Tema 3 - Checkered Flag

- Deadline: 30.05.2022
- Data publicării: 16.05.2022
- Responsabili:
  - Ilinca Struțu
  - Andreea Drehuta
  - Robert Grancsa
  - Valentin-Razvan Bogdan
  - Cristian Vijelie
- Actualizări:
  - 16.05.2023 - publicare tema
  - 19.05.2023 - modificare restrictii task 1
  - 21.05.2023 - clarificare restrictii task 1
  - 22.05.2023 - clarificare restrictii task 1

Task 1 - Reversing vowels (20p)

Scopul acestui task este sa folositi doar stiva, prin operatii push si pop, pentru a lucra cu memoria.

Dupa sucesul de la Baku, Ferrari si-au dat seama ca mesajele criptate au functionat, iar Mercedes nu a avut nici o sansa impotriva lor. Dar din pacate Red Bull si-au dat seama foarte rapid ce vor sa faca, asa ca dupa ce au trecut calificarile, nu au mai avut nici o sansa impotriva echipei austriace.

Dar cei de la Ferrari si-au dat seama ca nu pot sa ii intreaca la inteligenta pe cei de la Red Bull, asa ca s-au gandit la un mod cat mai simplu de a cripta un mesaj. Ei s-au gandit sa ia fiecare vocala din mesajul lor, si sa le afiseze doar pe ele in ordine inversa. Asadar, atunci cand o sa vrea ei vreodata sa vorbeasc despre Red Bull, tot ce va trebui sa faca e sa afiseze mesajul:

```
rud bell    // "red bull" cu vocalele scrise in ordine inversa
```

Asa, Red Bull nu va stii niciodata ca vorbesc despre ei. Din pacate aceasta metoda nu este la fel de eficienta daca vrem sa vorbim despre Mercedes, dar avem celalalt mod de encriptare, asa ca ar trebui sa le iasa.

Pentru a implementa functia noastra, trebuie sa pornim de la string-ul primit ca parametru, si sa facem toate modificarile in place. **Nu** aveti voie sa va definiti un alt vector pentru a pastra o copie a string-ului sau pentru a stoca vocalele (dar in schimb aveti voie sa va definiti un vector pentru a stoca "aeiou").

```
void reverse_vowels(char *string);
```

Exemplu

Aceste string-ul vor contine doar litere mici ale alfabetului englez si whitespace

```
"hello" -> "holle"
"max verstappen" -> "mex varsteppan"
```

Singurele instructiuni de transfer de date permise sunt push si pop. NU este permisa utilizarea instructionilor din familia mov (mov, cmov, stos, lods, etc), leave si enter, xchg etc. Este permisa utilizarea instructiounilor aritmetice si logice si a instructiunilor de control (mai multe detalii in laboratorul 5).

Nu este este permisa apelarea altor functii externe in afara de strchr.

Task-ul 2 - PWD (20p)

Scopul acestui task este sa apelati diverse functii pentru anumiti parametrii ai functiei cerute.

Am vazut ce poate face Aston Martin cu noua lor masina, scotand numai locuri pe podium in ultimele curse. Dar isi doresc victoria campionatului, asa ca inginerii AM decid sa sparga baza de date Mercedes pentru a gasi diverse upgrade-uri care i-ar putea ajuta in functie de circuit.

Cand au reusit in sfarsit sa sparga aceasta baza de date, au descoperit ca, desi Mercedes este o echipa avansata din punctul de vedere al ingineriei, nu au instalat pe aceasta comanda *pwd*.

Pentru acest task trebuie implementata urmatoarea functie, care reprezinta folderul curent care se obtine dupa aplicarea comenzii *cd* din Linux asupra unor directoare, in ordinea in care acestea apar. Se vor implementa si "." (folder curent) si ".." (folder anterior).

```
void pwd(char **directories, int n, char *output)
```

Parametrii acestei functii sunt urmatorii:

- **directories** - vectorul de siruri de caractere ce reprezinta folderele
- **n** - numarul de foldere din vector
- **output** - sirul de caractere in care va trebui sa stocati calea finala

Comportamentul functiei *cd* cu argumentul ".." este asemanator cu cel din Linux, deci aceasta va produce un efect doar daca exista un folder parinte pentru cel curent.

