**Курсова работа по дисциплина ПИК2 за студенти ФКСТ**

**Емилиян Емилов Ризов - 121219091, 31гр, КСИ, ФКСТ**

**Бюлбюл Зюлямова -**

**Текст на заданието**

Да се състави програма тип „меню“ за поддръжка на компютърен библиотечен каталог със следните изисквания:

1. За всяка библиотечна единица, намираща се в библиотеката, се съхраняват следните данни

* Каталожен номер – буквено – цифрена информация (20) – уникален индекс
* Заглавие – текст (80)
* Автор – 50 символа
* Цена – реална стойност
* Година на издаване
* Издателство – текст (80)

1. Информацията се съхранява в подходяща бинарна файлова структура, като се осигуряват следните функции:
2. Въвеждане на нова каталожна единица
3. Актуализация на данните по каталожен номер
4. Търсене на каталожна единица по заглавие
5. Търсене по автор
6. Данните се поддържат в динамична структура – едносвързан списък в оперативната памет

**Описание на използвани функции**

**void add\_new\_unit(t\_node\*\*, char \*);**

* Отваря бинарен файл в режим „ab”
* Създава временна променлива от тип структурата, която съдържа само променливите за информация относно дадена книга
* Преминава през свързания списък, като целта е крайният елемент
* Щом достигне крайния елемент, неговата информация се записва в предварително създадената променлива
* След това тази променлива се записва в бинарния файл
* Файлът се затваря и функцията приключва
* Параметри - двоен указател, който служи за head node-a, както и указател от тип char, който се ползва за името на бинарния файл

**void delete\_list(t\_node\*\*);**

* Параметър - двоен указател, който служи за head-a
* Най – просто, преминава през списъка, като следи текущия и следващия елемент и трие(освобождава от паметта) всеки текущ елемент

**void add\_tail(t\_node \*\*head, t\_lib temp\_storage);**

* Параметри – двоен указател – head и променлива(структура) от тип t\_lib
* Функцията служи за добавяне на нов елемент в края на свързания списък, като първо проверява дали списъка е празен, ако е, head-a сочи към новия първи елемент, ако не е, преминава докрая и добавя новия елемент там

**void read\_list(t\_node \*\*head, char \*file\_name);**

* Параметри – двоен указател – head и указател от тип char, служещ за името на бинарния файл
* Създава се временна променлива от тип t\_lib(структурата за информацията)
* Отваря се файлът в режим за четене
* Прочетената информация се запазва в предварително създадената променлива
* След това се извиква функцията add\_tail(), която се ползва да добави прочетената информация в края на свързания списък

**void search\_by\_title(t\_node \*\*head, char\*);**

* Параметри – двоен указател – head и указател тип char – име на търсената книга въведено от потребителя
* Преминава през целия списък, като сравнява името на всяка книга с търсената. След като има съвпадение изписва пълната информация относно дадената книга

**void search\_by\_author(t\_node \*\*head, char \*u\_author);**

* Параметри – двоен указател – head и указател тип char – име на търсения автор въведен от потребителя
* Преминава през целия списък, като сравнява името на всеки автор с търсения. След като има съвпадение изписва пълната информация на всички книги, които са във файла, написани от съответния автор

**void update\_by\_id(t\_node \*\*head, char \*u\_id, char\*);**

* Параметри – двоен указател – head, указател тип char – търсен каталожен номер въведен от потребителя, указател от тип char, служещ за името на бинарния файл
* Отваря се бинарния файл
* Прочита се информацията от него, като се създава списък
* Преминава през целия списък, като сравнява въведения от потребителя каталожен номер с каталожния номер на всеки един елемент, докато няма съвпадение.
* След като има съвпадение потребителя избира дали да промени цената на дадената книга или годината на издаване
* След промените целият списък се презаписва във бинарния файл

**Общо описание за функционирането на програмата**

Програмата започва с меню, от което може да се избират различните операции описани в заданието. Избира се желаната операция съответстваща на числата от 1 до 5, след което се извиква съответстващата функция. Ако случайно се извика някоя от функциите, които за търсене, но файлът е празен, на екрана не се появява нищо, след което менюто се зарежда отново. При избиране на последната опция за прекратяване на програмата, динамично създаденият свързан списък се освобождава елемент по елемент и чак след това приключва програмата.

