Fişiere de tip text şi prelucrări simple

Capitolul 2

- ❖ **Exploatarea unui fişier de tip text**
- ❖ **Prelucrări simple ale datelor citite din fişiere**
- ❖ **Determinarea elementului minim/maxim**
- ❖ **Implementări sugerate**
- ❖ **Probleme propuse**
- ❖ **Soluţiile problemelor**

Un fişier de tip text este o colecţie de înregistrări unde înregistrările sunt caractere (sau şiruri de caractere) structurate pe linii. Liniile sunt separate prin marcaje formate din caracterele „sfârşit de linie" (#10#13 adică LF+CR).

Un astfel de fişier se poate crea cu editorul din mediul de programare, prin program sau cu orice alt editor cu care se pot crea fişiere ASCII.

Lungimea liniilor fiind variabilă, nu se poate realiza un acces direct la o anumită linie din fişier.

2.1. Exploatarea unui fişier de tip text

Variabila de lucru care se asociază prin program unui fişier de pe suport se declară în secţiunea de declaraţii **`var`**:

identificator`:Text;`

Procedura predefinită care realizează legătura între o variabilă de tip fişier şi fişierul existent pe suport este `Assign(`*identificator, nume_fişier*`)` unde *nume_fişier* este o expresie de tip *şir de caractere* (evident poate fi o variabilă sau o constantă de tip şir de caractere).

2.1.1. Deschidere/Închidere

Procedurile predefinite create pentru a deschide, respectiv închide un fişier de tip text identificat prin variabila `f` sunt:

- `ReWrite(f)`: apelul acestei proceduri are ca efect crearea fişierului sau rescrierea acestuia dacă el există; altfel spus, dacă un fişier de tip `Text` este deschis cu `ReWrite`, vom putea scrie în acesta;

- `Reset(f)`: apelul acestei proceduri are ca efect deschiderea fişierului pentru citire, indicatorul de fişier aflându-se la începutul fişierului, adică, după deschiderea lui, poziția curentă pentru citire este prima din fişier;
- `Append(f)`: provoacă poziționarea indicatorului pe marcajul de sfârşit de fişier în vederea adăugării de înregistrări; precizăm că scrierea începe din poziția curentă ceea ce înseamnă suprascrierea marcajului de sfârşit de fişier, apoi continuarea scrierii înregistrărilor dorite în fişier;
- `Close(f)`: realizează închiderea fişierului. Scrierea efectivă a datelor în fişierul de ieşire se realizează pe blocuri (până la „umplerea" unui bloc acestea se păstrează temporar în zone tampon de memorie). Ultimele date (adică blocul început şi neumplut în întregime) se scrie efectiv în fişier la închiderea acestuia. Rezultă că dacă nu închidem fişierul de ieşire s-ar putea ca în fişier să se transfere doar o parte din rezultate, sau în cazul în care aceste rezultate făceau parte dintr-un prim bloc început şi neumplut, să avem fişier vid.

2.1.2. Intrări/Ieşiri

Dintr-un/într-un fişier de tip text se pot citi sau se pot scrie caractere, şiruri de caractere, numere întregi şi numere reale. Subprogramele care realizează citirea şi scrierea sunt:

- `Read(f,`a_1`,`a_2`,...,`a_n`)`: are ca efect citirea a *n* date din fişier;
- `ReadLn(f,`a_1`,`a_2`,...,`a_n`)`: are ca efect citirea a *n* date din fişier, după care următoarea citire se va face de pe linie nouă;
- `Write(f,`a_1`,`a_2`,...,`a_n`)`: se realizează scrierea a *n* date în fişier;
- `WriteLn(f,`a_1`,`a_2`,...,`a_n`)`: se realizează scrierea a *n* date în fişier şi a marcajului de sfârşit de linie;
- `Rename(f,nume_nou)`: redenumeşte un fişier la nivel fizic, (adică pe suport);
- `Eof(f)`: funcție booleană care returnează valoarea `true` dacă următorul caracter care ar urma să fie citit din fişier este marcajul de sfârşit de fişier şi `false` altfel.
- `EoLn(f)`: funcție booleană care întoarce valoarea `true` dacă următorul caracter care ar urma să fie citit din fişier este marcajul de sfârşit de linie sau marcajul de sfârşit de fişier şi `false` în caz contrar.

2.2. Prelucrări simple ale datelor citite din fişiere

Prin prelucrare simplă vom înțelege calcularea unor *sume*, *produse*, *medii* (aritmetice sau geometrice), *căutarea* unui element având o anumită proprietate, *numărarea* elementelor având o anumită proprietate.