Exemplu

```
n = 5

directories =

home
folder1
.
folder2
..
folder3
```

```
output = /home/folder1/folder3
```

Comportamentul este urmatorul:

- Se va adauga '/home', apoi se va adauga '/folder1', deci rezultatul este: '/home/folder1'
- Se va gasi '.' care se refera la folderul curent, deci rezultatul ramane: '/home/folder1'
- Dupa se adauga 'folder2', deci rezultatul devine: '/home/folder1/folder2'
- Se va gasi '..', care se refera la folderul anterior, deci va iesi din cel curent, rezultand: '/home/folder1'
- Apoi este adugat 'folder3', si rezultatul final este: '/home/folder1/folder3'

Task-ul 3 - Sortare de cuvinte (25p)

Mecanicii de la Haas s-au plictisit in timpul steagurilor rosii, asa ca s-au decis sa joace un joc de scrabble. Pentru ca li s-a parut fun, au decis ca pot folosi doar cuvinte din propozitiile spuse de cei inginerii radio.

Pentru acest task veți avea de separat un text în cuvinte dupa niște delimitatori și, după aceea, să sortați aceste cuvinte folosind funcția qsort. Sortarea se va face întâi după lungimea cuvintelor și în cazul egalității se va sorta lexicografic.

Va trebui să implementați 2 funcții cu semnăturile `void get_words(char *s, char **words, int number_of_words);` și `void sort(char **words, int number_of_words, int size);` din fișierul task3.asm.

Scopul task-ului acesta este să folosiți funcții din biblioteca standard C. Dacă există ceva în libc ce va poate ajuta, folosiți cu incredere!

Antetul primei funcții este:

```
void get_words(char *s, char **words, int number_of_words);
```

Semnificația argumentelor este:

- **s** textul din care extragem cuvintele
- **words** vectorul de string-uri în care se salvează cuvintele găsite
- **number_of_words** numărul de cuvinte

Atenție, funcția **nu** returnează nimic, cuvintele se salveaza in vectorul words!

Antetul celei de-a doua funcții este:

```
void sort(char **words, int number_of_words, int size);
```

Semnificația argumentelor este:

- **words** vectorul de cuvinte ce trebuie sortat
- **number_of_words** numărul de cuvinte
- **size** dimensiunea unui cuvânt

Atenție, funcția **nu** returnează nimic, sortarea se face in-place!

Precizări

- lungimea textului este mai mica decat 1000;
- vectorul de cuvinte va avea maxim 100 de cuvine a 100 de caractere fiecare;
- delimitatorii pe care trebuie sa ii luati in calcul sunt: spatiu( ), virgula(,), punct(.), endline (\n)
- nu aveti voie sa folositi alta metoda de sortare in afara de **qsort**. In cazul in care veti folosi alta metoda punctajul pe acest task se va pierde;

Exemplu

```
number_of_words: 9
s: "Ana are 27 de mere, si 32 de pere."
dupa apelul get_words: words = ["Ana", "are", "27", "de", "mere", "si", "32", "de", "pe
dupa apelul sort: words = ["27", "32", "de", "de", "si", "Ana", "are", "mere", "pere"]
```

Pentru mai multe informatii despre qsort puteti accesa linkul: qsort

---

Task-ul 4 - Binary Tree (25p)

Scopul acestui task este de a va obisnui cu apeluri de functii si recursivitate in limbajul de asamblare.

Strategiile din timpul cursei sunt foarte importante, asa ca inginerii de la McLaren au inceput sa dezvolte un sistem de predictie pentru cea mai buna strategie pentru pitstops si pneuri. Au nevoie de voi insa sa ii ajutati sa implementeze anumite functionalitati peste o structura de date cunoscuta voua deja, si anume arbore binar.

