementele simetrice din sir (`s[0]` cu `s[n-1]`, `s[1]` cu `s[n-2]`, ..., `s[i]` cu `s[n-1-i]`...), parcurgem caracterele sirului până la jumătate și dacă găsim o pereche simetrică în care valorile nu sunt egale, îi atribuim variabilei `palindrom` valoarea 0 și părăsim ciclul `for`;
- urmează instrucțiunile de afișare.

Efect

Varianta 1	Varianta 2
Sirul: Co2j2oc	Sirul: ABC12asdc345
Numarul de litere mari este 1!	Numarul de litere mari este 3!
Numarul de litere mici este 4!	Numarul de litere mici este 4!
Numarul de cifre este 2	Numarul de cifre este 5
Sirul cu litere mari este CO2J2OC	Sirul cu litere mări este ABC12ASDC345
Sirul este palindrom!	Sirul nu este palindrom

12.9. Afișarea pe ecran a conținutului unui fișier cu numele dat de la tastatură.

Program

```
#include <stdio.h>

void main()
{
    char c, s[16];
    FILE *f;
    printf("Numele fisierului: ");
    scanf("%s", s);
    f = fopen(s, "r");
    while ( f && !feof(f) )
    {
        c = fgetc(f);
        putc(c, stdout);
    }
    fclose( f );
}
```

Analiza programului

- se vor declara variabilele `c` (caracterul curent citit din fișier și scris pe ecran), `s` (sir cu maximum 16 caractere reprezentând numele fișierului care se introduce de la tastatură), `f` (variabilă de tip `stream` pentru a citi din fișierul dorit);
- în continuare, se citește numele fișierului și se deschide `f` pentru citire;
- urmează o instrucțiune repetitivă cu test inițial: cât timp `f` există și nu s-a depistat încă sfârșitul fișierului, se citește un caracter din fișier și se scrie pe ecran;
- se închide `stream-ul` `f`.

12.10. Scrierea parametrilor din linia de comandă.

Program

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int contor;
    puts("Argumentele programului sunt:");
    for (contor=0; contor<argc; contor++)
        puts(argv[contor]);
    return 0;
}
```

Analiza programului

- se va declara variabila `contor` de tip `int`, cu ajutorul căreia vom parcurge elementele vectorului `argv[]` reprezentând parametrii introdusi;
- în cadrul unui ciclu `for` vom parcurge toate elementele vectorului `argv[]` și le vom afișa pe ecran utilizând funcția `puts()`, ce scrie un sir de caractere la ieșirea standard.

12.11. Utilizarea unor pointeri care reprezintă o matrice, deplasarea și accesarea elementelor, generarea de numere aleatorii, funcții definite de utilizator. Se cere popularea unei matrice cu elemente numere aleatorii mai mici decât 1.000, matricea fiind reprezentată de un pointer care să fie alocat (după citirea de la tastatură a numărului de linii, respectiv coloane). Să se scrie o metodă pentru afișarea în formă pătratică aliniată a matricei, dar și o formă de afișare care să parcurgă elementele pointerului liniar de la stânga la dreapta, prin folosirea operatorului de incrementare. Să se verifice dacă alocarea memoriei pentru matrice a fost realizată cu succes; în caz contrar, să se scrie un mesaj corespunzător și să se întrerupă programul. Exemple de execuție:

1.

Numar linii: 2

Numar coloane: 4

Matricea este:

941	526	247	934
986	1	418	361

Cele 8 valori de la adresa lui a:

941 526 247 934 986 1 418 361

2.

Numar linii: 35346546

Numar coloane: 575675675

Eroare la alocarea memoriei!

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void UmpleMatrice(int *a, int m, int n){
    int i, j;
    srand( (unsigned)time( NULL ) );
    for( i=0; i<m; i++ )
        for( j=0; j<n; j++ )
            *(a+(i*n)+j) = (int)(rand()*1000/RAND_MAX);
}

void ScrieMatrice(int *a, int m, int n){
    int i, j;
    printf( "\n Matricea este: \n" );
    for( i=0; i<m; i++ ){
        for( j=0; j<n; j++ ){
            printf( "%6d", *(a+(i*n)+j) );
        }
        printf( "\n" );
    }
}

void ScrieValoriA( int *a, int m, int n )
{
    int i;
    printf( "\nCele %d valori de la adresa lui a: \n", m*n );
    for( i=0; i<m*n; i++, a++ ){
        printf( "%d ", *a );
    }
}

int main()
{
    int *a;
    int m, n;
    printf( "Numar linii: " ); scanf( "%d", &m );
    printf( "Numar coloane: " ); scanf( "%d", &n );
    a = (int*) malloc ( m*n*sizeof(int) );
    if( a == NULL ) {
        printf( "Eroare la alocarea memoriei!" ); exit(1);
    }
}
```

```
    else{
        UmpleMatrice( a, m, n );
        ScrieMatrice( a, m, n );
        ScrieValoriA( a, m, n );
    }
}
```

Analiza programului

Dacă reprezentarea matematică a matricei este :

$$\begin{matrix} a_{00} & a_{01} & \dots & a_{0,n-1} \\ a_{10} & a_{11} & \dots & a_{1,n-1} \\ \dots \\ a_{m-1,0} & a_{m-1,1} & \dots & a_{m-1, n-1} \end{matrix}$$

atunci reprezentarea sa în memorie va fi :

$$a_{00}|a_{01}|...|a_{0,n-1}|a_{10}|a_{11}|...|a_{1,n-1}|...|a_{m-1,0}|a_{m-1,1}|...|a_{m-1, n-1}$$

Adică, în notație liniară, ele sunt :

$$e_0|e_1|...|e_{n-1}|e_n|e_{n+1}|...|e_{2n-1}|...|e_{1*(m-1)}|e_{1*(m-1)+1}|...|e_{m*n-1}$$

Adresa primului element $a[0][0]$ este chiar a , a următorului element, $a[0][1]$, este $a+1$ și aşa mai departe. În total, sunt $m \times n$ locații de memorie și deducem că elementul $a[i][j]$ este în locația cu numărul $(i*n+j)$. De acest lucru foarte important trebuie să ținem seama atunci când vom aloca elementele în procesul de populare a matricei, dar și pentru afișare.

Programul principal

- observăm includerea bibliotecii <stdlib.h>, care conține metoda pentru alocare dinamică `malloc()`, și a funcției `rand()` pentru generarea de numere aleatorii;
- de asemenea, este inclusă și biblioteca <time.h>, necesară pentru pornirea generatorului de numere aleatorii prin apelul :

```
strand( (unsigned)time( NULL ) );
```

- corpul funcției `main()` începe cu declararea variabilelor necesare : pointerul a de tip `int`, care va desemna adresa de început a matricei și numărul de linii, respectiv de coloane, m, n ;
- următorul pas este extrem de important, anume alocarea memoriei pentru a :

```
a = (int*) malloc ( m*n*sizeof(int) );
```

- aşa cum am văzut la capitolul despre pointeri, un program bun trebuie să testeze dacă memoria cerută a fost cu succes alocată, în caz contrar sistându-se execuția programului;
- în cazul alocării cu succes a memoriei, va trebui să umplem acest spațiu cu valorile care ne interesează, iar acest lucru îl va executa metoda definită de noi, `UmpleMatrice()` ;

- matricea generată va fi afișată de două ori succesiv folosind metodele diferite `ScrieMatrice()`, care afișează a sub formă de matrice, și `ScrieValoriA()`.

Funcția `UmpleMatrice()`

- are ca parametri un pointer la `int a`, reprezentând matricea de populat, și `m, n`, reprezentând numărul de linii, respectiv de coloane ale acesteia;
- tipul returnat este `void` deoarece în corpul metodei se va modifica chiar parametrul `a`;
- se lansează generatorul de numere aleatorii apelat de `rand()` ulterior:

```
| srand( (unsigned)time( NULL ) );
```

- (prin eliminarea acestei linii vom genera întotdeauna aceleași numere);
- urmează o iterație dublă pentru parcurgerea tuturor elementelor matricei, de fiecare dată atribuindu-i-se elementului a_{ij} sau, cum am văzut mai sus, $*(a+(i*n)+j)$ (elementul a_{ij} din matricea în formă matematică este elementul cu numărul $i*n+j$ în memorie):

```
| * (a+(i*n)+j) = (int) (rand()*1000/RAND_MAX);
```

Remarcăm ajustarea cu `1000/RAND_MAX` deoarece `rand()` returnează un număr întreg cuprins între 0 și `RAND_MAX`, iar noi dorim numere mai mici decât 1000.

Funcția `ScrieMatrice()`

- are ca parametri un pointer la `int a` reprezentând matricea de afișat și `m, n` reprezentând numărul de linii, respectiv de coloane ale acesteia;
- scrie o linie goală, mesajul „Matricea este:” și din nou o linie goală prin apelul:

```
| printf( "\n Matricea este: \n" );
```

- pentru fiecare element imaginar al matricei, de sus în jos și de la stânga la dreapta, se scriu numerele a_{ij} prin linia:

```
| printf( "%6d", *(a+(i*n)+j) );
```

(remarcăm că fiecare element se scrie pe șase caractere, aliniat la dreapta, folosind formatarea `%6d`; după fiecare linie se trece la linie nouă prin `printf("\n");`).

Funcția `ScrieValoriA()`

- are aceiași parametri ca și funcția anterioară și face în principiu același lucru, cu diferența că afișarea se execută liniar prin incrementarea adresei pointerului `a`;
- înțînd cont că elementele sunt în secvență consecutivă în memorie și că avem $m \times n$ valori, folosind o instrucțiune `for()` cu al treilea câmp instrucțiune compusă, vom înainta concomitent cu indexul `i` spre dreapta și vom scrie valoarea de la adresa `a` prin:

```
| for( i=0; i<m*n; i++, a++ ){
    printf( "%d ", *a );
}
```

O serie de exerciții legate de această problemă

1. Eliminați din program testul dacă cererea de alocare a memoriei a fost rezolvată cu succes și testați apoi cu valori foarte mari pentru `m` și `n`. Ce se întâmplă?
2. Revenind la programul inițial, eliminați linia:

```
| srand( (unsigned)time( NULL ) );
| si rulați de câteva ori programul. Ce remarcăți?
```

3. Implementați metoda `UmpleMatrice()` doar cu un singur `for()`, ghidându-vă după funcția `ScrieValoriA()`.
4. Adăugați în funcția `ScrieValoriA()`, după primul `for()`, următoarele:

```
| a--;
| printf( "\nCele %d valori ale lui a: \n", m*n );
| for( i=0; i<m*n; i++, a-- ){
|     printf( "%d ", *a );
| }
```

Ce se întâmplă? Explicați. Ce se întâmplă dacă în loc de `a--` în `for()` se seriează `a -= 2`? Modificați și executați programul. Ce se întâmplă?

13. Întrebări, exerciții și probleme propuse

1. Scrieți câteva caracteristici ale limbajului C. Cine sunt creatorii lui? Ce alte limbaje celebre îl au ca bază?
2. Ce caracter are codul ASCII cu valoarea 0? Poate fi acest caracter tipărit? Pentru ce este folosit?
3. Ce este un identificator? La ce este folosit? Este 234sortare un identificator?
4. Definiți un tip de date și proprietățile acestuia. Ce fel de tipuri de date există? Dați exemple.
5. Definiți noțiunea de variabilă. Este un tablou o variabilă? Argumentați.
6. Ce sunt comentariile? Pentru ce sunt ele folosite?
7. Definiți noțiunea de expresie. Ce este un operand? Dar un operator? Dați exemple.
8. Descrieți structura generală a unui program C. Ce sunt directivele de procesare?
9. Ce este o bibliotecă? Dați exemple.
10. Definiți noțiunea de funcție. Care sunt anteturile posibile pentru funcția `main()`?
11. Dați exemple de instrucțiuni vide și expresii. Care este sintaxa instrucțiunii `if`? Care este instrucțiunea repetitivă cu `test inițial și număr necunoscut de pași`? Dar cea cu `număr cunoscut de pași`? Dar instrucțiunea repetitivă fără `test inițial`, cu `număr necunoscut de pași și cu test final`?
12. Descrieți instrucțiunile `switch`, `break` și `continue`. Dați exemple.
13. Care sunt funcțiile de scriere și citire cu format? În ce bibliotecă se află ele?
14. Referitor la fișiere: funcțiile de citire/scriere, verificarea sfârșitului de fișier.
15. Cum putem grupa împreună mai multe tipuri de date folosind tipul `struct`? Putem defini `structuri în structuri`? Dați astfel de exemple.
16. Ce este tipul enumerare? Cum putem redefini tipurile? Ce este `union`?

17. Presupunem următoarea declarație :

```
typedef struct punct {
    double x, y;
} PUNCT;
```

Determinați valoarea de adevăr a următoarelor propoziții :

- A. Secvența PUNCT p; p->x=5; este corectă.
- B. Secvența PUNCT p; p.x=5; este corectă.
- C. struct punct p; p.y=45; nu este corectă.
- D. punct p; p.y = 56.78; este corectă.
- E. struct PUNCT p; p.x = p.y; este corectă.

Scrieți un program-test care să decidă dacă ați răspuns corect.

18. Cum se poate determina tipul de date curent stocat într-o variabilă de tip union ?
 19. Determinați valoarea de adevăr pentru următoarele propoziții :

- A. Într-un același tablou se pot stoca tipuri diferite de date.
- B. int a[20]; a[20] = 56; este corectă.
- C. float a[20]; a[-1] = 4.52; este corectă.
- D. int a[2] = {1, 2, 3}; este corectă.
- E. int a[2] = {1}; nu este corectă.
- F. float temp[300] = {5.0, 6.7};

În acest caz, ultimele 298 de valori sunt inițializate cu 0.

- G. Tablourile de caractere pot fi manipulate cu ajutorul funcțiilor din biblioteca string.h.

- H. Un pointer este o variabilă care conține o adresă de memorie.

- I. Dacă i este adresat de pointerul p, adică p = &i;, atunci i și *p semnifică aceeași valoare.

- J. Relativ la declarația int *p;, decucem că p este un int.

- K. Relativ la declarația int *p;, decucem că *p este un int.

- L. Prin declarația unui pointer se alocă spațiu în memorie pentru adresă, dar nu și pentru variabila spre care el pointează.

- M. Prin declarația unui pointer se alocă spațiu în memorie pentru adresă, dar și pentru variabila spre care el pointează.

- N. Pointerii permit alocarea și dealocarea dinamică a memoriei în timpul execuției.

- O. Alocarea dinamică a memoriei pentru un pointer se poate face cu funcția malloc(n), care alocă n octeți de memorie.

- P. Eliberarea memoriei se poate face utilizând funcția free(p).

- Q. Prin malloc(2000) se alocă memorie pentru 2.000 de numere întregi.

- R. Prin malloc(sizeof(int)) se alocă memorie pentru 2.000 de întregi.

- S. Elementele unui tablou sunt amplasate în spații succesive în memorie.

- T. Un pointer nu este o variabilă; dacă a[] este un tablou și p un pointer, atunci nu putem scrie p=a; p++;

- U. Un tablou este o variabilă; dacă a[] este un tablou și p un pointer, atunci putem scrie a=p și a++.

Algoritmi elementari

24 de probleme complet rezolvate, 26 de programe C, 78 de exerciții și probleme propuse. Probleme rezolvate :

- divizibilitate prin scăderi
- algoritmul lui Euclid
- test de primalitate
- numerele lui Fibonacci
- poziția punctului în cerc, distanța minimă între puncte
- dimensiunea spațiului de memorie
- media aritmetică, sir recurrent
- funcție sinonimă cu atol
- informații despre caracter
- palindrom și suma cifrelor
- radical infinit, serie de valoare π
- instrucțiunea de decizie ?:
- perechi speciale
- seria Farey, divizori comuni
- transformare dintr-o bază în baza 10
- formatarea numerelor naturale
- numere perfecte, numere prietene
- suma cuburilor
- codurile ASCII

Ceea ce a mai fost aceea va mai fi, și ceea ce s-a întâmplat se va mai petrece: căci nu este nimic nou sub soare.

Ecclesiastul 1,9

Problema 1. Divizibilitate prin scăderi

Scrieți un program care să determine dacă un număr natural p este divizibil cu un număr natural q utilizând doar operații de scădere. Ambele numere sunt introduse de la tastatură și sunt diferite de 0.

Exemplu :

```
Introduceti numerele:  
p = 45  
q = 5  
45 se divide cu 5 !!
```

Analiza problemei și proiectarea soluției

După citirea celor două numere de la tastatură, vom putea să determinăm dacă p se divide cu q folosind *metoda scăderilor succesive*. Aceasta înseamnă că vom scădea pe q din p de atâtea ori de câte este posibil. Dacă rezultatul obținut în p în urma acestor scăderi successive este 0, atunci putem afirma că p se divide cu q . Altfel, vom afirma contrariul. În programul următor vom considera în plus și variabila p_1 , care se va initializa cu p la citire deoarece valoarea inițială a lui p va fi modificată în cadrul operațiilor de scădere.

Program

```
#include <stdio.h>  
int main(){  
    unsigned p, q, p1;  
    printf("Introduceti numerele: \np = ");  
    scanf("%d", &p); p1 = p;  
    printf("q = ");  
    scanf("%d", &q);  
    while(p1 >= q)  
        p1 -= q;  
    if (!p1) printf("%d se divide cu %d !!\n", p, q);  
    else printf("%d nu se divide cu %d !!\n", p, q);  
}
```

Exerciții

- Modificați programul anterior încât să facă și o verificare dacă cele două numere sunt strict pozitive.
- Modificați programul astfel încât să afișați și rezultatul împărțirii lui p la q, în cazul în care q divide pe p.

Problema 2. Algoritmul lui Euclid

Implementați algoritmul lui Euclid, prezentat în capitolul 2. Pentru două numere naturale ce se încadrează în tipul `unsigned long`, determinați cel mai mare divizor comun.

Exemplu :

```
a = 73356
b = 326
-----
cmmdc(73356, 326) = 2
```

Analiza problemei și implementarea soluției

Vom simula pașii algoritmului descris: cât timp b nu este zero, i-l vom atribui pe b lui a și pe (a modulo b) lui b. Remarcăm că, deoarece a și b inițial se modifică, ei trebuie salvați în variabile auxiliare.

Program

```
#include <stdio.h>

void main(){
    unsigned long a, b, r;
    unsigned long auxa, auxb;
    printf("a = "); scanf("%ld", &a);
    printf("b = "); scanf("%ld", &b);
    auxa = a; auxb = b;
    while( b != 0){
        r = a % b;
        a = b;
        b = r;
    }
    printf("-----\n");
    printf("cmmdc(%ld, %ld) = %ld", auxa, auxb, a);
}
```

Exerciții

- Ce se întâmplă dacă bucla `while()` va arăta astfel:

```
while( b != 0){
    a = b;
    b = r;
}
```

- Scriți un program care determină cel mai mare divizor comun a două numere naturale folosind următoarea metodă prin scăderi:

```
While a ≠ b
    If a > b
        Then a = a - b
    Else b = b - a
End While
cmmdc = a
```

Problema 3. Test simplu de primalitate

Să se testeze dacă un număr natural este prim. Presupunem că numărul se încadrează în `unsigned long`. În cazul când numărul dat nu este prim, să se afișeze cel mai mic divizor propriu al său.

Exemplu :

n = 35347 35347 NU este număr prim! Primul divizor propriu = 13	n = 7919 7919 este prim!
-----------------------------------------------------------------------	-----------------------------

Analiza problemei și proiectarea soluției

Vom folosi o variabilă auxiliară k, inițializată cu 2, care se va incrementa succesiv. La început, presupunem că numărul dat este prim, prin folosirea variabilei `prim`, după care k va lua toate valorile până la radical din n.

Program

```
#include <stdio.h>
#include <math.h>

void main(){
    unsigned long n, k;
    short prim = 1;
    printf("n = "); scanf("%ld", &n);
    k = 2;
    while( prim && k*k<=n ){
        prim = (n % k != 0);
        k++;
    }
}
```

```

if( prim ) printf( "\n%d este prim!", n );
else{
    printf( "\n%d NU este numar prim!", n );
    printf( "\nPrimul divizor propriu = %d", --k );
}

```

Exerciții

- Înlocuiți bucla `while()` cu una `for()`, apoi cu una `do{}while()`, păstrând funcționalitatea programului.
 - Ce se întâmplă dacă instrucțiunea `while()` se substituie cu aceasta:
- ```
while((prim=(n%k++!=0)?1:0) && k*k<=n);
```
- Explicați.
- Este suficient să testăm doar pentru 2 și numerele impare. Îmbunătățiți algoritmul astfel încât să se ia în considerare această afirmație.

### Problema 4. Distanță minimă între puncte

Se consideră un punct  $P_0$  în plan și o mulțime  $A$  cu  $n$  puncte. Realizați un program care determină cel mai apropiat punct de  $P_0$  din mulțimea  $A$  și distanța corespunzătoare. Datele de intrare se vor citi de la tastatură și rezultatele se vor afișa pe ecran.

Exemplu :

```

Punctul P0:
x = 5.43
y = 3.21
Numarul de puncte din mult A: 3
Primerul punct: x = 1.2
y = 2.43
Punctul 2:
x = 5.43
y = 1.2
Punctul 3:
x = 6.54
y = 3.21
Punctul cel mai apropiat este (6.54, 3.21).
Distanța intre P0 si acesta este 4.301.

```

### Analiza problemei și proiectarea soluției

Vom prezenta în continuare două variante de rezolvare a problemei utilizând facilitățile limbajului C și același algoritm. Citim întâi punctul  $P_0$ , apoi, pe rând, toate cele  $n$  puncte. Vom păstra în variabila  $dist$  distanța minimă curentă (găsită până la prelucrarea punctului  $i$ ). Inițial această distanță este cea dintre  $P_0$  și primul punct. În cadrul primei variante vom lucra fără a folosi tipuri de date și funcții definite de utilizator. În cadrul

celei de-a doua variante vom utiliza tipul `TPunct` de tip structură pentru reprezentarea punctelor, plus funcțiile cu anteturile :

```
double Dist(TPunct p, TPunct q);
```

și respectiv

```
int CitestePunct(TPunct *p);
```

pentru determinarea distanței dintre două puncte date de tip `TPunct`, respectiv pentru citirea unui punct de acest tip.

Observați operatorul de adresă utilizat atunci când citim un punct cu ajutorul funcției `CitestePunct()`. Complexitatea algoritmului este  $O(n)$ , datorită buclei `for`, care parcurge toate punctele.

### Program varianta 1

```

#include <stdio.h>
#include <math.h>

int main(){
 double x0, y0, x1, y1, x, y;
 double dist, d;
 int n, i;
 printf("Punctul initial: \nx = ");
 scanf("%lf", &x0);
 printf("y = ");
 scanf("%lf", &y0);
 printf("Numarul de puncte: ");
 scanf("%d", &n);
 printf("Introduceti punctele: \n");
 printf("Primerul punct: \nx = ");
 scanf("%lf", &x); x1 = x;
 printf("y = "); scanf("%lf", &y); y1 = y;
 dist = sqrt((x0 - x)*(x0 - x) + (y0 - y)*(y0 - y));
 for(i = 2; i <= n; i++) {
 printf("Punctul %d: \nx = ", i);
 scanf("%lf", &x);
 printf("y = ");
 scanf("%lf", &y);
 d = sqrt((x0 - x)*(x0 - x) + (y0 - y)*(y0 - y));
 if (d < dist){
 x1 = x;
 y1 = y;
 dist = d;
 }
 }
}

```

```

printf(
 "Distanta minima este %.4lf, corespunzătoare punctului
 (%.2lf, %.2lf)\n",
 dist, x1, y1);
return 0;
}

```

### Program varianta 2

```

#include <stdio.h>
#include <math.h>

typedef struct{
 double x, y;
}TPunct;

double Dist(TPunct p, TPunct q){
 return
 sqrt((p.x - q.x)*(p.x - q.x) + (p.y - q.y)*(p.y - q.y));
}

int CitestePunct(TPunct *p){
 printf("x = "); scanf("%lf", &p->x);
 printf("y = "); scanf("%lf", &p->y);
 return 0;
}

int main(){
 int n, i;
 TPunct P0, P1, P;
 double dAux, dist;
 printf("Punctul P0: \n"); CitestePunct(&P0);
 printf("Numarul de puncte din mult A: ");
 scanf("%d", &n);
 printf("Primul punct: \n"); CitestePunct(&P1);
 dist = Dist(P0, P1);
 for(i=2; i<=n; i++){
 printf("Punctul %d:\n", i);
 CitestePunct(&P);
 dAux = Dist(P0, P);
 if(dAux < dist) { P1 = P; dist = dAux; }
 }
 printf(

```

```

 "Punctul cel mai apropiat este (%.2lf, %.2lf).\n",
 P1.x, P1.y);
 printf("Distanta intre P0 si acesta: %.3lf.\n", dist);
}

```

### Exerciții propuse

- Modificați programul de mai sus astfel încât să se citească mai întâi toate datele, apoi să se realizeze prelucrarea acestora și în final să se afișeze rezultatele (folosind un tablou unidimensional cu elemente de tip TPunct).
- Modificați programul de mai sus astfel încât să se determine punctul cel mai depărtat de P0.
- Se consideră o mulțime A de n puncte în plan. Scrieți un program care determină cel mai îndepărtat punct de OX și cel mai îndepărtat punct de OY.

### Problema 5. Dimensiunea spațiului de memorie

În cadrul unei funcții implementate într-un limbaj de programare sunt utilizate n tipuri de date, pentru fiecare tip fiind declarate un număr de variabile de tipul respectiv. Știind că o variabilă de tipul i ocupă p\_i octeți în memorie, să se determine dimensiunea totală a spațiului alocat. Presupunem că dimensiunea totală nu depășește valoarea maximă a timpului long.

#### Exemplu :

```

Numarul de tipuri folosite: 3
Nr de var si dim coresp fiecarui tip:
Variabila 1:
Numar: 1
Dimensiune: 200

Variabila 2:
Numar: 2
Dimensiune: 300

Variabila 3:
Numar: 3
Dimensiune: 400
Dimensiunea spatiului de memorie = 2000 bytes.

```

### Analiza problemei și proiectarea soluției

Vom citi secvențial valorile corespunzătoare numărului de variabile și dimensiunii și vom adăuga la s produsul lor. Variabila s va fi inițializată la declarare cu 0. Complexitatea este liniară O(n).

### Program

```
#include <stdio.h>

int main(){
 int n, i;
 int nv, dim;
 long s = 0;
 printf("Numarul de tipuri folosite: ");
 scanf("%d", &n);
 printf("Nr de var si dim coresp fiecarui tip: ");
 for(i=1; i<=n; i++){
 printf("\nVariabila %d: \nNumar: ", i);
 scanf("%d", &nv);
 printf("Dimensiune: ");
 scanf("%d", &dim);
 s += nv * dim;
 }
 printf("Dimensiunea spatiului de memorie = %ld bytes.\n",
 s);
 return 0;
}
```

### Exercițiu

Presupunând că dimensiunea totală poate depăși valoarea maximă a tipului `long`, modificați programul astfel încât să se afișeze un mesaj corespunzător când se întâmplă acest lucru.

### Problema 6. Numerele lui Fibonacci



*Leonardo Pisano* (?1170-1250, Pisa, Italia) este cunoscut după porecla *Fibonacci*. El a scris opere importante privind diverse aspecte matematice și a adus o serie de contribuții proprii valoroase (*Liber abaci* - 1202, *Practica geometriae* - 1220, *Flos* - 1225, *Liber quadratorum*). Una dintre problemele din *Liber abaci* a dus la apariția celebrelor numere Fibonacci, pentru care este renumit astăzi: *Cineva pune o pereche de iepuri într-un loc împrejmuit de ziduri. Câte perechi de iepuri se pot produce din această pereche într-un an, presupunând că în fiecare lună fiecare pereche produce o nouă pereche care din a doua lună devine productivă?*

Rezultatul a fost secvența: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... (Fibonacci a omis primul număr în *Liber abaci*.) Această secvență s-a dovedit extrem de fructuoasă în multe domenii diferite ale matematicii și științelor, existând chiar o

revistă specializată, *Fibonacci Quarterly*, care se ocupă doar cu aspecte matematice legate de ea.

Scrieți un program care să determine al  $n$ -lea termen din sirul de fracții  $u(n)/u(n+1)$ , unde  $u(n)$  este al  $n$ -lea termen din sirul lui Fibonacci:

$u(0) = u(1) = 1$   
 $u(n) = u(n-2) + u(n-1)$   
 $n (0 \leq n \leq 40)$  se va citi de la tastatură și rezultatul se va scrie pe ecran cu patru zecimale, ca în exemplele următoare.

#### Exemplul 1:

Introduceti indicele raportului: 2  
 $u_2/u_3 = 0.6667$

#### Exemplul 2:

Introduceti indicele raportului: 37  
 $u_{37}/u_{38} = 0.6180$

### Analiza problemei și proiectarea soluției

În cadrul unei bucle `while()` se va execuția, iterativ, de  $n$  ori pasul de definiție al sirului, utilizând  $u_0$  și  $u_1$  ca ultimele două valori calculate.

### Program

```
#include <stdio.h>

void main(){
 long u0 = 1, u1 = 1;
 long uaux;
 int n, naux;
 printf("Introduceti indicele raportului: ");
 scanf("%d", &n); naux = n;
 while(naux){
 naux--;
 uaux = u0 + u1;
 u0 = u1;
 u1 = uaux;
 }
 printf("u%d/u%d = %.4f", n, n+1, (double)u0/u1);
}
```

### Exerciții

1. Scrieți un program care, pentru un număr natural  $n (0 \leq n \leq 40)$  introdus de la tastatură, determină al  $n$ -lea termen din sirul lui Fibonacci.

2. Pentru programul de mai sus afișați toate valorile  $u(i)/u(i+1)$  pentru toți  $i$  de la 0 la 40. Ce observați?

### Problema 7. Poziția punctului în cerc

Se citesc de la intrare următoarele 5 numere reale:  $x$ ,  $y$ ,  $r$ ,  $x_0$ ,  $y_0$ , reprezentând coordonatele centrului unui cerc, raza sa și coordonatele unui punct în plan. Să se scrie un program care să decidă dacă punctul  $(x_0, y_0)$  aparține interiorului, frontierei sau exteriorului cercului  $C(x_0, y_0, r)$ .

| Exemplul 1                                                                                                                                        | Exemplul 2                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Elementele cercului:<br>Centrul:<br>$x = 0$<br>$y = 0$<br>$raza = 2$<br>Punctul P: $x_0 = 1$<br>$y_0 = 1$<br>Punctul este în interiorul cercului! | Elementele cercului:<br>Centrul:<br>$x = -67$<br>$y = -45.71$<br>$raza = 34.56$<br>Punctul P: $x_0 = 1000.78$<br>$y_0 = 678.90$<br>Punctul este; exterior cercului! |

### Analiza problemei și proiectarea soluției

Considerând un punct  $O(x_0, y_0)$  reprezentând centrul unui cerc și  $r$  un număr real pozitiv reprezentând raza cercului, ecuația acestuia este:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

Punctele  $(x, y)$  din plan care satisfac această ecuație se află pe cerc.

Punctele  $(x, y)$  din plan care satisfac inecuația:

$$(x - x_0)^2 + (y - y_0)^2 < r^2$$

se află în interiorul cercului și cele care satisfac inecuația:

$$(x - x_0)^2 + (y - y_0)^2 > r^2$$

se află în exteriorul cercului. Pe baza acestor identități scriem programul de mai jos.

Complexitatea este  $O(1)$ .

### Program

```
#include <stdio.h>
#include <math.h>

void main() {
 float x, y, r, x0, y0;
 double d;
 printf("Elementele cercului:\n ");
 printf("Centrul:\n x = "); scanf("%f", &x);
 printf("y = "); scanf("%f", &y);
 printf("raza = "); scanf("%f", &r);
 printf("Punctul P: \n x0 = "); scanf("%f", &x0);
 printf("y0 = "); scanf("%f", &y0);
 d = sqrt((x-x0)*(x-x0)+(y-y0)*(y-y0));
 if (d>r) printf("Punctul este exterior cercului!");
 else if (d == r)
 printf("Punctul este pe cerc!!!!");
 else
 printf("Punctul este in interiorul cercului!");
}
```

```
printf(" y = "); scanf("%f", &y);
printf(" raza = "); scanf("%f", &r);
printf("Punctul P:");
printf(" x0 = "); scanf("%f", &x0);
printf(" y0 = "); scanf("%f", &y0);
d = sqrt((x-x0)*(x-x0)+(y-y0)*(y-y0));
d -= r;
if (d>0) printf("Punctul este exterior cercului!");
else if(d == 0)
 printf("Punctul este pe cerc!!!!");
else
 printf("Punctul este in interiorul cercului!");
}
```

### Exerciții

Se consideră o elipsă cu centrul de coordonate  $O(0, 0)$ , razele  $a, b \in (0, 1000]$  și un punct oarecare în plan  $P(x_0, y_0)$ . Scrieți un program care determină dacă punctul  $P$  este pe elipsă, în interiorul acesteia sau în exterior.

### Problema 8. Media aritmetică

Scrieți un program care calculează media aritmetică a  $n$  numere reale introduse de la tastatură. Presupunem că media rezultată se încadrează în tipul `float`, ca și elementele date, dar suma elementelor poate fi oricât de mare, n este mai mic decât valoarea `MAX_INT`. Programul nu trebuie să utilizeze tablouri.

Exemplu :

```
Numarul de numere: 7
Numerele: 1 2 3 4 5 6 7
Media aritmetica: 4.000
```

### Analiza problemei și proiectarea soluției

Media aritmetică a numerelor  $a_1, a_2, \dots, a_n$  date se calculează cu formula:

$$\bar{m}_a = \frac{a_1 + a_2 + \dots + a_n}{n} = \frac{a_1}{n} + \frac{a_2}{n} + \dots + \frac{a_n}{n}$$

Deoarece suma elementelor poate fi oricât de mare, vom utiliza cea de a două inegalități.

Complexitatea este liniară  $O(n)$ .

**Program**

```
#include <stdio.h>

int main(){
 float a;
 int n, i;
 float mA = 0;
 printf("Numarul de numere: ");
 scanf("%d", &n);
 printf("Numerele: ");
 for(i=0; i<n; i++){
 scanf("%f", &a);
 mA += a/n;
 }
 printf("Media aritmetica: %.3f\n", mA);
 return 0;
}
```

**Exerciții**

1. Aceeași problemă, cu diferența că se cere determinarea valorii:

$$m_b = \sqrt{\frac{a_1^2 + a_2^2 + \dots + a_n^2}{n}}$$

(valorile  $a_i^2$  se încadrează în tipul **float**, suma  $a_1^2 + a_2^2 + \dots + a_n^2$  poate fi oricât de mare).

2. Scrieți un program care generează aleatoriu numere cu condițiile date și un număr de cazuri de test. Să se verifice în program inegalitatea:

$$m_b \leq m_a$$

prin afișarea unui mesaj corespunzător pentru fiecare caz. Ce observați? Încercați demonstrarea matematică a acestei inegalități.

**Problema 9. Sir recurrent**

Scrieți un program care să calculeze al  $i$ -lea element din sirul:

$$f(0) = 0, f(n) = (f(n-1) + n) / 2$$

*Exemplu :*

$$\begin{aligned} n &= 5 \\ f(5) &= 4.031 \end{aligned}$$

**Analiza problemei și proiectarea soluției**

În cadrul unei bucle **for()** determinăm secvențial elementele sirului și îl afișăm pe ultimul.

**Program**

```
#include <stdio.h>

int main(){
 float f;
 int n, i;
 printf("n = ");
 scanf("%d", &n);
 f=0;
 for(i=1; i<=n; i++)
 f = (f + i)/2;
 printf("f(%d) = %.3f\n", n, f);
 return 0;
}
```

**Exerciții**

Modificați programul astfel încât să fie afișate toate valorile lui  $f(i)$ , pentru  $i$  de la 1 la 100. Ce observați? Demonstrați rezultatul (eventual, prin inducție)!

**Problema 10. Funcție sinonimă cu atol**

În biblioteca **stdlib.h** există o serie de funcții de conversie numerică, de exemplu **atol()**, care convertește un sir în echivalentul să de tip **long**.

Să se scrie o funcție **atol()** proprie care convertește un sir de caractere în echivalentul lui numeric. Considerăm că echivalentul numeric al sirului introdus se încadrează în tipul **long**.

*Exemplu :*

Sirul: -67801  
**atol(-67801) = -67801**

Sirul: t5678  
 Numarul introdus nu este întreg!

**Analiza problemei și proiectarea soluției**

Vom scrie o funcție cu antetul:

**short corect(char[30]);**

care decide dacă sirul introdus poate reprezenta un număr întreg prin parcursarea secvențială a caracterelor.

*Program*

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

long atoi(char[30]);
short corect(char[30]);

void main(){
 char s[30];
 long n = 0;
 int ok = 1;
 printf("Sirul: ");
 scanf("%s", &s);
 if(! corect(s))
 printf("Numarul introdus nu este intreg!");
 else
 printf("atol(%s) = %ld ", s, atol(s));
}

short corect(char s[30]){
 short ok = 1;
 unsigned i;
 if(!isdigit(s[0]) && s[0] != '-' && s[0] != '+')
 return 0;
 for(i=1; ok && i<strlen(s); i++)
 if(!isdigit(s[i]) && i > 0) ok = 0;
 return ok;
}

long atol(char s[30]){
 int semn = 1;
 unsigned i = 0;
 long result = 0;
 if(s[0] == '+') i++;
 else if(s[0] == '-') { i++; semn = -1; }
 while (i < strlen(s)){
 result = result * 10 + (s[i] - '0');
 i++;
 }
 result *= semn;
 return result;
}
```

*Exerciții*

1. Scrieți un program care transformă un număr întreg în sirul corespunzător.
2. Modificați funcția `corect()` de mai sus astfel încât să verifice și dacă numărul introdus se încadrează în tipul de date `long`.

*Problema 11. Informații despre caracter*

Scrieți un program care citește un caracter de la tastatură și afișează: caracterul introdus, codul ASCII al acestuia și adresa la care a fost depus în memorie, ca în exemplul următor.

*Exemplu:*

```
Caracterul: C
Valoarea: C
Codul ASCII: 67
Adresa: 0012FF7C
```

*Analiza problemei și proiectarea soluției*

Se vor folosi formatele de scriere și operatorul de adresă `&`.

*Program*

```
#include <stdio.h>

void main(){
 char c;
 printf("Caracterul: ");
 scanf("%c", &c);
 printf(" Valoarea: %c\n", c);
 printf(" Codul ASCII: %d\n", c);
 printf(" Adresa: %p ", &c);
}
```

*Exercițiu*

Scrieți un program care afișează un tabel cu toate caracterele: valoare și cod ASCII.

*Problema 12. Palindrom și suma cifrelor*

Spunem că un număr natural este *palindrom* dacă este egal cu oglinditul său. Se dă de la tastatură un număr natural care se încadrează în tipul `unsigned long`. Să se decidă dacă numărul dat este palindrom și să se calculeze suma cifrelor numărului dat.

*Exemplul 1*

```
Introduceti numarul: 72027
```

72027 este palindrom!

Suma cifrelor lui 72027 este 18.

*Exemplul 2*

```
Introduceti numarul: 4353425
```

4353425 nu este palindrom!

Suma cifrelor lui 4353425 este 26.

*Analiza problemei și proiectarea soluției*

Pentru a calcula inversul (oglinditul) unui număr natural dat  $n$  vom considera cifrele numărului de la dreapta spre stânga și vom construi corespunzător numărul scris de la dreapta la stânga:

```
while(n){
 m = m*10 + n%10;
 n /=10;
}
```

Pentru a calcula oglinditul unui număr natural dat scriem funcția cu antetul :

```
unsigned long invers(unsigned long n)
```

Pentru a determina suma cifrelor unui număr natural dat scriem funcția cu antetul :

```
short sumaC(unsigned long n)
```

care va aduna toate cifrele numărului dat, secvențial, de la cea mai puțin semnificativă spre cea mai semnificativă.

*Program*

```
#include <stdio.h>

unsigned long invers(unsigned long n){
 unsigned long m = 0;
 while(n){
 m = m*10 + n%10;
 n /= 10;
 }
 return m;
}
```

```
short sumaC(unsigned long n){
 short suma=0;
 while(n){
 suma += (short)(n%10);
 n /= 10;
 }
 return suma;
}

void main(){
 unsigned long n;
 printf("Introduceti numarul: ");
 scanf("%ld", &n);
 if(n == invers(n))
 printf("\n %ld este palindrom!\n", n);
 else printf("\n %ld nu este palindrom!\n", n);
 printf(" Suma cifrelor lui %ld este %d. ",
 n, sumaC(n));
}
```

*Exerciții*

1. Scrieți programul de mai sus fără a folosi funcții definite de utilizator.
2. Scrieți un program care determină toate numerele prime de 4 cifre.
3. Scrieți un program care citește un număr natural  $n$  și se încadrează în tipul `unsigned long` și care determină, fără a folosi operatorul `%`, folosind doar criteriile de divizibilitate, dacă este divizibil cu 2, 3, 4, 5, 9, 25.

*Problema 13. Radical infinit*

Să se găsească valoarea expresiei:

$$\text{radical}(20 + \text{radical}(20 + \text{radical}(20 + \dots)))$$

*Analiza problemei și proiectarea soluției*

Notăm valoarea expresiei de mai sus cu  $s$ .

Rezultă ecuația :

$$20 + s = s^2$$

și singura soluție pozitivă este 5. Deci rezultatul este 5.

*Program*

```
#include <stdio.h>
#include <math.h>
```

```

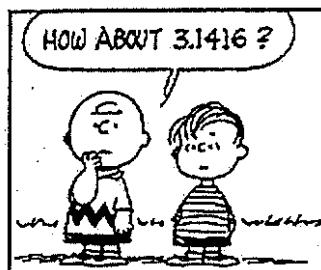
void main(){
 double s = 20;
 while(s != sqrt(20+s))
 s = sqrt(20 + s);
 printf("\nRez = %f !!!\n", s);
}

```

### Exercițiu

Care este ultima cifră a numărului  $7^{(7^7)^7}$ ? Dat fiind un număr natural N cu maximum 9 cifre afișați ultima cifră a numărului  $7^{(N^N)^N}$ .

### Problema 14. Serie cu valoarea $\pi$



Prin definiție,  $\pi$  este raportul dintre circumferința unui cerc și diametrul său (el este tot timpul același număr, indiferent de cerc).  $\pi$  cu ceva mai multe zecimale arată aşa: 3,14159265358979323846. Acest număr este foarte vechi, se știe că vechii egipteni și babilonieni știau despre existența unui raport constant  $\pi$ .  $\pi$  este număr irațional, are un număr infinit de zecimale și nu poate fi scris ca o fracție.  $\pi$  are foarte multe întrebunțări în matematică, și-a găsit chiar locul în teoria probabilităților, ca și în faimoasa „ecuație a celor cinci” (formula lui Euler), ce leagă cele cinci cele mai importante numere:  $e^{i\pi} + 1 = 0$ . Cu ajutorul computerelor s-a calculat  $\pi$  cu foarte multe zecimale. Ziua lui  $\pi$  este celebrată în fiecare an la 14 martie (14 martie este 3/14) la Exploratorium din San Francisco.

Scrieți un program care să calculeze cu precizia EPSILON valoarea lui  $\pi$ , folosind dezvoltarea în serie:

$$\pi = 4 \cdot \left( 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots \right)$$

Valoarea lui EPSILON se va citi de la tastatură. Considerăm că numărul PI este calculat cu precizia EPSILON dacă  $|PI(n+1) - PI(n)| < EPSILON$ , unde  $PI(n)$  și  $PI(n+1)$  sunt două aproximări succesive ale lui  $\pi$ .

*Exemplu :*

```

EPSILON = 0.0001

PI cu precizia 0.000100 = 3.141547

```

### Analiza problemei și proiectarea soluției

Vom considera variabilele întregi v1 și v2, care iau, pe rând valorile  $PI(n)/4$  și  $PI(n+1)/4$  (valorile din paranteză) pentru  $n = 1, 2, 3, \dots$

Valoarea  $|v2 - v1|$  se calculează utilizând operatorul condițional ?: astfel:

$v2 > v1 ? v2-v1 : v1-v2$

În scopul calculării următorului termen din sir folosim o variabilă semn care alternează (-1, +1) și variabila i care ia valorile 5, 7, 9, 11...

### Program

```

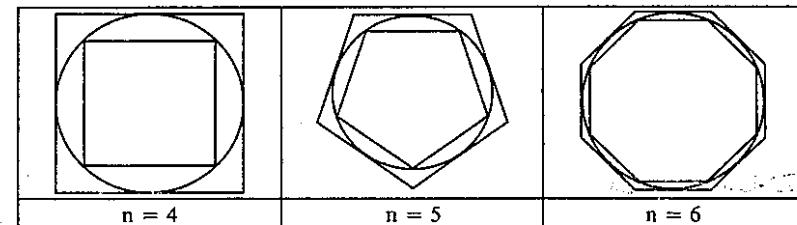
#include <stdio.h>

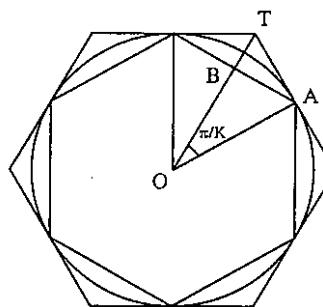
void main(){
 float EPSILON;
 float v1, v2;
 int i = 5, semn = 1;
 printf("EPSILON = ");
 scanf("%f", &EPSILON);
 v1 = 1.0;
 v2 = (float)1.0 - (float)1.0/3;
 while(4*(v2>v1?v2-v1:v1-v2) >= EPSILON){
 v1 = v2;
 v2 += semn * ((float)1.0/i);
 i += 2;
 semn *= -1;
 }
 printf("\n-----\n");
 printf("PI cu precizia %f\n", EPSILON, 4*v2);
}

```

### Exercițiu

- Modificați programul anterior astfel încât să se afișeze și iterarea la care s-a găsit aproximarea corespunzătoare lui EPSILON.
- Metoda lui Arhimede.* Arhimede din Siracuza (287-212 i.e.n.) a fost primul care a determinat o metodă științifică de calcul cu acuratețe a lui  $\pi$ . El a observat că perimetru poligonului regulat cu  $n$  laturi inscris în cerc este mai mic decât lungimea cercului, iar perimetru poligonului regulat circumscris cercului este mai mare decât această lungime. Când  $n$  tinde la infinit, aceste perimetre se apropiie de lungimea cercului. Imaginele următoare exemplifică acest concept:





Metoda folosită de Arhimede este următoarea: se consideră cercul cu raza unitate, în care se înscrie poligonul regulat cu  $3 \times 2^{n-1}$  laturi și semiperimetru  $b_n$  și se circumscrize poligonul regulat cu  $3 \times 2^{n-1}$  și semiperimetru  $a_n$ . Se definesc astfel sirul crescător  $b_1, b_2, b_3, \dots$  și cel descrescător  $a_1, a_2, a_3, \dots$ . Aceste două siruri au ambele limită  $\pi$  pentru  $n$  tînzând la infinit.

$$OA = 1$$

$$AB = \sin(\pi/K)$$

$$AT = \tan(\pi/K)$$

$$\text{unde, } K = 3 \times 2^{n-1}$$

Cu notațiile din figură, deducem identitățile:

$$a_n = K \tan\left(\frac{\pi}{K}\right), b_n = K \sin\left(\frac{\pi}{K}\right), \text{ unde } K = 2^{n-1} \quad (1)$$

Similar se obține:

$$a_{n+1} = 2K \tan\left(\frac{\pi}{2K}\right), b_{n+1} = 2K \sin\left(\frac{\pi}{2K}\right) \quad (2)$$

Folosind identități trigonometrice de bază, deducem:

$$a_{n+1} = \frac{2a_n b_n}{a_n + b_n} \text{ și } b_{n+1} = \sqrt{a_{n+1} b_n} \quad (3)$$

$a_0$  și  $b_0$  sunt semiperimetrele triunghiurilor echilaterale circumscris, respectiv încris cercului de rază 1. Aceste valori inițiale sunt aşadar:  $a_0 = 2\sqrt{3}$  și  $b_0 = 3$ . Algoritmul în pseudocod va fi:

```

ALGORITM_ARHIMEDE
 n ← 1
 a ← 2√3
 b ← 3
 n ← numar_iteratii
 for(i=1; i≤n; step 1) execute
 a ← $\frac{2ab}{a+b}$
 b ← \sqrt{ab}
 end_for
 return a
END_ALGORITM_ARHIMEDE

```

Implementați aceast algoritm pentru determinarea constantei  $\pi$ .

3. Demonstrați, pe baza unor identități trigonometrice, formulele de recurență pentru sirurile  $a_n$ , respective  $b_n$  din cadrul metodei lui Arhimede.
4. Scrieți un program care să reprezinte grafic metoda lui Arhimede.
5. Metoda tangentei. Baza metodei este egalitatea:

$$\tan \frac{\pi}{4} = 1$$

de unde se deduce:  $\pi = 4 \cdot \arctan 1$

Matematicienii James Gregory (1638-1675) și Gottfried Wilhelm Leibniz (1646-1716), care au lucrat independent, au descoperit în jurul lui 1670 următoarea serie infinită pentru  $\arctan$ :

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots, x \in \mathbb{R}, |x| \leq 1,$$

care poate fi rescrisă ca:

$$\arctan x = \sum_{n=0}^{\infty} (-1)^n \left( \frac{x^{2n+1}}{2n+1} \right), x \in \mathbb{R}, |x| \leq 1$$

Experimentele furnizează următoarele rezultate:

| n      | $\approx \pi$      |
|--------|--------------------|
| 0      | 4                  |
| 1      | 2,666666666666667  |
| 2      | 3,466666666666668  |
| 3      | 2,8952380952380956 |
| 4      | 3,3396825396825403 |
| 5      | 2,9760461760461765 |
| 100000 | 3,1416026534897203 |
| 100001 | 3,1415826537897158 |
| 200000 | 3,1415976535647618 |
| 200001 | 3,1415876536397613 |
| 300000 | 3,1415959869120198 |
| 300001 | 3,1415893202786864 |
| 400000 | 3,1415951535834941 |
| 400001 | 3,1415901536022441 |

Din tabelul de mai sus observăm că sirul termenilor de rang par este descrescător și valorile sunt mai mari decât  $\pi$ . Sirul termenilor de rang impar este crescător și valorile sunt inferioare lui  $\pi$ . Scrieți programul pentru determinarea lui  $\pi$  pe baza algoritmului:

```

ALGORITM_ARCTAN
 n ← numar_iteratii
 x ← 1
 semn ← 1
 numerator ← x
 numitor ← 1
 for(i=1; i≤n; step 1) execute
 rez ← semn × $\frac{\text{numerator}}{\text{numitor}}$
 numitor ← numitor * (2*i + 1)
 semn ← -semn
 end_for

```

```

numarator ← numarator × x2
numitor ← numitor + 2
semn ← semn × (-1)
end_for
return 4.0*rez;
END_ALGORITM_ARCTAN

```

6. *Îmbunătățirea lui Machin.* În 1706, profesorul londonez de astronomie John Machin (1680-1751) a descoperit o formulă pentru aflarea mai rapidă a lui  $\pi$  și a folosit-o pentru a-l determina cu 100 de zecimale exacte. Această formulă este:

$\frac{\pi}{4} = 4 \arctan \frac{1}{5} - \arctan \frac{1}{239}$ . El a combinat-o cu o extensie a seriilor Taylor pentru inversa tangentei. Scrieți algoritmul și programul corespunzătoare.

7. *Numărul e.* Dacă numărul  $\pi$  este foarte vechi, nu se poate afirma același lucru și despre numărul  $e$ , care este prin comparație un relativ nou-venit pe scena matematică. Prima dată a intuit existența logaritmilor naturali Napier, atunci când își crea tabelele de logaritmi, apoi pare să apară într-o tabelă de logaritmi scrisă de Oughtred și, câțiva ani mai târziu, în 1624,  $e$  este foarte aproape să se nască, prin aproximarea numerică a logaritmului în baza 10 din  $e$  de către Briggs, dar fără a fi menționat explicit. O altă posibilă apariție pare să fie în 1647, când Saint-Vincent a calculat aria sub o hiperbolă rectangulară. Cu siguranță Huygens în 1661 a înțeles relația dintre hiperbolă rectangulară și logaritmul. În 1668, Nicolaus Mercator publică *Logarithmotechnica*, lucrare ce



contine extensiile ale seriei  $\log(1+x)$ . În aceeași lucrare, Mercator folosește termenul de „logaritmul natural” pentru prima dată pentru logaritmul în baza  $e$ . Dar numărul  $e$  rămâne ascuns după colț, încă nu vrea să se arate... În 1683, Jacob Bernoulli privește problema cu interes și încearcă să demonstreze că limita lui  $(1+1/n)^n$  tinde la infinit, dar nu a recunoscut nici o legătură între munca sa și logaritmi. Prima apariție cunoscută a lui  $e$  este în 1690, când Leibniz i-a scris o scrisoare lui Huygens și a folosit notația  $b$  pentru ceea ce noi numim astăzi  $e$ . Notația  $e$  se datorează lui Euler (nu din cauza primei vocale a numelui său, ci pentru că  $e$  este următoarea vocală după  $a$ , pe care deja o folosise). Euler i-a scris despre acest număr lui Goldbach în 1731, iar ulterior a publicat descoperirile sale în *Introductio in Analysis* în 1748. Acolo a arătat că

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

că limita pentru  $(1 + \frac{1}{n})^n$  când  $n$  tinde la infinit este  $e$ , a dat o aproximare cu 18 zecimale:  $e = 2,718281828459045235$ . Alte rezultate importante sunt legăturile

dintre funcțiile trigonometrice și funcția complexă exponențială, pe care le-a dedus folosind formulele lui Moivre. A mai găsit relația:

$$\frac{e - 1}{2} = \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{10 + \cfrac{1}{14 + \cfrac{1}{18 + \dots}}}}}$$

Să se scrie un program care calculează cu precizia epsilon valoarea numărului  $e$  utilizând formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Considerăm că  $e$  este calculat cu precizia EPSILON dacă  $|e(n+1) - e(n)| < \text{EPSILON}$ , unde  $e(n)$  și  $e(n+1)$  sunt două aproximări succesive ale numărului  $e$ . Afipați și iterația la care s-a determinat aproximarea. Atenție: valoarea  $n!$  crește foarte repede!

8. Similar cu problema de mai sus, scrieți programe care calculează termenul stâng cu precizia epsilon și apoi verifică formulele:

a)  $1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots \pm \frac{1}{n!} \mp \dots = \frac{1}{e}$

b)  $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots \pm \frac{1}{n} \mp \dots = \ln 2$

c)  $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n} + \dots = 2$

d)  $1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \dots \pm \frac{1}{2^n} \mp \dots = \frac{2}{3}$

e)  $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{n(n+1)} + \dots = 1$

f)  $\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \frac{1}{5 \cdot 7} + \dots + \frac{1}{(2n-1)(2n+1)} + \dots = \frac{1}{2}$

g)  $\frac{1}{1 \cdot 3} + \frac{1}{2 \cdot 4} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{(n-1)(n+1)} + \dots = \frac{3}{4}$

h)  $\frac{1}{3 \cdot 5} + \frac{1}{7 \cdot 9} + \frac{1}{11 \cdot 13} + \dots + \frac{1}{(4n-1)(4n+1)} + \dots = \frac{1}{2} - \frac{\pi}{8}$

i)  $\frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \dots + \frac{1}{n(n+1)(n+2)} + \dots = \frac{1}{4}$

j)  $1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots + \frac{1}{n^2} + \dots = \frac{\pi^2}{6}$

k)  $1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \dots \pm \frac{1}{n^2} \mp \dots = \frac{\pi^2}{12}$

### Problema 15. Instrucțiunea de decizie :

Operația matematică  $\min(x, y)$  se poate reprezenta ca o expresie condițională astfel :

$x < y \ ? \ x : y$

Scrieți un scurt program care calculează valoarea  $\min(x, y, z)$  utilizând doar operatorul condițional  $::$ .

*Exemplu :*

Introduceti numerele: 3 1 2

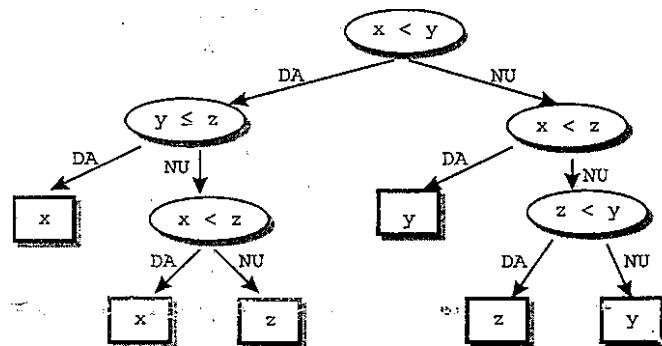
Valoarea minima: 2

### Analiza problemei și proiectarea soluției

Vom scrie expresia condițională :

$x < y \ ? \ (y \leq z \ ? \ x : (x < z \ ? \ x : z)) : (x < z \ ? \ y : (z < y \ ? \ z : y))$

corespunzătoare arborelui de decizie :



### Program

```

#include <stdio.h>

void main(){
 int x, y, z;
 int min;
 printf("Introduceti numerele: ");
 scanf("%d %d %d", &x, &y, &z);
 min = x < y ? (y <= z ? x : (x < z ? x : z)) : (x < z ? y : (z < y ? z : y));
 printf("Valoarea minima: %d\n", min);
}

```

### Exerciții

- Modificați programul anterior astfel încât să se calculeze maximul celor trei numere introduse de la tastatură.
- Scrieți un alt program care să calculeze  $\max(x, y, z, t)$  și  $\min(x, y, z, t)$  pentru patru numere întregi introduse de la tastatură, folosind tot numai operatorul condițional.

### Problema 16. Perechi speciale

Se consideră un număr natural  $n$ ,  $0 < n < 9000$ . Scrieți un program C care :

a) Găsește toate perechile de numere întregi  $(x, y)$  pentru care :

$$x^2 + n = y^2$$

b) Găsește perechea  $(a, b)$  pentru care

$$n = 2^x * (2^y + 1)$$

Formatul datelor de ieșire trebuie să fie ca în exemplul următor.

*Exemplu :*

$$n = 456$$

Perechi punctul a):

- $113^2 + 456 = 115^2$
- $55^2 + 456 = 59^2$
- $35^2 + 456 = 41^2$
- $13^2 + 456 = 25^2$

Există 4 astfel de perechi.

Perechea de la punctul b):

$$456 = 2^3 * (2^3 + 2^3 - 1)$$

### Analiza problemei și proiectarea soluției

Relativ la egalitatea de la punctul a) putem scrie succesiv :

$$x^2 + n = y^2 \longleftrightarrow n = y^2 - x^2 \longleftrightarrow n = (y - x) \cdot (y + x)$$

Așadar  $n$  se scrie ca produs a două numere cu valorile  $y-x$ , respectiv  $y+x$ .

Dacă notăm cu  $a$  și  $b$  doi divizori ai lui  $n$  cu proprietățile:  $a < b$  și  $a \cdot b = n$ , atunci pentru perechea  $(x, y)$  care satisfac ecuația de mai sus putem construi sistemul simplu :

$$\begin{cases} y - x = a \\ y + x = b \end{cases}$$

care are soluțiile :

$$x = \frac{b - a}{2}$$

$$y = \frac{b + a}{2}$$

Pentru ca  $x$  și  $y$  să fie numere întregi mai trebuie adăugată la condițiile de mai sus referitoare la  $a$  și  $b$  și condiția:  $2$  divide pe  $(a+b)$ .

Urmăriți în program cum au fost implementate aceste observații!

Relativ la punctul b),  $x$  este puterea lui  $2$  din descompunerea lui  $n$  în factori primi și se obține corespunzător din cel mai mare număr impar care îl divide pe  $n$ .

### Program

```
#include <stdio.h>
#include <math.h>

int main(){
 unsigned long x, y, n;
 int contor = 0;
 printf("n = "); scanf("%ld", &n);
 printf("\nPerechi punctul a): \n ");
 for(x = 1; x<=sqrt(n); x++)
 if(n%x == 0){
 y = n/x;
 if((x+y)%2 == 0){
 contor++;
 printf("\n %d. ", contor);
 printf("%4ld^2 + %4ld = %4ld^2",
 (y-x)/2, n, (y+x)/2);
 }
 }
 printf("\n Există %d astfel de perechi.", contor);
 x = 0;
 y = n;
 while(y%2 == 0){
 x++;
 y /= 2;
 }
 printf("\n\nPerechea de la punctul b): \n");
 printf(" %ld = 2^%ld * (2 * %ld + 1) ",
 n, x, (y-1)/2);
 return 0;
}
```

### Exerciții

- Modificați programul de mai sus astfel încât să calculați mai întâi, relativ la punctul b), valoarea lui  $y$ .
- Găsiți o altă metodă de rezolvare și pentru punctul a), „fără a ști” raționamentul prezentat mai sus în partea de *analiză a problemei*.

### Problema 17. Seria Farey

Definim seria Farey de ordinul  $n$  ca fiind sirul fracțiilor reduse, cuprinse între  $0$  și  $1$ , cu numitorul  $\leq n$ . De exemplu, seria Farey de ordinul  $7$  este:

$0/1, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 2/5, 3/7, 1/2, 4/7, 3/5, 2/3, 5/7, 3/4, 4/5, 5/6, 6/7, 1/1$

Relativ la seria Farey de ordinul  $n$  sunt adevărate următoarele relații:

$x_0 = 0, x_1 = y_0 = 1, y_1 = n$

și

$x_{k+2} = ((x_k+n)/y_{k+1}) * x_{k+1} - x_k$

$y_{k+2} = ((y_k+n)/y_{k+1}) * y_{k+1} - y_k$

unde  $x_0/y_0, x_1/y_1, x_2/y_2, \dots$  sunt elementele seriei Farey de ordin  $n$ .

*Exemplu :*

$n = 5$

Succesiunea Farey de ordin 5:

$0/1 \quad 1/5 \quad 1/4 \quad 1/3 \quad 2/5 \quad 1/2 \quad 3/5 \quad 2/3 \quad 3/4 \quad 4/5 \quad 1/1$

### Analiza problemei și proiectarea soluției

Considerăm trei perechi de variabile  $(x_0, y_0)$ ,  $(x_1, y_1)$  și  $(x_2, y_2)$  cu ajutorul cărora calculăm succesiv elementele seriei Farey de ordin  $n$ . Pentru aceasta, următorul program este sugestiv.

### Program

```
#include <stdio.h>
void main(){
 unsigned n, x0, x1, x2, y0, y1, y2;
 printf(" n= ");
 scanf("%u", &n);
 x0=0; x1=y0=1; y1=n;
 printf("Succesiunea Farey de ordin %u:\n", n);
 printf(" %u/%u %u/%u ", x0, y0, x1, y1);
 while(x1!=1||y1!=1){
 x2 = (y0+n)/y1*x1-x0;
 y2 = (y0+n)/y1*y1-y0;
 x0 = x1; y0 = y1;
 x1 = x2;
 y1 = y2;
 printf(" %u/%u ", x1, y1);
 }
}
```

**Exercițiu**

Pentru un număr natural dat, să se scrie primii  $n$  termeni din *seria Rank*:  $1/1, 1/2, 2/1, 1/3, 3/1, 1/4, 2/3, 3/2, 4/1, 1/5, \dots$  în care termenii sunt, pe rând, fracțiile care au suma numărător + numitor 2, 3, 4, 5, ... și în fiecare grup ele sunt în ordine lexicografică (numărător, numitor).

**Problema 18. Divizori comuni**

Date fiind două numere naturale  $n$  și  $d$ , afișați toate perechile de numere naturale diferite mai mici decât  $n$  și care îl au pe  $d$  divizor comun. Numerele se încadrează în tipul **unsigned int**.

*Exemplu :*

| intrare (tastatură) | ieșire (ecran)                                                    |
|---------------------|-------------------------------------------------------------------|
| Introduceti n: 218  | (74, 37) (111, 37) (111, 74) (148, 37)                            |
| Introduceti d: 37   | (148, 74) (148, 111) (185, 37) (185, 74)<br>(185, 111) (185, 148) |

**Analiza problemei și proiectarea soluției**

Parcurgem toate numerele mai mari sau egale cu  $d$  și pentru fiecare astfel de număr încercăm dacă există un număr  $j$  mai mic decât el cu proprietatea că ambele îl au pe  $d$  divizor comun:

```
if(i%d==j%d && i%d==0 && i!=j)
 printf("(d, %d)", i, j);
```

**Program**

```
#include <stdio.h>
#include <conio.h>

void main(){
 unsigned int d, n, i, j;
 printf("Introduceti n: ");
 scanf("%d", &n);
 printf("Introduceti d: ");
 scanf("%d", &d);
 for(i=d; i<=n; i++)
 for(j=d; j<=i; j++)
 if(i%d==j%d && i%d==0 && i!=j)
 printf("(d, %d)", i, j);
 getch();
}
```

**Exercițiu**

Scrieți un program și pentru problema următoare: Pentru o pereche de numere naturale date, determinați cel mai mic număr natural  $n$  cu proprietatea că le are pe amândouă divizori.

**Problema 19. Transformare dintr-o bază în baza 10**

Scrieți un program, utilizând funcția de bibliotecă **strtol()**, care să convertească un număr dintr-o bază introdusă de la tastatură în echivalentul său în baza 10.

*Exemplu :*

| intrare (tastatură)                           | ieșire (ecran)                                                    |
|-----------------------------------------------|-------------------------------------------------------------------|
| Baza: 5<br>Nr. în baza 5: 1230321             | 1230321(baza 5) = 23836(baza 10)                                  |
| Baza: 2<br>Nr. în baza 2:<br>0101011111110011 | Scanare stopata la: stop<br>0101011111110011stop = 45043(baza 10) |

**Analiza problemei și proiectarea soluției**

Funcția **strtol()** are antetul:

```
long strtol(const char *nptr, char **endptr, int baza)
```

și convertește un sir de caractere la un întreg de tip **long**, returnând valoarea reprezentată în sirul **nptr**, exceptând cazul când se realizează o depășire și va returna **LONG\_MAX** sau **LONG\_MIN**.

**Program**

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

void main(){
 int baza;
 char numar[30], *p;
 long nr;
 printf("Baza: "); scanf("%d", &baza);
 printf("Nr. în baza %d: ", baza);
 scanf("%s", numar);
 nr = strtol(numar, &p, baza);
 if(!p) printf("\nEroare la caracterul : %c ", *p);
```

```

else {
 if(*p]!='\0')printf("\nScanare stopata la: %s", p);
 *p='\0';
 printf("\n%s(baza %d)=%ld(baza %d)", numar,baza,nr,10);
}
getch();
}

```

### Exerciții

- Implementați o funcție similară lui `strtol()`.
- Analizați și funcțiile `strtod()` și `strtoul()` și scrieți un program demonstrativ care le utilizează. Implementați funcții similare acestora.

### Problema 20. Testarea formatării afișării numerelor naturale

Scrieți un program-test pentru a afișa numărul 12345 cu diferite formate (aliniat la stânga, la dreapta, pe un număr de caractere...).

*Exemplu :*

```

12345
12345
 12345
12345
0000012345
12345

```

### Analiza problemei și proiectarea soluției

Vom folosi diverse formate pentru afișarea numerelor zecimale.

### Program

```

#include <conio.h>
#include <stdio.h>

void main(){
 int y=12345;
 printf("%d\n",y);
 printf("%5d\n",y);
 printf("%10d\n",y);
 printf("%03d\n",y);
 printf("%010d\n",y);
 printf("%-10d\n",y);
 getch();
}

```

### Exercițiu

Testați și afișarea numerelor binare, octale, hexazecimale.

### Problema 21. Numere perfecte



Nu se știe când au fost descoperite numerele perfecte, dar unele studii ne trimit la timpuri foarte îndepărtate, atunci când numerele au stârnit pentru prima dată curiozitatea. Este aproape sigur că egiptenii au ajuns în mod natural la aceste numere. Pentru prima dată ele au fost studiate de Pitagora, apoi de urmășii săi, mai mult din cauza proprietăților mistice decât pentru proprietățile teoretice. Despre aceste numere scrie și Euclid în cartea sa celebră *Elementele*, precum și succesorii ai acestuia. Astăzi sunt cunoscute 39 de numere perfecte, 288 (289 - 1) fiind ultimul descoperit (prin calcule manuale) până în 1911. Computerele au renăscut interesul pentru aceste

numere odată cu descoperirea numerelor prime ale lui Mersenne și, implicit, a numerelor perfecte. (Un număr prim Mersenne este un număr prim de forma  $2^p - 1$ . Dacă  $2^p - 1$  este număr prim, atunci  $2^{p-1}(2^p - 1)$  este număr perfect.) Astăzi, cel mai mare număr prim al lui Mersenne este  $2^{13466917} - 1$  (care este și cel mai mare număr prim) și, corespunzător, cel mai mare număr perfect este  $2^{13466916}(2^{13466917} - 1)$ .

