1 TDIC (Terminál Database Interpretre Calculator)

TDIC je terminálový program schopný primitivní práce s databází alb a také je schopen počítat matematické výrazy. Databáze je realizovaná lineárním seznamem, v němž každý prvek ukazuje na předchozí a na následující prvek. Z terminálového okna je uživatel schopen načíst databázi ze souboru .csv se správnou strukturou. Dále je možné setřídit seznam dle libovolného parametru, vyfiltrovat podle jednoho nebo více parametrů, spočítat množství záznamů, přidat, odebrat nebo změnit záznam a "databázi" opět uložit do souboru .csv. Výsledek filtru je také možno uložit do samostatného souboru .csv.

TDIC se skládá z několika souborů, které mají na starost řešení logických celků. Jednotlivé soubory obsahují dva typy funkcí, "private" a "public". Private funkce jsou přístupné jen z daného souboru, když jsem odkázán na čisté C a nemám private metody, tak je to hold takhle.

2 Obsluha programu

2.1 Početní část

Při používání početní části programu lze libovolný numerický příklad zadat v přirozené podobě
např. $2/4-5*(4+2/(2+3))$ program vyhodnotí jako -21.50. Implementované operátory jsou "+", "-", "*", "/" a "%" jakož to modulo. Krom zmíněných znamének může příklad obsahovat čísla a libovolné množství závorek. Výsledek se zobrazí jako prosté číslo v desetinném tvaru s přesností na dvě desetinná místa. Pokud při zadávání dojde k chybě v zadání (např. neuzavřete závorku nebo se v příkladu vyskytne nematematický symbol) program vypíše "bad syntax in " a část příkladu, ve která našel chybu. Příklad přesto vyhodnotí a výsledek vypíše hned pod chybovou hlášku, pokud jste tedy jen zapomněli uzavřít závorku, nemusíte příklad psát celý znova.

2.2 databázová část

Pokud zadáte funkci, která není definovaná, vypíše se "unknown function". Konkrétně jsou funkce popsány v následující kapitole. Pokud funkce přebírá parametry budou vždy odděleni mezerou. "_i" v názvu funkce značí, že jde o verzi funkce, která je určena pro parser a existuje i stejná funkce bez "_i" která je součástí "_i" a slouží pro použití uvnitř programu.

Implementované funkce	
název funkce v konzoly	název funkce uvnitř programu
load-file <file_path>	loadFile_i
del-album	delAlbum_i
save-album-list <file_path>	saveAlbumsList_i
list	printfAllAlbums_i
add-album <album data>	addNewAlbum_i
sort-albums <compare_faktor> <sort_dir>	sortAlbums_i
album-count	getAlbumCount_i
album <name>	getAlbum_i
filter-album <filter seting>	getAlbumSortedList_i
save-filter <file_path>	saveFilteredAlbumList_i
change-album <name> <changing param>	changeAlbumRecord
print-filter	printFilteredAlbumList_i

3 Popis databázových terminálových funkcí

3.1 Private funkce

```
bool readWold(char *buffer, char *word, int *length);
int getComperFaktor(char *compare_faktor);
bool getComperDirectory(char *compare_faktor);
```

Tyto funkce slouží pro zvýšení přehlednosti kódu a jejich název je podle mě plně vystihuje.

3.2 Public funkce

```
/*
    parse text, loaded file from file path from parameter, and print album list.
    you can use " ." as default file path "output/album_list.csv"
    you can coll the function from terminal as "load-file <file_path>"
*/
void loadFile_i(char *file_path);
```

Načte databázi ze souboru na adrese zadané v předané textu, který nejprve zkontroluje. Pro pohodlnost můžete využívat výchozí soubor na adrese "output/album_list.csv" pomocí parametru " .".
Příklad volání: "load-file output/album_list.csv" nebo "load-file ."

```
/*
    parse text and deleted album with name from the text
    you can coll the function from terminal as "del-album "
*/
void delAlbum_i(char *album_name);
```