2.2.1. Sume şi produse

Orice sumă trebuie iniţializată cu elementul neutru faţă de adunare şi anume cu 0. Apoi, cu o prelucrare simplă se calculează suma. Singura problemă care poate să apară se referă la faptul că mediile de programare nu ne atenţionează când o astfel de sumă sau produs depăşeşte valoarea maximă posibilă de reprezentat în cadrul tipului variabilei sumă. În anumite medii de programare există totuşi posibilitatea de a cere controlarea depăşirii. De exemplu, dacă în cazul următorului program simplu, poziţionăm opţiunea *Options/Compiler/Range checking* pe *ON*, execuţia acestuia se va întrerupe cu un mesaj de eroare: `Range check error`, altfel se afişează un rezultat eronat ($s = 128$) fără nici o atenţionare.

```
Program Test;
var x,s:Byte;
Begin
  s:=0;
  for x:=1 to 255 do s:=s+x;              { s nu va „încape” pe un octet }
  WriteLn(s)
End.
```

Dar un program care se întrerupe cu un mesaj de eroare nu ne poate mulţumi. Chiar dacă nu se poate calcula suma de mai sus, programul nostru trebuie să se finalizeze ca un produs funcţional şi să ne comunice fie rezultatul corect, fie să ne comunice printr-un mesaj prevăzut pentru asemenea cazuri că nu a fost posibilă finalizarea calculelor.

Problema protecţiei faţă de depăşiri se poate înlătura declarând variabila sumă de un tip de date „sigur”. Dar pentru cazurile în care această soluţie nu este la îndemână, ne putem feri cu o decizie simplă. De exemplu, să presupunem că în cazul programului de mai sus vrem să ştim care este valoarea cea mai mare a variabilei *x* care încă se mai poate aduna, fără să se producă depăşire. Efectul programului următor va fi afişarea valorii 23.

```
Program Test;
var x,s:Byte;
    prea_mare:Boolean;
Begin
  s:=0; x:=1;
  prea_mare:=false;
  while not prea_mare and (x<=255) do
    if s <= 255 - x then begin
      s:=s+x;
      Inc(x)
    end else prea_mare:=true;
  if prea_mare then WriteLn('Ultima data s-a adunat ',x)
  else WriteLn(s)
End.
```

2.2.2. Medii

Pentru a calcula o medie aritmetică, mai întâi se calculează o sumă. Dacă numărul termenilor din sumă se cunoaşte, media se calculează împărţind suma la acest număr. Dar se întâmplă frecvent că în sumă se adună doar anumite numere, de exemplu, doar cele pozitive. Acestea se vor număra, în paralel cu calcularea sumei. Să nu uităm să întrebăm înainte de împărţire, dacă numărul termenilor din sumă nu cumva este 0, pentru a evita o împărţire nepermisă. De asemenea, în acest caz, fie afişăm un mesaj, fie realizăm sarcinile specificate în enunţul problemei.

2.2.3. Căutarea secvenţială

Putem căuta un element având o valoare cunoscută sau având o anumită proprietate. Rezultatul va fi un răspuns: „da, l-am găsit" sau „nu l-am găsit". Referitor la această subproblemă dorim să atragem atenţia asupra faptului că o valoare căutată şi găsită nu trebuie căutată în restul datelor. Drept consecinţă nu vom căuta cu **for**, ci cu o structură repetitivă cu număr necunoscut de paşi. În algoritmul următor presupunem că se caută o valoare dată, denumită *căutat*, printre date de acelaşi tip, citite dintr-un fişier.

```
Algoritm căutare:
  citeşte căutat
  citeşte x
  cât timp nu urmează marca de sfârşit de fişier şi (x ≠ căutat) execută:
    citeşte x
  sfârşit cât timp
                    { am ieşit din ciclu fie din cauza că s-a găsit elementul căutat, }
                  { fie din cauza că urmează marca de sfârşit de fişier; în concluzie }
          { trebuie să aflăm care din cele două cazuri a condus la părăsirea ciclului }
  dacă x = căutat atunci
    scrie 'L-am găsit.'
  altfel
    scrie 'Nu l-am găsit.'
  sfârşit dacă
sfârşit algoritm
```

Dacă alegem versiunea în care lucrăm cu o variabilă logică, compararea elementului curent cu cel căutat o facem explicit în corpul structurii repetitive. De asemenea, dacă dintr-un motiv oarecare vrem totuşi să lucrăm cu **for**, în Pascal avem posibilitatea să întrerupem căutarea în momentul găsirii elementului căutat, apelând subprogramul `Break`.

2.3. Determinarea elementului minim/maxim

În foarte multe probleme intervine o subproblemă în care se cere determinarea celui mai mic sau celui mai mare element dintr-o mulțime de date. Conținutul unui fişier poate fi privit ca fiind o astfel de mulțime, bineînțeles, cât timp ne referim la un conținut format din acelaşi tip de date.

