Proiect 1 - Cerc C

Obiectiv principal:

Implementarea functiilor din proiectul **data_struct** pentru urmatoarele tipuri de date:

- **Hash table** – din fisierele cchashtable.c/h
- **Heap** – din fisierele ccheap.c/h
- **Stiva** – din fisierele ccstack.c/h
- **Arbore binar** – din fisierele cctree.c/h
  - **Arbore binar de cautare echilibrat**
- **Vector** – din fisierele ccvector.c/h

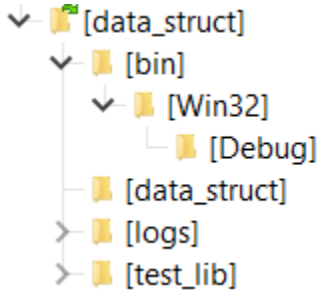
Observatii:
Nu se vor putea folosi functii externe.

Exceptii:

- functii pentru lucrul cu fisiere (fopen, fscanf, fprintf, ...)
- malloc, realloc, free,
- printf
- Win32 APIs.

Proiectul de testare nu va fi evaluat, doar functionalitatea bibliotecii **data_struct** va fi testata folosind o aplicatie dezvoltata de catre noi. In acest scop, proiectul predat trebuie sa contina:

- Tot proiectul, inclusiv binarele compilate (configuratia **Win32 Debug**) intr-o arhiva cu numele de forma **<prenumenume>.zip**
- Arhiva ar trebui sa contina urmatoarele foldere cu fisierele aferente, exact asa cum sunt in **data_struct.zip**:



Important:
Daca nu implementati una dintre functii, va rugam sa nu stergeti implementarea implicita si sa nu stergeti semnatura functiei din header.
In cazul in care o functie nu reuseste sa faca operatiile, va returna **-1**.
Pentru implementare incercati sa folositi recomandarile din sectiunile **„Stil de codare”** si **„Practici de codare defensiva”.**

Stil de codare

1 Indentarea codului

Pentru editarea codului se va folosi indentarea cu 4 spatii, fara TAB-uri. In Visual Studio, setarile se fac astfel: la Tools — Options — Text Editor — C/C++ se seteaza: Tab size 4, Ident size 4, [x] Insert spaces.

2 Identificarea fisierelor header

Fiecare fisier .H va avea la inceput si sfarsit un #ifndef / #define / #endif cu un macro unic sau *#pragma once*, cu numele fisierului. De exemplu, pentru un fisier help_tools.h vom folosi:

```
#ifndef _HELP_TOOLS_H_
#define _HELP_TOOLS_H_

...

#endif // _HELP_TOOLS_H_
```

sau

```
#pragma once

...
```

3 Definirea si utilizarea macro-urilor

De regula definitiile de macro se scriu **UPPER CASE**. De exemplu:

```
#define FLAGS_REQUEST_NO_REPLY 0x00000001
#define PREPROCSTR(x) PREPROCSTR2(x)
```

Se va evita utilizarea excesiva a macro-urilor, si tendinta de definire a unui ’meta limbaj’. Macro-urile nu trebuie sa ascunda multiple functionalitati complexe, intrucat citirea codului si urmariea flow-ul de executie ar putea fi ingreunate.
Este interzisa in mod ferm crearea unor macro-uri care sa contina tratari automate de erori, sau instructiuni care modifica flow-ul de executie. Exemple de instructiuni interzise in macro-uri: return, goto, leave, break, continue etc.

4 Denumirea parametrilor functiilor

Denumirea parametrilor functiilor se face folosind **CapitalCase**. Exemplu:

```
int MyFunction( int FirstParameter );
```

5 Denumirea variabilelor locale

Denumirea variabilelor locale se face folosind **almostCapitalCase**. De exemplu:

```
int retStatus;
CC_LIST bufferLength;
```

Identificatorii dintr-un bloc de cod nu au voie sa aiba nume identic cu identificatori din blocurile ”parinte”, pentru a nu-i ascunde pe acestia.

5 Denumirea variabilelor globale

Denumirea variabilelor globale se face folosind **aImostCapitalCase** iar numele va fi prefixat cu „**g**”. De exemplu:

```
int gCcVector;
CC_LIST gList;
```

Identificatorii dintr-o functie nu au voie sa aiba nume identic cu identificatori globali, pentru a nu-i ascunde pe acestia.

6 Denumiri pentru struct, union

Denumirea de struct si union-uri se face **UPPER CASE**, iar definirea se va face atat pentru structura, cat si prefixat P pentru pointer la structura. Totodata se va folosi si definire prefixata cu _ la inceputul definitiei. Denumirea campurilor din definitii se va face CapitalCase. De exemplu:

```
typedef struct _MY_STRUCT {
    int Field;
    ...
} MY_STRUCT, *PMY_STRUCT;
```

7 Descrierea blocurilor de cod, utilizarea ¸si pozi¸tionarea acoladelor

Fiecare for, while, if, do, switch va fi urmat de { }, chiar daca exista o singura instructiune in bloc. Blocul {} se scrie la acelasi nivel de identare cu instructiunea, iar continutul blocului, indentat cu un nivel. In cazul if-ului, else-ul se scrie la acelasi nivel de identare cu if-ul. De exemplu:

```
for (i = 0; i < 100; i++)
{
...
}

if (a > b)
{
    while (x < 5)
    {
        x++;
    }
    // ...
}
else
{
...
}
```

8 Semnificatia numelor de variabile locale, functii, structuri etc

Parametrii, variabilele locale, functiile, structurile etc. trebuie sa aiba nume descriptive (si nu x, cucu, aB, myList, etc.). Singura exceptie este cazul variabilelor de ciclu / indecsi, dar si asta doar in cazul unor cicluri / indexari simple, evidente.

Practici de codare defensiva

1 Validarea parametrilor de intrare ai functiilor

Este obligatorie validarea parametrilor de intrare in functii. De exemplu, pentru un string se poate (presupunem ca nu acceptam parametrii de tip string de lungime 0):

```
if ((NULL == StrParam1) || (0 == StrParam1[0]))
{
    return -1;
}
```

Toate functiile care acceseaza un parametru trebuie sa il valideze inainte de utilizare.

2 Alocare/dezalocare memorie

Toate functiile care aloca memorie pentru folosire locala sau pentru apelarea altor functii, trebuie sa se asigure ca la iesirea din functie, se elibereaza memoria alocata.

Functiile care aloca memorie pentru folosire globala (de ex: vectorul in care se retin informatiile pentru ccvector), la apelul functiei de destroy, va trebui sa dezaloce memoria folosita de instanta respectiva. De asemenea, *trebuie sa se asigure ca nu se elibereaza memorie deja eliberata*.