TUTORIAT ASC 3

MARIA PREDA

1. Structuri repetitive

Data trecuta am discutat despre salturi (conditionate sau neconditionate), care vor sta la baza "simularii" structurilor repetitive. Distingem doua metode prin care putem crea aceste cicluri:

- **cu eticheta loop:** aceasta metoda este asemanatoare cu for-ul din limbajul C++, cu adaugarea ca este ca si cum am scadea la fiecare pas indexul cu 1, pana cand acesta ajunge la 0. Daca vrem sa "simulam" un break, trebuie sa adaugam un salt conditionat catre o eticheta din afara zonei de cod destinata "simularii" structurii repetitive.
  Exemplu: for(i=10; i≥0; i−−)

```
.data
    factorial: .long 1   ; initializam factorial cu 1
    n: .long 5       ; n, numarul pentru care vrem sa calculam factorialul

.text

.global main

main:
    mov n, %ecx       ; mutam valoarea lui index in registrul %ecx
                      ; (folosit pentru ‘loop‘)
    mov $1, %eax      ; initializam %eax cu 1 pentru a stoca
                      ; rezultatul factorialului

loop_start:
    mul %ecx              ; inmultim %eax (factorialul curent) cu %ecx
                          ; rezultatul ramane in %eax
    loop loop_start       ; decrementeaza %ecx, sare la loop_start daca %ecx > 0

end_loop:
    mov %eax, factorial   ; salvam rezultatul final in variabila ‘factorial‘

    mov $1, %ebx
    mov $0, %eax
    int $0x80
```

Decrementarea se face automat doar pentru registrul %ecx (nu ne putem alege noi registrul care sa fie decrementat), acesta fiind un registru contor, asa cum am precizat in primul tutoriat. De asemenea, este important de precizat ca se iese din structura repetitiva atunci cand, la o decrementare, valoarea din registru ajunge sa fie 0. Putem modfica valoarea din registru in cadrul structurii repetitive, ceea ce inseamna ca numarul de pasi nu va mai fi cel dorit initial, ci in conformitate cu

actualizarea lui %ecx. De asemenea, daca valoarea lui %ecx era 0 atunci cand programul a inceput executarea structurii repetitive, la decrementare, %ecx va ajunge sa fie $0xffffffff$ si va trebui sa execute toti acesti pasi, pana cand, prin decrementare, va ajunge la valoarea 0.

- **fara eticheta loop:** aceasta metoda se aseamana cu while-ul din limbajul C++. Pentru iesirea din acest ciclu, suntem nevoiti sa adaugam un "if", adica un salt conditionat care sa verifice la fiecare pas daca s-a indeplinit conditia pentru iesire din structura repetitiva, facand un salt in afara acesteia.

  Avem urmatoarea secventa de cod, prin care ne propunem sa calculam $n!$:

```
.data
    factorial: .long 1
    index: .long 0
    n: .long 5

.text

.global main

main:

et_repetitiva:
    inc index

    mov index, %eax
    cmp %eax, n
    jl et_cont    ; prin aceste 3 randuri, simulam un if care verifica
                  ; daca index a trecut sau nu de n

    mul factorial ; vom inmulti valoarea de pana acum din factorial,
                  ; cu valoarea din %eax, adica valoarea curenta a lui index

    mov %eax, factorial ; aveam rezultatul inmultirii in %eax

    jmp et_repetitiva ; daca s-a ajuns la acest jmp, inseamna ca index <= n,
                      ; deci continuam, facand exact aceiasi pasi ca inainte

et_cont:

    mov factorial, %eax


et_exit:
    mov $, %eax
    xor %ebx, %ebx
    int $0x80
```

Dupa executarea acestuii cod, putem pune un breakpoint la et_exit si sa verificam cu i r %eax valoarea pe care voiam sa o calculam. In mod evident, pentru aceeasi functionalitate a codului, puteam scrie programul in mai multe feluri. De exemplu, puteam sa nu utilizam deloc variabila factorial, pastrand mereu rezultatul in %eax sau puteam sa nu utilizam variabila index, scazand la fiecare pas 1 din n, pana cand ajungea la valoarea 1. Metoda aleasa este mai putin importanta, cat timp rezultatul la care ajungem este cel corect.