Vymaže z databáze album jehož jméno jste zadali
Příklad volání: "del-album Vlasy" pro smazání alba s názvem Vlasy.

```
/*
    parse text and save album list to file on file address in the text
    the default address is "output/album_list.csv"
    you can coll the function from terminal as "save-album-list <file_path>"
*/
void saveAlbumsList_i(char *file_path);
```

Uloží databázi do souboru na zadané adrese. Opět můžete využít " ." namísto zdlouhavého "output/album_list.csv".
Příklad volání: "save-album-list output/album_list.csv" nebo "save-album-list ."

```
/*
    print all album
    you can coll the function from terminal as "list "
*/
void printfAllAlbums_i();
```

Vypíše všechna alba v databázi.
Příklad volání: "list".

```
/*
    parse text and make a new album with data from text
    data taken in order:
    name interpreter year genre score
    divided by spaces
    you can call the function from a terminal as "add-album <album data>"
*/
void addNewAlbum_i(char *album);
```

Vytvoří nové album z předaných parametrů.
Příklad volání: "add-album jmeno interpret 2010 zanr 8" pro vytvoření alba se jménem "jmeno" od autora "interpret" z roku "2010" žánru "zanr" a hodnocením 8.

```
/*
    parse text and sort album according to "compare_faktor" with the director from "sort_dir".
    compare_factor:
        name
        interpreter
        year
        genre
        score
    sort_dir: defoult is ascending
        VV
    to end, print the album list
    you can call the function from a terminal as "sort-albums <compare_faktor> <sort_dir>"
*/
void sortAlbums_i(char *compare_factor_and_sort_dir);
```

Seřadí alba podle zadaného parametru, ve výchozím stavu vzestupně a při zadání přepínače ”VV” sestupně. Parametr lze zadat klíčovými slovy ”name”, ”interpreter”, ”year”, ”genre” a ”score”.
Příklad volání: ”sort-albums score” pro seřazení alb podle jejich hodnocení.

```
/*
    print album count
    you can call the function from a terminal as "album-count "
*/
void getAlbumCount_i();
```

Spočítá alba v databázi a vypíše jejich počet.
Příklad volání: ”album-count”.

```
/*
    parse text and print album
    you can call the function from a terminal as "album <name>"
*/
void getAlbum_i(char *name);
```

Vypíše záznam o albu zadaného jména. Předpokládá že jméno slouží jako jednoznačný identifikátor alba.
Příklad volání: ”album Vlasy” pro výpis záznamu o albu Vlasy.

```
/*
    parse text and filter album list.
    the argument is desired values in filtered list in order:
    name interpreter year genre score
    divided by spaces
    if some value is not important you can replace it with "-" symbol.

    for example, you can use the command "filter-album - - - Rock -" to filter Rock albums
    you can call the function from a terminal as "filter-album <filter seting>"
*/
void getAlbumSortedList_i(char *album_prototype);
```

Vyfiltruje alba jejichž položky odpovídají zadaným. Pokud je třeba filtrovat jen podle některých položek zadejte místo položek nevyužitých ”-”. Všechny parametry musí být vždy odděleny mezerou.
Příklad volání: ”filter-album - - 2003 - -” pro vyfiltrovat všech alb z roku 2003.

```
/*
    parse text and save filtered album list to file on address in the text.
    the default address is "output/filtered_list.csv"
    you can call the function from a terminal as "save-filter <file_path>"
*/
void saveFilteredAlbumList_i(char *file_path);
```

Uloží vyfiltrovaná alba do souboru který je předán v parametru. Opět vze použít ”.” namísto ”output/filtered_list.csv” jakožto výchozí soubor.
Příklad volání: ”save-filter output/filtered_list.csv” nebo ”save-filter .”.

```
/*
    parse text and album with entered name.
    the argument is desired values in filtered list in order:
    name interpreter year genre score
    divided by spaces
    if some value is not change you replace it with "-" symbol.

    for example, you can use the command "change-album Vlasy - 2020 - -" to change yer of publication in albu
    you can call the function from a terminal as "change-album <name> <changing param>"
*/
void changeAlbumRecord(char *change_list);
```