Prezentarea algoritmului o vom face pentru determinarea valorii minime dintre mai multe date de tip întreg care se citesc dintr-un fişier de tip text. Variabila desemnată să rețină valoarea minimului (*min*) va fi inițializată fie cu prima valoare citită din fişier, fie cu o valoare „străină", aleasă cu grijă pentru a nu risca să fie cea mai mică în comparație cu datele din fişier. (Această a doua soluție se recomandă doar în cazul în care prima variantă nu este posibilă sau este mult prea anevoioasă.)

În algoritm, în continuare, se citesc pe rând datele din fişier şi fiecare se compară cu valoarea curentă a variabilei *min* care se actualizează după caz.

```
Algoritm Minim:
  citeşte nr
  min ← nr
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte nr
    dacă min > nr atunci
      min ← nr
    sfârşit dacă
  sfârşit cât timp
sfârşit algoritm
```

În cazul maximului vom proceda similar, fiind atenți la operatorul relațional utilizat. Este posibil, ca enunțul să ceară şi *numărul de ordine* al elementului minim (maxim). În acest caz, pe ramura **atunci** din structura alternativă vom adăuga şi actualizarea numărului de ordine care trebuie inițializată cu valoarea 1 (dacă prelucrarea a început cu elementul de indice 1) înainte de a intra în structura repetitivă **cât timp**. În acest fel, chiar dacă în fişier se află mai multe date egale cu valoarea cea mai mică, numărul de ordine va fi al primului astfel de număr.

Dacă ni se cere ultimul număr egal cu valoarea minimă, trebuie doar să modificăm operatorul relațional $>$ în $\geq$.

Dacă trebuie să afişăm *toate* numerele de ordine ale datelor care sunt egale cu minimul, atunci trebuie să recurgem la o altă tehnică. Imediat după inițializarea variabilei *min*, scriem valoarea lui şi numărul de ordine 1 *în fişierul de ieşire*. Urmează compararea celorlalte date cu *min*. Dacă numărul curent este mai mare decât *min*, nu trebuie să facem nimic. În caz contrar vom întreba dacă numărul curent este egal cu *min*. În caz afirmativ, deschidem fişierul de ieşire pentru *adăugare* (în Pascal cu `Append`) şi scriem în el numărul de ordine al elementului care a avut valoarea egală cu *min*. Dacă nu-

mărul curent este mai mic decât *min*, deschidem fişierul pentru *scriere* (în Pascal cu `ReWrite`) şi scriem noua valoarea a minimului şi numărul de ordine corespunzător. Astfel, practic ştergem vechiul conţinut al fişierului scriind în el rezultatul care, conform stadiului prelucrării, este corect. În final, vom avea unul sau mai multe numere de ordine care corespund elementelor egale cu valoarea minimă scrisă în fişier.[*)]

2.4. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie rezolvate problemele în care intervin operaţii cu fişier de tip text, vă sugerăm să încercaţi să implementaţi algoritmi pentru:

1. citirea unui număr dintr-un fişier şi scrierea sa în altul;
2. citirea tuturor numerelor dintr-un fişier care conţine câte un număr pe o linie şi scrierea lor în alt fişier, câte unul pe o linie;
3. citirea datelor dintr-un fişier care conţine numere, mai multe pe o linie, şi scrierea lor într-un alt fişier în aceeaşi formă;
4. citirea unei secvenţe de numere dintr-un fişier şi scrierea ei în alt fişier;
5. citirea unor nume (şiruri de caractere) dintr-un fişier şi scrierea lor într-un alt fişier;
6. citirea unui număr natural *N* dintr-un fişier şi scrierea într-un alt fişier a tuturor numerelor naturale cuprinse între 0 şi *N*, câte unul pe o linie;
7. citirea unui număr natural *N* dintr-un fişier şi a *N* nume (aflate pe următoarele linii ale fişierului) şi scrierea celor *N* nume într-un alt fişier;
8. copierea integrală a conţinutului unui fişier într-un alt fişier;
9. redeschiderea fişierelor atunci când este necesar.

Pentru a vă familiariza cu prelucrări simple, cum sunt calculul sumelor, determinarea minimului/maximului, vă sugerăm să încercaţi să implementaţi algoritmi pentru:

- calcularea sumei primelor *N* numere naturale;
- calcularea sumei unor date păstrate într-un fişier de tip text;
- determinarea minimului/maximului dintr-un şir de date păstrate într-un fişier de tip text;
- iniţializarea minimului/maximului (cu elementul neutru sau cu primul element);
- citirea unor numere dintr-un fişier şi scrierea într-un fişier a minimului lor precum şi a primei poziţii pe care apare acest minim;
- citirea unor numere dintr-un fişier şi scrierea într-un fişier a minimului lor precum şi a ultimei poziţii pe care apare acest minim;
- citirea unor numere dintr-un fişier şi scrierea într-un fişier a minimului lor precum şi a tuturor poziţiilor pe care apare acest minim.