Spunem că un număr natural este perfect dacă este egal cu suma divizorilor săi, fără el însuși. Scrieți un program care determină toate numerele perfecte mai mici decât 20.000.

*Exemplu :*

```

Numere perfecte pana la 20000:
6 28 496 8128

```

### Analiza problemei și proiectarea soluției

Pentru fiecare număr natural pînă la 20.000 se va calcula suma divizorilor săi în cadrul unei bucle `for()`.

### Program

```

#include <stdio.h>
#include <conio.h>
#define max 20000

void main(){
 unsigned long long i, j, s;
 printf("Numere perfecte pana la %d:\n", max);
 for (i=2; i<= max; i++)
}

```

```

 { s=0;
 for (j=1;j<=i/2;j++)
 if (i%j==0) s += j;
 if (i==s) printf("%5u",i);
 getch();
}

```

### Exercițiu

Adăugați programului instrucțiunile necesare astfel încât să se afișeze și divizorii săi, sub forma:

```

6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14
496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
8128 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 127 + 254 + 508 + 1016 + 2032 + 4064

```

### Problema 22. Numere prietene

Spunem că două numere naturale diferite sunt prietene dacă suma divizorilor proprii ai unuia este egală cu celălalt și invers. Scrieți un program care determină toate numerele prietene mai mici decât 20.000.

*Exemplu :*

Perechile de numere prietene mai mici decat 20000:  
(284, 220) (1210, 1184) (2924, 2620) (5564, 5020) (6368, 6232)  
(10856, 10744) (14595, 12285) (18416, 17296)

### Analiza problemei și proiectarea soluției

*Varianta 1.* Vom parcurge toate elementele din intervalul posibil, pentru fiecare pereche calculându-se sumele respective și comparându-se.

*Varianta 2.* Mai întâi se calculează toate sumele divizorilor și se stochează într-un tablou. Apoi se vor parcurge toate perechile posibile și sumele divizorilor se citesc din tabloul creat, fără a se mai calcula în cadrul acestui pas. În cadrul primei variante, fiecare sumă se calculează de mai multe ori, deci viteza de execuție va fi foarte mică. Implementați ambele variante pentru a face o comparație. Încercați întâi cu max = 500, 600, 700, ..., 2000.

### Program varianta 1

```

#include <stdio.h>
#include <conio.h>

#define max 2000

void main(){
 unsigned long i,k,j1,j2,s1,s2;
 printf("Perechile de numere prietene");
 printf(" mai mici decat %ld:\n", max);

```

```

 for (i=1;i<max;i++) {
 s1=1;
 for (j1=2; j1<=i/2;j1++)
 if (i%j1==0) s1=s1+j1;
 for(k=i+1;k<=max;k++) {
 s2=1;
 for (j2=2;j2<=k/2;j2++)
 if (k%j2==0) s2=s2+j2;
 if(s2==i&&s1==k)
 printf("(%ld,%ld)",i,k);
 }
 }
 getch();
}

```

### Program varianta 2 (utilizarea unui tablou unidimensional)

```

#include <stdio.h>
#include <conio.h>

#define max 20000

void main(){
 unsigned long i,j,j1,s1;
 unsigned long a[max];
 for (i=1;i<max;i++) {
 s1=1;
 for(j1=2; j1<=i/2;j1++) /*crearea tabloul cu sumele */
 if(i%j1==0) s1=s1+j1; /*divizorilor */
 a[i-1]=s1;
 }
 printf("Perechile de numere prietene");
 printf(" mai mici decat %ld:\n", max);
 for(i=1; i<max; i++)
 for(j=1; j<i; j++)
 if(a[i-1]==j && a[j-1]==i)
 printf("(%d, %d) ", i, j);
 getch();
}

```

### Exerciții

1. Pentru fiecare element, să se scrie și divizorii sub forma de sumă, în ordine crescătoare.
2. Folosind funcția time() din biblioteca time.h, determinați timpul de execuție al fiecărui algoritm în secunde.

### Problema 23. Suma cuburilor

Dat fiind un număr natural  $n$ , determinați toate numerele naturale mai mici decât el cu proprietatea că suma cuburilor cifrelor lor este egală cu numărul însuși.

*Exemplu :*

```
Dati nr. : 400
Numere egale cu suma cuburilor cifrelor sale (<400):
 153
 370
 371
```

### Analiza problemei și proiectarea soluției

Se parcurg toate numerele mai mici decât  $n$  și se calculează pentru fiecare suma cuburilor cifrelor.

### Program

```
#include <stdio.h>
#include <conio.h>

void main(){
 int n,i,j,s;
 printf("Dati nr. : ");
 scanf("%d",&n);
 printf("Numere egale cu suma cuburilor");
 printf(" cifrelor sale (<%d>):\n", n);
 for(i=2;i<=n;i++){
 s=0;
 j=i;
 while(j!=0){
 s=s+(j%10)*(j%10)*(j%10);
 j=j/10;
 }
 if(i==s) printf(" %d\n",i);
 }
 getch();
}
```

### Exercițiu

Să se găsească toate posibilitățile pentru  $(a, b, c, d)$  numere naturale din mulțimea  $\{1, 2, \dots, 100\}$  care satisfac condiția:  $a^3 = b^3 + c^3 + d^3$ .

### Problema 24. Coduri ASCII

Scrieți un program care afișează toate caracterele ASCII împreună cu codul lor.

|          |          |           |
|----------|----------|-----------|
| .....    | .....    | .....     |
| 0x1c   ? | 0x1d   ? | 0x1e   ?  |
| 0x1f   ? | 0x20     | 0x21   !  |
| 0x22   " | 0x23   # | 0x24   \$ |
| 0x25   % | 0x26   & | 0x27   '  |
| 0x28   ( | 0x29   ) | 0x2a   *  |
| 0x2b   + | 0x2c   , | 0x2d   -  |
| 0x2e   . | 0x2f   / | 0x30   0  |
| 0x31   1 | 0x32   2 | 0x33   3  |
| 0x34   4 | 0x35   5 | 0x36   6  |
| 0x37   7 | 0x38   8 | 0x39   9  |
| 0x3a   : | 0x3b   ; | 0x3c   <  |
| 0x3d   = | 0x3e   > | 0x3f   ?  |
| 0x40   @ | 0x41   A | 0x42   B  |
| 0x43   C | 0x44   D | 0x45   E  |
| 0x46   F | 0x47   G | 0x48   H  |
| 0x49   I | 0x4a   J | 0x4b   K  |
| 0x4c   L | 0x4d   M | 0x4e   N  |
| .....    | .....    | .....     |

### Analiza problemei și proiectarea soluției

Vom parcurge toate caracterele și vom afișa valorile cerute.

### Program

```
#include <stdio.h>
#include <conio.h>

int main(){
 unsigned char c;
 char buf[50];
 for(c = 0; c < 255; c++){
 sprintf(buf, "0x%x", (int)c);
 sprintf(buf, "%s |%c|", buf, c);
 printf("%-20s", buf);
 if((c % 3) == 0) printf("\n");
 }
 printf("\n");
 getch();
 return 0;
}
```

**Exercițiu**

Să se afișeze, pe trei coloane, toate aceste informații despre toate caracterele dintr două caractere date (de exemplu, dacă se vor dori codurile doar pentru litere sau doar pentru cifre etc.).

**Exerciții și probleme propuse (nu sunt permise tablouri)**

1. Se dă un număr natural  $N \leq 30.000$ . Să se calculeze numărul cifrelor pare și numărul cifrelor impare ale lui  $N^2$ .
2. Să se determine toate numerele prime palindrom dintr-un interval  $[a, b]$  dat, cu  $a$  și  $b$  date de la tastatură.
3. Să se determine toate numerele naturale palindrom mai mici decât un număr natural dat ( $< 50.000$ ).
4. Determinați toate numerele prime cu  $k$  ( $1 \leq k \leq 6$ ) cifre care au suma cifrelor  $s$  ( $1 < s < 60$ ).
5. Dat fiind un număr natural  $N$  cu maximum 9 cifre, să se determine numărul maxim care se poate scrie prin eliminarea unei singure cifre a lui  $N$ .
6. Dat un număr natural  $N$  cu maximum 9 cifre, să se determine numărul maxim care se poate scrie cu cifrele distincte ale lui  $N$ .
7. Un număr natural  $n$  este deosebit dacă există un număr natural  $m$  astfel încât  $n = m + S(m)$ , unde  $S(m)$  este suma cifrelor lui  $m$ . Să se scrie un program care afișează toate numerele deosebite din intervalul  $[a, b]$ , cu  $a$  și  $b$  numere naturale din intervalul  $[5, 50.000]$ .
8. Scrieți un program care rezolvă ecuația cu necunoscutele numere naturale  $n, m, k$ :  $n! + m! = k$ , unde  $k$  este un număr natural dat ( $0 \leq k \leq 1.000.000$ ).
9. Un număr natural scris sub forma  $c_1c_2\dots c_m$  este munte (vale) dacă există  $k$  număr natural,  $1 < k < m$ , astfel încât cifrele sunt strict crescătoare (descrescătoare) până la poziția  $k$  și apoi strict descrescătoare (crescătoare) de la poziția  $k$  până la poziția  $m$ . Exemplu:  $136732 \rightarrow$  munte,  $753469 \rightarrow$  vale.
10. Pentru două numere de maximum 4 cifre introduse de la tastatură, să se calculeze produsul acestora fără a utiliza operația de înmulțire din C, ci algoritmul prezentat în capitolul 2: „Algoritmi – elemente definitorii”.
11. Determinați cel mai mare număr prim mai mic decât  $10^9$ .
12. Să se determine cel mai mic număr natural  $n$  cu proprietatea că 7 divide pe  $n^3 + n - 2$ . Același lucru pentru 5 și  $n^2 - 3n + 6$ .
13. Să se afișeze valorile de forma  $2^p - 2$  pentru  $p$  număr prim care se încadrează în tipul `unsigned long int`. Pentru fiecare astfel de pereche, scrieți și restul împărțirii lui  $2^p - 2$  prin  $p$ . Ce constatați?
14. Dacă  $n$  este un număr natural, atunci numărul  $E = n(n + 1)(n + 2)(n + 3) + 1$  este un patrat perfect. Cercetați acest lucru pentru primele 12 numere prime.
15. Scrieți un program care vă ajută să calculați ultima cifră a numerelor:  $323^{700}$ ,  $1244^{51}$ ,  $1982^{83}$ ,  $164^{41}$ ,  $194^{53}$ ,  $17^{60} + 12^{40}$ ,  $13^{20} + 22^{30}$ .
16. Scrieți un program care determină ultima cifră a numărului  $1! + 2! + 3! + \dots + n!$  pentru  $n$  ce se încadrează în tipul `unsigned long`. Programul va avea următoarea

funcționare: se vor introduce numere până când o condiție de oprire este satisfăcută, pentru fiecare număr dat se va scrie ultima cifră a numărului  $1! + 2! + 3! + \dots + n!$ . Ce observați?

17. Să se găsească primele 10 numere prime de forma  $4k - 1$  și primele 10 numere prime de forma  $6k - 1$ , cu  $k$  număr natural.
18. Pentru un număr natural dat  $N$  ( $< 20$ ), să se determine toate numerele prime  $p$  mai mici decât el, iar pentru fiecare astfel de număr să se scrie și restul împărțirii lui  $p^2$  prin 12. Ce observați?
19. Dacă vom nota cu  $P(M)$  produsul cifrelor numărului natural  $M$ , să se scrie un program care calculează  $P(1) + P(2) + \dots + P(N)$  pentru un număr natural  $N$  dat.
20. În câte moduri se poate scrie 447 ca sumă de numere naturale impare consecutive? Generalizare.
21. Fie  $a, b \in N$ ,  $ab = 300$ . Câte perechi diferite  $(a, b)$  există, știind că nu au divizori comuni? Generalizare.
22. Fiecare cifră corespunde unei litere diferite. Atunci KANGAROO + 10000 \* AROO - 10000 \* KANG este: a) AROOAROO; b) AROOKANG; c) KANGKANG; d) KANGAROO; e) KANGOOOO. Alegeți varianta corectă.
23. Dat fiind un număr de 9 cifre  $N$ , eliminați 4 cifre așa încât numărul rămas să fie cel mai mic posibil (cifrele sunt în ordine cele din  $N$ , fără cele eliminate).
24. Să se scrie un program care rezolvă pentru numere întregi ecuațiile  $x + y = x \cdot y$  și respectiv  $5 \cdot x + 7y^2 = 1600$ .
25. Să se scrie un program care rezolvă în numere pozitive întregi ecuația  $1/x + 1/y = 1/2$ .
26. Scrieți un program care determină ultimele două cifre ale numărului  $2^n$  pentru  $n$  de tip `unsigned long`. Generalizare pentru  $m^n$ .
27. Date fiind două numere întregi  $a$  și  $b$  să se verifice dacă sunt prime între ele. Dacă da, atunci să se genereze aleatoriu 20 de numere mai mari decât  $n$  și pentru fiecare dintre ele să se găsească perechea  $(x, y)$  de numere naturale cu proprietatea  $n = ax + by$ .
28. *Triunghi sau trapez.* Scrieți un program care, pentru un număr natural  $n$  dat, afișează pe ecran un triunghi (trapez) isoscel ca în exemplele de mai jos. Fiecare linie diferă de precedenta prin două caractere la extremități, iar partea de sus este formată dintr-unul sau două caractere „\*”. În caz că nu există soluție, se va scrie mesajul corespunzător.

*Exemplu :*

| $n=1$ | $n=4$    | $n=5$              | $n=12$              | $n=16$                     |
|-------|----------|--------------------|---------------------|----------------------------|
| *     | *<br>*** | Nici o<br>solutie. | **<br>****<br>***** | *<br>***<br>*****<br>***** |

29. *Cifra de control.* Cifra de control a unui număr natural dat se obține prin însumarea succesivă a cifrelor sale, până când obținem o cifră. De exemplu, cifra de control pentru 465329 este 2:  $\text{sumaCifre}(465329) = 29$ ,  $\text{sumaCifre}(29) = 11$ ,  $\text{sumaCifre}(11) = 2$ , care este o cifră. Vom defini *cifra de control a unei date calendaristice* prin cifra de control a numărului întreg obținut prin alipirea numerelor

zilei (maximum 2 cifre), lunii (maximum 2 cifre), respectiv anului (maximum 4 cifre). Să se scrie un program care determină cifra de control a unei date calendaristice introduse de la tastatură.

*Exemplu :*

| tastatură | ecran                       |
|-----------|-----------------------------|
| Zi=18     | Cifra_Control(18.05.1987)=3 |
| Luna=5    |                             |
| An=1987   |                             |

30. *Puteri ale lui k.* Pentru un număr natural  $k$  dat și un interval  $[a, b]$ , determinați toate numerele naturale din intervalul  $[a, b]$  care sunt puteri ale lui  $k$  ( $2 \leq k \leq 9$ ,  $2 \leq a \leq b \leq 9.000.000$ ).

*Exemplu :*

| tastatură | ecran    |
|-----------|----------|
| $k=2$     | 16 32 64 |
| $a=15$    |          |
| $b=70$    |          |

$$(16 = 2^4, 32 = 2^5, 64 = 2^6).$$

## Tablouri, pointeri, pointeri la funcții

18 probleme complet rezolvate, 20 de programe C, 41 de exerciții și probleme propuse.  
Probleme rezolvate:

- element minim în tablou
- sortările *Bubble Sort* și *Insertion Sort*
- derivata unui polinom
- ciurul lui Eratostene
- selectarea unor componente în tablou
- înmulțirea unui polinom cu  $(X-a)$
- produsul a două matrice
- conjectura lui Goldbach
- elemente vecine cu diferență 1
- rezolvarea ecuațiilor de gradul al II-lea
- căutarea unui element în tablou
- *cmmdc*, *cmmmc* în tablou pătratic
- pointer de parcursare a unui tablou
- adrese în tablouri
- interschimbare de octeți
- tablou cu pointeri la funcții
- *puncte șa* în matrice

*Nu-ți fie teamă că înaintezi prea încet. Teme-te dacă te oprești.*

Sun Tzu

### **Problema 1. Element minim într-un tablou unidimensional**

Scrieți un program care, având la intrare un sir de numere naturale, determină valoarea minimă din sir și afișează pozițiile acesteia în secvența de numere dată. Elementele sirului se vor introduce secvențial până la citirea numărului 0, care va indica sfârșitul sirului la elementul anterior. Toate elementele sirului dat sunt strict pozitive și vor fi maximum 100 de elemente de valoarea maximă 60.000. Numerele sirului dat nu trebuie stocate într-un tablou unidimensional.

*Exemplu:*

Introduceti sirul de elemente:  
45 6 3 21 3 67 8 90 3 12 69 3 81 0

Valoarea minima este 3!

Aceasta valoare se gaseste pe pozitiile: 3 5 9 12

#### *Analiza problemei și proiectarea soluției*

Vom stoca succesiv în tabloul unidimensional `a[]` pozițiile pe care se află elementul minim găsit până la numărul curent. Vom utiliza variabila `imax` pentru a stoca numărul de elemente deja introduse în tabloul `a[]`, `lmin`-pentru a păstra elementul minim curent. Dacă elementul citit este egal cu minimul curent, atunci se va adăuga indexul său în tabloul `a[]`, dacă elementul citit este mai mic decât minimul curent `lmin`, atunci se vor schimba corespunzător valorile variabilelor `imax`, a și `lmin`.

#### *Program*

```
#include <stdio.h>

int main(){
 unsigned imax = 0, lmin, i = 0;
 int a[100];
 unsigned el;
 printf("Introduceti sirul de elemente:\n");
 scanf("%d", &el);
 a[imax++] = 1; lmin = el; i++;
 while(scanf("%d", &el) == 1 && el>0){
 i++;
 }
}
```

```

 if(el == lmin) a[imax++] = i;
 if(el < lmin){
 imax = 0; a[imax++] = i;
 lmin = el;
 }
}
printf("Valoarea minima este %d\n", lmin);
printf("Aceasta valoare se gaseste pe pozitiile: ");
for(i=0; i<imax; i++) printf("%d ", a[i]);
printf("\n\n");
}

```

### Exerciții

- Rescrieți programul anterior astfel încât sirul dat să fie citit într-un tablou unidimensional.
- Modificați programul anterior astfel încât să se determine valoarea maximă din sir și pozițiile unde se află această valoare.

### Problema 2. Bubble Sort

Dat de la tastatură un tablou unidimensional cu maximum 200 de elemente ce se încadrează în tipul int, ordonați-l crescător.

*Exemplu :*

| intrare(tastatură)    | ieșire(ecran)                |
|-----------------------|------------------------------|
| n = 4                 |                              |
| Elementele tabloului: | Tabloul ordonat: -2 1 23 456 |
| 23 456 1 -2           |                              |

### Analiza problemei și proiectarea soluției

Vom ordona tabloul dat utilizând o metodă de tip *Bubble Sort*, parcurgem secvențial elementele tabloului de la stânga la dreapta. Pentru un element fixat, îl vom compara cu toate de după el și, dacă vreunul este mai mic, executăm o interschimbare. În acest fel se construiește succesiv vectorul final ordonat.

Interschimbarea a două variabile a și b se poate face folosind o variabilă auxiliară („metoda paharelor”), cu o secvență de forma:

```
(aux = a; a = b; b = aux;)
```

sau, dacă variabilele reprezintă numere, se poate face interschimbarea nefolosind nici o variabilă auxiliară, cu o secvență de forma:

```
(a = a - b; b = a + b; a = b - a)
```

Complexitatea algoritmului *Bubble Sort* este  $O(n^2)$ , datorită celor două bucle for imbricate.

### Program

```

#include <stdio.h>

void main(){
 int i, j, n, a[100], aux;
 printf("n = "); scanf("%d", &n);
 printf("Elementele tabloului:\n");
 for(i=0; i<n; i++)
 scanf("%d", &a[i]);
 for(i=0; i<n-1; i++)
 for(j=i+1; j<n; j++)
 if(a[i]>a[j])
 {aux=a[i]; a[i]=a[j]; a[j]=aux;}
 printf("Tabloul ordonat: ");
 for(i=0; i<n; i++) printf(" %d ", a[i]);
}

```

### Exerciții

1. *Insertion Sort*. Ordonare prin inserție (mâna cu cărți de joc): elementele se parcurg de la stânga spre dreapta, păstrând invariantul că la stânga elementului curent valorile sunt ordonate și elementul curent este inserat pe poziția corespunzătoare:

```

insereazaValoare(tablou a, int n, valoare){
 int i = n-1;
 while (i≥0 && a[i]>valoare) {
 a[i+1] = a[i];
 i--;
 }
 a[i+1] = valoare;
}

sortareInsertie(tablou a, int n){
 int i;
 for(i=1; i<n; i++) {
 insereazaValoare (a, i, a[i]);
 }
}

```

2. Următoarea secvență de cod execută interschimbarea a două numere reale:

```
(a = a - b; b = a + b; a = b - a)
```

Dată fiind secvența  $\{b = a + ?; ? = ? ? ?; ? = ? ? ?\}$ , completați semnele „?” cu caractere din {a, b, +, -} astfel încât să se obțină același lucru.

### Problema 3. Derivata unui polinom

Scrieți un program care afișează derivata unui polinom și calculează valoarea acesteia pentru o valoare reală introdusă de la tastatură.

*Exemplu:*

Introduceti polinomul:

Gradul polinomului: 3

Introduceti coeficientii:

a0 = 1

a1 = 2

a2 = 3

a3 = 4

Valoarea de calculat derivata: 1.45

P'(X) = 2 + 6X^1 + 12X^2

Valoarea polinomului derivat P'(1.450) = 35.930

### Analiza problemei și proiectarea soluției

Considerând polinomul

$$P(X) = a_0 + a_1 X + a_2 X^2 + \dots + a_n X^n$$

Derivata acestuia este:

$$P'(X) = a_1 + 2a_2 X + 3a_3 X^2 \dots + n a_n X^{n-1}$$

Programul de mai jos este sugestiv privind calculul valorii polinomului într-un punct dat și determinarea derivatei acestuia.

Complexitatea este liniară  $O(n)$ , deoarece este nevoie doar de o buclă for pentru parcurserea coeficienților.

### Program

```
#include <stdio.h>

int CitestePolinom(int *n, int a[100]){
 int i;
 printf("Gradul polinomului: ");
 scanf("%d", n);
 printf("Introduceti coeficientii:\n");
 for(i=0; i<=*n; i++){
 printf("a%d = ", i);
 scanf("%d", &a[i]);
 }
 return 0;
}
```

```
double ValoarePolinom(int n, int a[], double x){
 double x_aux = 1, val = a[0];
 int i;
 for(i=1; i<=n; i++){
 x_aux *= x;
 val += a[i]*x_aux;
 }
 return val;
}

int DerivaPolinom(int *n, int a[]){
 int i;
 for(i=0; i<*n; i++)
 a[i] = a[i+1]*(i+1);
 (*n)--;
 return 0;
}

void ScriePolinom(int *n, int a[]){
 int i;
 if(*n<0) return;
 printf("%d", a[0]);
 for(i=1; i<=*n; i++)
 printf(" + %dX^%d", a[i], i);
 printf("\n");
}

void main(){
 int n; p[100];
 float x;
 printf("Introduceti polinomul:\n");
 CitestePolinom(&n, p);
 DerivaPolinom(&n, p);
 printf("Valoarea de calculat derivata:");
 scanf("%f", &x);
 printf("P'(X) = "); ScriePolinom(&n, p);
 printf("Valoarea polinomului derivat P'(% .3lf) = %.3lf",
 x, ValoarePolinom(n, p, x));
}
```

### Exerciții

- Modificați programul anterior astfel încât să afișați într-o formă corespunzătoare cele două polinoame: cel introdus și polinomul derivat.
- Dați fiind coeficienții întregi, din intervalul  $[-1000, 1000]$ , ai două polinoame  $P$  și  $Q$ , calculați coeficienții polinomului sumă.

### Problema 4. Ciurul lui Eratostene

Scrieți un program care afișează toate numerele prime mai mici sau egale cu un număr natural dat  $n$  ( $n \leq 60.000$ ) folosind algoritmul sitei lui Eratostene.

*Exemplu :*

```
n = 40
2 3 5 7 11 13 17 19 23 29 31 37
```

### Analiza problemei și proiectarea soluției

Ideea algoritmului lui Eratostene, aşa cum este și prezentat în capitolul 2. „Algoritmi – elemente definitorii”, este de a amplasa la început în „sită” toate numerele de la 1 la  $n$ , apoi „cernerea” succesivă a acestora (eliminarea elementelor compuse). Vom folosi tabloul unidimensional `sita[]` cu elemente 0 și 1, cu semnificația `sita[i] = 0` (element eliminate din sită) dacă elementul  $i$  nu este prim; `sita[i] = 1` (element rămas în sită) dacă elementul  $i$  este prim. Pentru toate numerele prime, succesiv, se vor elimina din sită toți multiplii lor (`sita[i*j] = 0`). Programul de mai jos este reprezentativ în acest sens.

### Program

```
#include <stdio.h>

short sita[60001];
unsigned i, j, n;

void main(){
 printf("n = ");
 scanf("%d", &n);
 memset(sita, 1, sizeof(sita));
 for(i = 2; i*i <= n; i++)
 if(sita[i]){
 j = 2;
 while(i*j <= n){
 sita[i*j] = 0;
 j++;
 }
 }
 for(i = 2; i <= n; i++)
 if(sita[i])
 printf("%d ", i);
}
```

### Exercițiu

Înlocuiți instrucțiunea

```
while(i*j <= n) {
 sita[i*j] = 0;
 j++;
}
```

cu o instrucțiune do-while{}.

### Problema 5. Selectarea numerelor ce depășesc un prag

Se dă o mulțime de maximum 200 de numere reale cuprinse în intervalul [-32.000, 32.000]. Scrieți un program care să selecteze din această mulțime doar pe cele mai mari decât un prag  $x_0$  dat,  $x_0$  fiind un număr real din același interval.

*Exemplu :*

```
DATE DE INTRARE:
Numarul de elemente: 5
Elementele tabloului: -9.87 456.78 -321.09 -456.901 5.432
Pragul x0: -10.34
```

```
DATE DE IEȘIRE:
Sunt 3 numere mai mari decat pragul -10.340:
-9.870 456.780 5.432
```

### Analiza problemei și proiectarea soluției

Mulțimea dată se parcurge secvențial și elementele care satisfac condiția din enunț sunt salvate în vectorul `b[]`, afișat ulterior. Complexitatea este liniară  $O(n)$ .

### Program

```
#include <stdio.h>

int main(){
 float a[201], x0, b[201];
 int n, i, m;
 printf("DATE DE INTRARE: \n");
 printf("Numarul de elemente: ");
 scanf("%d", &n);
 printf("Elementele tabloului: ");
 for(i=0; i<n; i++)
 scanf("%f", &a[i]);
 printf("Pragul x0: ");
 scanf("%f", &x0);
 m=0;
```

```

for(i=0; i<n; i++)
 if(a[i]>x0) b[m++]=a[i];
printf("\nDATE DE IESIRE:\n");
printf("Sunt %d numere mai mari decat pragul %.3f: \n",
 m, x0);
for(i=0; i<m; i++) printf("%.3f ", b[i]);
return 0;
}

```

### Exercițiu

Adăugați la programul anterior liniile corespunzătoare, astfel încât să afișați și toate elementele care sunt mai mici decât  $\sqrt{x_0}$ .

### Problema 6. Înmulțirea unui polinom cu $(X-a)$

Se citesc de la intrare: un număr real  $a$  și un polinom de gradul  $n$  (maximum 100). Determinați coeficienții polinomului obținut prin înmulțirea polinomului dat cu polinomul  $(X - a)$ . Presupunem că atât coeficienții polinomului inițial, cât și coeficienții polinomului produs se încadrează în tipul de date float.

*Exemplu:*

```

a = 4
Gradul polinomului n = 2
Coeficientii:
p1[0] = 1
p1[1] = 2
p1[2] = 3
Coeficientii polinomului produs:
p2[0] = -4.000
p2[1] = -7.000
p2[2] = -10.000
p2[3] = 3.000

```

### Analiza problemei și proiectarea soluției

Dacă vom considera polinomul  $P(X) = a_0 + a_1X + a_2X^2 + \dots + a_nX^n$ , atunci

$$P(X) \cdot (X-a) = -a \cdot a_0 + (a_0 - aa_1)X + (a_1 - aa_2)X^2 + \dots + (a_{n-1} - aa_n)X^n + a_nX^{n+1}$$

Adică valorile coeficienților sunt:  $b_0 = -a \cdot a_0$ ,  $b_{n+1} = a_n$ ,  $b_i = a_{i-1} - aa_i$ ,  $i \in \{1, \dots, n\}$ .

### Program

```

#include <stdio.h>

int main(){
 int n, i;
 float a, p1[101], p2[101];
 printf("a = ");
 scanf("%f", &a);
 printf("Gradul polinomului n = ");
 scanf("%d", &n);
 printf("Coeficientii:\n");
 for(i=0; i<=n; i++){
 printf("p1[%d] = ", i);
 scanf("%f", &p1[i]);
 if(i) p2[i] = p1[i-1]-a*p1[i];
 }
 p2[0] = (-1)*a*p1[0];
 p2[n+1] = p1[n];
 printf("Coeficientii polinomului produs:");
 for(i=0; i<=n+1; i++)
 printf("\np2[%d] = %.3f", i, p2[i]);
}

```

### Exerciții

1. Determinați polinomul cât și restul împărțirii polinomului dat la  $(X-a)$ .
2. Scrieți un program care determină coeficienții polinoamelor sumă și produs a două polinoame cu grad  $n$ , respectiv introduse de la tastatură.

### Problema 7. Produs a două matrice

Scrieți un program care determină produsul a două matrice. Elementele matricelor date și produs se încadrează în tipul int, dimensiunile matricelor sunt numere naturale mai mici decât 100. Se presupune că numărul de coloane din prima matrice este egal cu numărul de linii din cea de-a două matrice.

*Exemplu:*

```

INTRARE:
Matricea A:
Numarul de linii m = 3
Numarul de coloane n = 2
Elementele matricii: 1 2 3 4 5 6
Matricea B:
Numarul de linii m = 2

```

```
Numarul de coloane n = 2
Elementele matricii: 7 8 9 10
```

IESTIRE:

Matricea produs:

|    |     |
|----|-----|
| 25 | 28  |
| 57 | 64  |
| 89 | 100 |

### Analiza problemei și proiectarea soluției

Dacă matricele date sunt A și B, de forma

$$A_{mn} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, m, n \in \mathbb{N}, \quad B_{np} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{pmatrix}, n, p \in \mathbb{N}$$

atunci matricea produs C are m linii și p coloane și este de forma:

$$C_{mp} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{pmatrix}$$

unde  $c_{ij} = a_{1j} * b_{1i} + a_{2j} * b_{2i} + \dots + a_{nj} * b_{ni}$ , pentru  $\forall i \in \{1, \dots, m\}$ ,  $\forall j \in \{1, \dots, p\}$ , formulă ilustrată în program de secvență:

```
for(i=0; i<A.m; i++)
 for(j=0; j<B.n; j++)
 for(k=0; k<A.n; k++)
 C->a[i][j] += A.a[i][k] * B.a[k][j];
```

O matrice este complet definită prin tipul struct TMatrice, care conține numărul de linii (m), numărul de coloane (n) și elementele (tabloul a[]). Complexitatea este polinomială  $O(n \cdot m \cdot p)$ , datorită celor trei for-uri imbricate.

### Program

```
#include <stdio.h>

typedef struct{
 int m, n;
 int a[100][100];
}TMatrice;
```

```
int CitesteMatrice(TMatrice *A){
 int i, j;
 printf("Numarul de linii m = ");
 scanf("%d", &A->m);
 printf("Numarul de coloane n = ");
 scanf("%d", &A->n);
 printf("Elementele matricii: ");
 for(i=0; i<A->m; i++)
 for(j=0; j<A->n; j++)
 scanf("%d", &A->a[i][j]);
 return 0;
}

int Produs(TMatrice A, TMatrice B, TMatrice *C){
 int i, j, k;
 if (A.m != B.m) return 0;
 C->m = A.m;
 C->n = B.n;
 for(i=0; i<A.m; i++)
 for(j=0; j<B.n; j++)
 C->a[i][j] = 0;
 for(i=0; i<A.m; i++)
 for(j=0; j<B.n; j++)
 for(k=0; k<A.n; k++)
 C->a[i][j] += A.a[i][k] * B.a[k][j];
 return 1;
}

int AfiseazaMatrice(TMatrice A){
 int i, j;
 for(i=0; i<A.m; i++){
 printf("\n");
 for(j=0; j<A.n; j++)
 printf("%7d", A.a[i][j]);
 }
 return 0;
}

int main(){
 TMatrice A, B, C;
 printf("INTRARE:\nMatricea A:\n");
 CitesteMatrice(&A);
 printf("Matricea B:\n");
 CitesteMatrice(&B);
```

```

Produs(A, B, &C);
printf("\nIESIRE:\nMatricea produs:");
AfiseazaMatrice(C);
return 0;
}

```

### Exerciții

- Adăugați la programul anterior funcții care să determine suma și diferența a două matrice.
- Adăugați la programul anterior o funcție care să calculeze diferența simetrică:  $(A-B) + (B-A)$  pentru două matrice date.
- Modificați programul de mai sus astfel încât să se facă înmulțirea unei matrice cu un vector: creați tipul TVector asemănător cu TMatrice și antetul funcției va avea forma: Produs(TMatrice, TVector, TVector\*).

*Notă:* În cadrul funcțiilor de mai sus trebuie făcută și validarea dimensiunilor celor două matrice.

### Problema 8. Conjectura lui Goldbach

Christian Goldbach a fost un matematician celebru. S-a născut la 18 martie 1690 în Königsberg, Prusia (astăzi Kalininsgrad, Rusia) și a murit la 20 noiembrie 1764 în Moscova. El a studiat dreptul și s-a preocupat, de asemenea, de matematică, în special de teoria numerelor. Marginal, s-a mai interesat de domenii ca sumele infinite, teoria curbelor, teoria ecuațiilor. El a călătorit mult în Europa și a avut discuții fructuoase cu matematicieni celebri, precum Leibniz, Bernoulli, de Moivre, Hermann. O mare parte din munca sa se reflectă în corespondența cu Euler. Această corespondență a durat aproape 20 de ani și reprezintă unul dintre cele mai importante documente științifice din secolul al XVIII-lea. A lucrat ca profesor de matematică și istoric la nou-înființata academie din St. Petersburg și i-a predat printului Petru (viitorul țar Petru al II-lea). De la el provine o celebră conjectură în teoria numerelor. Aceasta este menționată pentru prima dată într-o scrisoare adresată lui Euler la 7 iulie 1742 în care afirmă că „se pare că orice număr mai mare decât 2 se poate scrie ca suma a trei numere prime”. Goldbach consideră 1 număr prim, convenție ce nu va fi urmată ulterior. Euler a reexprimat această conjectură în forma: „orice număr întreg pozitiv  $\geq 4$  poate fi scris ca suma a două numere prime” (numită și conjectura „puternică” sau „binară” a lui Goldbach).

Se consideră un tablou bidimensional de dimensiuni  $m \times n$  cu elemente numere naturale din intervalul  $[4, 1000]$ . Să se indice pozițiile elementelor pare ale matricei și toate descompunerile acestora ca sumă de numere prime, ca în exemplul următor.

*Exemplu:*

| intrare(tastatură)                            | ieșire(ecran)         |
|-----------------------------------------------|-----------------------|
| Dimensiunile matricei:<br>2 3                 | Matricea introdusa:   |
| Elementele matricei:<br>12 346 789<br>44 7 82 | 12 346 789<br>44 7 82 |

| intrare(tastatură) | ieșire(ecran)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | <p>Elem pare &gt;4 și desc lor Goldbach:</p> <p>a[1][1] = 12<br/>     12 = 5 + 7</p> <p>a[1][2] = 346<br/>     346 = 29 + 317<br/>     346 = 53 + 293<br/>     346 = 83 + 263<br/>     346 = 89 + 257<br/>     346 = 107 + 239<br/>     346 = 113 + 233<br/>     346 = 149 + 197<br/>     346 = 167 + 179<br/>     346 = 173 + 173</p> <p>a[2][1] = 44<br/>     44 = 3 + 41<br/>     44 = 7 + 37<br/>     44 = 13 + 31</p> <p>a[2][3] = 82<br/>     82 = 3 + 79<br/>     82 = 11 + 71<br/>     82 = 23 + 59<br/>     82 = 29 + 53<br/>     82 = 41 + 41</p> |

### Analiza problemei și proiectarea soluției

Vom scrie funcția clasică de testare a primalității prim() cu antetul:

```
short prim(int n)
```

care decide dacă un număr natural dat este număr prim. Programul principal utilizează această funcție.

### Program

```

#include <stdio.h>
#include <math.h>

short prim(int n){
 int i;
 if(n==1) return 0;
 for(i=2; i*i<=n; i++)
 if(n%i==0) return 0;
 return 1;
}

```

```

void main(){
 int m, n, i, j, k;
 int a[100][100];
 printf("Dimensiunile matricei:\n");
 scanf("%d %d", &m, &n);
 printf("Elementele matricei:\n");
 for(i=0; i<m; i++)
 for(j=0; j<n; j++)
 scanf("%d", &a[i][j]);
 printf("Matricea introdusa: \n");
 for(i=0; i<m; i++){
 printf("\n");
 for(j=0; j<n; j++)
 printf("%4d", a[i][j]);
 }
 printf("\n\nElem pare >4 si desc Goldbach:");
 for(i=0; i<m; i++)
 for(j=0; j<n; j++)
 if(a[i][j]&2==0){
 printf("\n a[%d][%d] = %d\n", i+1, j+1, a[i][j]);
 for(k=2; k<=a[i][j]/2; k++)
 if(prim(k) && prim(a[i][j]-k))
 printf("%d = %d + %d \n", a[i][j], k, a[i][j]-k);
 }
}

```

### Exerciții

- Singurul număr prim par este 2. Scrieți funcția de testare a unui număr prim mai efficient, folosind această observație.
- Modificați programul astfel încât să nu folosiți funcția definită de utilizator `prim()`.
- Să se scrie un program care determină perechile de *numere prime gemene* (prime și impare consecutive) mai mici decât 2.000. Exemple: 5 și 7, 11 și 13.
- Să se determine toate numerele prime situate în intervalul  $[a, b]$ , unde  $a$  și  $b$  sunt două numere naturale introduse de la tastatură.

### Problema 9. Elemente vecine cu diferență 1

Se dă de la tastatură un tablou unidimensional cu elemente numere întregi din intervalul  $[-1000, 1000]$ . Să se determine perechile de numere de pe poziții consecutive care au valori a căror diferență în modul este egală cu 1 și multimea corespunzătoare vectorului. Vectorul introdus va avea maximum 200 de elemente și se presupune că datele de intrare sunt corecte.

### Exemplu:

Numarul de elemente: 10  
 Introduceti elementele:  
 12 11 23 11 10 12 13 11 6 45  
 $|a[1] - a[2]| = |12 - 11| = 1$   
 $|a[4] - a[5]| = |11 - 10| = 1$   
 $|a[6] - a[7]| = |12 - 13| = 1$   
 Multimea corespunzătoare vectorului:  
 { 12, 11, 23, 10, 13, 6, 45 }

### Analiza problemei și proiectarea soluției

Vom parcurge secvențial vectorul în cadrul unei instrucțiuni `for()` și vom decide dacă prima condiție este satisfăcută. Tot în cadrul acestui `for()` se verifică dacă elementul curent este deja în *vectorul-mulțime*; dacă nu, atunci se adaugă la acesta cu instrucțiunea:

```
if(!este) b[m++]=a[i];
```

( $m$  reprezintă numărul de elemente din *vectorul-mulțime*).

### Program

```

#include <stdio.h>
#include <math.h>

void main(){
 int n, i, k, m;
 int a[200], b[200];
 short este;
 printf("Numarul de elemente: ");
 scanf("%d", &n);
 printf("Introduceti elementele:\n");
 for(i=0; i<n; i++)
 scanf("%d", &a[i]);
 m = 0;
 for(i=0; i<n; i++){
 if(i<n-1 && abs(a[i]-a[i+1])==1)
 printf("\n |a[%d] - a[%d]| = |%d - %d| = 1 ", i+1, i+2, a[i], a[i+1]);
 este = 0;
 for(k=0; !este && k<m; k++)
 if(b[k]==a[i]) este=1;
 if(!este) b[m++]=a[i];
 }
}

```

```

printf("\n Multimea corespunzătoare vectorului:\n");
for(i=0; i<m-1; i++) printf(" %d,", b[i]);
if(m) printf(" %d", b[m-1]);
printf("\n");
}

```

### Exercițiu

Modificați programul astfel încât mulțimea elementelor vectorului să fie afișată în ordine crescătoare.

### Problema 10. Rezolvarea ecuațiilor de gradul al II-lea

Se consideră un tablou bidimensional cu  $n$  linii și 3 coloane ( $1 \leq n \leq 200$ ) cu elemente numere reale. Afipați ecuațiile de gradul al II-lea ce se pot forma cu elementele de pe o linie, consecutiv, drept coeficienți  $a$ ,  $b$ ,  $c$ ; în dreptul lor, soluțiile reale ale acestora, dacă acestea există. Se consideră că elementele de pe prima coloană sunt nenule, adică toate cele  $n$  ecuații date sunt de gradul al II-lea.

*Exemplu:*

Numarul de linii: 3

Introduceti matricea:

```

1 2 1
1 0 1
2.34 4.67 0.32

```

----

REZULTATE

Ecuatia 1:

```

1.000X^2 + 2.000X + 1.000 = 0
Solutie dubla x1 = x2 = -1.000

```

Ecuatia 2:

```

1.000X^2 + 0.000X + 1.000 = 0
Nu are solutii reale!

```

Ecuatia 3:

```

2.340X^2 + 4.670X + 0.320 = 0
x1 = -1.925 x2 = -0.071

```

### Analiza problemei și proiectarea soluției

După citirea elementelor matricii, fiecare linie este prelucrată secvențial, aplicându-se algoritmul matematic clasic de rezolvare a ecuațiilor de gradul al II-lea:

$$ax^2 + bx + c = 0, a \neq 0$$

$$\Delta = b^2 - 4 \cdot a \cdot c$$

$$\text{Dacă } \Delta \geq 0 \text{ atunci ecuația are soluții reale: } x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$(\text{dacă } \Delta = 0 \text{ atunci } x_1 = x_2 = \frac{-b}{2a}).$$

### Program

```

#include <stdio.h>
#include <math.h>

void main(){
 int n, i;
 float m[200][3];
 float a, b, c, delta;
 printf(" Numarul de linii: ");
 scanf("%d", &n);
 printf("\nIntroduceti matricea:\n");
 for(i=0; i<n; i++)
 scanf("%f %f %f", &m[i][0], &m[i][1], &m[i][2]);
 printf("\n----\nREZULTATE\n----\n");
 for(i=0; i<n; i++){
 a=m[i][0]; b=m[i][1]; c=m[i][2];
 printf("\nEcuatia %d:\n ", i+1);
 printf("%3.3fX^2 + %3.3fX + %3.3f = 0 \n", a, b, c);
 delta = b*b - 4*a*c;
 if(delta < 0) printf(" Nu are solutii reale!\n");
 else if(delta == 0)
 printf(" Solutie dubla x1 = x2 = %3.3f\n",
 -b/(2*a));
 else printf(" x1 = %3.3f x2 = %3.3f\n",
 (-b-sqrt(delta))/(2*a), (-b+sqrt(delta))/(2*a));
 }
}

```

### Exerciții

1. Modificați programul anterior astfel încât dacă ecuația nu are soluții reale să se afișeze soluțiile complexe ale acesteia.

2. Modificați programul astfel încât să se facă și validarea datelor, adică dacă elementul de pe coloana 1 (coeficientul a) este 0, respectiva ecuație să nu mai apară în cadrul datelor de ieșire.
3. Adaptați problema de mai sus corespunzător ecuațiilor de gradul 1.

### **Problema 11. Căutarea unui element într-un tablou unidimensional**

Fiind dat un tablou unidimensional cu maximum 200 de elemente numere naturale ce se încadrează în tipul `unsigned long`, să se decidă dacă un număr dat este în tablou. Dacă nu, să se insereze acest număr la începutul tabloului, pe prima poziție.

#### *Exemplu 1:*

```
Numar de elemente: 4
Introduceti elementele:
1 2 3 4
Elementul de cautat: 5

5 NU este in tablou!
Noul tablou: 5 1 2 3 4
```

#### *Exemplu 2:*

```
Numar de elemente: 4
Introduceti elementele:
3 5 7 9
Elementul de cautat: 7
7 este in tablou!
```

### **Analiza problemei și proiectarea soluției**

Vom parcurge tabloul secvențial de la stânga la dreapta și vom seta corespunzător variabila `este`, 0 dacă elementul dat nu se găsește în tablou și 1 în caz contrar. Dacă elementul nu este găsit, se vor deplasa toate elementele tabloului la dreapta cu o poziție și se completează prima poziție cu acesta.

### **Program**

```
#include <stdio.h>

void main(){
 int n, i;
 unsigned long a[201], elem;
 short este = 0;
 printf("Numar de elemente: ");
 scanf("%d", &n);
 printf("Introduceti elementele:\n");
 for(i=0; i<n; i++) scanf("%ld", &a[i]);
```

```
printf("Elementul de cautat: ");
scanf("%ld", &elem);
for(i=0; !este && i<n; i++)
 if(elem == a[i]) este = 1;
if(este) printf("\n %ld este in tablou!\n", elem);
else{
 printf("\n %ld NU este in tablou!\n", elem);
 for(i=n; i>0; i--) a[i] = a[i-1];
 a[0] = elem;
 printf(" Noul tablou: ");
 for(i=0; i<=n; i++) printf(" %d ", a[i]);
}
```

### **Exerciții**

1. Fiind date două tablouri bidimensionale cu numere întregi din intervalul [-500, 500] care au același număr de coloane, să se concateneze cele două tablouri pe verticală.
2. Fiind date două tablouri bidimensionale cu numere întregi din intervalul [-500, 500] cu același număr de linii, să se concateneze cele două tablouri pe orizontală. Numărul de linii, respectiv de coloane din cele două tablouri este maximum 100.

### **Problema 12. Sortarea coloanelor în matrice**

Fiind dat de la tastatură un tablou bidimensional cu m linii și n coloane,  $1 \leq m, n \leq 50$ , cu elemente numere întregi din intervalul [-5000, 5000], scrieți un program care transformă matricea astfel încât elementele de pe coloanele pare să fie ordonate crescător și elementele de pe coloanele impare să fie ordonate descrescător.

#### *Exemplu :*

```
Dimensiunile matricei: 5 6
Elementele:
231 5 67 98 0 43
321 45 6 76 8 0
21 34 5 87 9 231
5 89 -45 2 12 67
32 1 768 9 1 -23
```

#### Matricea transformată:

```
321 1 768 2 12 -23
231 5 67 9 9 0
32 34 6 76 8 43
21 45 5 87 1 67
5 89 -45 98 0 231
```

### Analiza problemei și proiectarea soluției

Pentru fiecare coloană de la 0 la n aplicăm tehnica de ordonare *Bubble Sort*, utilizată și în problema anterioară. Pentru a ști dacă vom realiza pentru coloana fixată j ordonare crescătoare sau descrescătoare, folosim variabila *semn*, care are valoarea -1 dacă respectiva coloană este impară și 1 dacă este pară. Astfel, cu j fixat, se compară elementele  $a[i][j]$  și  $a[k][j]$ , pentru toți  $k > i$ . Dacă elementele nu sunt în ordinea necesară, echivalent cu condiția:

$$(a[i][j] - a[k][j]) * \text{semn} > 0$$

atunci se realizează interschimbarea acestora prin secvența:

$$\begin{aligned} a[i][j] &= a[i][j] - a[k][j]; \\ a[k][j] &= a[i][j] + a[k][j]; \\ a[i][j] &= a[k][j] - a[i][j]; \end{aligned}$$

(metodă de interschimbare fără a utiliza variabilă auxiliară).

### Program

```
#include <stdio.h>

void main(){
 int a[50][50], n, m, semn;
 int i, j, k;
 printf("Dimensiunile matricei: ");
 scanf("%d %d", &m, &n);
 printf("Elementele:\n");
 for(i=0; i<m; i++)
 for(j=0; j<n; j++)
 scanf("%d", &a[i][j]);
 for(j=0; j<n; j++){
 if(j%2==0) semn = -1;
 else semn = 1;
 for(i=0; i<m-1; i++)
 for(k=i+1; k<m; k++)
 if((a[i][j]-a[k][j])*semn > 0){
 a[i][j] = a[i][j] - a[k][j];
 a[k][j] = a[i][j] + a[k][j];
 a[i][j] = a[k][j] - a[i][j];
 }
 }
 printf("\nMatricea transformata:\n");
 for(i=0; i<m; i++)
 printf("\n");
}
```

```
for(j=0; j<n; j++)
 printf("%4d", a[i][j]);
}
```

### Exercițiu

Transformați programul de mai sus astfel încât să ordoneze crescător elementele de pe *liniile pare* ale matricei și de către elementele de pe *liniile impare*.

### Problema 13. Elemente simetrice în tablou pătratic

Dat fiind de la tastatură un tablou bidimensional pătratic de dimensiuni maximum  $50 \times 50$  cu elemente numere naturale  $\leq 500$ , să se calculeze cmmdc și cmmmc ai elementelor simetrice față de diagonala principală. Pentru fiecare pereche se vor preciza pozițiile elementelor, elementele, cmmdc și cmmmc, ca și în exemplul următor.

#### Exemplu :

Matricea:  
 n = 4  
 Elementele:  
 43 12 6 236  
 8 378 36 6  
 76 432 124 23  
 65 7 488 38  
 ----

#### REZULTAT

Matricea introdusa:

|               |                    |              |                  |
|---------------|--------------------|--------------|------------------|
| 43            | 12                 | 6            | 236              |
| 8             | 378                | 36           | 6                |
| 76            | 432                | 124          | 23               |
| 65            | 7                  | 488          | 38               |
| <hr/>         |                    |              |                  |
| a[ 1 ][ 2 ] = | 12, a[ 2 ][ 1 ] =  | 8, cmmdc =   | 4, cmmmc = 24    |
| a[ 1 ][ 3 ] = | 6, a[ 3 ][ 1 ] =   | 76, cmmdc =  | 2, cmmmc = 228   |
| a[ 1 ][ 4 ] = | 236, a[ 4 ][ 1 ] = | 65, cmmdc =  | 1, cmmmc = 15340 |
| a[ 2 ][ 3 ] = | 36, a[ 3 ][ 2 ] =  | 432, cmmdc = | 36, cmmmc = 432  |
| a[ 2 ][ 4 ] = | 6, a[ 4 ][ 2 ] =   | 7, cmmdc =   | 1, cmmmc = 42    |
| a[ 3 ][ 4 ] = | 23, a[ 4 ][ 3 ] =  | 488, cmmdc = | 1, cmmmc = 11224 |

### Analiza problemei și proiectarea soluției

Parcugem zona matricei de deasupra diagonalei principale și analizăm perechile de elemente simetrice față de această diagonală. Pentru a calcula cel mai mare divizor comun utilizăm metoda scăderilor succesive, pe baza formulei:

$$\text{cmmdc}(a, b) = \begin{cases} a, a = b \\ \text{cmmdc}(a, b - a), b > a \\ \text{cmmdc}(a - b, b), a > b \end{cases}$$

Pentru a afla cel mai mic multiplu comun aplicăm formula:

$$\text{cmmmc}(a, b) = \frac{a \cdot b}{\text{cmmdc}(a, b)}$$

Elementele simetrice față de diagonala principală sunt  $a[i][j]$  und  $a[j][i]$ ,  $1 \leq i, j \leq n$ .

### Program

```
#include <stdio.h>

void main(){
 int a[50][50];
 int n, i, j;
 int p, q, cmmdc, cmmmc;
 printf("Matricea: \n");
 printf(" n = ");
 scanf("%d", &n);
 printf(" Elementele:\n");
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 scanf("%d", &a[i][j]);
 printf("----\nREZULTAT\n----\n");
 printf("Matricea introdusa:\n");
 for(i=0; i<n; i++){
 printf("\n");
 for(j=0; j<n; j++) printf("%5d", a[i][j]);
 }
 printf("\n\n");
 for(i=0; i<n-1; i++){
 for(j=i+1; j<n; j++){
 p = a[i][j]; q = a[j][i];
 while(p>q)
 if(p>q) p -= q;
 else q -= p;
 cmmdc = p;
 cmmmc = a[i][j]*a[j][i]/cmmdc;
 printf("a[%2d][%2d] = %4d, a[%2d][%2d] = %4d, ", i+1, j+1, a[i][j], j+1, i+1, a[j][i]);
 printf("cmmdc = %4d, cmmmc = %4d\n", cmmdc, cmmmc);
 }
 }
}
```

```
cmmdc = a[i][j]*a[j][i]/cmmdc;
printf("a[%2d][%2d] = %4d, a[%2d][%2d] = %4d, ",
 i+1, j+1, a[i][j], j+1, i+1, a[j][i]);
printf("cmmdc = %4d, cmmmc = %4d\n", cmmdc, cmmmc);
}
```

### Exercițiu

Rezolvați aceeași problemă pentru elementele simetrice față de diagonala secundară.

### Problema 14. Pointer de parcursere al unui tablou

Se citește de la tastatură un tablou care are foarte multe elemente nule. Să se parcure tabloul utilizând un pointer la `int` și să se afișeze primul element nenul utilizând acest pointer, ca în exemplul de mai jos.

*Exemplu :*

| intrare(tastatură)                                                          | ieșire(ecran)                      |
|-----------------------------------------------------------------------------|------------------------------------|
| n = 5<br>a[ 0 ] = 0<br>a[ 1 ] = 0<br>a[ 2 ] = 0<br>a[ 3 ] = 8<br>a[ 4 ] = 7 | Primul elem. nenul: 8 la pozitia 3 |

### Analiza problemei și proiectarea soluției

După citirea elementelor tabloului `a`, vom inițializa pointerul `p` cu adresa de început a tabloului, aceasta fiind chiar `a` (`a`, `a+1`, `a+2` etc. sunt adresele de început ale elementelor tabloului, `*a = a[0]`, `*(a+1) = a[1]`, `*(a+2) = a[2]` etc. sunt valorile tabloului). Ne vom deplasa prin incrementarea pointerului până găsim un element nenul:

```
while(*p==0 && p<a+n) p++;
```

Instrucțiunea `p++` în acest caz are ca efect deplasarea în memorie a pointerului `p` cu un număr de octeți egal cu dimensiunea tipului definit.

### Program

```
#include <stdio.h>
#include <conio.h>

void main(){
 int a[20], n, i, *p;
 printf("n = "); scanf("%d", &n);
```

```

for(i=0;i<n;i++) {
 printf("a[%2d] = ", i);
 scanf("%d", &a[i]);
}
p=a;
while(*p==0 && p<a+n) p++;
i=p-a;
if(i==n)
 printf("Nu există element nenul!");
else printf("Primul elem. nenul: %d la poziția %d", *p,i);
getch();
}

```

### Exercițiu

Afișați adresele elementelor nenule din vectorul a[].

### Problema 15. Adrese în tablouri

Scrieți un program care calculează factorialul primelor 6 numere naturale, utilizând o variabilă de tip tablou de întregi și un pointer la int care indică aceeași adresă de memorie.

*Exemplu :*

```

0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720

```

### Analiza problemei și proiectarea soluției

Vom inițializa pointerul p cu adresa de început a tabloului a[]. Pentru a demonstra că ele pointează la aceeași adrese de memorie se execută instrucțiunea :

```
* (p+i) = i*fact[i-1];
```

De fapt  $* (p+i) = fact[i]$  și  $p+i = fact+i$  pentru toți i de la 0 la dimensiunea tabloului.

### Program

| Varianta 1                                                                                                                                                                                                                                                                                                                                 | Varianta 2                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #define max 6  void main() {     unsigned long     fact[max+1], *p, i;     p=fact;     *p=1;     for(i=1;i&lt;=max;i++)         * (p+i)=i*fact[i-1];     printf("\n"); i=0;     for(;i&lt;=max;p++)         printf("\n%2d! = %d", i++,                *(p));     getch(); } </pre> | <pre> #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #define max 6  void main() {     unsigned long     fact[max+1], *p, i;     p=fact;     *p=1;     for(i=1;i&lt;=max;i++)         * (p+i)=i*fact[i-1];     printf("\n"); i=0;     for(i=0;i&lt;=max;i++)         printf("\n%2d! = %d", i,                fact[i]);     getch(); } </pre> |

### Exercițiu

Să se folosească același procedeu pentru afișarea numerelor din sirul lui Fibonacci :  $x_0 = x_1 = 1$ ,  $x_n = x_{n-1} + x_{n-2}$ , pentru  $\forall n > 1$ .

### Problema 16. Interschimbare de octeți

Scrieți o funcție cu antetul

```
void intersch(void *ob1, void *ob2, size_t dim)
```

care interschimbă conținutul a dim octeți.

*Exemplu :*

```

N1 = 123456
N2 = 98012

```

Interschimbare...

Dupa interschimbare:

```
N1 = 98012, N2 = 123456
```

### Analiza problemei și proiectarea soluției

Știind că tipul `char` are un octet, considerăm două variabile de tip `char*` cu care ne deplasăm de la adresele respective de dim ori și executăm o interschimbare a conținutului:

```
aux =*(p+i);
(p+i)=(q+i);
*(q+i)=aux;
```

Vom utiliza funcția în program pentru a interschimba două valori de tip `long int`.

### Program

```
#include <conio.h>
#include <stdio.h>

void intersch(void *ob1, void *ob2, size_t dim) {
 char *p, *q, aux;
 int i;
 p=(char*)ob1;
 q=(char*)ob2;
 for(i=0;i<(int)dim;i++) {
 aux=*(p+i);
 (p+i)=(q+i);
 *(q+i)=aux;
 }
}

void main() {
 long int n1, n2;
 printf("N1 = ");
 scanf("%ld",&n1);
 printf("N2 = ");
 scanf("%ld",&n2);
 puts("\nInterschimbare...");
 intersch(&n1,&n2,sizeof(long int));
 puts("\nDupa interschimbare:");
 printf("N1 = %ld, N2 = %ld", n1, n2);
 getch();
}
```

### Exercițiu.

Testați funcția și pentru alte tipuri de date: de exemplu, `unsigned, double, tablouri`.

### Problema 17. Tablou cu pointeri la funcții

Scrieți o funcție care returnează o valoare de tip `double` reprezentând multiplul cu 100 al parametrului. Folosiți un tablou de pointeri la funcții pentru a afișa în forma prezentată mai jos toate valorile funcțiilor respective între 0.01 și 1.00 cu un pas de 0.1.

Exemplu:

| VAL   | SIN   | COS   | TAN    | EXP   | LOG    | FUN    |
|-------|-------|-------|--------|-------|--------|--------|
| 0.010 | 0.010 | 1.000 | 0.010  | 1.010 | -4.605 | 1.000  |
| 0.110 | 0.110 | 0.994 | -0.110 | 1.116 | -2.207 | 11.000 |
| 0.210 | 0.208 | 0.978 | 0.213  | 1.234 | -1.561 | 21.000 |
| 0.310 | 0.305 | 0.952 | 0.320  | 1.363 | -1.171 | 31.000 |
| 0.410 | 0.399 | 0.917 | 0.435  | 1.507 | -0.892 | 41.000 |
| 0.510 | 0.488 | 0.873 | 0.559  | 1.665 | -0.673 | 51.000 |
| 0.610 | 0.573 | 0.820 | 0.699  | 1.840 | -0.494 | 61.000 |
| 0.710 | 0.652 | 0.758 | 0.860  | 2.034 | -0.342 | 71.000 |
| 0.810 | 0.724 | 0.689 | 1.050  | 2.248 | -0.211 | 81.000 |

### Analiza problemei și proiectarea soluției

După definirea funcției `fun1()` vom declara tabloul de pointeri la funcții astfel:

```
double (*tabfun[])(double)={sin,cos,tan,exp,log,fun1};
```

Pentru o valoare `x` cu proprietatea specificată în enunț, funcția `i` (`0→sin, ..., 5→fun1`) are valoarea:

```
(*tabfun[i])(x))
```

### Program

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

double fun1(double x){
 return x*100;
}

void main(){
 int i;
 double x;
 double (*tabfun[])(double)={sin,cos,tan,exp,log,fun1};
 printf(
 "-----\n"
);
```

```

);
 printf(
 " | VAL | SIN | COS | TAN | EXP | LOG | FUN | \n
 ");
 printf(
 "-----\n"
);
 for(x=0.01;x<1.01;x+=0.10){
 printf("%8.3f",x);
 for(i=0;i<6;i++)
 printf("%8.3f",(*tabfun[i])(x));
 printf("\n");
 }
 getch();
}

```

### Exercițiu

Ce execută următorul program?

```

#include <stdio.h>
#include <stdlib.h>

void func(int);

main(){
 void (*pf)(int);
 pf = func;
 (*pf)(2);
 pf(5);
}

void func(int arg){
 printf("%d\n", arg);
}

```

### Problema 18. Puncte-șa

Dată fiind o matrice cu elemente numere întregi, să se determine punctele-șa din matrice, adică acele puncte care sunt minime pe linia lor și maxime pe coloana lor.

### Exemplu:

| intrare (tastatură)                                                                                                                                                                                                                          | ieșire (ecran)                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| Dimensiunile matricei:<br>Nr. de linii: 3<br>Nr. de coloane : 3<br>Introduceti elementele matricei:<br>a[1][1] = 4<br>a[1][2] = 3<br>a[1][3] = 6<br>a[2][1] = 12<br>a[2][2] = 1<br>a[2][3] = 4<br>a[3][1] = 54<br>a[3][2] = 2<br>a[3][3] = 5 | Matricea introdusa:<br>4 3 6<br>12 1 4<br>54 2 5<br><br>a[1,2] = 3 este punct sa!<br><br>Numar puncte sa: 1 |

### Analiza problemei și proiectarea soluției

Vom parcurge toate elementele matricei și pentru fiecare decidem dacă îndeplinește condiția de punct-șa:

```

ok=1; /* pentru elementul a[i][j] */
for(k=0;k<m;k++)
 if (a[i][k]<a[i][j]) ok=0;
for(k=0;k<n;k++)
 if (a[k][j]>a[i][j]) ok=0;

```

### Program

```

#include <stdio.h>;
#include <conio.h>;

void main(){
 int ok , i , j , k , m , n , a[20][20];
 int nrPSa = 0;
 printf("Dimensiunile matricei: ");
 printf("\nNr. de linii: ");
 scanf("%d",&n);
 printf("Nr. de coloane: ");
 scanf("%d",&m);
 printf("Introduceti elementele matricei : \n");
 for(i=0;i<n;i++)
 for(j=0;j<m;j++){
 printf("a[%d][%d] = ",i+1,j+1);
 scanf("%d",&a[i][j]);
 }
}
```

```

printf("\nMatricea introdusa: \n");
for(i=0;i<n;i++){
 for(j=0;j<m;j++)
 printf("%5d",a[i][j]);
 printf("\n");
}
for(i=0;i<n;i++)
 for(j=0;j<m;j++) {
 ok=1;
 for(k=0;k<m;k++)
 if(a[i][k]<a[i][j]) ok=0;
 for(k=0;k<n;k++)
 if(a[k][j]>a[i][j]) ok=0;
 if(ok){
 printf("\n a[%d,%d] = %d este punct sa!\n",
 i+1,j+1,a[i][j]);
 nrPSa++;
 }
 }
if(nrPSa)printf("\nNumar puncte sa: %d", nrPSa);
else printf("\nNu s-a gasit nici un punct sa!");
getch();
}

```

### Exercițiu

Dat fiind un tablou bidimensional  $A$  pătratic cu elemente numere naturale  $\leq 500$ , să se afle elementele maxime în modul pe linia lor și minime în modul pe coloana lor.

### Probleme propuse

- Să se testeze dacă o matrice pătratică cu elemente numere întregi din intervalul  $[-250, 250]$  este părat magic (dacă suma elementelor de pe fiecare linie, coloană și diagonală este aceeași).

*Exemplu :*

```

n = 4
Introduceti matricea:

```

```

16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1

```

Patrat magic cu suma magica: 34.

- O matrice  $A_{m \times n}$  conține numere naturale din intervalul  $[1, 100]$ . Să se afișeze numerele distincte din fiecare linie a matricei și apoi, dacă există, numerele comune tuturor liniilor.

- Se citește de la tastatură un număr natural  $n$  ( $1 \leq n \leq 10$ ). Se cere să se construiască un tablou bidimensional  $A$  cu  $n$  linii și  $n$  coloane cu primii  $n \cdot n$  termeni ai șirului lui Fibonacci. Ordinea de construire va fi:  $A[0][0], A[0][1], A[0][n-1], \dots, A[n-1][0], \dots, A[n-1][n-1]$ . Să se scrie matricea pe ecran. Dându-se de la tastatură un termen al șirului lui Fibonacci, să se afișeze linia și coloana în tabloul  $A$  acolo unde se află valoarea respectivă. Dacă termenul nu se află în tablou, atunci se va afișa pe ecran propoziția: "Nu este în tablou!".
- Traseul calului.* Se dă o matrice pătratică de dimensiuni  $N \times N$  ( $N \leq 50$ ) ce conține toate valorile de la 1 la  $N^2$ . Matricea codifică posibile deplasări succesive ale unui cal pe o tablă de săh de dimensiunea  $N \times N$ , acoperind toate pătrătelele acestaia. Punctul de plecare al calului este dat de valoarea 1 din matrice. Se cere să se stabilească dacă matricea considerată reprezintă sau nu un traseu corect de acoperire a tablei de săh prin deplasări ale calului (conform regulilor de deplasare a acestei piese în L la jocul de săh), începând cu poziția initială.

*Date de intrare :*

Se citesc din fișierul text *traseu.in*, care are următoarea structură:

```

N //numarul de linii si de coloane ale matricei
A11 A12 A13 ... Aln // linia 1 din matrice
A21 A22 A23 ... A2n // linia 2 din matrice
.....
An1 An2 An3 ... Ann // linia n din matrice

```

*Date de ieșire :*

Se vor scrie în fișierul *traseu.out* care va avea următoarea structură (două variante posibile):

- va conține pe prima linie răspunsul DA dacă matricea reprezintă un traseu corect;
- va conține pe prima linie răspunsul NU dacă matricea nu reprezintă un traseu corect, iar pe a doua linie se va scrie pasul din care calul nu-și poate continua deplasarea, precum și poziția lui pe tablă (indicii).

*Exemple :*

| traseu.in                                                           | traseu.out |
|---------------------------------------------------------------------|------------|
| 5 25 6 9 16 23 24 15 2 7 10 5 8 11 22 13 20 1 14 3 19 17 4 18 12 21 | NU 16 1 4  |

- Celule inteligente.* Un biolog experimentează comportarea ADN-ului asupra unei culturi liniare de bacterii. Prin schimbarea ADN-ului, el este capabil să programeze bacteriile să fie influențate de bacteriile vecine. Populația unei culturi este măsurată pe o scală de patru puncte (de la 0 la 3). Informația ADN-ului este reprezentată ca un tablou ADN, indexat de la 0 la 9, iar valorile de densitate a populației sunt interpretate astfel:
  - în orice cultură dată de bacterii, fie  $k$  suma densităților acestei culturi și a densităților culturilor vecine la stânga și la dreapta. Atunci, în următoarea zi, cultura respectivă va avea densitatea populației  $ADN[K]$ ;

- cultura cea mai din stânga se consideră că are vecinul stâng cu populație de densitate 0 ;
  - cultura cea mai din dreapta se consideră că are vecinul drept cu populație de densitate 0 .

În consecință, unele programe ADN cauzează moartea bacteriilor, altele pot duce la o explozie de populație. Biologul este interesat cum evoluează aceste programe ADN. Scrieți un program care simulează creșterea unei culturi de bacterii pe o structură liniară de 40 astfel de culturi, presupunând că a 20-a începe cu o populație de densitate 1 și toate celelalte culturi au densitatea populației 0.

*Date de intrare:*

În fișierul de intrare `bacterii.in` se găsește pe prima linie numărul natural  $n$  ( $1 \leq n \leq 50$ ) și pe a doua linie un vector DNA [], adică zecă valori cuprinse între 0 și 3 inclusiv.

### *Date de iesire -*

În fișierul de ieșire bacterii.out se va scrie densitatea celor 40 de culturi pe următoarele n zile. Fiecare zi este scrisă pe o linie a fișierului de ieșire și ocupă 40 de caractere. Fiecare cultură este reprezentată de un singur caracter pe acea linie. Pentru o populație de densitate 0 se va scrie caracterul 'b', pentru una de densitate 1 se va scrie caracterul '.', pentru o populație de densitate 2 se va scrie caracterul 'x', pentru o populație de densitate 3 se va scrie caracterul 'w'.

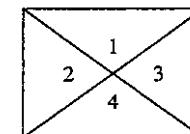
*Exemplu:*

6. Fișierul `tablou.in` conține pe prima linie numărul  $m$  de linii și numărul  $n$  de coloane, separate printr-un spațiu ( $1 \leq m, n \leq 50$ ), ale unui tablou bidimensional cu elemente 0 și 1. Pe următoarele  $m$  linii se află câte  $n$  numere 0 sau 1. Se cere:

  - să se scrie în fișierul `tablou.out` pe prima linie numărul maxim de valori 1 conținute de o coloană.
  - Incepând cu linia a doua, fișierul `tablou.out` va conține perechile de linii complementare (numărul corespunzător liniilor din tablou cu spațiu între ele). Două linii se numesc complementare dacă suma oricărora două elemente de pe aceeași coloană este 1. *Exemplu*: 110101 și 001010.

7. Se consideră un vector conținând  $n$  ( $n \leq 50$ ) numere naturale cu valori cuprinse între 0 și 30.000. Să se scrie un program care să ordoneze crescător doar numerele impare din vector, fără însă a afecta pozițiile pe care se află numerele pare. Datele de intrare se citesc de la tastatură și constau din numărul  $n$  urmat de cele  $n$  numere naturale ce constituie sirul. Programul va afișa pe ecran vectorul obținut după ordonarea elementelor impare.

8. Un tablou bidimensional pătratic cu numere naturale  $\leq 30.000$  se împarte în patru regiuni astfel:



(se vor exclude diagonale)

Scriți un program care determină numerele prime din regiunea 1 și suma elementelor din regiunea 4, numărul elementelor impare din regiunea 2 și a celor pare din regiunea 3.

9. Scrieți un program care citește de la tastatură un vector a cu maximum 50 de elemente numere reale și numără câte dintre acestea sunt mai mici decât media lor aritmetică.

10. Scrieți un program care calculează puterea a n-a a unei matrice pătratice de dimensiuni  $m \times m$  ( $2 \leq m, n \leq 10$ ).

11. *Ciori și copaci.* Pe n copaci așezăți în cerc se află  $n-1$  ciori, maximum una în fiecare copac. Dându-se o configurație inițială și una finală a ciorilor și știind că în fiecare moment poate zbură o cioară de pe copacul său curent pe cel liber, determinați modul în care o fac pentru a ajunge în configurația finală.

*Datele de intrare* se citesc de la tastatură și *datele de ieșire* se scriu pe ecran, sub forma din exemplul următor: pe prima linie – numărul n de copaci, pe următoarea linie – configurația initială, pe următoarea linie – configurația finală a copacilor.

| tastatură   | ecran                                         |
|-------------|-----------------------------------------------|
| 6           | Cioara 1 zboara de pe copacul 1 pe copacul 3! |
| 1 2 0 3 4 5 | Cioara 3 zboara de pe copacul 4 pe copacul 1! |
| 3 5 1 0 2 4 | Cioara 2 zboara de pe copacul 2 pe copacul 4! |
| .           | Cioara 5 zboara de pe copacul 6 pe copacul 2! |
|             | Cioara 4 zboara de pe copacul 5 pe copacul 6! |
|             | Cioara 2 zboara de pe copacul 4 pe copacul 5! |

## **Şiruri de caractere. Operaţii cu fişiere**

**12 probleme complet rezolvate, 50 de exerciţii şi probleme propuse.**  
**Probleme rezolvate :**

- Copierea unui sir constant
- Utilizarea funcţiilor `strcat()`, `strlen()`, `strchr()`, `strstr()`
- Compararea a două şiruri
- Primul subşir care începe cu un caracter din alt şir
- Găsirea unui subşir într-un şir
- Adunarea a două numere în baza p
- Secvenţă de căutat
- Păsăreşte
- Utilizarea funcţiilor `strncpy()`, `strcat()`, `strlen()`
- Utilizarea funcţiei `strtok()`
- Concatenarea a două fişiere
- Argumente în linia de comandă

*Niciodată nu se minte mai mult ca înainte de alegeri,  
în timpul războiului și după vânătoare.*

Otto von Bismarck

### ***Problema 1. Copierea unui sir constant (caracter cu caracter, pointer)***

Scrieți un program simplu, în care să se evidențieze cele două modalități de a trata un sir: la nivel de caracter și la nivel de pointer.

#### ***Analiza problemei și proiectarea soluției***

Vom declara un sir constant mesaj și vom implementa două metode de copiere a conținutului acestuia în alte două siruri, suficient alocate la declarare.

#### ***Program***

```
#include "string.h"
#include "stdio.h"

void main(){
 char text_1[100], text_2[100];
 char *ta, *tb;
 int i;
 char mesaj[] = "Buna, eu sunt un sir; ce esti tu?";
 printf("Mesajul original: %s\n", mesaj);
 /* prima metoda: caracter cu caracter */
 i=0
 while(mesaj[i] != '\0'){
 text_1[i] = mesaj[i];
 i++;
 }; text_1[i] = '\0';
 printf("Text_1: %s\n", text_1);
 /* a doua metoda: folosind un pointer */
 ta=mesaj;
 tb=text_2;
 while(*ta != '\0'){
 *tb = *ta;
 tb++; ta++;
 }; *tb = '\0';
 printf("Text_2: %s\n", text_2);
}
```

*Exerciții propuse*

1. Încercați să rulați programul cu o alocare insuficientă de memorie pentru unul dintre șiruri, de exemplu, pentru

```
char text_1[100], text_2[3];
```

în linia de declaratie. Ce constatați?

2. Analizați și următoarele două metode de realizare a copierii, relativ la posibilitățile limbajului C:

| copiere caracter cu caracter                           | copiere folosind pointeri                                      |
|--------------------------------------------------------|----------------------------------------------------------------|
| i=0;<br>while(<br>(text_1[i] = mesaj[i]) != '\0') i++; | ta=mesaj;<br>tb=text_2;<br>while(<br>(*tb++ = *ta++) != '\0'); |

prin înlocuirea lor în program.

**Problema 2. Utilizarea funcțiilor strcat, strlen, strchr, strrchr**

Scrieți un program simplu care citește un șir, concatenează un altul la sfârșit și efectuează operații de căutare a unui caracter.

*Analiza problemei și proiectarea soluției*

Vom scrie un program în care ieșirea să aibă forma:

```
Sir citit: Buna, eu sunt un sir.

