Tehnici de programare

fișiere binare; scriere; citire

Fișierele binare sunt destinate folosirii lor direct de către diverse aplicații. În general ele păstrează informația în același fel în care ea este păstrată în memoria calculatorului sau eventual aplică asupra informației diverși algoritmi de compresie, criptare, normalizare, etc. Din acest motiv ele nu sunt direct folosibile de către oameni, de exemplu pentru citire sau editare. În schimb, aplicațiile pot folosi mult mai ușor fișierele binare decât pe cele text, deoarece nu mai trebuie să convertească datele din binar la text (ex: *“fprintf(fis,”%g”,x);”*) înainte de a le scrie într-un fișier text sau să le convertească din text în binar, la citirea dintr-un fișier text (ex: *“fscanf(fis,”%g”,&x);”*). Mai mult decât atât, unele asemenea conversii duc la pierdere de precizie, ceea ce poate să nu fie acceptabil. De exemplu, un număr de tip *double* are o precizie de ~15 digiți, iar dacă e formatat ca un text de 10 caractere, înseamnă că poate pierde 5 digiți din precizie. Exemple de fișiere binare sunt: arhive (zip, gz, 7z, rar), audio-video (mp3, mp4, avi, mpeg), imagini (jpg, png, gif), executabile sau cod (exe, dll, so, jar). Inclusiv unele formate destinate documentelor (ex: epub, doc (Microsoft Word)) sunt de fapt tot fișiere binare, iar altele (ex: pdf) sunt o combinație de fișiere text cu unele secțiuni binare.

Pentru a vizualiza conținutul unui fișier binar se pot folosi următoarele programe:

- în Linux: *hexdump -C nume_fisier*
- în Windows: *Total Commander*, se selectează fișierul de vizualizat, se dă comanda *View* (tasta F3) și din meniul *Options* se selectează *Hex* (tasta 3)

Fișierele vor fi afișate în forma următoare:

```
00000000: 89 50 4E 47 0D 0A 1A 0A|00 00 00 0D 49 48 44 52  | ëPNG........IHDR
00000010: 00 00 00 64 00 00 00 64|08 02 00 00 00 FF 80 02  | ...d...d..... Ç.
00000020: 03 00 00 00 06 74 52 4E|53 00 FF 00 80 00 00 45  | .....tRNS. .Ç..E
00000030: F7 DB 9D 00 00 00 09 70|48 59 73 00 00 0B 13 00  | ¸■Ł....pHYs.....
00000040: 00 0B 13 01 00 9A 9C 18|00 00 15 AA 49 44 41 54  | .....Üť....¬IDAT
00000050: 78 DA ED 9D 7B 74 14 55|9E C7 BF B7 BA F3 4E 80  | x┌Ýŀ{t.U×ă┐Ě‖ˇNÇ
00000060: 24 10 42 12 9E E1 25 42|30 04 18 DE B0 89 40 0C  | $.B.×ß%B0..Ů▒ë@.
00000070: CA 80 8C 20 0B E2 38 AE|E8 3A 33 EB D9 5D 0F 73  | ╩Çî .Ô8«Ŕ:3Ú┘].s
00000080: 70 07 F5 80 CE 9C B3 7B|5C 74 3C 0A 33 C8 78 40  | p.§Ç╣ť|{\t<.3╚x@
00000090: 10 70 D6 21 2E 42 94 D5|E0 6A 82 8F 55 12 91 81  | .pÍ!.BöŇÓjéĆU.Ĺü
000000A0: 10 08 10 5E 81 06 12 A0|F3 EC AE BA 77 FF A8 47  | ...^ü..á˘ý«‖w ĘG
000000B0: DF AA BA 55 DD 09 24 E0|8C 75 EE E9 73 AB 3A A9  | ▀¬‖UŢ.$Óîuţ Úsź:ę
000000C0: EE FE F4 F7 F7 BB BF FB|BB 8F 26 EC 79 FC 70 44  | ţ■˘¸¸╗ú╖Ć&ýyŘpD
000000D0: 78 48 3F 20 88 FC F0 DE|C2 D7 6E 0C A0 BA DE 53  | xH? ŁŘ-Ů┬În.á‖ŮS
000000E0: 17 88 0B C4 C7 D1 84 28|25 D6 CB 62 24 E6 25 8C  | .ł.-ăĐä(%Í╬b$Š%î
000000F0: 30 C6 28 A8 82 80 82 B6|20 69 96 49 63 30 BA 39  | 0Ă(EéĆéÂ iľIc0‖9
```

