OOP - Laborator 1 - cateva sfaturi si indicii

Citirea documentatiei si lucrul cu fisiere in C

Daca sunteti obisnuiti cu functiile din <fstream>, sa folositi functii din C (fopen, fclose, fgets, etc.) poate fi un pic peste mana. Dar ele sunt foarte bine documentate si o parte foarte foarte importanta din programare este citirea atenta a documentatiei si cautarea de exemple prin care sa intelegeti functionalitatea cum trebuie:

http://www.cplusplus.com/reference/cstdio/fopen/
http://www.cplusplus.com/reference/cstdio/fclose/
http://www.cplusplus.com/reference/cstdio/fgets/

!Obervatie: este posibil, daca folositi Visual Studio, sa primiti erori de genul acesta:
warning C4996: 'fopen': This function or variable may be unsafe. Consider using fopen_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
Este foarte bine ca, in productie, sa urmariti sa folositi variantele mai sigure ale functiilor. In scop didactic, insa, puteti folosi linistiti *_CRT_SECURE_NO_WARNINGS*. Tot ce trebuie sa faceti este sa scrieti INAINTE de a include headerele urmatorul define:
#define _CRT_SECURE_NO_WARNINGS

Sa luam ca exemplu fopen. Este foarte important, cand cititi orice documentatie, sa fiti atenti la urmatoarele aspecte:

- **ce face functia?**

Cred ca e primul lucru pe care trebuie sa il faceti. Nu vreti sa folositi o functie de care nu aveti nevoie. Vreau sa deschid un fisier, folosesc fopen, nu fread. Si tot asa. Cititi cu atentie, de 2 sau 3 ori daca e nevoie.

- **ce tip de valoare returneaza functia?**

Trebuie sa stiti ce tip de variabila o sa folositi ca sa pastrati valoarea returnata de acea functie, pentru verificari ulterioare; pentru fopen, de exemplu, avem prototipul:

*FILE * fopen (const char * filename, const char * mode);*

Asta inseamna ca ce returneaza va trebui sa fie pastrat intr-o variabila de tip FILE:

*FILE *f;*
f = fopen (<lista parametri blabla>)...

Intotdeauna, daca o functie returneaza ceva, este important sa tratati ce returneaza acea functie. Nu este pusa degeaba valoarea aceea acolo.

- **ce tip de parametri primeste acea functie?**

Intotdeauna respectati tipul parametrilor, ca sa nu va treziti cu erori neasteptate. Ea lucreaza cu parametri de un anumit tip si pentru asta a fost facuta.

- **ce reprezinta acei parametri?**

La fel ca si la primul subpunct de aici: cititi cu atentie detaliile despre acei parametri. De exemplu, pentru parametrul *mode* de la *fopen* aveti niste valori pe care le puteti folosi. Daca folositi ceva care nu este suportat o sa aveti o surpriza neplacuta la runtime.

- **ce returneaza acea functie?**

Si aici nu ma refer la tipul valorii returnate, pe care l-am tratat deja; ci la valorile posibile pe care le poate returna functia. Cred ca printre principalele chestii pe care trebuie sa le aveti in vedere sunt urmatoarele: ce returneaza atunci cand a functionat cum trebuie si ce returneaza atunci cand a intampinat erori. Iar de la acestea 2 putem sa avem mai multe ramificatii. Exemplul de la *fopen*:
If the file is successfully opened, the function returns a pointer to a FILE object that can be used to identify the stream on future operations.
Otherwise, a null pointer is returned.
On most library implementations, the errno variable is also set to a system-specific error code on failure.
Ce avem noi aici? Pai, daca totul a mers cum trebuie, avem un pointer valid la un obiect de fip FILE (adica *FILE* *). Daca are loc vreo eroare, o sa avem NULL returnat acolo. Si mai zice ca variabila errno va contine un cod de eroare.
Daca tot suntem aici: o sa mai vedeti, atunci cand folositi functii de prin C, ca in caz de eroare se seteaza *errno*. Nu e mare lucru de capul lui, ca principiu: este doar o variabila in care se pun niste informatii, care poate fi afisata daca includeti *errno.h* (sau *cerrno*, e acelasi lucru)

```
#include <cerrno>
void main() {
   printf("errno = %d\n", errno);
}
```