Přepíše kteroukoli položku alba krom jména alba, podle kterého identifikuje upravované album. Parametru které mají zůstat stejné budou nahrazeny pomlčkou "-".
Příklad volání: "change-album Vlasy - - - 8" pro změnu původního hodnocení alba "Vlasy" na hodnocení 8.

```
/*
    print all albums in the filtered list
    you can coll the function from terminal as "print-filter "
*/
void printFilteredAlbumList_i();
```

Vypíše všechna alba která která jsou aktuálně vyfiltrovaná.
Příklad volání: "print-filter ".

4 Parser

4.1 Private funkce aritmetické části parseru

```
bool isNumber(char x);
bool isOperator(char x);
bool isMathSim(char x);
bool isLatter(char x);
bool isSymbolValid(char x);
```

Tyto funkce slouží pro pohodlnější definici podmínek a jediné co dělají je, že ověřují, zda je daný znak číslo, operátor atd.

```
readed_number_t loadAnother(char *another);
readed_number_t loadNumber(char *number);
readed_number_t loadBracket(char *bracket);
```

Tyto funkce slouží matematickému parseru pro přečtení části řetězce s matematickým výrazem. Funkce "*loadAnother*" slouží jako rozcestí, které rozhoduje, zda se má zavolat *loadNumber* nebo *laadBracket*. Všechny tři následně vracejí strukturu, která obsahuje hodnotu dané části výpočtu a její délku, kterou zabírala ve vstupním řetězci.

```
readed_number_t combineNumber(readed_number_t a, readed_number_t b, char marker);
```

Tato funkce slouží pro samotné kombinování (sčítání, odčítání atd.) čísel ze struktur "a" a "b" podle znaménka v parametru "marker".

4.2 Private funkce databázového části parseru

```
void doDatabaseFunction(char *command, int command_length);
```

Tato funkce je rozcestí pro konkrétní databázové funkce, které jsou vždy popsané při své deklaraci.

4.3 Funkce pro použití mimo knihovnu

```
/*
    load number from text and return his value and his length in text format
*/
readed_number_t loadNumber(char *number);
```

Načte číslo z textu v paměti na adrese "number" a vrátí ho ve struktuře "readed_number_t". Tato struktura obsahuje číselnou hodnotu čteného čísla a počet znaků, který v textu zaujímalo.

```
/*
    load line from the terminal and add ";;" to the end of the string
*/
int readLine(char *buffer);
```

Načte do paměti na adresu "buffer" textu, který uživatel zadal do konzole a ukončí jej dvěma středníky ";;" podle kterých se pak orientuje následující část programu.

```
/*
    interpret text ended ";;"
    you can calculate math expressions and do database functions
*/
bool interpretLine(char *line);
```

Spočítá matematické příklady a vykoná databázové příkazy uložené v textovém řetězci v paměti na adrese "line". Předpokládá zakončení řetězce dvěma středníky ";;".

```
/*
    calculate expressions from text ended ";;"
*/
readed_number_t interpretArithmeticsFunction(char *example);
```

Vypočte matematický příklad uložený v textovém řetězci v paměti na adrese "example". Předpokládá zakončení řetězce dvěma středníky ";;".

```
/*
    do databas function describ in text "command" and end ";;"
    function is describ in file "album_driver_interpret.h"
*/
void interpretDatabaseFunction(char *command);
```

vykoná databázové příkazy uložené v textovém řetězci v paměti na adrese "command". Předpokládá zakončení řetězce dvěma středníky ";;".