[*)] Aceşti algoritmi se vor trata din nou în cazul în care datele vor fi structurate în tablouri.

2.5. Probleme propuse

2.5.1. Număr par

Stabiliți dacă printre numerele întregi scrise într-un fişier de tip text există sau nu cel puțin un număr par.

Date de intrare
În fişierul de intrare **PAR.IN** se află mai multe numere întregi.

Date de ieşire
Dacă în fişier există cel puțin un număr par, în fişierul de ieşire **PAR.OUT** se va scrie `'DA'`, altfel se va scrie `'NU'`.

Restricții şi precizări
- $0 \leq$ *număr* ≤ 1000000000.

Exemplu

PAR.IN	**PAR.OUT**
1 2 3	DA

2.5.2. Numere pare pătrate perfecte în fişier

Într-un fişier sunt scrise mai multe numere naturale. Alegeți numerele pare care sunt pătrate perfecte şi scrieți-le într-un alt fişier.

Date de intrare
Numerele sunt scrise în fişierul de intrare **NUMERE.IN** pe mai multe rânduri. Pe un rând sunt scrise un număr oarecare de numere, separate prin unul sau mai multe spații.

Date de ieşire
Numerele pare care sunt pătrate perfecte se vor scrie câte unul pe linie, în fişierul de ieşire **NUMERE.OUT**.

Restricții şi precizări
- $0 \leq$ *număr* ≤ 1000000000.

Exemplu

NUMERE.IN	**NUMERE.OUT**
12 4 24 100 25 900	4 100 900

2.5.3. Concatenare de fişiere

Se consideră două fişiere de tip text **F1.IN** şi **F2.IN**. Să se creeze un al treilea fişier de tip text **F3.OUT** care să conţină toate datele din **F1.IN**, apoi întreg conţinutul fişierului **F2.IN**.

Date de intrare

Fişierele sunt specificate prin numele lor, **F1.IN** şi **F2.IN** şi au conţinuturi oarecare pe care le vom privi ca fiind caractere. Acestea se vor citi începând cu primul şi terminând cu ultimul, până la marca de sfârşit de fişier.

Date de ieşire

Fişierul **F3.OUT** va fi format din datele din fişierul **F1.IN**, apoi o linie vidă, după care urmează datele din fişierul **F2.IN**.

Restricţii şi precizări

- o linie din fişier conţine cel mult 255 de caractere.

Exemplu

F1.IN
```
Ana are mere.
23 24
156432
```

F2.IN
```
123 c 123
Astăzi plouă.
```

F3.OUT
```
Ana are mere.
23 24
156432
linie vidă
123 c 123
Astăzi plouă.
```

2.5.4. Caractere

Se consideră două numere naturale *a* şi *b* care reprezintă capetele unui interval de numere întregi şi un număr oarecare de caractere. Alegeţi dintre aceste caractere pe acelea care au codurile ASCII cuprinse în intervalul [*a*, *b*].

Date de intrare

Pe prima linie a fişierului **CARACTER.IN** se află numerele *a* şi *b*, separate printr-un spaţiu. Pe fiecare dintre următoarele linii sunt scrise mai multe caractere.

Date de ieşire

Caracterele care corespund cerinţei problemei se vor scrie în fişierul **CARACTER.OUT** pe aceeaşi linie, separate prin câte un spaţiu.

Restricţii şi precizări

- $0 \le a \le b \le 255$.

Exemplu

CARACTER.IN
```
50 124
Ana are 10 mere
```

CARACTER.OUT
```
A n a a r e m e r e
```

2.5.5. Medii

Cunoscând mediile semestriale la disciplina *informatică* ale unor elevi dintr-o clasă, să se determine media semestrială a clasei la această disciplină.

Date de intrare

În fişierul **MEDII.IN** sunt scrise mediile elevilor, câte un număr real pe fiecare linie.

Date de ieşire

Media clasei se va scrie în fişierul de ieşire **MEDII.OUT**. Dacă, din nefericire, nu se poate calcula media, deoarece toţi elevii au rămas corigenţi, în fişier se va scrie mesajul: `'Nu se poate calcula media.'`

Restricţii şi precizări

- mediile sunt numere reale, cu două zecimale exacte;
- $3 \leq medie \leq 10$;
- mediile mai mici decât 5 nu participă la calcule.