2. Tablouri unidimensionale

Observaţie 2.1. *Data trecuta, in timpul tutoriatului, am vorbit despre "ce este un pointer". Vom aminti acest concept, deoarece, ulterior, vom folosi notiunea de "adresa de memorie", care se va tot repeta in acest semestru.*

Inainte sa incepem discutia, trebuie sa precizez ca, desi vorbim de adrese de memorie, adresele folosite de program nu sunt adresele fizice, din memoria "reala". Sunt niste adrese virtuale. Programul crede ca foloseste adresa reala, din memoria calculatorului vostru, dar, de fapt, foloseste adrese false, pe care sistemul de operare de interpreteaza, cu ajutorul unui tabel, care face legatura intre adresele virtuale si cele reale.

Sa ne imaginam un sir de cutii identice in care putem stoca informatii. Fiecare dintre aceste cutii are un numar in acest sir. Similar, memoria este impartita, iar fiecare locatie are un "numar", pe care il vom numi adresa. Rolul unui pointer este sa retina o astfel de adresa. Pointerul nu va retine informatia din acea "cutie", ci doar adresa ei.

Fiecare variabila este salvata undeva in memorie.

Registrii sunt intr-o zona speciala a memoriei, pe procesor, asa ca nu au adresa de memorie.

Pentru a afla adresa de memorie a unei variabile (ulterior vom vedea ca este aproximativ la fel si pentru tablouri) putem folosi instructiunea lea.

Sintaxa: *lea nume_variabila, registru*
Aceasta va pune adresa de memorie a variabilei aflate pe prima pozitie in registrul specificat

In continuare, avem scrierea urmatoare, la fel ca in laborator:

$$a(b, c, d)$$

Aceasta valoare trebuie interpretata ca fiind: $a + b + c \cdot d$. Vom folosi aceasta scriere pentru a calcula adrese de memorie. Unele dintre aceste 4 pozitii pot sa nu fie ocupate.

Asa cum ati vazut si anterior, daca declaram la rand mai multe variabile, acestea erau reprezentate in memorie una dupa alta. Acest lucru este util atunci cand stim ca vrem sa retinem informatii de acelasi tip, in acelasi scop. Asa cum avem array-urile din C++, declarate sub forma *int* $v[100]$, numite si tablouri unidimensionale, vom avea tablouri si unidimensionale in Assembly.

Acestea sunt zone de memorie alocate pentru un tip de date pe care il vom specifica la declarare. De ce trebuie specificat? Pentru ca, pentru fiecare valoare din tablou, se va rezerva un anumit numar de bytes din memoria totala alocata tabloului. De exemplu, daca avem un tablou de numere de tipul long, atunci pe primii 4 bytes se va considera intotdeauna ca vom avea valoarea primului element (cu index 0), pe urmatorii 4 vom avea al doilea element (cu index 1) etc.

Un tablou unidimensional se declara in felul urmator:

$$nume: \ .tip\_date \ valori$$

Exemplu 2.2. *Avem urmatorul tablou unidimensional de longuri:*

$$v: \ .long \ 10, \ 20, \ 30$$

Ce se intampla de fapt si ce reprezinta v? v reprezinta de fapt o adresa de memorie, de la care incepe tabloul unidimensional, care va avea lungimea: $4 \cdot 3 = 12$ *bytes, adica* 3 *secvente de cate* 4 *bytes, care vor fi unul dupa altul in memorie. Asadar, avand adresa de la care incepe tabloul, care coincide cu adresa de la care incepe primul element al tabloului, pentru afla valoarea primului element, este suficient sa luam valoarea stocata in primii* 4 *bytes incepand cu byte-ul cu adresa stocata in v. Astfel, obtinem valoarea* 10*. Mai departe, pentru a afla valoarea* 20*, a doua din v, trebuie sa ne deplasam* 4 *bytes de la adresa stocata in v spre dreapta si sa luam cei* 4 *bytes care incep de aici.*