Pe fiecare linie sunt afișați câte 16 octeți din fișier. Se începe cu poziția acestora în fișier (offsetul), după care urmează valorile hexazecimale ale octeților și în final reprezentarea lor ASCII, dacă este posibilă. Dacă un octet nu este tipăribil în ASCII, se înlocuiește printr-un punct. Valorile se afișează în baza 16 deoarece fiecare octet se poate reprezenta exact prin 2 digiți hexazecimali, ceea ce ușurează formatarea conținutului.

În C, fișierele binare se deschid și se închid la fel ca cele text, folosind *fopen/fclose*, folosind “wb” sau “rb” la *fopen* (write binary, read binary).

Scrierea într-un fișier binar

Pentru a scrie într-un fișier binar se folosește funcția **fwrite** declarată în *stdio.h*:

size_t **fwrite**(void *elemente, size_t dim_element, size_t nr_elemente, FILE *fisier);

Funcția *fwrite* are următorii parametri:

- **elemente** - un pointer la zona de memorie ce conține elementele care vor fi scrise în fișier
- **dim_element** - dimensiunea în octeți a unui element care va fi scris în fișier. Se poate afla cu *sizeof*
- **nr_elemente** - numărul de elemente de scris
- **fisier** - fisierul destinație

fwrite returnează numărul de elemente scrise integral în fișier. Dacă intervine o eroare (ex: nu mai este spațiu pe disc), numărul returnat va fi mai mic decât *nr_elemente*, posibil chiar 0. Pentru claritatea codului, vom omite în acest laborator testarea dacă *fwrite* a funcționat cu succes.

Exemplul 1: Să se scrie într-un fișier valoarea 0x1177AAEE de tip *int*, așa cum este ea reprezentată în memorie:

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

int main()
{
   FILE *fis;
   int n=0x1177AAEE;

   if((fis=fopen("1.dat","wb"))==NULL){
      printf("nu se poate deschide fisierul\n");
      exit(EXIT_FAILURE);
      }

   fwrite(&n,sizeof(int),1,fis);