Exemplu

MEDII.IN
```
8.95
9.50
4.67
6.88
```

MEDII.OUT
```
8.44
```

Explicaţie
Media 4.67 nu participă la calcule. Media aritmetică s-a calculat din 3 medii.

2.5.6. Factorial

Să se calculeze produsul primelor N numere naturale (factorialul)!

Date de intrare

În fişierul de intrare **FACT.IN** este scris un singur număr natural.

Date de ieşire

Valoarea factorialului se va scrie în fişierul de ieşire **FACT.OUT**.

Restricţii şi precizări

- $5 \leq N \leq 18$

Exemplu

FACT.IN	**FACT.OUT**
5	120

2.5.7. Vârste

Dintr-un fişier de tip text se citesc numere naturale, reprezentând anii de naştere a unor persoane. Afişaţi anul de naştere şi numărul de ordine a liniei din fişier pe care se află anul naşterii al celei mai tinere persoane. Procedaţi la fel în cazul celei mai vârstnice persoane.

Date de intrare

Pe fiecare linie a fişierului **ANI.IN** se află un număr întreg, reprezentând anul naşterii unei persoane.

Date de ieşire

Pe prima linie a fişierului de ieşire **ANI.OUT** se va scrie anul de naştere a celei mai tinere persoane. Pe linia a doua se va scrie numărul de ordine a celei mai tinere persoane. Pe a treia linie se va scrie anul de naştere a celei mai vârstnice persoane. Pe a patra linie se scrie numărul de identificare a celei mai vârstnice persoane. Dacă într-un acelaşi an s-au născut mai multe persoane, se vor furniza toate numerele de ordine corespunzătoare, despărţite prin câte un spaţiu.

Restricţii şi precizări

- $1900 \leq an \leq 2003$.

Exemplu

ANI.IN	**ANI.OUT**
1991	2002
1998	4 7
2000	1978
2002	5 6
1978	
1978	
2002	

2.5.8. Caractere

Se consideră un fişier de tip text care conţine caractere. Afişaţi, în ordinea în care apar în fişier acele caractere care sunt mai mari decât ultimul caracter din fişier.

Date de intrare

Pe fiecare linie a fişierului de intrare **CARACTER.IN** se află un caracter.

Date de ieşire
Pe prima linie a fişierului de ieşire **CARACTER.OUT** se va scrie ultimul caracter citit din fişierul de intrare. Pe următoarea linie se vor scrie caracterele cerute, separate prin câte un spațiu.

Exemplu

CARACTER.IN
```
f
z
3
s
w
r
t
1
n
j
```

CARACTER.OUT
```
j
z s w r t n
```

2.5.9. Pare, impare

Dintr-un fişier de tip text se citesc mai multe numere naturale de pe fiecare linie. Pentru fiecare linie din fişier afişați cea mai mică valoare pară şi cea mai mare valoare impară precizând şi numărul de ordine a liniei.

Date de intrare
Fiecare linie a fişierului de intrare **LINII.IN** conține numere naturale, separate prin câte un spațiu.

Date de ieşire
Fişierul de ieşire **LINII.OUT** va avea tot atâtea linii câte linii există în fişierul de intrare. Pe fiecare linie vor fi scrise două numere care reprezintă cel mai mic număr par şi cel mai mare număr impar de pe linia corespunzătoare a fişierului de intrare.

Restricții şi precizări
- $0 < \textit{număr} \leq 32767$;
- pe fiecare linie din fişierul de intrare există atât numere pare, cât şi impare.

Exemplu

LINII.IN
```
1 2 3 4 5 2 3 4
2 2 2 2 3 3 3
1 3 4 5 66 77 23
12 23 134 1234 12
```

LINII.OUT
```
2 5
2 3
4 77
12 23
```

2.6. Soluțiile problemelor propuse

2.6.1. Număr par

Această problemă cere să stabilim dacă printre o mulțime de numere apare cel puțin un număr par.

```
Algoritm Există_Pare_1:
  găsit ← fals
  cât timp nu am ajuns la sfârşitul fişierului şi nu găsit execută:
    citeşte nr                                          { citim alt număr }
    găsit ← găsit sau nr este par
  sfârşit cât timp
  dacă găsit atunci
    scrie 'DA'
  altfel
    scrie 'NU'
sfârşit algoritm
```

Dacă vrem să evităm folosirea variabilei logice `găsit`, putem proceda în felul următor:

```
Algoritm Există_Pare_2:
  citeşte nr
  cât timp nu am ajuns la sfârşitul fişierului şi numărul este impar execută:
    citeşte nr                                          { citim alt număr }
  sfârşit cât timp
  dacă nu am ajuns la sfârşitul fişierului sau nr este par atunci
      { dacă nu urmează marca de sfârşit de fişier, înseamnă că am găsit un număr }
   { par, altfel mai este posibil ca nr din faţa mărcii de sfârşit de fişier să fi fost par }
    scrie 'DA'
  altfel
    scrie 'NU'
sfârşit algoritm
```

