Tehnici de programare

biblioteci; generarea de numere; măsurarea timpului

1. Crearea si gestionarea de biblioteci software in C

Se foloseste pentru a "impacheta" mai multe fisiere obiect intr-un fisier care spre exemplu permite reutilizarea de functionalitate (fspre exemplu iserul biblioteca pentru operatii matematica are antetul declarat in fisierul math.h).

Spre exemplu putem crea o biblioteca de utilitare simple cu numele **hs_utils** in felul urmator:

- cream fisierul hs_utils.c cu urmatorul continut

```
/* hs_utils.c */
int estePrim(unsigned long long n) {
      if (n%2==0){
         return 0;
      }
      //restul codului
}
```

-iar mai apoi cream fisierul hs_utils.h in care declaram functiile fin hs_utils.c

```
/* hs_utils.h */
unsigned estePrim(unsigned long long);
```

Astfel am putea folosi biblioteca creata mai sus intr-un program in felul urmator:

```
/* main.c */
#include "hs_utils.h"
void main(void) {
   if (estePrim(23)){
     //...
   }
}
```

in care solicitam includerea la preprocesare a fisierului **hs_utils.h** care permite compilatorului sa aiba acces la antetul functiei **estePrim** urmand ca implementarea ei sa fie accesibila la linkare din fisierul obiect obtinut prin compilarea lui **hs_utils.c**.

2. Generarea de numere pseudoaleatoare

Se foloseste funcata rand() din stdlib.h care returneaza la fiecare apel un numar natural cuprins intre [0, RAND_MAX).

Initializarea generatorului de numere pseudoaleatoare se poate face folosind functia srand(unsigned) care primeste un *seed* si permite generarea de secvente pseudoaleatoare distincte, intre rulari succesive. Apelul la srand() trebuie facut o singura data, ca parte a rutinei de initializare, inainte de orice apel la rand().

O practica uzuala este utilizarea rezultatului functiei time(0), care returneaza o data de tipul time_t, cu valoare distincta la fiecare apel (timpul curge unidirectional) si ganranteaza ca la fiecare apel se obtine o alta secventa pseudoaleatoare. Astfel se va folosi srand(time(0));

```
// genereaza o secventa de numrere pseudoaleatoare, distincta la fiecare apel
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    // Foloseste ora curenta pentru initializarea PRNG-ului
    srand(time(0));
    for(int i = 0; i<4; i++){
        printf(" %d ", rand());
          }
    return 0;
}
```

3. Masurarea timpului de executie

Utilizarea functiilor din time.h

Pentru o analiza mai amanuntita (spre exemplu a unei singure functii sau chiar a unei portiuni dintr-o functie) se poate folosi functia clock, declarata in time.h.
In mod uzual se apeleaza clock() la inceputul si sfarsitul portiunii de analizat, se scad valorile si converteste in timp-real, prin impartire la CLOCKS_PER_SEC (numarul de "clocks" ai procesorului), astfel:

```
/* preluat din referinta [2] */
#include <time.h>

clock_t start, end;
double cpu_time_used;

start = clock();
… /* Do the work. */
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

Deoarece pentru aplicatii practice de multe ori timpii de executie, pentru seturi reduse de date, procesoare performante si algoritmi efiecienti, sunt in zona pragului de zgomot/eroare se recomanda masurarea unui set mai mare de executii ale aceluiasi algoritm (spre exemplu 100 de apeluri ale functiei de sortare analizate) si impartirea timpului gasit la numarul de apeluri.

Aplicatii

1. Implementati biblioteca descrisa succint sub numele de hs_utils (fisierele hs_utils.c si hs_utils.h) adaugând și alte funcții precum estePar, esteImpar, esteNegativ, estePozitiv. Testati biblioteca hs_utils.

2. Implementati o functie **unsigned* makeRandArray(unsigned seed, unsigned n)**, care primeste un *seed* si un numar natural **n** si returneaza un vector alocat dinamic, continand **n** numere naturale pseodoaleatoare, uniform distribuite.

3. Implementati o functie **int* makeRandLimitArray(int seed, unsigned n, int a, int b)** care primeste un seed, si numerele naturale **n**, **a** si **b** si returneaza un vector alocat dinamic, continand **n** numere intregi pseudoaleatoare, uniform distribuite, cuprinse intre **a** si **b.**

4.
a) Implementati o functie cu prototipul **int findElemLin(int v[], unsigned n, int x)** care returneaza pozitia primei aparitii a elementului **x** in vectorul **v** avand **n** elemente sau **-1** daca acel numar nu apare in vector. Se va folosi un algoritm cu complexitate liniara (spre exemplu https://en.wikipedia.org/wiki/Linear_search#Basic_algorithm).
b) Implementati o functie cu prototipul **int findElemBin(int v[], unsigned n, int x)** cauta elementul, similar cu cerinta de la punctul **a)**, dar opereaza asupra unui vector sortat. Pentru rezolvare se va folosi un algoritm cu complexitate logartimica (spre exemplu algoritmul cautarii binare, descris in psudocod la https://en.wikipedia.org/wiki/Binary_search_algorithm#Procedure).
c) Masurati timpul de executie pentru un numar semnificativ de rulari ale celor doua functii (spre exemplu 100 de rulari) si repetati acest proces pentru un set de date din ce in ce mai mare

(spre exemplu de la 100 de elemente la 50000 de elemente din 100 de rulări).
Tipariti datele pe iesirea standard.
Obs:
Pentru generarea vectorilor cu date se va folosi una dintre functiile create la cerintele 2-3 (pentru cerinta (b) puteti genera un vector pseudoaleator uniform distribuit pe care mai apoi sa il sortati);

4. Resurse

1. Laborator preluat de la cursul Tehnici de programare, an 1 CTI, seria 1, s.l. dr. ing. Alexandru Iovanovici și din cursul Programming Techniques, an 1 CTI EN, s.l. dr. Ing. Alin Anton.
2. https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/
3. https://www.gnu.org/software/libc/manual/html_node/CPU-Time.html