   fclose(fis);
   return 0;
}
```

Funcțiile *fopen/fclose* au fost folosite la fel ca la fișierele text: fișierul *1.dat* a fost deschis în mod de scriere cu opțiunea "wb". Dacă el nu există, "wb" îl va crea. Dacă el există, atunci tot conținutul lui anterior va fi șters.

Funcția *fwrite* a fost apelată cu adresa zonei de memorie pe care dorim să o scriem în fișier (*&n* - adresa variabilei *n*), dimensiunea în octeți a tipului *int*, numărul de elemente pe care dorim să le scriem (1) și fișierul destinație.

Dacă programul a funcționat corect, pe disc va apărea fișierul *1.dat*, având dimensiunea de 4 octeți (*sizeof(int)* pe calculatoarele din laborator). Când vizualizăm conținutul acestui fișier, vom constata că el conține octeții: EE, AA, 77, 11 (în hexazecimal). Ordinea acestor octeți este inversă decât cea din număr, deoarece microprocesoarele Intel și AMD păstrează în memorie numerele întregi în format **little-endian**, adică se începe întotdeauna cu octetul cel mai puțin semnificativ. Astfel, octeții numărului sunt de fapt păstrați în memorie de la dreapta la stânga. Denumirea *little-endian* vine de la expresia *"în șir indian"*, adică unul după altul și, în acest caz, primul din șir este octetul cel mai puțin semnificativ.

Exemplul 2: Să se scrie un program care menține o bază de date cu produse, fiecare produs fiind definit prin nume (max 28 caractere) și preț (float). Programul va implementa într-o buclă infinită următorul meniu: 1-adăugare produs, 2-afișare produse, 3-ieșire. La ieșirea din program, baza de date va fi scrisă automat pe disc:

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

typedef struct{
   char nume[28];
   float pret;
   }Produs;

Produs produse[1000];
int nProduse;

void adaugare()
{
   Produs p;
   printf("nume: ");
   fgets(p.nume,28,stdin);
   p.nume[strcspn(p.nume,"\r\n")]='\0';
   printf("pret: ");
   scanf("%g",&p.pret);
   produse[nProduse]=p;
   nProduse++;
}

void afisare()
{
   int i;
   for(i=0;i<nProduse;i++){
      printf("%s: %g\n",produse[i].nume,produse[i].pret);
      }
}

void scriere()
{
   FILE *fis;

   if((fis=fopen("produse.dat","wb"))==NULL){
      printf("nu se poate deschide fisierul\n");
      exit(EXIT_FAILURE);
      }
   fwrite(produse,sizeof(Produs),nProduse,fis);
   fclose(fis);
}

int main()
{
   int optiune;
   do{
      printf("optiune: ");
      scanf("%d",&optiune);
```

```
        getchar();
        switch(optiune){
            case 1:adaugare();break;
            case 2:afisare();break;
            case 3:scriere();break;
            default:printf("optiune invalida\n");
            }
        }while(optiune!=3);
    return 0;
}
```

Se poate constata că pentru scrierea în fișier a întregii baze de date am avut nevoie de o singură instrucțiune *fwrite*. Dacă am fi folosit un fișier text, ar fi fost nevoie să iterăm toate produsele din baza de date și să le scriem pe fiecare în fișier, procesul incluzând și alte operații cum ar fi adăugare de separatori (virgule, \n) sau transformări de tip (de la *float* la șir de caractere). Acesta este un exemplu de ce aplicațiile pot folosi mai ușor și mai eficient formate binare decât formate text.

Dacă adăugăm în baza de date 2 produse (mere:5, paine:3), fișierul *produse.dat* va avea un conținut de forma:

```
00000000: 6D 65 72 65 00 00 CC CC|CC CC CC CC CC CC CC CC  | mere..╠╠╠╠╠╠╠╠
00000010: CC CC CC CC CC CC CC CC|CC CC CC CC 00 00 A0 40  | ╠╠╠╠╠╠╠╠..á@
00000020: 70 61 69 6E 65 00 00 CC|CC CC CC CC CC CC CC CC  | paine..╠╠╠╠╠╠╠╠
00000030: CC CC CC CC CC CC CC CC|CC CC CC CC 00 00 40 40  | ╠╠╠╠╠╠╠╠..@@
```

Deoarece *sizeof(Produs)==32*, fiecare produs va fi reprezentat pe două linii. Se vede că la început este numele produsului, urmat de terminatorul de șir, \0. După aceasta urmează octeți nefolosiți, deoarece un nume este un vector de 28 de caractere. Octeții nefolosiți pot avea orice valori care se aflau în stivă când s-a apelat funcția *adaugare,* rezervându-se astfel memorie pentru variabila locală *p*. În final, ultimii 4 octeți din fiecare produs sunt reprezentarea binară a unui număr de tip float, în format IEEE-754.

Întrebare: De ce după fiecare nume de produs apar doi terminatori de șir?

Aplicația 5.1: Modificați exemplul anterior, astfel încât să se scrie la început de fișier numărul de produse din baza de date.