2.6.2. Numere pare pătrate perfecte în fişier

Variabilele `f` şi `g` de tip `Text` se vor asocia fişierelor **`NUMERE.IN`**, respectiv **`NUMERE.OUT`**. Numele fişierelor precizate în enunț sunt declarate sub forma unor constante simbolice la începutul programului. Bineînțeles, aceste constante de tip şir de caractere s-ar fi putut scrie ca parametri actuali în apelul procedurilor `Assign`. Primul fişier se deschide cu `Reset(f)`, ceea ce înseamnă că se va parcurge de la început în scopul citirii datelor scrise în el. Al doilea fişier se deschide cu `ReWrite(g)` pentru scriere.

Se parcurge primul fişier cât timp acesta conţine date (până la sfârşitul fişierului). Pentru fiecare număr citit din fişier se verifică dacă este pătrat perfect şi dacă este par. Dacă aceste condiţii sunt îndeplinite, numărul se va scrie în fişierul de ieşire.

Algoritmul care realizează aceste operaţii este:

```
Algoritm Numere_pare_pătrate_perfecte:
  cât timp nu urmează marcajul de sfârşit de fişier execută:
    citeşte n                                          { din fişier }
    dacă n este număr par atunci
      nr ← 2
      cât timp nr*nr < n execută:
        nr ← nr + 2
      sfârşit cât timp
      dacă nr*nr = n atunci
        scrie n                                        { în fişier }
      sfârşit dacă
    sfârşit dacă
  sfârşit cât timp
sfârşit algoritm
```

Verificarea proprietăţii de pătrat perfect se realizează generând, pe rând pătratele numerelor naturale, începând de la 2, deoarece nu are rost să generăm pătratele unor numere impare, acestea fiind şi ele impare. Cât timp un astfel de pătrat este mai mic decât numărul, generăm următorul pătrat. Dacă acesta nu este mai mic, poate fi egal cu numărul dat sau poate fi mai mare decât el. În caz de egalitate, scriem numărul în fişierul de ieşire.

2.6.3. Concatenare de fişiere

Principalele obiective pe care ni le propunem în rezolvarea acestei probleme sunt:

- folosirea corectă a subprogramelor predefinite pentru fişiere: `Assign`, `Reset`, `ReWrite`, `Append`, `Close`;
- parcurgerea corectă a unei linii şi citirea marcajului de sfârşit de linie;
- citirea şi scrierea corectă din şi respectiv într-un fişier de tip text;
- realizarea scrierii unei linii vide în fişier.

Se parcurg pe rând conţinuturile celor două fişiere, caracter cu caracter. Fiecare caracter citit se scrie în fişierul **`F3.OUT`**. Când se întâlneşte marcajul de sfârşit de linie în fişierul de ieşire se va face salt la linie nouă. După ce primul din cele două fişiere de intrare a fost parcurs în întregime, se scrie o linie vidă în fişierul de ieşire.

Prezentăm citirea datelor din fişierul **`F1.IN`** şi scrierea acestora în **`F3.OUT`**, precum şi scrierea liniei vide. Variabila `c` este de tip caracter.

```
Algoritm concatenare:
  cât timp nu urmează marca de sfârşit de fişier în f1 execută:
    cât timp nu urmează marca de sfârşit de linie în f1 execută
      citeşte c                          { se citeşte un caracter din fişierul f1 }
      scrie c                                    { se scrie caracterul în f3 }
    sfârşit cât timp
    citeşte marcajul de sfârşit de linie în f1
    scrie marcaj de sfârşit de linie în f3
  sfârşit cât timp
  scrie marcaj de sfârşit de linie  în f3 (se lasă o linie vidă)
  Se copiază în aceeaşi manieră conţinutul fişierului F2.IN în F3.OUT.
sfârşit algoritm
```

2.6.4. Medii

Problema cere determinarea unei medii aritmetice. În paralel cu citirea vom număra termenii sumei pe baza căreia se va determina media aritmetică, deoarece nu cunoaştem numărul datelor. Nu este suficient să numărăm pur şi simplu datele, deoarece eventualele medii mai mici decât 5, nu intră în calcule. În final suma se împarte la numărul termenilor, dacă acest număr este diferit de 0.

```
Algoritm Medii:
  s ← 0                                    { element neutru pentru adunare }
  nr ← 0                                          { numărul termenilor }
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte medie                                  { media curentă în fişier }
    dacă medie ≥ 5 atunci{ doar mediile mai mari sau egale cu 5 interesează }
      nr ← nr + 1                              { creşte numărul termenilor }
      s ← s + medie                             { adunăm termenul curent }
    sfârşit dacă
  sfârşit cât timp
  dacă nr ≠ 0 atunci  { dacă avem cel puţin o medie mai mare sau egală cu 5 }
    scrie s/nr                                    { afişăm media aritmetică }
  altfel
    scrie 'Nu se poate calcula media.'
sfârşit algoritm
```