<strcat>Dupa apel: Buna, eu sunt un sir. Ce esti tu?

<strlen>Lungimea sirului: 33

<strchr>Sirul ce incepe cu "e" -> eu sunt un sir. Ce esti tu?
<strchr>Sirul ce incepe cu "s" -> sunt un sir. Ce esti tu?
<strchr>Sirul ce incepe cu "C" -> Ce esti tu?

Sirul s este -> Buna, eu sunt un sir. Ce esti tu?
<strrchr>Sirul ce incepe cu "u" -> u?
```

Programul următor este sugestiv în acest scop.

*Program*

```
#include <stdio.h>
#include <string.h>

void main(){
 char s[100], *sub_text;

 /* initializare sir */
 strcpy(s, "Buna, eu sunt un sir.");
 printf("Sir citit: %s\n", s);

 /* concatenare la sfârșitul sirului */
 strcat(s, " Ce esti tu?");
 printf("\n<strcat>Dupa apel: %s\n", s);

 /* determina lungimea unui sir */
 /* strlen returneaza lungimea de tip size_t */
 printf("\n<strlen>Lungimea sirului: %d\n", (int)strlen(s));

 /* gaseste prima aparitie a unui caracter */
 if(s != NULL && (sub_text = strchr (s, 'e'))!= NULL)
 printf("\n<strchr>Sirul ce incepe cu \"e\" -> %s\n", sub_text);
 if(s != NULL && (sub_text = strchr (sub_text, 's'))!= NULL)
 printf("<strchr>Sirul ce incepe cu \"s\" -> %s\n", sub_text);
 if(s != NULL && (sub_text = strchr (sub_text, 'C'))!= NULL)
 printf("<strchr>Sirul ce incepe cu \"C\" -> %s\n", sub_text);

 /* gaseste ultima aparitie a unui caracter */
 printf("\nSirul s este -> %s\n", s);
 if(s != NULL && (sub_text = strrchr (s, 'u'))!= NULL)
 printf("<strrchr>Sirul ce incepe cu \"u\" -> %s\n", sub_text);
}
```

*Exercițiu*

Testați ce se întâmplă prin aplicarea uneia dintre funcțiile strchr sau strrchr unui șir nul.

**Problema 3. Compararea a două șiruri (strcmp, strncmp)**

Scrieți un program care utilizează metodelor strcmp și strncmp pentru a compara două șiruri date.

*Analiza problemei și proiectarea soluției*

Vom scrie un program a cărui ieșire să fie:

Compara sirurile:

Elefantul roz e mai roz decat porcusorul albastru.  
Elefantul roz e mai ROZ decat porcusorul albastru.

strcmp: Sirul 1 este mai mare decat sirul 2  
strcmp: Sirul 1 este egal cu sirul 2

Programul următor este sugestiv în acest sens (este folosită și funcția strcpy).

*Program*

```
#include <string.h>
#include <stdio.h>
#include <conio.h>

char string1[] =
"Elefantul roz e mai roz decat porcusorul albastru.";
char string2[] =
"Elefantul roz e mai ROZ decat porcusorul albastru.";

int main(void){
 char tmp[20];
 int result;
 /* Case sensitive */
 printf("Compara sirurile:\n %s\n %s\n\n", string1, string2);
 result = strcmp(string1, string2);
 if(result > 0)
 strcpy(tmp, "mai mare decat");
 else if(result < 0)
 strcpy(tmp, "mai mic decat");
 else
 strcpy(tmp, "egal cu");
 printf(" strcmp: Sirul 1 este %s sirul 2\n", tmp);
 /* Case insensitive */
 result = stricmp(string1, string2);
 if(result > 0)
 strcpy(tmp, "mai mare decat");
 else if(result < 0)
 strcpy(tmp, "mai mic decat");
 else
 strcpy(tmp, "egal cu");
}
```

```
printf(" stricmp: Sirul 1 este %s sirul 2\n", tmp);
getch();
}
```

*Exercițiu*

Modificați programul astfel încât să folosiți și funcțiile strncmp și strnicmp.

*Problema 4. Primul subșir care începe cu un caracter din alt șir (strupbrk)*

Scrieți un program care utilizează sugestiv metoda strupbrk pentru a returna șirul care începe cu un caracter din alt șir.

*Analiza problemei și proiectarea soluției*

Vom scrie un program a cărui ieșire să fie:

```
1: Cei 2 copaci plus 3 masini fac 5 portocale
2: 2 copaci plus 3 masini fac 5 portocale
3: 3 masini fac 5 portocale
4: 5 portocale
```

Programul următor este sugestiv în acest sens.

*Program*

```
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void){
 char string[100] =
 "Cei 2 copaci plus 3 masini fac 5 portocale\n";
 char *result;
 /*return: pointer la primul caracter din "string"*/
 printf("1: %s\n", string);
 result = strupbrk(string, "0123456789");
 printf("2: %s\n", result++);
 result = strupbrk(result, "0123456789");
 printf("3: %s\n", result++);
 result = strupbrk(result, "0123456789");
 printf("4: %s\n", result);
 getch();
}
```

**Exercițiu**

Modificați programul astfel încât să citească de la tastatură (sau dintr-un fișier) un text și să scrie mai apoi pe ecran (sau într-un fișier) toate propozițiile din text, ținând cont că fiecare propoziție începe cu literă mare.

**Problema 5. Găsirea unui subșir într-un sir (strstr)**

Scrieți un program care ilustrează utilizarea funcției strstr pentru căutarea unui subșir într-un sir și care scrie și poziția la care subșirul a fost găsit.

**Analiza problemei și proiectarea soluției**

Vom scrie un program al cărui rezultat va fi de forma:

Sirul de căutat:

Primul pas este cel mai greu.

|                                |   |   |
|--------------------------------|---|---|
| 1                              | 2 | 3 |
| 123456789012345678901234567890 |   |   |

greu gasit la pozitia 25.

**Program**

```
#include <string.h>
#include <stdio.h>
#include <conio.h>

char str[] = "greu";
char string[] = "Primul pas este cel mai greu.";
char fmt1[] = " 1 2 3";
char fmt2[] = "123456789012345678901234567890";

int main(void){
 char *pdest;
 int result;
 printf("Sirul de căutat:\n%s\n", string);
 printf("\n%s\n%s\n", fmt1, fmt2);
 pdest = strstr(string, str);
 result = (int)(pdest - string + 1);
 if(pdest != NULL)
 printf("%s gasit la pozitia %d.\n", str, result);
 else
 printf("%s nu a fost gasit.\n", str);
 getch();
}
```

**Exercițiu**

Modificați programul astfel încât șirurile să fie citite de la tastatură și să se găsească toate aparițiile, cu pozițiile acestora, unui subșir în celălalt.

**Problema 6. Adunarea a două numere (utilizarea funcțiilor strlen, strcat)**

Scrieți un program care să adune două numere în baza p ( $2 \leq p \leq 10$ ) și să afișeze rezultatul. Numerele vor fi memorate în tablouri (lungimea maximă a acestora fiind 100). Toate datele vor fi preluate de la utilizator.

**Exemplul 1 :****DATE DE INTRARE:**

Primul numar: 54301256

Al doilea numar: 12131314322521312312312

Baza p = 7

**OPERATIA DE ADUNARE:**

|                         |   |
|-------------------------|---|
| 54301256                | + |
| -----                   |   |
| 12131314322521312312312 |   |
| -----                   |   |
| 12131314322521366613601 |   |

**Exemplul 2 :****DATE DE INTRARE:**

Primul numar: 1238258372583752873425648234872548723

Al doilea numar: 2352436234263246342634

Baza p = 9

**OPERATIA DE ADUNARE:**

|                                       |   |
|---------------------------------------|---|
| 1238258372583752873425648234872548723 | + |
| -----                                 |   |
| 2352436234263246342634                |   |
| -----                                 |   |
| 1238258372583755335862883508230002457 |   |

### *Analiza problemei și proiectarea soluției*

Definim tipul string ca fiind char[102], pe care îl vom utiliza în manipularea numerelor. Vom scrie funcția NumarValid(), care verifică dacă un sir are toate caracterele cifre, adică dacă este reprezentarea unui număr natural. Funcția ReverseString() transformă un sir în imaginea lui în oglindă. Realizăm acest lucru interschimbând elementele simetrice față de mijloc. Funcția AfiseazaNumar() va afișa sirul dat într-un format specific, de exemplu, apelul AfiseazaNumar("1234567", 11) va construi intern sirul sFormat="\n11s" și va executa printf(sFormat, "1234567"), ceea ce va avea ca efect scrierea sirului dat pe 11 caractere, aliniat la dreapta. Secvența:

```

while(n){
 sAux[i++]='0'+n % 10;
 n /=10;
}
sAux[i]='\0';
ReverseString(sAux);

```

transformă numărul natural n într-un sir de caractere sAux; în acest scop putem utiliza și funcția itoa() din biblioteca stdlib.h.

Funcția Aduna() parcurge algoritmul de calcul al sumei, însumând cifrele de pe poziții similare, pornind de la dreapta spre stânga și utilizând variabila t pentru transport.

### *Program*

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

typedef char string[102];

int NumarValid(string s){
 int aux = 1;
 unsigned i;
 if(!s) return 0;
 for(i=0; i<strlen(s); i++)
 if(!isdigit(s[i])) aux = 0;
 return aux;
}

int ReverseString(string s){
 unsigned i;
 char c;
 for(i=0; i<strlen(s)/2; i++){
 c = s[i];
 s[i] = s[strlen(s)-1-i];

```

```

 s[strlen(s)-1-i] = c;
 }
 return 0;
}

int AfiseazaNumar(string s, int n){
 string sAux;
 int i = 0;
 string sFormat;
 strcpy(sFormat, "\n%");
 while(n){
 sAux[i++]='0'+n%10;
 n /=10;
 }
 sAux[i]='\0';
 ReverseString(sAux);
 strcat(sFormat, sAux);
 strcat(sFormat, "s");
 printf(sFormat, s);
 return 0;
}

int Aduna(string s1, string s2, short p, string rez){
 int i;
 int t=0;
 int l1 = strlen(s1);
 int l2 = strlen(s2);
 i = 0;
 while(l1 - i && l2 - i){
 t = t + s1[l1-1-i] + s2[l2-1-i] - 2*'0';
 rez[i++] = (char)('0'+t % p);
 t = t / p;
 }
 while(l1 - i){
 t = t + s1[l1-1-i]-'0';
 rez[i++] = (char)('0'+t % p);
 t = t / p;
 }
 while(l2 - i > 0){
 t = t + s2[l2-1-i]-'0';
 rez[i++] = (char)('0'+t % p);
 t = t / p;
 }
 if(t) rez[i++] = (char)('0'+t);
 rez[i++] = '\0';
}

```

```

 ReverseString(rez);
 return 0;
}

int CitesteDate(string s1, string s2, short* baza){
 printf("DATE DE INTRARE: \n\n ");
 do{
 printf("Primul numar: ");
 gets(s1);
 }
 while(!NumarValid(s1));
 do{
 printf(" Al doilea numar: ");
 gets(s2);
 }
 while(!NumarValid(s2));
 printf(" Baza p = ");
 scanf("%d", baza);
 return 0;
}

int ScrieRezultat(string s1, string s2, string rez){
 unsigned i;
 printf("\n\nOPERATIA DE ADUNARE: \n");
 AfiseazaNumar(s1, strlen(rez) + 1);
 printf(" + ");
 AfiseazaNumar(s2, strlen(rez) + 1);
 printf("\n");
 for(i=0; i<strlen(rez)+1; i++) printf("-");
 AfiseazaNumar(rez, strlen(rez) + 1);
 return 0;
}

void main(){
 string s1, s2, rez;
 short p;
 CitesteDate(s1, s2, &p);
 Aduna(s1, s2, p, rez);
 ScrieRezultat(s1, s2, rez);
}

```

*Exerciții*

- Modificați programul anterior astfel încât AfiseazaNumar() să utilizeze funcția de bibliotecă itoa().
- Scrieți un program asemănător care execută operația de înmulțire în baza p ( $2 \leq p \leq 10$ ) a două numere cu maximum 100 de cifre. Afisarea operației de înmulțire trebuie să se facă desfășurat, ca la matematică.

*Exemplu :*

| Intrare (tastatură)                                                         | Ieșire (écran)                                                                  |
|-----------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| DATE DE INTRARE:<br>Primul numar: 123<br>Al doilea numar: 97<br>Baza p = 10 | OPERATIA DE INMULTIRE:<br>123 *<br>97<br>-----<br>861<br>1107<br>-----<br>11931 |

- Un număr natural N se numește *număr mare* dacă are cel mult 200 de cifre. Să se scrie un program care:
  - calculează suma a două *numere mari* reprezentate ca vectori;
  - calculează produsul unui *număr mare* cu o cifră;
  - calculează produsul a două *numere mari*, folosind tablouri unidimensionale.

*Problema 7. Secvență de căutat*

Se citește de la tastatură un număr natural cu cel mult 100 de cifre. Determinați cea mai lungă secvență de cifre consecutive, poziția de început a acesteia și conținutul; în caz că există mai multe secvențe de aceeași dimensiune maximă, afișați-o pe prima din sir.

*Exemplu :*

```

Numarul: 4321560123456982345678012
Lungimea maxima in sir: 7
Prima pozitie: 6
Subsirul: 0123456

```

*Analiza problemei și proiectarea soluției*

Vom defini tipul TNumar ca fiind char[102].

Funcția GasestePozitia() parcurge secvențial elementele sirului dat, sărind peste elementele consecutive și „reținând” informațiile necesare despre succesiunea curentă: iCurrent (poziția de început a secvenței) și i (poziția de început a secvenței succesoare). Parametrul lMax al funcției se modifică corespunzător și valoarea returnată de funcție este prima poziție a primei secvențe de dimensiune maximă din sir.

*Program*

```
#include <stdio.h>
#include <string.h>

typedef char TNumar[102];

int GasestePozitia(TNumar s, int* lMax) {
 int i=0;
 int iMax = 0;
 int iCurrent = 0;
 *lMax = 1;
 while(i <= strlen(s)){
 if(i - iCurrent > *lMax){
 iMax = iCurrent;
 *lMax = i - iCurrent;
 }
 iCurrent = i;
 while(s[i]+1 == s[i+1] && i<strlen(s)) i++;
 i++;
 }
 return iMax;
}

void main(){
 int lMax, iMax, i;
 TNumar s;
 printf("Numarul: ");
 gets(s);
 iMax = GasestePozitia(s, &lMax);
 printf("Lungimea maxima in sir: %d\n", lMax);
 printf("Prima pozitie: %d\n", iMax);
 printf("Subsirul: ");
 for(i=iMax; i<iMax+lMax; i++)
 printf("%c", s[i]);
}
}
```

*Exerciții*

- În caz că există mai multe secvențe de dimensiune maximă și se dorește afișarea ultimei, programul anterior se modifică doar prin adăugarea unui caracter. Operați această modificare.
- Modificați programul anterior astfel încât să se afișeze toate secvențele de lungime maximă, în caz că există mai multe.

*Problema 8. Păsărește*

Dându-se o propoziție, se cere propoziția transformată prin înlocuirea fiecărei vocale <v> cu grupul <v><p><v>. Se cere și implementarea transformării inverse, ca în exemplul de mai jos.

*Exemplu:*

LIMBA PASAREASCA

1. NORMAL -> PASARESTE (cola ->copolapa)
2. PASARESTE -> NORMAL (copolapa -> cola)
3. IESIRE.

Optiunea dvs. (1 sau 2 sau altceva): 1

Propozitia: Nu Te Intreba Ce Fac Altii, Mai Bine Vezi-Ti De Treaba Ta.

Transformare: Nupu Tepe IPIntrepebapa Cepe Fapac APAltipiipi, Mapaipi Bipinepe Vepezipi-Tipi Depe Trepeapabapa Tapa.

Optiunea dvs. (1 sau 2 sau altceva): 2

Propozitia: Faparapa Papasipiupunepe Nupu Epexipistapa Gepenipiupu. Transformare: Fara Pasiune Nu Exista Geniu.

Optiunea dvs. (1 sau 2 sau altceva): 1

Propozitia: Orice Inceput Este Greu.

Transformare: OPOripicepe&cPIncep&eput EPEstepe Grepeupu.

Optiunea dvs. (1 sau 2 sau altceva): 3

TERMINARE

*Analiza problemei și proiectarea soluției*

Vom scrie o funcție cu antetul:

```
int vocala(char ch)
```

Care decide dacă o literă este vocală, utilizând funcția `strchr()` din biblioteca `string.h`. Funcția `tr_pas()` va parcurge cuvântul dat ca parametru și la întâlnirea fiecărei vocale <v> scrie pe ecran formația de trei litere <v><p><v>, altfel scrie numai caracterul întâlnit.

*Program*

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>

#define VOCALE "aeiouAEIOU"

int vocala(char ch){
 return (strchr(VOCALE, ch) != NULL);
}

char P(char c){
 if (islower(c)) return 'p';
 else return 'P';
}

void tr_pas(char *cuv){
 int i;
 for(i=0;i<(int)strlen(cuv);i++)
 if(vocala(cuv[i])){
 printf("%c%c%c",cuv[i],P(cuv[i]),cuv[i]);
 }
 else{
 printf("%c",cuv[i]);
 }
}

void tr_nor(char * cuv){
 int i=0;
 while(i<(int)strlen(cuv)){
 if(vocala(cuv[i])&&tolower(cuv[i+1])=='p'
 &&tolower(cuv[i+2])==tolower(cuv[i]))
 i+=2;
 printf("%c",cuv[i]);
 i+=1;
 }
}

void main(){
 char* cuv, c;
 cuv=(char*)malloc(200*sizeof(char));

```

```
printf("\n LIMBA PASAREASCA");
printf("\n 1. NORMAL -> PASARESTE (cola ->copolapa)");
printf("\n 2. PASARESTE -> NORMAL (copolapa -> cola)");
printf("\n 3. IESIRE.");
do{
 printf("\n\n Optiunea dvs. (1 sau 2 sau altceva): ");
 c=getchar();
 if(!(c=='1' || c == '2')){
 printf("\n_____TERMINARE_____");
 continue;
 }
 printf("\nPropozitia: ");
 cuv=(char*)malloc(200*sizeof(char));
 gets(cuv); gets(cuv);
 printf("Transformare: ");
 if(c=='1')tr_pas(cuv);
 else if (c=='2')tr_nor(cuv);
}while(c == '1' || c=='2');
getch();
}
```

*Exercițiu*

Transformați funcțiile `tr_pas()` și `tr_nor()` astfel încât șirurile rezultate să fie returnate în vederea unor prelucrări ulterioare și afișarea să se facă doar în programul principal.

*Problema 9. Utilizarea funcțiilor `strncpy`, `strcat`, `strlen`*

Dându-se de la tastatură două șiruri de caractere, scrieți un program care inserează al doilea șir introdus în primul la o poziție specificată, ca în exemplul de mai jos.

*Exemplu:*

```
Primul sir:
Timpul este prea pretios, ca sa il pierdem cu lucruri de nimic.
Sirul de inserat:
 mult
Pozitia de inserare:
12
Sirul final este:
 Timpul este mult prea pretios, ca sa il pierdem cu lucruri de
nimic.
```

### *Analiza problemei și proiectarea soluției*

După citirea celor două siruri p și s, vom aloca memorie pentru o nouă variabilă de tip sir de caractere t, în care se va construi succesiv sirul cerut: mai întâi se copiează în t primele poz-1 caractere din c, după aceea se concatenează al doilea sir c la t, ca în final să adăugăm caracterele din primul sir de după poz în t:

```
t=(char*) malloc(strlen(c)+strlen(p)+1);
strncpy(t, p, poz-1);
t[poz-1]='\0'; /* indicator de sfarsit de sir!! */
strcat(t,c); /* concat. cu dep. rez. in t */
strcat(t,p+poz-1); /* concat. a doua parte a lui p */
t[strlen(c)+strlen(p)+1]='\0'; /* indic sfarsit de sir*/
```

La început se alocă fiecărui sir câte un număr de caractere, pentru a putea efectua corect operația de citire cu gets().

### *Program*

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
 char *p, *c, *t;
 int poz;
 p=(char*)malloc(25);
 c=(char*)malloc(10);
 if(!p){
 puts("Eroare de alocare!");
 exit(1);
 }
 if(!c){
 puts("Eroare de alocare!");
 exit(1);
 }
 puts("Primul sir: "); gets(p);
 puts("Sirul de inserat: "); gets(c);
 puts("Poziția de inserare: ");
 scanf("%d", &poz);
 if(!p){
 puts("Eroare de reallocare !");
 exit(1);
 }
```

```
t=(char*) malloc(strlen(c)+strlen(p)+1);
strncpy(t, p, poz-1);
t[poz-1]='\0';
strcat(t,c);
strcat(t,p+poz-1);
t[strlen(c)+strlen(p)+1]='\0';
printf("\nSirul final este: \n ");
puts(t);
getch();
}
```

### *Exerciții*

- Rezolvați problema neutilizând nici o funcție specială de prelucrare cu siruri, ci doar alocarea și parcurgerea secvențială.
- Rezolvați și problema inversă: dat fiind un sir, să se steargă din acesta toate aparițiile distincte ale unui alt sir dat. Să se scrie o funcție care înlocuiește toate aparițiile distincte ale unui sir dat cu un alt sir dat de utilizator.
- Date fiind un sir, poziția de început a unui subșir și lungimea acestuia din urmă, să se eliminate subșirul din sirul inițial.

### *Problema 10. Utilizarea funcției strtok()*

Scrieți un program pentru a demonstra utilizarea funcției strtok() din biblioteca string.h. În fișierul de intrare se află un text, iar în fișierul de ieșire vor trebui scrise toate cuvintele din acest text.

*Exemplu :*

| text.txt                                                       | text.out                                                                  |
|----------------------------------------------------------------|---------------------------------------------------------------------------|
| "La manie, nimic ? nu este<br>mai: potrivit; ! decat tacerea." | La<br>manie<br>nimic<br>nu<br>este<br>mai<br>potrivit<br>decat<br>tacerea |

### *Analiza problemei și proiectarea soluției*

Vom parcurge linie cu linie fișierul de intrare și vom procesa token-urile din fiecare linie relativ la sirul de delimitatori "\t\n,:?!,-"\\"!".

*Program*

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main(){
 FILE *fIn, *fOut;
 char line[80];
 char *delimitatori = " \t\n.:?!,-\\"\'";
 char *token;
 if((fIn = fopen("text.txt", "r")) == NULL){
 puts("Fisier inexistent!");
 return 0;
 }
 fOut = fopen("text.out", "w");
 while(!feof(fIn)){
 fgets(line, 80, fIn);
 token = strtok(line, delimitatori);
 while(token != NULL){
 fputs(token, fOut); fputs("\n", fOut);
 token = strtok(NULL, delimitatori);
 }
 }
 printf("Procesare terminata...");
 fcloseall();
 getch();
 return 0;
}
```

*Exercițiu*

Scrieți o funcție echivalentă cu `strtok` și utilizați-o în program.

**Problema II. Concatenarea a două fișiere**

Scrieți un program care concatenează conținutul a două fișiere cu numele introduse de la tastatură și scrie rezultatul în primul fișier.

*Analiza problemei și proiectarea soluției*

Scriem funcția `concatenare()` cu parametri numele fișierelor; primul se va deschide pentru adăugare, cel de al doilea – pentru citire. Caracterele din al doilea fișier se citesc secvențial și se adaugă la conținutul primului fișier.

*Program*

```
#include <stdio.h>

FILE *f1,*f2;
void concatenare(char s1[], char s2[]){
 char c;
 f1=fopen(s1,"at");
 f2=fopen(s2,"rt");
 if(f1==NULL){
 printf(" Fisier %s nu exista!", s1);
 return;
 }
 if(f2==NULL){
 printf(" Fisier %s nu exista!", s2);
 return;
 }
 while(f2 && !feof(f2)){
 c=fgetc(f2);
 fputc(c,f1);
 }
 if(f1) fclose(f1);
 if(f2) fclose(f2);
}

void main(){
 char nume1[15], nume2[15];
 printf("Numele primului fisier: ");
 scanf("%s",nume1);
 printf("Numele celui de-al doilea fisier: ");
 scanf("%s",nume2);
 concatenare(nume1, nume2);
}
```

*Exerciții*

1. Adăugați în primul fișier conținutul celui de-al doilea, linie cu linie.
2. Modificați programul astfel încât copiera să se facă linie cu linie. Determinați și numărul de linii al fișierului-sursă.
3. Să se creeze un fișier text obținut prin concatenarea a două fișiere, cu mențiunea că liniile din al doilea fișier care coincid cu cel puțin o linie din primul nu vor mai fi adăugate la fișierul-rezultat.

### Problema 12. Argumente în linia de comandă

Scrieți un program care copiează conținutul unui fișier-sursă într-un fișier-destinație ale căror nume sunt specificate în linia de comandă.

#### Analiza problemei și proiectarea soluției

Antetul funcției main() atunci când sunt necesare argumente în linia de comandă este:

```
int main(int argc, char **argv)
```

unde argc este de fapt un contor care specifică numărul de parametri plasați în linia de comandă, iar argv este un vector ce conține pointeri la siruri reprezentând parametrii respectivi, cu specificarea că argv[0] este numele programului. În cadrul programului de mai jos se va verifica mai întâi dacă în linia de comandă sunt furnizați toți parametrii necesari, după care se va parcurge fișierul de intrare, copiindu-se, linie cu linie, conținutul acestuia în fișierul de ieșire. Presupunând că numele programului de mai jos este pb12.c, după compilare, dacă se va executa în directorul unde se află executabilul comanda:

```
pb12 text.txt text.out
```

unde text.txt este un fișier existent, atunci se va crea fișierul text.out (dacă acesta nu există, în caz că există se va suprascrie), care va avea același conținut cu text.txt.

#### Program pb12.c

```
#include <stdio.h>

int main(int argc, char **argv){
 FILE *in, *out;
 int key;
 if(argc < 3){
 puts("Utilizare: pb12 sursa destinatie\n");
 puts("Sursa trebuie sa fie un fisier existent!");
 puts(
 "Daca destinatia exista atunci va fi suprascrisa!");
 return 0;
 }
 if((in = fopen(argv[1], "r")) == NULL){
 puts("Deschiderea fisierului sursa imposibila!");
 return 0;
 }
 if((out = fopen(argv[2], "w")) == NULL){
 puts("Deschiderea fisierului destinatie imposibila!");
 return 0;
 }
}
```

```
while (!feof(in)){
 key = fgetc(in);
 if(!feof(in)) fputc(key, out);
}
fclose(in); fclose(out);
return 0;
}
```

#### Exercițiu

Modificați programul de mai sus astfel încât numărul de argumente este variabil, iar primele fișiere date să fie concatenate și copiate în fișierul cu numele destinație:

```
pb12 fis1 fis2...fisn destinatie
```

#### Probleme și exerciții propuse

- Care este funcția pentru găsirea unui subșir într-un sir?
- Scrieți o metodă pentru adăugarea unui caracter dat c la sfârșitul unui sir de caractere s.
- Pentru ce este folosită funcția strtok()?
  - pentru detectarea unui subșir;
  - pentru inversarea literelor sirului;
  - pentru detectarea unuia dintre caracterele unui sir într-un alt sir.
- Decideți valoarea de adevăr pentru următoarele afirmații:
  - funcția strcmp() este *case insensitive*;
  - strcmp() compară ordinea lexicografică a două siruri s1 și s2 și returnează valoare pozitivă dacă s1 este mai mic decât s2;
  - funcția stricmp() compară două siruri și este *case insensitive*.
- Care dintre următoarele este un literal de tip sir de caractere, static?
  - 'Static String'
  - "Static String"
  - Static String
  - char string[101];
- Cu ce caracter dintre următoarele se termină toate sirurile de caractere?
  - '\0'
  - ' '
  - '.'
  - '\n'
- Care dintre următoarele instrucțiuni citește un sir cu numele str cu 100 de caractere?
  - fgets(str, 101, stdin);
  - fgets(str, 100, stdin);
  - readline(str, 100, '\n');
  - read(str);
- Care dintre următoarele funcții compară două siruri de caractere?
  - compare();
  - stringcompare();

- c) `cmp()`;  
d) `strcmp()`;
9. Care dintre următoarele funcții adaugă un sir la sfîrșitul altuia ?  
a) `append()`;  
b) `stringadd()`;  
c) `strcat()`;  
d) `stradd()`;
10. Scrieți un program demonstrativ pentru verificarea metodei `strup()`:  

```
char* strup(const char *strSursa)
```
11. În fișierul `cuvinte.in` se găsește un text care se încheie cu o linie goală. Să se scrie în fișierul `cuvinte.out` toate cuvintele textului în ordine alfabetică și toate cu litere mici, câte unul pe fiecare linie. Lungimea unui cuvânt este maximum de 20 de caractere, numărul de cuvinte în fișierul de intrare este maximum 1.000, cuvintele sunt despărțite prin unul sau mai multe spații.
12. În fișierul `persoane.in` se găsesc, separate de spații, mai multe nume de persoane scrise dezordonat, cu caractere mari sau mici. Presupunând că numele de fete se termină cu -a, să se creeze fișierul `persoane.out`, astfel încât numele de fete și de băieți să fie scrise ordonat, alfabetic, pe două coloane, ca în exemplul următor :

| <code>persoane.in</code> | <code>persoane.out</code> |
|--------------------------|---------------------------|
| IonuT Maria CoRNel IOAna | Ana Cornel                |
| Cristi IriNA ANa Sorin   | Ioana Cristi              |
|                          | Irina Ionut               |
|                          | Maria Sorin               |

13. Se citește de la tastatură un text format din litere, spații și semne de punctuație. Scrieți în fișierul `text.out`, aliniat la stânga și la dreapta, corespunzător unei lățimi efective a paginii, date de la tastatură (opțiunea *Justify* din Word !).
14. Se citește un sir de caractere de lungime maximă 50. Să se afișeze în formă de triunghi toate secvențele mediane ale cuvântului, pornind de la întregul cuvânt, până la secvența formată din caracterul (sau cele două caractere) din mijloc.
15. În fișierul `text.in` se găsește un text format din cuvinte, spații și semne de punctuație. De la tastatură se dă un număr natural k. Să se scrie în fișierul `text.out`, câte unul pe linie, toate cuvintele de lungime k din textul dat.
16. Mașina de scris a făcut greșeli și în locul unei litere date L1 a scris mereu litera L2. În fișierul `text.in` se găsește un text incorrect în acest sens. Literele L1 și L2 fiind date, ca și fișierul `text.in`, creați fișierul `text.out` corectat (litera L2 este înlocuită cu L1).
17. *Agenda telefonică*. Scrieți un program care execută operații simple corespunzătoare unei agende telefonice : citirea și salvarea datelor (nume, prenume, adresă, număr de telefon), regăsirea lor (căutarea numărului sau a adresei unei persoane etc.).
18. Să se scrie un program care determină numărul de apariții ale fiecărei litere într-un text. Textul se află în fișierul `text.in`, iar rezultatele se vor scrie în fișierul `aparitii.out`.
19. Să se afișeze toate prefixele și sufixele unui cuvânt dat de la tastatură, în ordine, câte unul pe linie. Se cer pentru fiecare cel puțin doi algoritmi distincți.

20. Creează un fișier text obținut prin concatenarea a n fișiere text, ale căror nume sunt date de la tastatură. Numerotează liniile fișierului rezultat.
21. Scrieți un program care afișează pe ecran, caracter cu caracter, făcând o pauză de 0,5 secundă după fiecare caracter, un fișier cu numele dat de la tastatură.
22. Se consideră un fișier text care conține numere pe mai multe rânduri, numerele de pe același rând fiind despărțite între ele prin spații. Să se determine câte numere sunt pe fiecare linie. Dar dacă numerele sunt despărțite prin virgule ?
23. Să se afișeze pe ecran toate caracterele unui fișier text dat, caracterele netipăribile (cu codul ASCII mai mic decât 32) fiind înlocuite de codul lor ASCII, precedat de caracterul „#”.
24. Să se scrie un program care verifică dacă două fișiere cu numele date de la tastatură au conținut identic.
25. Să se scrie un program care „șterge” eventualele linii vide sau „albe” (formate numai din spații sau caracter TAB) dintr-un fișier text cu numele dat de la tastatură.
26. *Lista de cuvinte*. Se consideră o secvență formată din n cuvinte. Să se construiască cu ele cel mai lung sir în care fiecare cuvânt folosit în sir are ultima literă mai mică în sens lexicografic decât prima literă din cuvântul următor. În cazul când există mai multe soluții, se va tipări doar una.
27. Fiind dat un sir de caractere, să se afișeze fiecare caracter folosit și numărul său de apariții.
28. Se consideră fișierul `text.txt` care conține un text și un cuvânt dat, cuvant. Scrieți funcții care execută următoarele operații :  
 a) ștergerea din fișier a tuturor aparițiilor cuvântului dat ;  
 b) înlocuirea tuturor aparițiilor cuvântului dat cu un altul.
29. Scrieți un program care să copieze dintr-un fișier într-un alt fișier liniile care nu conțin numere.
30. Scrieți un program care transformă toate literele mari dintr-un fișier dat în litere mici. Păstrați literele neschimbate pentru sirurile de caractere aflate între ghilimele sau apostrofuri.

## **Structuri, uniuni, câmpuri de biți**

**7 probleme complet rezolvate, 25 de exerciții și probleme propuse.**  
**Probleme rezolvate :**

- Puncte colineare
- Sumă a două fracții
- Reuniune de intervale
- Diferența, reuniunea și intersecția a două mulțimi
- Apartenența unui punct la disc
- Test union
- Câmpuri de biți

*Abia de pierdem felul din ochi, și ne-am și dublat eforturile.*

Mark Twain

### **Problema 1. Puncte colineare**

Scrieți un program care, citind de la tastatură coordonatele a trei puncte diferite în plan, decide dacă aceste puncte sunt colineare.

*Exemplul 1:*

Introduceti coordonatele celor 3 puncte:

Punctul 1:

x = 1

y = 2

Punctul 2:

x = 3

y = 4

Punctul 3:

x = 5

y = 6

Punctele sunt colineare !!!

*Exemplul 2:*

Introduceti coordonatele celor 3 puncte:

Punctul 1:

x = 3.45

y = 6.78

Punctul 2:

x = 2.34

y = 1.23

Punctul 3:

x = 6.75

y = 7.43

Punctele nu sunt colineare !!!

### **Analiza problemei și proiectarea soluției**

Considerăm punctele ca fiind  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$ ,  $P_3(x_3, y_3)$ .

O metodă de a decide dacă cele trei puncte sunt colineare este de a verifica dacă panta dreptei determinată de segmentul  $P_1P_2$  este egală cu panta dreptei determinată de segmentul  $P_2P_3$ . Panta dreptei  $P_1P_2$  este egală cu:

$$\frac{y_1 - y_2}{x_1 - x_2}$$

Pentru a aplica această metodă trebuie să verificăm în prealabil dacă  $x_1 = x_2$  sau  $x_2 = x_3$  (pentru a evita împărțirea la 0).

O altă metodă de rezolvare a problemei se poate baza pe următoarea observație. Cele trei puncte sunt colineare dacă și numai dacă:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0, \text{ echivalent cu a stabili dacă:}$$

$$x_1(y_2 - y_3) - y_1(x_2 - x_3) + (x_2y_3 - y_2x_3) = 0.$$

### Program

```
#include <stdio.h>

typedef struct{
 double x, y;
}TPunct;

int main(){
 TPunct P[3];
 int i;
 printf("Introduceti coordonatele celor 3 puncte: ");
 for(i=0; i<3; i++){
 printf("Punctul %d:\n x = ", i+1);
 scanf("%ld", &P[i].x);
 printf(" y = ");
 scanf("%ld", &P[i].y);
 }
 if(P[0].x == P[1].x || P[1].x == P[2].x)
 if(P[0].x == P[2].x)
 printf("Punctele sunt colineare !!!");
 else printf("Punctele nu sunt colineare !!!");
 else
 if((P[0].y-P[1].y)/(P[0].x-P[1].x)==(P[1].y-P[2].y)/(P[1].x-P[2].x))
 printf("Punctele sunt colineare !!!");
 else
 printf("Punctele nu sunt colineare !!!");
 }
}
```

### Exerciții

1. Scrieți un program care rezolvă problema anterioară bazându-se pe a două metodă.
2. Scrieți un program care decide dacă  $n$  puncte date de la tastatură sunt colineare.

### Problema 2. Suma a două fracții

Scrieți un program care calculează suma a două fracții. Cele două fracții de însumat se introduc de la tastatură și rezultatul se va scrie ca o operație de fracții ireductibile, ca în exemplele de mai jos. Dacă se introduce o valoare 0 pentru numitor, atunci se va cere o altă valoare până când aceasta va fi nenulă. Se va lucra cu numere naturale pozitive pentru număratori și numitori, din intervalul [1, 60.000].

| Exemplul 1:                    | Exemplul 2:                    |
|--------------------------------|--------------------------------|
| Introduceti cele două fracții: | Introduceti cele două fracții: |
| Fractia 1:                     | Fractia 1:                     |
| Număratorul: 2                 | Număratorul: 8                 |
| Numitorul: 3                   | Numitorul: 31                  |
| Fractia 2:                     | Fractia 2:                     |
| Număratorul: 9                 | Număratorul: 4                 |
| Numitorul: 18                  | Numitorul: 5                   |
| Rezultat:                      | Rezultat:                      |
| 2/3 + 1/2 = 7/6                | 8/31 + 4/5 = 164/155           |

### Analiza problemei și proiectarea soluției

Vom defini tipul TFracție ca fiind un tip structură ce conține două câmpuri: numărătorul și numitorul de tip întreg fără semn (unsigned). Definim, de asemenea, funcția cmmdc (unsigned, unsigned), care determină cel mai mare divizor al două numere date utilizând *algoritmul lui Euclid*. Funcția SimplificaFractie(TFracție) împarte numărătorul și numitorul fracției date ca parametru la cel mai mare divizor comun al acestora.

### Program

```
#include <stdio.h>

typedef struct{
 unsigned nr, num;
}TFracție;

unsigned cmmdc(unsigned a, unsigned b){
 unsigned aux;
 while(b){
 aux = a % b;
 a = b;
 b = aux;
 }
 return a;
}
```

```

int SimplificaFractie(TFractie *f){
 unsigned d;
 d = cmmdc(f->num, f->nr);
 f -> num /= d;
 f -> nr /= d;
 return 0;
}

int CitesteFractie(TFractie *f){
 printf("Numaratorul: ");
 scanf("%d", &f->nr);
 printf("Numitorul:");
 while(scanf("%d", &f->num) && !f->num)
 printf("\nNumitor zero!!! Introduceti altul: ");
 SimplificaFractie(f);
 return 0;
}

int Suma(TFractie f1, TFractie f2, TFractie *fs){
 fs->nr = f1.nr*f2.num + f2.nr*f1.num;
 fs->num = f1.num*f2.num;
 SimplificaFractie(fs);
 return 0;
}

int main(){
 TFractie f1, f2, fs;
 printf("Introduceti cele doua fractii: \nFractia 1: \n");
 CitesteFractie(&f1);
 printf("Fractia 2:\n");
 CitesteFractie(&f2);
 Suma(f1, f2, &fs);
 printf("Rezultat:\n %u/%u + %u/%u = %u/%u\n",
 f1.nr, f1.num, f2.nr, f2.num, fs.nr, fs.num);
 return 0;
}

```

### Exerciții

- Îmbunătățiți programul anterior astfel încât valorile introduse pentru numărător și numitor să poată fi și negative. Prelucrați datele în consecință.
- scrieți un program corespunzător celui de mai sus care determină produsul a două fracții, determinând rezultatul în formă ireductibilă.

### Exemplu:

Introduceti cele doua fractii:  
 Fractia 1:  
 Numaratorul: 8  
 Numitorul: 25  
 Fractia 2:  
 Numaratorul: 5  
 Numitorul: 2  
 Rezultat:  
 $8/25 \times 5/2 = 4/5$

### Problema 3. Reuniune de intervale

Se dă o mulțime de intervale definită prin reuniune de intervale închise:  $A = [a_0, b_0] \cup [a_1, b_1] \cup \dots \cup [a_{n-1}, b_{n-1}]$  unde  $n$  este un număr natural mai mic decât 101, iar  $a_0, b_0, \dots, a_{n-1}, b_{n-1}$  reprezintă capetele intervalelor închise. Scrieți un program care, pentru un număr real  $x$  dat de la tastatură, decide dacă aparține mulțimii  $A$ .

### Exemplu:

Numarul de intervale: 3  
 Intervalul 1: a = -5.67  
 b = 6.78  
 Intervalul 2: a = -6  
 b = -3.45  
 Intervalul 3: a = 1.23  
 b = 3.45  
 Valoarea x = -5.54  
 $-5.540$  se gaseste in reuniunea de intervale!!

### Analiza problemei și proiectarea soluției

După citire, se parcurge iterativ vectorul de intervale: cât timp valoarea nu este găsită și mai sunt intervale de verificat, continuăm cu următorul interval din tablou. Complexitatea este liniară  $O(n)$ .

### Program

```

#include <stdio.h>

typedef struct{
 float a, b;
}TInterval;

int main(){
 TInterval a[101];
 int n, i;

```

```

float x;
int ok = 0;
printf("Numarul de intervale: ");
scanf("%d", &n);
for(i=0; i<n; i++){
 printf("Intervalul %d: ", i+1);
 printf(" a = "); scanf("%f", &a[i].a);
 printf(" b = "); scanf("%f", &a[i].b);
}
printf("Valoarea x = ");
scanf("%f", &x);
for(i=0; !ok && i<n; i++)
 if(a[i].a <= x && x <=a[i].b) ok = 1;
if(ok)
 printf("%.3f se gaseste in reuniunea de intervale!!", x);
else
 printf("%.3f nu se gaseste in reuniunea de intervale!!", x);
return 0;
}

```

### Exercițiu

Presupunem că mulțimea  $A = [a_0, b_0] \cap [a_1, b_1] \cap \dots \cap [a_{n-1}, b_{n-1}]$ . Această mulțime poate fi: mulțimea vidă, un punct sau un interval. Scrieți un program care determină mulțimea  $A$ , scrisă sub una dintre cele trei forme posibile.

### Problema 4. Diferența, reuniunea și intersecția a două mulțimi

Să se scrie un program care afișează diferența, intersecția și reuniunea a două mulțimi de numere naturale cu valori cuprinse între 0 și 50 inclusiv. Numărul de elemente al fiecărei mulțimi și elementele sale sunt preluate de către utilizator, forma intrării și ieșirii fiind ca în exemplul următor. Presupunem că datele de intrare sunt introduse corect.

*Exemplu :*

Multimea A:

Numarul de elemente ale multimii: 4

Elementele: 2 5 45 49

Multimea B:

Numarul de elemente ale multimii: 5

Elementele: 1 3 5 45 46

Mulțimi introduse:

A = {2, 5, 45, 49}

B = {1, 3, 5, 45, 46}

----

### REZULTATE

----

Intersectia: {5, 45}

Diferenta: {2, 49}

Reuniunea: {1, 3, 5, 45, 46, 2, 49}

### Analiza problemei și proiectarea soluției

Întrucât o mulțime este o entitate complet determinată de numărul său de elemente (cardinalul mulțimii) și de elementele acesteia, vom considera tipul struct TMultime, care conține două variabile: n, reprezentând cardinalul, și un tablou unidimensional m[], reprezentând elementele mulțimii.

Date fiind două mulțimi A și B:

- intersecția lor este mulțimea cu elemente care sunt conținute de ambele mulțimi date;
- diferența lor (A-B) este mulțimea cu elemente care sunt conținute de A și nu sunt conținute de B;
- reuniunea lor este mulțimea cu toate elementele din A și B luate, bineînțeles, câte o singură dată.

În scopul rezolvării problemei vom scrie o funcție cu antetul:

```
short apartine(TMultime A, short elem)
```

care returnează 1 dacă elementul elem aparține mulțimii A și 0 în caz contrar.

Pentru citirea, respectiv scrierea unei mulțimi vom scrie două funcții cu anteturile:

```
void citesteMultime(TMultime *M)
```

```
void afiseazaMultime(struct TMultime M)
```

Tinând cont de definiția celor trei noțiuni: intersecție, diferență și reuniune, pentru a le determina, scriem următorul ciclu for(), precedat de instanțierile corespunzătoare pentru mulțimile-rezultat:

```
I.n = 0; D.n = 0; R = B;
for(i=0; i<A.n; i++)
 if(apartine(B, A.m[i])) I.m[I.n++] = A.m[i];
 else{D.m[D.n++] = A.m[i]; R.m[R.n++] = A.m[i];}
```

### Program

```
#include <stdio.h>
```

```
struct TMultime{
```

```
 short n;
```

```
 short m[51];
```

```
};
```

```

short apartine(struct TMultime A, short elem){
 short i, este = 0;
 for(i=0; !este && i<A.n; i++)
 if(elem == A.m[i]) este=1;
 return este;
}

void citesteMultime(struct TMultime *M){
 short i;
 printf("Numarul de elemente al multimii: ");
 scanf("%d", &M->n);
 printf("Elementele: ");
 for(i=0; i<M->n; i++) scanf("%d", &M->m[i]);
}

void afiseazaMultime(struct TMultime M){
 short i;
 printf(" ");
 for(i=0; i<M.n-1; i++) printf("%d, ", M.m[i]);
 if(M.n>0) printf("%d", M.m[M.n-1]);
 printf("\n");
}

void main(){
 short i;
 struct TMultime A, B, I, R, D;
 printf("Multimea A:\n"); citesteMultime(&A);
 printf("Multimea B:\n"); citesteMultime(&B);
 I.n = 0; D.n = -0; R.n = B.n;
 for(i=0; i<A.n; i++)
 if(apartine(B, A.m[i]))(I.m[I.n++] = A.m[i]);
 else{ D.m[D.n++] = A.m[i]; R.m[R.n++] = A.m[i]; }
 printf("\nMultimi introduse:\n");
 printf("A = "); afiseazaMultime(A);
 printf("B = "); afiseazaMultime(B);
 printf("----\nREZULTATE\n----\n");
 printf("Intersectia: "); afiseazaMultime(I);
 printf("Diferenta: "); afiseazaMultime(D);
 printf("Reuniunea: "); afiseazaMultime(R);
}

```

**Exerciții**

- Rescrieți programul anterior fără a utiliza tipul structură TMultime, ci doar tipuri de bază.

- Îmbunătățiți programul anterior astfel încât la citire să se verifice proprietatea de mulțime a datelor introduse (elementele nu trebuie să se repete) și la afișare elementele mulțimii să apară în ordine crescătoare.

**Problema 5. Apartenența unui punct la disc**

Se citesc la intrare n cercuri, date fiecare prin coordonatele centrului și rază, ca numere reale ce se încadrează în tipul standard float, și un punct în plan. Decideți căruia dintre discurile cercurilor îi aparține punctul dat.

**Exemplu:**

Numarul de cercuri: 3  
Introduceti cercurile:

Cercul 1:  
Centrul: 1.23 3.4  
Raza: 3.67

Cercul 2:  
Centrul: 0 1.2  
Raza: 2.34

Cercul 3:  
Centrul: -1.23 8.9  
Raza: 2.34  
Introduceti punctul: 1.56 2.34

**REZULTAT**

-----  
Punctul (1.560, 2.340) aparține cercurilor:  
1 -> C( 1.230, 3.400, 3.670 )  
2 -> C( 0.000, 1.200, 2.340 )

**Analiza problemei și proiectarea soluției**

Vom defini tipurile structură TPunct și TCerc, care reprezintă entitățile *punct*, respectiv *cerc* corespunzătoare.

Implementăm funcția distanta(), care calculează distanța dintre două puncte, cu antetul:

```
float distanta(TPunct P1, TPunct P2)
```

Observăm în program că pentru a determina pătratul unui număr definim macroul sqr().

Pentru a determina dacă un punct aparține unui cerc scriem funcția cu antetul :

```
short apartineCerc(TCerc C, TPunct P)
```

care returnează 1 dacă punctul dat P aparține cercului C.

### Program

```
#include <stdio.h>
#include <math.h>

#define sqr(a) ((a)*(a))

typedef struct{
 float x, y;
}TPunct;

typedef struct{
 TPunct O;
 float R;
}TCerc;

float distanta(TPunct P1, TPunct P2){
 return
 (float)sqrt(sqr(P1.x-P2.x) + sqr(P1.y-P2.y));
}

short apartineCerc(TCerc C, TPunct P){
 if(distanta(C.O, P) < C.R) return 1;
 else return 0;
}

void main(){
 int i, n;
 TCerc C[50];
 TPunct P;
 printf("Numarul de cercuri: ");
 scanf("%d", &n);
 printf("\nIntroduceti cercurile:\n");
 for(i=0; i<n; i++){
 printf("\n Cercul %d: \n", i+1);
 printf(" Centrul: ");
 scanf("%f %f", &C[i].O.x, &C[i].O.y);
 printf(" Raza: ");
 scanf("%f", &C[i].R);
 }
}
```

```
printf("Introduceti punctul: ");
scanf("%f %f", &P.x, &P.y);
printf("\n----\nREZULTAT\n----\n");
printf("Punctul (%3.3f, %3.3f) aparține discurilor:\n",
 P.x, P.y);
for(i=0; i<n; i++)
 if(apartineCerc(C[i], P))
 printf("d -> D(%3.3f, %3.3f, %3.3f)\n",
 i+1, C[i].O.x, C[i].O.y, C[i].R);
}
```

### Exercițiu

Scrieți un program care rezolvă problema modificată : dându-se o mulțime de cercuri și o mulțime de puncte, să se determine punctele care aparțin discurilor celor mai mari număr de cercuri.

### Problema 6. Test union

Pe lângă cerința de a grupa componente de tipuri diferite într-o singură entitate, limbajul C poate satisface cerința suplimentară ca aceste componente să ocupe aceeași zonă de memorie. În acest scop se folosește **tipul uniune (union)**.

Presupunând că se fac măsurători ale unor obiecte și pentru fiecare obiect numai una este necesară, să se citească și afișeze aceste valori măsurate, folosind tipul **union**. Valoarea măsurată pentru un obiect poate fi de tip **int**, **double** sau **char**. *Exemplu:*

```
Cate masuratori se fac? 3
Introduceti masuratorile:
Tip de masuratoare(integer/1, double/2, char/3):3
CHR value = D
Tip de masuratoare(integer/1, double/2, char/3):1
INT value = -12
Tip de masuratoare(integer/1, double/2, char/3):2
DBL value = 23.45
Masuratorile introduse:
CHR D
INT -12
DBL 23.450000
```

### Analiza problemei și proiectarea soluției

Pentru a remarcă tipul de măsurătoare folosim tipul **enum MesType**, iar pentru măsurătoarea propriu-zisă folosim tipul **struct MesValue**. O întreagă măsurătoare este complet definită de tipul ei și de valoarea măsurătorii; aceasta este **Measure**.

*Program*

```

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

enum MesType{
 UNKNOWNType,
 INTEGERType,
 DOUBLEType,
 CHARType
};

union MesValue{
 int intValue;
 double dblVal;
 char chrVal;
};

typedef struct MeasureStruct{
 enum MesType type;
 union MesValue value;
} Measure;

void printMeasure(Measure *pMeasure){
 if(!pMeasure){
 printf("Masuratoare goala");
 }
 switch (pMeasure->type){
 case INTEGERType:
 printf("INT %d\n", pMeasure->value.intValue);
 break;
 case DOUBLEType:
 printf("DBL %lf\n", pMeasure->value.dblVal);
 break;
 case CHARType:
 printf("CHR %c\n", pMeasure->value.chrVal);
 break;
 default:
 printf("UNKNOWN MEASURE TYPE\n");
 }
}

```

```

int main(){
 int n, tip, i;
 Measure aM[10];
 printf("Cate masuratori se fac? ");
 scanf("%d", &n);
 printf("Introduceti masuratorile:\n");
 for(i=0; i<n; i++){
 printf("Tip de masuratoare(integer/1, double/2, char/3):");
 scanf("%d", &tip);
 aM[i].type = tip;
 switch(tip){
 case INTEGERType:
 printf("INT value = ");
 scanf("%d", &aM[i].value.intValue);
 break;
 case DOUBLEType:
 printf("DBL value = ");
 scanf("%lf", &aM[i].value.dblVal);
 break;
 case CHARType:
 printf("CHR value = ");
 aM[i].value.chrVal = getche();
 putch('\n');
 break;
 default:
 printf("UNKNOWN MEASURE TYPE\n");
 }
 }
 printf("Masuratorile introduse:\n");
 for(i=0; i<n; i++)
 printMeasure(&aM[i]);
 return 0;
}

```

*Exerciții*

1. Căutați în *Help* uniuni definite în ANSI C.
2. O mulțime de persoane din cadrul unui spital trebuie înregistrată. Persoanele pot fi angajați sau pacienți, iar datele de salvat sunt numele, prenumele, data nașterii, sexul. Scrieți un program care utilizează uniuni pentru stocarea datelor de acest tip.

### Problema 7. Câmpuri de biți

Pot fi folosite numai în interiorul datelor de tip structură sau uniune și sunt utile pentru minimizarea consumului de memorie folosind direct biții și nu octetul. Câmpurile de biți își găsesc aplicații în criptare, arhivare, controlul dispozitivelor fizice etc. Reguli :

- tipul câmpurilor de biți poate fi orice tip întreg cu sau fără semn, mai puțin long, chiar și enumerare;
- numărul de biți poate avea valori întregi cuprinse între 1 și 16;
- nu se pot folosi tablouri de câmpuri de biți;
- nu se pot folosi operatorii de adesare, nici operatorul sizeof;
- numele de identificare al câmpului poate lipsi.

Presupunem că vrem să reținem într-un mod compact data calendaristică: zi în săptămână (0..6), zi în lună, luna și anul (0..99). Scrieți un program simplu, folosind câmpuri de biți, care să stocheze o dată calendaristică.

*Exemplu :*

```
Introduceti data:
Zi sapt:2
Zi luna:6
Luna:11
An:74
Data introdusa:
Miercuri, 6/11/74
```

### Analiza problemei și proiectarea soluției

Considerăm tipul struct Data care conține câmpurile de biți corespunzătoare.

### Program.