Scrierea datelor care conțin pointeri

În exemplul de mai sus scrierea în fișier s-a realizat simplu, deoarece un produs este stocat integral în interiorul unei structuri. Dacă s-ar fi dorit ca numele produsului să poată fi oricât de lung, atunci ar fi trebuit în interiorul structurii să avem un pointer la nume, iar acesta să fie alocat dinamic în altă zonă de memorie. În această situație, dacă am scrie o structură în fișier, s-ar scrie în locul numelui adresa de memorie la care pointează pointerul *nume*.

În general, dacă avem structuri de date care conțin pointeri, avem următoarele variante să le scriem în fișier:

1. Scriem separat fiecare componentă a structurii, folosind chiar valoarea efectivă a acelei componente, nu adresa ei. De exemplu, dacă avem un pointer, vom scrie direct valoarea pointată, nu pointerul. Dacă valorile au dimensiune variabilă (ex: șiruri de caractere), putem scrie în fața lor dimensiunea actuală, pentru a ști câți octeți ocupă în fișier acea valoare.
2. Scriem în fișier structurile ca de obicei dar, pentru fiecare structură, în loc de pointerul ei, memorăm locația (offsetul) la care a fost scrisă în fișier. În acest fel vom avea pentru fiecare structură două adrese: adresa din memorie (pointerul) și locația din fișier (offsetul). La scrierea în fișier vom înlocui toți pointerii cu locațiile din fișier corespunzătoare. În acest fel, fișierul devine efectiv o zonă de memorie în care

adresele de memorie (pointerii) sunt înlocuite cu locații în fișier. Această metodă se folosește de obicei la structuri mai complexe de date (arbori, grafuri, etc).

Exemplul 3: Să se modifice exemplul de mai sus, astfel încât în baza de date fiecare nume de produs să ocupe minimul necesar de memorie. Fiecare nume va putea avea maxim 1000 de caractere:

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

typedef struct{
   char *nume;
   float pret;
   }Produs;

Produs produse[1000];
int nProduse;

void adaugare()
{
   Produs p;
   char nume[1000];

   printf("nume: ");
   fgets(nume,1000,stdin);
   nume[strcspn(nume,"\r\n")]='\0';
   if((p.nume=(char*)malloc((strlen(nume)+1)*sizeof(char)))==NULL){
      printf("memorie insuficienta");
      exit(EXIT_FAILURE);
      }
   strcpy(p.nume,nume);
   printf("pret: ");
   scanf("%g",&p.pret);
   produse[nProduse]=p;
   nProduse++;
}

void afisare()
{
   int i;
   for(i=0;i<nProduse;i++){
      printf("%s: %g\n",produse[i].nume,produse[i].pret);
      }
}

void scriere()
{
   FILE *fis;
   int i;
   unsigned short nNume;              // numarul de caractere dintr-un nume de produs

   if((fis=fopen("produse.dat","wb"))==NULL){
      printf("nu se poate deschide fisierul\n");
      exit(EXIT_FAILURE);
```

```
    }

  for(i=0;i<nProduse;i++){
    nNume=(unsigned short)strlen(produse[i].nume);
    fwrite(&nNume,sizeof(unsigned short),1,fis);          // scriere lungime sir in fisier
    fwrite(produse[i].nume,sizeof(char),nNume,fis);       // scriere sir, fara terminator
    fwrite(&produse[i].pret,sizeof(float),1,fis);
    }