2.6.5. Factorial

În această problemă vom citi valoarea lui *n* şi pe baza lui, cu ajutorul unei instrucţiuni **pentru**, vom genera toate numerele naturale.

```
Algoritm Fact:                          { în p vom obține valoarea factorialului }
  p ← 1                                   { element neutru pentru înmulțire }
  citeşte n
  pentru i=1,n execută:
    p ← p * i                             { înmulțim cu factorul curent }
  sfârşit pentru
sfârşit algoritm
```

Dar să facem abstracție pentru moment de precizarea din enunț cu privire la limitele lui *n*. Să verificăm programul de mai sus pe rând pentru valorile 5, 8, ..., 25, ... 40, poate şi 50. În funcție de tipul variabilei *p* se pot întâmpla lucruri interesante. De exemplu, dacă rezultatul este declarat ca fiind de tipul `Byte`, vom obține `6!=208`, iar `8!=128`. Dar `8!` ar trebui să fie egal cu `720`. Dacă modificăm tipul lui *p* din `Byte` în `Longint` şi rulăm programul pentru $n = 20$, obținem `20!=-2102132736` (număr negativ!), iar pentru $n = 50$, `50!=0`. Dacă o declarăm pe *p* de tip `Real`, programul se întrerupe pentru $n > 33$ cu mesajul de eroare: `Floating point overflow`!

Să luăm în considerare situația în care se preferă ca rezultatul să fie de tip `Longint`. Evident, încercările ne-au convins, că începând cu o anumită valoare a lui *n* se produce depăşire. Mai ales atunci când trebuie calculat factorialul unui *n* care constituie un rezultat calculat în program, ne trebuie „un instrument" care să ne protejeze de neplăceri, cum ar fi obținerea unui rezultat fals. Cel mai mare număr de tipul `Longint`, posibil de reprezentat în calculator este `2147483647`, reținut de constanta simbolică a unit-ului *System*, numită în Pascal `MaxLongint`. Ar urma să putem pune o întrebare de genul **`if`** `p > MaxLongint` **`then...`** Dar este absurd să întrebăm dacă un număr este mai mare decât cel mai mare număr posibil...

În consecință, vom pune întrebarea cu un pas înainte de efectuarea înmulțirii curente care ar putea produce o depăşire în felul următor:

```
...
  p:=1;
  i:=1;
  while (p<>0) and (i<=n) do begin
    if p > MaxLongint div i then
      p:=0
    else begin
      p:=p*i;
      Inc(i)
    end
  end
...
```

Valoarea lui *p* se va testa în blocul apelant pentru a decide dacă s-a putut calcula factorialul cerut, sau nu, deoarece s-ar fi produs o depăşire.

2.6.6. Caractere

În această problemă în fişierul de intrare avem caractere ASCII care pot fi cifre, litere sau alte semne. Având în vedere că dintre acestea trebuie să „culegem" caracterele având codul ASCII în intervalul [*a*, *b*], vom analiza fiecare caracter citit din fişierul de intrare şi dacă acesta are codul ASCII în intervalul dat, îl copiem în fişierul de ieşire.

Fie, de exemplu, conținutul fişierului de intrare **CARACTER.IN**:

```
50 125
Ana are 10 mere
```

Intervalul este [50, 125]. Se parcurg caracterele de pe a doua linie şi se compară, pe rând, codurile ASCII ale caracterelor citite cu capetele intervalului. Codul ASCII al unui caracter `c` se poate obține (în Pascal) folosind funcția `Ord(c)`. De exemplu, `Ord('A')=65`, `Ord('n')=110` etc.

Pentru fişierul dat ca exemplu, doar caracterele `'0'`, `'1'` şi caracterul spațiu care au codurile ASCII `48`, `49`, respectiv `32` nu se află în intervalul cerut. În concluzie, în fişierul de ieşire vom avea următoarele caractere, separate prin câte un spațiu:

```
A n a a r e m e r e
```

```
Algoritm Caractere:
  citeşte a,b
  cât timp nu urmează marcajul de sfârşit de fişier în CARACTER.IN execută:
    cât timp nu urmează marcajul de sfârşit de linie în CARACTER.IN execută:
      citeşte c
      dacă codul ASCII al caracterului c ≥ a şi
           codul ASCII al caracterului c ≤ b atunci
        scrie c,' '                          { scriem în fişierul de ieşire }
      sfârşit dacă
    sfârşit cât timp
    citeşte marcajul de sfârşit de linie din CARACTER.IN
  sfârşit cât timp
sfârşit algoritm
```

2.6.7. Vârste

Obiectivele propuse în această problemă sunt:

- aflarea valorii minime şi maxime dintre valorile păstrate într-un fişier de tip text;
- căutarea acestor valori în fişier şi raportarea pozițiilor pe care valoarea minimă, respectiv maximă apare în fişier;
- parcurgerea fişierului text de mai multe ori, ceea ce presupune închiderea şi deschiderea lui în mod repetat.