```
#include <stdio.h>

typedef struct SData {
 unsigned short nZiSapt : 3; /* 0..7 (3 biți) */
 unsigned short nZiLuna : 6; /* 0..31 (6 biți) */
 unsigned short nLuna : 5; /* 0..12 (5 biți) */
 unsigned short nAn : 8; /* 0..100 (8 biți) */
} Data;

char Zile[7][10] = {"Luni", "Martii", "Miercuri",
 "Joi", "Vineri", "Sambata",
 "Duminica"};
```

```
void main() {
 Data d;
 int i;
 printf("Introduceti data:\n");
 printf("Zi sapt:"); scanf("%d", &i); d.nZiSapt = i;
 printf("Zi luna:"); scanf("%d", &i); d.nZiLuna = i;
 printf("Luna:"); scanf("%d", &i); d.nLuna = i;
 printf("An:"); scanf("%d", &i); d.nAn = i;
 printf("Data introdusa: \n");
 printf("%s, %d/%d/%d", Zile[d.nZiSapt],
 d.nZiLuna, d.nLuna, d.nAn);
}
```

### Exercițiu

Modificați programul astfel încât anul să fie de patru cifre.

### Probleme propuse

1. Se citesc de la tastatură data nașterii unei persoane și data curentă. Să se determine vîrstă în ani împliniți a persoanei.
2. Se citesc de la tastatură n perechi de numere reale reprezentând puncte în plan (abscisă și ordinată). Scrieți un program care să determine lungimea celui mai mare segment având ca extremități două dintre punctele date.
3. Se dau n orașe ( $2 \leq n \leq 20$ ) împreună cu coordonatele lor (latitudine, longitudine). Fiind precizat unul dintre orașe, afișați orașul cel mai apropiat de acesta.
4. Se dau numele a n elevi și notele obținute de aceștia la proiectul de informatică. Să se calculeze media notelor și să se scrie elevii în fișierul info.out, astfel încât elevii să fie numerotați și în ordinea descrescătoare a notelor. Cei care au nota mai mare decât media se vor scrie cu literă mari.
5. Se dă o listă conținând numele unor domnitori din mai multe țări, țara, anii de domnie și numele soților. Să se afișeze :
  - a) domnitorii dintr-o anumită perioadă de timp, introdusă de la tastatură;
  - b) lista pe țări a domnitorilor, în ordinea cronologică a perioadelor de domnie.
6. Caracteristicile unui produs sunt date de prețul în lei, data intrării în magazin și data expirării. Știind că prețul scade progresiv cu aceeași sumă, astfel încât în ziua imediat următoare datei de expirare acesta să devină 0, să se calculeze prețul la o anumită dată sau să se semnaleze că produsul este expirat.
7. Scrieți un program pentru lucrul cu numere complexe (citire, scriere, înmulțire, adunare, modul etc.).
8. Proiectați o structură de date pentru reprezentarea poligoanelor convexe în plan, cu coordonatele vîrfurilor numere reale, și scrieți o funcție care determină aria unui astfel de poligon.
9. Proiectați structuri de date pentru reprezentarea unui punct, triunghi, dreptunghi, cerc. Scrieți căte o metodă pentru determinarea ariei fiecărui tip de figură, aria unui punct fiind 0. Folosind union, definiți un tip general de date care să reprezinte oricare dintre figurile geometrice definite anterior.

10. Polinoamele rare cu coeficienți întregi sunt polinoame de grade mari cu mulți coeficienți nuli și pot fi reprezentate printr-o structură definită astfel:

```
typedef struct
{
 int Coef;
 unsigned int Exponent;
} TMonom;
typedef TMonom TPolinom[50];
```

Scrieți proceduri de scriere, citire, adunare, înmulțire a polinoamelor rare reprezentate astfel.

11. Se consideră declarația

```
struct Complex * point_struct;
```

Găsiți semnificațiile expresiilor: `(++point_struct)->membru, (point_struct++)->membru, ++(point_struct->membru), (point_struct->membru)++.`

12. Să se definească o funcție de citire și una de scriere pentru citirea și afișarea timpului. Câmpurile componente sunt: ora, minutul, secunda. Să se implementeze apoi o funcție pentru calculul diferenței, exprimate în secunde, între doi timpi. Citirea și scrierea datelor se vor face sub forma „hh: mm: ss”.

13. Scrieți un program demonstrativ ce utilizează o structură cât mai compactă care conține informații despre echipamentele instalate la un anumit moment într-un calculator:

- numărul de porturi paralele: 2 biți;
- imprimantă serială: 1 bit;
- adaptor de jocuri: 1 bit;
- număr de proturi seriale RS232: 3 biți;
- DMA present/absent: 1 bit;
- număr de unități de dischetă: 2 biți;
- dimensiunea memoriei RAM dă pe placă de bază: 2 biți;
- prezența unui coprocesor: 1 bit;
- verificarea prezenței a cel puțin unei unități de disc: 1 bit.

14. Fișierul complexe.in conține mai multe linii, pe fiecare linie aflându-se câte o pereche de numere reale reprezentând partea reală, respectiv partea imaginară a unui număr complex. Scrieți un program care: citește numerele într-un tablou cu numere complexe, parcurge tabloul cu ajutorul unui pointer pentru prelucrare, calculează perimetrul poligonului care are drept vîrfuri în plan reprezentările celor n numere complexe (presupunem că ele sunt, în ordine, vîrfurile unui poligon), suma și produsul numerelor complexe date.

## Operații pe biți

6 probleme complet rezolvate, 14 exerciții și probleme propuse.  
Probleme rezolvate:

- Reprezentare binară
- Operații elementare pe biți
- Împachetarea datei
- Diferite operații folosind biți
- Numărul biților 1 în reprezentarea în baza 2
- Ciurul lui Eratostene pe biți

*Nu am vreo însușire specială, ci sunt numai un pasionat curios.*

Albert Einstein

### ***Problema 1. Reprezentare binară***

Scriți un program care să afișeze valoarea în binar a unui număr întreg de tipul int introdus de la tastatură, utilizând operațiile pe biți.

*Exemplu varianta 1 :*

```
Introduceti numarul: 32123
32123 in binar este: 0000000000000000111110101111011
```

*Exemplu varianta 2 :*

```
Introduceti numarul: 32123
32123 in binar este: 111110101111011
```

#### ***Analiza problemei și proiectarea soluției***

Întrucât pe sisteme diferite dimensiunea tipului int diferă, vom defini constanta SIZ\_INT care reprezintă numărul de biți corespunzători unui număr întreg.

Pentru a găsi reprezentarea în binar a numărului dat vom proceda astfel : deplasăm toți biții, începând de la dreapta spre stânga, pe poziția 0 (prima din dreapta reprezentării binare, cea mai puțin reprezentativă) și efectuăm operația AND (&) cu masca 1 ; dacă rezultatul este 0, atunci bitul este 0, altfel este 1.

Vom prezenta mai jos două variante de program : în prima variantă rezultatul conține toții biții numărului dat, chiar dacă sunt nesemnificativi. În cadrul celei de a doua variante, zerourile nesemnificative sunt eliminate, diferența fiind introducerea unei variabile flag de tip short, care devine 1 în momentul când este găsit primul bit 1 de la stânga la dreapta.

#### ***Program varianta 1***

```
#include <stdio.h>

#define SIZ_INT (sizeof(int)*8)

void main(){
 int i, n;
 printf("Introduceti numarul: ");
 scanf("%d", &n);
 printf("%d in binar este: ", n);
```

```

 for(i=SIZ_INT-1; i>=0; i--)
 printf("%d", (n>>i)&1);
}

```

### Program varianta 2

```

#include <stdio.h>

#define SIZ_INT (sizeof(int)*8)

void main(){
 int i, n;
 short flag = 0;
 printf("Introduceti numarul: ");
 scanf("%d", &n);
 printf("%d in binar este: ", n);
 for(i=SIZ_INT-1; i>=0; i--){
 if(!flag && (n>>i)&1==1) flag=1;
 if(flag) printf("%d", (n>>i)&1);
 }
}

```

### Exercițiu

Determinați reprezentarea în baza 2 a unui număr natural dat fără a folosi operațiile pe biți, ci împărțind succesiv la 2 și reținând resturile într-un tablou unidimensional.

### Problema 2. Operații elementare pe biți

scrieți un program care să afișeze un tabel corespunzător operațiilor pe biți de negare (~), AND (&), OR () și XOR (^),

*Exemplu :*

| p   q   ~p   p&q   p q   p^q |
|------------------------------|
| 0   0   1   0   0   0        |
| 0   1   1   0   1   1        |
| 1   0   0   0   1   1        |
| 1   1   0   1   1   0        |

### Analiza problemei și proiectarea soluției

Vom folosi operațiile pe biți, utilizând un format corespunzător.

### Program

```

#include <stdio.h>

void main(){
 int p, q;
 printf("-----\n");
 printf(" | p | q | ~p | p&q | p|q | p^q | \n");
 printf("-----\n");
 for(p=0; p<2; p++)
 for(q=0; q<2; q++)
 printf(" | %d | %d | %d | %d | %d | %d | \n",
 p, q, ~p&1, p&q, p|q, p^q);
 printf("-----\n");
}

```

### Exercițiu

Pentru valorile binare  $p, q, r \in \{0, 1\}$ , scrieți un program care afișează pe ecran toate valorile posibile  $p\&q\&r, \sim p + \sim q + \sim r, (p-q) + r$  ca în tabelul de mai sus.

### Problema 3. Împachetarea datei

Zilele secolului XX pot fi notate folosind numere întregi în forma zi.luna.an. Tînând cont de faptul că pentru zile (1..31) sunt suficienți 5 biți, pentru lună (1..12) sunt suficienți 4 biți și pentru ani (0..99) sunt suficienți 7 biți, scrieți o funcție de împachetare și o funcție de despachetare a datei prin intermediul tipului unsigned în cadrul unui program care să furnizeze datele de ieșire ca în exemplul următor. De asemenea, se va afișa și dimensiunea tipului de date unsigned.

*Exemplu :*

```

sizeof(unsigned) = 4
zi = 18
luna = 3
an = 80

Data codificata: 37328
Data despachetata: 18.3.80

```

### Analiza problemei și proiectarea soluției

Tînând cont de faptul că tipul unsigned poate avea număr diferit de octeți pe sisteme de operare diferite, vom rezolva problema independent de sistemul de operare utilizat.

Dacă notăm cu  $b$  numărul de biți din reprezentarea unui număr `unsigned`, atunci forma numărului este:

$$a_{b-1}a_{b-2}\dots a_2a_1a_0$$

unde  $a_i$  sunt biții din reprezentarea în binar a numărului.

Pentru a compacta ziua, luna și anul într-un număr de tip `unsigned` efectuăm următorii pași:

1. inițializăm cu 0 o variabilă de tip `unsigned`, data;
2. deplasăm zi cu 11 ( $biți(luna+an)$ ) biți la stânga și o adăugăm datei, ceea ce înseamnă că vom completa biții  $a_{15}a_{14}a_{13}a_{12}a_{11}$  cu valoarea zilei, restul biților fiind 0;
3. deplasăm luna cu 7 biți la stânga și o adăugăm datei, ceea ce înseamnă că vom completa în variabila data și biții  $a_{10}a_9a_8a_7$ ;
4. adunând an la variabila data vom completa și biții  $a_6\dots a_0$ .

Funcția `unpackData()` execută operațiile inverse.

#### Program

```
#include <stdio.h>

int packData(unsigned zi, unsigned luna, unsigned an){
 unsigned data=0;
 data = zi<<11;
 data += luna<<7;
 data += an;
 return data;
}

void unpackData(unsigned data, unsigned *zi,
 unsigned *luna, unsigned *an){
 int b = sizeof(unsigned)*8;
 *zi = data>>11;
 *luna = data<<(b-11)>>(b-4);
 *an = data<<(b-7)>>(b-7);
}

void main(){
 unsigned data, z, l, a;
 printf("\nsizeof(unsigned) = %d\n", sizeof(unsigned));
 printf("zi = ");scanf("%u",&z);
 printf("luna = ");scanf("%u",&l);
 printf("an = ");scanf("%u",&a);
 data = packData(z, l, a);
 printf("\n*****\n");
 printf("Data codificata: %u", data);
}
```

```
unpackData(data, &z, &l, &a);
printf("\nData despachetata: %u.%u.%u\n", z,l,a);
}
```

#### Exerciții

1. Dezvoltați programul pentru a vedea descompunerea în binar a datei împachetate.
2. Dezvoltați programul astfel încât să vizualizați în binar, pe rând, valorile cu care se operează în funcții (`packData()`:  $data>>11$ ,  $luna<<7$ ,  $data$ ,  $an$ ,  $data$ ; `unpackData()`:  $data>>11$ ,  $data<<(b-11)$ ,  $data<<(b-4)>>(b-4)$ ,  $data<<(b-7)>>(b-7)$ ).

#### Problema 4. Diferite operații folosind biți

*Utilizarea operatorilor pe biți este deosebit de avantajoasă datorită vitezei mari de execuțare a operațiilor corespunzătoare lor.*

Folosind operatorii pe biți, scrieți funcții C care:

- a) pentru un  $n$  dat, testează dacă un număr natural dat este divizibil cu  $2^n$ ;
- b) testează dacă un număr dat este pozitiv sau negativ;
- c) pentru un număr natural  $m$  dat calculează multiplul acestuia de  $2^n$ ;
- d) pentru un număr natural  $m$  dat, calculează  $\lfloor m/2^n \rfloor$ ;
- e) folosind descompunerea pe biți a lui  $n$ , calculează  $m^n$ .

Datele de intrare și de ieșire pot avea formatul din exemplul următor.

*Exemplu :*

```
m = -1024
n = 3
```

```

a. 2^3 divide -1024
b. -1024 este negativ!
c. -1024 * 2^3 = -8192
d. -1024 / 2^3 = -128
```

```

e. Introduceti cele doua numere:
```

```
m = 3
n = 5
```

```
e. 3^5 = 243
```

#### Analiza problemei și proiectarea soluției

Fie reprezentarea binară a lui  $m$  astfel:

$$a_k a_{k-1} a_{k-2} a_{k-3} \dots a_1 a_0$$

Atunci,  $m$  se divide cu  $2^n$  dacă și numai dacă  $a_0 = a_1 = \dots = a_{n-1} = 0$ .  
Cum  $a_i = (m>>i) \& 1$ , în cadrul funcției cu antetul:

```
long doi_n_divide(long m, int n)
```

vom scrie un ciclu for() pentru a verifica egalitatea multiplă cu 0 de mai sus.

Numărul  $m$  este pozitiv dacă și numai dacă, în reprezentarea completă a sa pe sizeof(m) biți, cel mai semnificativ bit,  $a_k$ , este 0. Acest bit este:

```
(m >> d) & 1
```

unde  $d = \text{sizeof(long)} * 8 - 1$ .

Pentru  $m$  multiplu, respectiv cât de  $2^n$ , se utilizează operatorii de deplasare la stânga, respectiv la dreapta lui  $m$ , ca în funcțiile cu anteturile:

```
long m_ori_doi_la_n(long m, int n)
long m_supra_doi_la_n(long m, int n).
```

Tinând cont de formula:

$$m^n = m^{a_0+a_1 \cdot 2 + a_2 \cdot 2^2 + \dots + a_k \cdot 2^k} = m^{a_0} \cdot (m^{a_1})^2 \cdot (m^{a_2})^{2^2} \cdot \dots \cdot (m^{a_k})^{2^k}$$

unde  $a_k a_{k-1} \dots a_1 a_0$  este descompunerea lui  $n$  în binar, rezultă că  $m^{a_1}$  este 1 sau  $m$ .

Pentru a calcula  $m^n$  scriem funcția cu antetul:

```
long m_la_n(long m, int n)
```

bazându-ne pe aceste observații.

### Program

```
#include <stdio.h>
#include <math.h>

long doi_n_divide(long m, int n){
 int i;
 int ok=1;
 for(i=0; ok && i<n; i++)
 if((m>>i)&1) ok=0;
 return ok;
}

long pozitiv(long m){
 int d = sizeof(long)*8-1;
 return
 ((m>>d) & 1)==0;
}

long m_ori_doi_la_n(long m, int n){
 return m<<n;
}
```

```
long m_supra_doi_la_n(long m, int n){
 return m>>n;
}

long m_la_n(long m, int n){
 long i, pM = m;
 long rez = 1;
 for(i=0; i<=log(n)/log(2)+1.0; i++){
 if((n>>i) & 1) rez *= pM;
 pM *= pM;
 }
 return rez;
}

void main(){
 long m;
 int n;
 printf(" m = "); scanf("%ld", &m);
 printf(" n = "); scanf("%ld", &n);
 printf("\n*****");
 if(doi_n_divide(m,n))
 printf("\n a. 2^%ld divide %ld ", n, m);
 else printf("\n a. 2^%ld nu divide %ld ", n, m);
 if(pozitiv(m))
 printf("\n b. %ld este pozitiv!", m);
 else printf("\n b. %ld este negativ!", m);
 printf("\n c. %ld * 2^%ld = %ld",
 m, n, m_ori_doi_la_n(m, n));
 printf("\n d. %ld / 2^%ld = %ld",
 m, n, m_supra_doi_la_n(m, n));
 printf("\n\n*****");
 printf("\ne. Introduceti cele doua numere: ");
 printf("\n m = "); scanf("%ld", &m);
 printf(" n = "); scanf("%ld", &n);
 printf("\n e. %ld^%ld = %ld ", m, n, m_la_n(m, n));
}
```

### Exercițiu

Folosind descompunerea pe biți a lui  $m$ , respectiv  $n$ , calculați  $m \cdot n$ .

### Problema 5. Numărul bițiilor 1 în reprezentarea în baza 2

Scrieți o funcție care numără câți biți nenuli sunt în reprezentarea în baza 2 a unui număr între 0 și 255.

*Exemplu:*

Introduceti un numar (0 - 255 zecimal): 159  
Numar de biti 1 in 159 (baza 2) = 6

#### Analiza problemei și proiectarea soluției

Vom scrie funcția `nrBiti()`, care va număra numărul de biți 1 din reprezentarea în baza 2 a parametrului utilizând deplasarea la dreapta a reprezentării numărului (operatorul `>>`) și aplicarea măștii 1 cu operatorul AND (`&1`). Rezultatul `x&1` va fi bitul cel mai puțin semnificativ (din dreapta) al lui `x`. Vom incrementa contorul în cazul când acest bit are valoarea 1.

#### Program

```
#include <stdio.h>
#include <conio.h>

unsigned char nrBiti(unsigned char); /*antetul functiei*/

void main(){
 unsigned char i8, count;
 int i;
 printf("Introduceti un numar (0 - 255 zecimal): ");
 scanf("%d", &i);
 if((i < 0) || (i > 255)){
 printf("Eroare !(numar in afara intervalului) = %d\n", i);
 getch();
 return;
 }
 i8 = (unsigned char) i;
 count = nrBiti(i8);
 printf("\nNumar de biti 1 in %d (baza 2) = %d\n", i,
 count);
 getch();
}

unsigned char nrBiti(unsigned char x){
 unsigned char count;
 for(count = 0; x!=0; x>>=1)
 if(x&01)
 ++count;
 return count;
}
```

### Exercițiu

Dacă fiind un număr întreg `n`, scrieți o metodă pentru determinarea numărului `m` care are reprezentarea binară ca și `n`, în care fiecare bit este inversat.

### Problema 6. Ciurul lui Eratostene pe biți

Să se scrie un program care marchează numerele prime utilizând metoda *ciurului lui Eratostene*, într-un mod compact, pentru fiecare număr folosindu-se doar un bit. În fișierul `numere.in` se găsesc mai multe numere naturale mai mici decât 30.000. Creați fișierul `prime.out` care să conțină un raport privind primalitatea numerelor date, ca în exemplul:

| numere.in             | prime.out       |
|-----------------------|-----------------|
| 12 13 7 177 337 12321 | 12 -- COMPUS    |
| 51 541 107            | 13 -- PRIM      |
|                       | 7 -- PRIM       |
|                       | 177 -- COMPUS   |
|                       | 337 -- PRIM     |
|                       | 12321 -- COMPUS |
|                       | 51 -- COMPUS    |
|                       | 541 -- PRIM     |
|                       | 107 -- PRIM     |

#### Analiza problemei și proiectarea soluției

Vom utiliza o mapare a numerelor naturale ca vector de elemente de tip `char`. Pentru că dimensiunea tipului `char` este de 1 octet (8 biți), deducem că putem încapsula într-un astfel de element informațiile necesare pentru 8 numere naturale. Folosim tabloul `sita[]` ce conține elemente de tip `char`. Pentru un element dat `n`, acesta se găsește în compozitia cu numărul `n/8` a vectorului, iar în cadrul acestei componente este bitul al `n%8`-lea. Vom scrie funcția `isOne()`, care are ca parametri un număr întreg `n` și poziția unui bit în cadrul acestui număr. Această funcție returnează bitul de la poziția `poz`, prin deplasarea la dreapta a lui `n` cu pozitii și aplicarea operatorului AND relativ la 1. Pentru toate numerele prime de la stânga spre dreapta î vom marca cu 1 toți biții corespunzători multiplilor posibili ai acestora  $i \cdot j$ . Pentru a marca cu 1 aceste poziții  $i \cdot j$ , folosim variabila auxiliară `aux`, care va fi masca aplicată elementului `sita[(i*j)/8]`. `aux` va conține 1 pe poziția  $(i*j) \% 8$ , iar în rest – zerouri. Prin aplicarea operatorului OR cu această mască se vor păstra valorile tuturor celorlalți biți, doar al  $(i*j) \% 8$ -lea bit fiind setat cu 1:

```
aux=1; aux <=> (i*j)%8;
sita[(i*j)/8] |= aux;
```

*Program*

```
#include <stdio.h>
#include <memory.h>

#define maxsize 30000
char sita[maxsize/8+1]

int estePrim(int n){
 if(n<=1) return 0;
 return (!isOne(sita[n/8], n%8));
}

int isOne(int n, int poz){
 return
 (n>>poz)&1;
}

void cerne(){
 int i, j, aux;
 memset(sita, 0, sizeof(sita));
 for(i=2; i*i<=maxsize; i++){
 if(isOne(sita[i/8], i%8)==0){
 j=2;
 while(i*j<=maxsize){
 aux=1; aux <= (i*j)%8;
 sita[(i*j)/8] |= aux;
 j++;
 }
 }
 }
}

void main(){
 FILE *fin, *fout;
 int n;
 fin = fopen("numere.in", "r");
 fout = fopen("prime.out", "w");
 cerne();
 while(fscanf(fin, "%d", &n)==1){
 fprintf(fout, "\n%d", n);
 if(estePrim(n)) fprintf(fout, " -- PRIM");
 else fprintf(fout, " -- COMBUS");
 }
 fclose(fin); fclose(fout);
}
```

*Exerciții*

- Modificați programul astfel încât să fie posibilă verificarea numerelor prime până la  $10^9$ .
- Elementele pare mai mari decât 2 sunt neprime. Folosiți această observație pentru a realiza o și mai eficientă compactare a datelor.

*Probleme propuse*

- Să se determine poziția celui mai semnificativ bit cu valoarea 1 din reprezentarea unui număr întreg  $n$ .
- Să se determine dacă un număr întreg  $n$  dat este putere a lui 2.
- Scrieți o metodă care să inverseze valoarea unui bit de la o poziție dată a unui număr întreg.
- Dat fiind un număr natural  $n$ , considerăm mulțimea  $M = \{0, 1, 2, \dots, n\}$ . O submulțime a lui  $M$  poate fi reprezentată ca un vector cu  $n+1$  biți, în care bitul de la poziția  $i$  are valoarea 0 sau 1, după cum elementul  $i$  nu aparține, respectiv aparține submulțimii. Scrieți un program care operează cu astfel de mulțimi, folosind o reprezentare compactă pe biți (creare, apartenență, reunire, diferență, intersecție, afișare).
- Scrieți un program care afișează toate submulțimile mulțimii  $M = \{0, 1, 2, \dots, n\}$  folosind operatorii pe biți.
- Totalul tranzițiilor.* Date fiind două siruri de biți  $w_1$  și  $w_2$  de lungime  $n$ , se definește *distanța Hamming* dintre cele două siruri ca fiind numărul de biți de pe poziții similare care sunt inversați. Exemplu:  $d(1010, 0100)=3$  deoarece pentru fiecare dintre primele 3 poziții, biții corespunzători se inversează. Date fiind  $n$  cuvinte  $w_1, w_2, \dots, w_n$  și o permutare  $s$  a mulțimii  $\{1, 2, \dots, n\}$ , definim numărul total de tranziții corespunzător ca fiind:  $N_r(s, w_1, w_2, \dots, w_n) = \sum_{j=1}^{n-1} d(w_{sj}, w_{sj+1})$  (suma tuturor distanțelor „consecutive” relativ la permutarea  $s$ ). Date fiind  $n$  cuvinte codificate ca numere întregi în baza 10 (cuvintele sunt reprezentările în baza 2) și o permutare  $s$  a mulțimii  $\{1, 2, \dots, n\}$ , determinați numărul total de tranziții.

*Exemple:*

| tastatură                                                              | ecran |
|------------------------------------------------------------------------|-------|
| n=5<br>5432 65 9876 12987 34<br>Permutarea:<br>2 5 1 4 3               | 24    |
| n=7<br>5430 126 7890 1271 987 3212 654<br>Permutarea:<br>1 2 4 3 5 7 6 | 30    |

*Explicație pentru primul exemplu:*  $w_1 = 5432 \rightarrow 00000001010100111000$ ,  $w_2 = 65 \rightarrow 0000000000001000001$ ,  $w_3 = 9876 \rightarrow 00000010011010010100$ ,  $w_4 = 12987 \rightarrow 00000011001010111011$ ,  $w_5 = 34 \rightarrow 00000000000000100010$ . Răspunsul este  $d(w_2, w_5) + d(w_5, w_1) + d(w_1, w_4) + d(w_4, w_3) = 4 + 6 + 7 + 7 = 24$ . Scrieți în acest fel desfășurat pe hârtie și pentru al doilea exemplu.

## Funcții de timp, numere aleatorii

8 probleme complet rezolvate, 21 de exerciții și probleme propuse.

Probleme rezolvate :

- Ziua din săptămână
- rand(), qsort() și bsearch()
- Cap și pajură
- Hârtie-pumn-foarfece
- Timp simplu
- Pauză în secunde
- Data și ora curente
- Trecut sau viitor

*Întâmplarea este singurul stăpân legitim al Universului.*  
Napoleon

### **Problema 1. Ziua din săptămână**

Scrieți un program care afișează ziua din săptămână pentru serbarea zilei de naștere a unei persoane (născute după 1900!), inclusiv pentru anul 2004.

*Exemplu :*

Introduceti data nasterii:

Ziua: 18  
Luna: 3  
Anul: 1989  
Anul 1990: Duminica  
Anul 1991: Luni  
Anul 1992: Miercuri  
Anul 1993: Joi  
Anul 1994: Vineri  
Anul 1995: Sambata  
Anul 1996: Luni  
Anul 1997: Marti  
Anul 1998: Miercuri  
Anul 1999: Joi  
Anul 2000: Sambata  
Anul 2001: Duminica  
Anul 2002: Luni  
Anul 2003: Marti  
Anul 2004: Joi

#### *Analiza problemei și proiectarea soluției*

Limbajul C standard conține *header-ul time.h*, care permite lucrul cu date calendaristice. Tipurile de date *clock\_t* și *time\_t* sunt capabile să reprezinte date calendaristice și timpul-sistem ca întregi de tip *long*.

Tipul structură *tm* este definit astfel:

```
struct tm {
 int tm_sec; /* secunde: 0..59 */
 int tm_min; /* minute: 0..59 */
 int tm_hour; /* ore: 0..23 */
 int tm_mday; /* ziua din luna: 1..31 */
 int tm_mon; /* luna incep. cu Ian: 0..11 */
```

```

int tm_year; /* anul de la 1900: 0..> */
int tm_wday; /* zilele din saptamana incep cu Duminica: 0..6 */
int tm_yday; /* zilele incep cu 1 Ian: 1..365, 366 */
int tm_isdst; /* indicator de timp */
}

```

Funcția `mktimes()`, cu antetul:

```
time_t mktimes(struct tm *timeptr);
```

returnează o dată calendaristică echivalentă cu cea găsită la structura dată ca parametru. Câmpurile din `timeptr` `tm_wday` și `tm_yday` pot să nu fie complete, ele sunt setate de către funcție la momentul apelului.

#### Program

```

#include <stdio.h>
#include "time.h"

void main(){
 char zi[7][9] = {"Duminica", "Luni", "Marti",
 "Miercuri", "Joi", "Vineri", "Sambata"};
 struct tm t;
 time_t t_zi_nastere;
 int z,l,a,i;
 printf("\nIntroduceti data nasterii:\n");
 printf(" Ziua: "); scanf("%d", &z);
 printf(" Luna: "); scanf("%d", &l);
 printf(" Anul: "); scanf("%d", &a);
 for(i=a+1; i<=2004; i++){
 t.tm_year = i - 1900;
 t.tm_mon = l-1;
 t.tm_mday = z;
 t.tm_hour = 0;
 t.tm_min = 0;
 t.tm_sec = 1;
 t.tm_isdst = 0;
 t_zi_nastere = mktimes(&t);
 printf("Anul %d: %s\n",
 i, zi[t.tm_wday]);
 }
}

```

#### Exerciții

- Modificați programul de mai sus astfel încât să afișeze anii până în 2004 inclusiv în care data nașterii a fost sămbătă.

2. Folosind funcția `time()` cu antetul:

```
time_t time(time_t *timer);
```

se poate determina data curentă. Îmbunătăjiți programul anterior astfel încât să se determine toate zilele săptămânii în care a avut respectiva persoană ziua de naștere, până în prezent.

#### Problema 2. `rand()`, `qsort()` și `bsearch()`

Scrieți un program care generează aleatoriu un tablou de numere întregi, folosind funcția `rand()`. Apoi, programul trebuie să afișeze tabloul generat, să îl ordoneze folosind funcția standard C `qsort()`, să citească un element pe care să îl caute în tablou utilizând funcția de bibliotecă `bsearch()`. De asemenea, la început, să se afișeze valoarea constantei `RAND_MAX`, ca în exemplele prezentate mai jos.

##### Exemplu 1:

```
RAND_MAX = 32767
Numarul de elemente al tabloului: 5
```

```
Tabloul introdus: 2151 5088 10568 30539 385
Tabloul ordonat: 385 2151 5088 10568 30539
Elementul de cautat: 10568
Elementul 10568 a fost gasit pe pozitia 4!
```

##### Exemplu 2:

```
RAND_MAX = 32767
Numarul de elemente al tabloului: 4
```

```
Tabloul introdus: 2383 14561 972 2330
Tabloul ordonat: 972 2330 2383 14561
Elementul de cautat: 367
Elementul 367 nu a fost gasit in tablou!
```

#### Analiza problemei și proiectarea soluției

Funcția `rand()` se găsește în librăria `stdlib.h` și returnează un număr întreg cuprins între 0 și `RAND_MAX`.

Înainte de utilizarea funcției `rand()` se folosește funcția `srand()` pentru a seta generatorul de numere aleatorii (în caz contrar, se vor genera mereu aceleași numere!).

Tot în aceeași bibliotecă se găsesc funcțiile `qsort()` și `bsearch()`, cu anteturile:

```
void qsort(void *base, size_t num, size_t width,
 int(__cdecl *compare)(const void *elem1, const void *elem2));

void *bsearch(
 const void *key, const void *base, size_t num, size_t width,
 int (__cdecl *compare)(const void *elem1, const void *elem2));
```

în care parametrii sunt :

- base - adresa de început a tabloului ;
- num - numărul de elemente din tablou ;
- width - dimensiunea unui element în octeți ;
- compare - funcția de comparare ;
- elem1 - pointer la cheia de căutat ;
- elem2 - pointer la elementul de comparat cu cheia.

Funcția `qsort()` nu returnează nimic, doar ordonează tabloul.

Funcția `bsearch()` are în plus parametrul :

- key - adresa elementului de căutat în tablou,

și returnează adresa în tablou a elementului egal cu cheia dată, dacă acesta există, `NULL` în caz contrar.

#### Program

```
#include <stdio.h>
#include <stdlib.h>
#include "time.h"

int compara(const void *a, const void *b){
 return *(int*)a - *(int*)b;
}

void main(){
 int a[200], n, i, elem;
 int* gasit;
 srand((unsigned)time(NULL));
 printf("RAND_MAX = %ld\n", RAND_MAX);
 printf("Numărul de elemente al tabloului: ");
 scanf("%d", &n);
 for(i=0; i<n; i++) a[i] = rand();
 printf("\nTabloul introdus:");
 for(i=0; i<n; i++)
 printf("%6d", a[i]);
 qsort((void*)a, n, sizeof(int), compara);
 printf("\nTabloul ordonat:");
 for(i=0; i<n; i++)
 printf("%6d", a[i]);
 printf("\nElementul de căutat: ");
 scanf("%d", &elem);
 gasit =
}
```

```
bsearch((void*)&elem, (void*)a, n,
 sizeof(int), compara);
if(gasit)
 printf("Elementul %d a fost gasit pe pozitia %d!\n",
 *gasit, gasit-a+1);
else
 printf("Elementul %d nu a fost gasit in tablou!", elem);
}
```

#### Exercițiu

Scripteți o funcție `qsort()` proprie care are aceiași parametri și utilizați-o într-un program similar.

#### Problema 3. Cap și pajură

Presupunem că dispunem de o monedă obișnuită și doi jucători joacă *Cap și pajură* după următoarele reguli :

- a) Se fac un număr total de  $n$ ,  $1 \leq n \leq 200$  aruncări.
- b) Al doilea jucător aruncă moneda și primul spune *cap* sau *pajură*.
- c) Dacă acesta ghicește, i se incrementează scorul și se repetă procedura în același fel.
- d) Dacă acesta nu ghicește, atunci scorul rămâne același și se inversează jucătorii.
- e) La sfârșit trebuie afișat scorul și procentajul de câștig al fiecărui.

#### Exemplu :

`n = 6`

##### Mutarea 1. Jucător 1.

```
Cap('c'/'C' sau Pajura('P'/'p')): p
Moneda: cap
*** SCHIMB JUCATOR ***
Scor j1 = 0, scorj2 = 0
```

##### Mutarea 2. Jucător 2.

```
Cap('c'/'C' sau Pajura('P'/'p')): c
Moneda: cap
Scor j1 = 0, scorj2 = 1
```

##### Mutarea 3. Jucător 2.

```
Cap('c'/'C' sau Pajura('P'/'p')): c
Moneda: cap
Scor j1 = 0, scorj2 = 2
```

##### Mutarea 4. Jucător 2.

```
Cap('c'/'C' sau Pajura('P'/'p')): p
Moneda: pajura
Scor j1 = 0, scorj2 = 3
```

**Mutarea 5. Jucator 2.**

```
Cap('c'/'C' sau Pajura('P'/'p')): p
Moneda: cap ** SCHIMB JUCATOR **
Scor j1 = 0, scorj2 = 3
```

**Mutarea 6. Jucator 1.**

```
Cap('c'/'C' sau Pajura('P'/'p')): C
Moneda: cap
Scor j1 = 1, scorj2 = 3
```

---

```
Scor J1 = 1 Scor J2 = 3
Procent J1 = 16.67% Procent J2 = 50.00%
```

**Analiza problemei și proiectarea soluției**

Tinând cont de faptul că funcția de generat numere aleatorii `rand()` generează un număr întreg între 0 și `RAND_MAX`, o formulă pentru generarea unui număr întreg care să fie 0 sau 1 cu probabilitate egală este:

```
2*((double)rand()/(RAND_MAX+1))
```

În cadrul programului, vom codifica pajura cu 1 și capul cu 0.

Astfel, în interiorul unui ciclu `while()`, simulăm condițiile problemei, programul următor fiind sugestiv pentru descrierea algoritmului utilizat.

**Program**

```
#include <stdio.h>
#include <stdlib.h>
#include "time.h"

void main(){
 int n, i;
 int juc, moneda;
 int scor[2]={0,0};
 char ch;

 srand((unsigned)time(NULL));
 printf("n = ");
 scanf("%d", &n);
 i=0;
 juc = 0;
 while(i < n){
 printf("\nMutarea %d. Jucator %d. ", i+1, juc+1);
 printf("\n Cap('c'/'C' sau Pajura('P'/'p')): ");
 fflush(stdin);
 ch = getchar();
```

```
if('c'==ch || 'C'==ch || 'p'==ch || 'P'==ch){
 ++i;
 moneda = (int)(2*((double)rand()/(RAND_MAX+1)));
 printf(" Moneda: %s\t", moneda==0?"cap":"pajura");
 if(moneda==0 && ('c' == ch || 'C' == ch) ||
 moneda==1 && ('c' == ch || 'P' == ch)) scor[juc]++;
 else
 if(i<n){printf(" ** SCHIMB JUCATOR ** ");juc = 1-juc;}
 printf("\n Scor j1 = %d, scorj2 = %d\n",
 scor[0], scor[1]);
}
else {printf("Trebuie sa introduci c/C/p/P/!!!!");}
printf("\n-----");
printf("\n Scor J1 = %d Scor J2 = %d ", scor[0], scor[1]);
printf("\n Procent J1 = %3.2f% Procent J2 = %3.2f% ",
 (float)(100*scor[0])/n, (float)(100*scor[1])/n);
```

**Exerciții propuse**

- Explicați care este comportarea programului de mai sus dacă stergem linia:  
`fflush( stdin );`
- Modificați programul de mai sus astfel încât să se introducă ca dată de intrare și jucătorul care „este la rând” (1 sau 2).
- Modificați programul de mai sus tinând cont de faptul că punctele c) și d) ale problemei date se modifică astfel:
  - Dacă acesta ghicește, se inversează jucătorii și scorul rămâne același.
  - Dacă acesta nu ghicește, atunci i se incrementează scorul jucătorului care a aruncat moneda și acesta o aruncă din nou.

**Problema 4. Hârtie-pumn-foarfece**

Presupunem că două persoane își folosesc mâna dreaptă pentru a reprezenta următoarele obiecte :

- *hârtie* – palma întinsă ;
- *pumn* – mâna strânsă sub formă de pumn ;
- *foarfece* – două degete depărtate (semnul Victoriei), și joacă un joc. Ei își arată simultan mâna dreaptă în una dintre aceste trei configurații (de mai multe ori). Dacă ei arată același lucru este remiză (nu câștigă nici unul dintre ei). Dacă nu, atunci se aplică una dintre următoarele trei reguli :
  - a) hârtia acoperă pumnul (palma întinsă câștigă față de pumn) ;
  - b) pumnul sparge foarfecele (mâna strânsă câștigă față de semnul Victoriei) ;
  - c) foarfecele tăie hârtia (semnul Victoriei câștigă față de palma întinsă).

Să se simuleze acest joc, generând un număr arbitrar de evenimente și precizând scorul final. Se cere să se joace calculator-calculator.

*Exemplu:*

```
Numar de evenimente: 5
Configuratie 1: pumn hartie

J1 = 0 J2 = 1
Configuratie 2: pumn pumn

J1 = 0 J2 = 1
Configuratie 3: foarfece pumn

J1 = 0 J2 = 2
Configuratie 4: hartie hartie

J1 = 0 J2 = 2
Configuratie 5: pumn foarfece

J1 = 1 J2 = 2
```

*Analiza problemei și proiectarea soluției*

Vom codifica hârtia cu 0, pumnul cu 1 și foarfecele cu 2. Considerând aceste numere doar simboluri, corespunzător cu enunțul problemei putem scrie relațiile: 0 > 1, 1 > 2 și 2 > 0. Pentru a genera un număr întreg 0, 1 sau 2 utilizăm formula:

```
(int) (3 * ((double)rand() / (RAND_MAX + 1)))
```

*Program*

```
#include <stdio.h>
#include <stdlib.h>
#include "time.h"

void main(){
 int semn[2], scor[2] = {0, 0};
 int k, i = 0, n;
 printf("Numar de evenimente: ");
 scanf("%d", &n);
 srand((unsigned)time(NULL));
 while(i < n){
 semn[0] = (int)(3 * ((double)rand() / (RAND_MAX + 1)));
 semn[1] = (int)(3 * ((double)rand() / (RAND_MAX + 1)));
 printf("\nConfiguratie %d:", ++i);
 for(k = 0; k < 2; k++)
 if(semn[k] == 0) printf(" pumn");
 else if(semn[k] == 1) printf(" hartie");
 else if(semn[k] == 2) printf(" foarfece");
 }
}
```

```
switch(semn[k]){
 case 0: printf(" hartie "); break;
 case 1: printf(" pumn "); break;
 case 2: printf(" foarfece ");
}
if(semn[0] - semn[1] == -1) scor[0]++;
if(semn[0] - semn[1] == 1) scor[1]++;
if(semn[0] - semn[1] == 2) scor[0]++;
if(semn[0] - semn[1] == -2) scor[1]++;
printf("\n *****\n J1 = %d J2 = %d\n",
 scor[0], scor[1]);
}
```

*Exercițiu*

Transformați programul următor astfel încât să existe posibilitatea de a fi jucat *persoană-calculator* și *persoană-persoană*.

*Problema 5. Timp simplu*

Scrieți un program care execută un ciclu foarte mare de iterații (while, do...while sau for) contorizat astfel încât să afișați durata în secunde și numărul de tacturi CPU în secunde, folosind funcțiile *time()* și *clock()* din biblioteca *time.h*.

*Exemplu:*

| intrare | ieșirea (ecran)                                 |
|---------|-------------------------------------------------|
|         | Timp = 12 secunde<br>CPU Time = 11.0780 secunde |

*Analiza problemei și proiectarea soluției*

Funcția *clock()* returnează timpul-sistem într-o variabilă de tip *clock\_t*. Funcția *time()* returnează timpul real (de obicei, măsurat după o origine arbitrară, de exemplu miezul nopții, 1 ianuarie 1970). Pentru a converti timpul-sistem în secunde va trebui să îl împărțim la constanta *CLOCKS\_PER\_SEC*.

*Program*

```
#include <time.h>
#include <math.h>
#include <conio.h>

main(){
 unsigned long i;
 time_t start, acum;
 clock_t start_cpu, acum_cpu;
```

```

i = pow(10, 12);
start=time(NULL);
start_cpu = clock();
while(i) i--;
acum=time(NULL);
acum_cpu = clock();

printf("Timp = %d secunde\n", acum-start);
printf("CPU Time = %.8lf secunde\n",
 (acum_cpu-start_cpu)/(double)CLOCKS_PER_SEC);
getch();
}

```

### Exercițiu

Creați o aplicație asemănătoare folosind funcția `difftime()`.

### Problema 6. Pauză în secunde

Să se scrie o funcție care face o pauză de maximum 100 de secunde cerute de utilizator.

*Exemplu :*

```

Nr. secunde asteptare = 4
Asteptare 4 secunde
Gata cu asteptarea!

```

### Analiza problemei și proiectarea soluției

Dacă `n` este numărul de secunde de așteptare, atunci va trebui să așteptăm `n*CLOCKS_PER_SEC` tacturi ale mașinii, transformate în tipul `clock_t`. Funcția `sleep()` adaugă la timpul curent acest număr de tacturi și într-o buclă `while()` testează dacă s-a ajuns la această valoare.

### Program

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>

void sleep(clock_t wait);

int main(void){
 short n;
 /*Asteapta un numar de secunde...*/
 printf("Nr. secunde asteptare = ");

```

```

scanf("%d", &n);
printf("Asteptare %d secunde\n", n);
sleep((clock_t)n * CLOCKS_PER_SEC);
printf("Gata cu asteptarea!\n");
getch();
}

/* Pauza pentru un nr specificat de secunde.*/
void sleep(clock_t wait){
 clock_t goal;
 goal = wait + clock();
 while(goal > clock());
}

```

### Exercițiu

Modificați programul anterior astfel încât la fiecare secundă scursă să afișeze simbolul `..`.

### Problema 7. Data și ora curente

Să se scrie un program care afișează data și ora curente.

*Exemplu :*

```

Data de azi: Tue Jan 13 08:53:26 2004

```

### Analiza problemei și proiectarea soluției

Vom utiliza funcția `time()` pentru a obține timpul curent, apoi `localtime()` pentru a converti această valoare la timpul local și apoi `asctime()` pentru a converti timpul într-un sir de caractere.

### Program

```

#include <stdio.h>
#include <time.h>
#include <conio.h>

main(){
 time_t acum;
 struct tm date_time;
 acum = time(NULL);
 date_time = *localtime(&acum);
 printf("Data de azi: %s", asctime(&date_time));
 getch();
}

```

**Exercițiu**

Scrieți data în limba română.

**Problema 8. Trecut sau viitor**

Afișați data în trecut sau în viitor cu un număr de zile introdus de utilizator. Acest număr de zile poate fi negativ sau pozitiv și o instanță a programului va executa mai multe cazuri până la introducerea numărului 0.

*Exemplu :*

```
Numar de zile: 45
Data este Fri Feb 27 09:40:26 2004
Numar de zile: -89
Data este Thu Oct 16 10:40:26 2003
Numar de zile: 147
Data este Tue Jun 08 10:40:26 2004
Numar de zile: 2000
Data este Sun Jul 05 10:40:26 2009
Numar de zile: -2345
Data este Tue Aug 12 10:40:26 1997
Numar de zile: 0
```

**Analiza problemei și proiectarea soluției**

Tinând cont că o zi are 86400 secunde, se va adăuga la data curentă, furnizată de funcția `time()`, numărul `dif*86400`, unde `dif` este numărul de zile dorit (care poate fi negativ sau pozitiv).

**Program**

```
#include <stdio.h>
#include <time.h>
#include <conio.h>

main(){
 int dif;
 time_t acum, nou;
 time(&acum);
 while(1){
 printf("Numar de zile: ");
 scanf("%d", &dif);
 if(!dif)break;
 nou = acum+dif*86400;
 if(nou<0)
 printf("Depasire... \n");
 }
}
```

```
 else
 printf("Data este %s", ctime(&nou));
 getch();
}
```

**Exercițiu**

Scrieți data în limba română.

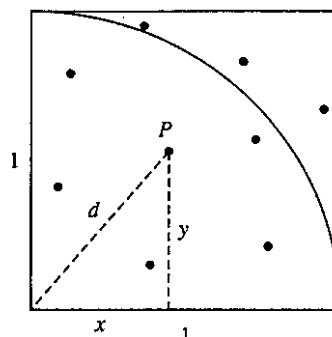
**Probleme propuse**

1. Scrieți un program care calculează câte zile a trăit o persoană cu data nașterii dată de la tastatură.
2. Să se scrie un program care calculează numărul de zile cuprinse între două date calendaristice introduse de la tastatură.
3. Testați funcțiile de bibliotecă `qsort()`, `find()`, `search()` și `bsearch()` pentru diverse tipuri de tablouri (întregi, reale, pointeri la siruri).
4. Scrieți un program care simulează jocul la LOTO, cunoscând numărul de participanți `p`, numărul `m` de numere extrase din urnă ( $m \leq 10$ ) și numărul `L` de numere cuprinse între 1 și 99 care se trec pe un bilet LOTO (`L` fixat,  $L \leq 9$ ). Pentru fiecare participant la joc, se cunosc: numele, numărul biletului LOTO, cele `L` numere trecute pe bilet și numerele extrase din urnă.
5. Scrieți un program ajutător pentru învățarea înmulțirii și adunării cifrelor de la 0 la 9. Operația și cele două cifre se vor genera aleatoriu, utilizatorul va trebui să introducă răspunsul. Pentru răspuns corect, se adună un punct, pentru răspuns incorrect se scade un punct. La sfârșit trebuie făcut un raport despre sesiunea de învățare (număr total de întrebări, câte au fost corecte, câte au fost greșite etc.).
6. Să se scrie un program care simulează aruncarea unui zár de `n` ori și afișează numărul de apariții ale fiecărei fețe.
7. *Metoda Monte Carlo a lui Dewdney.* Metodele *Monte Carlo* (după Monte Carlo, Monaco – loc de referință pentru jocurile de noroc) sunt metode care implică tehnici probabilistice, cum ar fi generarea aleatorie, pentru determinarea soluțiilor unor probleme de matematică, fizică, economie... În această categorie se înscrie și metoda lui Dewdney de calcul al lui  $\pi$ . Această metodă a fost descrisă de *A.K. Dewdney* în 1988. Considerăm un pătrat de latură 1 în care este înscris un sfert de cerc, ca în figură. Presupunem că o mulțime de puncte aleatorii sunt „aruncate” în acest pătrat. Deoarece aria pătratului este 1 și a sfertului de cerc  $\pi/4$ , deducem că probabilitatea ca un punct aleatoriu să se afle în interiorul sfertului de cerc este  $\pi/4$ . Considerăm punctul  $P(x, y)$  cu proprietatea:

$$x, y \in [0, 1] \quad (1)$$

$P$  se află în interiorul sfertului de cerc dacă:

$$x^2 + y^2 < 1 \quad (2)$$



și în afara să dacă :

$$x^2 + y^2 \geq 1 \quad (4)$$

Dacă vom nota cu  $N_{tot}$  numărul total de puncte considerate și cu  $N_{int}$  numărul celor care se află în interiorul cercului, atunci deducem :

$$\frac{N_{int}}{N_{tot}} \approx \frac{\pi}{4} \quad (5)$$

de unde deducem valoarea aproximativă pentru  $\pi$  :

$$\pi \approx 4 \frac{N_{int}}{N_{tot}} \quad (6)$$

Prezentăm în continuare un experiment pentru valori într-adevăr mari ale lui  $n$ . Pentru trei miliarde de puncte, afișăm rezultatele parțiale și remarcăm că valoarea dată de metodă se apropie de  $\pi$  :

| $N_{tot}$  | $N_{int}$  | $\pi$                        |
|------------|------------|------------------------------|
| 500000000  | 392688018  | $\approx 3,1415041440000002$ |
| 1000000000 | 785385468  | $\approx 3,1415418719999999$ |
| 1500000000 | 1178073866 | $\approx 3,1415303093333335$ |
| 2000000000 | 1570763251 | $\approx 3,1415265020000001$ |
| 2500000000 | 1963450604 | $\approx 3,1415209663999999$ |
| 3000000000 | 2356144139 | $\approx 3,1415255186666666$ |

Determinați valoarea lui  $\pi$  folosind acest algoritm :

```

ALGORITM_MONTE_CARLO
 n, n_In ← numar_aruncari
 n_In ← 0
 for(i=1; i<n; step 1) execute
 x ← random([0, 1])
 y ← random([0, 1])
 if(x^2+y^2 < 1) execute
 n_In++
 end_if
 end_for
 return 4.0*n_In/n;
END_ALGORITM_MONTE_CARLO

```

8. Scrieți programe care să reprezinte grafic într-un mod cât mai sugestiv metoda Monte Carlo a lui Dewdney.
9. Metoda Monte Carlo a lui Dewdney poate fi încă rafinată prin considerarea doar a unei valori aleatorii din intervalul  $[0, 1]$ . Pentru valorile aleatorii  $x_1, x_2, \dots, x_n$  din acest interval se calculează corespunzător, folosind ecuația cercului  $y_1, y_2, \dots, y_n$  cu

proprietatea:  $y_i = \sqrt{1 - x_i^2}$ . Atunci, aria sfertului de cerc se poate aproxima cu aria unui dreptunghi cu laturile 1 și media aritmetică a numerelor  $y_i$ . În acest fel,  $\pi$  se poate aproxima cu valoarea  $4 \cdot \frac{y_1 + y_2 + \dots + y_n}{n}$ . Scrieți un program care utilizează această metodă.

10. *Metoda acelor lui Buffon*. Există numeroase alte metode mult mai eficiente de calcul al constantei  $\pi$ . Metoda *Monte Carlo* de mai sus o determină doar cu instrumente simple din teoria probabilităților, fără a utiliza formalizări matematice elaborate. O altă celebră metodă ce se înscrie în acest tip și calculează valoarea constantei  $\pi$  este *Metoda acelor lui Buffon*. În 1777, nobilul francez *Georges Louis Lecrec* (1707-1788), supranumit și *Contele de Buffon*, a propus următoarea problemă: *Dacă cineva lasă să cadă un ac pe o filă liniată echidistant, care este probabilitatea ca acul să încrucișeze o linie?* Această probabilitate depinde, desigur, de lungimea acului 1 și de distanța d dintre linii pe hârtie. Presupunem că acul este mai scurt decât distanța dintre linii:  $1 \leq d$ . Răspunsul la problema de mai sus este surprinzător pentru că în rezultat apare (ați ghicit!) celebrul  $\pi$ . Și anume, această valoare este  $\frac{2 \cdot 1}{\pi \cdot d}$ . Rezultatul implică faptul că o valoare aproximativă pentru  $\pi$  poate fi calculată experimental: dacă un ac este lăsat de  $N$  ori să cadă și de  $T$  ori atinge o linie, atunci  $\frac{T}{N}$  este aproximativ  $\frac{2 \cdot 1}{\pi \cdot d}$ , de unde deducem că  $\pi$  este aproximativ  $\frac{2 \cdot 1 \cdot N}{d \cdot T}$ . Scrieți un program care determină valoarea aproximativă a lui  $\pi$  utilizând *metoda lui Buffon*. Se va considera o succesiune de ace aleatorii, determinate de unghiul (cuprins în  $[0, \pi]$ ) pe care acestea îl formează cu liniile paralele și mijlocul lor. Determinați formula care decide dacă un ac a intersectat o linie. Scrieți și un program care să ilustreze grafic procedeul (se consideră un pătrat de latură 6 și două linii paralele la distanțele 2, respectiv 4; acele vor avea lungimea 1 și cele ce intersectează liniile vor fi colorate diferit de celelalte... Exemplu: [www.fh-friedberg.de/users/jingo/mathematics/buffon/buffonneedle.html](http://www.fh-friedberg.de/users/jingo/mathematics/buffon/buffonneedle.html)). Construiți o demonstrație a rezultatului lui Buffon!



## Structuri cu autoreferire

11 probleme complet rezolvate, 38 de exerciții și probleme propuse.  
Probleme rezolvate :

- Cuvinte în propoziție
- Cuvinte cu majuscule ordonate crescător
- Cuvinte cu majuscule ordonate crescător și luate o singură dată
- Matrice rare
- Joc de copii I
- Joc de copii II
- Tabela de dispersie (*Hash Table*)
- Crearea listelor
- Stiva (*Stack*)
- Cărți de joc
- Parcurgerea arborelui binar

*Nu îndrăznim nu pentru că este greu, ci este  
greu pentru că nu îndrăznim.*

Seneca

### ***Problema 1. Cuvinte în propoziție***

Fiind introdusă de la tastatură o propoziție care se termină cu unul dintre caracterele „.”, „!” sau „?”, să se scrie un program utilizând liste care afișează cuvintele din propoziție în ordinea introducerii, câte unul pe linie. Lungimea cuvintelor nu depășește 15 caractere.

*Exemplu :*

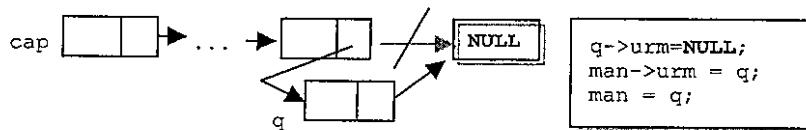
| intrare (tastatură)                                                                                   | iesire (ecran)                                                                                                                |
|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Propozitia:<br>Viata este 1 2 ceea3 ce se3<br>intampla34 cand esti ocupat cu alte<br>456897: planuri. | Cuvintele din propozitie:<br>Viata<br>este<br>ceea<br>cé<br>se<br>intampla<br>cand<br>esti<br>ocupat<br>cu<br>alte<br>planuri |

### ***Analiza problemei și proiectarea soluției***

Vom construi secvențial o listă simplu înlățuită care memorează cuvintele din propoziția introdusă. Cuvintele se construiesc la citire : când se întâlnește un caracter care nu este literă, se verifică dacă este succesorul imediat al unui cuvânt ; dacă da, atunci se adaugă cuvântul respectiv la listă și se reinicializează dimensiunea cuvântului curent cu 0 (șirul vid). Pe liniile 6-9 se definește structura TCellula, care conține câmpul de informație cuvant și un pointer la elementul următor din listă. În liniile 12-15 se declară și se inițializează variabilele necesare : cap (va fi întotdeauna primul element al listei căutate, inițializat cu NULL), man (un pointer de manevră de tip TCellula), șirul de caractere cuv pentru a stoca cuvântul curent citit și dimensiunea sa k (inițializată cu șirul vid), caracterul c care va parcurge caracterele citite. În cadrul unui ciclu cu test final do {...}while citim caracterul c. Dacă acesta nu este literă, atunci avem două posibilități : ori este succesor imediat după un cuvânt, caz în care cuvântul se adaugă la listă și apoi se inițializează dimensiunea sa cu 0, ori cuvântul curent este vid și se trece mai departe. Acest lucru se

execută în cadrul secvenței 19-28. Transcris în C, adăugarea unui nou cuvânt înseamnă alocarea memoriei pentru un element *q* de tip *TCelula* (linia 21), completarea câmpului de informație cu cuvântul curent folosind funcția *strcpy* din biblioteca *string.h* (linia 22); acest element *q* va fi adăugat ultimul la listă, deci successorul său este *NULL* (linia 23); lungimea cuvântului curent *k* se initializează cu 0. Inserarea cuvântului se execută în liniile 25-26: dacă lista este vidă, atunci se initializează cu *q* și *man* devine tot *q*, altfel successorul lui *man* devine *q* și *man* devine *q* (*man* va reține mereu ultimul element din listă!).

Cazul când *cap* nu este vid se poate reprezenta grafic astfel:



În cazul în care caracterul *c* citit este literă, se adaugă la cuvântul curent *cuv* prin completarea ultimului caracter, deplasarea cu variabila *ce* indică numărul de caractere la dreapta și adăugarea caracterului „\0” ce indică sfârșit de sir (linia 29). Testarea caracterului citit pentru a vedea dacă este sfârșit de propoziție se realizează utilizând funcția *strchr()* din biblioteca *string.h*, care returnează un pointer la prima apariție a unui caracter într-un sir, dacă acesta se află în sir, și *NULL* în caz contrar.

### Program

```

1: #include <stdio.h>
2: #include <string.h>
3: #include <ctype.h>
4: #include <malloc.h>

5: char sfarsit[] = "?!.,";

6: typedef struct nod{
7: char cuvant[16];
8: struct nod *urm;
9: } TCelula;

10: void main()
11: {

12: TCelula *cap, *man;
13: char cuv[16], c;
14: int k;
15: cap = NULL; cuv[0] = '\0'; k = 0;
16: printf("Propozitia:\n");

17: do {
18: scanf("%c", &c);

```

```

19: if(!isalpha(c)) {
20: if(k) {
21: TCelula *q = (TCelula*)malloc(sizeof(TCelula));
22: strcpy(q->cuvant, cuv);
23: q->urm = NULL;
24: k = 0;
25: if(cap == NULL) {cap = q; man = q; }
26: else { man->urm = q; man = q; }
27: }
28: }
29: else { cuv[k++]=c; cuv[k]='\0'; }
30: } while(!strchr(sfarsit, c));

31: printf("\nCuvintele din propozitie:");
32: while(cap){
33: printf("\n%s", cap->cuvant);
34: man = cap;
35: cap = cap ->urm;
36: free(man);
37: }
38: }

```

### Exerciții

- Dacă unele cuvinte se repetă, modificați programul anterior astfel încât un astfel de cuvânt să se afișeze o singură dată (la adăugare se va verifica în plus dacă respectivul cuvânt este deja în listă).
- Operați numărul minim de modificări astfel încât să se afișeze toate cuvintele în ordinea inversă a introducerii fără a mai utiliza variabila de manevră *man*.

### Problema 2. Cuvinte cu majuscule ordonate crescător

Fiind introdusă de la tastatură o propoziție care se termină cu unul dintre caracterele „.”, „!” sau „?”, să se scrie un program utilizând liste care afișează cuvintele din propoziție în ordine alfabetică scrise cu majuscule, câte unul pe linie. Lungimea cuvintelor nu depășește 15 caractere.

#### Exemplu :

| intrare (tastatură)                                   | iesire (ecran)                                                          |
|-------------------------------------------------------|-------------------------------------------------------------------------|
| Propozitia:<br>@3Nu vei afla, daca58* nu<br>90intreb! | Cuvintele din propozitie:<br>AFLA<br>DACA<br>INTREBI<br>NU<br>NU<br>VEI |

### *Analiza problemei și proiectarea soluției*

Problema se asemănă extrem de mult cu cea anterioară, cu deosebirile că modalitatea de construire a cuvântului curent este diferită (se vor adăuga majusculele) și că lista trebuie păstrată ordonată. Se va înlocui linia 29 cu linia :

```
29: else { cuv[k++]=toupper(c); cuv[k]='\0'; }
```

Pentru introducerea lui q în listă și păstrarea proprietății de ordonare vom înlocui linia 26 cu secvența :

```
26.01: else {
26.02: man = cap;
26.03: if(strcmp(cap->cuvant, q->cuvant)>0) {
26.04: q->urm=cap; cap=q;
26.05: } else {
26.06: while(man->urm&&strcmp(man->urm->cuvant, q->cuvant)<0)
26.07: { man=man->urm; } /*end while*/
26.08: if(man->urm) {
26.09: q->urm = man -> urm; man -> urm = q;
26.10: } else{
26.11: man -> urm = q;
26.12: } /*3.else*/ } /*2. else*/ } /*1.else*/
```

Primul else de la linia 26.01 se referă la faptul că lista cap construită succesiv până acum nu este vidă, deci va trebui să decidem poziția unde se inserează elementul q. Dacă cuvântul construit este mai mic decât primul cuvânt din cap, nodul q se va adăuga înaintea acestuia (26.03-26.05), altfel se va parurge lista cât timp cuvântul următor este mai mare decât cel de inserat (26.06-26.07). Dacă există un cuvânt mai mare sau egal cu cuvântul de inserat, atunci anteriorul ciclului while{} se va opri și în succesorul lui man îl deținem pe primul cu această proprietate, vom adăuga q ca fiind succesorul lui man cu refacerea legăturilor necesare (26.08-26.10). Dacă cuvântul tocmai creat este mai mare decât ultimul cuvânt din listă (implicit decât toate cuvintele, deoarece ultimul este cel mai mare), atunci se face adăugarea lui q la sfârșit (26.10-26.12).

Programul rămâne ca exercițiu.

### *Exerciții*

1. Modificați programul astfel încât să se afișeze cuvintele în ordine descrescătoare.
2. Modificați programul astfel încât la citirea caracterelor să se creeze lista fără a se face și ordonarea (fără folosirea variabilei auxiliare man), scrieți o funcție de tip *Bubble Sort* care face ordonarea interschimbând doar informațiile din noduri și folosiți-o la afișare.

### *Problema 3. Cuvinte cu majuscule ordonate crescător și luate o singură dată*

Fiind introdusă de la tastatură o propoziție care se termină cu unul dintre caracterele „.”, „!” sau „?”, să se scrie un program utilizând liste care afișează cuvintele din propoziție în ordinea alfabetice scrisă cu majuscule către o sigură dată și cu numărul de apariții, căte unul pe linie, ca în exemplul următor. Lungimea cuvintelor nu depășește 15 caractere.

*Exemplu :*

| intrare (tastatură)                                                                                            | ieșire (ecran)                                                                                         |
|----------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Propozitia:<br>Cel ce\$^\$ nu<br>poate *&cea ^&\$ce<br>vrea<br>,<br>trebuie sa vrea %\$\$%cea2312 ce<br>poate! | Cuvintele din propozitie:<br>CE 3<br>CEL 1<br>CEEA 2<br>NU 1<br>POATE 2<br>VREA 2<br>SA 1<br>TREBUIE 1 |

### *Analiza problemei și proiectarea soluției*

Vom transforma tipul TCellula prin adăugarea unei noi informații nr de tip int, care va memora numărul de apariții ale cuvântului respectiv în propoziție. Acest tip devine :

```
typedef struct nod{
 char cuvant[16];
 int nr;
 struct nod *urm;
} TCellula;
```

Vom opera asupra programului de la problema anterioară, cu diferența că atunci când se creează un nou cuvânt se va verifica dacă acesta este deja în listă; dacă da, se va incrementa numărul de cuvinte al acestuia și se va trece la citirea următorului cuvânt :

```
man = cap;
while(man && strcmp(man->cuvant, cuv))
 { man = man ->urm; }
if(man) {
 man->nr++; k=0; cuv[0]='\n';
 continue;
}
```

Variabila q trebuie definită în linia 12, care devine :

```
12: TCellula *cap, *man, *q;
```

și este inițializată în linia 21:

```
| 21: *q = (TCelula*)malloc(sizeof(TCelula));
```

Când se inițializează variabila q, se va inițializa și numărul de cuvinte, prin inserarea liniei 22.1 după linia 22:

```
| 22.1: q->nr = 1;
```

Se mai modifică afișarea, prin înlocuirea liniei 33:

```
| 33: printf("\n%s %d", cap->cuvant, cap->nr);
```

Programul rămâne ca exercițiu.

### Exerciții

- Modificați programul astfel încât să se afișeze cuvintele și numărul în ordine descrescătoare.
- Modificați programul astfel încât la citirea caracterelor să se creeze lista fără a se face și ordonarea (fără folosirea variabilei auxiliare man); scrieți o funcție de tip *Bubble Sort* care face ordonarea interschimbând doar informațiile din noduri și folosiți-o la afișare.

### Problema 4. Matrice rare

Spunem că o matrice este matrice rară dacă numărul de elemente diferite de zero conținute de aceasta este foarte mic. Se cere scrierea unui program care citește și afișează astfel de matrice, dar calculează și suma lor. Se vor utiliza structuri de date simplu înălțuită. Vom considera că matricile sunt pătratice de dimensiune cel mult 100, introduse de la tastatură, și că elementele diferite de zero (și în matricea sumă) se încadrează în tipul int. Se va efectua în cod validarea pozițiilor matricei și a valorii introduse (nu este permis să fie nulă). Nu uitați să eliberați la sfârșit memoria alocată!

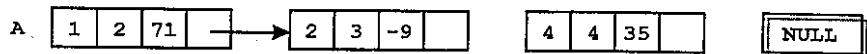
*Exemplu :*

| intrare (tastatură)                | ieșire (ecran)                               |
|------------------------------------|----------------------------------------------|
| n = 4                              | REZULTATE:                                   |
| Introduceti prima matrice:         | Matricea A:                                  |
| Numar de elemente diferite de 0: 3 | 0 71 0 0<br>0 0 -9 0<br>0 0 0 0<br>0 0 0 35  |
| Elementul 1:                       |                                              |
| Linia: 1                           |                                              |
| Coloana: 2                         |                                              |
| Valoare: 71                        | Matricea B:                                  |
| Elementul 2:                       | 0 65 0 0<br>0 0 0 0<br>0 0 0 0<br>0 0 0 -893 |
| Linia: 4                           |                                              |
| Coloana: 4                         |                                              |
| Valoare: 35                        |                                              |

| intrare (tastatură)                                               | ieșire (ecran)                                                             |
|-------------------------------------------------------------------|----------------------------------------------------------------------------|
| Elementul 3:<br>Linia: 2<br>Coloana: 3<br>Valoare: -9             | Matricea sumă rezultată:<br>0 136 0 0<br>0 0 -9 0<br>0 0 0 0<br>0 0 0 -858 |
| Introduceti a doua matrice:<br>Numar de elemente diferite de 0: 2 |                                                                            |
| Elementul 1:<br>Linia: 4<br>Coloana: 4<br>Valoare: -893           |                                                                            |
| Elementul 2:<br>Linia: 1<br>Coloana: 2<br>Valoare: 65             |                                                                            |

### Analiza problemei și proiectarea soluției

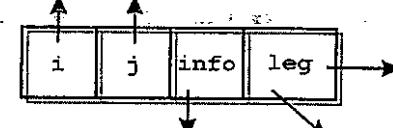
Vom considera o structură simplu înălțuită, în cadrul căreia fiecare celulă conține patru informații: linia, coloana, valoarea diferită de zero și adresa următorului element din listă. De exemplu, pentru matricea A din exemplul de mai sus, lista corespunzătoare matricei este:



Vom defini tipul MRara astfel:

linia, respectiv coloana în matrice

```
typedef struct nod{
 short i, j;
 int info;
 struct nod* leg;
} MRara;
```



informația adresa următorului element

Utilizând acest tip, vom scrie funcțiile necesare rezolvării problemei date: Citeste-Matrice(), compPoz(), ScrieMatrice() și SumaMatrici().

Vom scrie funcția compPoz() cu antetul:

```
short compPoz(MRara* p, MRara* q)
```

care testează relația de ordine lexicografică dintre perechile (line, coloană) ale celor două elemente p și q și returnează -1 dacă poziția lui p este înainte de poziția lui q, 0 dacă se află pe aceeași poziție și 1 dacă poziția lui p este mai mare decât poziția lui q (de

exemplu  $(1, 2) < (1, 3) < (4, 4) \dots$ . Vom utiliza această funcție pentru a crea lista-matrice în ordine (strict) crescătoare a pozițiilor elementelor nenule.

În cadrul funcției `CitesteMatrice()`, pentru fiecare element citit se testează mai întâi dacă poziția și valoarea sunt valide; în caz contrar se afișează un mesaj corespunzător și se continuă cu citirea următorului element:

```
if(l>n || c>n || l<1 || c<1)
 printf(" Poziție incorrectă!!!"); continue;
printf(" Valoare: "); scanf(" %d", &val);
if(val == 0){ "Valoare nula!!"; continue; }
```

Dacă poziția și valoarea introduse sunt corespunzătoare, va trebui operată introducerea acesteia în matricea-listă. Vom aloca un element pentru acest scop și îi vom completa câmpurile corespunzătoare cu valorile tocmai citite:

```
q = (MRara*) malloc(sizeof(MRara));
q->i = l; q->j = c; q->info = val;
```

Depinzând de elementele deja introduse în listă, se va găsi poziția corespunzătoare elementului `q` nou creat; dacă nu a fost adăugat încă nici un element, atașăm `q` matricei finale:

```
if(*matr == NULL){ q->leg=NULL; *matr=q; }
```

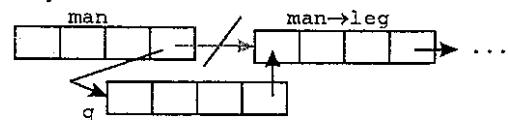
Altfel, dacă poziția este mai mică decât primul element din matrice, se va insera `q` înaintea sa și se va actualiza pointerul `*matr` (acesta va fi tot timpul primul element al listei):

```
if(compPoz(q, *matr) < 0){
 q->leg = *matr;
 *matr = q;
}
```

Altfel, vom parcurge matricea-listă până la găsirea poziției corespunzătoare; folosim variabila auxiliară `man` pentru a ne deplasa în listă (matr trebuie să rămână adresa primului element!).

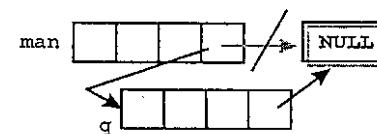
```
man = *matr;
while(man->leg && compPoz(man->leg, q) < 0)
{ man = man->leg; }
if(man->leg){
 q->leg = man->leg;
 man->leg = q;
} else{
 q->leg = NULL;
 man->leg = q;
}
```

După execuția ciclului `while()`, știm că `q` trebuie adăugat imediat după `man`; dacă `man` mai are elemente valide după el, vom stabili legăturile corespunzătoare: succesorul lui `q` devine succesorul lui `man` și succesorul lui `man` devine `q`. Dacă `man` nu are nici un succesor: succesorul lui `q` este adresa vidă `NULL` și succesorul lui `man` este `q`. Grafic, pentru primul caz:



```
if(man->leg){
 q->leg = man->leg;
 man->leg = q;
}
```

Pentru al doilea caz:



```
if(!man->leg){
 q->leg = NULL;
 man->leg = q;
}
```

Funcția `ScrieMatrice()` utilizează faptul că elementele sunt ordonate lexicografic după poziție în matricea-listă. Se vor parcurge iterativ toate pozițiile și dacă poziția curentă este în listă, se va afișa și ne deplasăm cu un element la dreapta, altfel se va afișa zero:

```
elem = 0;
if(matr && i==matr->i && j==matr->j){
 elem = matr->info;
 matr = matr->leg;
}
printf("%6d", elem);
```

Funcția `SumaMatrici()` cu antetul

```
MRara* SumaMatrici(MRara* A, MRara *B)
```

returnează suma matricelor `A` și `B` introduse ca parametri. Tehnica de determinare a matricei sumă este asemănătoare cu cea de obținere a vectorului ordonat din doi vectori deja ordonați, folosită, de exemplu, în *Merge Sort* (sortarea prin interclasare). Cât timp ambele liste sunt nevide, se compară pozițiile primelor elemente din cele două liste și se prelucrează corespunzător. Primul lucru care se execută este alocarea spațiului pentru variabila nouă ce se va crea și setarea succesorului `NULL`:

```
q = (MRara*) malloc(sizeof(MRara));
q->leg = NULL;
```

Dacă poziția lui A este mai mică decât poziția lui B, atunci vom seta valorile corespunzătoare din q cu cele din A și elementul curent din A devine neinteresant, după care ne deplasăm la elementul succesor:

```
if(compPoz(A, B) < 0){
 q->i = A->i; q->j=A->j; q->info = A->info;
 A = A->leg;
}
```

Dacă poziția lui A este aceeași cu poziția lui B, atunci setăm informația din q ca fiind suma informațiilor din A și B și pointerii A și B vor fi deplasati pe legăturile lor:

```
if(compPoz(A, B) == 0){
 q->i = A->i; q->j = A->j;
 q->info = A->info + B->info;
 A = A->leg; B = B->leg;
}
```

Altfel, stim că poziția lui B este mai mică decât cea a lui A și efectuăm din nou operațiile necesare:

```
q->i = B->i; q->j = B->j; q->info = B->info;
B = B->leg;
```

După executarea acestor operații, mai rămâne de inserat elementul nou creat în listă. Dacă aceasta este vidă, atunci se vor inițializa cu q atât lista C, cât și variabila auxiliară man (care este ultimul element al listei!):

```
if(C == NULL){ C = q; man = q; }
```

Altfel, se va lega acest element la ultimul element curent man și se va reactualiza man:

```
man->leg = q;
man = q;
```

Dacă mai sunt elemente în una dintre listele A sau B, acestea se adaugă după man, în caz că acesta există:

```
if(A) { if(man) man->leg = A; else C = A; }
if(B) { if(man) man->leg = B; else C = B; }
```

### Program

```
#include <stdio.h>
#include <malloc.h>
#include <cchio.h>
```

```
typedef struct nod{
 short i, j;
 int info;
 struct nod* leg;
} MRara;

short compPoz(MRara* p, MRara* q){
 short rez;
 if(p->i < q->i) rez = -1;
 if(p->i == q->i && p->j<q->j) rez = -1;
 if(p->i == q->i && p->j == q->j) rez = 0;
 if(p->i > q->i) rez = 1;
 if(p->i == q->i && p->j>q->j) rez = 1;
 return rez;
}

void CitesteMatrice(MRara **matr, short n){
 short l, c;
 int i, k, val;
 MRara *man, *q;
 printf("\n Numar de elemente diferite de 0: ");
 scanf("%d", &k);
 *matr = NULL;
 for(i=0; i<k; i++){
 printf(" Elementul %d: \n", i+1);
 printf(" Linia: ");scanf("%d", &l);
 printf(" Coloana: ");scanf("%d", &c);
 if(l>n || c>n || l<1 || c<1)
 { printf(" Poziție incorectă!!"); continue; }
 printf(" Valoare: ");scanf("%d", &val);
 if(val == 0){"Valoare nula!!"; continue; };
 q = (MRara*)malloc(sizeof(MRara));
 q->i = l; q->j = c; q->info = val;
 if(*matr == NULL){ q->leg=NULL; *matr=q; } else
 if(compPoz(q, *matr) < 0){
 q->leg = *matr;
 *matr = q;
 } else
 { man = *matr;
 while(man->leg && compPoz(man->leg, q) < 0)
 { man = man->leg; }
 if(man->leg){
 q->leg = man->leg;
 man->leg = q;
 }
 }
 }
}
```

```

 } else {
 q->leg = NULL;
 man->leg = q;
 } /* else */
 } /* for */
} /* CitesteMatrice*/

void ScrieMatrice(MRara *matr, short n){
 short i, j;
 int elem;
 for(i=1; i<=n; i++){
 printf("\n");
 for(j=1; j<=n; j++){
 elem = 0;
 if(matr && i==matr->i && j==matr->j){
 elem = matr->info;
 matr = matr->leg;
 }
 printf("%6d", elem);
 }
 printf("\n");
 }
}

MRara* SumaMatrici(MRara* A, MRara *B){
 MRara *C, *man, *q;
 C = NULL;
 while(A && B){
 q = (MRara*) malloc(sizeof(MRara));
 q->leg = NULL;

 if(compPoz(A, B) < 0){
 q->i = A->i; q->j=A->j; q->info = A->info;
 A = A->leg;
 } else
 if(compPoz(A, B) == 0){
 q->i = A->i; q->j = A->j;
 q->info = A->info + B->info;
 A = A->leg; B = B->leg;
 } else
 {
 q->i = B->i; q->j = B->j; q->info = B->info;
 B = B->leg;
 };
 }
}

```

```

 if(C == NULL){ C = q; man = q; }
 else{
 man->leg = q;
 man = q;
 }
}
if(A) { if(man) man->leg = A; else C = A; }
if(B) { if(man) man->leg = B; else C = B; }
return C;
}

void Sterge(MRara* matrice){
 if(matrice != NULL){
 Sterge(matrice->leg);
 free(matrice);
 }
}

void main(){
 short n;
 MRara *A, *B, *C;
 printf("n = "); scanf("%d", &n);
 printf("\nIntroduceti prima matrice:");
 CitesteMatrice(&A, n);
 printf("\nIntroduceti a doua matrice:");
 CitesteMatrice(&B, n);
 C = SumaMatrici(A, B);
 printf("\nRFZULTATE:\n");
 printf("Matricea A: "); ScrieMatrice(A, n);
 printf("Matricea B: "); ScrieMatrice(B, n);
 printf("\nMatricea suma rezultata:\n");
 ScrieMatrice(C, n);
 Sterge(A); Sterge(B); Sterge(C);
 getch();
}

```

### Exerciții

- Modificați programul astfel încât afișarea să se facă în formă de listă, de exemplu pentru matricea A să se afișeze doar:  
 $(1, 2, 71), (2, 3, -9), (4, 4, 35)$
- Modificați programul astfel încât dimensiunea matricei să nu fie neapărat pătratică, iar fiecare matrice să aibă și un nume (A, B, C, M, N, T) care se introduce la citire și se utilizează la afișare.