  fclose(fis);
}

void eliberare()
{
  int i;
  for(i=0;i<nProduse;i++)free(produse[i].nume);
}

int main()
{
  int optiune;
  do{
    printf("optiune: ");
    scanf("%d",&optiune);
    getchar();
    switch(optiune){
      case 1:adaugare();break;
      case 2:afisare();break;
      case 3:scriere();eliberare();break;
      default:printf("optiune invalida\n");
      }
    }while(optiune!=3);
  return 0;
}
```

Câmpul *nume* din structura *Produs* pointează către o zonă de memorie alocată dinamic, în care este memorat numele produsului. Funcția *adaugare* s-a modificat pentru a aloca dinamic numele produsului, după ce aceasta s-a citit temporar în vectorul *nume*.

Funcția *scriere* iterează toate produsele și pentru fiecare produs scrie în fișier pe rând toate câmpurile sale. Deoarece numele are dimensiune variabilă, în fața lui s-a scris și lungimea sa, pentru a se ști câți octeți ocupă în fișier. Din moment ce numele poate avea maxim 1000 de caractere, ajung 2 octeți pentru memorarea sa, deci s-a folosit tipul de date *unsigned short*. Deoarece s-a scris la început lungimea șirului, nu mai este nevoie să se scrie și terminatorul de șir, deoarece se știe exact când se va termina acel șir. Câmpul *pret* s-a scris simplu, ca o zonă de memorie de tip *float*.

Fișierul *produse.dat*, pentru produsele (mere:5, paine:3), va avea conținutul de forma:

```
00000000: 04 00 6D 65 72 65 00 00|A0 40 05 00 70 61 69 6E  | ..mere..á@..pain
00000010: 65 00 00 40 40          |                       | e..@@
```

Fișierul începe cu octeții {04, 00} care reprezintă lungimea primului produs (mere), pe doi octeți, în convenție *little-endian*. Urmează 4 octeți care reprezintă numele produsului în cod ASCII {6D, 65, 72, 65}. După aceștia

urmează 4 octeți {00, 00, A0, 40} care reprezintă numărul 5 de tip *float*. Toți acești octeți formează primul produs. După acesta începe imediat al doilea produs, în același format.

Citirea dintr-un fișier binar

Pentru citire din fișiere binare se folosește funcția **fread**, care exact aceleași argumente și valoare returnată ca și funcția *fwrite*:

size_t **fread**(void *elemente, size_t dim_element, size_t nr_elemente, FILE *fisier);

Funcția *fread* are următorii parametri:

- **elemente** - un pointer la zona de memorie în care se vor citi elementele din fișier
- **dim_element** - dimensiunea în octeți a unui element care va fi citit din fișier. Se poate afla cu *sizeof*
- **nr_elemente** - numărul de elemente de citit
- **fisier** - fisierul sursă

fread returnează numărul de elemente citite integral din fișier. Nu se face distincție între cazurile în care nu s-a citit un element din cauză că s-a ajuns la sfârșit de fișier sau din cauză că a existat o eroare de citire. Dacă dorim să diferențiem între cele două cazuri, se poate folosi funcția *ferror(fis)*, care returnează *true* în caz că a apărut o eroare cu fișierul.

Exemplul 4: Să se completeze exemplul anterior, astfel încât la execuția programului prima oară să se încarce baza de date de pe disc. Dacă încă nu există o bază de date, programul va funcționa normal, fără să se emită un mesaj de eroare.

```
// au fost redate doar partile din program care au fost modificate sau adaugate

void citire()
{
   FILE *fis;
   unsigned short nNume;

   if((fis=fopen("produse.dat","rb"))==NULL)return;   // iesire din functie in caz cu nu exista baza de date
   // deoarece in fisier nu exista numarul de elemente
   // se foloseste o bucla infinita, care continua atata timp cat se mai pot citi noi elemente
   for(;;){
      // se incearca citirea unui nou produs, care incepe cu dimensiunea numelui
      // daca fread nu reuseste sa citeasca elementul dimensiune, inseamna ca s-a ajuns la sfarsit de fisier
      // in acest caz se inchide fisierul si se iese din functie
      if(fread(&nNume,sizeof(unsigned short),1,fis)==0){
         fclose(fis);
         return;
         }
      char *pNume=(char*)malloc((nNume+1)*sizeof(char));          // alocare memorie pentru nume si terminator
      if(pNume==NULL){
         printf("memorie insuficienta");
         fclose(fis);
         eliberare();
         exit(EXIT_FAILURE);
         }
      fread(pNume,sizeof(char),nNume,fis);                        // citire caractere nume
      pNume[nNume]='\0';      // deoarece in fisier nu exista terminatorul de sir, acesta se adauga separat
      produse[nProduse].nume=pNume;
      fread(&produse[nProduse].pret,sizeof(float),1,fis);         // citeste pretul produsului
```