**Source code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

//DATA STRUCTURE

typedef struct {

char id[20];

char title[80];

char author[50];

float price;

int publish\_year;

char publisher[80];

}t\_lib;

//NODE STRUCTURE

typedef struct node {

t\_lib data;

struct node \*next;

}t\_node;

//FUNCTION PROTOTYPES

void add\_new\_unit(t\_node\*\*, char \*);

void delete\_list(t\_node\*\*);

void add\_tail(t\_node \*\*head, t\_lib temp\_storage);

void read\_list(t\_node \*\*head, char \*file\_name);

void search\_by\_title(t\_node \*\*head, char\*);

void search\_by\_author(t\_node \*\*head, char \*u\_author);

void update\_by\_id(t\_node \*\*head, char \*u\_id, char\*);

int main() {

t\_node \*head = NULL;

char \*file\_name = "info.bin";

FILE \*temp\_fp = fopen(file\_name, "rb");

if (temp\_fp == NULL) {

temp\_fp = fopen(file\_name, "wb");

}

fclose(temp\_fp);

FILE \*tp = fopen("ids.txt", "r");

if (tp == NULL) {

tp = fopen("ids.txt", "w");

}

fclose(tp);

t\_lib temp\_storage;

int answer;

int check = 0;

char \*u\_title = (char\*) malloc(sizeof(char)\*80);

char \*u\_author = (char\*) malloc(sizeof(char)\*50);

char \*u\_id = (char\*) malloc(sizeof(char)\*20);

char \*temp\_id = (char\*) malloc(sizeof(char)\*20);

while(1) {

printf("\n1. Add new\n2. Update by id\n3. Search by title\n4. Search by author\n5. Exit\nAnswer: ");

scanf("%d", &answer);

if (answer == 1) {

tp = fopen("ids.txt", "r");

printf("\n");

printf("Type in info\n======================\n");

fflush(stdin);

gets(temp\_storage.id);

gets(temp\_storage.title);

gets(temp\_storage.author);

scanf("%f", &temp\_storage.price);

scanf("%d", &temp\_storage.publish\_year);

fflush(stdin);

gets(temp\_storage.publisher);

while (fscanf(tp, "%s", temp\_id) != EOF) {

if (strcmp(temp\_id, temp\_storage.id) == 0) {

printf("ID already exists\nTry again\n");

fclose(tp);

check = 1;

break;

}

}

if (check != 1) {

add\_tail(&head, temp\_storage);

add\_new\_unit(&head, file\_name);

printf("\n");

}

check = 0;

}

else if (answer == 2) {

printf("\n");

printf("Type in the desirable id: ");

fflush(stdin);

gets(u\_id);

delete\_list(&head);

read\_list(&head, file\_name);

update\_by\_id(&head, u\_id, file\_name);

}

else if (answer == 3) {

printf("\n");

printf("Type in the desirable book title: ");

fflush(stdin);

gets(u\_title);

delete\_list(&head);

read\_list(&head, file\_name);

search\_by\_title(&head, u\_title);

}

else if (answer == 4) {

printf("\n");

printf("Type in the desirable author: ");

fflush(stdin);

gets(u\_author);

delete\_list(&head);

read\_list(&head, file\_name);

search\_by\_author(&head, u\_author);

}

else if (answer == 5) {

delete\_list(&head);

free(u\_title);

free(u\_id);

free(u\_author);

free(temp\_id);

return 0;

}

}

}

//FUNCTIONS

void delete\_list(t\_node \*\*head) {

t\_node \*current = \*head;

t\_node \*next;

while (current != NULL) {

next = current->next;

free(current);

current = next;

}

\*head = NULL;

}

void add\_tail(t\_node \*\*head, t\_lib temp\_storage) {

t\_node \*temp = \*head;

t\_node \*new\_node = (t\_node\*) malloc(sizeof(t\_node));

new\_node->data = temp\_storage;

new\_node->next = NULL;

if (\*head == NULL) {

\*head = new\_node;

return;

}

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new\_node;

}

void add\_new\_unit(t\_node \*\*head, char \*file\_name) {

FILE \*fp;

t\_node \*temp = \*head;

t\_lib temp\_storage;

fp = fopen(file\_name, "ab");

if (fp == NULL) {

printf("cant open file...\n");

}

while (temp->next != NULL) {

temp = temp->next;

}

temp\_storage = temp->data;

FILE \*tp = fopen("ids.txt", "a");

fprintf(tp, "%s\n", temp\_storage.id);

fclose(tp);

if (fwrite(&temp\_storage, sizeof(t\_lib), 1, fp) == 1) {

//printf("writing successful\n\n");

}

else

printf("error writing...\n");

fclose(fp);

