Tema 1 - Momente disperate

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 aceia.
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
```

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

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 mix 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("%"PRId8"\n", val)   //unde val este un int8_t

printf("%"PRId16"\n", val)  //unde val este un int16_t

printf("%"PRId32"\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 troate string-urile si structurile folosite. Pentru un string cu continutul “Ana”, se vor aloca 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(size_of_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 chinuiti aveti deja la dispozitie o astfel de rulare prin comanda `make check`.

Stocarea datelor

Pentru a diversifica datele retinute, vom folosit 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);
```

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
```

Trimitere și notare

Temele vor trebui încărcate pe platforma vmchecker (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.
- Fisierul sursa se va numi main.c.
- Arhiva va trebui incarcata pe vmchecker. Pe langa codul sursa, va trebui sa fie adaugat si un fisier README. Acesta va reprezenta 10p din nota finala.
- Dimensiunea maxima a unei linii citite este de 256 de caractere
- Recomandam parcurgerea regulamentului 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.