5 Vzhled a základní funkce databáze

Každý záznam o albu je realizován pomocí struktury "album_t", která nese informaci o názvu"char name[32]", autorovy "char interpreter[32]", roku vydání "int year", žánru "char genre[32]", hodnocení "float score" a ukazatel na předchozí a následující album v lineárním seznamu.

```
typedef struct album_t
{
        char name[32];
        char interpreter[32];
        int year;
        char genre[32];
        float score;
        int info;
        album_t *next;
        album_t *prev;
} album_t;
```

5.1 Private funkce databáze

```
album_t *loadAlbum(album_t *prev, FILE *file);
int compareAlbums(album_t *a, album_t *b, int compare_faktor, bool sort_dir);
void controllPrintfAllAlbums(album_t *album);
```

5.2 Public funkce databáze

```
/*
        load file .csv on address file_path to memory and return address to the
        first member of the list from the file
    if the file non exist return NULL
*/
album_t *loadFile(char *file_path);
```

Načte databázi ze souboru na adrese "file_path" a vrátí ukazatel na první prvek této databáze. Pokud soubor neexistuje nebo z jiného důvodu nejde otevřít, vypíše do konzole "Error!!! the file \"%s\" does not exist or is not readable" a vrátí NULL.

```
/*
        save album "album" list to file on address file_path
*/
bool saveAlbumsList(album_t *album, char *file_path);
```

Uloží databázi do souboru na adrese "file_path" ve formátu csv.

```
/*
        add new album to album list "album" with name "name", interpreter "interpreter",
        year "year", genre "genre", score "score"
        return pointier to the new album
*/
album_t *addNewAlbum(album_t *album_list, char *name, char *interpreter, int year,
                     char *genre, float score);
```

Vytvoří nové album a zařadí ho na konec lineárního seznamu na adrese "album_list". Vrací ukazatel na nové album.

```
/*
        remove album on address "album" from the album list and deleted it
*/
void delAlbum(album_t *album);
```

Odstraní album na adrese "album" z lineárního seznamu a následně uvolní jeho paměť.

```
/*
    flip order of albums "a" and "b"
*/
album_t *switchAlbums(album_t *a, album_t *b);
```

Prohodí pořadí alb "a" a "b" v lineárním seznamu.

```
/*
        sort album according to "compare_faktor" in order "sort_dir"
        you can us enum compare_faktor:
        name            = 0
        interpreter = 1
        year            = 2
        genre           = 3
        score           = 4
*/
album_t *sortAlbums(album_t *first_album, album_factor compare_factor, bool sort_dir);
```

Seřadí databázi podle položky předané v parametru "compare_factor" ve směru určeném parametrem "sort_dir". Pro určení "compare_factor" můžete využít enum "album_factor".

```
typedef enum
{
        albumName,
        albumInterpreter,
        albumYear,
        albumGenre,
        albumScore,
} album_factor;
```

```
/*
        print album on address "album"
*/
void printfAlbum(album_t *album);
```

Vytiskne do konzole album na adrese "album".

```
/*
        print all albums in list start it on address "album"
*/
void printfAllAlbums(album_t *album_list);
```

Vytiskne do konzole všechna alba ze seznamu začínajícího na adrese "album_list".

```
/*
    calculate the count of albums in the list starting with address "album_list"
*/
int getAlbumCount(album_t *album_list);
```

Vrátí počet alb v seznamu na adrese "album_list"

```
/*
        return address to the album with a name "name" from the album list "album list"
*/
album_t *getAlbum(album_t *album_list, char *name);
```

Vrátí ukazatel na album s názvem "name".

6 Filtrování alb

6.1 Private funkce filtru

```
    bool compareAlbumFactor(album_t *album, album_compare_prototype_t *compare_prototype);
```

Porovná alba na adrese "album" s prototypem "album_compare_prototype_t" na adrese "compare_prototype". V prototypu jsou zapsány hodnoty všech faktorů alba, pokud se některý faktor porovnávat nemá, v prototypu bude v dané složce '\0' pro text a -1 pro číslo.