3. În cadrul funcției SumaMatrici() folosim de două ori o secvență asemănătoare pentru copierea informațiilor din A, respectiv B, în q:

```
q->i = A->i; q->j=A->j; q->info = A->info;
```

Scrieți o funcție care realizează acest lucru și folosiți-o în program.

4. Modificați programul astfel încât să se citească mai multe seturi de date din fișier și să se afișeze rezultatele în alt fișier.  
 5. Folosind această structură, să se implementeze și alte operații cu matrice: diferență, produsul, inversa. Să se implementeze o funcție pentru calculul determinantului.  
 6. Scrieți o funcție (cu parametru o astfel de listă) care calculează procentul numerelor diferite de zero dintr-o matrice rară, a celor strict negative, respectiv strict pozitive.

| Intrare                                                                                                                                                                                                                              | ieșire                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>n = 4 Introduceti matricea:   Numar de elemente diferite de 0: 3 Elementul 1:   Linia: 1   Coloana: 2   Valoare: 71 Elementul 2:   Linia: 4   Coloana: 4   Valoare: 35 Elementul 3:   Linia: 2   Coloana: 3   Valoare: -9</pre> | <pre>REZULTATE: Matricea introdusa:   0  71  0  0   0  0  -9  0   0  0  0  0   0  0  0  35  Elemente diferite de zero: 18.75% Elemente strict negative: 6.25% Elemente strict pozitive: 12.5%</pre> |

7. Scrieți funcții pentru calculul următoarelor valori: suma elementelor de dedesubtul, deasupra și de pe diagonala principală a matricei pătratice.  
 8. Un *polinom rar* este un polinom cu grade foarte mari și cu puțini coeficienți diferiți de zero. Scrieți un program, utilizând liste, care operează cu astfel de polinoame: scriere, citire, adunare, înmulțire, înmulțirea cu polinomul  $x-a$ , valoarea unui polinom într-un punct dat.

### Problema 5. Joc de copii I

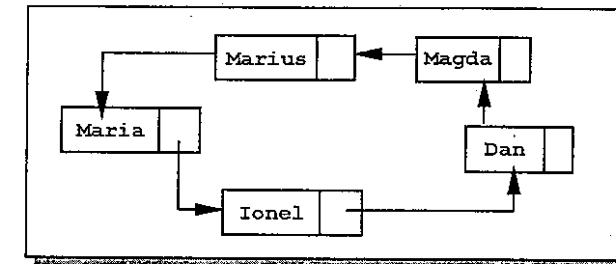
Un număr de copii joacă *de-a v-ați ascunselea* și trebuie să-l aleagă pe cel care stă de pândă. Ei folosesc o poezioară de genul „Din oceanul Pacific/ A ieșit un pește mic/ Și pe coada lui scria/leși afară dumenata!”, pe care o spun pe silabe, iar copilul la care se termină poezia ieșe afară din cerc. Se repetă procedeul până când rămâne doar un copil. Se cere scrierea unui program utilizând o listă circulară simplu înlățuită și care să afișeze pe rând copilul ce iese. Poezia are mai mult de o silabă.

Exemplu :

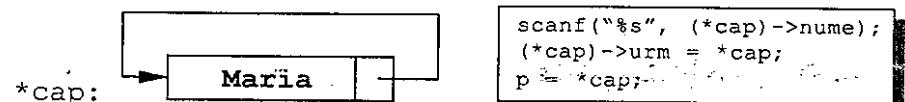
| intrare(tastatură)                                                                                                                            | ieșire(ecran)                                                                                                      |
|-----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <pre>Nr copii: 5 Numele primei pers: Maria Nume copil 2: Ionel Nume copil 3: Dan Nume copil 4: Magda Nume copil 5: Marius Nr silabe = 3</pre> | <pre>1 Iese din joc: Dan 2 Iese din joc: Maria 3 Iese din joc: Marius 4 Iese din joc: Ionel A castigat Magda</pre> |

### Analiza problemei și proiectarea soluției

Vom considera o listă simplu înlățuită, în care ultimul element introdus pointează la primul :

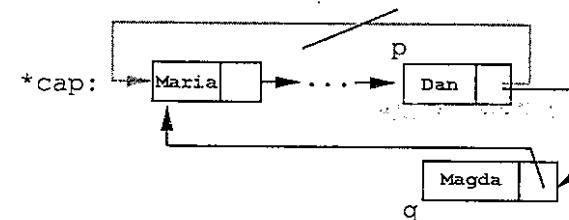


Numele persoanelor se vor citi succesiv și se introduc în listă, păstrând structura de listă liniară simplu înlățuită. La citirea primului nume se va crea lista cap, care are următoarea reprezentare vizuală:



\*cap va rămâne tot timpul primul element introdus și vom folosi variabila auxiliară p care va fi întotdeauna ultimul element, actualizându-se succesiv; în acești prim caz vom avea deci p = \*cap.

Mai departe, introducerea unui nou nume în listă se va face natural: alocăm memorie unei noi variabile q de tip pers\*, completăm q->nume cu valoarea citită de la tastatură, succesorul acestuia este \*cap (primul element), predecesorul acestuia este p și, când q este complet personalizat, decidem că acesta este ultimul element curent al listei (p = q):



adică alocarea memoriei pentru nodul q, completarea câmpurilor nume și succesor ale acestuia, stabilirea sa ca succesor al lui p și reactualizarea lui p cu q:

```
q = (pers*)malloc(sizeof(pers));
scanf("%s", q->nume); q->urm = *cap;
p->urm = q; p = q;
```

În cadrul programului principal se va executa o buclă while(){} cu condiția de continuare cap!=cap->urm, adică să fie mai mult de un element în listă. La fiecare numărare, ne deplasăm de k-1 ori la dreapta și eliminăm succesorul lui cap din listă, cu eliberarea memoriei și reactualizarea elementului curent cap:

```
for (i=0;i<k-2;i++) cap=cap->urm;
q = cap->urm;
cap->urm = cap->urm->urm;
cap = cap->urm;
free(q);
```

### Program

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>

typedef struct nod{
 char nume[20];
 struct nod *urm;
}pers;

void creare(pers **cap){
 pers *q, *p;
 int i, n;
 printf(" Nr copii: ");
 scanf("%d", &n);
 cap=(pers)malloc(sizeof(pers));
 printf("Numele primei pers: ");
 scanf("%s", (*cap)->nume);
 (*cap)->urm=*cap;
 p=*cap;
 for(i=2;i<=n;i++){
 printf("Nume copil %d: ", i);
 q=(pers*)malloc(sizeof(pers));
 scanf("%s", q->nume);
 q->urm=*cap;
 p->urm=q;
 p=q;
 }
}
```

```
void main(){
 int i, k;
 pers *cap, *q;
 int contor = 0;
 creare (&cap);
 printf("Nr silabe = ");
 scanf("%d", &k);
 while(cap!=cap->urm){
 for(i=0;i<k-2;i++) cap=cap->urm;
 printf("%d Iese din joc: %s\n",
 ++contor, cap->urm->nume);
 q=cap->urm;
 cap->urm=cap->urm->urm;
 cap=cap->urm;
 free(q);
 }
 printf("\nA castigat %s", cap->nume);
 free(cap); getch();
}
```

### Exercițiu

Să se modifice programul de mai sus astfel încât să funcționeze și pentru un număr de silabe egal cu 1.

### Problema 6. Joc de copii II

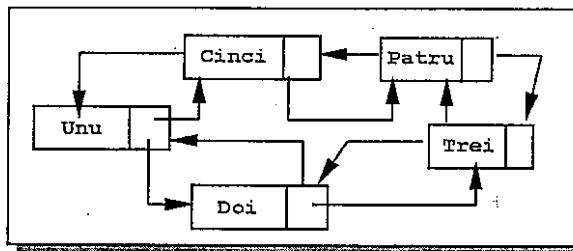
Deoarece copiii s-au plătit să joace mereu același joc, s-au gândit să aplice alt procedeu de eliminare: se va merge succesiv, o dată în sensul acelor de ceasornic, o dată în sens trigonometric și a.m.d. până la rămânerea doar a unui copil în joc. Numărul de silabe este un număr pozitiv care poate fi și 1.

#### Exemplu :

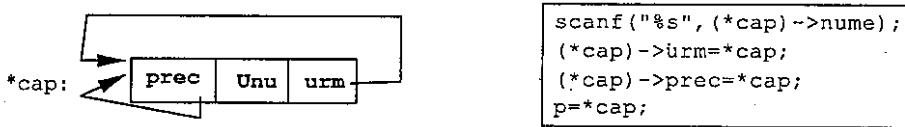
| intrare(tastatură)       | ieșire(ecran)         |
|--------------------------|-----------------------|
| Nr copii: 5              | 1 Iese din joc: Doi   |
| Numele primei pers.: Unu | 2 Iese din joc: Unu   |
| Nume copil 2: Doi        | 3 Iese din joc: Trei  |
| Nume copil 3: Trei       | 4 Iese din joc: Cinci |
| Nume copil 4: Patru      |                       |
| Nume copil 5: Cinci      | A castigat Patru      |
| Nr silabe = 2            |                       |

### Analiza problemei și proiectarea soluției

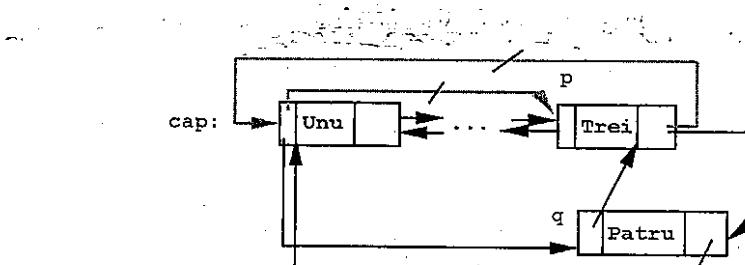
Vom crea o listă dublu înălțuită în care fiecare element conține successorul și predecesorul său. De exemplu, se va crea următoarea listă:



La citirea primei persoane se va crea un singur nod care își va fi propriul predecesor și successor:



Mai departe, introducerea unui nou nume în listă se vă face natural: alocăm memorie unei noi variabile *q* de tip *pers\**, completăm *q->nume* cu valoarea citită de la tastatură, successorul acestuia este *\*cap* (primul element), predecesoul acestuia este *p*, predecesorul lui *\*cap* este *q*, successorul lui *p* este *q* și, când *q* este complet personalizat, decidem că acesta este ultimul element curent al listei (*p = q*):



```

q = (pers*) malloc(sizeof(pers));
scanf("%s", q->nume);
q->urm = *cap;
q->prec = p;
(*cap)->prec = q;
p->urm = q;
p = q;

```

În cadrul programului principal se va executa o buclă `while()` cu condiția de continuare `cap!=cap->urm`, adică să fie mai mult de un element în listă. La fiecare numărare, ne deplasăm de *k-1* ori la dreapta și eliminăm succesorul lui *cap* din listă, cu eliberarea memoriei și reactualizarea elementului curent *cap*.

Pentru simularea sensului de numărare folosim o variabilă *sens* care poate fi -1 sau 1 (sens înainte sau înapoi) care se modifică corespunzător succesiv după eliminarea unei persoane din listă.

### Program

```

#include <stdio.h>
#include <malloc.h>
#include <conio.h>

typedef struct nod
{
 char nume[20];
 struct nod *prec, *urm;
}pers;

void creare(pers **cap)
{
 pers *q, *p;
 int i, n;
 printf(" Nr copii: ");
 scanf("%d", &n);
 cap=(pers)malloc(sizeof(pers));
 printf("Numele primei pers.: ");
 scanf("%s", (*cap)->nume);
 (*cap)->urm = *cap;
 (*cap)->prec = *cap;
 p = *cap;
 for(i=2;i<=n;i++)
 {
 printf("Nume copil %d: ", i);
 q=(pers*)malloc(sizeof(pers));
 scanf("%s", q->nume);
 q->urm = *cap;
 q->prec = p;
 (*cap)->prec = q;
 p->urm = q;
 p = q;
 }
}

```

```

void ScrieLista(pers* l) {
 pers *aux = l;
 do{
 printf(" %s ", l->nume);
 l = l->urm;
 } while(l != aux);
}

void main()
{
 int i, k, sens = 1;
 pers *cap, *q;
 int contor = 0;
 creare(&cap);
 printf("Nr silabe = ");
 scanf("%d", &k);

 while (cap!=cap->urm)
 {
 for(i=0; i<k-1; i++){
 if(sens+1) cap = cap->urm;
 else cap = cap->prec;
 }
 printf("%diese din joc: %s\n",
 ++contor, cap->nume);
 q=cap;
 cap->prec->urm = cap->urm;
 cap->urm->prec = cap->prec;
 if(sens+1){
 cap=cap->urm;
 }else{
 cap=cap->prec;
 }
 free(q);
 sens *= -1;
 }
 printf("\nA castigat %s", cap->nume);
 free(cap); getch();
}

```

**Exercițiu**

Modificați programul astfel încât la eliminarea unui copil din joc se schimbă și numărul de silabe.

Dacă sensul este 1, ne deplasăm înainte, altfel ne deplasăm înapoi.

refacerea legăturilor

eliberarea memoriei

**Problema 7. Tabela de dispersie (Hash Table)**

În fișierul hashtable.in se găsesc numere naturale care se incadrează în tipul int. Scrieți o structură de tip HashTable care să memoreze aceste numere în liste, fiecare listă conținând numere diferite în ordine crescătoare care au același semn și aceeași ultimă cifră. Scrieți funcții care creează o astfel de structură și o afișează, folosind tipurile de date:

|                            |                                |
|----------------------------|--------------------------------|
| <b>typedef struct nod{</b> | <b>typedef struct nodhash{</b> |
| int info;                  | TLista *lista;                 |
| struct nod *leg;           | struct nodhash *leg;           |
| <b>} TLista;</b>           | <b>) THashTable;</b>           |

Exemplu:

| intrare (hashtable.in)      | ieșire (tastatură)      |
|-----------------------------|-------------------------|
| 456 78                      | -9: -879 -789 -459      |
| 9                           | -7: -67                 |
| 90                          | -6: -56                 |
| 0 45 67 8764 -879           | -2: -72 -42 -12         |
| 546 65876                   | -1: -11                 |
| 435                         | 0: 0 90                 |
| 324                         | 2: 32                   |
| 435 5675 -72 -42            | 3: 3543                 |
| 5765 45                     | 4: 234 324 8764 23534   |
| -12 -459                    | 5: 45 435 545 5675 5765 |
| -56 -67                     | 6: 56 456 546 65876     |
| -11 -67 56 32 -56 45 9      | 7: 67                   |
| -789 545 23534 234 3543 435 | 8: 78                   |
| 456                         | 9: 9                    |

**Analiza problemei și proiectarea soluției**

Funcția `TLista *nodLista(int info, TLista *leg)` returnează un element de tip `TLista*` cu informația și succesorul date ca parametri. Va fi utilizată în cadrul funcției de creare a unei liste.

Funcția `THashTable *nodHashTable(TLista *lista, THashTable *leg)` returnează un element de tip `THashTable*` și va fi utilizată la crearea unei structuri de tip HashTable.

În cadrul funcției `insereazaInLista(TLista **cap, int nr)` trebuie să inserăm un nou nod cu informația dată nr, astfel încât să păstrăm condiția de ordonare.

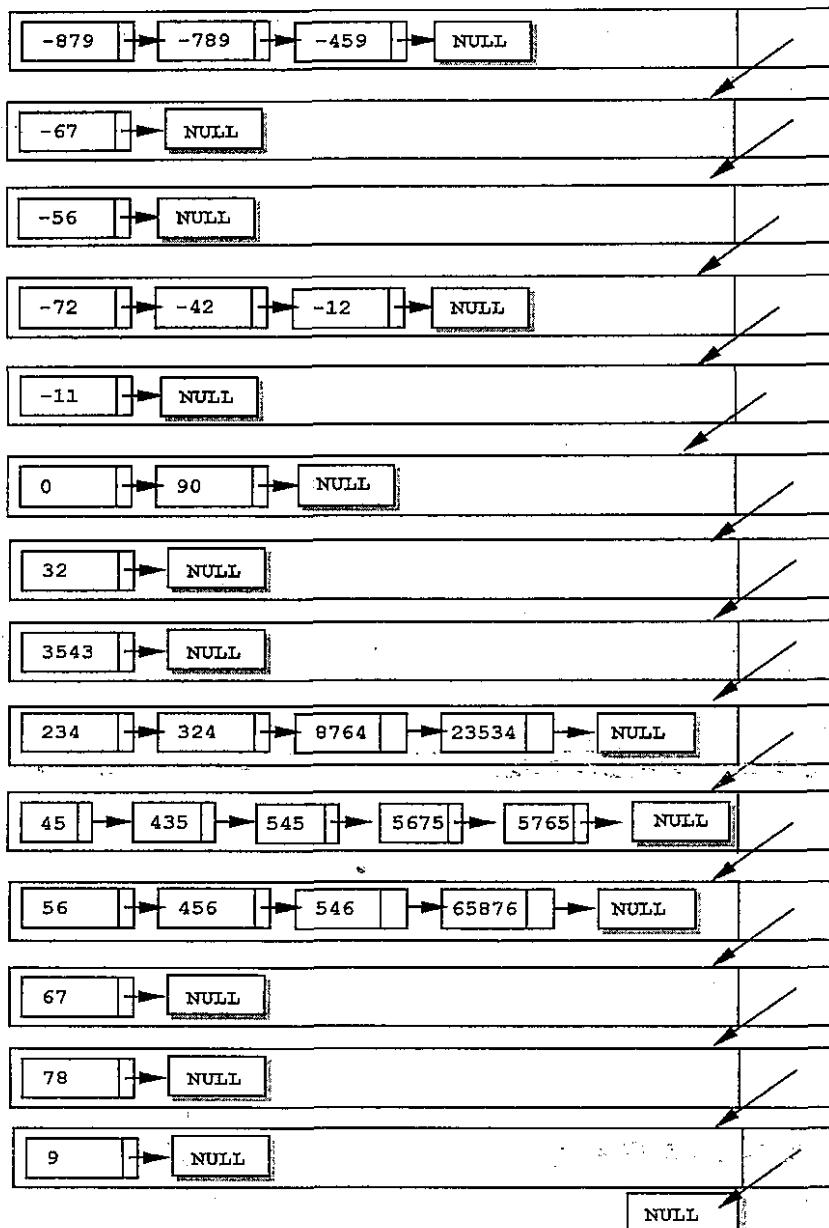
Analizați explicațiile aflate în casetele din dreptul funcției:

`insereazaInLista(TLista **cap, int nr)`

Funcția `insereazaInHashTable(THashTable **t, int nr)` va insera un număr într-o tabelă Hash, în lista corespunzătoare, care are proprietatea că elementele sale au aceeași ultimă cifră și același semn. Dacă această listă nu există (nu s-a inserat

până acum nici un număr cu aceste proprietăți), atunci ea se va crea și se va insera corespunzător. Analizați implementarea funcției.

Pentru setul de date din exemplu se va crea următoarea structură:



### Program

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>

typedef struct nod{
 int info;
 struct nod *leg;
} TLista;

typedef struct nodhash{
 TLista *lista;
 struct nodhash *leg;
} THashTable;

TLista *nodLista(int info, TLista *leg){
 TLista *nou = (TLista*) malloc(sizeof(TLista));
 nou->info = info;
 nou->leg = leg;
 return nou;
}

THashTable *nodHashTable(TLista *lista, THashTable *leg){
 THashTable *nou =
 (THashTable*) malloc(sizeof(THashTable));
 nou->lista = lista;
 nou->leg = leg;
 return nou;
}

void insereazaInLista(TLista **cap, int nr){
 TLista *q, *p;
 p = *cap;
 if(p == NULL){
 q = nodLista(nr, NULL);
 *cap = q; return;
 }

 if(nr < p->info){
 q = nodLista(nr, *cap);
 *cap = q; return;
 }
}
```

Dacă lista \*cap este vidă, se va crea un nod cu legătura NULL și informația nr, acesta fiind și returnat.

Dacă lista \*cap este vidă, se va crea un nod cu legătura NULL și informația nr, acesta fiind și returnat.

```

if(nr != p->info){
 p = *cap;
 while(p->leg&&p->leg->info<nr)
 {
 p = p->leg;
 }
 if(!p->leg){
 q = nodLista(nr, NULL);
 p->leg = q;
 } else
 if(p->leg->info > nr) {
 q = nodLista(nr, p->leg);
 p->leg = q;
 }
 }
}

void insereazaInHashTable(THashTable **t, int nr){

 THashTable *t2, *aux;
 aux = *t;
 if(aux == NULL){
 TLista* q = nodLista(nr, NULL);
 *t = nodHashTable(q, NULL);return;
 }
 if(aux->lista->info%10 > nr%10){
 TLista *q = nodLista(nr, NULL);
 aux = nodHashTable(q, *t);
 *t = aux; return;
 }
 if(aux->lista->info%10 == nr%10){
 insereazaInLista(&aux->lista, nr);
 return;
 }
}

while(aux->leg && aux->leg->lista->info%10 < nr%10)
 aux = aux->leg;
if(!(aux->leg)){
 TLista *q = nodLista(nr, NULL);
 t2 = nodHashTable(q, NULL);
 aux->leg = t2;
}

```

Dacă nr este diferit de informația din p, atunci va trebui adăugat în listă undeva după p, se va parcurge lista până se găsește o informație mai mare decât nr și îl introducem în listă prin crearea unui nou nod q, care se adaugă în poziția corespunzătoare (la capăt sau intermediar).

tabela goală

dacă nr%10 este mai mic decât cheia primei liste deja introduse, se va crea o nouă listă și se va introduce la început!

dacă nr trebuie introdus în prima listă existentă

căutăm poziția unde trebuie introdus nr; dacă lista corespunzătoare există, atunci se adaugă la aceasta, dacă nu, se va crea o nouă listă și se va adăuga în poziția corespunzătoare

```

} else {
 if(aux->leg->lista->info%10 == nr%10){
 insereazaInLista(&aux->leg->lista, nr);
 } else{
 TLista *q = nodLista(nr, NULL);
 t2 = nodHashTable(q, aux->leg);
 aux->leg = t2;
 }
}

void scrieLista(TLista *cap){
 while(cap != NULL){
 printf("%6d", cap->info);
 cap = cap->leg;
 }
}

void scrieHashTable(THashTable* t){
 while(t){
 TLista *lista = t->lista;
 printf("\n%2d: ", lista->info%10);
 scrieLista(lista);
 t = t->leg;
 }
}

void main(){
 THashTable *table = NULL;
 TLista *l1 = NULL;
 int n;
 FILE * fIn = NULL;
 fIn = fopen("hashtable.in", "r");
 while(fIn && !feof(fIn)){
 fscanf(fIn, "%d", &n);
 insereazaInHashTable(&table, n);
 }
 scrieHashTable(table);
 fclose(fIn);
 getch();
}

```

**Exerciții**

- Modificați programul de mai sus astfel încât să fie introduse în aceeași listă toate numerele care au ultima cifră identică, de exemplu -567 va fi introdus în aceeași listă cu 37.
- Scripti funcții de eliberare a memoriei folosite, atât pentru o listă, cât și pentru Hash Table:

stergeLista(TLista\*) și stergeHashTable(THashTable \*cap).

**Problema 8. Crearea listelor**

Să se implementeze următoarele funcții relativ la liste simplu înlănuite: crearea (două variante, una în care elementele sunt introduse în ordine directă, la sfârșitul listei, cea de a două în care elementele sunt introduse în față), crearea unei liste care păstrează doar k elemente din lista-parametru, afișarea, ștergerea din memorie. Scripti un program test pentru funcțiile voastre.

*Exemplu :*

| intrare (tastatură)             | ieșire (ecran)                  |
|---------------------------------|---------------------------------|
| CREARE1...                      | <u>REZULTATE</u>                |
| Introduceti un element? (d/n)d  | Prima lista: 67 2 34            |
| Dati elementul: 34              | A doua lista: 1 2 9 4           |
| Introduceti un element? (d/n)d  | CONCATENARE...                  |
| Dati elementul: 2               | Lista obtinuta: 67 2 34 1 2 9 4 |
| Introduceti un element? (d/n)d  | ELIMINARE...                    |
| Dati elementul: 67              | Lista obtinuta: 67 2 34 1 2     |
| Introduceti un element? (d/n)n  |                                 |
| CREARE2...                      |                                 |
| Introduceti un element (d/n)? d |                                 |
| Informatia: 1                   |                                 |
| Introduceti un element (d/n)? d |                                 |
| Informatia: 2                   |                                 |
| Introduceti un element (d/n)? d |                                 |
| Informatia: 9                   |                                 |
| Introduceti un element (d/n)? d |                                 |
| Informatia: 4                   |                                 |
| Introduceti un element (d/n)? n |                                 |
| Nr. de elem. care raman = 5     |                                 |

**Analiza problemei și proiectarea soluției**

În cadrul primei modalități de creare (funcția creare1()) vom adăuga elementul curent în față.

În cadrul celei de a doua metode (funcția creare2()), elementul curent se va adăuga la sfârșit, se va reține mereu ultimul element în variabila q.

Pentru eliberarea memoriei ocupate de o listă se va scrie funcția recursivă sterge(). Funcția cu antetul:

```
void elimin_k(TLista **cap, int k)
```

execută ștergerea elementelor de la coada listei astfel încât doar primele k elemente sunt păstrate.

**Program**

```
#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <conio.h>

typedef struct lista
{
 int inf;
 struct lista* leg;
} TLista;

/** in ordine inversa **/
void creare1(TLista** cap)
{
 TLista *q;
 short ok;
 char ch;
 *cap = NULL;
 printf("\nCREARE1...\n");
 do
 {
 printf(" Introduceti un element? (d/n)");
 ch=getchar();
 ch=getchar();
 if(tolower(ch) !='n')
 {
 q = (TLista*)malloc(sizeof(TLista));
 printf(" Dati elementul: ");
 scanf("%d",&q->inf);
 q->leg = *cap;
 *cap = q;
 ok=1;
 }
 } while(ok);
}
```

```

 else ok=0;
 }while(ok);
}

/** in ordine directă */
void creare2(TLista** cap)
{
 TLista *q, *p;
 unsigned firstElem=0, ok;
 char ch;
 *cap=NULL;
 q=*cap;
 printf("\nCREARE2...\n");
 do
 {
 printf(" Introduceti un element(d/n)? ");
 scanf("%c",&ch);
 scanf("%c",&ch);
 if(tolower(ch)!='n')
 {
 p = (TLista*)malloc(sizeof(TLista));
 printf(" Informatia: ");
 scanf("%d",&p->inf);
 p->leg=NULL;
 if(firstElem==0){*cap=p;q=p;}
 else
 {
 q->leg=p;
 q=p;
 }
 ok=1;
 firstElem++;
 }
 else ok=0;
 }while(ok);
}

/** concatenarea naturală
 a două liste; rezultatul în cap1 */
void concat(TLista* cap1, TLista* cap2)
{
 TLista *q;
 if(cap1)

```

```

 q=cap1;
 while(q->leg)q=q->leg;
 q->leg=cap2;
 }
 else cap1=cap2;
}

/** afisarea unei liste */
void afisare(TLista *cap)
{
 while(cap)
 {
 printf(" %d", cap->inf);
 cap=cap->leg;
 }
}

/** pastrarea primelor k elemente cu
 eliberarea memoriei elementelor sterse*/
void elimin_k(TLista **cap, int k)
{
 TLista *kp, *q;
 kp = *cap;
 /* ma deplasez k-1 elemente la dreapta */
 while(kp&&k-1){kp=kp->leg;k--;}
 q=kp->leg;
 kp->leg=NULL;
 while(q)
 {
 kp=q;
 q=q->leg;
 free(kp);
 }
}

/** eliberarea recursiva a memoriei utilizate de liste*/
void sterge(TLista* l)
{
 if(l != NULL){
 sterge(l->leg);
 free(l);
 }
}

```

Dacă prima listă este nevidă, ne poziționăm pe ultimul său element și îi stabilim ca legătură cea de a două listă, altfel primei liste îi se atribuie a două.

Ne poziționăm pe al  $k+1$ -lea element în listă și executăm eliberarea memoriei de după acesta.

```

void main()
{
 TLista* cap1, *cap2;
 int k;
 creare1(&cap1); creare2(&cap2);
 printf("\nNr. de elem. care raman = ");
 scanf("%d",&k);
 printf("\n _____REZULTATE_____ \n");
 printf("\nPrima lista: ") ;afisare(cap1);
 printf("\nA doua lista: ") ;afisare(cap2);
 printf("\n\nCONCATENARE... \n");
 concat(cap1, cap2);
 printf("Lista obtinuta: ") ;afisare(cap1);
 printf("\n\nELIMINARE... ");
 elimin_k(&cap1, k);
 printf("\nLista obtinuta: ") ;afisare(cap1);
 sterge(cap1);
 getch(); getch();
}

```

### Exercițiu

Scrieți funcții de inversare, ștergere a unui element dat, numărarea elementelor dintr-o listă.

### Problema 9. Stiva (Stack)

*Stiva este un exemplu simplu, dar important, de tip abstract de date. Este folosit pentru a stoca elementele (de obicei de același tip) după principiul Last In – First Out (LIFO – ultimul element introdus este primul element extras). Ca o analogie, putem să ne imaginăm un vatră de farfurii: numai ultima poate fi luată și o nouă farfurie va fi amplasată în vârful stivei. Operațiile elementare care pot fi efectuate cu o stivă sunt:*

| Operație                       | Semnificație                                                      |
|--------------------------------|-------------------------------------------------------------------|
| isEmpty:stack -> boolean       | Testează dacă stiva este vidă.                                    |
| top:stack ->Element_Type       | Returnează elementul din vârful stivei și îl elimină din aceasta. |
| pop:stack->stack               | Elimină elementul din vârful stivei, modificând-o în acest sens.  |
| push:Element_Type×stack->stack | Adaugă un element dat la stivă.                                   |

Scrieți un program, folosind structuri cu autoreferire, care implementează și testează aceste operații. Elementele introduse în stivă trebuie să fie de tip struct, ce conține două câmpuri: nume (șir cu maximum 10 caractere) și cod (intreg int).

### Analiza problemei și proiectarea soluției

Vom defini tipul Stack, ca o structură autoreferită:

```

typedef struct stiva{
 Element el;
 struct stiva *leg;
}Stack;

```

și operațiile ce operează cu aceasta: creare (create), afișare (write), top, pop și push. Un exemplu de rulare a programului:

Cate elemente se introduc? n=3

Elementul 1:

Nume = Ion

Cod = 12

Elementul 2:

Nume = Maria

Cod = 31

Elementul 3:

Nume = Gigi

Cod = 89

Stiva introdusa:

(Gigi, 89) (Maria, 31) (Ion, 12)

Introdu nou element:

Nume: Moni

Cod: 76

Stiva după push: (Moni, 76) (Gigi, 89) (Maria, 31) (Ion, 12)

Stiva după pop: (Gigi, 89) (Maria, 31) (Ion, 12)

Stiva după pop: (Maria, 31) (Ion, 12)

Apel top(): Element: (Maria, 31)

Stiva: (Ion, 12)

Programul următor este reprezentativ pentru implementarea operațiilor specifice stivei.

### Program

```

#include <stdio.h>

typedef struct{
 char nume[10];
 int cod;
} Element;

```

```

typedef struct stiva{
 Element el;
 struct stiva *leg;
}Stack;

void create(Stack **stack){
 Stack *q;
 Element e;
 int n, i;
 *stack = NULL;
 printf("Cate elemente se introduc? n=");
 scanf("%d", &n);
 for(i=0; i<n; i++){
 q = (Stack*) malloc(sizeof(Stack));
 printf("\nElementul %d:\n", i+1);
 printf("Nume = ");
 scanf("%s", q->el.nume);
 printf("Cod = "); scanf("%d", &q->el.cod);
 q->leg = *stack;
 *stack = q;
 }
}

void write(Stack *stack){
 while(stack){
 printf("(%, %d) ", stack->el.nume, stack->el.cod);
 stack = stack->leg;
 }
}

int isEmpty(Stack* stack){
 if(stack==NULL) return 1;
 else return 0;
}

Element top(Stack **stack){
 Element e = (*stack)->el;
 Stack *q = *stack;
 (*stack) = (*stack)->leg;
 free(q);
 return e;
}

```

```

void pop(Stack **stack){
 Stack *q = *stack;
 if(isEmpty(*stack)) return;
 *stack = (*stack)->leg;
 free(q);
}

void push(Element e, Stack **stack){
 Stack *q = (Stack*) malloc(sizeof(Stack));
 q->el = e;
 q->leg = *stack;
 *stack = q;
}

void main(){
 Stack* stack;
 Element el;
 create(&stack);
 printf("Stiva introdusa: \n");
 write(stack);
 printf("\nIntrodu nou element: ");
 printf("\nNume: "); scanf("%s", el.nume);
 printf("Cod: "); scanf("%d", &el.cod);
 push(el, &stack);
 printf("\nStiva dupa push: ");
 write(stack);
 pop(&stack);
 printf("\nStiva dupa pop: ");
 write(stack);
 pop(&stack);
 printf("\nStiva dupa pop: ");
 write(stack);
 el = top(&stack);
 printf("\nApel top(): ");
 printf("Element: (%s, %d)", el.nume, el.cod);
 printf("\nStiva: "); write(stack);
}

```

### Exercitii

1. Implementați și metoda size(), care returnează numărul de elemente din stivă.
2. Implementați operațiile caracteristice stivei folosind tablouri unidimensionale.
3. *Coada (Queue).* Un alt exemplu important de tip abstract de date este coada. Este folosit pentru a stoca elemente (de obicei, de același tip) după principiul First In – First Out (FIFO – primul element introdus este ultimul element extras). Prin analogie,

ne putem imagina o coadă la ghișeu la bancă : primul venit – primul servit. Operațiile elementare care pot fi efectuate cu o coadă sunt :

| Operație                                 | Semnificație                                                          |
|------------------------------------------|-----------------------------------------------------------------------|
| isEmpty:queue -> boolean                 | Testează dacă coada este vidă.                                        |
| size:queue -> int                        | Returnează numărul de elemente din coadă.                             |
| front:queue-> Element_Type               | Returnează (dar nu sterge) primul element introdus (cel mai „vechi”). |
| back:queue-> Element_Type                | Returnează (dar nu sterge) ultimul element introdus (cel mai „nou”).  |
| Insert:<br>Element_Type < queue -> queue | Adaugă un element la coadă.                                           |
| pop:queue-> queue                        | Șterge (dar nu returnează) primul element introdus (cel mai „vechi”). |

Să se implementeze toate operațiile folosind structuri simplu înlățuite, ca în cazul stivei prezentate mai sus.

4. Să se implementeze conceptul de coadă cu toate operațiile prezentate, folosind tablouri unidimensionale.

### Problema 10. Cărți de joc

Scrieți un program care împarte cărți de joc dintr-un pachet clasic (54 de cărți, culorile treflă, caro, cupă, pică + 2 Jokeri). Se vor introduce de la tastatură numărul de jucători și numărul de cărți care trebuie împărțite. Dacă nu este posibil, se va afișa un mesaj corespunzător. Scrieți o funcție care „ameșteacă” pachetul dat.

Exemplu :

| intrare<br>(tastatură) | ieșire (ecran)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Numar jucatori:<br>5   | --REZULTATE--<br>Pachetul la inceput: 53 52 51 50 49 48 47 46 45 44<br>43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27<br>26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9<br>8 7 6 5 4 3 2 1 0<br>Pachetul amestecat:<br>12 48 2 8 0 23 24 41 21 44 20 7 35 40 5 14 50 37 47<br>34 45 46 29 6 39 52 27 15 4 9 42 43 22 36 28 19 51<br>17 16 26 18 32 53 11 33 9 25 3 30 31 4 10 38 1 13<br>Jucatorul nr. 1<br>POPA TREFLA   AS PICA   4 TREFLA   10 TREFLA<br>  2 TREFLA   VALET CARO   DAMA CARO   4 PICA  <br>Jucatorul nr. 2<br>10 CARO   7 PICA   9 CARO   9 TREFLA   AS CUPA<br>  3 PICA   7 TREFLA   3 CARO |

| intrare<br>(tastatură) | ieșire (ecran)                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        | Jucatorul nr. 3<br>DAMA PICA   DAMA CUPA   10 PICA   10 CUPA   8<br>PICA   9 PICA   5 CUPA   8 TREFLA                                                                                                                                                                                                                                                                                                                                                     |
|                        | Jucatorul nr. 4<br>2 PICA   JOCKER   3 CUPA   4 CARO   VALET PICA<br>  5 PICA   6 PICA   AS CARO                                                                                                                                                                                                                                                                                                                                                          |
|                        | Jucatorul nr. 5<br>VALET CUPA   4 CUPA   8 CARO   POPA PICA   6<br>CARO   5 CARO   2 CUPA   7 CARO                                                                                                                                                                                                                                                                                                                                                        |
|                        | Pachetul ramas: 32 53 11 33 9 25 3 30 31 4 10 38 1<br>13                                                                                                                                                                                                                                                                                                                                                                                                  |
| Numar jucatori:<br>8   | --REZULTATE--<br>Pachetul la inceput: 53 52 51 50 49 48 47 46 45 44<br>43 42 41 40 39 38 37 36 35<br>34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18<br>17 16 15 14 13 12 11 10 9 8 7<br>6 5 4 3 2 1 0<br>Pachetul amestecat:<br>12 48 2 8 0 23 24 41 21 44 20 7 35 40 5 14 50 37 47<br>34 45 46 29 6 39 52 27 15 4<br>9 42 43 22 36 28 19 51 17 16 26 18 32 53 11 33 9 25<br>3 30 31 4 10 38 1 13<br>Prea multi jucatori sau prea multe carti cerute! |
| Numar carti: 12        | Pachetul ramas: 12 48 2 8 0 23 24 41 21 44 20 7 35<br>40 5 14 50 37 47 34 45 46 2<br>9 6 39 52 27 15 49 42 43 22 36 28 19 51 17-16 26 18<br>32 53 11 33 9 25 3 30 31 4 1<br>0 38 1 13                                                                                                                                                                                                                                                                     |

### Analiza problemei și proiectarea soluției

Cărțile de joc se vor codifica prin numere de la 0 la 53, primele 13 cărți fiind treflele (2, 3, ..., 10, As, Valet, Dama, Popa), următoarele 13 sunt carourile (2, 3, ..., Popa), cupele și picile. Ultimele două cărți (52-53) sunt Jokerii. Deci vom putea considera pachetul ca fiind mulțimea numerelor de la 0 la 53. Considerând cartea i, vom decide culoarea ca fiind i/13 (0 → treflă, 1 → caro, 2 → cupă, 3 → pică, altfel → Joker) și numărul de pe carte ca fiind determinat de i%13 (0 → 2, 1 → 3, ..., 8 → 10, 9 → As, 10 → Valet, 11 → Dama, 12 → Popa). Vom defini structura TPachet care va conține numere de la 0 la 53.

Funcția cu antetul:

```
void ad_la_pachet(TPachet** cap, int nr, short *n)
```

adaugă o carte cu numărul nr la pachet, modificând pointerul \*cap și numărul de cărți din pachet n.

Funcția void scrie\_pachet(TPachet\* cap) afișează numerele din pachet.  
Funcția cu antetul:

```
void schimba_val(TPachet **cap, int i, int j)
```

va interchimba valorile elementelor din lista referită de cap ; de pe pozițiile i și j, ne vom poziționa cu pointerii p și q pe elementele i și j și interschimbăm valorile acestora :

```
if(q && p!=q)
{
 aux = p->k;
 p->k=q->k;
 q->k=aux;
}
```

Funcția :

```
void amesteca_pachet(TPachet **cap, short n)
```

va interschimba de un număr aleatoriu de ori valori aleatorii din pachet :

```
int m = rand();
long int i;
long int i1, i2;
for(i=0;i<m;i++)
{
 i1 = rand() % n;
 i2 = rand() % n;
 if(i1==i2)schimba_val(cap,i1,i2);
}
```

Funcția :

```
void transf_nr_carte(short nr, char numar[6], char culoare[7])
```

determină culoarea și numărul cărții codificate cu nr, folosind algoritmul descris mai sus.

Funcția :

```
short ultima_carte(TPachet **cap, short *n)
```

va returna ultima carte din pachet cu eliminarea acesteia prin eliberarea memoriei, stergerea din listă și decrementarea numărului de cărți.

### Program

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <stdlib.h>

typedef struct pachet
{
 short k;
 struct pachet* next;
}TPachet;
```

```
void ad_la_pachet(TPachet** cap, int nr, short *n)
{
 TPachet *q;
 q = (TPachet*)malloc(sizeof(TPachet));
 q->k=nr;
 q->next=*cap;
 *cap=q;
 (*n)++;
}
```

se adaugă cartea cu numărul nr în  
față și se incrementează numărul total  
de cărți din pachet

```
void scrie_pachet(TPachet* cap)
{
 if(!cap) printf("Pachetul gol!");
 while(cap)
 {
 printf(" %d", cap->k);
 cap=cap->next;
 }
}
```

```
void schimba_val(TPachet **cap, int i, int j)
{
 TPachet *p, *q;
 short aux;
 int min, max;
 p=*cap;
 min=i<j?i:j;
 max=i+j-min;
 while(p && min){p=p->next; min--; max--;}
 if(p)
```

ne poziționăm pe elementul al  
i-lea și pe elementul al j-lea  
în listă (p și q) și interschimbăm  
informațiile din acestea

```

 {
 q=p;
 while(q && max) {q=q->next; max--};
 if(q && p!=q)
 {
 aux = p->k;
 p->k=q->k;
 q->k=aux;
 }
 }

void amesteca_pachet(TPachet **cap, short n)
{
 int m = rand();
 long int i;
 long int i1, i2;
 for(i=0;i<m;i++)
 {
 i1 = rand() % n;
 i2 = rand() % n;
 if(i1>i2)schimba_val(cap,i1,i2);
 }
}

void transf_nr_carte(short nr, char numar[6], char culoare[7])
{
 short i = nr;
 if(i<52)
 {
 i=nr % 13;
 if(i>0&&i<=8)printf(" %d",i+2);
 else
 {
 switch(i)
 {
 case 9: numar="AS";break;
 case 10: numar="VALET";break;
 case 11: numar="DAMA";break;
 case 12: numar="POPA";
 }
 printf("%s",numar);
 }
 }
}

```

interschimbarea a câte două cărți de un număr aleatoriu de ori

0.. 12: 2, 3, ..., 10, A, V, D, K  
Treflă  
13.. 25: 2, ..., 10, A, V, D, K  
Caro  
26.. 38: 2, ..., 10, A, V, D, K  
Cupă  
39.. 51: 2, ..., 10, A, V, D, K  
Pică  
52, 53: Jokeri  
  
nr/13 este culoarea cărții (0 → Treflă, 1 → Caro, 2 → Cupă, 3 → Pică)  
  
(nr%13 + 2) este valoarea cărții (11 → As, 12 → Valet, 13 → Damă, 14 → Popă)

```

i = nr / 13;
switch(i)
{
 case 0: culoare = "TREFLA"; break;
 case 1: culoare = "CARO"; break;
 case 2: culoare = "CUPA"; break;
 case 3: culoare = "PICA"; break;
 default:culoare = "JOKER";
}
printf(" %s ",culoare);

short ultima_carte(TPachet **cap, short *n)
{
 if(n)
 {
 short aux = (*cap)->k;
 TPachet *q = (*cap);
 *cap=(*cap)->next;
 free(q);
 (*n)--;
 return aux;
 }
 else
 printf(" Pachet gol!");
 return -1;
}

void main()
{
 TPachet *cap;
 short n=0, i, j;
 char numar[4], culoare[7];
 short nj, nc, nr;
 printf("Numar jucatori: "); scanf("%d",&nj);
 printf("Numar carti: "); scanf("%d",&nc);
 printf("\n\n---REZULTATE--- ");
 cap=NULL;
 for(i=0; i<54; i++)
 ad_la_pachet(&cap, i, &n);
 printf("\n Pachetul la inceput: ");
 scrie_pachet(cap);
 amesteca_pachet(&cap,n);
 printf("\n Pachetul amestecat:\n");
}

```

returnează ultima carte din pachet ștergând-o pe aceasta din listă și decrementând numărul de cărți din pachet

```

scire_pachet(cap);
if(nj*nc<=54)
{
 for(i=0;i<nj; i++)
 {
 printf("\n\n Jucatorul nr. %d\n",i+1);
 for(j=0; j<nc; j++)
 {
 nr = ultima_carte(&cap, &n);
 transf_nr_carte(nr, numar, culoare);
 }
 }
}
else printf(
 "\n Prea multi jucatori sau prea multe carti cerute!");
printf("\n\n Pachetul ramas: "); scire_pachet(cap);
getch();
}

```

### Exerciții

1. Afişați cărțile din mână în ordine „crescătoare”, adică în ordinea culorilor și în ordinea mărimarilor cărților.

*Exemplu:* Pentru jucătorii 1 și 2 se va afișa:

Jucatorul nr. 1  
 2 TREFLA | 4 TREFLA | 10 TREFLA | POPA TREFLA | VALET CARO  
 | DAMA CARO | 4 PICA | AS PICA |

Jucatorul nr. 2  
 7 TREFLA | 9 TREFLA | 3 CARO | 9 CARO | 10 CARO | AS CUPA  
 | 3 PICA | 7 PICA |

și așa mai departe. Jokerul se va afișa la sfârșit.

2. În cazul programului de mai sus, maniera de împărțire după amestecarea pachetului este următoarea: se împart toate cărțile primului jucător, apoi celui de al doilea și.a.m.d., până când toți jucătorii își primesc cărțile. Faceți împărțirea clasică a cărților, câte o carte circular, modificând programul de mai sus.

### Problema II. Parcurgerea arborelui binar

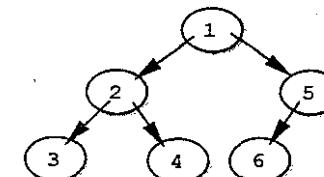
Scrieți un program pentru descrierea operațiilor pentru arbori binari: creare, numărare, căutare, stergere, inserare, parcurgere (inordine, preordine, postordine).

*Exemplu:*

| Intrare (tastatură)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | ieșire (écran)                                                                                                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| informatia: 1<br>Subarborele stang pentru 1? (d/n) informatia: 2<br>Subarborele stang pentru 2? (d/n) informatia: 3<br>Subarborele stang pentru 3? (d/n) Subarborele drept pentru 3? (d/n)<br>Subarborele drept pentru 2? (d/n) informatia: 4<br>Subarborele stang pentru 4? (d/n) Subarborele drept pentru 4? (d/n)<br>Subarborele drept pentru 1? (d/n) informatia: 5<br>Subarborele stang pentru 5? (d/n) informatia: 6<br>Subarborele stang pentru 6? (d/n) Subarborele drept pentru 6? (d/n)<br>Subarborele drept pentru 5? (d/n)<br>Dati un număr: 4 | ---REZULTATE---<br>Parcurgere in preordine: 1 2 3 4 5 6<br>Parcurgere in inordine: 3 2 4 1 6 5<br>Parcurgere in postordine: 3 4 2 6 5 1<br>Numar de elemente in arbore = 6<br>4 este in arbore |

### Analiza problemei și proiectarea soluției

Arborele binar introdus mai sus este:



Un astfel de arbore este complet determinat de rădăcină, subarborele stang și subarborele drept. Folosind această observație, scriem funcțiile recursive. Urmăriți explicațiile din dreptul funcțiilor.

*Program*

```

#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <ctype.h>

struct nod
{
 int inf;
 struct nod *st, *dr;
};

struct nod* creare()
{
 struct nod* p = NULL;
 p=(struct nod*)malloc(sizeof(struct nod));
 printf(" informatie: ");
 scanf("%d",&p->inf);
 printf(" Subarborele stang pentru %d? (d/n)", p->inf);
 if(tolower(getch())=='d')p->st=creare();
 else p->st=NULL;
 printf(" Subarborele drept pentru %d? (d/n)", p->inf);
 if(tolower(getch())=='d')p->dr=creare();
 else (printf("\n");p->dr=NULL);
 return p;
}

void parc_pre(struct nod* p)
{
 if(p!=NULL)
 {
 printf(" %d",p->inf);
 parc_pre(p->st);
 parc_pre(p->dr);
 }
}

void parc_in(struct nod* p)
{
 if(p!=NULL)
 {
 parc_in(p->st);
 printf(" %d",p->inf);
 parc_in(p->dr);
 }
}

```

**crearea recursivă a arborelui : întâi informația curentă, apoi crearea subarborelui stâng, apoi a celui drept**

**parcurea în preordine a arborelui : întâi se exploatează rădăcina, apoi subarborele stâng, apoi cel drept**

**parcurea în inordine a arborelui : întâi se exploatează subarborele stâng, apoi rădăcina, apoi subarborele drept**

```

 parc_in(p->dr);
 }

 void parc_post(struct nod* p)
 {
 if(p!=NULL)
 {
 parc_post(p->st);
 parc_post(p->dr);
 printf(" %d",p->inf);
 }
 }

 int numar_elem(struct nod * p)
 {
 if(p==NULL) return 0;
 else return 1+numar_elem(p->st)+numar_elem(p->dr);
 }

 void stergere(struct nod* p)
 {
 if(p!=NULL)
 {
 stergere(p->st);
 stergere(p->dr);
 free(p);
 }
 }

 int este_k(struct nod* p, int n)
 {
 if(p==NULL) return 0;
 else
 if(p->inf==n) return 1;
 else return
 este_k(p->dr,n)||este_k(p->st,n);
 }

 void main()
 {
 struct nod* rad;
 int k;
 rad=creare();
 }

```

**parcurea în postordine a arborelui : întâi se exploatează subarborele stâng, apoi subarborele drept, iar în final, rădăcina**

**numărul de elemente ale unui arbore este egal cu 1 plus numărul de elemente din subarborele stâng plus numărul de elemente din subarborele drept**

**ștergerea recursivă a arborelui : întâi subarborele stâng, apoi cel drept, apoi se eliberează memoria corespunzătoare rădăcinii**

**căutarea unui element în arbore : dacă arborele este NULL, atunci se va returna valoarea 0; dacă informația din nod este cea căutată, atunci se va returna 1, altfel se va căuta în subarborele stâng și în cel drept**

```

printf(" Dati un numar: ");
scanf("%d",&k);
printf("\n ---REZULTATE---");
printf("\n Parcursere in preordine: "); parc_pre(rad);
printf("\n Parcursere in inordine: "); parc_in(rad);
printf("\n Parcursere in postordine:"); parc_post(rad);
printf("\n Numar de elemente in arbore = %d ",numar_elem(rad));
if(este_k(rad,k))printf("\n %d este in arbore",k);
else printf("\n %d nu este in arbore!", k);
getch();
}

```

### Exercițiu

Scrieți o funcție recursivă care determină suma elementelor dintr-un arbore binar dat.

### Probleme propuse

1. Scrieți un program cu liste simplu înlănuite ce conțin în zonele de informație numere întregi, care să citească și să parcurgă astfel de liste. Scrieți o funcție care să inverseze legăturile într-o astfel de listă simplu înlănuită, specificată ca parametru, fără a utiliza noi zone de memorie alocate dinamic. Funcția va întoarce adresa primului nod din lista inversată.
2. Se citesc de la tastatură mai multe numere reale, ultimul fiind 0. Să se creeze o listă alocată dinamic care să le conțină, iar cu ajutorul ei: să se afișeze numerele în ordinea citirii, să se afișeze numerele în ordinea inversă citirii, să se afișeze numerele în ordine crescătoare (dacă există mai multe numere egale, se vor afișa numai o dată = mulțimea corespunzătoare intrării).
3. Într-o clasă sunt  $n$  elevi,  $n < 50$ . Să se scrie un program care să răspundă următoarelor cerințe:
  - a) Să se creeze o listă liniară simplu înlănuită, ale cărei noduri să conțină în partea de informație numele și nota la purtare ale elevilor clasei.
  - b) Să se afișeze pe ecran conținutul acestei liste astfel încât informațiile relative la cei  $n$  elevi să apară în ordinea în care au fost introduse.
  - c) Folosind lista de la punctul a), să se creeze o nouă listă care să conțină numele elevilor ce au media 10 la purtare, astfel încât, în orice moment al creării, lista să fie ordonată alfabetic după numele elevilor.
  - d) Să se șteargă din lista de la punctul a) primul element, apoi afișați lista.
4. Scrieți un subprogram care inserează între oricare două elemente ale unei liste dublu înlănuite media lor aritmetică.
5. În fișierul Examen.in se găsesc informațiile referitoare la un examen cu două probe. Pe fiecare linie este înregistrată câte o persoană: *nume, nota1, nota2*. Utilizând tipul struct:
 

```

struct Persoana{

```

```

 char *nume;
 float nota1, nota2, media;
}

```

creați fișierul Examen.out cu persoanele admise în ordinea descrescătoare a mediilor (cu mediile mai mari sau egale cu 5) pe primele linii numerotate, apoi persoanele respinse pe următoarele linii (numerotate) în ordine alfabetică.

6. Construiți o listă simplă cu numere reale și scrieți o funcție care elimină elementele negative.
7. De la tastatură se introduc prenumele unor persoane. Creați o listă liniară simplu înlănuită care să le conțină în ordine alfabetică astfel încât în orice moment al adăugării unui cuvânt lista să-și mențină proprietatea de ordonare.
8. Fiind dată o listă dublă cu informații de tip sir de caractere, să se facă ordonarea elementelor listei.
9. Fiind dată o listă dublă, să se separe în două liste elementele prime și neprime.
10. Fiind date două liste simple cu informații de tip întreg, care reprezintă elementele unor mulțimi (verificați la crearea lor proprietatea de mulțime), să se scrie funcții care să execute următoarele operații: intersecția, reuniunea, diferența și diferența simetrică.
11. *Evaluarea unei expresii în notație postfixată*. Într-un fișier este memorată o expresie aritmetică în notăție postfixată, corectă sintactic. Unitățile lexicale (operatori și operanzi) sunt citite din fișier și sunt introduse într-o coadă. Evaluarea expresiei este făcută folosind o stivă (de operanzi), cu algoritmul:

```

atât timp cat coada nu este vida
 extrage un element din coada - E
 daca E este operand atunci
 pune E în stiva
 daca E este operator atunci
 extrage OP2 din stiva
 extrage OP1 din stiva
 calculeaza R = OP1 E OP2
 pune R în stiva

```

Implementați acest algoritm cu liste simplu înlănuite.

12. *Gestiunea proceselor*. Se citește dintr-un fișier, câte una pe linie, o secvență de comenzi cu formatul următor:

**INSERT <nr\_id\_proces> <prioritate>**

sau

**PROCEEDING**

reprezentând operațiile pe care trebuie să le efectueze gestionarul de procese ale unui sistem de operare. Prima operație înseamnă introducerea procesului cu numărul de identificare **<nr\_id\_proces>** într-o coadă de așteptare, ordonată descrescător după **<prioritate>**. A doua operație înseamnă eliminarea primului proces din coadă (cel cu prioritatea cea mai mare) și afișarea identificatorului lui. Să se vizualizeze activitatea gestionarului de procese, folosind o structură cu autoreferire.

## Grafuri, tehnici de programare

Recursivitate, *Divide et impera*, *Greedy*, *Backtracking*, programare dinamică.  
12 probleme complet rezolvate, 38 de exerciții și probleme propuse:

### Grafuri

- Parcurgeri *DFS*, *BFS* ale unui graf
- Matricea drumurilor

### Recursivitate

- Suma cifrelor unui număr natural
- Numărul 4

### *Divide et impera*

- Aflarea celui mai mare divizor comun a n numere naturale
- Turnurile din Hanoi

### *Greedy*

- Problema continuă a rucsacului
- Colorarea hărții

### *Backtracking*

- Colorarea hărții
- Problema fotografiei (*Backtracking* în plan)

### Programare dinamică

- Determinarea combinărilor
- Cuvinte potrivite

*Oamenii normali nu știu că totul e posibil.*

David Rousset

## 1. Grafuri

Matematicianul Grigore C. Moisil afirma: „Azi, teoria grafurilor a devenit o disciplină majoră, deși nu-și găsește locul într-o clasificare dogmatică a capitolelor matematice. Folosirea teoriei grafurilor în domenii variate, de la chimie la economie, de la studiul rețelelor electrice la critica textelor și la politică, îi dă azi un prestigiu de care cel ce clasifică științele trebuie să țină seama”.

**Graf neorientat.** Se numește graf neorientat o pereche ordonată de mulțimi  $(V, E)$ ,  $V$  fiind o mulțime finită și nevidă de elemente numite noduri sau vârfuri, iar  $E$  o mulțime de perechi neordonate din  $X$ , numite muchii.

**Graf parțial.** Un graf parțial al grafului  $G = (V, E)$  este un graf  $H = (V, F)$  astfel încât  $F \subseteq E$ , adică  $H$  are aceeași mulțime de vârfuri ca  $G$ , iar mulțimea de muchii  $F$  este chiar  $E$  sau o submulțime a acesteia.

**Subgraf.** Un subgraf al unui graf  $G = (V, E)$  este un graf  $H = (Y, F)$  astfel încât  $Y \subseteq V$  și  $F$  conține toate muchiile din  $E$  care au ambele extremități în  $Y$ . Vom spune că subgrafa  $H$  este inducă sau generată de mulțimea de vârfuri  $Y$ .

**Multigraf.** Un multigraf este un graf care poate avea mai multe muchii între două vârfuri.

**Cadrul unui vârf  $x$**  este numărul muchiilor incidente cu  $x$ .

**Lanț (ărum) în graf.** Un lanț (ărum) în graf este o succesiune consecutivă de vârfuri de-a lungul unei secvențe de muchii. Dacă vârfurile sunt diferite două câte două, atunci lanțul se numește elementar. Un circuit în graf este un drum cu aceleași vârfuri inițial și final.

**Conexitate.** Un graf se numește conex dacă pentru oricare două vârfuri diferite ale sale, există un lanț care le leagă.

**Ciclu hamiltonian.** Într-un graf  $G = (V, E)$  se numește ciclu hamiltonian un ciclu elementar care conține toate vârfurile grafului. Se numește graf hamiltonian un graf care conține un ciclu hamiltonian.

**Ciclu eulerian.** Într-un graf  $G = (V, E)$  se numește ciclu eulerian un ciclu elementar care conține toate muchiile grafului. Se numește graf eulerian un graf care conține un ciclu eulerian.

**Arbore.** Se numește arbore un graf conex și fără cicluri. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.

**Arbore parțial.** Fie  $G = (V, E)$  un graf. Un graf parțial al său care este și arbore se numește arbore parțial.

### Metode de reprezentare

1. Matricea de adiacență: o matrice pătrată de ordinul  $n$ , în care:

$$A[i, j] = \begin{cases} 1, & \text{dacă } (i, j) \in E \\ 0, & \text{încă} \end{cases}$$

Exemplu :

| Graful | Matricea de adiacență                                                                                                                                                                                                                                                                                    |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | $A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$ |

Din modul de definire al matricei rezultă că ea este simetrică.

2. Se precizează numărul  $n$  de vârfuri și, pentru fiecare vârf  $i$ , lista  $L_i$  a vecinilor săi, adică lista vârfurilor  $j$  pentru care  $(i, j) \in E$ . Pentru graful anterior, aceste liste sunt:

| Vârf $i$ | Lista $L_i$ a vecinilor săi |
|----------|-----------------------------|
| 1        | 2, 3, 5                     |
| 2        | 1, 3, 4                     |
| 3        | 1, 2, 4, 5, 6, 8            |
| 4        | 2, 3, 5                     |
| 5        | 3, 4, 6, 7                  |
| 6        | 3, 5, 7, 8                  |
| 7        | 5, 6, 8                     |
| 8        | 3, 6, 7                     |

3. Lista muchiilor: o listă (un tablou) cu perechi de vârfuri (muchiile).

### Problema 1.1. Parcurgerile BFS și DFS

Prin parcurgerea unui graf se înțelege examinarea în mod sistematic a nodurilor sale, plecând dintr-un vârf dat  $i$ , astfel încât fiecare nod accesibil din  $i$  pe două muchii adiacente două câte două să fie atins o singură dată. Metoda BFS (Breadth First Search)

parcurgea graful „în lățime”: se vizitează întâi vârful  $i$ , apoi vecinii acestuia, apoi vecinii nevizitați ai acestora și aşa mai departe. Metoda DFS (Depth First Search) se caracterizează prin faptul că se merge „în adâncime” ori de câte ori acest lucru este posibil. Scrieți un program care implementează metodele BFS și DFS pentru parcurgerea unui graf, dat prin numărul vârfurilor și matricea sa de adiacență.

Exemplu :

| graf.in                                                                                                                                                                      | bfsdfs.out                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| 8<br>0 1 1 0 0 0 0 0<br>1 0 1 1 0 0 0 0<br>1 1 0 1 1 1 0 1<br>0 1 1 0 1 0 1 0<br>0 0 1 1 0 1 1 0<br>0 0 1 1 0 1 1 0<br>0 0 1 0 1 0 1 1<br>0 0 0 0 1 1 0 1<br>0 0 1 0 0 1 1 0 | Parcurgerea BFS:<br>1 2 3 4 5 6 8 7<br>Parcurgerea DFS:<br>1 2 3 4 5 6 7 8 |
|                                                                                                                                                                              |                                                                            |

### Analiza problemei și proiectarea soluției

Vom folosi un tablou unidimensional  $viz[]$ , care conține elemente 0 sau 1, după cum vârful respectiv a fost, respectiv nu a fost vizitat. În vectorul  $C[]$  vom gestiona o coadă în care prelucrarea unui vârf constă în introducerea în celălalt capăt al ei a tuturor vârfurilor  $j$  vecine cu  $v$ , nevizitate încă. Inițial,  $v$  este egal cu  $i$  și  $viz[i] \leftarrow 1$ . Algoritmul BFS în pseudocod:

```

ALGORITM_BFS
Pentru $j \leftarrow 1, n$, pas 1 execută
 $viz[j] \leftarrow 0$ (toate varfurile nevizitate)
 $C[1] \leftarrow i$ (coada $C[]$ conține initial doar varful i)
 $p \leftarrow 1; u \leftarrow 1;$ (primul și ultimul element în coada)
 Cat_timp ($p \leq u$) execută
 $v \leftarrow C[p];$
 Pentru (toti vecinii j ai lui v , nevizitati încă) execută
 $u \leftarrow u+1;$
 $C[u] \leftarrow j;$ (adaugă varful j în coada $C[]$)
 Viziteaza varful j
 $viz[j] \leftarrow 1;$ (varful j se consideră vizitat)
 $p \leftarrow p+1;$ (se trece la urmatorul element din coada $C[]$)
STOP_ALGORITM_BFS

```

Pentru implementarea algoritmului DFS se utilizează vectorul  $viz[]$  cu aceeași semnificație ca și la algoritmul BFS și se înlocuiește coada  $C$  cu o stivă  $S$  care ne permite să plecăm în fiecare moment de la vârful curent spre primul dintre vecinii săi nevizitați, acesta din urmă fiind plasat în vârful stivei; cu el se continuă în același mod. Vom folosi un „pointer de stivă”  $ps$ , care indică elementul din vârful stivei. Dacă un element

corespunzător nevizitat este găsit, atunci acesta se plasează în vârful stivei și ps se mărește cu 1, altfel ps se micșorează cu 1. Algoritmul *DFS* în pseudocod:

```

ALGORITM_DFS
 Pentru j ← i, n, pas 1 execută
 viz[j] ← 0 (toate varfurile nevizitate)
 S[1] ← i (stiva S[] contine initial doar pe i)
 ps ← 1; (ps "poarteaza" spre elementul din varful stivei)
 Cat_timp (ps ≥ 1) execută
 j ← S[ps];
 k ← primul dintre vecinii lui j, nevizitati inca;
 Daca (un asemenea k nu exista) Atunci,
 ps ← ps - 1; (stiva coboara)
 Altfel
 Viziteaza varful k
 viz[k] ← 1; (varful k este vizitat)
 ps ← ps + 1; (pointerul de stiva se mareaște)
 S[ps] ← k; (se adauga k la stiva)
 STOP_ALGORITM_DFS

```

Programul următor implementează cei doi algoritmi (se ține cont de faptul că în C elementele tablourilor sunt considerate de la 0 la n-1).

### Program

```

#include <stdio.h>
#include <memory.h>

FILE *fin, *fout;

void BFS(int a[][20], int n, int i){
 int v, j, p, u;
 int viz[20], C[20];
 C[0]=i;
 memset(viz, 0, sizeof(viz));
 p=0; u=0; viz[0]=1;
 while(p<=u){
 v=C[p];
 for(j=0; j<n; j++)
 if(0==viz[j]&&1==a[v][j])
 C[++u]=j; viz[j]=1;
 p++;
 }
 printf(fout, "Parcurgerea BFS: \n");
 for(j=0; j<n; j++) fprintf(fout, " %d ", C[j]+1);
}

void DFS(int a[][20], int n, int i){
 int viz[20], S[20];
 int j, k, ps, gasit;
 memset(viz, 0, sizeof(viz));
 fprintf(fout, "\nParcurgerea DFS: \n");
 S[0]=i; ps=0; viz[i]=1;
 fprintf(fout, " %d ", i+1);
 while(ps>=0){
 j=S[ps]; gasit=0;
 for(k=0; !gasit&&k<n; k++)
 if(a[j][k]&&!viz[k]) gasit=1;
 if(!gasit) ps--;
 else{
 fprintf(fout, " %d ", k);
 viz[k-1]=1;
 S[++ps]=k-1;
 }
 }
}

void main(){
 int i, j, n, a[20][20];
 fin = fopen("graf.in", "r");
 fout = fopen("bfsdfs.out", "w");
 fscanf(fin, "%d", &n);
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 fscanf(fin, "%d", &a[i][j]);
 BFS(a, n, 0); DFS(a, n, 0);
 fclose(fin); fclose(fout);
}

```

