Dokumentacja do projektu

Przedmiot: **PRI**
Autor: **Rafał Kulus**

Temat projektu:
Przychodnia lekarska

Zastosowany algorytm sortowania:
Merge sort

Informacje szczegółowe:
Kompilacja poleceniem *make*. Program powinien być odporny na nieprawidłowy input zarówno z konsoli, jak i z pliku. Menu jest intuicyjne, a co za tym idzie, użytkownik nie powinien mieć najmniejszego problemu z odkryciem wszystkich funkcji oferowanych przez program. Każdy element listy przechowuje następujące dane o pacjencie: imię, nazwisko, numer PESEL, płeć, stan pacjenta (zarejestrowany, zapisany na wizytę, w szpitalu) i liczbę odbytych wizyt.

Link do projektu na GitHubie:
https://github.com/BlueAlien99/PRI_Projekt3

PatientsDB.h

Plik zawiera wszystkie funkcje niezbędne do działania programu i bezpośrednio z nim związane, lecz które nie dotyczą operacji wczytywania lub zapisywania z/do pliku.

`void addPatientWizard(Patient **head, Patient **tail);`
Funkcja odpowiada za wczytywanie od użytkownika imienia, nazwiska, nr PESEL i płci pacjenta, które następnie zostaną przekazane funkcji *addPatient*.

`void addPatient(Patient **head, Patient **tail, char name[], char surname[], char pesel[], int sex, int state, int visits);`
Funkcja odpowiada za prawidłowe dodanie pacjenta do bazy wykorzystując przy tym otrzymane parametry.

`Patient* findPatient(Patient **m_head, Patient **m_tail, _Bool info);`
Funkcja wyszukuje pacjenta na podstawie jego imienia i nazwiska lub numeru PESEL. W zależności od kontekstu woła także *printPatientInfo* i *findPatientMenu*.
Return: Wskaźnik do znalezionego pacjenta, NULL w przeciwnym wypadku.

`void printPatientInfo(Patient *head);`
Funkcja wypisuje na ekran wszystkie dane pacjenta znalezionego za pomocą *findPatient*.

`int findPatientMenu(Patient **m_head, Patient **m_tail, Patient *head);`
Funkcja wyświetla menu akcji, które można wykonać na pacjencie. Jeśli użytkownik nie postanowi powrócić do głównego menu, po wykonaniu akcji funkcja zostanie wywołana ponownie.
Return: 1 jeśli użytkownik wybrał, by powrócić do głównego menu, 0 w przeciwnym wypadku.

`void delPatient(Patient **head, Patient **tail, Patient *el);`
Funkcja usuwa pacjenta z listy. Wskaźnik do niego może zostać przekazany jako parametr lub funkcja zawoła *findPatient*.

`void moveEl(Patient **src, Patient **target, Patient **targetTail);`
Funkcja przenosi pierwszy element listy *src* na koniec listy *target*, a więc dołączając go do *targetTail*.

`void mergeSortWizard(Patient **head, Patient **tail);`
Funkcja woła *mergeSort*, a po zakończonym sortowaniu ustawia *tail* na ostatni element listy.

`Patient* mergeSort(Patient *head);`
Główna funkcja odpowiadająca za sortowanie. Dzieli duże listy na mniejsze przy pomocy *moveEl*, które następnie łączy za pomocą *merge*.
Return: Wskaźnik do pierwszego elementu posortowanej listy (zwrócony przez *merge*).

`Patient* merge(Patient *left, Patient *right);`
Funkcja dodaje elementy jednej listy do drugiej porównując pola *surname*, a w przypadku równości także *name*.
Return: Wskaźnik do pierwszego elementu posortowanej listy.

`int printPatients(Patient *head, _Bool info);`
Funkcja wypisuje imiona i nazwiska pacjentów wraz z ich stanami.
Return: Liczba pacjentów w bazie.

`void getStats(Patient *head);`
Funkcja generuje statystyki na podstawie danych pacjentów znajdujących się w bazie, a następnie woła *printStats*.

`void printStats(uint stats[][NOFSTATS], double avgvisit[]);`
Funkcja wypisuje na ekran statystyki wygenerowane przez *getStats* z częściową pomocą funkcji *printStatsLine*.

`void printStatsLine(char header[], uint stats[][NOFSTATS], int i);`
Funkcja pomocnicza do funkcji *printStats*.

`void populateDB(Patient **head, Patient **tail);`
Funkcja generuje losowe osoby, które następnie dodaje do listy.

`void freeDB(Patient **head, Patient **tail);`
Funkcja odpowiada za dealokację pamięci zaalokowanej na dane pacjentów. Jest to ostatnia funkcja wołana przed zakończeniem programu.

IOHandler.h

Plik zawiera funkcje odpowiadające za wczytywanie i zapisywanie z/do pliku danych pacjentów.

