

# Tema 1 - Momente disperate

---

- Deadline: 20.04.2022
- Data publicării: 01.04.2022
- Ultima actualizare a enuntului:
  - 09.04.2022 - Adaugat explicatii in plus pentru intelegerea temei.
- Responsabili:
  - Tudor Hodoboc-Velescu [mailto:tudorhoboc@yahoo.com]
  - Catalin-Gabriel Popa [mailto:catalingabrielpopa847@gmail.com]

## Enunt

---

Esti Bodocan Magnificul. Scoti cea mai mentolata manea, dar nu prinde si esti nevoit sa canti la nunti si botezuri. Oamenii fac coada sa arunce cu bani la dedicatii, dar unii pleaca pentru ca trebuie sa astepte la coada. Il angajezi pe Florin Copilul de Biti sa te ajute. Doresti sa uniformizezi tot procesul astfel incat fiecare dedicatie sa fie pe tipuri, depinzand de banii din dedicatie, sa aiba numele celui care face dedicatia si numele caruia i se face dedicatia. De asemenea doresti ca suma sa fie salvata in cate 2 bancnote de marimi diferite. Deoarece Florin mai scapa cate o bancnota din cand in cand in buzunarul lui, tu vrei sa iti faci un program simplu astfel incat sa il poti da afara. Acesta va retine diversele tipuri de dedicatii si va face diverse tipuri de operatii cu ele(afisare, stergere, etc).

## Implementare

---

Implementarea va consta in stocarea datelor intr-un "vector generic" `void *arr`. Fiecare element va fi definit de un header ce va descrie continutul, urmat de datele efective. Spre exemplu, la adresa `arr` se va afla structura header specifica, iar la adresa `arr + sizeof(struct header)` se vor afla datele efective ale primului element.

Pasarea elementelor pentru adaugarea in vector se va face prin intermediul structurii `data_structure`. Aceasta contine adresa catre o variabila struct head in care se stocheaza header-ul elementului adaugat si o adresa catre o zona generica de date. In vector va trebui sa fie copiate atat header-ul, cat si datele efective, nu dorim sa stocam pointerii ci bytii de la adresa aceea.

Mai jos aveti implementarea celor doua structuri.

```
typedef struct head {
    unsigned char type; //tipul de date stocate
    unsigned int len; //lungimea totala
} head; //structura header

typedef struct data_structure {
    head *header; //header-ul datelor
    void *data; //datele efective
} data_structure; //structura de pasare a datelor
```

In vectorul final trebuie copiate datele de la adresa `data`, nu adresa efectiva. Alta forma de implementare va aduce cu ea anulara tuturor punctelor.

Recomandam citirea punctului 7. *memory management* inainte de a va apuca de scris cod.

Programul va citi comenzi de la tastatura pana la primirea comenzii exit, in urma careia programul va elibera memoria si va iesi.

Comenzile primite vor veni in urmatorul format:

- `insert tip dedicator suma1 suma2 dedicatul` - se va apela functia `add_last`
- `insert_at index tip dedicator suma1 suma2 dedicatul` - se va apela functia `add_at`

- delete\_at index - se va apela functia delete\_at
- find index - se va apela functia find
- print - se va afisa tot vectorul
- exit - se va elibera memoria si se va iesi din program

Detaliile despre afisarea si pasarea argumentelor de la input va fi prezentata in cele ce urmeaza

[Intrati aici pentru cod gratis si avantajos](#)

## 1. add\_last - 20p

```
int add_last(void **arr, int *len, data_structure *data);
```

Functia adauga un element la finalul vectorului arr. Primeste adresa vectorului generic (arr), adresa lungimii vectorului reprezentata in numarul de elemente din vector (len), si pointer catre structura de pasare a elementului (data). Lungimea va fi incrementata la finalul operatiilor. Daca operatiile s-au terminat cu succes, atunci functia va intoarce 0, altfel va intoarce o valoare diferita de 0.

## 2. add\_at - 20p

```
int add_at(void **arr, int *len, data_structure *data, int index);
```

Similar functiei add\_last, add\_at primeste adresa vectorului, adresa lungimii si adresa structurii de date, dar si index-ul la care se doreste sa se stocheze elementul. Daca index-ul este mai mic decat 0, nu se va adauga nimic si se va intoarce o eroare. Daca este mai mare decat lungimea vectorului, se va adauga la final.