```

void DFS(int a[][20], int n, int i){
 int viz[20], S[20];
 int j, k, ps, gasit;
 memset(viz, 0, sizeof(viz));
 fprintf(fout, "\nParcurgerea DFS: \n");
 S[0]=i; ps=0; viz[i]=1;
 fprintf(fout, " %d ", i+1);
 while(ps>=0){
 j=S[ps]; gasit=0;
 for(k=0; !gasit&&k<n; k++)
 if(a[j][k]&&!viz[k]) gasit=1;
 if(!gasit) ps--;
 else{
 fprintf(fout, " %d ", k);
 viz[k-1]=1;
 S[++ps]=k-1;
 }
 }
}

void main(){
 int i, j, n, a[20][20];
 fin = fopen("graf.in", "r");
 fout = fopen("bfsdfs.out", "w");
 fscanf(fin, "%d", &n);
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 fscanf(fin, "%d", &a[i][j]);
 BFS(a, n, 0); DFS(a, n, 0);
 fclose(fin); fclose(fout);
}

```

### Exercițiu

Considerați o serie de exemple „pe hârtie” și determinați parcurgerile *BFS* și *DFS* pentru ele.

### Problema 1.2. Matricea drumurilor

Se numește matricea a drumurilor asociată grafului G matricea  $M = (m_{ij})_{n \times n}$ , dată prin:

$$m_{ij} = \begin{cases} 1, & \text{dacă există în } G \text{ drum de la } x_i \text{ la } x_j \\ 0, & \text{în caz contrar} \end{cases}$$

Să se scrie un program care determină matricea drumurilor pentru un graf dat prin matricea de adjacență.

*Exemplu :*

| graf.in         | drumuri.out     |
|-----------------|-----------------|
| 8               | 1 1 1 0 1 1 1 0 |
| 0 0 1 0 0 0 0 0 | 1 1 1 0 1 1 1 0 |
| 0 0 1 0 0 0 0 0 | 1 1 1 0 1 1 1 0 |
| 1 1 0 0 1 1 0 0 | 1 1 1 0 1 1 1 0 |
| 0 0 0 0 0 0 1 0 | 1 1 1 0 1 1 1 0 |
| 0 0 1 0 0 1 1 0 | 1 1 1 0 1 1 1 0 |
| 0 0 1 0 1 0 0 0 | 1 1 1 0 1 1 1 0 |
| 0 0 0 0 1 0 0 0 | 1 1 1 0 1 1 1 0 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |

### *Analiza problemei și proiectarea soluției*

#### *Observații :*

1. Dacă  $m_{ii} = 1$ , înseamnă că există un circuit ce trece prin  $x_i$ .
2. Dacă linia  $i$  și coloana  $i$  conțin doar elemente zero, deducem că  $x_i$  este un vârf izolat.

Un algoritm simplu de determinare a matricei drumurilor unui graf este algoritmul Roy-Warshall. Acest algoritm construiește matricea drumurilor pornind de la matricea de adiacență a grafului, astfel:

```

 Pentru k ← 1, n, pas 1
 Pentru i ← 1, n, pas 1 (iak)
 Pentru j ← 1, n, pas 1 (jak)
 Daca (a[i][j]==0 AND a[i][k]==1 AND a[k][j]==1) Atunci
 a[i][j] ← 1

```

Complexitatea este  $O(n^3)$  datorită celor 3 for-uri imbricate. Programul următor implementează acest algoritm.

#### *Program*

```

#include <stdio.h>

FILE *fin, *fout;

void detDrumuri(int a[][20], int n){
 int i, j, k;
 for(k=0; k<n; k++)
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 if(i!=k&&j!=k&&0==a[i][j]&&1==a[i][k]&&1==a[k][j])
 a[i][j] = 1;
}

```

```

void main(){
 int i, j, n, a[20][20];
 fin = fopen("graf.in", "r");
 fout = fopen("drumuri.out", "w");
 fscanf(fin, "%d", &n);
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 fscanf(fin, "%d", &a[i][j]);
 detDrumuri(a, n);
 for(i=0; i<n; i++){
 fprintf(fout, "\n");
 for(j=0; j<n; j++)
 fprintf(fout, "%d ", a[i][j]);
 }
 fclose(fin); fclose(fout);
}

```

#### *Exerciții*

1. Modificați programul astfel încât să se scrie în matricea rezultat cele mai scurte lungimi ale drumurilor dintre vârfuri.
2. Să se scrie un program care, folosind matricea drumurilor, determină componentele conexe ale unui graf dat prin matricea de adiacență.

## 2. Recursivitate

Recursivitatea este una dintre noțiunile fundamentale din informatică și deseori permite rezolvarea concisă a unor probleme. Această noțiune este derivată din matematică: afirmăm că o noțiune este definită recursiv dacă în cadrul definiției apare însăși noțiunea care se definește. În programare, recursivitatea se exprimă cu ajutorul funcțiilor, deoarece ele au un nume care este specificat în apeluri. Spunem că un subprogram este recursiv dacă apelul său poate apărea când subprogramul este activ. În cadrul unui algoritm recursiv este suficient să ne gândim ce se întâmplă la un anumit nivel, pentru că la orice nivel se întâmplă același lucru. Orice funcție recursivă trebuie să conțină o condiție de oprire, adică trebuie să gândim algoritmul astfel încât procesul să fie finit. Recursivitatea indirectă are loc atunci când o funcție apelează o altă funcție care, la rândul ei, o apelează.

### *Problema 2.1. Suma cifrelor unui număr natural*

Să se scrie o funcție recursivă ce calculează suma cifrelor unui număr natural dat de la tastatură.

#### *Exemplu :*

```

Introduceti n: 98076
SumaCifre(98076) = 30

```

### Analiza problemei și proiectarea soluției

Suma cifrelor unui număr natural dat este de fapt suma dintre ultima cifră și suma cifrelor numărului obținut prin eliminarea ultimei cifre. Ultima cifră a unui număr natural  $n$  este  $n \% 10$ , iar numărul obținut prin eliminarea ultimei cifre este  $n / 10$ , aşadar:

$$\text{sumaCifrelor}(n) = n \% 10 + \text{sumaCifrelor}(n / 10)$$

### Program

```
#include <stdio.h>

short sumaCifre(unsigned long n){
 if(0==n) return 0;
 return (n%10)+sumaCifre(n/10);
}

void main(){
 unsigned long n;
 printf("Introduceti n: ");
 scanf("%ld", &n);
 printf("SumaCifre(%ld) = %d", n, sumaCifre(n));
}
```

### Exerciții

- Să se scrie o funcție recursivă pentru determinarea valorii factorialului:  
 $n! = 1 \cdot 2 \cdots (n-1) \cdot n$
- Să se scrie un program recursiv care afișează descompunerea unui număr natural în factori primi. Exemplu: pentru  $n = 12$  se va afișa  $12 = 2^2 \cdot 3^1$ .
- Să se scrie o metodă recursivă pentru a verifica dacă un număr natural este perfect (este egal cu suma divizorilor săi, mai mici decât el).
- Să se scrie un program recursiv care calculează valoarea funcției lui Ackermann pentru două valori  $m, n$  date:

$$\begin{aligned} \text{Ack}(0, n) &= n+1, \quad n \in \mathbb{N} \\ \text{Ack}(m, 0) &= \text{Ack}(m-1, 1), \quad m \in \mathbb{N}^* \\ \text{Ack}(m, n) &= \text{Ack}(m-1, \text{Ack}(m, n-1)), \quad m, n \in \mathbb{N}^* \end{aligned}$$

### Problema 2.2. Numărul 4

Se știe că din numărul 4 se poate obține orice număr aplicând următoarele operații:

- împărțire la 2;
- adăugarea unui 4 la sfărșitul numărului;
- adăugarea unui 0 la sfărșitul numărului.

Dat fiind un număr natural  $n$ , determinați o succesiune de astfel de pași care, pornind de la numărul 4, să ajungă la numărul dat.

În fișierul nr4.in se găsesc mai multe numere naturale, câte unul pe o linie. Rezultatul se va scrie în fișierul nr4.out, câte unul pe linie, ca în exemplul următor.

### Exemplu:

| nr4.in | nr4.out                            |
|--------|------------------------------------|
| 2524   | 4->2->1->10->5->50->504->252->2524 |
| 564    | 4->44->22->224->112->56->564       |
| 12     | 4->2->24->12                       |
| 3      | 4->2->24->12->6->3                 |

### Analiza problemei și proiectarea soluției

Pentru a construi sirul care duce la numărul dat, se vor face operațiile inverse de la  $n$ , până ajungem la numărul 4; acestea sunt:

- stergerea zeroului de la sfârșit, dacă acesta există;
- stergerea numărului 4 de la sfârșit, dacă acesta există;
- înmulțirea cu 2, altfel. Programul este sugestiv în acest sens.

### Program

```
#include <stdio.h>

FILE *fin, *fout;

void nr_paturu(int n){
 if(n==4){
 switch(n%10){
 case 0: nr_paturu(n/10); break;
 case 4: nr_paturu(n/10); break;
 default: nr_paturu(n*2);
 }
 fprintf(fout, "->%d", n);
 }
}

void main(){
 int n;
 fin = fopen("nr4.in", "r");
 fout = fopen("nr4.out", "w");
 while(fin && fscanf(fin, "%d", &n)==1){
 fprintf(fout, "\n4");
 nr_paturu(n);
 }
}
```

**Exercițiu**

Scrijeți „pe hârtie” pașii și valorile pe stivă (STACK) pentru  $n = 35$ .

**3. Divide et impera**

*Metoda Divide et Impera este o metodă generală de elaborare a algoritmilor și se bazează pe un principiu simplu: descompunem problema în subprobleme, pe care, la rândul lor, pentru a le rezolva, le descompunem în subprobleme și aşa mai departe; acest procedeu se repetă recursiv, până când subproblemele se încadrează într-un caz de bază (admit rezolvare imediată) și le rezolvăm corespunzător. Nu toate problemele se pot rezolva prin această metodă, datorită proprietăților necesare pentru ca problema să se poată împărți și rezolva recursiv.*

*Etapele rezolvării unei probleme prin această metodă sunt:*

1. descompunerea recursivă a problemei inițiale în subprobleme independente, similare problemei inițiale, de dimensiuni din ce în ce mai mici, până când subproblemele capătă dimensiuni care permit o rezolvare simplă;
2. rezolvarea subproblemelor ale căror dimensiuni sunt cele mai mici;
3. construirea soluțiilor subproblemelor de dimensiuni din ce în ce mai mari prin combinarea soluțiilor subproblemelor mai mici, până când vom obține soluția problemei noastre (prin combinarea ultimelor subprobleme).

**Problema 3.1. Aflarea celui mai mare divizor comun a n numere naturale**

În fișierul numere.in se găsesc mai multe (maximum 2.000) numere naturale strict pozitive, separate prin spații sau prin caracterul de sfârșit de linie. Să se calculeze cel mai mare divizor al lor utilizând un algoritm de tip Divide-et-Impera. Să se afișeze rezultatul pe ecran.

*Exemplu :*

| numere.in  | ecran     |
|------------|-----------|
| 45 60      | cmmdc = 5 |
| 125 345 65 |           |
| 9875 4555  |           |

**Analiza problemei și proiectarea soluției**

Vom citi sirul de numere în tabloul unidimensional a. Datorită structurii problemei, este evident că putem să aflăm cmmdc din prima jumătate a tabloului, apoi putem să aflăm cmmdc și din a doua jumătate a tabloului. Cel mai mare divizor comun dintre cele două rezultate este numărul căutat, adică cel mai mare divizor comun al întregului tablou. Programul următor este sugestiv în acest sens.

**Program**

```
#include <stdio.h>

unsigned cmmdc(unsigned a, unsigned b){
 unsigned r;
 while(b) {
 r = a % b;
 a = b;
 b = r;
 }
 return a;
}

unsigned Divide_et_Impera(unsigned *a, int inf, int sup){
 if(inf < sup) {
 int mijloc = (inf + sup) / 2;
 int d1 = Divide_et_Impera(a, inf, mijloc);
 int d2 = Divide_et_Impera(a, mijloc + 1, sup);
 return cmmdc(d1, d2);
 }
 return a[inf];
}

void main(){
 FILE *fin = fopen("numere.in", "r");
 unsigned a[2000], el, n = 0;
 while(fin && fscanf(fin, "%d", &el)==1) a[n++]=el;
 printf("cmmdc = %d", Divide_et_Impera(a, 0, n-1));
}
```

**Exerciții**

1. Scrijeți un program care utilizează tot această tehnică pentru a calcula minimul și maximul dintre numerele naturale din fișierul numere.in.
2. Scrijeți un program care căută un număr introdus de la tastatură în sirul de numere naturale din fișierul numere.in.

### Problema 3.2. Turnurile din Hanoi

În 1883, Edouard Lucas a inventat sau poate a reinventat unul dintre cele mai populare jocuri din toate timpurile: *Turnurile din Hanoi*<sup>1</sup>. Acesta este folosit în toate cărțile de programare pentru a exemplifica recursivitatea și metoda *Divide et impera*. Regulile jocului:

- considerăm trei tije verticale, notate cu A, B și C;
- considerăm n discuri de diametre diferite, așezate inițial pe tija A în ordinea crescătoare a diametrelor, de la bază spre vârf;
- scopul jocului este transferarea întregului turn de discuri de pe tija A pe tija B, folosind c ca tijă de manevră;
- la un moment dat, un disc se poate muta doar peste un alt disc de diametru mai mare. Pentru un număr natural n dat de la tastatură, scrieți în fișierul hanoi.out succesiunea minimă de mutări necesare pentru a ajunge în starea finală.

*Exemplu :*

| tastatură | hanoi.out                                        |
|-----------|--------------------------------------------------|
| 3         | (A, B) (A, C) (B, C) (A, B) (C, A) (C, B) (A, B) |

### Analiza problemei și proiectarea soluției

Aceasta este o problemă clasică pentru metoda *Divide et impera*. Rezolvarea se bazează pe următoarea observație: a muta n discuri de pe tija A pe tija B prin intermediul tijei C înseamnă a muta n-1 discuri de pe tija A pe tija C prin intermediul tijei B, a face mutarea A → B, a muta n-1 discuri de pe tija C pe tija B prin intermediul tijei A, adică:

$$\text{Hanoi}(n, A, B, C) = \begin{cases} A \rightarrow B, & \text{dacă } n = 1 \\ \text{Hanoi}(n - 1, A, C, B), A \rightarrow B, \text{Hanoi}(n - 1, C, B, A), & \text{altfel} \end{cases}$$

### Program

```
#include <stdio.h>

FILE *fout;

void Scrie(char A, char B) {
 fprintf(fout, "(%c, %c)", A, B);
}

void Hanoi(int n, char A, char B, char C) {
 if(n == 1) Scrie(A, B);
 else {
 Hanoi(n-1, A, C, B);
 Scrie(A, B);
 Hanoi(n-1, C, B, A);
 }
}
```

1. Comercializat ca jucărie în 1883, inițial purta denumirea de „Prof. Claus” de la colegiul „Li-Sou-Stian”, dar s-a descoperit curând că aceste nume sunt anagrame pentru „Prof. Lucas” de la colegiul „Saint Louis”. În mod obișnuit, jucăria conțineea opt discuri.

```
else{
 Hanoi(n-1, A, C, B);
 Scrie(A, B);
 Hanoi(n-1, C, B, A);
}

void main(){
 int n;
 fout = fopen("hanoi.out", "w");
 printf("n="); scanf("%d", &n);
 Hanoi(n, 'A', 'B', 'C');
}
```

### Exerciții

1. O modalitate de a ne forma o imagine de ansamblu asupra jocului este simularea grafică. Scrieți un program care, pentru un n dat, simulează mișcarea discurilor în cadrul jocului *Turnurilor din Hanoi*.
2. Iată un algoritm necursiv pentru determinarea mutărilor de pe discul inițial pe discul final respectând condițiile din enunț: pentru mișcările impare se mută cel mai mic disc (discul 1) de pe tija unde se află pe următoarea în succesiunea circulară de tije ABCABCABC... ; pentru mișcările pare, se face mișcarea posibilă care nu implică discul 1. Scrieți un program care rezolvă jocul bazat pe acest algoritm.
3. Numărul minim de mutări necesare aplicând algoritmul anterior este  $2^n - 1$ . Pentru două numere naturale n și m date,  $n \in [0, 100]$  și  $m \in [0, 2^n - 1]$ , determinați configurația tijelor la mișcarea a m-a (n reprezintă numărul de discuri).

### 4. Greedy

Metoda Greedy se aplică problemelor de optimizare. Această metodă determină întotdeauna o singură soluție a problemei. Ea construiește soluția treptat: inițial, soluția este vidă, apoi se adaugă câte un element care este cel mai promițător la momentul respectiv (de unde provine și denumirea de greedy = lacom). Alegând în orice moment elementul optim pentru situația locală, se asigură un optim local, dar nu se garantează că se va obține optimul global. Pentru a se garanta acest lucru, ar trebui demonstrat sau cunoscut faptul că, în contextul problemei, această modalitate de alegere duce la obținerea unei soluții optime. Algoritmii care implementează această metodă sunt performanți, chiar și în cazul problemelor de dimensiuni mari (timpul este liniar). Pentru anumite probleme se pot projecța algorimi Greedy care nu furnizează soluția optimă, dar sunt o bază pentru care se găsesc soluții apropiate de cea optimă. Această metodă se asemănă cu Backtracking prin faptul că vectorul-soluție se construiește progresiv, dar nu se mai execută o întoarcere la nivelul inferior (cea ce explică și diferența enormă de timp de execuție relativ la cele două metode), ci se trece la alegerea directă a următorului element.

Forma generală a unui algoritm Greedy ar putea fi:

```

S1 ← S
SOL ← Ø
Cat_Timp (nu are loc condiția de terminare) Executa
 alege x din S1 pe baza unui criteriu de optim local
 S1 ← S1 - {x}
 Dacă (SOL ∪ {x}) satisfacă criteriul) Atunci
 SOL ← SOL ∪ {x}
 Sfarsit_Cat_Timp

```

Metoda Greedy se aplică cu succes unor probleme cunoscute: algoritmul de determinare a arborelui parțial de cost minim, algoritmul lui Dijkstra pentru determinarea celor mai scurte drumezi pornind dintr-un vârf al unui graf, varianta continuă a rucsacului, problema spectacolelor, planificarea activităților etc.

#### Problema 4.1. Problema continuă a rucsacului

Se consideră un rucsac de capacitate M și n obiecte, ale căror greutate și valoare sunt date. Să se găsească o modalitate de a introduce cât mai multe obiecte în rucsac, astfel încât valoarea globală să fie maximă. Se consideră că obiectele pot fi fracționate. Presupunem că datele de intrare sunt corecte și valorile obiectelor sunt strict pozitive. Pe prima linie a fișierului de intrare obiecte.in se găsește capacitatea rucsacului M. Numărul maxim de obiecte este 50.

Exemplu :

| obiecte.in   | rucsac.out                            |
|--------------|---------------------------------------|
| 41           | În rucsac vor fi introduse obiectele: |
| 12.34 123.99 | Obiectul 2: 23.45 600.54 - COMPLET    |
| 23.45 600.54 | Obiectul 1: 12.34 123.99 - COMPLET    |
| 12.78 90.67  | Obiectul 3: 12.78 90.67 - 5.21kg      |
| 9.34 45.32   |                                       |

#### Analiza problemei și proiectarea soluției

Obiectele se sortează descrescător în ordinea raportului valoare/greutate, apoi se introduc, în această ordine, în rucsac. Ultimul obiect introdus va fi eventual fracționat. Acesta este un algoritm de tip Greedy: la fiecare pas se selectează din mulțimea obiectelor încă neintroduse obiectul care are cea mai mare valoare unitară valoare/greutate. Pentru reprezentarea unui obiect vom defini tipul TObiect de tip struct, care conține cele trei elemente definitorii ale unui obiect: greutatea, valoarea și indexul acestuia în cadrul obiectelor citite. Pentru citirea datelor din fișierul de intrare obiecte.in scriem funcția citesteDate(), care va salva obiectele în tabloul a[]. Funcția returnează, de asemenea, numărul de obiecte n și capacitatea rucsacului M. Pentru sortarea vectorului a[] de obiecte vom utiliza funcția de bibliotecă qsort(). Predicatul binar folosit de această metodă este compara() și se referă la comparația valorilor unitare a două obiecte (rapoartele valoare/greutate). După sortarea obiectelor relativ la valoarea

unitară a acestora, vom introduce pe rând obiecte în rucsac, atât timp cât acestea încap. Dacă ultimul obiect introdus este fracționat, atunci cantitatea introdusă va fi salvată în M, dar cu semnul negativ, pentru a încheia bucla while. Complexitatea algoritmului este O(n · log n), dată de algoritmul de sortare.

#### Program

```

#include <stdio.h>
#include <stdlib.h>

FILE *fout;

typedef struct{
 float g, v;
 int idx;
} TObiect;

citesteDate(TObiect *a, int *n, float *M){
 FILE *fin = fopen("obiecte.in", "r");
 float g, v;
 *n=0;
 if(fin) fscanf(fin, "%f", M);
 while(fin && fscanf(fin, "%f %f", &g, &v) == 2){
 fscanf(fin, "%f", &v);
 a[*n].g = g;
 a[*n].v = v;
 a[*n].idx = *n;
 (*n)++;
 }
}

int compara(const void *a, const void *b){
 TObiect *o1 = (TObiect*) a;
 TObiect *o2 = (TObiect*) b;
 return ((o2->v/o2->g - o1->v/o1->g)>0);
}

void main(){
 int n, i, j;
 TObiect a[50];
 float M;
 FILE *fout;
 fout = fopen("rucsac.out", "w");
 citesteDate(a, &n, &M);
 qsort((void*)a, n, sizeof(TObiect), compara);
}

```

```

i=0;
while (M>0) {
 if(M>=a[i].g) {
 M -= a[i].g;
 i++;
 } else {
 M = -M;
 }
}
fprintf(fout, "In rucsac vor fi introduse obiectele:\n");
for(j=0; j<i; j++){
 fprintf(fout, "\nObiectul %2d: %4.2f %4.2f - COMPLET",
 a[j].idx+1, a[j].g, a[j].v);
}
if(M<0) {
 fprintf(fout, "\nObiectul %2d: %4.2f %4.2f - %4.2fkg",
 a[i].idx+1, a[i].g, a[i].v, -M);
}
}

```

### Exercițiu

Considerăm varianta discretă a rucsacului, în care obiectele pot fi introduse doar întregi sau deloc. În cadrul acestei variante, metoda *Greedy* nu furnizează întotdeauna soluția optimă. Găsiți un exemplu sugestiv, pentru care aplicarea metodei *Greedy* nu determină soluția optimă.

### Problema 4.2. Colorarea hărții<sup>1</sup>

Se dă relațiile de vecinătate între cele  $n$  țări ale unei hărți. Se cere determinarea unei colorări a fiecărei țări astfel încât căriile două țări vecine să fie colorate diferit și numărul de culori folosite să fie minim. *Exemplu*:

| harta.in      | colorare.out  |
|---------------|---------------|
| 7             | 1 2 3 4 1 2 5 |
| 0 1 1 1 0 0 1 |               |
| 1 0 1 1 0 0 0 |               |
| 1 1 0 1 1 0 1 |               |
| 1 1 1 0 1 0 1 |               |
| 0 0 1 1 0 1 1 |               |
| 0 0 0 1 0 1 1 |               |
| 1 0 1 1 1 1 0 |               |

1. Teorema celor 4 culori afirma că orice hartă poate fi colorată cu patru culori. Teorema a fost demonstrată de către Appel și Haken în 1976 cu ajutorul calculatorului. Demonstrația cuprinde 700 de pagini de raționamente combinatorii ilustrate cu 10.000 de diagrame și 30.000 de pagini de imprimantă și a fost realizată în decursul a patru ani.

### Analiza problemei și proiectarea soluției

În capitolul „Algoritmi – elemente definitorii” problema este amplasată în cadrul problemelor NP-complete, adică nu există un algoritm polinomial de rezolvare a acesteia. Rezolvarea optimală se face printr-un algoritm de tip *Backtracking*, a cărui complexitate este exponențială. Algoritmul *Greedy* produce o colorare cu un număr acceptabil de culori, dar nu totdeauna cel mai mic. Vom colora prima țară cu 0, apoi succesiv țările cu prima culoare accesibilă (nu există o țară vecină cu rang mai mic de la colorată cu această culoare!).

### Program

```

#include <stdio.h>

void main(){
 FILE *fin = fopen("harta.in", "r");
 FILE *fout = fopen("colorare.out", "w");
 int n, i, j, k;
 int a[50][50], cul[50];
 int ok;
 fscanf(fin, "%d", &n);
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 fscanf(fin, "%d", &a[i][j]);
 cul[0]=0;
 for(i=1; i<n; i++){
 j=-1;
 do{
 j++; ok=1;
 for(k=0; ok&&k<i; k++)
 if(l==a[k][i]&&cul[k]==j)
 ok=0;
 }while(!ok);
 cul[i]=j;
 }
 for(i=0; i<n; i++)
 fprintf(fout, "%d ", cul[i]+1);
}

```

- cul[0] ← 0
- căută succesiv prima culoare accesibilă pentru țara  $i$  (dacă există o țară mai mică cu aceeași culoare,  $i == a[k][i]$  și  $cul[k] == j$ )
- atunci  $ok \leftarrow 0$
- ...

### Exercițiu

Pentru exemplul considerat, desenați graful și găsiți o colorare cu 4 culori.

## 5. Backtracking

Deși este o metodă care îmbunătățește căutarea exhaustivă, timpul de execuție este exponențial. De aceea, această tehnică nu poate fi aplicată fără discernământ unei probleme, ci doar atunci când suntem convinși că nu există o cale mai eficientă de rezolvare (de exemplu, un algoritm polinomial). În familia problemelor rezolvabile prin Backtracking :

- se cer toate soluțiile problemei ;
- o soluție se poate reprezenta cu ajutorul unui vector, nu neapărat cu număr fix de elemente ;
- o soluție se poate construi progresiv :  $x[1], x[2], \dots, x[k]$  ;
- putem decide, la momentul  $k$ , dacă valoarea  $x[k]$  atribuită elementului  $k$  poate conduce la o soluție, considerând fixate valorile  $x[1], x[2], \dots, x[k-1]$  ;
- putem decide, la un moment dat, dacă am obținut o soluție.

În funcție de tipul soluțiilor, o problemă Backtracking se poate încadra în mai multe categorii : Backtracking liniar de lungime fixă (problema celor  $n$  regine), Backtracking în plan (problema calului pe tabla de șah, problema labirintului, problema fotografiei), Backtracking cu soluție de lungime variabilă (problema partajiei unui număr natural) etc.

### Problema 5.1. Colorarea hărții

Să se rezolve optimal problema colorării hărții, adică să se furnizeze numărul minim de culori și o modalitate de a fi colorată harta dată, ce conține maximum 20 de țări.

*Exemplu :*

|               |               |
|---------------|---------------|
| 7             | 4             |
| 0 1 1 1 0 0 1 | 1 2 3 4 1 3 2 |
| 1 0 1 1 0 0 0 |               |
| 1 1 0 1 1 0 1 |               |
| 1 1 1 0 1 0 1 |               |
| 0 0 1 1 0 1 1 |               |
| 0 0 0 0 1 0 1 |               |
| 1 0 1 1 1 1 0 |               |

### Analiza problemei și proiectarea soluției

Pentru determinarea soluției optime se va folosi un algoritm de tip *Backtracking*, deci complexitatea este exponențială. Se vor genera toate modalitățile de a colora cele  $n$  țări și se va reține cea cu număr minim de culori. Algoritmul în pseudocod :

```

nrCul ← n
k ← 0
x[k] ← -1
Cat_Timp (k ≥ 0) Executa
 Caut pentru țara k o culoare neanalizata si corecta in
 raport cu colorarile 1, 2, ..., k-1
 Daca aceasta culoare exista Executa
 x[k] ← culoare
 Daca k = n-1 Atunci
 Daca
 (numar culori pentru x[] mai mic decat numarul
 minim curent de culori nrCul)
 Atunci reactualizeaza nrCul si cul[]
 Sfarsit_Daca
 Sfarsit_Daca
 Altfel
 Daca k < n-1 Atunci Executa
 k ← k + 1 // pasul INAINTE (forward)
 x[k] ← -1 // ne plasam pe prima pozitie
 // inainte de start
 Sfarsit_Daca
 Altfel Executa
 k ← k - 1 // pasul INAPOI (backward)
 Sfarsit_Altfel
 Sfarsit_Daca
 Sfarsit_Cat_Timp

```

Remarcăm că vom considera culorile și vîrfurile de la 0 la  $n-1$ , incrementarea cu 1 făcându-se la afișare.

### Program

```

#include <stdio.h>

void main(){
 FILE *fin = fopen("harta.in", "r");
 FILE *fout = fopen("colorare.out", "w");
 int n, i, j, k;
 int a[20][20], x[20], cul[20];
 int ok, maxCul, nrCul;
 fscanf(fin, "%d", &n);
 nrCul=n;
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
 fscanf(fin, "%d", &a[i][j]);
 x[0]=-1;
 k=0;
}

```

```

while(k>=0) {
 ok=0; x[k]++;
 while(!ok&&x[k]<n-1) {
 ok=1;
 for(j=0; j<k; j++)
 if(a[j][k]==1&&x[j]==x[k]) ok=0;
 if(!ok) { x[k]++; }
 }
 if(ok) {
 if(k==n-1) {
 maxCul=0;
 for(j=0; j<n; j++)
 if(x[j]>maxCul) maxCul=x[j];
 if(maxCul<nrCul) {
 nrCul=maxCul;
 for(j=0; j<n; j++) cul[j]=x[j];
 }
 } else {
 x[++k]=-1;
 }
 } else {
 k--;
 }
}
fprintf(fout, "%d\n", nrCul+1);
for(i=0; i<n; i++)
 fprintf(fout, "%d ", cul[i]+1);
fclose(fin); fclose(fout);
}

```

### Exercițiu

Modificați programul astfel încât să fie afișate toate colorările posibile cu număr minim de culori.

### Problema 5.2. Problema fotografiei

O fotografie alb-negru este prezentată sub forma unei matrice binare. Ea înfățișează unul sau mai multe obiecte. Pozițiile corespunzătoare obiectelor se reprezintă în matrice prin valoarea 1, iar cele corespunzătoare fundalului prin valoarea 0. Se cere să se determine numărul de obiecte nObiecte și să se coloreze obiectele cu culori diferite: 2, 3, ..., nObiecte+1. Două puncte în fotografie care au valoarea 1 aparțin aceluiași obiect dacă ele sunt vecine pe linie sau coloană.

### Exemplu:

| foto.in   | foto.out  |
|-----------|-----------|
| 4 5       | 3         |
| 1 1 0 1 1 | 2 2 0 3 3 |
| 0 0 0 1 0 | 0 0 0 3 0 |
| 1 1 0 1 1 | 4 4 0 3 3 |
| 1 0 1 1 0 | 4 0 3 3 0 |

### Analiza problemei și proiectarea soluției

Vom scrie funcția recursivă `coloreaza()`, care verifică dacă punctul dat este 1 și se află amplasat în fotografie. În acest caz, se vor colora cu culoarea curentă toți vecinii posibili, pentru care se va apela recursiv funcția.

### Program

```

#include <stdio.h>

int a[30][30], m, n;

coloreaza(int i, int j, int cul){
 if(0<=i&&i<m&&0<=j&&j<n&&a[i][j]==1) {
 a[i][j]=cul;
 coloreaza(i, j-1, cul);
 coloreaza(i, j+1, cul);
 coloreaza(i-1, j, cul);
 coloreaza(i+1, j, cul);
 }
}

void main(){
 FILE *fin = fopen("foto.in", "r");
 FILE *fout = fopen("foto.out", "w");
 int i, j, cul;
 if(fin) fscanf(fin, "%d %d", &m, &n);
 for(i=0; i<m; i++)
 for(j=0; j<n; j++)
 fscanf(fin, "%d", &a[i][j]);
 cul=1;
 for(i=0; i<m; i++)
 for(j=0; j<n; j++)
 if(a[i][j]==1) coloreaza(i, j, ++cul);
 fprintf(fout, "%d", --cul);
 for(i=0; i<m; i++)
 fprintf(fout, "\n");
}

```

```

 for(j=0; j<n; j++)
 fprintf(fout, "%d ", a[i][j]);
 }
 fclose(fin); fclose(fout);
}

```

### Exerciții

1. Dezvoltați programul astfel încât se consideră că două puncte aparțin unui același obiect și dacă ele se află vecine pe diagonală (încă patru vecini).
2. Modificați programul astfel încât tabloul `a[][]` și variabilele `m`, `n` să nu mai fie declarate global.

## 6. Programare dinamică

Această metodă se aplică în rezolvarea problemelor de optim și constă în determinarea soluției pe baza unui șir de decizii  $d_1, d_2, d_3, \dots, d_n$ , unde  $d_i$  transformă problema din starea  $s_{i-1}$  în starea  $s_i$ . Spre deosebire de metoda Divide et impera, unde subproblemele rezultate trebuie să fie independente, în cadrul acestei metode, subproblemele care apar în descompunere nu sunt independente, iar în descompunerea unei probleme vor apărea subprobleme comune mai multor subprobleme de dimensiuni mai mari. Pentru a fi eficientă, metoda programării dinamice trebuie să rezolve fiecare subproblemă o singură dată și să memoreze soluția acesteia pentru a o utiliza în cazul reapariției sale în alte subprobleme. Spre deosebire de metoda Divide et impera, metoda programării dinamice operează de jos în sus, nerecursiv, și presupune cunoașterea exactă de la început a subproblemelor care apar în descompunerea unei probleme. Metoda programării dinamice nu este o tehnică standardizată asa cum este, de exemplu, metoda Backtracking, iar elaborarea algoritmului trebuie să se bazeze doar pe câteva principii generale. Din acest motiv, abordarea acestei metode devine anevoieasă și presupune serioase aptitudini de programare, deseori însotite cu aptitudini matematice. Deoarece metoda programării dinamice găsește o soluție fără a considera toate cazurile posibile, putem găsi o asemănare și cu metoda Greedy, diferența constând în faptul că în cadrul metodelor Greedy se alege optimul local la un moment dat, iar în cazul programării dinamice se exploatează proprietatea de substructură optimă a problemei (combinarea soluțiilor optime ale subproblemelor).

### Problema 6.1. Determinarea combinărilor

Să se scrie Triunghiul lui Pascal privind combinările, până la o perche dată  $(n, k)$ .

Exemplu :

| tastatură | comb.out |   |    |    |    |   |   |
|-----------|----------|---|----|----|----|---|---|
| $n = 6$   | 1        |   |    |    |    |   |   |
| $k = 4$   | 1        | 1 |    |    |    |   |   |
|           | 1        | 2 | 1  |    |    |   |   |
|           | 1        | 3 | 3  | 1  |    |   |   |
|           | 1        | 4 | 6  | 4  | 1  |   |   |
|           | 1        | 5 | 10 | 10 | 5  | 1 |   |
|           | 1        | 6 | 15 | 20 | 15 | 6 | 1 |

### Analiza problemei și proiectarea soluției

Pentru calculul combinărilor există mai multe metode. Una dintre acestea este bazată pe formula recursivă :

$$\begin{aligned} C_n^k &= 1, \text{ dacă } k = 0 \text{ sau } k = n \\ C_n^k &= C_{n-1}^{k-1} + C_{n-1}^k, \text{ altfel} \end{aligned}$$

Se poate scrie o funcție recursivă bazată pe această formulă, dar această variantă este nerecomandată, din cauza timpului foarte mare de execuție (aceeași valoare se va calcula de mai multe ori). Vom folosi metoda programării dinamice : valorile combinărilor se vor calcula progresiv și vor fi salvate într-un tablou bidimensional.

### Program

```

#include <stdio.h>

void main()
{
 FILE *fout = fopen("comb.out", "w");
 unsigned long long C[20][20];
 int i, j, k, n;
 printf("n = "); scanf("%d", &n);
 printf("k = "); scanf("%d", &k);
 C[0][0] = 1;
 for(i=1; i<=n; i++)
 for(j=0; j<=i; j++)
 if(0==j || j==i) C[i][j]=1;
 else C[i][j]=C[i-1][j-1]+C[i-1][j];
 for(i=0; i<=n; i++)
 fprintf(fout, "\n");
 for(j=0; j<=i; j++)
 fprintf(fout, "%6d", C[i][j]);
 }
 fclose(fout);
}

```

**Exercițiu**

Scrieți și metoda recursivă, apoi implementați și măsurarea comparativă a timpului pentru cele două metode (recursivă și programare dinamică).

**Problema 6.2. Cuvinte potrivite**

Într-un fișier se găsește un text, structurat pe mai multe linii, format din cuvinte scrise cu litere mici ale alfabetului englez, separate prin spații sau și marcaje de sfârșit de linie. Scrieți un program care să determine cea mai lungă înșiruire de cuvinte din text, în ordinea în care acestea apar în textul dat, construită astfel încât, pentru oricare două cuvinte consecutive, ultimele două litere din primul cuvânt să coincidă cu primele două litere din următorul cuvânt.

*Date de intrare:* în fișierul `cuvinte.in` se găsesc cel mult 1.000 de cuvinte (fiecare de maximum 15 caractere și minimum două caractere) separate prin spații sau prin caracterul de sfârșit de linie.

*Date de ieșire:* în fișierul `cuvinte.out` trebuie scrise, pe prima linie, numărul maxim de cuvinte ce formează șirul cerut, pe următoarele linii, cuvintele acestui șir, în ordine, câte unul pe o linie.

*Exemplu :*

| <code>cuvinte.in</code>     | <code>cuvinte.out</code> |
|-----------------------------|--------------------------|
| griul                       | 7                        |
| visului   ultim             | griul                    |
| cazu   imprecis in          | ultim                    |
| clipa   aceea de isterie    | imprecis                 |
| iesita parca din            | isterie                  |
| tainite vechi si tenebroase | iesita                   |
|                             | tainite                  |
|                             | tenebroase               |

**Analiza problemei și proiectarea soluției**

Vom utiliza programarea dinamică, metoda „înainte” (ne vom deplasa de la sfârșitul spre începutul tabloului de șiruri). Folosim doi vectori: `V` și `Urm`. `V[i]` semnifică lungimea șirului maximal care începe cu șirul `i` și conține șirurile de caractere de după șirul `i`. `iMax` este lungimea maximă a unui subșir cu proprietățile din enunț și se reactualizează pe măsura deplasării spre stânga. Matematic, putem scrie:

*Initial :*

`iMax = n - 1` (cuvintele din șir se consideră de la 0 la `n-1`)  
`V[n-1] = 1` (lungimea șirului de cuvinte care începe cu ultimul cuvânt este 1)  
`Urm[n-1] = n` (nu are succesor în șirul de cuvinte)

```

Pentru i de la n-2 la 0 Executa:
 V[i]=1 +
 max{ V[j] / j > i
 si ultimele2litere(a[i])=primele2litere(a[j]) }

daca exista un j cu proprietatea
 max{ V[j] / j > i
 si ultimele2litere(a[i])=primele2litere(a[j]) },
 rezulta ca succesorul lui i este acest j, altfel succesorul
 este 1;
Sfarsit_Pentru

```

Vom implementa o funcție cu antetul

```
int PerechePotrivita(string s1, string s2)
```

care returnează 1 dacă cele două cuvinte sunt în ordinea dată de enunț (ultimele două caractere din `s1` coincid cu primele două caractere din `s2`), altfel returnează 0.

Funcția `Scrie` va scrie în fișierul de ieșire pe prima linie valoarea `V[iMax]` și pe următoarele linii succesiunea de cuvinte reconstituite pe baza vectorului `Urm`. Reconstituirea șirului de cuvinte se face utilizând vectorul `Urm`, într-o manieră iterativă.

**Program**

```

#include <stdio.h>
#include <string.h>

typedef char string[16];
string a[1001];
int n = 0, iMax,
V[1001], Urm[1001];

void Citeste(){
 FILE *f = fopen("cuvinte.in", "r");
 while (!feof(f))
 fscanf(f, "%s", a[n++]);
 fclose(f);
}

int PerechePotrivita(string s1, string s2){
 return (s1[strlen(s1) - 2] == s2[0] &&
 s1[strlen(s1) - 1] == s2[1]);
}

```

```

void Proceseaza(){
 int i, j;
 Urm[n-1] = n;
 V[n-1] = 1;
 iMax = n-1;
 for(i=n-2; i>=0; i--){
 V[i] = 1;
 Urm[i] = n;
 for(j=i+1; j<n; j++){
 if(PerechePotrivita(a[i], a[j])&&V[i]<V[j]+1){
 V[i] = V[j] + 1;
 Urm[i] = j;
 }
 }
 if(V[iMax]<V[i]) iMax = i;
 }
}

void Scrie(){
 FILE *f = fopen("cuvinte.out", "w");
 fprintf(f, "%d\n", V[iMax]);
 while(iMax!=n){
 fprintf(f, "%s\n", a[iMax]);
 iMax = Urm[iMax];
 }
 fclose(f);
}

void main(){
 Citeste();
 Proceseaza();
 Scrie();
}

```

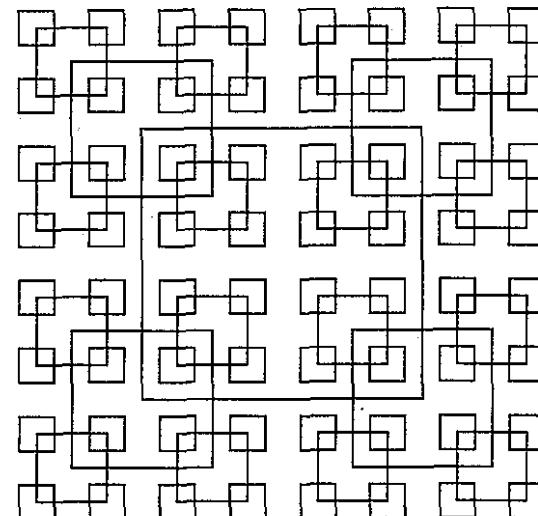
### Exerciții

- Pentru exemplul dat, scrieți fiecare iterație a algoritmului „pe hârtie”, pentru a putea analiza transformarea variabilelor.
- Rezolvați problema și prin metoda „înapoi”.

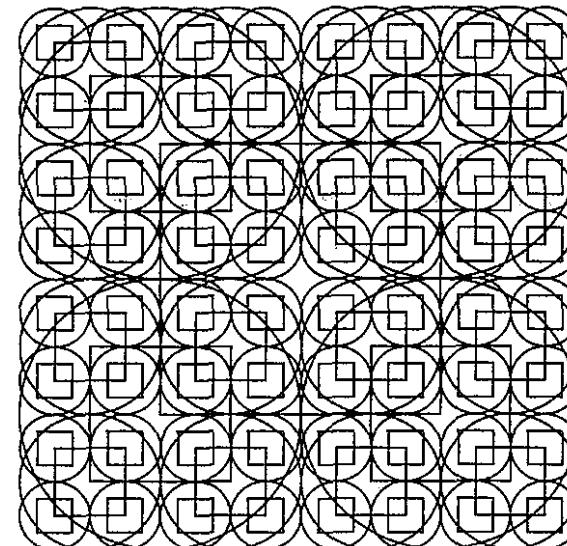
### 7. Probleme propuse

- Să se scrie un program care generează toate grafurile cu  $n$  vârfuri,  $n$  fiind introdus de la tastatură.
- Să se scrie un program care citește un graf reprezentat cu liste de adicență și determină matricea de adicență corespunzătoare. Implementați și transformarea inversă.

- Să se scrie un program recursiv care calculează valoarea unui polinom de grad  $n$  într-un punct  $x$ .
- Scrieți un program recursiv care să deseneze figura:



- Scrieți un program recursiv care să deseneze figura:



- Implementați metodele de sortare Quick Sort și Merge Sort.
- Implementați metoda de căutare binară într-un tablou ordonat.
- La un cabinet medical se prezintă simultan  $n$  pacienți. Să se determine ordinea în care medicul va trata pacienții astfel încât timpul mediu de așteptare să fie minim.

9. Se dau  $n$  șiruri ordonate crescător. Să se scrie un program care execută interclasarea lor, adică să furnizeze șirul ordonat crescător care conține toate elementele șirurilor date.
10. *Problema comis-voiajorului*. Un comis-voiajor trebuie să își vândă produsele în  $n$  orașe și să revină în orașul inițial. Se cunosc distanțele dintre oricare două orașe. Să se determine, cu ajutorul metodei Greedy, un traseu de lungime minimă pe care trebuie să îl parcurgă comis-voiajorul pentru a-și vindă produsele astfel încât să treacă prin fiecare oraș exact o singură dată, cu excepția orașului inițial (unde va fi de exact două ori). Orașele sunt identificate prin valorile  $1, 2, \dots, n$ , iar orașul inițial va fi considerat orașul 1.
11. Se citește de la tastatură un număr  $n \in N^*$  și o mulțime  $M$  de litere mici din alfabetul englezesc. Să se determine și să se scrie în fișierul text *CART.TXT*, câte unul pe linie, elementele produsului cartezian  $M \times M \times M \dots \times M = M^n$  (de  $n$  ori). Să se listeze conținutul acestui fișier și să se specifice numărul de elemente ale produsului.
12. Să se afișeze toate partiile unui număr natural  $n$  dat. Prin *partiile* a unui număr natural înțelegem o descompunere a sa ca sumă de numere naturale nenule. Fiecare parte se va afișa numai o dată, relativ la ordinea termenilor. Exemplu: pentru  $n = 4$  se va determina  $1+1+1+1, 1+1+2, 1+3, 2+2, 4$ .
13. Se dă un număr natural par  $n > 0$ . Să se determine toate șirurile de  $n$  paranteze care se închid corect. De exemplu, pentru  $n = 6$ , avem:  $((())), ((()()), ((())(), ((())()$ .
14. *Problema celor n regine*: pentru un număr natural  $n$  dat, să se determine toate posibilitățile de a amplasa  $n$  regine pe tabla de șah de dimensiune  $n \times n$  astfel încât oricare două să nu se atace între ele.
15. *Triunghi de numere*. Analizăm un triunghi de numere ca în exemplul de mai jos și drumul care merge de la prima spre ultima linie astfel încât se va începe cu primul număr și de fiecare dată se coboară pe următoarea linie exact dedesubt sau cu o poziție la dreapta. Găsiți drumul pentru care suma numerelor este maximă.

| <i>triangle.in</i>  | <i>triangle.out</i> |
|---------------------|---------------------|
| 7                   | Suma maxima = 346   |
| 10                  | 1                   |
| 82 81               | 2                   |
| 4 6 10              | 3                   |
| 2 14 35 7           | 3                   |
| 41 3 52 26 15       | 3                   |
| 32 90 11 87 56 23   | 4                   |
| 54 65 89 32 71 9 31 | 5                   |

16. *Calul pe tabla de șah*. O problemă celebră este determinarea unui mod de a parcurge tabla de șah de către un cal astfel încât acest cal să treacă prin fiecare patrat exact o singură dată. Vom încerca să rezolvăm în continuare o problemă înrudită cu aceasta: date fiind două patrate pe tabla de șah de dimensiune pătratică ( $n \times n$ ), determinați cel mai scurt drum dintră acestea utilizând mutările unui cal.
- Date de intrare*: de la tastatură se citesc cinci numere naturale  $n, 11, c1, 12, c2$ , cu semnificația:  $n$  – număr natural ( $4 \leq n \leq 20$ ),  $11, 12, c1, c2$  numere naturale cuprinse între 1 și  $n$  inclusiv.

*Date de ieșire*: pe ecran se va scrie propoziția:

Cel mai scurt drum este de  $p$  pasi.

unde  $p$  este numărul minim de pași necesari pentru a ajunge din poziția  $(11, c1)$  în poziția  $(12, c2)$ , urmată de drumul parcurs, ca în exemplul de mai jos.

*Exemplu*:

| intrare<br>(tastatură) | ieșire<br>(ecran)                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------------------------|
| 8 1 1 8 7              | Cel mai scurt drum este de 5 pasi.<br>Drumul parcurs: (1, 1) (2, 3) (3, 5) (4, 7) (6, 6)<br>(8, 7)                  |
| 5 5 2 1 4              | Cel mai scurt drum este de 2 pasi.<br>Drumul parcurs: (5, 2) (3, 3) (1, 4)                                          |
| 15 2 13 14 15          | Cel mai scurt drum este de 6 pasi.<br>Drumul parcurs: (2, 13) (4, 12) (6, 11) (8, 12)<br>(10, 13) (12, 14) (14, 15) |

*Observație*:

Drumul parcurs nu este neapărat unic!

17. *Colecționarul de cărți rare*. Un colecționar de cărți rare a descoperit o carte scrisă într-o limbă neobișnuită, care folosește aceleași litere ca și alfabetul englez. Cartea conține un scurt index, dar ordinea cuvintelor în index este diferită de cea din alfabetul englez. Colecționarul a încercat apoi să se folosească de acest index pentru a determina ordinea caracterelor și a reușit cu greu să rezolve această problemă. Problema noastră este de a scrie un program care, pe baza unui index sortat dat, să determine ordinea literelor în alfabetul necunoscut.

*Date de intrare*: În fișierul *alfabet.in* se găsesc cel puțin unul și cel mult 100 de cuvinte, urmate de o linie care conține caracterul „#”. Fiecare cuvânt are cel mult 20 de caractere. Cuvintele dintr-un index conțin numai litere mari din alfabetul englez și apar în ordinea crescătoare corespunzătoare alfabetului. Nu toate literele mari vor apărea neapărat în index, dar un index va determina o ordine unică a caracterelor care apar.

*Date de ieșire*: Pentru datele din fișierul *alfabet.in*, să se scrie în fișierul *alfabet.out* ordinea literelor, sub forma unui șir de litere mari, fără spații între ele.

*Exemplu*:

| <i>alfabet.in</i> | <i>alfabet.out</i> |
|-------------------|--------------------|
| ION               | IANOD              |
| ANA               |                    |
| ADONIA            |                    |
| DOINA             |                    |
| DOINN             |                    |
| DDAN              |                    |
| DDAO              |                    |
| #                 |                    |

| alfabet.in | alfabet.out |
|------------|-------------|
| L          | L           |
| #          |             |
| XWY        | XZYW        |
| ZX         |             |
| ZXY        |             |
| ZXW        |             |
| YWWX       |             |
| #          |             |

(Adaptarea unei probleme propuse la finala ACM, 1990)

## Probleme de concurs

15 probleme complet rezolvate, 32 de exerciții și probleme propuse.  
Probleme rezolvate :

- Cuburi
- Numere Bangla
- Să tăiem pizza !
- Secvență Collatz
- PI
- Intersecție de cercuri
- Sortarea DNA-ului
- Cutii ascunse
- Numere fel de fel
- Big Mod
- Numere umile
- Moneda contrafăcută
- Numere unu
- Codul secret
- Împărțirea cărților la Bridge

*Matematica este atât de serioasă, că nu ar trebui să pierdem nici o ocazie ca să o facem distractivă.*

Blaise Pascal

### **Problema 1. Cuburi**

Ionuț are o pasiune pentru jocurile cu cuburi. El le aşază unul peste altul și construiește coloane de diferite înălțimi. Îi arată surorii sale „opera”, lăudându-se cu „zidul” pe care l-a construit, dar aceasta îi reproșează: „Ca să fie zid, trebuie să aibă toate coloanele de aceeași înălțime!”, moment în care Ionuț și-a dat seama că ea are dreptate. Așa că s-a gândit să rearanjeze cuburile, unul câte unul, astfel încât coloanele obținute să aibă aceeași înălțime. Îl puteți ajuta pe Ionuț să realizeze acest lucru printr-un număr minim de mutări?

*Date de intrare :* în fișierul `cuburi.in` se găsesc mai multe cazuri de n coloane pe care Ionuț le-a construit. Pe o linie se află numărul n de stive, pe următoarea se află n numere naturale reprezentând înălțimile celor n stive. Considerăm că  $1 \leq n \leq 50$  și  $1 \leq h_i \leq 100$ . Numărul total de cuburi este divizibil cu numărul de stive, astfel încât vom putea să construim n coloane de aceeași înălțime. Datele de intrare se încheie la citirea lui n = 0. Acest set de date nu trebuie procesat.

*Date de ieșire :* pentru fiecare caz, așa cum s-a prezentat mai sus, trebuie afișate două propoziții: numărul cazului și numărul minim de mutări în forma din exemplu.

*Exemplu :*

| <code>cuburi.in</code> | <code>cuburi.out</code>          |
|------------------------|----------------------------------|
| 6                      | Cazul #1                         |
| 5 2 4 1 7 5            | Numărul minim de mutări este 5.  |
| 6                      | Cazul #2                         |
| 4 7 2 3 9 5            | Numărul minim de mutări este 6.  |
| 3                      | Cazul #3                         |
| 11 23 2                | Numărul minim de mutări este 11. |
| 0                      |                                  |

*(ACM Southwestern European Regional Programming Contest 1997, enunț modificat)*

#### **Analiza problemei și proiectarea soluției**

Lungimea corespunzătoare unei coloane finale se poate afla ușor însumând lungimile coloanelor inițiale și împărțind această sumă la numărul lor.

Conform datelor problemei, vor trebui mutate cuburi de pe stivele mai înalte decât această lungime pe cele cu lungimea mai mică, până când vor fi completate toate coloanele. Numărul minim de mutări va fi aşadar suma diferențelor dintre lungimea

necesară și lungimea coloanelor mai mici decât media. Următorul program tocmai specifică acest lucru.

### Program

```
#include <stdio.h>

FILE *fin;
FILE *fout;
unsigned short n, a[100];
unsigned int s, rez, caz = 0;

void CitesteDate(){
 short i;
 s = 0;
 for(i=0; i<n; i++){
 fscanf(fin, "%d", &a[i]);
 s += a[i];
 }
 fscanf(fin, "\n");
 printf("n = %d", n);
}

void Prelucreaza(){
 unsigned short aux = s/n;
 int i;
 rez = 0;
 for(i=0; i<n; i++)
 if(a[i]<aux) rez += aux - a[i];
}

void Afiseaza(){
 fprintf(fout, "Cazul #%d\n", ++caz);
 fprintf(fout, "Numarul minim de mutări este %d.\n", rez);
}

void main(){
 fin = fopen("cuburi.in", "r");
 fout = fopen("cuburi.out", "w");
 while(!feof(fin)&&fscanf(fin, "%d\n", &n)>0&&n){
 CitesteDate();
 Prelucreaza();
 Afiseaza();
 }
 fclose(fin); fclose(fout);
}
```

### Exercițiu

Determinați și o succesiune de mutări corespunzătoare obținerii cerinței problemei.

### Problema 2. Numere Bangla

Numerele Bangla utilizează cuvintele kuti (10.000.000), lakh (100.000), hajar (1000), shata (100) pentru a converti în text un număr dat. Scrieți un program pentru a converti un număr în textul corespunzător.

*Date de intrare :* Fișierul de intrare bangla.in conține câteva cazuri de test. Fiecare caz este o linie cu un număr natural  $\leq 999.999.999.999.999$ .

*Date de ieșire :* Pentru fiecare caz de test din fișierul de intrare se va scrie în fișierul de ieșire bangla.out numărul expandat la text, precedat de numărul cazului, ca în exemplul următor.

*Exemplu :*

| bangla.in       | bangla.out                                          |
|-----------------|-----------------------------------------------------|
| 999999999999999 | 1. 9 kuti 99 lakh 99 hajar 9 shata 99 kuti 99 lakh  |
| 0               | 99 hajar 9 shata 99                                 |
| 1               | 2. 0                                                |
| 2               | 3. 1                                                |
| 3               | 4. 2                                                |
| 7               | 5. 3                                                |
| 8               | 6. 7                                                |
| 9               | 7. 8                                                |
| 10              | 8. 9                                                |
| 11              | 9. 10                                               |
| 3243545         | 10. 11                                              |
| 312323          | 11. 32 lakh 43 hajar 5 shata 45                     |
| 212123          | 12. 3 lakh 12 hajar 3 shata 23                      |
| 6565            | 13. 2 lakh 12 hajar 1 shata 23                      |
| 323             | 14. 6 hajar 5 shata 65                              |
| 45455555        | 15. 3 shata 23                                      |
| 111111111111111 | 16. 4 kuti 54 lakh 55 hajar 5 shata 55              |
| 23764           | 17. 1 kuti 11 lakh 11 hajar 1 shata 11 kuti 11 lakh |
| 45897458973958  | 11 hajar 1 shata 11                                 |
| 100000000       | 18. 23 hajar 7 shata 64                             |
| 1000000000      | 19. 45 lakh 89 hajar 7 shata 45 kuti 89 lakh 73     |
| 11000000000     | hajar 9 shata 58                                    |
| 10000000001     | 20. 10 kuti 0                                       |
| 23064           | 21. 1 shata 0                                       |
|                 | 22. 1 hajar 1 shata 0                               |
|                 | 23. 10 hajar 1                                      |
|                 | 24. 23 hajar 64                                     |

(Tanbir Ahmed)

### *Analiza problemei și proiectarea soluției*

Având un număr cu 15 cifre  $a_0a_1a_2\dots a_{14}$ , extragem din el numerele întregi cu maximum 2 cifre astfel:  $a_0, a_1a_2, a_3a_4, a_5, a_6a_7, a_8a_9, a_{10}a_{11}, a_{12}, a_{13}a_{14}$ . Acest lucru îl realizează funcția cu antetul:

```
void transforma(string s, int *a)
```

care pune aceste numere în ordine în tabloul unidimensional  $a[]$ .

În programul principal se citește numărul dat într-un string, se completează cu zerouri primele poziții până la 15 și se apelează funcția de mai sus. Apoi, se afișează rezultatul utilizând tabloul constant  $den[9][6]$ .

### *Program*

```
#include <stdio.h>
#include <string.h>

typedef char string[150];

const char den[9][6] = {"kuti", "lakh", "hajar", "shata",
 "kuti", "lakh", "hajar", "shata",
 ""};

FILE *fin;
FILE *fout;

void transforma(string s, int *a){
 int i, k = 0;
 a[k++] = s[0] - '0';
 for(i=1; i<4; i+=2)
 a[k++] = s[i+1]-'0'+10*(s[i]-'0');
 a[k++] = s[5] - '0';
 for(i=6; i<11; i+=2)
 a[k++] = (s[i+1]-'0') + 10*(s[i]-'0');
 a[k++] = s[12] - '0';
 a[k++] = (s[13]-'0')*10 + (s[14]-'0');
}

void main(){
 string s, s1;
 int i, j, a[16], caz=0;
 fin = fopen("bangla.in", "r");
 fout = fopen("bangla.out", "w");
 while(fin&&fscanf(fin, "%s\n", s1)>0){
 for(i=0; i<15-strlen(s1); i++) s[i]='0';
 s[15-strlen(s1)]='\0';
 strcat(s, s1);
 if(caz) fprintf(fout, "\n");
 fprintf(fout, "%2d.", ++caz);
 transforma(s, a);
 i=0;
 while(!a[i]) i++;
 if(i==9) fprintf(fout, " %d", 0);
 for(j=i; j<9; j++){
 while(a[j]==0 && j<9) j++;
 if(j==9) fprintf(fout, " %d", 0);
 else
 fprintf(fout, " %d %s", a[j], den[j]);
 }
 }
 fclose(fin); fclose(fout);
}
```

### *Exercițiu*

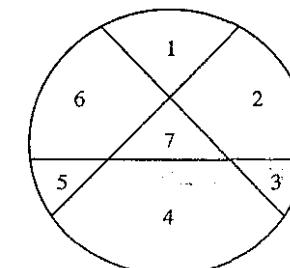
Rezolvați și problema inversă: dat fiind un sir ce reprezintă un număr ca text, transformați-l în numărul natural corespunzător.

### *Problema 3. Să tăiem pizza !*

Într-o zi, Felix a fost rugat să tăie o pizza în șapte bucăți, pentru a o distribui prietenilor. (Dimensiunile bucătilor de pizza nu trebuie neapărat să fie identice, deci putem obține o bucată mai mare sau mai mică decât alta.) El s-a gândit un pic și a ajuns la concluzia că poate să tăie pizza din trei tăieturi drepte astfel încât să rezulte șapte bucăți. Încercați să realizați o performanță similară cu ajutorul computerului! Adică, dat fiind numărul de linii drepte, calculați numărul maxim de bucăți care se pot obține.

*Date de intrare :* în fișierul `pizza.in` se găsesc mai multe cazuri de test, câte unul pe linie ( $0 \leq N \leq 210.000.000$ ), fiecare linie reprezentând numărul de linii drepte. Un număr negativ încheie lista cazurilor din fișierul de intrare.

*Date de ieșire :* în fișierul `pizza.out` vor trebui scrise numărul maxim de bucăți de pizza ce se pot obține pentru fiecare număr din fișierul de intrare, în forma din exemplu.



*Exemplu:*

| Pizza.in  | pizza.out                                       |
|-----------|-------------------------------------------------|
| 0         | 0 tăieturi -> 1 bucăti.                         |
| 1         | 1 tăieturi -> 2 bucăti.                         |
| 2         | 2 tăieturi -> 4 bucăti.                         |
| 5         | 5 tăieturi -> 16 bucăti.                        |
| 6         | 6 tăieturi -> 22 bucăti.                        |
| 7         | 7 tăieturi -> 29 bucăti.                        |
| 8         | 8 tăieturi -> 37 bucăti.                        |
| 9         | 9 tăieturi -> 46 bucăti.                        |
| 10        | 10 tăieturi -> 56 bucăti.                       |
| 210000000 | 210000000 tăieturi -> 22050000105000001 bucăti. |
| 678       | 678 tăieturi -> 230182 bucăti.                  |
| 9043      | 9043 tăieturi -> 40892447 bucăti.               |
| -100      |                                                 |

### *Analiza problemei și proiectarea soluției*

Dacă notăm cu  $S_n$  numărul de bucăți care se vor obține din  $n$  linii trasate, atunci obținem următoarea recurență:

$$\begin{aligned}S_0 &= 1 \\S_n &= S_{n-1} + n\end{aligned}$$

De unde obținem:

$$S_n = 1 + n*(n+1)/2$$

Întrucât dimensiunea  $n$  maximă va furniza un rezultat care depășește tipul unsigned long, vom simula operația de înmulțire cu depunerea rezultatului într-un tablou, prin funcția cu antetul:

```
void operatie(long int a, "long int b)
```

Transportul  $t$  se inițializează cu 1, datorită formulei de mai sus (pentru a nu face încă o operație de adunare ulterior).

### *Program*

```
#include <stdio.h>

short v[50], k;
long n;
FILE *fin, *fout;

void operatie(long int a, "long int b){
 unsigned long t = 1;
 short c;
 k = 0;
```

```
while(a){
 c = a % 10;
 a /= 10;
 v[k++] = (t + c*b) % 10;
 t = (t + c*b) / 10;
}
while(t){
 v[k++] = t % 10;
 t = t / 10;
}

void scrie(){
 int i;
 fprintf(fout, "%ld tăieturi -> ", n);
 for(i=k-1; i>=0; i--)
 fprintf(fout, "%d", v[i]);
 fprintf(fout, " bucăti.");
 fprintf(fout, "\n");
}

void main(){
 fin = fopen("pizza.in", "r");
 fout = fopen("pizza.out", "w");
 while(fscanf(fin, "%ld\n", &n) && n>=0){
 if(n%2 == 0) operatie(n+1, n/2);
 else operatie(n, (n+1)/2);
 scrie();
 }
}
```

### *Exercițiu*

Dat fiind numărul maxim de bucăți în care s-a tăiat o pizza din  $n$  linii, determinați  $n$  (problema inversă).

### *Problema 4. Secvență Collatz*

*Lothar Collatz* a introdus o metodă de generare a unui sir de numere după următoarele reguli:

Pas 1. Se consideră un număr natural strict pozitiv.

Pas 2. Dacă  $A = 1$  stop.

Pas 3. Dacă  $A$  este par, atunci se înlocuiește cu  $A/2$  și se trece la pasul 2.

Pas 4. Dacă  $A$  este impar, atunci se înlocuiește cu  $3*A+1$  și se trece la pasul 2.

S-a arătat că algoritmul se oprește întotdeauna pentru A mai mic sau egal cu 109, dar unele valori ale lui A pot să depășească dimensiunea unui întreg în cadrul programului. În această problemă dorim să determinăm lungimea unei secvențe care include toate valorile produse până când algoritmul se oprește (în pasul 2) sau întâlnim o valoare mai mare decât o limită specificată (în pasul 4).

*Date de intrare:* mai multe cazuri de test și pentru fiecare caz intrarea conține o linie cu două numere întregi, primul fiind valoarea inițială a lui A (pentru cazul 1) și al doilea fiind L, valoarea limită pentru termenii din secvență. Ambele se încadrează într-un întreg pe 32 de biți. Valoarea inițială a lui A este întotdeauna mai mică decât L. Ultima linie conține două numere negative.

*Date de ieșire:* pentru fiecare caz de intrare se va scrie numărul cazului (secvențial și începând cu 1), o virgulă, valoarea inițială a lui A, valoarea-limită L și numărul de termeni calculați.

*Exemplu :*

| tastatură     | ecran                                                     |
|---------------|-----------------------------------------------------------|
| 3 100         | Case 1: A = 3, limit = 100, number of terms = 8           |
| 34 100        | Case 2: A = 34, limit = 100, number of terms = 14         |
| 75 250        | Case 3: A = 75, limit = 250, number of terms = 3          |
| 27 2147483647 | Case 4: A = 27, limit = 2147483647, number of terms = 112 |
| 101 304       | Case 5: A = 101, limit = 304, number of terms = 26        |
| 101 303       | Case 6: A = 101, limit = 303, number of terms = 1         |
| -1 -1         |                                                           |

(ACM North Central Regionals, 1998)

### Analiza problemei și proiectarea soluției

*Funcția cu antetul:*

```
long nrTerms(long A, long L)
```

returnează numărul de termeni ceruți, respectând regulile de formare a sirului Collatz.

### Program

```
#include <stdio.h>

long nrTerms(long A, long L){
 long rez = 1;
 while (A!=1){
 if(A%2==0) (A/=2; rez++);
 else
 if(A>(L-1)/3) A=1;
 else {A = 3*A + 1; rez++;}
 }
 return rez;
}
```

```
void main(){
 long A, L;
 long caz = 1;
 while(scanf("%ld %ld\n", &A, &L)>0 && A>0 && L>0)
 printf(
 "Case %ld: A = %ld, limit = %ld, number of terms = %ld\n",
 caz++, A, L, nrTerms(A, L));
}
```

### Exercițiu

Dat fiind un număr întreg L,  $1 \leq L \leq 100$ , determinați toate numerele naturale A, care se încadrează pe 4 octeți și au lungimea sirului Collatz L, dar nu se termină forțat (nici un termen al sirului nu depășește reprezentarea pe 4 octeți!).

### Problema 5. PI

Profesorul Robert A.J. Matthews de la Departamentul de Matematică Aplicată și Computer al Universității Aston din Anglia a descris recent cum pozițiile stelelor pe cer pot să furnizeze valoarea aproximativă a lui  $\pi$ . Acest rezultat a fost urmat de aplicarea unor teoreme din teoria numerelor, astfel: dintre toate perechile dintr-o mulțime de numere aleatorii, probabilitatea ca două dintre ele să nu aibă divizor comun este

$$\frac{6}{\pi^2}$$

De exemplu, utilizând o mulțime mică de numere: 2, 3, 4, 5, 6, avem exact 10 perechi ce pot fi formate: (2, 3), (2, 4) etc. 10 dintre aceste perechi nu au un factor comun, altul decât 1: (2, 3), (2, 5), (3, 4), (3, 5), (4, 5) și (5, 6). Folosind relația de mai sus, obținem:

$$\frac{6}{n^2} \approx \frac{6}{10}$$

$$\pi \approx 3,162$$

În această problemă se primește un set de date care reprezintă o mulțime de numere pseudo-aleatorii. Corespunzător fiecărui set de date, trebuie să determinați aproximarea lui  $\pi$  pe baza regulii de mai sus.

*Date de intrare:* sunt o mulțime de seturi de date. Pe prima linie a fiecărui set de date se află un număr întreg pozitiv N, mai mare decât 1 și mai mic decât 50. Pe fiecare dintre următoarele N linii se află câte o valoare întreagă pozitivă mai mare decât 0 și mai mică decât 32768. Valoarea 0 va încheia sirul datelor de intrare.

*Date de ieșire:* pentru fiecare set de date de intrare se va scrie pe ecran o linie cu numărul  $\pi$  corespunzător calculat sau o propoziție care să spună că nu se poate căstiga, ca în exemplul următor.

*Exemplu :*

| tastatură | écran                          |
|-----------|--------------------------------|
| 5         | 3.162278                       |
| 2         | No estimate for this data set. |
| 3         |                                |
| 4         |                                |
| 5         |                                |
| 6         |                                |
| 2         |                                |
| 13        |                                |
| 19        |                                |
| 0         |                                |

(ACM East-Central Regionals, 1995)

### Analiza problemei și proiectarea soluției

Numărul de perechi diferite ce se pot forma din  $n$  numere este  $n * (n - 1) / 2$ . Înțând cont de această observație, obținem aproximarea lui  $\pi$  ca fiind:

$$\pi = \frac{6}{\text{rez}} \cdot \frac{n(n - 1)}{2}$$

unde rez este numărul de perechi cu cel mai mare divizor comun 1.

### Program

```
#include <stdio.h>
#include <math.h>

int cmmdc(int a, int b){
 while(a != b)
 if(a > b) a -= b;
 else b -= a;
 return a;
}

void main(){
 int rez;
 int a[100], i, j, n;
 double aux;
 while(scanf("%d\n", &n) > 0 && n) {
 rez = 0;
 for(i=0; i<n; i++) scanf("%d\n", &a[i]);
 for(i=0; i<n; i++)
 for(j=i+1; j<n; j++)
 if(cmmdc(a[i], a[j]) == 1) rez++;
 }
}
```

