Indicații

- Testul conține 3 subiecte, durează 120 de minute și valorează 50 de puncte, cu alte 5 bonus.
- Se pot obține punctaje parțiale pentru un subiect numai dacă acest lucru este specificat.
- Pentru a fi punctată, o rezolvare **trebuie** să includă și metoda de verificare a funcționalității acesteia.
- Pentru subiectul 1, pentru afișarea la ecran este recomandată utilizarea macroului `PRINTF32`.
- Pentru subiectul 2, pentru afișarea la ecran este obligatorie utilizarea funcției de bibliotecă `printf()`.
- Ordinea de rezolvare a subiectelor este la alegerea voastră.
- Subiectele se rezolvă pe mașina virtuală de PCLP2.

Subiecte

Subiectul 1

a) [5p] Afișați toate elementele `n` din lista `numes` care satisfac condiția `n` % 8 = 4.

HINT: Operații pe biți.

b) [5p] Verificați dacă suma unsigned a două numere `a` și `b` se încadrează în 2 octeți (16 biți).

c) [5p] Verificați dacă numărul `value` are mai mulți biți activați pe poziții pare sau pe poziții impare. Dacă sunt mai mulți biți pe poziții pare afișați `More bits even`, altfel afișați `More bits odd`. Atenție: Biții sunt indexați de la zero, deci primul bit este unul par.

d) [5p] Scrieți o secvență de instrucțiuni care să modifice in-place toate aparițiile `ee` în `zz` ale șirului salvat în variabila `string`.

Subiectul 2

Pentru că studenții s-au săturat de lipsa vectorilor dinamici din C la SDA, s-au hotărât să își facă propria implementare în assembly. Vectorii dinamici sunt o structură de date cu trei câmpuri:

- `arr`, un pointer la zona de memorie de pe heap care ține datele
- `capacity`, un întreg care ține numărul de elemente care încap în zona alocată dinamic
- `length` care ține numărul de elemente din vector

Atunci când dimensiunea alocată nu mai este suficientă, vectorul alocă un nou spațiu de memorie de două ori mai mare pentru date. Vectorul va fi folosit numai pentru numere întregi pe 4 octeți. Pentru acest exercițiu NU aveți voie să folosiți `PRINTF32`.

a) Implementați funcția `void vector_print(struct vector *v)` care afișează datele vectorului, lungimea și capacitatea sa. Pentru testare declarați o variabilă `dummy` care folosește șirul de date `data` cu lungimea și capacitatea egale cu 5. Atenție: Doar pentru acest subpunct vom folosi pentru membrul `vector.arr` zone de memorie care nu sunt de pe heap.

b) Implementați funcția `void vector_init(struct vector *v, int cap)` care inițializează vectorul cu o capacitatea `cap`. Declarați o variabilă `vec` pe care o inițializați cu `vector_init` cu o capacitate de 2. Pentru testare generați două numere random cu `rand` pentru primele două poziții din vector. Apelați `vector_print` pe `vec` ulterior.

c) Implementați funcția `void vector_push(struct vector *v, int value)` care va adăuga `value` la vector. În cazul în care nu mai este spațiu alocat, realocați zona de memorie din câmpul `arr` către una cu dimensiune dublă. Pentru testare folosiți valorile din `data` pe care le veți adăuga în vector folosind `vector_push`. Apelați `vector_print` pe `vec` ulterior.

HINT: `realloc`, `cap` == `len`

d) Implementați funcția `void vector_reduce(struct vector *v, int (*f)(int, int) , int *acc)` care parcurge toate elementele vectorului și, pentru fiecare element, actualizează valoarea lui `acc` folosind funcția `f`. Vrem să calculăm suma elementelor vectorului, deci vom folosi `f` ca fiind `sum`. Pentru testare apelați `vector_reduce` pe `vec` și afișați rezultatul.

HINT: `acc` = f(acc, v.arr[i])

Subiectul 3

a) [5p] Inspectați executabilul `basic` și găsiți o metodă pentru a afișa mesajul “Win!”.

b) [5p] Inspectați executabilul `mask` și găsiți o metodă pentru a afișa mesajul “PCLP2{*5_points*}”.

c) [5p] Inspectați executabilul `parrot` și găsiți o metodă pentru a afișa mesajul “You win!”.