De aceea avem indexarea de la 0*, pentru ca:*

- *pentru primul element, avem adresa de inceput in v, deci adaugam* 0 *bytes*
- *pentru al doilea element, ne deplasam cu* $1 \cdot 4$ *bytes de la adresa din v*
- *pentru al treilea element, ne deplasam cu* $2 \cdot 4$ *bytes de la adresa din v*

Intuitie: Sa ne imaginam un exemplu similar, dar, poate, mai clar. Avem un sir de cutii lipte intre ele, aflate de-a lungul unui zid, fiecare de lungime 4 metri, in care se afla carti (numarul cartilor din fiecare cutie difera). Stim locatia exacta a primei cutii. Pentru a ajunge la o anumita cutie, trebuie sa adunam la aceasta locatie dinstanta pana la cutia dorita. De exemplu, pentru a doua cutie, trebuie sa adunam 4 mentri. Pentru a afla cate carti sunt intr-o cutie, in mod evident, trebuie sa verificam intreaga cutie, adica ne trebuie cartile care se afla in toti acei 4 metri, adica lungimea cutiei. Am folosti 4 metri, pentru a fi echivalentul a 4 bytes din cadrul unui tablou unidimensional de longuri.

Observaţie 2.3. *Pentru a afla adresa de memorie a unui element dintr-un tablou unidimensional, vom tine cont de tipul de date.*

Exemplu 2.4. *Fie urmatorul exemplu, similar cu cel din laborator:*

```
    .data
        n: .long 6
        v: .word 10, 20, 40, 60, 100, 5

    .text
    .global main

    main:
        lea v, %edi
        mov $0, %ecx
        mov (%edi, %ecx, 4), %edx

    etexit:
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

Diferenta este ca, in acest exemplu este ca am utilizat tipul de date word, deci fiecare element ocupa doar 2 *bytes. Daca rulam cu gdb si scriem:*

b etexit
i r %edx

Atunci, vom obtine 40*.*

Exemplu 2.5. *Pentru a aduna toate elementele tabloului de mai devreme, putem utiliza urmatorul program:*

```
    .data
        n: .long 6                  # numarul de elemente din vector
        v: .long 10, 20, 40, 60, 100, 5   # elementele vectorului

.text
.global main

main:
        lea v, %edi                  # incarca adresa lui v in %edi
```

```
        mov $0, %ecx                    # initializeaza indexul la 0
        mov $0, %eax                    # initializeaza suma la 0
        mov n, %ebx                     # incarca numarul de elemente in %ebx

loop_start:
        cmp %ecx, %ebx                  # compara indexul cu numarul de elemente
        jbe end_loop                    # daca am ajuns la sfarsit, iesim din ciclu
        mov (%edi, %ecx, 4), %edx       # incarca elementul curent in %dx
        add %edx, %eax                  # adauga valoarea elementului la suma partiala
        inc %ecx                        # trece la urmatorul element
        jmp loop_start                  # repeta ciclul

