01101001
01101111
01100011
01101100
01100001

Nume și grupă:

Introducerea în Organizarea Calculatoarelor și Limbaje de Asamblare

17 iunie 2024

Timp de lucru: 120 de minute

1. Aflând că arta de a fi Xen (a se citi "Zen") constă în a face calcule folosind operații pe biți, bobocii s-au pus pe treabă. Urmăriți și voi, încercarea lor de relaxare.

a. Dându-se un număr pe 32 de biți, verificați dacă numărul format din prima jumătate a biților săi este egal cu negatul numărului format din cea de-a doua jumătate a biților săi. Pentru testare, folosiți variabila `n1` și variabila `n2`, deja definite în program. În cazul în care condiția este îndeplinită, afișați `I am Xen`, altfel `I am not Xen`. (**5 puncte**)

b. Afișați suma ultimelor cifre ale tuturor numerelor salvate în vectorul `arr1`. (**5 puncte**)

c. Printați toate perechile de elemente aflate pe poziții de forma 3k și 3k + 2 din vectorul `arr2` care au al doilea număr egal cu negatul primului - 1. Cu alte cuvinte, primul element al perechii se află pe o poziție divizibilă cu 3, iar al doilea element al tuplului este un număr aflat la 2 poziții distanță de primul element al perechii. **Atenție!** Valoarea lui k începe de la 0. (**5 puncte**)

2. Știind că la rețelistică reprezentarea favorită pentru numere este cea hexa, îndrumătorii de an vă propun următoarele task-uri de încălzire.

a. Creați funcția `read_array`, cu semnătura `void read_array(int *n, int *v)`, care citește de la tastatură un număr întreg n și apoi, n elemente întregi ale vectorului `v` **în format hexa**, folosind funcția `scanf`. Apelați funcția `read_array(int *n, int *v)` cu n și v declarate de voi în ce secțiune doriți. Puteți considera că numărul maxim de elemente citite este 100. Pentru verificare, printați vectorul citit, **tot în format hexa**. (**5 puncte**)

b. Creați funcția `print_ip` cu semnătura `void print_ip(int n)`, care printează cei 4 bytes ai numărului n în format ip. Formatul ip presupune printarea byte cu byte, începând cu primul din partea dreaptă a reprezentării little endian (cel mai semnificativ octet), fiecare byte fiind despărțit de câte un punct. Spre exemplu: pentru numărul 100, reprezentarea ip este 0.0.0.100 pentru numărul 25600, reprezentarea ip este 0.0.100.0 pentru numărul 6553600, reprezentarea ip este 0.100.0.0 pentru numărul 1677721600, reprezentarea ip este 100.0.0.0. Un exemplu mai detaliat ar fi numărul 356, cu reprezentarea ip 0.0.1.100. 356 are reprezentarea hexa 0x164 care este împărțită în octeții 0x00, 0x00, 0x1 și 0x64. Pentru verificare, folosiți primul element salvat în vectorul citit la punctul anterior. (**5 puncte**)

c. Creați funcția **recursivă** `fibo` cu semnătura `int fibo(int n)` care calculează al n-lea element al șirului Fibonacci **în mod recursiv**. Este garantat că rezultatul încape într-un element de tip `int`. **Atenție!** Numerotarea elementelor din cadrul secvenței Fibonacci începe de la 0. Cu alte cuvinte pentru secvența 0, 1, 1, 2, 3, 5 ..., 0 se află pe poziția 0, 1 pe poziția 1, 1 pe poziția 2, 2 pe poziția 3, 3 pe poziția 4 etc. (**5 puncte**)

3. Mr X încearcă să descopere secretul ... binarelor independente și pline de leak-uri. Vreți și voi să aflați acest secret?

a. Urmăriți conținutul funcției `todo_a`. Apelați funcția `find` astfel încât aceasta să întoarcă rezultatul 0. Implementarea funcției `find` se găsește în fișierul `file.asm`. Pentru verificare rulați `./main a` (**4 puncte**)

b. Urmăriți conținutul funcției `todo_b`. Inițializați corespunzător șirul `buf` astfel încât apelul funcției `check_string` să întoarcă rezultatul 0. Pentru verificare rulați `./main b` (**5 puncte**)

c. Urmăriți conținutul funcției `todo_c`. Apelați functia `catch_me` și identificați o vulnerabilitate în fișierul obiect hidden.o încât să se apeleze funcția `secret`. Pentru verificare rulați `./main c` (**6 puncte**)