Aflarea celei mai tinere persoane se reduce la aflarea maximului din fişier şi se realizează în algoritmul următor:

```
Algoritm Maxim:
  citeşte n
  max ← n
  cât timp nu urmează marcajul de sfârşit de fişier execută:
    citeşte n
    dacă max < n atunci
      max ← n
    sfârşit dacă
  sfârşit cât timp
  scrie max
sfârşit algoritm
```

După ce s-a scris pe prima linie a fişierului de ieşire valoarea maximului, urmează să se regăsească această valoare în fişierul de intrare. Regăsirea se realizează prin redeschiderea fişierului de intrare pentru parcurgere şi căutarea valorii maxime. La fiecare apariţie a maximului în fişierul de intrare se scrie valoarea numărului de ordine a liniei pe care se află o valoare egală cu valoarea maximă şi un spaţiu separator pentru cazul în care se va mai scrie un număr de ordine în fişier. După epuizarea elementelor din fişierul de intrare se scrie marcajul de sfârşit de linie în fişierul de ieşire pentru ca scrierea valorii minime să se realizeze pe rând nou.

Algoritmul este:

```
Algoritm ScrieMaxim:
  i ← 1                    { numărul de ordine al liniei în fişierul de intrare }
  cât timp nu urmează marcajul de sfârşit de fişier execută:
    citeşte n
    dacă n = max atunci
      scrie i,' '
    sfârşit dacă
    i ← i + 1
  sfârşit cât timp
sfârşit algoritm
```

Algoritmul pentru minim este analog cu cel descris mai sus, dar se poate proiecta un algoritm care determină valoarea minimă şi maximă cu o singură parcurgere a fişierului de intrare. Dacă vom lucra astfel, structura repetitivă **cât timp** va conţine prelucrările aferente minimului şi maximului, urmând ca regăsirea lor să se realizeze separat, deoarece numerele de ordine corespunzătoare maximului, respectiv minimului trebuie să le afişăm grupate.

2.6.8. Caractere

Etapele rezolvării acestei probleme sunt:

- citirea integrală a fişierului de intrare şi scrierea ultimului caracter citit în fişierul de ieşire;
- deschiderea fişierului de intrare pentru o a doua parcurgere;
- scrierea în fişierul de ieşire a tuturor caracterelor din fişierul de intrare care sunt mai mari decât ultimul caracter citit (păstrat în variabila *c*).

Pentru găsirea caracterelor mai mari decât *c* în fişierul de intrare, acesta se deschide pentru parcurgere. Se citesc pe rând câte un caracter *b* de pe fiecare linie şi se compară cu *c*. Dacă *b* este mai mare decât *c*, atunci acesta se scrie pe a doua linie în fişierul de ieşire, urmat de caracterul spaţiu.

```
Algoritm Caracter:
  cât timp nu urmează marcajul de sfârşit de fişier execută:
    citeşte c
  sfârşit cât timp
  scrie c                { în c avem ultimul caracter citit din fişierul de intrare }
  cât timp nu urmează marcajul de sfârşit de fişier execută:
    citeşte b
    dacă c < b atunci
      scrie b,' '  { în fişierul de ieşire scriem  caracterele „mai mari” decât c }
    sfârşit dacă
  sfârşit cât timp
sfârşit algoritm
```

2.6.9. Pare, impare

Problema constă în tratarea separată a fiecărei linii din fişierul de intrare în ideea găsirii valorii minime pare şi a valorii maxime impare.

```
Algoritm MinMax:
  cât timp nu urmează marcajul de sfârşit de fişier execută:
    min ← Maxint                  { suntem siguri că vom găsi unul mai mic }
    max ← 0                       { suntem siguri că vom găsi unul mai mare }
    cât timp nu urmează marcajul de sfârşit de linie execută:
      citeşte n
      dacă n este par atunci
        dacă n < min atunci
          min ← n
        sfârşit dacă
      altfel
```

```
          dacă n > max atunci
            max ← n
          sfârşit dacă
        sfârşit dacă
        scrie min,' ',max
      sfârşit cât timp
    sfârşit cât timp
sfârşit algoritm
```