## 3. find - 15p

```
void find(void *data_block, int len, int index);
```

Functia primeste vectorul de date, lungimea acestuia si indexul la care se doreste afisarea. Daca indexul este mai mare decat lungimea sau mai mic decat 0, nu se afiseaza nimic. Altfel, se va afisa elementul de la indexul respectiv dupa formatul specificat.

## 4. delete\_at - 15p

```
int delete_at(void **arr, int *len, int index);
```

Functia primeste inceputul vectorului, numarul de elemente curente in vector, si indexul elementului pe care dorim sa il eliminam. Vom parcurge vectorul pana la acel index si eliminam elementul din vector, avand grija sa mutam bitii ramasi dupa element, in locul elementului scos.

## 5. print

```
print(void *arr, int len)
```

Functia de print primeste un pointer catre vectorul de date si numarul de elemente din vector. Parcurge vectorul si printeaza datele din interiorul vectorului corespunzator cu tipul de date din interiorul sau.

Forma de printare este urmatoarea:

```
Tipul <type header>
<primul nume> pentru <al doilea nume>
<valoare prima bancnota>
<valoare a doua bancnota>
(spatiu gol dupa fiecare elem printat)
```

Pentru afisarea diverselor tipuri de int, va fi nevoie de folosirea unor macro-uri speciale. Avem urmatoarele tipuri de afisari:

```
printf("%"PRIu8"\n", val) //unde val este un int8_t
printf("%"PRIu16"\n", val) //unde val este un int16_t
printf("%"PRIu32"\n", val) //unde val este un int32_t
```

## 6. exit

La primirea acestei comenzi memoria este dezalocata si programul se opreste.

## 7. memory management - 20p

O alta parte foarte importanta a temei este intelegerea si lucrul cu memoria. Pentru asta, acest task va consta din doua parti:

1) Alocarea corecta de memorie: memoria va fi alocata dinamic pentru toate string-urile si structurile folosite. Pentru un string cu continutul "Ana", se vor aloci dinamic 4 octeti (3 pentru fiecare litera + unul pentru terminatorul de sir). Declaratiile statice de tipul `char a[50]` trebuie modificate in `char *a = malloc(sizeof_string);`.

2) Dezalocarea corecta a memoriei: memoria va fi dezalocata corect si complet. Pentru testare, vom folosi valgrind si ca punct de referinta, programul nu trebuie sa afiseze niciun read invalid sau orice leak de memorie. (erori de REDIR nu o sa fie depunctate).

`valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes ./main`. Ca sa nu va chinati aveti deja la dispozitie o astfel de rulare prin comanda `make check`.

Punctajul pe aceasta cerinta va fi oferit doar daca cel putin 50p au fost obtinute din alte cerinte.

Punctajul pe aceasta cerinta se acorda doar daca aveti memoria dezalocata corect si complet la finalul programului

## Stocarea datelor

Pentru a diversifica datele retinute, vom folosi tipurile de date din headerul `<inttypes.h>`. Datele vor fi stocate sub urmatorul format in functie de tipul header-ului:

```
- tip 1:
  'char*' - numele celui care dedica (marime variabila);
  'int8_t' prima bancnota;
  'int8_t' a doua bancnota;
  'char*' - numele celui caruia i se dedica (marime variabila);

- tip 2:
  'char*' - numele celui care dedica (marime variabila);
  'int16_t' prima bancnota;
  'int32_t' a doua bancnota;
  'char*' - numele celui caruia i se dedica (marime variabila);

- tip 3:
  'char*' - numele celui care dedica (marime variabila);
  'int32_t' prima bancnota;
  'int32_t' a doua bancnota;
  'char*' - numele celui caruia i se dedica (marime variabila);
```

Formatul nu poate fi schimbat, trebuie mai intai salvat un `char*` dupa cele 2 tipuri de int si apoi alt `char*`. Orice alta implementare nu o sa fie punctata. Prin alta implementare se intelege si stocarea intr-un format de tip 3 in loc de folosirea tipului specificat.