end_loop:
 mov $0, %edx

exit:
        mov $1, %eax                    # syscall pentru exit
        mov $0, %ebx
        int $0x80
```

Pentru a parcurge elementele unui taboul, trebuie sa folosim structuri repetitive. Nu este important daca alegem sa folosim loop sau nu.

3. Teoria Probabilitatilor

In aceasta sectiune, vom reaminti cateva notiuni de baza legate de teoria probabilitatilor.

- Probabilitatea ca un eveniment sa se intample (atunci cand nu este influentat de mediul exterior): $\frac{numar\ cazuri\ favorabile}{numar\ cazuri\ posibile}$
- Probabilitatea ca doua evenimente sa se intample, fara ca acestea sa fie influentate de ceva din exterior: (*prob sa se intample primul*) $\cdot$ (*prob sa se intample al doilea*). Acest lucru este adevarat din urmatorul motiv: sa ne imaginam ca primul s-a intamplat. Din totalul cazurilor, in cate se putea ajunge pana in acest punct? $\frac{numar\ cazuri\ favorabile}{numar\ cazuri\ posibile}$. Acum trebuie sa se intample si al doilea, dar cazurile posibile pentru ca ambele sa se intample nu mai sunt totalul cazurilor posibile, sunt intersectia dintre cele in care se putea intampla primul si cele totale. Adica trebuie sa tinem cont de cazurile care sunt favorabile pentru ambele evenimente.

4. Introducere in teoria informatiei

Ignorand teorii precum "Principiul incertitudinii" care sta la baza mecanicii cuantice, majoritatea "fenomenelor" din jurul nostru pot fi, intr-o oarecare masura, "prezise" sau aflate. Cu toate acestea, avem nevoie de informatii legate de starea lor. Acesta este un cuvant cheie: "informatii". Totusi, nu intotdeauna avem **toate** informatiile necesare, avem doar date **partiale**. Asadar, folosind ceea ce stim, putem umple golurile cu ajutorul probabilitatilor.

Exemplu 4.1. *Stim ca intr-un set de piese de construit avem cuburi si conuri, fiecare piesa putand fi de una dintre culorile: rosu, verde, albastru. In plus, cuburile pot fi si galbene. Trei copii, A, B si C joaca urmatorul joc: A scoate din cutie o forma pentru B si una pentru C, cei doi fiind legati la ochi. A ii spune lui B ca are un con, iar lui C ca are un cub. Pentru care dintre cei doi copii este mai usor sa ghiceasca exact cum arata piesa? (adica sa completeze si culoare, dupa ce le-a fost spusa forma)*

Pentru B este mai usor, adica exista o probabilitate mai mare de a ghici. De ce? Pentru ca sunt mai putine culori posibile pentru un con decat pentru un cub.

Practic, daca scopul final este sa aflam intreaga informatie despre piesa, informatia oferita lui B este mai exacta decat cea oferita lui C, pentru ca exista mai putine variante pentru a completa informatiile lipsa.

Exemplu 4.2. *Sa luam o alta situatie, pornind de la exemplul de mai sus. De data aceasta, A le spune doar culoarea, nu si forma piesei. Lui B ii spune ca piesa sa este galbena, iar lui C ca piesa sa este albastra.*

In acest caz, B stie cu siguranta ca are un cub galben, iar C inca nu stie ce forma are piesa sa?

De ce? Pentru ca, avand informatia din acest moment, probabilitatea ca B sa aiba cub este 100%, iar probabilitatea sa aiba con este 0%, in timp ce, pentru C, ambele probabilitati sunt de 50%.

Exemplu 4.3. *Sa adaugam inca o forma pentru piese: sfera. O piesa in forma de sfera poate avea aceleasi* 4 *culori pe care le poate avea si un cub. Repeta jocul din exemplul 3.2 si amandoi copii primesc aceleasi culori. De data aceasta, B va avea probabilitatea de 50% sa aiba sfera si 50% sa aiba cub, in timp ce C va avea o probabilitate de* $33,(3)\%$ *pentru fiecare dintre cele trei forme.*

Care este scopul acestor exemple? Scopul este sa accentuam ideea ca, atunci cand avem mai multe posibilitati pentru completarea informatiei, probabilitatea de a ghici este mai mica, deci, in consecinta, "cantitatea" de informatie pe care am obtinut-o prin acea informatie partiala este mai mica.

vom introduce conceptul de variabila aleatoare, pe care il veti intalni din nou anul viitor la materia "Probabilitati si Statistica". O variabila aleatoare ne ajuta sa modelam evenimente sau fenomene care au mai multe rezultate posibile si ne permite sa analizam si sa masuram incertitudinea in mod formal.

Avem o variabila aleatoare X, care poate lua una dintre valorile $(x_1, \cdots, x_N)$, cu probabilitatile $(p_1, p_2, \cdots, p_N)$ (x_1 cu probabilitatea p_1 etc).

Cu cat probabilitatea ca o valoare sa apara este mai mica, cu atat ne ajuta sa completam mai usor informatia lipsa, adica scade incertitudinea. In exemplele de mai sus, cu cat existau mai putine forme de o anumita culoare (i.e. probabilitatea ca acea culoare sa fi aparut era mai mica), cu atat avea sanse mai mari de a ghici forma acesteia, adica gradul de incertitudine era mai mic decat in cazul culorilor care apareau pentru mai multe forme.

Avem urmatoarea formula, care ne ajuta sa aflam "cata informatie ne ofera" o valoare in functie de care era probabilitatea ca aceasta valoare sa apara:

$$I(x_i) = \log_2\left(\frac{1}{p_i}\right)$$

Exemplu 4.4. *De data aceasta, avem* 12 *piese,* 3 *cuburi,* 3 *conuri,* 3 *sfere si* 3 *piramide. Fiecare tip de forma este de una dintre culorile: rosu, galben si albastru. Copilul A extrage din nou o piesa si il informeaza pe copilul B ca este rosie. Cata informatie primim?*

$$log_2\left(\frac{1}{4 \cdot \frac{1}{12}}\right) = log_2\,(4) = 2$$

De ce? Initial aveam 12 variante. Daca am fi stiut toate informatiile despre piesa, am fi avut $log_2 12$*, adica intreaga informatie. Dar noi avem* 4 *piese care se pot incadra in aceasta culoare.*

Daca am si stiut doar ca este o piesa din set, incertitudinea ar fi fost aceeasi (cazul in care raman in continuare toate variantele posibile).

5. Exercitii

(1) Fie urmatoarele declaratii in sectiunea .data:

```
a: .ascii "Assembly"
b: .word 0x25
c: .asciz "x86"
d: .asciz ";;;"
e: .long 0x15
```

Ce se intampla dupa aceasta secventa de cod?

```
mov $4, %eax
mov $1, %ebx
mov $a, %ecx
mov $b, %edi
sub %ecx, %edi
or %edi, e
mov e, %edx
int $0x80
```

Soluţie:

Se va afisa mesajul: "Assembly%x86;;". La apelul de sistem, adica in dreptul randului "int $0x80", se va verifica codul functiei care trebuie apelata, adica numarul aflat in %eax. Fiind 4, inseamna ca se va apela functia WRITE. In registrul %ebx avem valoarea 1, ceea ce inseamna ca se va afisa in consola. In registrul %ecx am pus adresa lui a, ceea ce inseamna ca afisarea incepe de la primul caracter din a.

In continuare, trebuie sa calculam valoarea aflata in registrul %edx la momentul apelului de sistem.

In %ecx avem adresa lui a, iar in %edi adresa lui b. Diferenta dintre adrese este 8 (lungimea in bytes a sirului a).

Aplicam operatia logica or intre %edi si e si obtinem 15, ceea ce inseamna ca trebuie sa afisam 15 bytes: 8 de la a, 2 de la b, 3 de la c si 2 de la d. Valoarea din b va fi convertita in ascii si obtinem caracterul %.

(2) Fie urmatorul cod:

```
.data
    array1: .long 2, 4, 6, 8, 10
    array2: .long 1, 3, 5, 7, 9
    size:   .long 5
    p: .long 0
    temp:  .long 0
    unused: .long 42

.text
.global main
```

```
main:
    movl $array1, %edi
    movl $array2, %esi
    movl size, %ecx
    xorl %eax, %eax
    xorl %ebx, %ebx

loop_start:
    cmpl $0, %ecx
    je loop_end

    movl (%edi), %edx
    movl (%esi), %eax
    imull %edx
    addl %eax, %ebx

    movl $100, %edx
    addl $42, %edx

    addl $4, %edi
    addl $4, %esi
    decl %ecx
    jmp loop_start

loop_end:
    movl %ebx, p

et_exit:
    movl $1, %eax
    xor %ebx, %ebx
    int $0x80
```

Ce valoare va fi stocata in p la final? Explicati in cuvinte ce face codul.

Soluţie:
La final, in variabila p va fi stocata valoarea 190. Codul realizeaza dot product al celor doua tablouri, adica suma produselor elementelor de pe aceeasi pozitie:

$$2 \cdot 1 + 4 \cdot 3 + 6 \cdot 5 + 8 \cdot 7 + 10 \cdot 9 = 190$$

La fiecare pas, se face inmultirea intre elementele curente, ale caror adrese se afla in %eax si %edx. Acest produs se adauga la valoarea din %ebx, in care calculam rezultatul, pe care il mutam dupa for in p. Totodata, la fiecare pas adaugam 4 la ambele adrese ale vectorilor pentru a ne muta o pozitie mai la dreapta in vectori. In final, decrementam %ecx pana cand ajunge la 0, cand iesim, deci facem $n = 5$ pasi.

(3) Avem urmatorul cod:

```
.data
.section .text
.global main
```

```
main:
    movl $0x12345678, %eax
    xorl %ecx, %ecx
    xorl %ebx, %ebx

    movw %ax, %cx
    movb %ah, %bl
    movb %al, %bh
    xorb %bh, %bh

et_while:
    addb %bl, %bh
    incb %bl
    loop et_while

et_exit:
    movl $1, %eax
    xorl %ebx, %ebx
    int $0x80
```

Ce valoare (in hexazecimal) vom avea in %ebx dupa ce scrie urmatoarele comenzi in gdb?

```
b et_exit
r
i r
```

Soluţie:

Dupa executarea comenzii **r**, programul va rula pana la punctul de intrerupere definit de **b et_exit**. In cadrul buclei **et_while**, valoarea registrului %ebx este modificata in functie de adunarea si incrementarea valorilor din %bl si %bh.

Initial, %eax contine valoarea $0x12345678$, astfel:

$$\%ax = 0x5678, \quad \%ah = 0x56, \quad \%al = 0x78$$

%cx primeste valoarea `0x5678`, iar %bl primeste `0x56` si %bh primeste `0x78`.

In bucla `et_while`, %bl se adauga la %bh, iar %bl este incrementat la fiecare iteratie. Bucla continua pâna când %ecx ajunge la 0. Dupa iesirea din bucla, valoarea finala a registrului %ebx va fi `0x34ce`. Acesta este rezultatul final dupa executia completa a buclei.

Rezultatul final in %ebx este `0x34ce`.

(4) Avem o urna cu 5 bile albe, 7 negre si 12 rosii.

(a) Extragem, la intamplare, o bila. Care este probabilitatea ca aceasta sa fie rosie?

Soluţie:

Avem 24 de bile. Dintre acestea, 12 sunt variante "favorabile" $\Rightarrow$ Probabilitatea este: $\frac{12}{24} = \frac{1}{2}$

(b) Extragem, la intamplare, doua bile. Care este probabilitatea ca amandoua sa fie negre?

Soluţie:

$\frac{C_7^2}{C_{24}^2}$: Cazurile favorabile sunt cele in care luam doua bile negre. Aceste doua pot fi oricare

dintre cele 7, fara sa conteze ordinea lor. Cazurile posibile sunt toate combinarile de cate 2 bile, fara a tine cont de culoare sau ordine dintre cele 24 de bile aflate in total in urna.

(c) Care este probabilitatea sa extragem o bila alba, stiind ca am extras deja o bila rosie?

Soluţie:
Nu ne intereseaza faptul ca prima bila a fost rosie, decat pentru a scadea cu 1 numarul de bile rosii, intrucat aceasta nu se mai afla in urna.
Deci, probabilitatea este: $\frac{5}{23}$.

(d) Care este probabilitatea ca, extragand, la intamplare, 3 bile, sa avem 2 rosii si 1 neagra?

Soluţie:
Cazuri favorabile totale = cazuri favorabile rosii $\cdot$ cazuri favorabile neagra $\Rightarrow C_{12}^2 \cdot 7$

Cazuri posibile: C_{24}^3.

Probabilitatea: $\frac{C_{12}^2 \cdot 7}{C_{24}^3}$

(e) Care este probabilitatea ca, extragand doua bile, acestea sa fie de culori diferite?

Soluţie:
Cazurile favorabile sunt combinarile de doua bile care provin din culori diferite. Totalul cazurilor posibile este dat de combinarile de doua bile din cele 24:

$$C_{24}^2.$$

Pentru cazurile favorabile, consideram toate combinarile de culori diferite:

- Alba si Neagra: $C_5^1 \cdot C_7^1 = 5 \cdot 7 = 35$
- Alba si Rosie: $C_5^1 \cdot C_{12}^1 = 5 \cdot 12 = 60$
- Neagra si Rosie: $C_7^1 \cdot C_{12}^1 = 7 \cdot 12 = 84$

Totalul cazurilor favorabile:

$$35 + 60 + 84 = 179.$$

Probabilitatea:

$$P = \frac{179}{C_{24}^2} = \frac{179}{276} = \frac{179}{276}.$$

(f) Care este probabilitatea ca, extragand o bila, aceasta sa fie rosie **sau** neagra?

Soluţie:
Cazurile favorabile includ toate bilele rosii si negre. Totalul cazurilor favorabile:

$$12 + 7 = 19.$$

Totalul cazurilor posibile este numarul total de bile:

$$24.$$

Probabilitatea:
$$P = \frac{19}{24}.$$

(g) Care este probabilitatea ca, extragand 4 bile, exact una sa fie alba?

Soluţie:
Pentru ca exact una dintre bile sa fie alba, consideram urmatoarele:
- Alegem 1 bila alba din cele 5:
$$C_5^1 = 5.$$
- Alegem 3 bile din celelalte 19 (7 negre si 12 rosii):
$$C_{19}^3 = \frac{19 \cdot 18 \cdot 17}{3 \cdot 2 \cdot 1} = 969.$$
Totalul cazurilor favorabile:
$$C_5^1 \cdot C_{19}^3 = 5 \cdot 969 = 4845.$$
Totalul cazurilor posibile este numarul de combinari de 4 bile din 24:
$$C_{24}^4 = \frac{24 \cdot 23 \cdot 22 \cdot 21}{4 \cdot 3 \cdot 2 \cdot 1} = 10626.$$
Probabilitatea:
$$P = \frac{4845}{10626}.$$
Simplificand fractia:
$$P = \frac{645}{1416}.$$

(h) Care este probabilitatea ca, extragand 3 bile, toate sa fie de aceeasi culoare?

Soluţie:
Cazurile favorabile includ toate situatiile in care bilele extrase sunt fie toate albe, fie toate negre, fie toate rosii. Calculam separat pentru fiecare caz:
- Toate albe:
$$C_5^3 = \frac{5 \cdot 4 \cdot 3}{3 \cdot 2 \cdot 1} = 10.$$
- Toate negre:
$$C_7^3 = \frac{7 \cdot 6 \cdot 5}{3 \cdot 2 \cdot 1} = 35.$$
- Toate rosii:
$$C_{12}^3 = \frac{12 \cdot 11 \cdot 10}{3 \cdot 2 \cdot 1} = 220.$$
Totalul cazurilor favorabile:
$$10 + 35 + 220 = 265.$$
Totalul cazurilor posibile este numarul de combinari de 3 bile din 24:
$$C_{24}^3 = \frac{24 \cdot 23 \cdot 22}{3 \cdot 2 \cdot 1} = 2024.$$
Probabilitatea:
$$P = \frac{265}{2024}.$$
Simplificand fractia:
$$P = \frac{53}{404}.$$

(i) Care este probabilitatea ca, extragand 3 bile, cel putin una sa fie rosie?

Soluţie:
Vom folosi complementul: calculam mai intai probabilitatea ca nicio bila sa nu fie rosie (adica toate sa fie fie albe, fie negre).
Totalul bilelor care nu sunt rosii:

$$5 + 7 = 12.$$

Numarul de cazuri in care alegem 3 bile doar din cele albe si negre:

$$C_{12}^3 = \frac{12 \cdot 11 \cdot 10}{3 \cdot 2 \cdot 1} = 220.$$

Totalul cazurilor posibile ramane:

$$C_{24}^3 = 2024.$$

Probabilitatea ca nicio bila sa nu fie rosie:

$$P_{\text{fara rosii}} = \frac{220}{2024}.$$

Probabilitatea ca cel putin una sa fie rosie este complementul:

$$P_{\text{cu rosii}} = 1 - P_{\text{fara rosii}} = 1 - \frac{220}{2024} = \frac{1804}{2024}.$$

Simplificand fractia:

$$P = \frac{451}{506}.$$

Zecimal	Hexazecimal	Caracter
32	20	(space)
33	21	!
34	22	”
35	23	#
36	24	$
37	25	%
38	26	&
39	27	’
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	¡
61	3D	=
62	3E	¿
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S

Tabela 1. Tabel ASCII

Zecimal	Hexazecimal	Caracter
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	‘
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	—
125	7D	}
126	7E	~

Tabela 2. Tabel ASCII