```
        nProduse++;
        }
}

int main()
{
    int optiune;

    citire();                       // citirea bazei de date la inceputul programului
    do{
        printf("optiune: ");
        scanf("%d",&optiune);
        getchar();
        switch(optiune){
            case 1:adaugare();break;
            case 2:afisare();break;
            case 3:scriere();eliberare();break;
            default:printf("optiune invalida\n");
            }
        }while(optiune!=3);
    return 0;
}
```

În funcția *citire*, deoarece în fișier nu există stocat numărul de produse, se citește pe rând câte un produs într-o buclă infinită. În momentul când s-a ajuns la sfârșit de fișer, *fread* va returna 0 elemente citite și astfel știm când să ne oprim din citire.

Pe lângă funcțiile descrise anterior pentru fișiere, mai sunt disponibile și alte funcții, printre care:

- long **ftell**(FILE *fis) - returnează poziția curentă (offsetul) din fișier, față de începutul acestuia. La această poziție vor avea loc următoarele operații de citire sau de scriere.
- int **fseek**(FILE *fis,long offset,int reper) - setează poziția curentă din fișier, folosind ca punct de referință reperul specificat. Reperul poate fi: **SEEK_SET** - începutul fișierului, **SEEK_CUR** - poziția curentă, **SEEK_END** - sfârșitul fișierului. De exemplu, *fseek(fis,0,SEEK_END)* va muta poziția curentă la sfârșit de fișier.

Aspecte conexe cu fișierele binare

Următoarele aspecte nu se cer la laborator sau la examen, dar ele sunt importante când se scriu aplicații care folosesc fișiere binare, astfel încât este bine să le știți pentru viitor. Aceste aspecte se aplică și în cazul aplicațiilor care utilizează transmitere de date binare în rețea (networking).

Dimensiunea tipurilor de date întregi

Standardul C nu specifică exact câți octeți are fiecare tip de date întregi. De exemplu, tipul *int* poate fi implementat pe 2, 4 sau 8 octeți. Din acest motiv, dacă fișierul binar trebuie să fie portabil pentru aplicații scrise folosind diverse compilatoare sau platforme, trebuie să ne asigurăm că tipurile de date din fișier au mereu aceeași lungine.

Putem implementa simplu cerința ca un tip întreg să aibă o dimensiune constantă, indiferent de compilator sau platformă, folosind antetul <*stdint.h*> (standard integers). În acest antet se află definite o serie de tipuri întregi, astfel încât fiecare tip are o dimensiune precis specificată. Tipurile întregi definite în *stdint.h* sunt de forma *int8_t, int16_t, int32_t, int64_t* pentru numere cu semn și *uint8_t, uint16_t, uint32_t, uint64_t* pentru numere fără semn.

Numărul din numele tipului arată câți biți are acel tip. Folosind de exemplu tipul *uint16_t*, știm că acesta va ocupa doi octeți, indiferent de compilatorul sau de platforma pe care compilăm programul.

Dispunerea octeților numerelor în memorie (endianness)

Așa cum s-a arătat mai sus, microprocesoarele Intel sau AMD folosesc convenția *little-endian* pentru stocarea numerelor întregi în memorie. Alte microprocesoare folosesc convenția inversă, *big-endian* (denumită și *network order*), în care se începe cu octetul cel mai semnificativ, deci se citește numărul de la stânga la dreapta. Mai sunt și alte convenții, care dispun octeții în memorie în diverse feluri.