In cadrul acestui task veti lucra cu un arbore binar, ale carui noduri au urmatoarea structura:

```
struct node {
    int value;
    struct node *left;
    struct node *right;
} __attribute__((packed));
```

Exercitiul I

Primul exercitiu consta in parcurgerea unui arbore binar de cautare in *inordine* si de a stoca valorile din noduri.

```
inorder_parc(struct node *node, int *array);
```

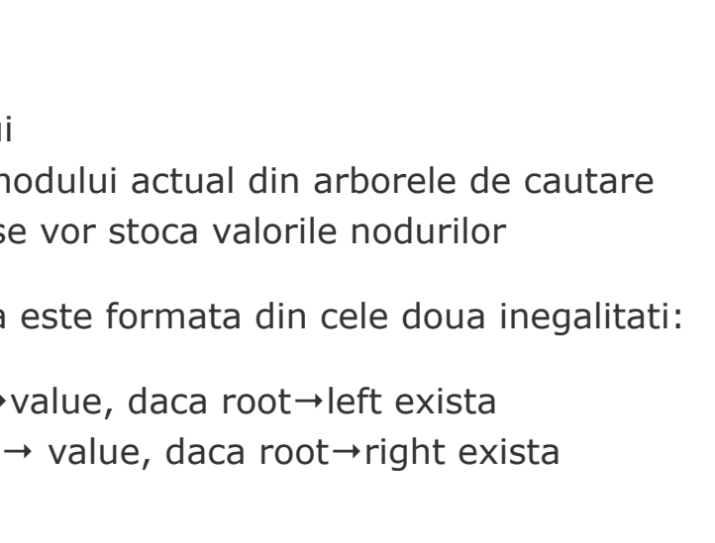
Functia primeste ca argumente:

- **node** - radacina arborelui
- **array** - vectorul in care se vor stoca valorile nodurilor

Pentru a retine pozitia din vectorul array unde veti stoca urmatorul element folositi variabila importata array_idx_1. Vectorul este indexat de la 0!

- Vectorul **array** este alocat in prealabil
- Variabila **array_idx_1** este setata pe 0 la fiecare test in prealabil

Exemplu



In acest exemplu vectorul array va contine elementele **[5, 10, 15, 19, 20, 25, 30, 35]**.

Exercitiul II

Al doilea exercitiu consta in parcurgerea unui arbore binar de cautare in *inordine* si de a stoca valorile din nodurile care nu respecta proprietatea de arbore binar de cautare.

```
inorder_intruders(struct node *node, struct node *parent, int *array);
```

Functia primeste ca argumente:

- **node** - radacina arborelui
- **parent** - tatal/parintele nodului actual din arborele de cautare
- **array** - vectorul in care se vor stoca valorile nodurilor

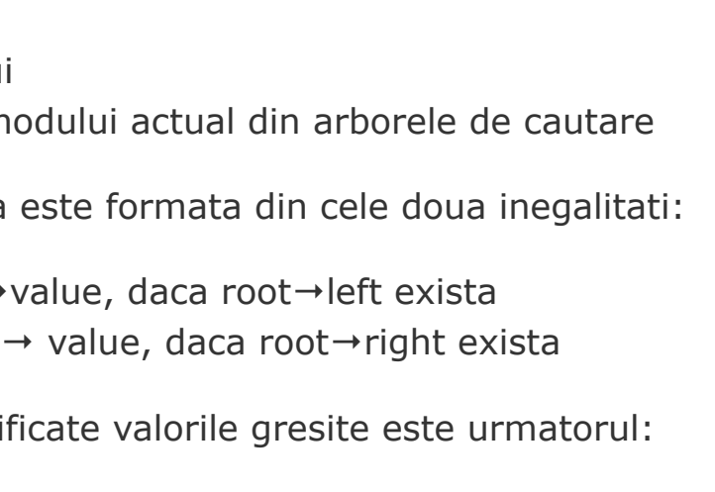
Proprietatea care ne intereseaza este formata din cele doua inegalitati:

1. root→value > root→left→value, daca root→left exista
2. root→value < root→right→ value, daca root→right exista

- Doar in **frunze** pot aparea **valori gresite**
- Pentru a retine pozitia din vectorul array unde veti stoca urmatorul element folositi variabila importata **array_idx_2**. Vectorul este indexat de la **0**

- Vectorul **array** este alocat in prealabil
- Variabila **array_idx_2** este setata pe 0 la fiecare test in prealabil
- Arborele **nu** trebuie modificat

Exemplu



In acest exemplu vectorul array va contine elementele **[12, 40]**.