Se salveaza memoria de la adresa `char *` nu adresa pointer-ului, de exemplu dorim ca marimea de bytes salvata sa fie variabila, daca salvati pointer-ul `sizeof (char *)` o sa salvati constant aceeasi marime.

## Exemplu rulare

---

Input:

```
insert 3 Andrei 100283912 12312312 Teo
insert_at 0 1 Andrei2 100 100 Teo2
print
find 1
delete_at 0
print
exit
```

Output:

```
Tipul 1
Andrei2 pentru Teo2
100
100

Tipul 3
Andrei pentru Teo
100283912
12312312

Tipul 3
Andrei pentru Teo
100283912
12312312

Tipul 3
Andrei pentru Teo
100283912
12312312
```

Daca ai citit pana aici, meriti un premiu. Spor!

## Trimitere și notare

---

Temele vor trebui încărcate pe platforma vmchecker [<https://vmchecker.cs.pub.ro/ui/#IOCLA>] (în secțiunea IOCLA) și vor fi testate automat. Arhiva încărcată trebuie să fie o arhivă **.zip** care să conțină:

- fișierul sursă ce conține implementarea temei, denumit **main.c**
- fișier **README** ce conține descrierea implementării

Punctajul final acordat pe o temă este compus din:

- punctajul obținut prin testarea automată de pe vmchecker - 90%
- fișier README - 10%

## Precizări suplimentare

---

- Checkerul se va rula folosind comanda **make checker** după ce ați dat drepturi de execuție fișierului.
- Fișierul sursa se va numi main.c.
- Arhiva va trebui încărcată pe vmchecker. Pe lângă codul sursa, va trebui să fie adăugat și un fișier README. Acesta va reprezenta 10p din nota finală.

- Dimensiunea maxima a unei linii citite este de 256 de caractere
- Recomandam parcurgerea regulamentului [[https://ocw.cs.pub.ro/courses/iocla/reguli-notare#realizarea\\_temelor](https://ocw.cs.pub.ro/courses/iocla/reguli-notare#realizarea_temelor)] daca nu ati facut-o deja
- Puteti folosi orice coding style, cat timp sunteti consecventi. Vom depuncta situatii in care coding style-ul ingreuneaza citirea codului.

## Resurse

---

- Scheletul de cod, Makefile-ul se gasesc pe repository-ul public IOCLA [<https://github.com/systems-cs-pub-ro/iocla>] in folderul `teme/tema-1/`. Scriptul de testare va fi adaugat in cel mai scurt timp.

## Explicatii suplimentare

---

Tema in sine este doar joc cu memoria. Dar asta provoaca multe probleme asa ca o sa facem un exemplu pentru explicarea cerintei. Sa zicem ca dorim sa creem o structura de tipul 1 si sa o adaugam in arr.

```
insert 1 Ana 0 1 Banana
```

Primul pas dupa citire este alocarea de memorie pentru structura `data_structure`. Se creeaza un header in care salvam tipul si lungimea care va fi ocupata de datele salvate in interiorul zonei de memorie `data`. Alocam memorie pentru `data` (exact cat avem nevoie) si incepem sa scriem in acea zona de memorie. Nu dorim sa salvam adresa de memorie a niciunei valori, dorim sa scriem exact memoria de la acea adresa. In exemplul de fata nu dorim sa salvam in void `*data` 2 adrese ci valoarea efectiva de la acele adrese (`'Ana\0'`, `'Banana\0'`). Aceeasi gandire o aplicam si pentru cele 2 int-uri, dorim sa salvam exact acei bytes, in cazul de fata avem 2 int-uri pe 1 byte. Astfel lungimea ocupata de toate aceste date este de 13 bytes ( $4 + 1 + 1 + 7$ ).

Dupa crearea acestei structuri dorim sa o salvam in arr (o zona de memorie), alocam/realocam (alocare in cazul in care arr e NULL, sau realocare cand acesta deja contine date) memorie exact cat pentru datele pe care o sa le salvam adica memorie pentru o structura header (aveti in vedere si padding-ul), cat si memorie pentru datele de la adresa de memorie la care pointeaza `data_structure` → `data`.

iocla/teme/tema-1.txt · Last modified: 2022/04/14 10:55 by teodor\_stefan.dutu