6.2 Funkce pro použití mimo knihovnu

```
/*
        add a new album to a virtual album list.
        the virtual album list is only a list of pointier to album in album list composite
    from album_t structs
*/
album_list_t *addNewAlbumToList(album_list_t *prev, album_t *new_album);
```

Přidá strukturu "album_list_t" do lineárního seznamu. Struktura "album_list_t" obsahující ukazatele na album uložená v lineárním seznamu složeném ze struktur "album_t". Dále obsahuje ukazatel na předchozí a následující položku seznamu.

```
/*
    remove album on address "album" from his virtual album list and deleted it
*/
void delAlbumFromList(album_list_t *album);
```

Smaže album na adrese "album" z filtrovaného seznamu.

```
/*
        filter album corresponding with "album_prototype"
        if some text value is not important write in '\0'
        if some number value is not important write in -1
*/
album_list_t *getAlbumSortedList(album_t *album_list,
                                 album_compare_prototype_t *album_prototype);
```

Vyfiltruje všechna alba odpovídající prototypu, vytvoří z nich lineární seznam a vrátí ukazatel na první prvek tohoto seznamu. V prototypu jsou zapsány hodnoty všech faktorů alba, pokud je pro filtraci nějaký faktor nepodstatný bude v dané složce prototypu "\0" pro text a -1 pro číslo.

```
/*
        print all album in virtual album list
*/
void printFilteredAlbumList(album_list_t *list);
```

Vytiskne do konzole všechna alba z filtrovaného seznamu na adrese "list".

```
/*
    save all albums in virtual album list to file on address "file_path"
*/
bool saveFilteredAlbumList(album_list_t *list, char *file_path);
```

uloží do souboru na adrese "fil0e_path" všechna alba z filtrovaného seznamu na adrese "list".

7 Asistent pro načítání souboru

Pro jednoduchost budu dále používat zkratku ANS (Asistent pro Načítání Souboru).

7.1 Private funkce ANS

```
    bool itsANmber(char sim);
    bool itsAText(char sim);
    bool itsASpecialSymbol(char sim);
```

Tyto funkce slouží pro pohodlnější definici podmínek a jediné co dělají je, že ověřují, zda je daný znak číslo, text nebo speciální znak.

7.2 Funkce pro použití mimo knihovnu ANS

```
/*
    open file on address "file_pad"
    if the file does not exist or is not readable print "Error!!! the file \"%s\"
    does not exist or is not readable" and return NULL
*/
FILE *openFile(char *file_path, char *mode);
```

Otevře soubor a vrátí ukazatel, pokud soubor neexistuje nebo není z nějakého důvodu čitelný, vypíše do terminálu "Error!!! the file \"%s\" does not exist or is not readable" a vrátí NULL. Na místo %s pochopitelně vypíše cestu k souboru který neotevřel.

```
/*
    read text in file to ';' and if the text in number translates it
    return struct call_t with read text, number value and actual state in reading file:
        end of call = normal  = 0
        end of line = endLine = 1
        end of file = endFile = 2
    in struct is info about call type:
        anything = 0 //
        number   = 1 // is translated to number
        text     = 2 // is contane only 'a'->'z' 'A'->'Z' '0'->'9' '.' ',' space '_' '-'
*/
cell_t readFileCell(FILE *file);
```

Přečte buňku ze souboru a vrátí ji ve struktuře "cell_t". Struktura "cell_t" slouží pro snadnou orientaci v načtených datech. Struktura obsahuje načtený text "char text[32]", pokud se jedná o číslo tak obsahuje jeho číselnou hodnotu "double number"(pokud nejde o číslo číselná složka bude = 0), informaci o typu dat "cell_type type" a nakonec informaci o průběhu čtení souboru "call_end index".
Pro složku "cell_type type" můžete využít enum "cell_type".

```
typedef enum
{
    anything = 0,
    number,
    text,
}cell_type;
```

Pro složku "call_end index" můžete využít enum "call_end".

```
typedef enum
{
    normal = 0,
    endLine,
    endFile,
}call_end;
```