```
if(rez) {
 aux = sqrt(6.0 / (float)rez * (float)(n * (n - 1) / 2));
 printf(" %f\n", aux);
}
else
 printf("No estimate for this data set.\n");
}
```

### Exercițiu

Îmbunătățiți programul astfel încât citirea și afișarea să se facă din/în fișiere.

### Problema 6. Intersecție de cercuri

Ecuația unui cerc cu raza  $r$  și centrul  $(x_c, y_c)$  este:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Scrieți un program care decide dacă două cercuri se intersecțează și, dacă da, calculează și punctele de intersecție.

*Date de intrare :* se citesc din fișierul circles.in. Pe fiecare linie a acestui fișier se află câte trei numere reprezentând, respectiv,  $x_c$ ,  $y_c$  și  $r$ , o pereche de linii reprezentând o intrare pentru problemă.

*Date de ieșire :* se scriu în fișierul circles.out. Pentru un set de date se va scrie în fișierul de ieșire câte o linie astfel: NO INTERSECTION dacă cele două cercuri nu se intersecțează; THE CIRCLES ARE THE SAME dacă cercurile au un număr infinit de puncte de intersecție; punctul sau cele două puncte de intersecție ca în exemplul de mai jos.

### Exemplu :

| circles.in       | circles.out                     |
|------------------|---------------------------------|
| 0.0 0.0 1.0      | (0.000,1.000)                   |
| 0.0 2.0 1.0      | (-1.734,3.234) (2.234,-0.734)   |
| 1.0 2.0 3.0      | NO INTERSECTION                 |
| 4.0 5.0 6.0      | THE CIRCLES ARE THE SAME        |
| 0.0 0.0 1.0      | (0.500,-0.866) (0.500,0.866)    |
| 3.0 0.0 1.0      | (80.396,23.915) (91.413,91.276) |
| 0.0 0.0 1.0      |                                 |
| 0.0 0.0 1.0      |                                 |
| 0.0 0.0 1.0      |                                 |
| 1.0 0.0 1.0      |                                 |
| 90.89 56.78 34.5 |                                 |
| 23.45 67.81 71.9 |                                 |

### Analiza problemei și proiectarea soluției

Vom scrie funcțiile sugestive `intVida()` și `coincid()`, care decid dacă cele două cercuri date nu se intersectează, respectiv dacă ele coincid. Pentru a decide dacă cele două cercuri se intersectează, scriem funcția `intersect()`, care returnează numărul de puncte de intersecție dintre cele două cercuri (1 sau 2).

Presupunând că abscisele centrelor sunt diferite ( $x_1 \neq x_2$ ) și că  $(x_0, y_0)$  ar fi un punct de intersecție, obținem pe rând următoarele relații:

$$\begin{cases} (x_0 - x_1)^2 + (y_0 - y_1)^2 = r_1^2 \\ (x_0 - x_2)^2 + (y_0 - y_2)^2 = r_2^2 \end{cases}$$

Dacă scădem a doua ecuație din prima, obținem:

$$2x_0(x_2 - x_1) + x_1^2 - x_2^2 + 2y_0(y_2 - y_1) + y_1^2 - y_2^2 = r_1^2 - r_2^2 \rightarrow \\ x_0 = \frac{y_1 - y_2}{2(x_2 - x_1)} y_0 + \frac{x_1^2 - r_2^2 - y_1^2 + y_2^2 - x_1^2 + x_2^2}{2(x_2 - x_1)}$$

Notăm:

$$A = \frac{y_1 - y_2}{2(x_2 - x_1)} \\ B = \frac{x_1^2 - r_2^2 - y_1^2 + y_2^2 - x_1^2 + x_2^2}{2(x_2 - x_1)}$$

și deci:

$$x_0 = Ay_0 + B$$

Introducând această relație în ecuația primului cerc vom obține o ecuație de gradul al doilea cu necunoscuta  $y_0$ , pe care o rezolvăm. Introducând valoarea găsită pentru  $y_0$  în ecuația celui de al doilea cerc, îl obținem pe  $x_0$ .

Dacă  $x_1 = x_2$ , atunci repetăm același raționament, calculând prima dată pe  $x_0$ .

### Program

```
#include <stdio.h>
#include <math.h>

typedef struct{
 float x, y;
} Punct;

Punct O1, O2, arrInt[2], aux;
float r1, r2; . . .
FILE *fin, *fout;
```

```
float sqr(float x){
 return x*x;
}

float abs(float x){
 if(x < 0) return -x;
 else return x;
}

float dist(Punct p, Punct q){
 return
 sqrt(sqr(p.x-q.x) + sqr(p.y-q.y));
}

int intVida(){
 return
 (dist(O1, O2) > r1 + r2)
 ||
 (dist(O1, O2) < abs(r1 - r2))
 ;
}

int coincid(){
 return
 O1.x == O2.x && O1.y == O2.y &&
 r1 == r2;
}

int intersect(){
 float A, B, C, D, A1, B1;
 int nrInt = 0;
 if(O1.x != O2.x){
 A = A1 = (O1.y - O2.y)/(O2.x - O1.x);
 B =
 sqr(r1)-sqr(r2)-sqr(O1.y)+sqr(O2.y)-sqr(O1.x)+sqr(O2.x);
 B /= 2 * (O2.x - O1.x);
 B1 = B;
 B -= O1.x;
 D = A;
 A = sqr(A) + 1;
 C = sqr(B) + sqr(O1.y) - sqr(r1);
 B = 2*D*B - 2*O1.y;
 D = sqr(B) - 4*A*C;
 if(D < 0) nrInt = 0;
 else if(D == 0) nrInt = 1;
 else nrInt = 2;
 }
}
```

```

if(D == 0){
 nrInt = 1;
 arrInt[0].y = -B/(2 * A);
 arrInt[0].x = A1 * arrInt[0].y + B1;
}
else
 if(D>0){
 nrInt = 2;
 arrInt[0].y = (-B - sqrt(D)) / (2 * A);
 arrInt[1].y = (-B + sqrt(D)) / (2 * A);
 arrInt[0].x = A1 * arrInt[0].y + B1;
 arrInt[1].x = A1 * arrInt[1].y + B1;
 }
else{
 B1 =
 (sqr(r1)-sqr(r2)-sqr(O1.y)+sqr(O2.y))/(2*(O2.y-O1.y));
 A = sqr(r2) - sqr(B1 - O2.y);
 if (A==0){
 nrInt = 1;
 arrInt[0].x = O1.x;
 arrInt[0].y = B1;
 }
 else
 if(A>0){
 nrInt = 2;
 arrInt[0].x = O2.x - sqrt(A);
 arrInt[1].x = O2.x + sqrt(A);
 arrInt[0].y = arrInt[1].y = B1;
 }
 }
return nrInt;
}

void main(){
 int n;
 fin = fopen("circles.in", "r");
 fout = fopen("circles.out", "w");
 while(fscanf(fin, "%f %f %f\n", &O1.x, &O1.y, &r1)>0 &&
 fscanf(fin, "%f %f %f\n", &O2.x, &O2.y, &r2)>0){
 n = intVida();
 if(coincid())
 fprintf(fout, "THE CIRCLES ARE THE SAME\n");
 else

```

```

if(intVida())
 fprintf(fout, "NO INTERSECTION\n");
else{
 n = intersect();
 if(n==2 &&
 (arrInt[0].x < arrInt[1].x || arrInt[0].x == arrInt[1].x &&
 arrInt[0].y < arrInt[1].y))
 { aux = arrInt[0]; arrInt[0] = arrInt[1];
 arrInt[1] = aux; }
 n--;
 while (n>=0){
 fprintf(fout, "(%.3f,%.3f)", arrInt[n].x, arrInt[n].y);
 n--;
 }
 fprintf(fout, "\n");
}
fclose(fin);fclose(fout);
}

```

### Exercițiu

Date fiind un cerc și un dreptunghi (definit prin colțurile din stânga-sus și dreapta-jos), determinați punctele de intersecție dintre cele două figuri.

### Problema 7. Sortarea DNA-ului

O metodă de a măsura „neordonarea” unei secvențe este numărarea perechilor care nu sunt ordonate. De exemplu, în secvența de litere DAABC, acest număr este 5 : D este mai mare dacă patru litere din dreapta și E este mai mare decât o singură literă din dreapta sa. Această măsură este numită „numărul de inversiuni din secvență”. Secvența AACEDGG are doar o inversiune (E și D) și este aproape sortată, în timp ce secvența ZWQM are 6 inversiuni (număr maxim care poate fi!).

*Date de intrare :* în fișierul dna.in se află mai multe seturi de date. Pentru fiecare set de date, pe prima linie se află un număr întreg pozitiv  $n$  ( $0 < n \leq 50$ ), reprezentând lungimea sirurilor, și un număr întreg pozitiv  $m$  ( $0 < m \leq 100$ ), reprezentând numărul de siruri. Apoi urmează  $m$  linii, fiecare conținând un sir de lungime  $n$ .

*Date de ieșire :* în fișierul dna.out se vor scrie sirurile, pentru fiecare set de date, de la cel mai bine sortat până la cel mai puțin sortat.

*Exemplu :*

| dna.in     | dna.out    |
|------------|------------|
| 10 6       | CCCGGGGGGA |
| AACATGAAGG | AACATGAAGG |
| TTTGGCCAA  | GATCAGATT  |
| TTTGGCCAA  | ATCGATGCAT |
| GATCAGATT  | TTTTGGCCAA |
| CCCCGGGGGA | TTTGGCCAAA |
| ATCGATGCAT | AAGAT      |
| 5 3        | GATTA      |
| TTGGA      | TTGGA      |
| GATTA      |            |
| AAGAT      |            |

(ACM East Central Regionals, 1998)

### *Analiza problemei și proiectarea soluției*

Vom scrie funcția cu antetul :

```
int nr(string s)
```

care determină numărul de „inversiuni” din sirul dat s.

Funcția Prelucreaza() construiește o permutare p[] reprezentând ordinea cerută de problemă, aplicând o tehnică de tip *Bubble Sort*.

### *Program*

```
#include <stdio.h>
#include <string.h>

typedef char string[52];

FILE *fin, *fout;
int i, j, m, n;
int a[101], p[101];
string as[101];

int nr(string s){
 int i, n = 0;
 for(i=0; i<strlen(s)-1; i++)
 for(j=i+1; j<strlen(s); j++)
 if(s[i] > s[j]) n++;
 return n;
}
```

```
int Citeste(){
 if(fscanf(fin, "%d %d\n", &n, &m)>0){
 for(i=0; i<m; i++)
 fscanf(fin, "%s\n", as[i]);
 return 1;
 }
 else
 return 0;
}

void Prelucreaza(){
 int aux;
 for(i=0; i<m; i++)
 {p[i] = i; a[i] = nr(as[i]);}
 for(i=0; i<m-1; i++)
 for(j=i+1; j<m; j++)
 if(a[i] > a[j]){
 aux = a[i]; a[i] = a[j]; a[j] = aux;
 aux = p[i]; p[i] = p[j]; p[j] = aux;
 }
}

void Scrie(){
 for(i=0; i<m; i++)
 fprintf(fout, "%s\n", as[p[i]]);
}

void main(){
 fin = fopen("dna.in", "r");
 fout = fopen("dna.out", "w");
 while(Citeste()){
 Prelucreaza();
 Scrie();
 }
}
```

### *Problema 8. Cutii ascunse*

Considerăm o „cutie” n-dimensională dată de dimensiunile sale. În două dimensiuni, cutia (2, 3) poate reprezenta o cutie cu lungimea 3 și lățimea 2; în trei dimensiuni, cutia (4, 8, 9) poate reprezenta o cutie  $4 \times 8 \times 9$  (lățime, lungime, înălțime). Este mai greu să ne imaginăm o cutie în 6 dimensiuni (4, 5, 6, 7, 8, 9), dar putem opera cu proprietățile unei astfel de cutii. În această problemă vom analiza proprietățile unei mulțimi de cutii n-dimensionale. Trebuie să determinați cel mai lung sir de cutii

incluse una în alta, adică șirul de cutii  $b_1, b_2, \dots, b_k$  este un astfel de șir dacă cutia  $b_i$  începe în cutia  $b_{i+1}$  ( $1 \leq i < k$ ). Spunem că o cutie  $D = (d_1, d_2, \dots, d_n)$  începe în cutia  $E = (e_1, e_2, \dots, e_n)$  dacă există o reaaranjare a elementelor cutiei  $D$  astfel încât  $d_i < e_i, \forall i=1..n$ ; adică putem să contorsionăm în orice mod cutia  $D$ .

De exemplu, cutia  $D = (2, 6)$  începe în cutia  $E = (7, 3)$  deoarece  $D$  poate fi reaaranjată  $D = (6, 2)$  și fiecare dimensiune este mai mică decât dimensiunea corespunzătoare din  $E$ . Cutia  $D = (9, 5, 7, 3)$  nu începe în cutia  $E = (2, 10, 6, 8)$  deoarece nici o reaaranjare a lui  $D$  nu va satisface proprietatea de mai sus, dar  $F = (9, 5, 7, 1)$  începe în cutia  $E$  deoarece poate fi reaaranjată ca  $(1, 9, 5, 7)$ . Formal, definim *încaperea* astfel: cutia  $D = (d_1, d_2, \dots, d_n)$  începe în cutia  $E = (e_1, e_2, \dots, e_n)$  dacă există o permutare  $\pi$  a lui  $1..n$  astfel încât  $(d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(n)})$  se potrivește în  $(e_1, e_2, \dots, e_n)$ :  $d_{\pi(i)} < e_i, \forall i = 1..n$ .

*Date de intrare:* în fișierul *cutii.in* se găsește o serie de secvențe de cutii. Fiecare secvență de cutii începe cu o linie conținând numărul de cutii  $k$  din secvență urmat de dimensiunea cutiilor  $n$ . Pe fiecare dintre următoarele  $k$  linii se află cele  $n$  dimensiuni ale fiecărei cutii. Pot exista mai multe seturi de date de intrare și vor trebui procesate toate, pentru fiecare caz determinându-se cel mai lung șir de cutii ce încap una în alta. Dimensiunea maximă a cutiilor este 10 și cea minimă este 1, iar numărul maxim de cutii într-o secvență este 30.

*Exemplu:*

| cutii.in          | cutii.out |
|-------------------|-----------|
| 5 2               | 5         |
| 3 7               | 3 1 2 4 5 |
| 8 10              | 4         |
| 5 2               | 7 2 5 6   |
| 9 11              |           |
| 21 18             |           |
| 8 6               |           |
| 5 2 20 1 30 10    |           |
| 23 15 7 9 11 3    |           |
| 40 50 34 24 14 4  |           |
| 9 10 11 12 13 14  |           |
| 31 4 18 8 27 17   |           |
| 44 32 13 19 41 19 |           |
| 1 2 3 4 5 6       |           |
| 80 37 47 18 21 9  |           |

(ACM Internet Programming Contest, 1990)

### Analiza problemei și proiectarea soluției

Vom reține fiecare cutie ca pe un vector. Creăm funcții care ordonează un vector de întregi – *ordoneaza()* –, compară doi astfel de vectori – *cmp()* –, testează dacă o cutie începe în alta prin compararea pozițiilor vectorilor corespunzători presupuși ordonați, interschimbă conținuturile a doi vectori considerați de aceeași dimensiune – *intersch()* –, afișează recursiv o soluție – *rec()*. Având aceste instrumente la

îndemână, vom trece la rezolvarea problemei, care se reduce practic la o problemă clasică de *programare dinamică*: determinarea subșirului maximal. Va trebui să determinăm șirul de cutii care încap una în alta de lungime maximă.

Vom considera tehnica *programării dinamice înapoi*.

### Program

```
#include <stdio.h>

int i, j, n;
int a[101][11];
int p[12];
FILE *fout;

void ordoneaza(int *a, int n){
 int aux;
 for(i=0; i<n-1; i++)
 for(j=i+1; j<n; j++){
 if(a[i] > a[j]){
 aux = a[i];
 a[i] = a[j];
 a[j] = aux;
 }
 }
}

int cmp(int *a, int *b, int n){
 int rez;
 i=0;
 while(a[i] == b[i] && i<n) i++;
 if(i==n) rez = 0;
 else
 if(a[i] < b[i]) rez = 1;
 else rez = -1;
 return rez;
}

int incape(int *a, int *b, int n){
 int ok = 1;
 for(i=0; i<n; i++)
 if(a[i] >= b[i]) ok=0;
 return ok;
}
```

```

void intersch(int *a, int *b, int n){
 int aux;
 for(i=0; i<n; i++){
 aux = a[i];
 a[i] = b[i];
 b[i] = aux;
 }
}

void rec(int *a, int j){
 if(j!= -1){
 fprintf(fout, "%d ", p[j] + 1);
 rec(a, a[j]);
 }
}

void main(){
 int aux, k, n, i, j;
 int b[50], succ[50];
 FILE *fin = fopen("căutări.in", "r");
 fout = fopen("căutări.out", "w");
 while(fscanf(fin, "%d %d\n", &k, &n) > 0){
 for(i=0; i<k; i++){
 for(j=0; j<n; j++) fscanf(fin, "%d", &a[i][j]);
 ordoneaza(a[i], n);
 p[i]=i;
 fscanf(fin, "\n");
 }
 for(i=0; i<k-1; i++)
 for(j=i+1; j<k; j++)
 if(cmp(a[i], a[j], n) < 0){
 intersch(a[i], a[j], n); aux = p[i];
 p[i]=p[j]; p[j]=aux;
 }
 b[k-1]=1;
 succ[k-1] = -1;
 for(i=k-2; i>=0; i--){
 b[i] = 1;
 succ[i] = -1;
 for(j=i+1; j<k; j++)
 if((incăpe(a[i], a[j], n)) && (b[i] < b[j] + 1)){
 b[i] = b[j] + 1;
 succ[i] = j;
 }
 }
 }
}

```

```

 }
 if(succ[i]!=-1 && b[i]==b[j] + 1&&p[j]<p[succ[i]]) succ[i] = j;
 }
}
j=0;
for(i=1; i<n; i++) if(b[i]>b[j]) j=i;
fprintf(fout, "%d\n", b[j]);
rec(succ, j);
fprintf(fout, "\n");
}
}

```

### Exerciții

1. Enumerați motivele pentru care algoritmul cutiilor ascunse este unul de programare dinamică.
2. Rezolvați problema folosind metoda programării dinamice, varianta înainte.

### Problema 9. Numere fel de fel

Din articolul „Teoria Numerelor” în Microsoft Encarta 1994: „Dacă a, b și c sunt numere întregi astfel încât a = bc, a este numit multiplu al lui b sau al lui c, iar b și c sunt denumiți divizori sau factori ai lui a. Dacă c nu este +1 sau -1, b este numit divizor propriu al lui a. Întregii pari, care îl includ și pe 0, sunt multipli ai lui 2, de exemplu -4, 0, 2, 10; un întreg impar este un întreg care nu este par, de exemplu -5, 1, 3, 9. Un număr perfect este un număr pozitiv care este egal cu suma tuturor divizorilor proprii pozitivi; de exemplu, 6, este egal cu  $1 + 2 + 3$ , iar 28, egal cu  $1 + 2 + 4 + 7 + 14$ , sunt numere perfecte. Un număr pozitiv care nu este perfect este imperfect și este deficit sau abundență în funcție dacă suma divizorilor săi proprii pozitivi este mai mică sau mai mare decât numărul însuși. Astfel, 9, cu divizorii 1 și 3, este deficit; 12, cu divizorii 1, 2, 3, 4, 6, este abundență.”

Dacă fiind un număr, să se determine dacă este perfect, abundență sau deficit, ca în exemplul de mai jos.

*Exemplu:*

|                   |
|-------------------|
| PERFECTION OUTPUT |
| 15                |
| 15 DEFICIENT      |
| 28                |
| 28 PERFECT        |
| 6                 |
| 6 PERFECT         |
| 56                |
| 56 ABUNDENT       |
| 60000             |

```

60000 ABUNDENT
22
22 DEFICIENT
496
496 PERFECT
0
END OF OUTPUT

```

### Analiza problemei și proiectarea soluției

Vom scrie o funcție pentru calculul sumei divizorilor și o vom folosi în programul principal.

#### Program

```

#include <stdio.h>

long int sumDiv(long int n){
 long int i, s = 0;
 for(i=1; i<n; i++)
 if(n%i==0) s += i;
 return s;
}

void main(){
 long int n, s;
 printf("PERFECTION OUTPUT\n");
 while(scanf("%ld", &n)>0 && n>0){
 s = sumDiv(n);
 if(s == n)
 printf("%ld PERFECT\n", n);
 else
 if(s < n)
 printf("%ld DEFICIENT\n", n);
 else
 printf("%ld ABUNDENT\n", n);
 }
 printf("END OF OUTPUT");
}

```

#### Exercițiu

Dezvoltați programul astfel încât citirea/afisarea să se facă din/in fișier.

### Problema 10. Big Mod

#### Calculații

$$R = B^P \bmod M$$

pentru valori mari ale lui  $B$ ,  $P$  și  $M$  utilizând un algoritm eficient. Se dau trei valori întregi (în ordinea  $B$ ,  $P$ ,  $M$ ) care se vor citi căte una pe linie.  $B$  și  $P$  se încadrează între 0 și 2.147.483.647 inclusiv,  $M$  este un număr natural cuprins între 1 și 46.340 inclusiv. Se cere un singur număr întreg, rezultatul formulei de mai sus.

*Exemplu :*

| intrare (tastatură) | ieșire (ecran) |
|---------------------|----------------|
| 3                   | 13             |
| 18132               | 2              |
| 17                  | 13195          |
| 17                  |                |
| 1765                |                |
| 3                   |                |
| 2374859             |                |
| 3029382             |                |
| 36123               |                |

### Analiza problemei și proiectarea soluției

*Propoziție.*  $(a \cdot b) \bmod c = ((a \bmod c) * (b \bmod c)) \bmod c$ , pentru  $\forall a, b, c$  numere naturale pozitive.

*Demonstrație:* Aplicând teorema împărțirii cu rest, obținem succesiv:

$$\begin{aligned} \exists R \in \{0, \dots, c-1\} \text{ a.i. } a \cdot b = Q \cdot c + R \Rightarrow \\ R = (a \cdot b) \bmod c = a \cdot b - Q \cdot c \end{aligned} \quad (1)$$

$$\begin{aligned} \exists R_1 \in \{0, \dots, c-1\} \text{ a.i. } a = Q_1 \cdot c + R_1, R_1 = a \bmod c \\ \exists R_2 \in \{0, \dots, c-1\} \text{ a.i. } b = Q_2 \cdot c + R_2, R_2 = b \bmod c \Rightarrow a \cdot b = Q_1 \cdot c + R_1 \cdot R_2 \Rightarrow \\ (1) \Rightarrow (a \cdot b) \bmod c = (R_1 \cdot R_2) \bmod c = (a \bmod c \cdot b \bmod c) \bmod c \end{aligned}$$

#### Program

```

#include <stdio.h>

long int BigMod(long int B, long int P, long int M){
 if (P == 0) return 1%M;
 if (B == 0) return 0;
 if (B==1 || P == 1) return B%M;
 if (M==1) return 0;
}

```

```

if (P%2==0){
 long int aux;
 aux = BigMod(B, P/2, M);
 return (aux*aux)%M;
}
else
 return (BigMod(B, P-1, M) * (B%M)) % M;
}

void main(){
 long int B, P, M;
 while(scanf("%ld\n%ld\n%ld\n", &B, &P, &M) > 0)
 printf("%ld\n", BigMod(B, P, M));
}

```

### Exercițiu

Citirea/afișarea din/în fișier.

### Problema II. Numere umile

Numim *număr umil* un număr care are drept divizori primi doar pe 2, 3, 5 sau 7. Secvența 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27 ... reprezintă primele 20 de *numere umile*.

Scopul problemei este scrierea unui program C care să afișeze al n-lea element din această secvență.

*Date de intrare*: sunt unul sau mai multe numere naturale, scrise, câte unul pe o linie, în fișierul *umile.in*. Fiecare număr n este cuprins în intervalul [1, 542]:  $1 \leq n \leq 5842$ . Sirul datelor de intrare se termină cu o valoare 0 pentru n.

*Date de ieșire*: pentru fiecare caz de test din fișierul de intrare se va scrie în fișierul de ieșire *umile.out* un mesaj în limba engleză corespunzător: „The nth humble number is *number*.”. Depinzând de valoarea lui n, se va folosi sufixul corespunzător: st (dacă se termină în 1, dar nu este 11), nd (dacă se termină în 2, dar nu este 12), rd (dacă se termină în 3, dar nu este 13) sau th (altfel).

*Exemplu*:

| umile.in | umile.out                     |
|----------|-------------------------------|
| 1        | The 1st humble number is 1.   |
| 2        | The 2nd humble number is 2.   |
| 3        | The 3rd humble number is 3.   |
| 4        | The 4th humble number is 4.   |
| 11       | The 11th humble number is 12. |
| 12       | The 12th humble number is 14. |
| 13       | The 13th humble number is 15. |
| 21       | The 21st humble number is 28. |
| 22       | The 22nd humble number is 30. |

| umile.in | umile.out                               |
|----------|-----------------------------------------|
| 23       | The 23rd humble number is 32.           |
| 100      | The 100th humble number is 450.         |
| 1000     | The 1000th humble number is 385875.     |
| 5842     | The 5842nd humble number is 2000000000. |
| 0        |                                         |

(ACM International Collegiate Programming Contest, 1996-1997)

### Analiza problemei și proiectarea soluției

Pe baza definiției numerelor umile se va construi un tablou unidimensional a[] care le va conține.

### Program

```

#include <stdio.h>

long a[5842];

void buildArray(){
 long t, n2, n3, n5, n7;
 int i, j, k, l, m;
 t = 1;
 a[0] = 1; m = 0;
 n2 = 2; i = 0;
 n3 = 3; j = 0;
 n5 = 5; k = 0;
 n7 = 7; l = 0;
 while(m<5841){
 if(n2<=n3 && n2<=n5 && n2<=n7) t = n2;
 else
 if(n3<=n2 && n3<=n5 && n3<=n7) t = n3;
 else
 if(n5<=n2 && n5<=n3 && n5<=n7) t = n5;
 else t = n7;
 a[++m] = t;
 if(n2 == t) n2 = 2 * a[++i];
 if(n3 == t) n3 = 3 * a[++j];
 if(n5 == t) n5 = 5 * a[++k];
 if(n7 == t) n7 = 7 * a[++l];
 }
}

void main(){
 int n;
 FILE *fIn = fopen("umile.in", "r");
 FILE *fOut = fopen("umile.out", "w");

```

```

char suffix[4][3] = {"st", "nd", "rd", "th"};
buildArray();
while(fscanf(fIn, "%d\n", &n)==1&&n!=0){
 fprintf(fOut, "The %d", n);
 if((n/10)%10!=1)
 switch(n%10){
 case 1: fprintf(fOut, "%s", suffix[0]); break;
 case 2: fprintf(fOut, "%s", suffix[1]); break;
 case 3: fprintf(fOut, "%s", suffix[2]); break;
 default: fprintf(fOut, "%s", suffix[3]);
 }
 else
 fprintf(fOut, "th");
 fprintf(fOut, " humble number is %ld.\n", a[n-1]);
}
fclose(fIn); fclose(fOut);
}

```

### Exercițiu

Formulați și implementați o variantă generalizată a problemei.

### Problema 12. Moneda contrafăcută

Sally are 12 monede de argint dintre care doar 11 sunt adevărate și una este contrafăcută, deși nu se poate distinge printre cele adevărate. Moneda contrafăcută are o greutate diferită de celelalte, dar Sally nu știe dacă este mai ușoară sau mai grea. Din fericire, ea are un prieten ce posedă o balanță cu ajutorul căreia Sally poate afla care este moneda contrafăcută. Acest prieten îi permite fetei să facă trei măsurători. De exemplu, dacă Sally pune două monede pe un taler și pe celălalt altă două monede și tăierele stau perfect în echilibru, atunci ea trage concluzia că cele 4 monede sunt adevărate. În continuare, dacă Sally cântărește una dintre cele 4 monede egale cu o a cincea monedă și balanța nu stă în echilibru, atunci știm că această a 5-a monedă este contrafăcută și, depinzând de înclinarea balanței, știm și dacă este mai grea sau mai ușoară. Prin alegerea cu atenție a monedelor ce sunt cântărite, Sally este capabilă să o găsească pe cea falsă din exact trei măsurători.

*Date de intrare:* se găsesc în fișierul coin.in. Pe prima linie se găsește un număr pozitiv n ce se încadrează în tipul int,  $n > 0$ , specificând numărul de cazuri ce vor urma. Pieleare caz este format din trei linii, câte una pentru fiecare cântărire. Sally identifică cele 12 monede cu literele mari A-L. Informațiile referitoare la o cântărire sunt date de două siruri de litere dintre cele 12 și de unul dintre cuvintele up (sus), down (jos) sau even (egal). Primul sir reprezintă monedele de pe talerul stâng, al doilea sir reprezintă monedele de pe talerul drept (Sally va așeza tot timpul pe talerul stâng și pe cel drept același număr de monede). Cuvântul de pe a treia poziție spune dacă talerul drept este mai sus (up), mai jos (down) sau rămâne egal cu talerul stâng. Datele de intrare se consideră corecte.

*Date de ieșire:* pentru fiecare caz, în fișierul de ieșire coin.out trebuie să se scrie pe o linie o propoziție care să specifică moneda falsă, ca în exemplu. În totdeauna va exista o soluție unică. Moneda contrafăcută poate fi mai ușoară (light) sau mai grea (heavy).

### Exemplu:

| coin.in            | coin.out                                   |
|--------------------|--------------------------------------------|
| 8                  | B is the counterfeit coin and it is light. |
| A B up             | A is the counterfeit coin and it is heavy. |
| B A down           | A is the counterfeit coin and it is light. |
| A C even           | L is the counterfeit coin and it is heavy. |
| A B up             | A is the counterfeit coin and it is light. |
| B C even           | A is the counterfeit coin and it is heavy. |
| DEFG HIJL even     | L is the counterfeit coin and it is light. |
| ABC DEJ down       | K is the counterfeit coin and it is heavy. |
| ACH IEF down       |                                            |
| AHK IDJ down       |                                            |
| ABCD EFGH even     |                                            |
| AB IJ even         |                                            |
| A L down           |                                            |
| EFA BGH down       |                                            |
| EFC GHD even       |                                            |
| BA EF down         |                                            |
| A B up             |                                            |
| A C up             |                                            |
| L K even           |                                            |
| ACEGIK BDFHJL up   |                                            |
| ACEGIL BDFHJK down |                                            |
| ACEGLK BDFHJI down |                                            |
| ACEGIK BDFHJL up   |                                            |
| ACEGIL BDFHJK down |                                            |
| ACEGLK BDFHJI up   |                                            |
| 1                  | K is the counterfeit coin and it is light. |
| ABCD EFGH even     |                                            |
| ABC I EFJK up      |                                            |
| ABIJ EFGH even     |                                            |

(ACM East Central Regional Contest, 1998, enunț modificat)

### Analiza problemei și proiectarea soluției

Vom utiliza tablourile bidimensionale vLight[], vHeavy[] și vEqual[] cu câte 12 elemente corespunzătoare, în ordine, monedelor A-L. Pentru fiecare caz, inițializăm la început toate elementele celor trei tablouri cu 0. Vom folosi variabila nrNoEqual pentru a număra de câte ori balanța s-a înclina (implicit a conținut moneda falsă). Pentru fiecare cântărire, dacă balanța rămâne în echilibru, atunci se incrementează pozițiile corespunzătoare tuturor monedelor din acea cântărire în vectorul vEqual[]:

```

for(j=0; j<strlen(sRight); j++){
 vEqual[sRight[j]-'A']++;
 vEqual[sLeft[j]-'A']++;
}

```

Dacă mesajul este up (talerul drept mai sus), atunci această cântărire conține moneda falsă pe talerul drept (dacă este mai ușoară) sau pe talerul stâng (dacă este mai grea), deci vom incrementa corespunzător în cei doi vectori:

```
for(j=0; j<strlen(sRight); j++){
 vLight[sRight[j]-'A']++;
 vHeavy[sLeft[j]-'A']++;
}
```

Dacă mesajul este down, atunci se aplică procedura inversă:

```
for(j=0; j<strlen(sRight); j++){
 vLight[sLeft[j]-'A']++;
 vHeavy[sRight[j]-'A']++;
}
```

După ce am actualizat corespunzător cele trei vectori și variabila nrNoEqual, vom găsi moneda contrafecțată cu valoarea nrNoEqual corespunzătoare în vLight[] sau vHeavy[] și 0 în vEqual[]:

```
for(j=0; j<12; j++){
 if(vEqual[j]==0 && vLight[j]==nrNoEqual)
 fprintf(fOut,
 "%c is the counterfeit coin and it is light.\n", j+'A');
 if(vEqual[j]==0 && vHeavy[j]==nrNoEqual)
 fprintf(fOut,
 "%c is the counterfeit coin and it is heavy.\n", j+'A');
}
```

Una dintre cele două condiții subliniate este îndeplinită doar de o singură monedă!

### Program

```
#include <stdio.h>
#include <string.h>

void main(){
 long n;
 int i, nrNoEqual;
 unsigned j;
 int vLight[12], vHeavy[12], vEqual[12];
 char sLeft[7], sRight[7], sDescr[5];
 FILE *fIn = fopen("coin.in", "r");
 FILE *fOut = fopen("coin.out", "w");
 fscanf(fIn, "%ld\n", &n);
```

```
while(n--) {
 for(i=0; i<12; i++) {vLight[i]=vHeavy[i]=vEqual[i]=0;}
 nrNoEqual = 0;
 for(i=0; i<3; i++) {
 fscanf(fIn, " %s %s %s\n", sLeft, sRight, sDescr);
 if(strcmp(sDescr, "even")){
 nrNoEqual++;
 if(!strcmp(sDescr, "up"))
 for(j=0; j<strlen(sRight); j++){
 vLight[sRight[j]-'A']++;
 vHeavy[sLeft[j]-'A']++;
 }
 else
 for(j=0; j<strlen(sRight); j++){
 vLight[sLeft[j]-'A']++;
 vHeavy[sRight[j]-'A']++;
 }
 }
 else
 for(j=0; j<strlen(sRight); j++){
 vEqual[sRight[j]-'A']++;
 vEqual[sLeft[j]-'A']++;
 }
 }
 for(j=0; j<12; j++){
 if(vEqual[j]==0 && vLight[j]==nrNoEqual)
 fprintf(fOut,
 "%c is the counterfeit coin and it is light.\n", j+'A');
 if(vEqual[j]==0 && vHeavy[j]==nrNoEqual)
 fprintf(fOut,
 "%c is the counterfeit coin and it is heavy.\n", j+'A');
 }
 fclose(fIn); fclose(fOut);
}
```

### Exerciții

1. Considerați problema astfel încât Sally să poată face mai multe măsurători pentru un caz.
2. Considerați problema astfel încât să nu fim siguri că există întotdeauna soluție: există posibilitatea ca moneda contrafecțată să nu poată fi identificată.

**Problema 13. Numere unu**

Dat fiind un număr întreg  $1 \leq n \leq 10.000$  care nu se divide prin 2 sau 5, există un multiplu al numărului  $n$  care în notație zecimală este o secvență de 1. Câte cifre are cea mai scurtă astfel de reprezentare a unui multiplu al lui  $n$ ?

*Datele de intrare* se află în fișierul *unu.in*, care conține maximum 500 de cazuri de test, câte unul pe linie.

*Datele de ieșire* trebuie scrise în fișierul *unu.out*, câte o linie pentru fiecare caz, care să conțină numărul cazului, numărul dat  $n$  și numărul căutat de cifre în formatul de mai jos.

*Exemplu :*

| unu.in | unu.out                                |
|--------|----------------------------------------|
| 3      | caz # 1: n = 3 --> nr. cifre = 3       |
| 7      | caz # 2: n = 7 --> nr. cifre = 6       |
| 9901   | caz # 3: n = 9901 --> nr. cifre = 12   |
| 97     | caz # 4: n = 97 --> nr. cifre = 96     |
| 5673   | caz # 5: n = 5673 --> nr. cifre = 60   |
| 791    | caz # 6: n = 791 --> nr. cifre = 336   |
| 543    | caz # 7: n = 543 --> nr. cifre = 180   |
| 81     | caz # 8: n = 81 --> nr. cifre = 81     |
| 23     | caz # 9: n = 23 --> nr. cifre = 22     |
| 37     | caz # 10: n = 37 --> nr. cifre = 3     |
| 901    | caz # 11: n = 901 --> nr. cifre = 208  |
| 9999   | caz # 12: n = 9999 --> nr. cifre = 36  |
| 5551   | caz # 13: n = 5551 --> nr. cifre = 60  |
| 5557   | caz # 14: n = 5557 --> nr. cifre = 926 |

**Analiza problemei și proiectarea soluției**

Vom nota cu  $(\star k)$  numărul natural 11...11 cu  $k$  cifre de 1.

*Demonstrarea existenței.* Presupunem că avem un  $n$  fixat. Considerăm sirul de numere 1, 11, 111, ...,  $(\star k)$ , ... Conform principiului lui Dirichlet, există două astfel de numere care au același rest la împărțirea cu  $n$ , fie acestea  $(\star k_1)$  și  $(\star k_2)$  cu  $k_1 < k_2$ . Acest lucru se poate scrie formal:

$$\begin{aligned} (\star k_1) &= nQ_1 + R \\ (\star k_2) &= nQ_2 + R \end{aligned}$$

Făcând diferență, obținem:  $(\star k_2) - (\star k_1) = n(Q_2 - Q_1)$ . (2)

Dată fiind forma numerelor  $(\star k_2)$  și  $(\star k_1)$  ca succesiuni de cifre 1, deducem că numărul  $(\star k_2) - (\star k_1)$  are forma 11111110...0000 =  $(\star(k_2 - k_1))10^{k_1}$ . (3)

Cum  $n$  nu este divizibil cu 2 sau cu 5, rezultă că  $10^{k_1}$  divide pe  $(Q_2 - Q_1)$ , adică există un număr natural  $t$  astfel încât:

$$Q_2 - Q_1 = 10^{k_1} \times t \quad (4)$$

Înlocuind relațiile (3) și (4) în (2), obținem:

$(\star(k_2 - k_1))10^{k_1} = n \times 10^{k_1} \times t \Rightarrow (\star(k_2 - k_1)) = n \times t$ , cu alte cuvinte succesiunea de  $(k_2 - k_1)$  de 1 este un multiplu al lui  $n$ .

Vom nota cu  $C_{k-1}$ ,  $R_{k-1}$  cîtul, respectiv restul împărțirii lui  $(\star k-1)$  la  $n$ . Atunci, se poate scrie:  $(\star k-1) = C_{k-1} * n + R_{k-1}$ .

Scopul nostru este determinarea celui mai mic număr natural  $k$  cu proprietatea că  $R_k$  este 0.

Presupunând că  $R_{k-1}$  este cunoscut, putem determina ușor  $R_k$ , pe baza relației:

$$R_k = (R_{k-1} * 10 + 1) \% n, \text{ care se obține succesiv din:}$$

$$\begin{aligned} (\star k) &= 10 * (\star k-1) + 1 \Leftrightarrow C_k * n + R_k = 10(C_{k-1} * n + R_{k-1}) + 1 \Leftrightarrow R_k = n * (10 \\ C_{k-1} - C_k) + 10 * R_{k-1} + 1 &\Rightarrow R_k = (10 * R_{k-1} + 1) \% n. \text{ Pe baza acestor observații construim} \\ \text{un ciclu iterativ while, ne oprim cînd valoarea restului } R_k \text{ este 0 și afișăm valoarea } k+1. \end{aligned}$$

**Program**

```
#include <stdio.h>

void main(){
 int n, R, k;
 int caz = 0;
 FILE* fIn = fopen("unu.in", "r");
 FILE* fOut = fopen("unu.out", "w");
 while(!feof(fIn)){
 fscanf(fIn, "%d", &n);
 k = 0;
 R = n==1 ? 0 : 1;
 while(R){
 k++;
 R = (R*10+1)%n;
 }
 fprintf(fOut, "caz # %3d: n = %6d --> nr. cifre = %6d\n",
 ++caz, n, k+1);
 }
 fclose(fIn); fclose(fOut);
}
```

**Exerciții**

1. Presupunând că în fișierul de intrare sunt două numere naturale  $a, b < 10.000$ ,  $a < b$ , scrieți în fișierul de ieșire rezultatul pentru toate numerele naturale din intervalul  $[a, b]$  ca în problema de mai sus.

*Exemplu :*

| unu.in    | unu.out                                |
|-----------|----------------------------------------|
| 9030 9045 | caz # 1: n = 9031 --> nr. cifre = 820  |
|           | caz # 2: n = 9033 --> nr. cifre = 9030 |
|           | caz # 3: n = 9037 --> nr. cifre = 1290 |
|           | caz # 4: n = 9039 --> nr. cifre = 4290 |
|           | caz # 5: n = 9041 --> nr. cifre = 1130 |
|           | caz # 6: n = 9043 --> nr. cifre = 4521 |

- Modificați programul de mai sus astfel încât să determinați numărul natural din intervalul dat care are cele mai multe cifre de 1 în reprezentarea multiplului.
  - Rezolvați și problema inversă: dat fiind un număr de cifre  $k < 500$ , determinați cel mai mic număr natural  $n$  mai mic decât 10.000 cu proprietatea că este divizor al numărului format din  $k$  cifre de 1.
  - Demonstrați că orice număr natural are un multiplu care are numai cifrele 0 și 1.

#### *Problema 14. Codul secret*

Sarcofagul este închis folosind un cod numeric secret. Dacă cineva dorește să îl deschidă, trebuie să găsească codul și să îl scrie exact în celulele de pe capac. Un mecanism extrem de complicat se va declanșa pentru a deschide sarcofagul; în caz că respectivul cod este unul corect. Codul (alcătuit din cel mult 100 de numere întregi) a fost ascuns în Biblioteca din Alexandria, dar, din păcate, aceasta a ars complet. Din fericire, un arheolog necunoscut a obținut o copie a codului ce datează din secolul al VIII-lea. Fiindcă s-a cunoscut de „persoane răuvoitoare”, el a codificat numerele într-un mod foarte special. A considerat un număr complex aleatoriu  $B$  care era mai mare în modul decât toate numerele din cod. Apoi a considerat că aceste numere sunt cifrele dintr-un sistem numeric cu baza  $B$ . Aceasta înseamnă că secvența  $a_n, a_{n-1}, \dots, a_1, a_0$  a fost codificată ca un număr  $X = a_0 + a_1B + a_2B^2 + \dots + a_nB^n$ . Scopul problemei este decodificarea codului secret, adică pe baza numerelor complexe  $X$  și  $B$  date se cere determinarea „cifrelor” de la  $a_n$  până la  $a_0$ .

*Date de intrare:* se află în fișierul cod.in și este compus din mai multe instanțe. Pe prima linie a acestui fișier se află numărul T al acestora. Fiecare caz este o singură linie ce conține patru întregi  $X_r, X_i, B_r, B_i$  ( $|X_r|, |X_i| \leq 1.000.000, |B_r|, |B_i| \leq 16$ ). Aceste numere indică părțile reală, respectiv imaginară ale lui X și B, adică  $X = X_r + i \cdot X_i, B = B_r + i \cdot B_i$ . B este baza sistemului ( $|B| > 1$ ), X este numărul complex în care are loc reprezentarea codului.

*Date de ieșire:* programul trebuie să scrie în fișierul de ieșire "cod.out" câte o singură linie pentru fiecare caz de test. Linia trebuie să conțină cifrele  $a_n$ ,  $a_{n-1}$ , ...,  $a_1$ , separate prin virgulă. Trebuie să fie satisfăcute următoarele condiții:

- pentru  $i \in \{0, 1, 2, \dots, n\}$ :  $0 <= a_i < |B|$
  - $X = a_0 + a_1 B + a_2 B^2 + \dots + a_n B^n$
  - dacă  $n > 0$ , atunci  $a_n < > 0$
  - $n \leq 100$

Dacă nu există numere care să satisfacă acest criteriu, atunci se va scrie în linia corespunzătoare din fisierul de ieșire propozitia „Nu există cod.”.

*Exemplu:*

| cod. in           | cod. out                  |
|-------------------|---------------------------|
| -262144 0 0 2     | 8, 11, 7, 3               |
| 6222 -16809 -2 13 | 2, 0, 2, 1, 1, 2, 1, 2, 1 |
| 7973 -1108 2 2    | 15, 3, 8                  |
| 3560 -1908 -16 4  | Nu existe cod.            |
| 28443 44463 -6 14 | 10, 1, 4, 3               |
| -11425 5080 4 -10 | 3, 5, 0                   |
| -8 -210 5 -6      |                           |

### *Analiza problemei si proiectarea solutiei*

Notăm cu  $C_z$  mulțimea numerelor complexe cu părțile reală și imaginară întregi:

$$X = a_0 + a_1B + a_2B^2 + \dots + a_nB^n = a_0 + B(a_1 + a_2B + \dots + a_nB^{n-1})$$

$$\Leftrightarrow X - a_0 = B(a_1 + a_2B + \dots + a_nB^{n-1}) \quad \quad \quad (1)$$

Deoarece numerele  $a_n, a_{n-1}, \dots, a_1, a_0$  sunt numere naturale și  $B \in C_z$ , rezultă că există numerele întregi  $P$  și  $O$  care satisfac condiția:

$$a_1 + a_2 B + \dots + a_n B^{n-1} = p + i \cdot q$$

*Propoziție.* Dacă numărul  $a_0$  cu proprietatea (1) și cu proprietatea  $0 \leq a_0 < |B|$  (2) există, atunci este *unic* cu aceste proprietăți.

*Demonstrație.* Vom folosi metoda reducerii la absurd. Presupunem că  $\exists a_n, a'_n$  diferite care să satisfacă (1) și (2). Atunci  $\exists \phi, \phi' \in C$  diferite astfel încât:

$$X = a_0 + B \cdot Q_0 \quad (3)$$

Făcând diferența, obținem:  $a_1 - a_2 = B \cdot (Q_1 - Q_2)$ , de unde rezultă

$$|a_0 - a_p| = |B| \cdot |(\Omega_0 + \Omega_p)| \quad (4)$$

Deoarece  $a_i, a'_i \in \{0, 1, \dots, |B| - 1\}$  rezultă că  $|a_i - a'_i| \leq |B|$

Deoarece  $Q_0, Q'_0 \in C_2$  sunt diferite, rezultă că  $|Q'_0 - Q_0| \geq 1$ , adică  $|B| \cdot |Q'_0 - Q_0| \geq |B|$ . Din aceste ultime două observații rezultă:

$$|a_0 - \bar{a}_0| < |B| \cdot |\langle Q_0 - \bar{Q}_0 \rangle| \quad (5)$$

Contradicție (4+5). Deci presupunerea făcută este falsă și  $a_6$  cu proprietățile (1) și (2) este unic.

*Dezvoltarea algoritmului.* Pe baza propoziției anterioare, deducem că, dacă un cod pentru perechea  $(X, B)$  dată există, atunci acesta este *unic*. Algoritmul de rezolvare a problemei este unul iterativ, în care se determină, pe rând, „cifrele”  $a_0, a_1, \dots, a_n$ . Pentru perechea  $(X, B)$  dată, determinarea lui  $a_0$  se va face prin parcurgerea valorilor posibile  $0, 1, 2, \dots, |B| - 1$  și verificarea dacă  $X - a_0$  este *multiplu* al lui  $B$ , adică  $X - a_0$  se poate scrie ca  $B \cdot Q$ , unde  $Q \in C_0$ . Dacă da, atunci acesta este unic pe

baza propoziției anterioare și se adaugă în vectorul cu soluția parțială posibilă. Q devine noul x și se continuă în aceeași manieră până la o condiție de oprire, care poate fi:

- x devine zero (am determinat o codificare);
- în încercarea de a-l determina pe  $a_1$  nu există nici o valoare posibilă dintre 0, 1, 2, ...,  $|B| - 1$  cu proprietatea că  $x - a_1$  este „multiplu” de B (nu există codificare);
- am construit  $a_0, a_1, \dots, a_{100}$  și x este nevid (nu există codificare, problema cere  $n \leq 100$ ).

**Multiplu.** În continuare, vom analiza cum se poate decide dacă un număr complex z este *multiplu* al numărului complex nenul B din mulțimea  $C_Z$ .

Presupunem că  $Z, B \in C_Z$ ,  $B \neq 0$ . Z este *multiplu* al lui B din  $C_Z$  dacă și numai dacă

$$\exists Q \in C_Z \text{ astfel încât } Z = B \times Q \Leftrightarrow \frac{Z}{B} = Q \Leftrightarrow \frac{Z_r + iZ_i}{B_r + iB_i} = Q \Leftrightarrow \frac{(Z_r + iZ_i) \cdot (B_r - iB_i)}{B_r^2 + B_i^2} = Q$$

$$\Leftrightarrow \frac{Z_r \cdot B_r + Z_i \cdot B_i + i(Z_r \cdot B_i - Z_i \cdot B_r)}{B_r^2 + B_i^2} = Q. \text{ Această ultimă egalitate este echivalentă cu faptul că numerele întregi } Z_r \cdot B_r + Z_i \cdot B_i \text{ și } Z_r \cdot B_i - Z_i \cdot B_r \text{ sunt divizibile cu } B_r^2 + B_i^2.$$

### Program

```
#include <stdio.h>
#include <math.h>

#define MSG_ERR "Nu există cod.\n"

typedef struct compl{
 long re, im;
}Complex;

double Modul(Complex z){
 return sqrt((long)(z.re*z.re + z.im*z.im));
}

short esteZero(Complex z){
 return (z.re == 0 && z.im == 0?1:0);
}

short divComplex(Complex *z, Complex u){
 long r = u.re;
 long i = u.im;
 long m2 = r*r + i*i;
 long tempre;
 if((z->re*r+z->im*i)%m2!=0 || (z->im*r-z->re*i)%m2 != 0)
 return 0;
 tempre = z->re;
 z->re = (z->re*r + z->im*i) / m2;
```

```
z->im = (z->im*r - i*tempre) / m2;
return 1;
}

void ScrieSolutia(int a[], int n, FILE *f){
 int i;
 if(n == 1) fprintf(f, "%ld", a[0]);
 for(i=n-1; n>i && i>=0; i--){
 fprintf(f, "%d ", a[i]);
 }
 fprintf(f, "\n");
}

void Proceseaza(Complex X, Complex B, FILE *f){
 int a[101], n = 0;
 double temp = Modul(B);
 int aMax = (int) temp;
 Complex XNou;
 short i, j;
 if(temp-aMax == 0) aMax--;
 if(esteZero(X)) a[n++]=0;
 for(j=0; (j<100) && (!esteZero(X)); j++){
 short gasit = 0;
 for(i=0; i<=aMax; i++){
 XNou.re = X.re - i;
 XNou.im = X.im;
 if(divComplex(&XNou, B)){
 a[n++] = i;
 X = XNou;
 gasit = 1;
 break;
 }
 }
 if(!gasit){fprintf(f, MSG_ERR); break;}
 }
 if(esteZero(X)){ScrieSolutia(a, n, f);}
 else if(j == 100){fprintf(f, MSG_ERR);}
}

void main(){
 long i, n;
 Complex X, B;
 FILE *fIn = fopen("cod.in", "r");
 FILE *fOut = fopen("cod.out", "w");
```

```

fscanf(fIn, "%ld", &n);
for(i=0; fIn && !feof(fIn) && i<n; i++){
 fscanf(fIn, "%ld%ld%ld%ld", &X.re, &X.im, &B.re, &B.im);
 Proceseaza(X, B, fOut);
}
fclose(fIn); fclose(fOut);
}

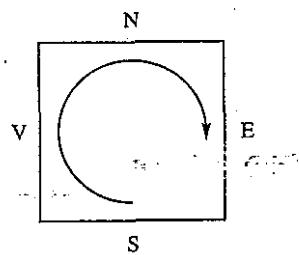
```

### Exercițiu

Rezolvați și problema inversă: pe baza numerelor  $a_0, a_{n-1}, \dots, a_1, a_0$  și a numărului complex B, determinați x cu condiția  $x = a_0 + a_1B + a_2B^2 + \dots + a_nB^n$ .

### Problema 15. Împărțirea cărților la Bridge

Multe dintre jocurile de cărți presupun distribuirea tuturor 52 de cărți din pachet la patru jucători. Mulți dintre jucători aranjează apoi într-o anumită ordine cărțile în mână. Vom presupune denumirea din limba engleză a cărților și ordinea: clubs (trefle) < diamonds (carouri) < spades (pici) < hearts (cupe), codificate cu literele corespunzătoare C, D, S și H. Deoarece asul este cel mai mare, valorile cărților vor fi  $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < J < Q < K < A$  (codificăm 10 cu T pentru o scriere unitară a tuturor cărților). Jucătorii sunt de obicei denumiți Nord, Sud, Est și Vest și ei sunt așezăți ca punctele cardinale:



Unul dintre jucători este *dealerul*, însemnând că acesta împarte cărțile din pachet câte una, începând cu cel din stânga să și în direcția acelor de ceasornic. Problema noastră este scrierea unui program care să simuleze împărțirea unui pachet dat în fișierul de intrare, ordonează conținutul fiecărei mâini și scrie într-un alt fișier cărțile primite de fiecare jucător în formatul prezentat mai jos.

*Date de intrare*: fișierul de intrare bridge.in va conține un set de date compus din litera desemnându-l pe cel care împarte cărțile și pachetul în ordine. Fișierul se va încheia cu caracterul #.

*Date de ieșire*: fișierul de ieșire bridge.out va conține 4 linii reprezentând mâinile ordonate, în ordinea și formatul prezentate mai jos.

Exemplu :

|                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bridge.in                                                                                                                                                                        |
| E<br>CQDTIC4D8S7HTDAH7D2S3D6C6S6D9S4SAD7H2CKH5D3CTS8C9H3C3<br>DQS9SQDJH8HAS2SKD4H4S5C7SJC8DKC5C2CAHQCJSTH6HKH9D5HJ<br>#                                                          |
| bridge.out                                                                                                                                                                       |
| S: CQ D2 D3 D7 DK S2 S5 S6 S7 SQ H3 HQ HK<br>W: C3 C5 C7 CT CJ D9 DT DJ S3 SK H2 H9 HT<br>N: C2 C4 CK D4 D5 D6 DQ DA S4 S8 ST SJ H8<br>E: C6 C8 C9 CA D8 S9 SA H4 H5 H6 H7 HJ HA |

(ACM Northwestern European Regionals Contest, 1992)

### Analiza problemei și proiectarea soluției

Vom defini structura TCarte ce reprezintă numărul și culoarea (elemente de tip char) unei cărți, deci un element de acest tip definește complet o carte din pachet. În scopul rezolvării problemei vom defini o serie de metode ajutătoare, de exemplu valoareValida() și culoareValida() cu parametri de tip char care decid dacă respectivul caracter poate defini valoarea, respectiv culoarea unei cărți. În acest scop vom utiliza macrourile predefinite VALORI\_CARTI și CULORI\_CARTI, precum și funcția strchr() din biblioteca string.h, care determină apariția unui caracter într-un sir de caractere. Metoda citesteCarti() execută citirea cărților din fișierul dat ca parametru și completarea carte cu carte a variabilei globale pachet. Observăm că pentru fiecare carte se va aloca spațiu necesar în memorie utilizând malloc(). Acest pachet va trebui împărțit celor NR\_JUCATORI. În acest scop se folosește variabila globală mana[ ][], care este un tablou bidimensional definit astfel:

TCarte mana[NR\_JUCATORI][NR\_CARTI/NR\_JUCATORI + 1];

ce va conține modul de împărțire a cărților. Împărțirea și ordonarea cărților fiecărui jucător se va realiza în cadrul metodei imparte(), astfel că după distribuția cărților (cartea i va reveni jucătorului  $i \% NR\_JUCATORI$  și este a  $i / NR\_JUCATORI$ -a carte distribuită acestuia). În cadrul metodei imparte() se va efectua de asemenea și ordonarea cărților fiecărui jucător folosind funcția qsort() din biblioteca stdlib.h. Pentru a compara două cărți date scriem funcția cartiCmp(), care are ca parametri două elemente de tip void\* (reprezentând două cărți) și returnează o valoare negativă dacă prima este mai mică decât a doua, zero dacă cele două cărți sunt egale și o valoare pozitivă dacă a două carte este mai mică decât prima. Această metodă cartiCmp() va fi utilizată ca parametru pentru qsort(). Scrierea rezultatului se va realiza utilizând metoda scrie() având drept parametru fișierul de ieșire. Vom folosi variabila globală JUCATORI[], ce conține numele jucătorilor în ordinea acelor de ceasornic și k va decide indicele primului jucător care primește cărțile.

*Program*

```

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>

#define NR_JUCATORI 4
#define NR_CARTI 52
#define VALORI_CARTI "23456789TJQKA"
#define CULORI_CARTI "CDSH"

char JUCATORI[4] = { 'N', 'E', 'S', 'W' };

typedef struct{
 char nr;
 char cul;
}TCarte;

char jucStart;
TCarte pachet[NR_CARTI];
TCarte mana[NR_JUCATORI][NR_CARTI/NR_JUCATORI + 1];

short valoareValida(char ch){
 short rez = 0;
 if(strchr(VALORI_CARTI, ch) != NULL) rez = 1;
 return rez;
}

short culoareValida(char ch){
 short rez = 0;
 if(strchr(CULORI_CARTI, ch) != NULL) rez = 1;
 return rez;
}

void citeste(){
 char ch;
 unsigned short i=0;
 char ct[2];
 short pIdx = 0;
 FILE *f = fopen("bridge.in", "r");
 if(!f) printf(" Fisierul de intrare inexistent! ");
 if(f&&!feof(f)){
 fscanf(f, "%c", &jucStart);
 }
 }

```

```

while(f && !feof(f)){
 fscanf(f, "%c", &ch);
 if(i==0 && culoareValida(ch)) (ct[i++]=ch);
 else if(i==1 && valoareValida(ch)) (ct[i++]=ch);
 if(i == 2){
 TCarte * carte=(TCarte*) malloc(sizeof(carte));
 carte->cul = ct[0];
 carte->nr = ct[1];
 pachet[pIdx++] = *carte;
 i = 0;
 }
 fclose(f);
}

int cartiCmp(const void *c1, const void *c2){
 TCarte *aux1 = (TCarte*) c1;
 TCarte *aux2 = (TCarte*) c2;
 char *ptr1, *ptr2;
 ptr1 = strchr(CULORI_CARTI, aux1->cul);
 ptr2 = strchr(CULORI_CARTI, aux2->cul);
 if(ptr1 && ptr2 && ptr1-ptr2) return (ptr1 - ptr2);
 ptr1 = strchr(VALORI_CARTI, aux1->nr);
 ptr2 = strchr(VALORI_CARTI, aux2->nr);
 if(ptr1 && ptr2) return (ptr1 - ptr2);
 return -100;
}

void imparte(){
 int i;
 int n = NR_CARTI/NR_JUCATORI;
 if(sizeof(pachet) < NR_CARTI){
 printf("Pachet incomplet!");
 return;
 }
 for(i=0; i<NR_CARTI; i++){
 TCarte carte = pachet[i];
 mana[i%NR_JUCATORI][i/NR_JUCATORI] = carte;
 }
 for(i=0; i<NR_JUCATORI; i++){
 qsort((void*)mana[i], n, sizeof(TCarte), cartiCmp);
 }
}

```

```

void scrie()
{
 FILE *f = fopen("bridge.out", "w");
 int i, j;
 char *ptrChar = strchr(JUCATORI, jucStart);
 int k = ptrChar - JUCATORI + 1;
 if(f){
 for(i=0; i<NR_JUCATORI; i++){
 fprintf(f, "%c: ", JUCATORI[k%4]);
 for(j=0; j<NR_CARTI/NR_JUCATORI; j++){
 TCarte carte = mana[i][j];
 fprintf(f, "%c%c ", carte.cul, carte.nr);
 }
 fprintf(f, "\n"); k++;
 }
 fclose(f);
 }

 void main(){
 citeste();
 imparte();
 scrie();
 }
}

```

### Exercitii

- Transformați programul presupunând că în fișierul de intrare se află mai multe seturi de date.
- Presupunând că pachetul nu mai este dat în fișierul de intrare, scrieți o metodă care amestecă (utilizând rand()) un pachet dat, afișează configurația inițială a acestuia și apoi efectuează distribuirea cărților ca în exemplul de mai sus. Scrieți, de asemenea, un program care generează un fișier de intrare pe această bază (cu mai multe seturi de date).
- Tinând cont de faptul că un J are valoarea 1 punct, Q – 2 puncte, K – 3 puncte și un as A – 4 puncte, calculați pentru fiecare jucător punctajul inițial și scrieți-l în fișierul de ieșire.

### Probleme propuse

- Cuburi perfecte.** De sute de ani, ultima *Teoremă a lui Fermat*, care afirmă că nu există trei numere întregi  $a, b, c > 1$  și un număr natural  $n > 2$  astfel încât  $a^n + b^n = c^n$ , a rămas nedemonstrată. Au existat diverse încercări de a demonstra acest lucru și toate au eşuat. Dar este posibil să găsim întregi mai mari decât 1 care satisfac *ecuația cubului perfect*:  $a^3 = b^3 + c^3 + d^3$  (de exemplu,  $12^3 = 6^3 + 8^3 + 10^3$  este adeverată). Problema cere scrierea unui program care să afișeze în fișierul *cuburi.out*

toate mulțimile de numere  $\{a, b, c, d\}$  din mulțimea  $\{1, 2, \dots, 100\}$  care satisfac această ecuație sub formă:

Cubul = 6, Triplet = (3, 4, 5)  
 Cubul = 12, Triplet = (6, 8, 10)  
 Cubul = 18, Triplet = (2, 12, 16)  
 ...  
 Cubul = 99, Triplet = (11, 66, 88)  
 Cubul = 100, Triplet = (16, 68, 88)  
 Cubul = 100, Triplet = (35, 70, 85)

(ACM South Central USA Collegiate Programming)

- Factori primi în factorial.** Factorialul unui număr se definește ca fiind produsul tuturor numerelor întregi de la 1 la n. Factorialele cresc foarte repede, de exemplu  $5! = 120$  și  $10! = 3.628.800$ . Una dintre modalitățile de reprezentare a factorialelor este aceea prin expoziții numerelor prime pe care le conțin. Scrieți un program care determină această reprezentare a numărului n.

Datele de intrare se citesc din fișierul *fact.in*, care va contine o serie de numere naturale, câte unul pe linie ( $2 \leq n \leq 500$ ), ultima linie conținând numărul 0.

Datele de ieșire se scriu în fișierul *fact.out*, unde se va scrie câte un bloc de linii pentru fiecare linie din datele de intrare. Fiecare bloc va începe cu numărul n, aliniat la dreapta într-un câmp de dimensiune 4, urmat de caracterul „!”, spațiu și caracterul „=”. Acesta va fi urmat de o listă a expozițiilor nenule ai numerelor prime considerate în ordine crescătoare (2, 3, 5, 7, 11, 13, 17 ...). Acestea trebuie să fie aliniate la dreapta în câmpuri de dimensiune 4 și fiecare linie din bloc (exceptând-o pe ultima, care poate avea mai puține caractere) trebuie să conțină 15 numere. Toate liniile după prima trebuie să fie indentate, ca în exemplul de mai jos. Între două blocuri se va scrie o linie vidă.

Exemplu :

| fact.in | fact.out                                                          |
|---------|-------------------------------------------------------------------|
| 123     | $123! = 117\ 59\ 28\ 19\ 12\ 9\ 7\ 6\ 5\ 4\ 3\ 3\ 3\ 2\ 2$        |
| 56      | 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1                                     |
| 5       |                                                                   |
| 78      | $56! = 53\ 26\ 13\ 9\ 5\ 4\ 3\ 2\ 2\ 1\ 1\ 1\ 1\ 1\ 1$            |
| 345     | 1                                                                 |
| 500     |                                                                   |
| 12      | $5! = 3\ 1\ 1$                                                    |
| 0       |                                                                   |
|         | $78! = 74\ 36\ 18\ 12\ 7\ 6\ 4\ 4\ 3\ 2\ 2\ 2\ 1\ 1\ 1$           |
|         | 1 1 1 1 1 1                                                       |
|         | $345! = 340\ 170\ 84\ 57\ 33\ 28\ 21\ 18\ 15\ 11\ 11\ 9\ 8\ 8\ 7$ |
|         | 6 5 5 4 4 4 4 3 3 3 3 3 3 3 3                                     |
|         | 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1                                     |
|         | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1                                     |
|         | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1                                     |

(ACM North Central Region USA, 1993)

3. *Lizuca și bunicul.* Într-o dimineată de mai, Lizuca și bunicul său se aflau în prisacă, să îngrijească albinele. Lizuca a constatat că faguri de miere au anumite proprietăți: ei sunt aşezăți pe coloane, fiecare coloană este ridicată sau coborâtă pentru a se potrivi; în unele celule din fagure se află mărci, există maximum 50 de coloane și fiecare coloană are maximum 20 de celule. Și a mai observat că pe fiecare coloană pot să existe maximum 3 mărci. În timp ce mâncă cu poftă dintr-un fagure auriu, fetița s-a gândit să transforme fiecare fagure într-un sir de numere și litere astfel încât, pe baza acestui sir, să poată „ține minte” fagurile. Iată cum a procedat: primele nC numere din sir reprezintă numerele de celule de pe fiecare coloană, apoi urmează litera c sau R, după cum prima coloană este coborâtă, respectiv ridicată, apoi urmează perechi de numere care reprezintă coloana, respectiv numărul celulei în cadrul coloanei (de sus în jos).

$n_1 \ n_2 \ \dots \ n_k \ Ch \ p_{11} \ p_{12} \ p_{21} \ p_{22} \ p_{31} \ p_{32} \ \dots \ p_{t1} \ p_{t2}$

$n_1$  reprezintă numărul de celule de pe linia  $i$ ,  $Ch$  este unul dintre caracterele C sau R,  $p_{j_1} \ p_{j_2}$  reprezintă o pereche de numere care semnifică poziția unei mărci în fagure ( $p_{j_1} =$  coloana,  $p_{j_2} =$  numărul celulei considerate de sus în jos). Lizuca i-a explicat lui Patrocle cum se poate transforma un fagură, așa că ei au început să se joace „de-a fagură” : unul propunea sirul de caractere corespunzător, iar celălalt trebuia să deseneze pe pământul moale fagurile ; matca trebuie reprezentată prin caracterul Q. Iar tu, ca șicusit programator, va trebui să reconstruiești fagurile folosind doar caracterele \ (slash), / (backslash), \_ (underscore), Q și spațiu. Pentru un sir citit de la tastatură, fagurile se va afișa pe ecran și se va adăuga în fișierul fagure.out, împreună cu sirul dat.

*Exemple :*

### Codificările pentru:

Ex. 1: 4 6 5 4 7 6 R 1 1 1 4 2 3 2 5 3 4 4 2 5 1 5 4 6 2 6 5  
 Ex. 2: 6 2 8 3 1 9 C 1 2 1 6 2 2 3 5 3 8 4 2 5 1 6 3 6 6 6 9  
 Ex. 3: 2 3 1 4 R 1 1 2 2 3 1 4 2 4 4  
 Ex. 4: 5 4 6 5 9 7 4 10 C 1 1 1 3 1 5 2 1 2 2 2 4 3 2 3 5 4 3  
 Ex. 5: 7 5 9 6 2 6 6 7 1 7 8 5 8 4 8 6

(ACM East European Regional, 1992, enunț modificat)

4. *Albă-ca-Zăpada pe Internet*. Albă-ca-Zăpada a descoperit Internetul. În urma călătoriilor sale pe nesfârșite liste de discuții, un print, cam hacker de felul lui, i-a trimis un joc de cuvinte, foarte asemănător cu celebrul Scrabble. Și printul i-a zis Albeii-ca-Zăpada că va veni să o salveze la nevoie doar dacă va juca acest joc perfect și va obține tot timpul punctaj maxim.

Iată cum se desfășoară jocul: printul îi trimite un rebus sub formă de tablă pătratică cu  $N \times N$  căsuțe care conțin câte o literă din alfabetul englez sau caracterul punct „.”. Literele formează pe orizontală sau pe verticală cuvinte delimitate prin succesiuni de caractere punct sau prin marginile tablei. Tabla este vizibilă pe ecran un interval de timp, după care aceasta dispăr. Albă-ca-Zăpada trebuie să trimită drept răspuns cât mai rapid cuvinte distincte, *de valoare maximă*, în ordine alfabetică, câte unul pe linie. Punctajul se calculează ca suma codurilor ASCII ale literelor distincte care

apar în soluție. Dacă nu există astfel de cuvinte, punctajul este 0. Un *cuvânt de valoare maximă* este acela care îndeplinește simultan condițiile:

- este palindrom (citindu-l de la dreapta la stânga sau de la stânga la dreapta se obține același cuvânt);
- are lungime maximă relativ la alte cuvinte cu proprietatea de a fi palindrom. Cuvintele sunt scrise cu litere mari și lungimea cuvintelor din soluție trebuie să fie cel puțin 2.

Se cere așadar scrierea unui program care să determine, pentru un careu dat, punctajul maxim și secvența de cuvinte care permite obținerea punctajului maxim. Dacă nu există o astfel de secvență se va afișa valoarea 0.

*Datele de intrare* se citesc din fișierul *rebus.in* cu următoarea structură:

```
n // dimensiunea careului
a11a12...a1n // elementele careului (fără să fie despărțite printr-un spațiu)
a21a22...a2n
...
an1an2...ann
```

*Date de ieșire*: în fișierul *raspuns.out* se vor scrie: pe prima linie, numărul de cuvinte de valoare maximă găsite, pe următoarele linii, aceste cuvinte în ordine alfabetică, câte unul pe linie, iar pe ultima linie se va scrie punctajul.

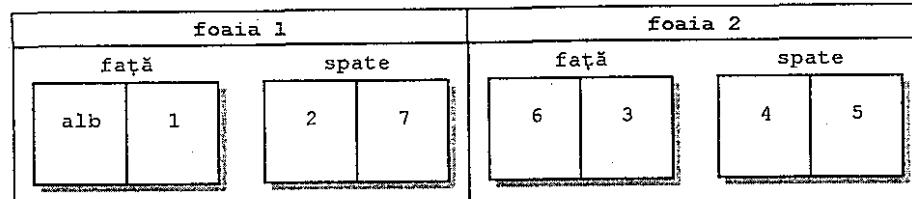
*Exemplu*:

| rebus.in   | raspuns.out |
|------------|-------------|
| 3          | 0           |
| A.A        | 0           |
| B.A        |             |
| ..C        |             |
| 5          | 0           |
| .....      | 0           |
| .....      |             |
| .....      |             |
| 10         | 5           |
| VERDE.CRIN | ABCBA       |
| DANA.DOROD | AOROA       |
| VOI.AROA.  | DOROD       |
| NU.ABOTE.. | EOTCE       |
| .UCOCU.AIA | EZAZE       |
| EZNGBIETAR | IALAI       |
| ORIOATZULI | NARAN       |
| TURN.NARAN | UCOCU       |
| O.ZOBOZ.IU | ZOBOZ       |
| ELEGATEL.O | 982         |

*Observație*: 982 =  

$$\text{cod('A')} + \text{cod('B')} + \text{cod('C')} + \text{cod('D')} + \text{cod('E')} + \text{cod('I')} + \text{cod('L')} + \text{cod('N')} + \text{cod('O')} + \text{cod('R')} + \text{cod('T')} + \text{cod('U')} + \text{cod('Z')}$$

5. *Listarea broșurilor*. În mod normal, când se tipărește un document, este tipărită mai întâi prima pagină, apoi a doua, a treia etc. Dar atunci când dorim să listăm o broșură cu câteva foi îndoite la mijloc, ordinea de listare se modifică. O astfel de broșură are patru pagini pe foaie, cu două pe față și două pe spate. Când așezăm toate foi în ordine și îndoim teancul în două, paginile apar în ordinea corectă, ca într-o carte obișnuită. De exemplu, o broșură cu 7 pagini se va tipări:



Problema este scrierea unui program care, având la intrare numărul de pagini ce trebuie tipărite, să genereze ordinea de tipărire.

*Datele de intrare* se găsesc în fișierul *brosura.in* și constau într-o serie de numere naturale, câte unul pe linie a fișierului și fiecare nu va depăși 100. Numărul 0 indică încheierea analizării cazurilor. Pentru fiecare caz de test, se va scrie în fișierul de ieșire *brosura.out* cum vor fi listate paginile, la fel în exemplul de mai jos. Dacă numărul de pagini dorit nu se încadrează complet în foaie, atunci se va scrie cuvântul „alb” în loc de număr. Dacă față sau spatele unei pagini sunt complet goale, atunci nu se va scrie nimic pentru acea parte de foaie. *Datele de ieșire* trebuie să fie în ordine crescătoare, de la prima spre ultima foaie, mai întâi față, apoi spatele.

*Exemplu*:

| brosura.in | brosura.out                                                                                                                                                                                                                                                                        |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 18         | Ordinea de listare pentru 7 pagini:<br>Foaia 1 fata: alb, 1<br>Foaia 1 spate: 2, 7<br>Foaia 2 fata: 6, 3<br>Foaia 2 spate: 4, 5                                                                                                                                                    |
| 4          | Ordinea de listare pentru 18 pagini:<br>Foaia 1 fata: alb, 1<br>Foaia 1 spate: 2, alb<br>Foaia 2 fata: 18, 3<br>Foaia 2 spate: 4, 17<br>Foaia 3 fata: 16, 5<br>Foaia 3 spate: 6, 15<br>Foaia 4 fata: 14, 7<br>Foaia 4 spate: 8, 13<br>Foaia 5 fata: 12, 9<br>Foaia 5 spate: 10, 11 |
| 1          | Ordinea de listare pentru 4 pagini:<br>Foaia 1 fata: 4, 1<br>Foaia 1 spate: 2, 3                                                                                                                                                                                                   |
| 0          | Ordinea de listare pentru 1 pagini:<br>Foaia 1 fata: alb, 1                                                                                                                                                                                                                        |

6. *Numerere frumoase.* Numerele frumoase sunt numerele care au ca factori primi doar pe 2, 3 și 5. Sirul numerelor frumoase este:

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, ...

*Cerință:* Dat fiind un număr natural  $n$  ( $1 \leq n \leq 1.500$ ), afișați pe ecran primele  $n$  numere frumoase.

*Exemplu:*

*Intrare:*

$n=50$

*Ieșire:*

1 2 3 4 5 6 8 9 10 12 15 16 18 20 24 25 27 30 32 36 40 45 48 50  
54 60 64 72 75 80 81 90 96 100 108 120 125 128 135 144 150 160  
162 180 192 200 216 225 240 243

(ACM New Zealand Contest, 1990, enunț modificat)

7. *Domino.* Se consideră un sir de piese de domino. Fiecare piesă poate fi rotită în jurul centrului ei cu  $180^\circ$ . Să se determine subșirul de piese de lungime maximă în care oricare două piese alăturate au înscris același număr: al doilea număr de pe prima piesă coincide cu primul număr înscris pe cea de-a două.

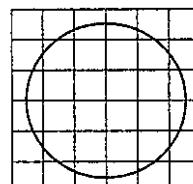
*Date de intrare:* în fișierul domino.in se găsesc cel mult 200 de piese de domino, căte o pereche pe linie. Fiecare pereche conține numere între 0 și 6.

*Date de ieșire:* în fișierul domino.out trebuie afișat pe prima linie numărul maxim de piese ale subșirului cu proprietatea din enunțul problemei, iar pe următoarele linii, căte una pe linie, trebuie scrise piesele în ordine.

*Exemplu:*

| domino.in | domino.out |
|-----------|------------|
| 5 6       | 6          |
| 1 1       | 6 5        |
| 2 5       | 5 2        |
| 3 2       | 2 3        |
| 1 5       | 3 5        |
| 6 4       | 5 5        |
| 5 1       | 5 2        |
| 5 3       |            |
| 2 1       |            |
| 1 4       |            |
| 5 5       |            |
| 5 2       |            |
| 3 3       |            |

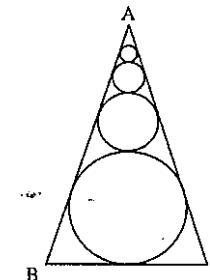
8. *Cerc și pătrățele.* Un cerc cu diametrul de  $2N-1$  unități a fost trasat în centrul unei tablă cu  $2N \times 2N$  pătrate. Această construcție pentru  $N = 3$  a fost ilustrată mai jos:



Scrieți un program care să determine numărul celulelor de pe tablă care conțin un segment de cerc și numărul celulelor de pe tablă care sunt cu totul incluse în cerc. Fiecare linie de intrare va conține un număr natural:  $0 < N < 200$ . Pentru fiecare valoare de intrare  $N$  scrieți două propoziții pe linii consecutive în fișierul de ieșire, ca în exemplul următor:

| input | output                                                                                                            |
|-------|-------------------------------------------------------------------------------------------------------------------|
| 3     | In cazul $n = 3$ , 20 celule contin segmente ale cercului.<br>Exista 12 celule complet continute in cerc.         |
| 7     |                                                                                                                   |
| 200   | In cazul $n = 7$ , 52 celule contin segmente ale cercului.<br>Exista 112 celule complet continute in cerc.        |
| 6     | In cazul $n = 200$ , 1712 celule contin segmente ale cercului.<br>Exista 124484 celule complet continute in cerc. |
|       | In cazul $n = 6$ , 44 celule contin segmente ale cercului.<br>Exista 76 celule complet continute in cerc.         |

9. *Cercuri în triunghi isoscel.* Fiind date două numere reale,  $B$  – baza unui triunghi isoscel în centimetri și  $H$  – înălțimea aceluiasi triunghi-isoscel în centimetri, calculați cu șase zecimale semnificative  $C$  – suma circumferințelor cercurilor înscrise unul deasupra altuia, de la bază spre vârf, astfel încât cel mai mic cerc înscris este tangent la cercul anterior și la cele două laturi. Presupunem că cea mai mică rază este mai mare decât 0,000001.



*Intrare:* mai multe linii de text conținând două numere reale pozitive ( $B$ ,  $H$ ) separate prin spații.

*Ieșire:* un sir de numere reale, căte unul pe linie, cu 12 poziții semnificative, șase zecimale după virgulă.

Datele de intrare și de ieșire se citesc de la tastatură și se afișează.

*Exemplu:*

| input               | output    |
|---------------------|-----------|
| 0.3456 0.6543       | 2.055531  |
| 0.789 0.456         | 1.432561  |
| 12.340987 10.909841 | 34.274265 |

10. *Căsuța lui Moș Crăciun.* De câte ori, copii fiind, nu l-am așteptat cu nerăbdare pe Moșul? Si cred că nu există cineva care să nu știe câte ceva despre căsuța lui Moș

Crăciun. Încercăm să o desenăm pe o coală fără a ridica creionul și fără să trecem de două ori prin aceeași latură. Iată cum arată căsuța de pe coală:

Dacă vreți ca Moș Crăciun să vă facă o vizită și anul acesta, trebuie să găsiți toate modalitățile de a-i construi casa, fără să ridicați creionul de pe hârtie, fără să treceți de două ori printr-o latură și pornind din punctul 1. Datele de ieșire se vor scrie în fișierul MosCraciun.out și vor consta în toate soluțiile în ordine lexicografică, numerotate, câte o propoziție pe linie pentru fiecare soluție:

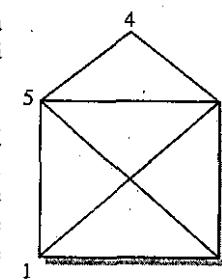
Solutia numar\_solutie: n1 n2 n3 n4 n5 n6 n7 n8 n9

Solutia 1: 1 .....

.....

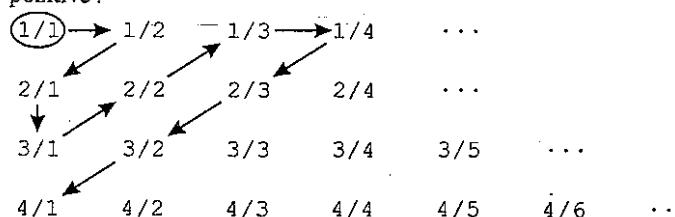
Solutia 26: 1 3 5 2 3 4 5 1 2

.....



11. *Mulțimea lui Cantor.* Una dintre cele mai cunoscute demonstrații din matematica modernă este demonstrația lui Georg Cantor a faptului că mulțimea numerelor raționale este numărabilă.

Se consideră următoarea modalitate de reprezentare a mulțimii numerelor raționale pozitive:



În această aranjare, termenul 1 este  $1/1$ , termenul 2 este  $1/2$ , termenul 3 este  $2/1$ , termenul 4 este  $3/1$ , termenul 5 este  $2/2$ , termenul șase este  $1/3$  și.a.m.d.

Dăt fiind un număr natural  $n$  ( $1 \leq n \leq 2.000.000$ ), să cere să se determine al  $n$ -lea termen din acest sir, sub forma numărator/numitor, ca în exemplul de mai jos.

În fișierul de intrare cantor.in se găsește o serie de numere, câte unul pe linie, pe ultima linie a fișierului găsindu-se valoarea 0. În fișierul cantor.out se vor scrie pe liniile corespunzătoare termenii din sirul lui Cantor.

*Exemplu :*

| cantor.in | cantor.out                                      |
|-----------|-------------------------------------------------|
| 1         | Primul termen Cantor este $1/1$ !               |
| 3         | Al 3-lea termen Cantor este $2/1$ !             |
| 14        | Al 14-lea termen Cantor este $2/4$ !            |
| 7         | Al 7-lea termen Cantor este $1/4$ !             |
| 789       | Al 789-lea termen Cantor este $9/32$ !          |
| 234561    | Al 234561-lea termen Cantor este $395/291$ !    |
| 3217      | Al 3217-lea termen Cantor este $57/24$ !        |
| 2000000   | Al 2000000-lea termen Cantor este $1000/1001$ ! |
| 3999999   | Al 3999999-lea termen Cantor este $2621/208$ !  |
| 0         |                                                 |

You see things; and you say, "Why?"  
But I dream things that never were; and I say, "Why not?"

George Bernard Shaw

## Aplicație : numere mari

### Problemă

Să se extragă radical de un ordin  $k$  ( $1 \leq k \leq 250$ ) dintr-un număr cu un număr mare de cifre (între 1 și 30.000) utilizând operații cu liste simplu înlățuite. Să se creeze un *header* care să conțină operații corespunzătoare cu astfel de numere. Să se scrie un program C separat care include acest *header* și testează toate funcțiile implementate. Dacă radicalul nu se poate extrage exact, să se afișeze partea întreagă a acestuia.

### Obiective

- realizarea unei aplicații complexe ;
- determinarea algoritmului ;
- definirea pașilor și a construcțiilor necesare ;
- crearea unui *header* cu declarațiile funcțiilor, a fișierului sursă .c aferent pentru definirea lor și a unui alt fișier pentru testare ;
- recursivitate ;
- lucrul cu liste liniare simplu înlățuite : creare, stergere din memorie, parcursare, inversare, lungimea numărului, oglindire, suma a două numere, produsul a două numere, ridicarea la putere, radical ;
- utilizarea modulului Numar în probleme concrete.

*Exemplu de execuție a programului:*

| intrare(tastatură)                                                                                       | ieșire(ecran)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --- DATE DE INTRARE ---<br>N1 = 456554687589758967598675<br>N2 = 567567676676<br>Radical de ordin k = 35 | --- REZULTATE ---<br>Primul numar N1 = 456554687589758967598675<br>Al doilea numar N2 = 567567676676<br>N1 X 8 = 3652437500718071740789400<br>!!!! N1 se incrementeaza cu 1, N1 = 456554687589758967598676<br>N1 + N2 = 456554687590326535275352<br>N1 X N2 = 259125683310856507450816900093680976<br>N1 ^ 2:<br>2084421827601824110627289654518801680137969<br>52976<br>Numar cifre din N1 = 24<br>Inversul lui N1 = 676895769857985786455654<br><br>--- Radical de ordin 35 din 456554687589758967598676<br>!!! NU se poate extrage exact radicalul !!!<br>Partea sa intreaga este: 4<br><br>Numerele 35-perfecte care incadreaza pe N1 sunt:<br>4^ 35 ---> 1180591620717411303424<br>N1 ---> 456554687589758967598676<br>5^ 35 ---> 2910383045673370361328125 |
| --- DATE DE INTRARE ---<br>N1 = 9979479338254335<br>N2 = 123456789<br>Radical de ordin k = 6             | --- REZULTATE ---<br>Primul numar N1 = 9979479338254335<br>Al doilea numar N2 = 123456789<br>N1 X 8 = 79835834706034680<br>!!! N1 se incrementeaza cu 1, N1 = 9979479338254336<br>N1 + N2 = 9979479461711125<br>N1 X N2 = 1232034474992725187887104<br>N1 ^ 2: 99590007862645199957883822800896<br>Numar cifre din N1 = 16<br>Inversul lui N1 = 6334528339749799<br><br>--- Radical de ordin 6 din 9979479338254336<br>!!! RADICAL EXACT !!!<br>Valoare = 464                                                                                                                                                                                                                                                                                                    |
| --- DATE DE INTRARE ---<br>N1 = 5767676876767687<br>N2 = 5756<br>Radical de ordin k = 65                 | --- REZULTATE ---<br>Primul numar N1 = 5767676876767687<br>Al doilea numar N2 = 5756<br>N1 X 8 = 4614141415014141496<br>!!! N1 se incrementeaza cu 1, N1 = 5767676876767688                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

| intrare(tastatură)                                                                         | ieșire(ecran)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                            | N1 + N2 = 5767676876773444<br>N1 X N2 = 3319874748102674812128<br>N1 ^ 2:<br>332660953089823500368027754720865344<br>Numar cifre din N1 = 18<br>Inversul lui N1 = 88676768676767675<br><br>--- Radical de ordin 65 din 5767676876767688<br>!!! NU se poate extrage exact radicalul !!!<br>Partea sa intreaga este: 1<br><br>Numerele 65-perfecte care incadreaza pe N1 sunt:<br>1^ 65 ---> 1<br>N1 -----> 5767676876767688<br>2^ 65 ---> 36893488147419103232                                                                                                                                                                                                                    |
| --- DATE DE INTRARE ---<br>N1 = 678687897897979879<br>N2 = 89989<br>Radical de ordin k = 3 | --- REZULTATE ---<br>Primul numar N1= 678687897897979879<br>Al doilea numar N2 = 89989<br>N1 X 8 = 5429503183183839032<br>!!! N1 se incrementeaza cu 1, N1.= 678687897897979880<br>N1 + N2 = 678687897898069869<br>N1 X N2 = 61074445243941311421320<br>N1 ^ 2:<br>460617262753178762417392584884814400<br>Numar cifre din N1 = 18<br>Inversul lui N1 = 088979798798786876<br><br>--- Radical de ordin 3 din 678687897897979880<br>!!! NU se poate extrage exact radicalul !!!<br>Partea sa intreaga este: 878799<br><br>Numerele 3-perfecte care incadreaza pe N1 sunt:<br>878799^ 3 ---> 678685643006316399<br>N1 ---> 678687897897979880<br>878800^ 3 ---> 678687959872000000 |
| --- DATE DE INTRARE ---<br>N1 = 10109221616390624<br>N2 = 567<br>Radical de ordin k = 6    | --- REZULTATE ---<br>Primul numar N1 = 10109221616390624<br>Al doilea numar N2 = 567<br>N1 X 8 = 80873772931124992<br>!!! N1 se incrementeaza cu 1, N1 = 10109221616390625<br>N1 + N2 = 10109221616391192<br>N1 X N2 = 5731928656493484375                                                                                                                                                                                                                                                                                                                                                                                                                                       |

| intrare(tastatură)                                                                           | ieșire(ecran)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                              | N1 ^ 2: 102196361689299480843652587890625<br>Numar cifre din N1 = 17<br>Inversul lui N1 = 52609361612290101<br><br>---Radical de ordin 6 din<br>10109221616390625<br>!!! RADICAL EXACT !!!<br>Valoare = 465                                                                                                                                                                                                                                                                                                        |
| --- DATE DE INTRARE ---<br>N1 = 99999999999999999999<br>N2 = 1234<br>Radical de ordin k = 20 | --- REZULTATE ---<br>Primul număr N1 = 99999999999999999999<br>Al doilea număr N2 = 1234<br>N1 X 8 = 79999999999999999999<br>!!! N1 se incrementează cu 1, N1 = 10000000000000000000<br>N1.+ N2 = 100000000000000000001234<br>N1 X N2 = 12340000000000000000000000<br>N1 ^ 2:<br>10000000000000000000000000000000000000000000<br>Numar cifre din N1 = 21<br>Inversul lui N1 = 000000000000000000000000000000001<br><br>---Radical de ordin 20 din<br>10000000000000000000<br>!!! RADICAL EXACT !!!<br>Valoare = 10 |
|                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

### Analiza problemei și proiectarea soluției

#### Algoritmul de determinare a radicalului de ordin k

Considerăm că se dă un număr natural A cu N cifre și un număr natural K (ordinul radicalului cerut,  $K \geq 1$ ).

*Observație:* numărul de cifre ale radicalului de ordin K din A va fi  $(N/K)$  dacă N se divide cu K și  $(N/K+1)$  dacă N nu se divide cu K.

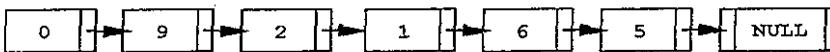
Determinarea cifrelor radicalului se va face succesiv, pornind de la cea mai semnificativă spre cea mai puțin semnificativă. Notăm cu x acest radical, care are forma:

$$x_m x_{m-1} x_{m-2} \dots x_1 x_0$$

Presupunând că cifrele  $x_m, x_{m-1}, \dots, x_{i+1}$  au fost determinate, cifra  $x_i$  va fi cea mai mare cifră cu proprietatea că:

$$(x_m x_{m-1} x_{m-2} \dots x_{i+1} x_i 0 \dots 0)^K \leq A$$

Vom păstra un număr într-o listă simplu înlăncuită, inversat, de exemplu numărul  $N = 561290$  va fi reprezentat prin lista:



iar definiția tipului va fi:

```

typedef struct nod{
 short c;
 struct nod *leg;
} TNumar;

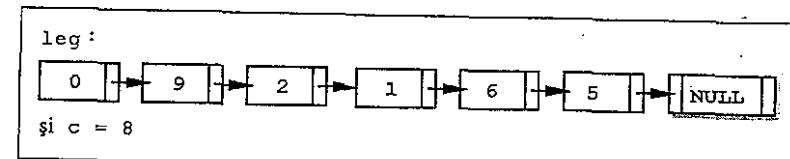
```

În scopul construirii funcției radical() va trebui să definim o serie de alte funcții ajutătoare. Vom scrie în fișierul-header Numar.h includerea librăriilor de care este nevoie, definirea tipului TNumar și definițiile funcțiilor necesare, care vor fi declarate în fișierul-sursă corespunzător Numar.c.

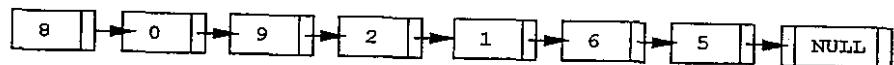
Funcția cu antetul:

```
TNumar *creeazaNod(short c, TNumar *leg);
```

Va executa crearea unui nod de tip TNumar\* cu câmpurile-informărie c (cifra) și leg (lista referită). Dacă:



Atunci creeazaLista(c, leg) va furniza rezultatul:



Funcția citesteNumar(TNumar\*\*, numar) va construi numărul cifră cu cifră, adăugând cifra tocmai citită în fața numărului. Pentru caracterul ch citit, se verifică dacă este cifră utilizând isdigit(ch); dacă da, atunci se va crea un nod cu informația ch-'0' (cifra corespunzătoare) și legătura NULL. Dacă aceasta este prima listă citită, atunci va fi și lista returnată \*nr, altfel se va adăuga în față prin secvență:

```
q->leg = *nr; *nr = q;
```

Afișarea, respectiv eliberarea memoriei utilizate de un astfel de număr se realizează cu ajutorul funcțiilor recursive scrie() și sterge().

Funcția cu antetul:

```
TNumar* inmulCifra(TNumar *numar, short cfr);
```

Va executa înmulțirea unui număr mare stocat în formă de listă număr cu o cifră c. Înmulțirea se face matematic, pornind de la cifra cea mai puțin semnificativă spre cea mai semnificativă, păstrând în t valoarea de transport (initial egală cu 0). În variabila cap se va păstra lista de returnat și se va folosi o variabilă ultim pentru a memora la fiecare pas ultimul element introdus.

Funcția cu antetul :

```
void adaugaLaSfarsit(TNumar **cap, TNumar **ultim, TNumar
sursa, short t);
```

va fi folosită pentru adunarea a două numere mari, în cazul când unul dintre numere s-a încheiat de adunat (a fost mai scurt decât celalalt) și ne rămâne de adăugat partea corespunzătoare din numărul mai lung, păinând cont de transportul curent.

Funcția suma() calculează suma a două numere mari, efectuând suma matematic, începând cu cifra cea mai puțin semnificativă; pentru fiecare poziție se creează un nod q, se recalculează transportul t, se adaugă la sfârșit folosind variabila ultim :

```
q = creeazaNod((short)((nr1->c+nr2->c+t)%10), NULL);
t = (nr1->c + nr2->c + t) / 10;
if(!cap) {cap = ultim = q;}
else {
 ultim->leg = q;
 ultim = q;
}
nrl=nr1->leg; nr2=nr2->leg;
```

Aceste instrucțiuni se repetă atât timp cât ambele numere mai au încă cifre de prelucrat. După încheierea acestei instrucțiuni repetitive while(), se va folosi funcția precedentă adaugaLaSfarsit() pentru a adăuga cifrele următoare, care pot fi în unul dintre cele două numere, ținând cont și de transportul t.

Urmează funcția :

```
TNumar* produs(TNumar *nrl, TNumar *nr2);
```

Care efectuează produsul matematic a două numere mari; pentru fiecare poziție din al doilea număr se înmulțește primul număr cu cifra respectivă, se adaugă numărul de zerouri și sfârșitul corespunzător cu rangul cifrei din al doilea număr (pentru cea mai puțin semnificativă, rangul este 0 și crește mereu cu 1; acest rang se va memora în variabila factor), se va adăuga acest număr q nou creat la cap, apoi se va șterge din memorie și vom trece la următoarea cifră din al doilea număr :

```
q = inmulCifra(nrl, nr2->c);
for(i=0; i<factor; i++){
 aux = creeazaNod(0, q);
 q=aux;
}
cap=suma(q, cap);
sterge(q);
nr2=nr2->leg; factor++;
```

Încercați o simulare a acestei funcții „cu creionul pe hârtie”, de exemplu încercați să înmulțești 2345 cu 651 (while{} se va „întâmpla” de 3 ori, numărul de cifre al celui de al doilea număr; scrieți de fiecare dată conținutul variabilelor factor, q, cap).

Funcția :

```
TNumar* putere(TNumar *nr, short k);
```

Calculează puterea a k-a a unui număr natural dat nr, considerat o astfel de listă. Se va inițializa variabila cap cu numărul 1, apoi se execută de k ori înmulțirea la cap a numărului dat :

```
for(i=0; i<k; i++){
 aux = produs(cap, nr);
 sterge(cap);
 cap = aux;
}
```

Observăm că de fiecare dată după ce s-a realizat produsul intermediu, memoria folosită de cap se eliberează utilizând funcția sterge().

Funcția nrCifre() este o funcție extrem de simplă, ea returnează numărul cifrelor unui astfel de număr prin parcursarea sa.

Funcția invers() inversează lista-număr dată ca parametru prin crearea succesivă a către unui nod pentru fiecare cifră a numărului dat și adăugarea lui în față.

Funcția nrCmp(TNumar \*nrl, TNumar \*nr2) compară cele două numere date; dacă sunt egale, atunci se va returna 0, dacă primul este mai mare decât al doilea, se returnează 1, altfel se va returna -1. Mai întâi, se compară numărul de cifre ale celor două numere și în funcție de acestea se va returna 1 sau -1 direct. Dacă numărul de cifre ale celor două numere sunt strict pozitive, atunci se vor inversa (pentru a porni cu cifra cea mai semnificativă), vom compara succesiv cifrele până când se obțin cifre diferite și se va returna valoarea corespunzătoare; dacă nu vom găsi niciodată cifre diferite, atunci conchidem că numerele sunt egale, adică vom returna 0 :

```
o1 = invers(nrl);
o2 = invers(nr2);
while(o1 && o2){
 if(o1->c > o2->c) return 1;
 if(o2->c > o1->c) return -1;
 o1=o1->leg; o2=o2->leg;
}
return 0;
```

Funcția plusUnu() va adăuga 1 la un număr mare (incrementarea cu 1) prin parcursarea numărului și ajustarea cifrelor, ținând cont de transportul t, inițializat cu 1.

Funcția nrCpy() va copia un număr mare în alt număr mare prin alocarea memoriei necesare, cifră cu cifră, folosind o variabilă auxiliară ultim care va reține la fiecare pas ultima cifră curentă.

În final, ajungem și la scopul problemei, adică calculul radicalului de ordin k dintr-un astfel de număr mare. Într-o primă etapă, decidem numărul de cifre n al radicalului :

```
n = nrCifre(nr);
if (n%k==0) n = n/k;
else n = n/k+1;
```

Pentru fiecare poziție de la n la 1 vom căuta cifra corespunzătoare în radical, așa cum reiese ea din algoritmul de calcul. În variabila rez vom memora numărul construit până acum, se va adăuga cifra c și se completează cu zerouri până la numărul de cifre n. Se execută acești pași până când numărul obținut prin ridicarea la puterea k este mai mare decât numărul dat nr. În acest moment se adaugă cifra c-1 la rez :

```
for(i=n; i>0; i--){
 c = -1;
 do{
 c++;
 if(man) sterge(man);
 nrCpy(&man, rez);
 q = creeazaNod(c, man);
 man=q;
 for(j=1; j<i; j++){
 q = creeazaNod((short)0, man);
 man=q;
 }
 } while(nrCmp(putere(man, k), nr) <= 0);
 q = creeazaNod((short)(c-1), rez);
 rez=q;
} /* for */
```

Analizați programul test.c, care utilizează operațiile cu numere mari tocmai implementate și care este sugestiv în acest sens.

#### Header Numar.h

```
#ifndef __NUMAR_H
#define __NUMAR_H

#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <conio.h>
```

```
typedef struct nod{
 short c;
 struct nod *leg;
}TNumar;

TNumar *creeazaNod(short c, TNumar *leg);

void citeste(TNumar **numar);
void scrie(TNumar *numar);
void sterge(TNumar *numar);
```

```
TNumar* inmulCifra(TNumar *numar, short c);
void adaugaLaSfarsit(TNumar **cap, TNumar **ultim, TNumar *sursa, short* t);
TNumar* suma(TNumar *nrl, TNumar *nr2);
TNumar* produs(TNumar *nrl, TNumar *nr2);
TNumar* putere(TNumar *nr, short k);
int nrCifre(TNumar *numar);
TNumar *invers(TNumar *numar);
short nrCmp(TNumar* nrl, TNumar *nr2);
void plusUnu(TNumar **numar);
void nrCpy(TNumar **dest, TNumar* sursa);
TNumar* radical(TNumar *nr, short k);
```

```
#endif
```

#### Implementare Numar.c

```
#include "Numar.h"

Crearea unui nod de tip TNumar cu informațiile date ca parametri: cifra și legatura ****
TNumar *creeazaNod(short c, TNumar *leg){
 TNumar *q = (TNumar*)malloc(sizeof(TNumar));
 q->c = c;
 q->leg = leg;
 return q;
}
```

```

Citirea unui număr mare: numărul se construiește cifra cu cifra și se păstrează inversat în lista ****
```

```

void citeste(TNumar **nr){
 TNumar *q=NULL;
 char ch;
 *nr = NULL;
 do{
 ch=getch(); putch(ch);
 if(isdigit(ch)) {
 q = creeazaNod((short)(ch-'0'), NULL);
 if(*nr==NULL) (*nr=q);
 else {q->leg = *nr; *nr = q;};
 }
 }while(isdigit(ch));
}

Afisarea unui numar mare: se realizeaza recursiv
tinand cont de faptul ca el este memorat inversat
in lista: primul element fiind cifra cea mai putin
semnificativa, cea a unitatilor
*****void scrie(TNumar *numar){
 if(numar){
 scrie(numar->leg);
 printf("%d", numar->c);
 }
}

Eliberarea memoriei utilizate de un numar mare,
tot o metoda recursiva...
*****void sterge(TNumar *numar){
 if(numar){
 sterge(numar->leg);
 free(numar);
 }
}

Inmultirea unui numar mare cu o cifra: se va
parcurge lista formata din numar, tinand cont
de cifra de transport si se creeaza succesiv
numarul rezultat

```

```

TNumar* inmulCifra(TNumar *numar, short cfr){
 TNumar *cap=NULL, *q = NULL, *ultim = NULL;
 short t = 0;
 while(numar){
 q = creeazaNod((short)((numar->c*cfr+t)%10), NULL);
 t = (numar->c*cfr + t) / 10;
 if(cap == NULL) {cap=ultim=q;}
 else{
 ultim->leg = q;
 ultim = q;
 }
 numar = numar->leg;
 }
 if(t){
 q = creeazaNod(t, NULL);
 ultim->leg = q;
 ultim = q;
 }
 return cap;
}

Adaugarea la sfarsitul unui numar deja creat numarul obtinut
dintr-o lista data sursa si cifra de transport t
*****void adaugaLaSfarsit(TNumar **cap, TNumar **ultim,
 TNumar *sursa, short *t){
 TNumar *q;
 while(sursa){
 q = creeazaNod((short)((sursa->c+*t)%10), NULL);
 *t = (sursa->c + *t) / 10;
 if(!(*cap)) (*cap = *ultim = q);
 else{
 (*ultim)->leg = q;
 (*ultim) = q;
 }
 sursa=sursa->leg;
 }
 if(*t){
 q = creeazaNod((short)(*t), NULL);
 (*ultim)->leg=q;
 *t = 0;
 (*ultim)=q;
 }
}

```

```

Calculul sumei a două numere mari: se parcurge
algoritmul matematic corespunzător, cand una
din trei liste este văzuta deja, se va continua cu
cea alătura, utilizând funcția adaugăLaSfarsit()

```

```
TNumar* suma(TNumar *nr1, TNumar *nr2){
 TNumar *cap=NULL, *ultim=NULL, *q=NULL;
 short t = 0;
 while(nr1 && nr2){
 q = creeazaNod((short)((nr1->c+nr2->c+t)%10), NULL);
 t = (nr1->c + nr2->c + t) / 10;
 if(!cap) {cap = ultim = q;}
 else{
 ultim->leg = q;
 ultim = q;
 };
 nr1=nr1->leg; nr2=nr2->leg;
 }
 adaugăLaSfarsit(&cap, &ultim, nr1, &t);
 adaugăLaSfarsit(&cap, &ultim, nr2, &t);
 return cap;
}

Produsul a două numere mari

```

```
TNumar* produs(TNumar *nr1, TNumar *nr2){
 int i, factor = 0;
 TNumar* cap = NULL, *q, *aux;
 while(nr2){
 q = inmulCifra(nr1, nr2->c);
 for(i=0; i<factor; i++){
 aux = creeazaNod(0, q);
 q=aux;
 }
 cap=suma(q, cap);
 sterge(q);
 nr2=nr2->leg; factor++;
 }
 return cap;
}
```

```

Un număr mare ridicat la o putere

```

```
TNumar* putere(TNumar *nr, short k){
 TNumar *cap = NULL, *aux = NULL;
 int i;
 cap = (TNumar*) malloc(sizeof(TNumar));
 cap->c = 1; cap->leg = NULL;
 for(i=0; i<k; i++){
 aux = produs(cap, nr);
 sterge(cap); cap=aux;
 }
 return cap;
}

Returnează lungimea unui număr natural mare

```

```
int nrCifre(TNumar *nr2){
 int lung = 0;
 while(nr2){
 lung++;
 nr2=nr2->leg;
 }
 return lung;
}

Returnează oglinditul unui număr...

```

```
TNumar *invers(TNumar *nr){
 TNumar *cap = NULL, *q = NULL;
 while(nr){
 q = creeazaNod(nr->c, NULL);
 if(cap == NULL){cap=q;}
 else{
 q->leg=cap; cap=q;
 }
 nr=nr->leg;
 }
 return cap;
}
```

```

Returneaza semnul
nrCmp(N1, N2) = | -1 daca N1<N2
| 0 daca N1=N2
| 1 daca N1>N2

short nrCmp(TNumar* nr1, TNumar *nr2){
 short decis = 0;
 TNumar *o1, *o2;
 if(nrCifre(nr1)>nrCifre(nr2)) return 1;
 if(nrCifre(nr1)<nrCifre(nr2)) return -1;
 o1 = invers(nr1);
 o2 = invers(nr2);
 while(o1 && o2){
 if(o1->c > o2->c) return 1;
 if(o2->c > o1->c) return -1;
 o1=o1->leg; o2=o2->leg;
 }
 return 0;
}

Incrementarea cu 1 a unui numar mare

void plusUnu(TNumar **numar){
 short t=1, aux;
 TNumar *q = NULL, *man, *naux;
 naux = man = *numar;
 while (t && man){
 aux = man->c;
 man->c = (aux + t) % 10;
 t = (aux + t) / 10;
 naux = man;
 man = man->leg;
 }
 if(t){
 q = creeazaNod(t, NULL);
 if((*numar) == NULL) (*numar=q);
 else
 naux->leg = q;
 }
}
```

```

copierea unui numar mare ... in altul...

void nrCpy(TNumar **dest, TNumar* sursa){
 TNumar *q, *ultim;
 *dest = NULL;
 if((*dest)==NULL) sterge(*dest);
 while(sursa){
 q = creeazaNod(sursa->c, NULL);
 if((*dest)==NULL){
 *dest=ultim=q;
 }else{
 ultim->leg=q; ultim=q;
 }
 sursa=sursa->leg;
 }
}

Returneaza radicalul de ordinul k dintr-un numar
mare

TNumar* radical(TNumar *nr, short k){
 int i, j, n;
 short c;
 TNumar *rez, *q, *man;
 n = nrCifre(nr);
 if(n%k==0) n = n/k;
 else n = n/k+1;
 rez=NULL; man=NULL;
 for(i=n; i>0; i--){
 c = -1;
 do{
 c++;
 if(man) sterge(man);
 nrCpy(&man, rez);
 q = creeazaNod(c, man);
 man=q;
 for(j=1; j<i; j++){
 q = creeazaNod((short)0, man);
 man=q;
 }
 } while (nrCmp(putere(man, k), nr) <= 0);
 q = creeazaNod((short)(c-1), rez);
 }
}
```

```

 rez=q;
} /* for */
return rez;
}

Program test.c
#include "numar.h"

void main(){
 TNumar *nr1=NULL, *nr2=NULL, *nr3=NULL;
 short k;
 printf(" --- DATE DE INTRARE ---");
 printf("\n N1 = ");
 citeste(&nr1); printf("\n");
 printf(" N2 = ");
 citeste(&nr2); printf("\n");
 printf("Radical de ordin k = ");
 scanf("%d", &k);
 printf("\n\n--- REZULTATE ---");
 printf("\nPrimul numar N1 = "); scrie(nr1);
 printf("\nAl doilea numar N2 = "); scrie(nr2);
 printf("\nN1 X 8 = "); scrie(inmulCifra(nr1,8));
 plusUnu(&nr1);
 printf("\n!!! N1 se incrementeaza cu 1, N1 = ");
 scrie(nr1);
 nr3 = suma(nr1, nr2);
 printf("\nN1 + N2 = "); scrie(nr3); sterge(nr3);
 nr3 = produs(nr1, nr2);
 printf("\nN1 X N2 = "); scrie(nr3); sterge(nr3);
 nr3=putere(nr1, 2);
 printf("\nN1 ^ 2: "); scrie(nr3); sterge(nr3);
 printf("\nNumar cifre din N1 = ");
 printf("%d", nrCifre(nr1));
 printf("\nInversul lui N1 = ");
 scrie(invers(nr1));
 printf("\n\n ---Radical de ordin %d din ", k);
 scrie(nr1);
 nr3 = radical(nr1, k);
 if(nrCmp(putere(nr3,k), nr1) == 0){
 printf("\n!!! RADICAL EXACT !!!");
 printf("\n Valoare = "); scrie(nr3);
 }
 else{
 printf("\n!!! NU se poate extrage exact radicalul !!!");
 }
}

```

```

printf("\nPartea sa intreaga este: ");
scrie(nr3);
printf("\n\nNumerele %d-perfecte care incadreaza ", k);
printf(" pe N1 sunt:\n");
scrie(nr3); printf("^ %d ---> ", k);
scrie(putere(nr3, k));
printf("\n N1 ----> ");
scrie(nr1);
plusUnu(&nr3);
printf("\n"); scrie(nr3); printf("^ %d ---> ", k);
scrie(putere(nr3, k));
}
sterge(nr1); sterge(nr2); sterge(nr3);
getch();
}

```

### Exerciții

1. Implementați și alte funcții cu numere mari: diviziune, modulo.
2. Folosiți biblioteca creată mai sus pentru afișarea numerelor 100!, 200!..
3. Pentru a verifica cât de importantă este eliberarea memoriei, ștergeți de peste tot din program apelurile funcției `sterge()` și rulați programul cu numere foarte mari, 400-500 de cifre. Ce se întâmplă?

*Călătorul care a urcat povârnișul unui munte se aşază pe creastă și odihna îi place nespus de mult. Ar fi el însă fericit dacă l-ai săli să se odihnească într-o una?*

Stehdhal

## Aplicație: fractali space-feeling

**Definiția fractalului:** un model geometric care se repetă la dimensiuni din ce în ce mai mici și produce curbe neregulate și suprafețe ce nu pot fi reprezentate de geometria clasică. Fractalii sunt în special folosiți în realizarea pe computer a modelelor neregulate și a structurilor din natură.

Vom descrie un model de fractal care are proprietatea de a „umple” spațiul pătratic  $[0, 1] \times [0, 1]$  și îl vom aplica iterativ pe diverse figuri geometrice de start.

### Obiective

- realizarea unei aplicații complexe;
- exemplu de fractal;
- aplicarea unei formule iterative;
- operatori pe biți;
- formule geometrice pentru diverse curbe de bază;
- lucrul cu o bibliotecă grafică.

### Fractali: curbe care simt spațiul

Fie  $P(0, 1) = \{(x, y) \mid 0 < x, y < 1\}$ , adică mulțimea tuturor punctelor din interiorul pătratului cu vîrfurile  $(0, 0)$ ,  $(0, 1)$ ,  $(1, 1)$ ,  $(1, 0)$ .

Vom descrie în continuare o metodă prin care o curbă înclusă în  $P(0, 1)$  simte acest spațiu astfel încât îl va umple.

Considerăm tablourile unidimensionale  $a, b, c, d, e, f$  date de tabelul:

| w | a   | b    | c    | d   | e   | f   |
|---|-----|------|------|-----|-----|-----|
| 1 | 0   | 0.5  | 0.5  | 0   | 0   | 0   |
| 2 | 0.5 | 0    | 0    | 0.5 | 0   | 0.5 |
| 3 | 0.5 | 0    | 0    | 0.5 | 0.5 | 0.5 |
| 4 | 0   | -0.5 | -0.5 | 0   | 1   | 0.5 |

Fie  $C_0$  curba initială din  $P(0, 1)$ .

Din curba  $C_i$  se obține curba  $C_{i+1}$ , transformând fiecare punct din  $C_i$  în alte patru puncte ce vor fi incluse în  $C_{i+1}$  astfel:

$$w_n \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_n & b_n \\ c_n & d_n \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_n \\ f_n \end{bmatrix}, n = 1, 4$$

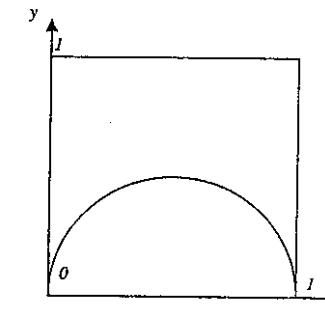
deci:

$$\begin{cases} x_{n,i+1} = a_n \cdot x_i + b_n \cdot y_i + e_n, \\ y_{n,i+1} = c_n \cdot x_i + d_n \cdot y_i + f_n, \end{cases} n = 1, 4$$

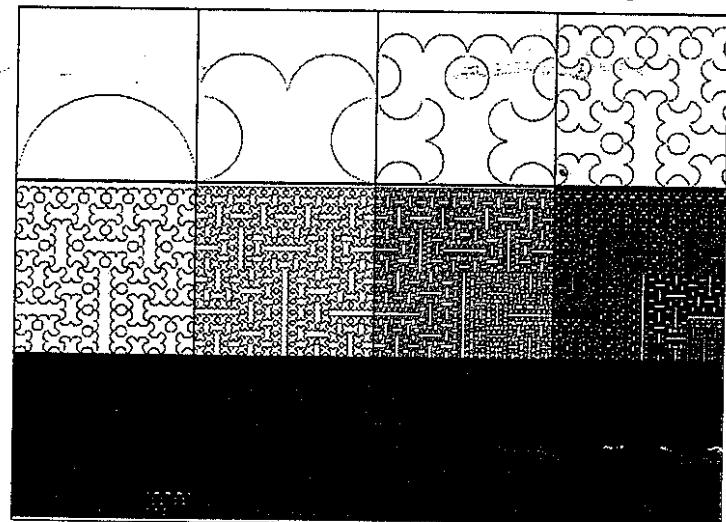
Așadar, putem conchide formarea curbei  $C_{i+1}$  din  $C_i$  astfel:

$$C_{i+1} = \bigcup_{n=1}^4 ((x_{n,i+1}, y_{n,i+1}) \mid (x_i, y_i) \in C_i)$$

Considerăm, de exemplu, curba  $C_0$ :

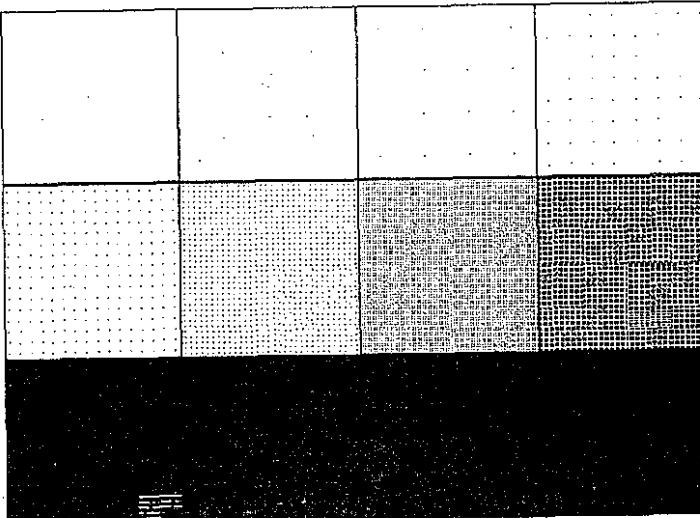


șirul transformărilor va construi următoarele curbe (de la stânga la dreapta și de sus în jos):

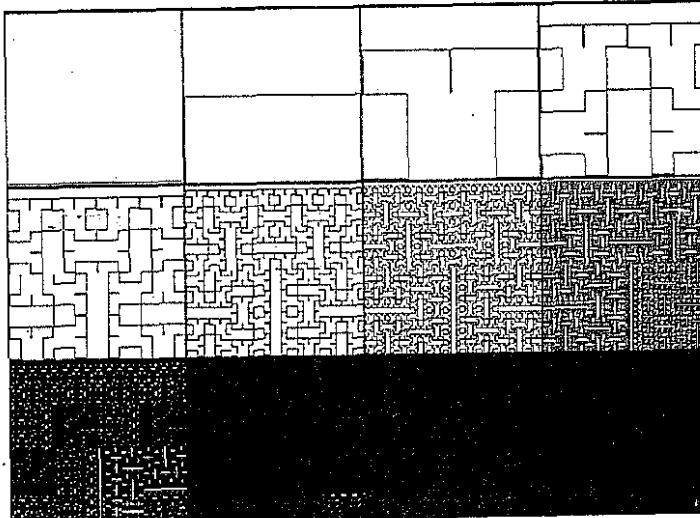


*Alte exemple:*

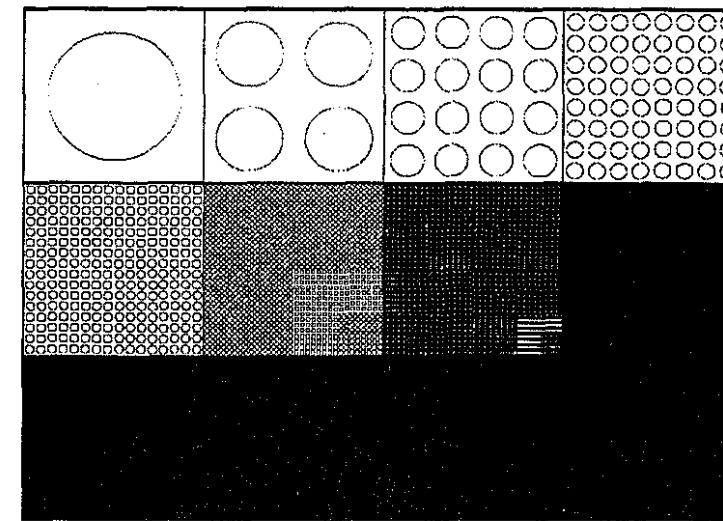
1. Curba inițială este un punct :



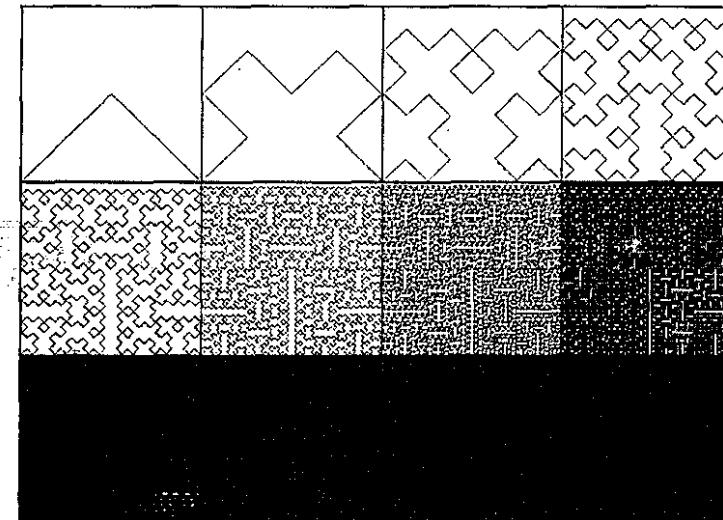
2. Curba inițială este un segment de dreaptă inclus în  $P(0, 1)$  :



3. Curba inițială este un cerc inclus în  $P(0, 1)$  :



4. Curba inițială este o linie frântă :



Vom scrie în continuare un program C care să ilustreze această proprietate a curbelor care simt spațiul. Vom lucra cu pătrate de  $160 \times 160$  biți, dar ca implementare în C, tipul definit va fi :

```
| typedef int Patrat[160][10];
```

unde fiecare element al matricei conține 16 biți, de aceea utilizăm doar 10 coloane.

Pentru a seta un element  $a[i][j]$ ,  $0 \leq i, j < 160$  cu 0 sau 1, utilizăm funcția cu antetul :

```
void set(Patrat p, int i, int j, short bit).
```

Elementul din matrice asupra căruia vom opera este:  $p[i][j/16]$ . Înănd cont de observația de mai sus, trebuie să setăm cu bitul al  $j \% 16$ -lea din reprezentarea binară a acestui număr. Pentru acesta, vom utiliza măștile  $1 \ll j \% 16$  (toate pozițiile binare 0, în afară de poziția  $j \% 16$ ; dacă bit este 1 se aplică un SAU LOGIC asupra numărului cu această mască, de unde rezultă că toți biții vor rămâne neschimbați, exceptându-l pe al  $j \% 16$ -lea, care va deveni 1) și  $\sim(1 \ll (j \% 16))$  (toate pozițiile binare 1, în afară de poziția  $j \% 16$ ; dacă bitul este 0, se aplică SI LOGIC asupra numărului cu această mască de unde rezultă că toți biții vor rămâne neschimbați, exceptându-l pe al  $j \% 16$ -lea, care va deveni 0).

Pentru a obține bitul de pe poziția  $(i, j)$ ,  $0 \leq i, j < 160$  scriem funcția cu antetul :

```
short get(Patrat p, int i, int j).
```

Bitul necesar se află pe poziția a  $j \% 16$ -a în reprezentarea binară a elementului  $n=p[i][j/16]$  și acesta este  $(n>>j \% 16) \& 1$ .

Vom implementa funcția cu antetul :

```
void deseneaza(Patrat p, int x, int y,
 int culFond, int culPixel)
```

Care va desena pe ecran, începând cu colțul din stânga-sus de coordonate  $(x, y)$ , pătratul  $p$ , înănd cont de biții acestuia. Pentru a reprezenta pe ecran  $P(0, 1)$  vom efectua o scalare la 160.

Vom scrie funcțiile  $init1()$  ...  $init10()$ , care inițializează o astfel de matrice bidimensională cu diferite curbe. Vom reprezenta pe ecran în toate punctele de coordonate  $cX[i] \times cY[j]$ ,  $0 \leq i \leq 4$ ,  $0 \leq j \leq 3$ , unde  $cX[]$  și  $cY[]$  sunt vectori constanți, declarati cu valori corespunzătoare unei dimensiuni clasice a ecranului grafic de 640 x 480 pixeli.

Programul complet este redat în continuare :

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>

typedef int Patrat[160][10];

const double a[4] = { 0, 0.5, 0.5, 0 },
 b[4] = { 0.5, 0, 0, -0.5 },
 c[4] = { 0.5, 0, 0, -0.5 },
 d[4] = { 0.0, 0.5, 0.5, 0.0 },
 e[4] = { 0.0, 0, 0.5, 1.0 },
 f[4] = { 0.0, 0.5, 0.5, 0.5 };
```

```
const int cx[4] = { 0, 162, 324, 486 };
const int cy[3] = { 2, 165, 328 };

void initGrafica()
{
 int gdet = DETECT, gm, err;
 initgraph(&gdet, &gm, "H:\\borlandc\\bgi");
 err = graphresult();
 if(err != grOk)
 {
 printf("Eroare la initializare mod grafic!!!");
 printf("Apasati o tasta pentru iesire!");
 getch();
 exit(1);
 }
}

void set(Patrat p, int i, int j, short bit)
{
 int n;
 n = p[i][j/16];
 if(bit)
 n |= 1 << j % 16;
 else
 n &= ~(1 << (j % 16));
 p[i][j/16]=n;
}

short get(Patrat p, int i, int j)
{
 short bit;
 int l, n;
 n = p[i][j/16];
 return (n>>j % 16) & 1;
}

void deseneaza(Patrat p, int x, int y,
 int culFond, int culPixel)
{
 int i, j;
 for(i=x; i<x+160; i++)
 for(j=y; j<y+160; j++)
 {
```

```

 if(get(p, i-x, j-y))
 putpixel(i, 2*y+159-j, culPixel);
 else
 putpixel(i, 2*y+159-j, culFond);
 }

void init1(Patrat w)
{
 int i;
 for(i=1; i<160; i++)
 if(i<80){ set(w, i, i, 1);
 set(w, 159-i, i, 1); }
}

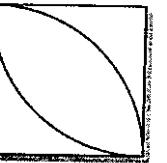
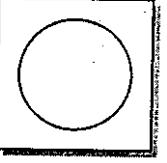
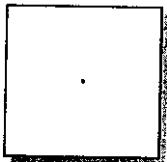
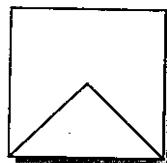
void init2(Patrat w)
{
 set(w, 79, 79, 1);
}

void init3(Patrat w)
{
 int i;
 for(i=1; i<159; i++)
 set(w, i, i, 1, 1);
}

void init4(Patrat w)
{
 int i;
 for(i=20; i<141; i++)
 {
 set(w, i, 80 + sqrt((140-i)*(i-20)), 1);
 set(w, i, 80 - sqrt((140-i)*(i-20)), 1);
 }
}

void init5(Patrat w)
{
 int i;
 for(i=1;i<160; i++)
 {
 set(w, i, sqrt(159*159-i*i), 1);
 }
}

```



```

 set(w, i, 159-sqrt(i*(318-i)), 1);
 }
}

void init6(Patrat w)
{
 int i, j;
 for(i=40; i<60; i++)
 for(j=100; j<150; j++)
 set(w, i, j, 1);
}

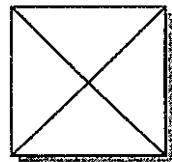
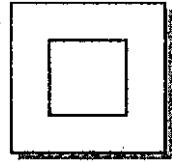
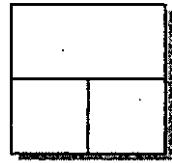
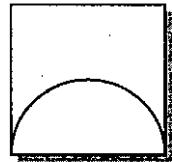
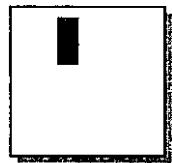
void init7(Patrat w)
{
 int i;
 for(i=1; i<160; i++)
 set(w, i, sqrt(80*80 -(i-80)*(i-80)), 1);
}

void init8(Patrat w)
{
 int i;
 for(i=1; i<160; i++)
 set(w, i, 80, 1);
 for(i=1; i<80; i++)
 set(w, 80, i, 1);
}

void init9(Patrat w)
{
 int i;
 for(i=40; i<120; i++)
 {
 set(w, i, 40, 1);
 set(w, i, 119, 1);
 set(w, 40, i, 1);
 set(w, 119, i, 1);
 }
}

void init10(Patrat w)
{
 int i;
 for(i=1; i<160; i++)

```



```

 {
 set(w, i, i, 1);
 set(w, i, 159-i, 1);
 }

int init(int i, Patrat w)
{
 int k, j;
 for(k=0; k<160; k++)
 for(j=0; j<10; j++)
 w[k][j]=0;
 switch(i)
 {
 case 1: init1(w); break;
 case 2: init2(w); break;
 case 3: init3(w); break;
 case 4: init4(w); break;
 case 5: init5(w); break;
 case 6: init6(w); break;
 case 7: init7(w); break;
 case 8: init8(w); break;
 case 9: init9(w); break;
 case 10: init10(w); break;
 }
 return(0<i && i<11);
}

void main()
{
 Patrat w, v;
 int contor, i, j, k, iNou, jNou;
 printf("Optiunea(1-10): ");
 while(scanf("%d", &k)==1 && init(k, w))
 {
 initGrafica();
 for(contor=0; contor<12; contor++)
 {
 deseneaza(w, cX[contor%4], cY[contor/4], 15, 1);
 for(i=0; i<160; i++)
 for(j=0; j<10;j++)
 v[i][j]=0;
 }
}

```

```

 for(i=0; i<160; i++)
 for(j=0; j<160; j++)
 if(get(w, i, j) == 1)
 for(k=0; k<4; k++)
 {
 iNou = 160*(a[k]*i/160.0+b[k]*j/160.0+e[k]);
 jNou = 160*(c[k]*i/160.0+d[k]*j/160.0+f[k]);
 set(v, iNou,jNou, 1);
 }
 for(i=0; i<160; i++)
 for(j=0; j<160; j++)
 {
 set(w, i,-j, get(v, i, j));
 set(v, i,j,0);
 }
 getch();
 closegraph();
 printf("Optiunea (1-10): ");
 }
}

```

Modul de funcționare a programului: atât timp cât se citește de la tastatură un număr natural cuprins între 1 și 10, se va inițializa matricea cu curba corespunzătoare numărului și se vor efectua 12 pași de transformare, la fiecare pas desenându-se pătratul în poziția corespunzătoare iterării.  $w[][]$  reprezintă matricea veche și  $v[][]$  este matricea nouă, care se initializează cu 0 prin:

```

 for(i=0; i<160; i++)
 for(j=0; j<10;j++)
 v[i][j]=0;

```

Când găsim un bit 1 în matricea  $w[][]$  ( $get(w, i, j) == 1$ ), construim cele 4 puncte corespunzătoare transformării date și le setăm pe 1 în matricea  $v[][]$ :

```

 for(k=0; k<4; k++)
 {
 iNou = 160*(a[k]*i/160.0+b[k]*j/160.0+e[k]);
 jNou = 160*(c[k]*i/160.0+d[k]*j/160.0+f[k]);
 set(v, iNou,jNou, 1);
 }
}

```

Se copiază conținutul lui  $v[][]$  în  $w[][]$  și  $v[][]$  se setează numai cu 0-uri prin:

```

 for(i=0; i<160; i++)
 for(j=0; j<160; j++)
 v[i][j]=0;
}

```

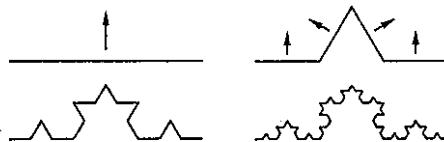
```

 {
 set(w, i, j, get(v, i, j));
 set(v, i,j,0);
 }
}

```

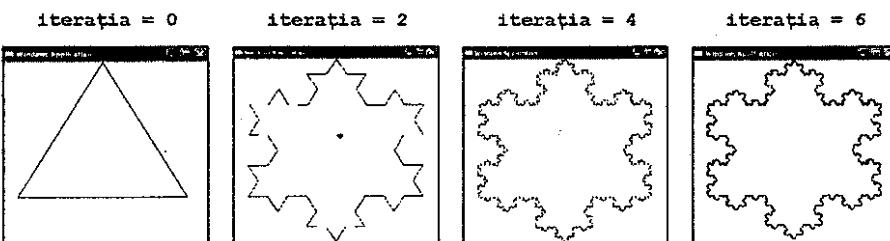
### Exerciții

1. Librăria grafică folosită este cea din DOS și probabil că folosiți alt sistem de operare. ;-) În acest caz, modificați programul astfel încât să folosească o bibliotecă grafică adecvată.
2. Modificați programul de mai sus astfel încât, pentru fiecare curbă, să numere în câți pași a umplut spațiul, adică după câte iterații tabloul bidimensional conține numai biți de 1.
3. Creați și alte funcții de inițializare cărora să le corespundă noi curbe în  $P(0, 1)$ .
4. Modelați fractalii în același pătrat, cu pauze, pentru a se remarcă efectul.
5. *Curba lui Koch (Curba fulgului de zăpadă)*. Curba lui Koch a fost introdusă în 1904 de către matematicianul suedez Helge von Koch (1870-1924) și este unul dintre primii fractali descriși formal. Curba se poate construi de-a lungul unui proces iterativ. La început este formată dintr-un singur segment de dreaptă și la fiecare iterație se împarte fiecare segment din curbă în trei părți egale și se execută construcția „în triunghi echilateral”, ca în figura următoare (sugestivă pentru primele trei iterații):



Scrieți un program care constuește curba lui Koch pentru diverse iterații. Curba *fulgului de zăpadă* se obține atunci când la început sunt trei segmente ce formează un triunghi echilateral. Scrieți un program care reprezintă grafic *fulgul de zăpadă Koch* pentru o iterație dată.

*Exemplu :*



Calculați aria curbei lui Koch pentru un triunghi inițial cu latura L dată și iterația n. Care este aria *fulgului de zăpadă* pentru cazul când n tinde la infinit?

### Bibliografie

1. Barnsley, M.F., *Fractals everywhere. Second edition*, Academic Press Inc., 1993.
2. Cook., S.A., *The complexity of theorem-proving procedures*, Proc. 3<sup>rd</sup> Annual ACM Symp. Theory of Computing, 1971, pp. 151-158.
3. Cormen, T.H. ; Leiersson, C.E. ; Rivest, R.R., *Introducere în algoritmi*, Computer Libris Agora, Cluj, 2000.
4. Garey, M.R. ; Johnson, D.S. ; *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1979.
5. Graham, R.L. ; Knuth, D.E. ; Patashnik O., *Concrete Mathematics*, Addison-Wesley, 1994.
6. Ivașc, C. ; Prună, M., *Bazele informaticii (Grafuri și elemente de combinatorică)*, Proiect de manual pentru clasa a X-a, Petripon, 1995.
7. Ivașc, C. ; Prună, M., *Tehnici de programare. Aplicații*, Petripon, 1999.
8. Karp, R.M., „On the computational complexity of combinatorial problems”, *Networks* 5, pp. 45-68, 1975.
9. Kerningham, B.W. ; Richie, D.M., *The C Programming language (2<sup>nd</sup> Edition)*, Prentice Hall, 1988.
10. Knuth, D.E., *Tratat de programarea calculatoarelor*, vol. I (Algoritmi fundamentali), Teora, București, 2000.
11. Knuth, D.E., *Tratat de programarea calculatoarelor*, vol. III (Sortare și căutare), Teora, București, 2000.
12. Logofătu, D. , C++. *Probleme rezolvate și algoritmi.*, Polirom, Iași, 2001.
13. Logofătu, D., „Programare orientată obiect : de la o problemă de codificare la elemente POO cu C++”, *GInfo*, pp. 40-44, 15/4 2005 (mai).
14. Logofătu, D., „Programare orientată obiect : problema cutiilor speciale și elemente POO cu C++”, *GInfo*, pp. 27-30, 15/5 2005 (iunie).
15. Logofătu, D., „Trei metode de aproximare a constantei  $\pi$ ”, *GInfo*, pp. 36-41, 15/6 2005 (octombrie).
16. Logofătu, D. ; Drechsler, R., *Efficient Evolutionary Approaches for the Data Ordering Problem with Inversion*, 3rd European Workshop on Hardware Optimisation Techniques (EvoHOT), 2006.
17. Michalewicz, Z. ; Fogel, B. David., *How To Solve It : Modern Heuristics*, Springer Verlag, Berlin, 2004.
18. Năstăescu, C. ; Niță, C. ; Popa, S., *Algebra*, manual pentru clasa a X-a, Editura Didactică și Pedagogică, București, 1997.
19. Negrescu, L., *Limbajul C*, Computer Libris-Agora, Cluj, 1997.
20. Rancea, D., *Limbajul Pascal*, Manual clasa a IX-a, Computer Libris Agora, 1997.
21. Rancea, D., *Limbajul Pascal. Algoritmi fundamentali*, manual pentru clasa a X-a, Computer Libris Agora, 1999.

22. Sipser, M., *Introduction to the Theory of Computation*, PWS-Kent, Belmont, California, 1997.
23. Stoilescu, D., *Manual de C/C++*, Radial, Galați, 1999.
24. [www.answers.com](http://www.answers.com)
25. [www.mathworld.com](http://www.mathworld.com)
26. [http://xxi.ac-reims.fr/blaise-pascal/pascal\\_qui\\_es\\_tu.htm](http://xxi.ac-reims.fr/blaise-pascal/pascal_qui_es_tu.htm)
27. <http://www.apologetics.org/articles/pascal.html>
28. [http://www.oeaw.ac.at/deutsch/aktuell/veranstaltungen/veranstaltung\\_661.html](http://www.oeaw.ac.at/deutsch/aktuell/veranstaltungen/veranstaltung_661.html)
29. <http://www.nectec.or.th/courseware/computer/computer-history/0018.html>
30. <http://www.biografiasyvidas.com/biografia/b/babbage.htm>
31. [http://www.anna-schmidt-schule.de/FB3/FIT/FIT\\_19.HTM](http://www.anna-schmidt-schule.de/FB3/FIT/FIT_19.HTM)
32. <http://www.computerhistory.org/tdih/index.php?seldate=II,2,1815>
33. <http://www.doug-long.com/bush.htm>
34. <http://kaluza.physik.uni-konstanz.de/a/gesch/gesch.htm>
35. <http://www.research.att.com/~njas/doc/shannon.html>
36. [http://www.atariarchives.org/deli/Time\\_Line.php](http://www.atariarchives.org/deli/Time_Line.php)
37. <http://radio.weblogs.com/0105910/2004/06/07.html>
38. <http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML>
39. [http://redescolar.ilce.edu.mx/redescolar/act\\_permanentes/mate/mate4n.htm#neu](http://redescolar.ilce.edu.mx/redescolar/act_permanentes/mate/mate4n.htm#neu)
40. <http://users.informatik.haw-hamburg.de/~owsnicki/celebs.html>
41. <http://www.mingyuen.edu.hk/history/4greece/08famous/euclid.jpg>
42. <http://www.matematicamente.it/immagini/eratosthenes.jpg>

**La Editura POLIROM**

**au apărut :**

Mircea Băduț - *Calculatorul în trei timpi*

Luminița Fînaru, Ioan Brava - *Visual Basic. Primii pași... și următorii*

Emanuela Cerchez, Marinel Șerban - *PC. Pas cu pas* (ediția a II-a)

Ştefan Tanasă, Cristian Olaru, Ştefan Andrei - *JAVA de la 0 la expert*

Emanuela Cerchez, Marinel Șerban - *Informatică. Culegere de probleme pentru liceu*

Mihai Cioată - *ActiveX. Concepțe și aplicații*

Doina Hrinciuc Logofătu - *C++. Probleme rezolvate și algoritmi*

Emanuela Cerchez, Marinel Șerban - *Programarea în limbajul C/C++ pentru liceu*

Mircea Băduț - *AutoCAD-ul în trei timpi. Ghidul proiectării profesionale*

Emanuela Cerchez, Marinel Șerban - *Programarea în limbajul C/C++ pentru liceu. Metode și tehnici de programare*

Traian Anghel - *Programarea în limbajul PHP. Ghid practic*

Doina Logofătu - *Bazele programării în C. Aplicații*

**în pregătire :**

Marius Mărușteri, Sabin Buraga, Dragoș Acostăchioae - *Primii pași în Linux*