Exercitiul III

Al treilea exercitiu consta in parcurgerea unui arbore binar de cautare in *inordine* si de a modifica valorile din nodurile care nu respecta proprietatea de arbore binar de cautare.

```
inorder_fixing(struct node *node, struct node *parent);
```

Functia primeste ca argumente:

- **node** - radacina arborelui
- **parent** - tatal/parintele nodului actual din arborele de cautare

Proprietatea care ne intereseaza este formata din cele doua inegalitati:

1. root→value > root→left→value, daca root→left exista
2. root→value < root→right→ value, daca root→right exista

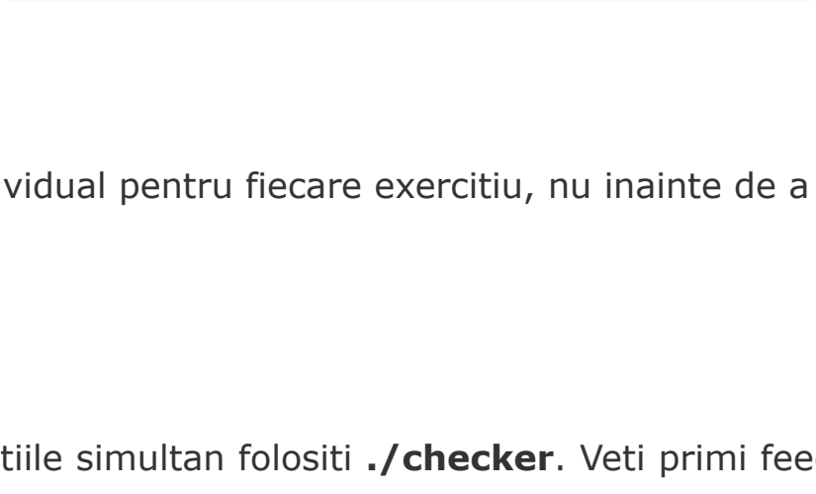
Algoritmul dupa care vor fi modificate valorile gresite este urmatorul:

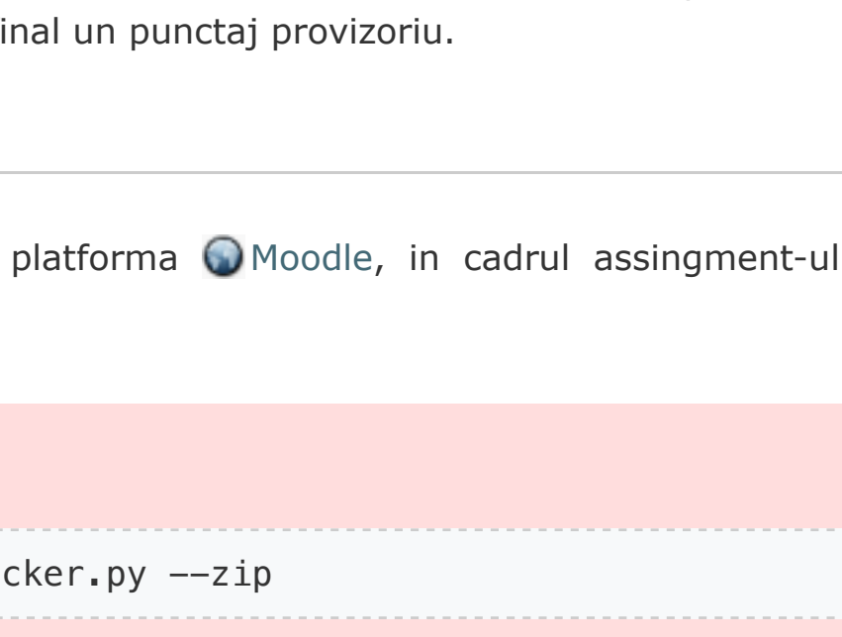
1. Daca nodul actual este fiul stang, acesta va primi valoarea tatalui -1, altfel spus: root→value = parent→value - 1
2. Daca nodul actual este fiul drept, acesta va primi valoarea tatalui +1, altfel spus: root→value = parent→value + 1