Functiile care vor modifica errno vor mentiona acest lucru si vor mai mentiona ce valoare vor pune acolo. In acest caz, este un *system-specific error code*. O lista cu acestea poate fi consultata aici:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms681382(v=vs.85).aspx
Sau daca folositi visual studio, *Tools -> Error Lookup* si scrieti acolo numarul erorii.
Insa, pentru ceva si mai simplu si mai sigur, textul erorii poate fi afisat folosind functia *strerror*, definita in header-ul *cstring* (aveti un pic mai jos un exemplu de utilizare).
Atunci cand folositi orice functie, aveti grija sa tratati toate cazurile de eroare care trebuie tratate in acel moment. Daca va bazati pe ceva returnat de acea functie, iar mai apoi functia esueaza si nu tratati eroarea, este posibil ca, prin faptul ca ati continuat executia, sa folositi ceva invalid.
De exemplu:

```
bool ReadBuffer(const char* filePath) {
   FILE *f;
   char buf[256];
   f = fopen("panicaintunel.txt", "r");
   fgets(buf, 10, f);
   return true;
}
```

In cazul de mai sus este posibil ca *fopen* sa returneze NULL si sa incercati sa cititi in buf de la adresa 0. Unde mai punem faptul ca nici erorile care pot aparea la *fgets* nu au fost tratate. Haideti sa vedem cum este tratat cazul mai bine:

```
bool ReadBuffer(const char* filePath) {
   FILE *f;
   char buf[256];
   f = fopen("panicaintunel.txt", "r");
   if(!f) {
      printf("could not open file; errno = %d\n", errno);
      return false;
   }
   if(!fgets(buf, 10, f)) {
      printf("could not read from file; errno = %d - %s\n", errno, strerror(errno));
      return false;
   }
   return true;
}
```

!Observatie: pentru o citire corecta folosind *fgets* trebuie sa aveti grija sa verificati erorile folosind si *feof* fi *ferror*, care sunt explicate in documentatie.

Sfaturi despre scrierea programelor voastre

Orice bucata de cod care este scrisa dezordonat din punct de vedere al limbajului de programare devine mult mai greu de mentinut si verificat pe termen lung. Asa ca am sa va prezint cateva sfaturi care v-ar putea ajuta:

- **modularizati, modularizati, modularizati**

In mod sigur puteti imparti orice proiect aveti in subproiecte. Pentru a fi mai usor sa vedeti structura proiectului final si pentru a reusi sa corectati mai usor erorile, incercati ca pentru fiecare "sarcina" a programului vostru sa aveti functii. De exemplu, pentru un program care citeste dintr-un fisier informatii, mai apoi le prelucreaza si apoi le afiseaza, puteti sa faceti 3 functii: una pentru citire, alta pentru prelucrare, si cealalta pentru afisare. Pentru functii, inca o idee buna este sa va folositi si de valoarea de return si de parametri. De exemplu, daca aveti o functie care face citirea dintr-un fisier a unui numar intreg, ati putea sa o declarati ceva de genul:

*bool ReadFromFile(char *filePath, int &retNr);*

Observati ca am declarat functia ca fiind de tip *bool*, cand probabil ati fi putut face si asa:

*int ReadFromFile(char *filePath);*

Chiar daca pare mai anevoios, primul mod de lucru e mai sigur. Sa presupunem ca *filePath* ar fi transmis, intr-o anume imprejurare, cu valoarea NULL. Daca avem un pointer la NULL nu vom putea face nimic mai departe. In al doilea caz, ce am putea returna? Daca returnam 0, -1, sau orice alt numar, este posibil sa scapam cazul in care fix acel numar este cel pe care l-am citit din fisier. Cu prima abordare, se poate returna *false*, iar in acest caz, nu ne vom mai baza pe ce se va afla in *retNr*.

Inca o observatie, poate un pic pe langa subiect: am folosit transmiterea prin referinta. O alternativa pentru aceasta este sa folositi, in loc de referinta, un pointer:
*bool ReadFromFile(char *filePath, int *retNr);*
Fiind tot o adresa, va avea acelasi efect, numai ca veti avea nevoie sa il dereferentiati cand veti schimba valoarea din el si sa aveti grija sa fie un pointer valid (!=NULL). Este un procedeu care pare mai complicat, insa este posibil sa va ajute sa intelegeti mai bine pointerii.