}

void read\_list(t\_node \*\*head, char \*file\_name) {

FILE \*fp = fopen(file\_name, "rb");

if (fp == NULL) {

printf("cant open the file\n");

return;

}

t\_lib temp\_storage;

while (!feof(fp)) {

fread(&temp\_storage, sizeof(t\_lib), 1, fp);

add\_tail(head, temp\_storage);

}

fclose(fp);

}

void search\_by\_title(t\_node \*\*head, char \*u\_title) {

t\_node \*temp = \*head;

printf("=====================\n");

while (temp->next != NULL) {

if (strcmp(u\_title, temp->data.title) == 0) {

printf("%s\n%s\n%s\n%.2f\n%d\n%s\n\n",

temp->data.id, temp->data.title, temp->data.author, temp->data.price, temp->data.publish\_year, temp->data.publisher);

}

temp = temp->next;

}

printf("=====================\n");

}

void search\_by\_author(t\_node \*\*head, char \*u\_author) {

t\_node \*temp = \*head;

printf("=====================\n");

while (temp->next != NULL) {

if (strcmp(u\_author, temp->data.author) == 0) {

printf("%s\n%s\n%s\n%.2f\n%d\n%s\n\n",

temp->data.id, temp->data.title, temp->data.author, temp->data.price, temp->data.publish\_year, temp->data.publisher);

}

temp = temp->next;

}

printf("=====================\n");

}

void update\_by\_id(t\_node \*\*head, char \*u\_id, char \*file\_name) {

t\_node \*temp = \*head;

int answer;

t\_lib temp\_storage;

printf("=====================\n");

while (temp->next != NULL) {

if (strcmp(u\_id, temp->data.id) == 0) {

printf("Update attribute\n1. price\n2. publishing year\nAnswer: ");

scanf("%d", &answer);

if (answer == 1) {

float new\_price;

printf("New price: ");

scanf("%f", &new\_price);

temp->data.price = new\_price;

}

else {

int new\_year;

printf("New year: ");

scanf("%d", &new\_year);

temp->data.publish\_year = new\_year;

}

}

temp = temp->next;

}

FILE \*fp = fopen(file\_name, "wb");

temp = \*head;

while (temp->next != NULL) {

temp\_storage = temp->data;

if (fwrite(&temp\_storage, sizeof(t\_lib), 1, fp) == 1) {

//printf("writing successful\n\n");

}

else

printf("error writing...\n");

temp = temp->next;

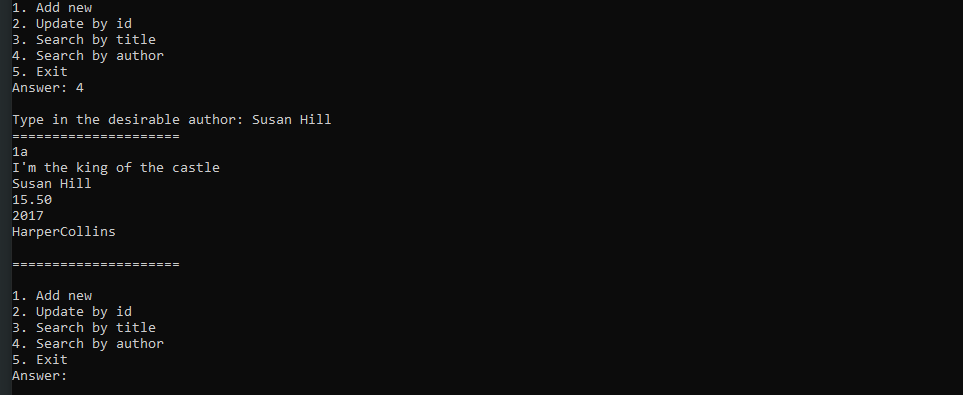
}

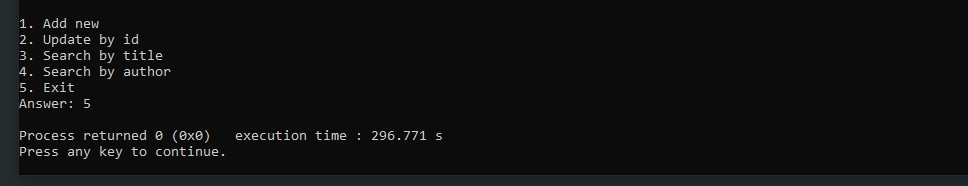
fclose(fp);

printf("=====================\n");

}

**Резултати от изпълнението на програмата**

****

****