Dacă fișierul binar trebuie să fie folosit pe arhitecturi diferite, atunci trebuie specificat exact ce convenție (*endianness*) folosește pentru stocarea numerelor. După ce s-a stabilit convenția, la scrierea/citirea numerelor în fișier se vor folosi funcții de conversie, care vor scrie/citi numerele în convenția adoptată. De exemplu, pentru a se scrie numerele în *big-endian*, se vor folosi operații pe biți, astfel încât să se izoleze fiecare octet din număr, începând cu cel mai semnificativ.

Aplicații propuse

Aplicația 5.2: Scrieți un program care copiază conținutul unui fișier sursă într-un fișier destinație. Numele fișierelor se citesc din linia de comandă. Pentru eficiența copierii, programul va citi câte un bloc de maxim 4096 de octeți, pe care îl va scrie în destinație.
Exemplu: ./copiere src.dat dst.dat -> copiază src.dat în dst.dat

Aplicația 5.3: Se citesc *m* și *n* de la tastatură, iar apoi o matrice *a[m][n]* cu elemente de tip întreg. Să se scrie matricea într-un fișier binar, prima oară scriindu-se *m* și *n*, iar apoi elementele, așa cum sunt dispuse ele în memorie. Să se citească matricea din fișier într-o variabilă *b* și să se afișeze.

Aplicația 5.4: Să se modifice exemplul 4, astfel încât la începutul fișierului să fie scris numărul de produse. Citirea va fi modificată pentru a folosi această informație, în loc de a se testa sfârșitul de fișier.

Aplicația 5.5: Să se scrie un program similar cu *hexdump*, care primește un nume de fișier în linia de comandă, citește câte 16 octeți din el și îi afișează pe câte o linie. Fiecare linie începe cu offsetul ei în fișier, afișat în hexazecimal pe 8 cifre cu 0 în față, după care valorile hexa ale celor 16 octeți pe câte 2 cifre și în final caracterele. Dacă codul unui caracter este în intervalul 32-255 se va afișa ca atare, altfel se va afișa un punct în locul său. Dacă ultima linie este incompletă, se va completa cu octeți de 0.

Aplicația 5.6: Să se scrie un program care primește 2 fișiere în linia de comandă și le compară între ele. Pentru fiecare octet diferit găsit, se afișează offsetul acestuia și valorile sale din fiecare fișier, în hexazecimal, pe câte 2 cifre. Dacă fișierele sunt de dimensiuni diferite, în final se vor afișa toți octeții din fișierul mai mare. Dacă fișierele sunt identice, programul nu va afișa nimic.

Aplicația 5.7: Un program primește în linia de comandă un nume de fișier, un offset dat în hexazecimal și apoi o serie de octeți, specificați tot în hexazecimal. Programul va scrie în fișierul specificat, începând de la offsetul dat toți octeții primiți în linia de comandă. Restul fișierului va rămâne nemodificat.
Exemplu: ./inlocuire 1.dat 4a0f 9e b0 51 -> scrie octeții *{9e, b0, 51}* în fișierul *1.dat*, începând cu poziția *4a0f*

Aplicația 5.8: Să se scrie un program care primește în linia de comandă un nume de fișier și o serie de octeți, dați în hexazecimal. Programul va afișa toate pozițiile din fișier la care se află secvența de octeți specificați.
Exemplu: ./cautare 1.dat 01 aa b2 45 -> caută în fișierul *1.dat* secvența *{01, aa, b2, 45}* și afișează toate pozițiile la care o găsește

Aplicația 5.9: Să se scrie un program care primește o serie de *n* programe în linia de comandă și le concatenează pe primele *n-1* într-un nou fișier având numele specificat pe ultima poziție din linia de comandă.
Exemplu: ./concat 1.dat 2.dat 3.dat rez.dat -> concatenează conținutul fișierelor *{1.dat, 2.dat, 3.dat}* într-un nou fișier, denumit *rez.dat*