Doar in **frunze** pot aparea **valori gresite**

Valorile gresite **nu** trebuie salvate, doar modificate in arbore

Exemplu





Checker-ul task-ului

Checkerul poate fi folosit individual pentru fiecare exercitiu, nu inainte de a compila sursele folosind **make**:

- ./checker 1
- ./checker 2
- ./checker 3

Pentru a verifica toate exercitiile simultan folositi **./checker**. Veti primi feedback (**failed** / **passed**) pentru fiecare test in parte, iar la final un punctaj provizoriu.

Trimitere și notare

Temele vor trebui încărcate pe platforma Moodle, in cadrul assingment-ului Tema 3 și vor fi testate automat.

Folositi comanda:

```
python3 local_checker.py --zip
```

pentru a crea arhiva.

Punctajul final acordat pe o temă este compus din:

- punctajul obținut prin testarea automată - 90p
- coding style si comentarii- 10p
- bonus - 55p

Coding style-ul constă în:

- prezența comentariilor în cod
- scrierea unui cod lizibil
- indentarea consecventă
- utilizarea unor nume sugestive pentru label-uri
- scrierea unor linii de cod (sau README) de maxim 80-100 de caractere

Pentru detalii despre coding style parcurgeti acest document: http://www.sourceformat.com/pdf/asm-coding-standard-brown.pdf

Temele care nu trec de procesul de asamblare (build) nu vor fi luate în considerare.

Arhivele care nu corespund structurii cerute vor fi depunctate cu 20 de puncte din nota finala.

Vă reamintim să parcurgeți atât secțiunea de depunctări cât și regulamentul de realizare a temelor.

Precizări suplimentare

Toate task-urile vor fi realizate in limbajul de asamblare x86 (32 de biti), in afara de cazurile cand se specifica explicit alt limbaj.

FAQ

- **Q**: Este permisă utilizarea variabilelor globale?
  - **A**: Da

Resurse

Scheletul şi checker-ul sunt disponibile pe repository-ul de IOCLA de pe GitLab.

Daca doriti sa folositi infrastructura de testare din cadrul GitLab, este nevoie sa va faceti un **fork privat** al repo-ului de tema.

Search

- Anunțuri
- Bune practici
- Calendar
- Catalog
- Feed RSS
- IOCLA Need to Know
- Reguli și notare
- Resurse utile

Cursuri

- Capitol 00: Prezentare
- Capitol 01: Programe și sistemul de calcul
- Capitol 02: Construirea programelor
- Capitol 03: Arhitectura sistemului de calcul
- Capitol 04: Reprezentarea numerelor
- Capitol 05: Interfața hardware - software x86
- Capitol 06: Stiva
- Capitol 07: Funcții
- Capitol 08: Interfața binară a funcțiilor
- Capitol 09: Gestiunea bufferelor
- Capitol 10: Curs ales de titular
- Capitol 11: Optimizări

Laboratoare

Accesibile pe GitHub aici, cu textul laboratoarelor în format Markdown.

Laboratoare format vechi

- Laborator 01: Reprezentarea numerelor, operații pe biți și lucru cu memoria
- Laborator 02: Operații cu memoria. Introducere în GDB
- Laborator 03: Compilare
- Laborator 04: Toolchain
- Laborator 05: Introducere în limbajul de asamblare
- Laborator 06: Rolul registrelor, adresare directă și bazată
- Laborator 07: Date Structurate. Structuri, vectori. Operatii pe siruri
- Laborator 08: Lucrul cu stiva
- Laborator 09: Apeluri de funcții
- Laborator 10: Interactiunea C-assembly
- Laborator 11: Gestiunea bufferelor. Buffer overflow
- Laborator 12: CTF

- Laborator facultativ: ARM assembly

Teme

- Tema 1 - To pit or not to pit... this is the strategy
- Tema 2 - The race is on
- Tema 3 - Checkered Flag

Table of Contents