`void getPath(char path[], _Bool bin);`
Funkcja odpowiada za wczytywanie ścieżki do odczytu/zapisu pliku, a także dostarcza domyślną lokalizację w przypadku braku inputu od użytkownika.

`void saveFile(Patient *head);`
Funkcja odpowiada za zapis danych pacjentów do pliku tekstowego w formacie à la JSON.

`void readFile(Patient **head, Patient **tail);`
Funkcja odpowiada za prawidłowe wczytywanie danych pacjentów z pliku tekstowego. Właściwe dodanie do listy odbywa się za pomocą funkcji *addPatient*.

`_Bool readChar(FILE *file, char c, char *readchar);`
Funkcja wczytuje z pliku linię, a następnie porównuje jej pierwszy znak z *c*.
Return: 1 jeśli znaki się zgadzają, 0 w przeciwnym wypadku.

`_Bool readString(FILE *file, const char str[], char des[]);`
Funkcja wczytuje z pliku string o podanym kluczu *str*.
Return: 1 jeśli udało się wczytać string i spełnia on wymagania *validateString*, 0 w przeciwnym wypadku.

`_Bool readInt(FILE *file, const char str[], int *n);`
Funkcja wczytuje z pliku liczbę typu int o podanym kluczu *str*. Wykorzystywana jest do tego funkcja *readString*, a następnie wczytany string, jeśli składa się tylko i wyłącznie z cyfr, jest konwertowany do inta.
Return: 1 jeśli udało się wczytać liczbę, 0 w przeciwnym wypadku.

`void saveFileBin(Patient *head);`
Funkcja odpowiada za zapis danych pacjentów do pliku binarnego.

`void readFileBin(Patient **head, Patient **tail);`
Funkcja odpowiada za prawidłowe wczytywanie danych pacjentów z pliku binarnego. Właściwe dodanie do listy odbywa się za pomocą funkcji *addPatient*.

Utilities.h

Plik akumuluje funkcje, które nie są wyłączne dla projektu, a mogą się przydać także poza nim.

`_Bool clearBuffer();`
Funkcja odpowiada za czyszczenie bufora.
Return: 1 jeśli w buforze zostały śmieci, 0 w przeciwnym wypadku.

`_Bool getString(char str[]);`
Funkcja odpowiada za wczytywanie stringa o ustalonej maksymalnej długości.
Return: 0 jeśli string przekroczył długość, w przeciwnym wypadku wynik funkcji *validateString*.

`_Bool getStringForm(char str[], const char form[]);`
Funkcja wypisuje na ekran stringa *form*, a następnie woła funkcję *getString*.
Return: Wynik funkcji *getString*.

`_Bool validateString(char str[]);`
Funkcja sprawdza, czy string składa się tylko i wyłącznie ze znaków alfanumerycznych (bez spacji).
Return: 1 jeśli string spełnia w.w. wymagania, 0 w przeciwnym wypadku.

`_Bool verifyPesel(char p[], int sex);`
Funkcja sprawdza, czy podany string jest prawidłowym numerem PESEL w zakresie: ilości znaków, występowania tylko cyfr, zgodności cyfry oznaczającej płeć i zgodności cyfry kontrolnej. Następnie jest wywoływana funkcja *getDoBfromPesel*.
Return: 1 jeśli string jest prawidłowym numerem PESEL, 0 w przeciwnym wypadku.

`_Bool getDoBfromPesel(char p[]);`
Funkcja generuje i wypisuje na podstawie numeru PESEL datę urodzenia osoby, a jeśli ta nie może zostać wygenerowana, zwraca informację o błędzie.
Return: 1 jeśli udało się wygenerować datę urodzenia, 0 w przeciwnym wypadku.

`void upFirstLowRest(char str[]);`
Funkcja zamienia pierwszy znak stringa na uppercase, a resztę na lowercase.

`int getInt();`
Funkcja odpowiada za prawidłowe wczytywanie liczby typu int.
Return: Wczytany int lub -1, jeśli nie udało się go wczytać.

`int getIntForm(const char form[]);`
Funkcja wypisuje na ekran stringa *form*, a następnie woła funkcję *getInt*.
Return: Wynik funkcji *getInt*.

`char* strGen(int min, int max);`
Funkcja generuje string o losowej długości z zakresu [min, max] składający się z losowych małych liter.
Return: Wskaźnik do wygenerowanego stringa.

`char* numGen(int l);`
Funkcja generuje string o długości l składający się z losowych cyfr. Została użyta do generowania losowych numerów PESEL.
Return: Wskaźnik do wygenerowanego stringa.

`int intGen(int min, int max);`
Return: Losowa liczba z zakresu [min, max].

`int max(int a, int b);`
Return: Większa z liczb a i b.

`int min(int a, int b);`
Return: Mniejsza z liczb a i b.