- **indentati, indentati, indentati**

Incercati intotdeauna sa indentati codul in functie de nivelul la care se afla. De exemplu, nu faceti asa:

```
if(conditie) {
instr1;
instr2;
if(conditie2) {
instr3;
}
}
```

Urat tare.Incercati mai degraba asa:

```
if(conditie) {
   instr1;
   instr2;
   if(conditie2) {
      instr 3;
   }
}
```

Este mult mai clar unde incepe si unde se termina un bloc de instructiuni. Iar cand veti avea foarte multe blocuri imbricate(desi ideal este sa nu aveti atat de multe, ci mai degraba sa incercati sa **modularizati**) va fi mult mai simplu de urmarit/citit/revizuit codul vostru.

- **intotdeauna dealocati memoria care a fost alocata/inchideti pointerii la fisiere**

Foarte important: fie ca folositi *new - delete*, fie *malloc - free*, intotdeauna delocati exact tot ce alocati. Si mare grija sa nu dealocati ceea ce nu e alocat. La fel se pune problema si cu variabilele de tip *FILE **. Aveti *fopen - fclose*. Printre cele mai mari probleme pot aparea pentru lucrul iresponsabil cu memoria. ATENTIE!

- **compilati pe parcurs**

Ceva frumos asta: ati implementat cod pentru citit de la tastatura, compilati. Ati facut o operatie cu niste valori citite, compilati. Chiar daca va structurati bine programul si logica din spate este sanatoasa, tot pot aparea probleme de compilare si la cei mai experimentati programatori; din cauza grabei, neatentiei, oboselii, emotiilor. Incercati, dupa ce ati implementat o parte mica din program, sa faceti un *Build*. Nu strica si astfel nu va treziti cu o gramada de cod cu multe erori la

final. Luati lucrurile treptat si puneti-le cap la cap. Sa vedeti multe erori una dupa alta e foarte frustrant.
!Observatie: daca tot suntem aici - cititi cu atentie erorile pe care le da compilatorul. Majoritatea sunt foarte explicite, iar daca nu le intelegeti, sa stiti ca de-a lungul timpului oamenii s-au intalnit cu majoritatea erorilor care va pot aparea. Aveti acces la atat de multa informatie incat este pacat sa nu profitati de asta: cautati si pe internet ce nu pricepeti; sunt sanse mari ca cineva sa explice pe intelesul vostru.

- **incercati sa pastrati acelasi stil in denumiri**

Incercati sa pastrati un anume fel de a denumi functiile, variabilele, constantele. De exemplu, eu prefer sa scriu numele functiilor incepand cu majuscule iar fiecare cuvant care compune numele sa fie tot cu litera mare (upper camel case se mai numeste - ex: *ReadFromFile*); pentru variabile folosesc ceva similar, dar incep cu litera mica (lower camel case - ex: *fileSize*), iar constantele si macro-urile au numele doar cu majuscule (*#define MADMAX 1*).
Asta le face mai usor de recunoscut in program. Nu este neaparat sa urmati acelasi standard ca si mine pentru codul vostru. Cel putin nu in acest laborator. Dar este important sa va stabiliti un stil si sa il folositi pentru a va gasi singuri mai usor problemele din codul propriu, iar sa va inspirati din cum folosesc gigantii industriei (ex: MSDN - CreateFile) este o idee de obicei buna.

- **folositi-va de comentarii**

In fiecare loc unde credeti ca, daca va uitati peste vreo luna in cod, v-ar lua prea mult sa pricepeti ce ati vrut sa faceti, puneti un comentariu. De exemplu, pentru *ReadFromFile* scris mai sus se poate pune lejer un comentariu precum:
// citeste un numar intreg din fisierul dat ca parametru
Sau orice credeti ca ar ajuta. Pentru orice conditie sau operatie care tine de un context anume si ar fi mai greu de inteles citind "din afara", incercati sa explicati ce ati vrut sa faceti, intr-un mod explicit si succint.