**Вступ**

При розробці системи для Санаторію була використана уніфікована мова моделювання UML.

Мова UML є інструментом візуального моделювання систем, тому основним за­собом її використання вважаються діаграми. Кожна з діаграм віддзеркалює певний аспект системи, а вся множина діаграм - становить структурну основу проекту.

Уся подальша робота над проектом має здійснюватись на основі прецедентів: для кожного прецеденту формується опис його динаміки у вигляді серії діаграм взаємодії та діаграм діяльності.

З отриманих описів шляхом виявлення об'єктів, що задіяні в реалізації прецедентів, будуються діаграми класів.

Для визначення поведінки класів із складною динамікою реагування на події можуть формуватись діаграми станів.

Розміщення об'єктів по програмних модулях описується в компонентних діаграмах, а розміщення програмних модулів по вузлах комп'ютерам мережі – у діаграмах розгортання.

1. **Вибір життєвого цикл**

При розробці системи Санаторій для досягнення оптимального результату найбільш підходящою є модель життєвого циклу при якій була б можливість повернутися на певний попередній етап розробки при виникненні певних помилок, що дало б можливість швидкого їх виправлення (ітераційна модель). В результаті це б дало можливість як найкраще, і як найшвидше виконати поставлене завдання.

Дану модель можна представити в графічному вигляді таким чином:

Аналіз вимог

Проектування

Кодування

Тестування

Рис.1- Графічне зображення моделі життєвого циклу.

Хоча дана модель не є повністю досконалою, але вона повністю підходить для вирішення поставленої задачі.

**2. Аналіз аналогів**

Перед початком розробки системи Санаторій було досліджено існуючі аналоги, які використовуються в Санаторіях для оптимізації робочого процесу.

Проаналізувавши існуючі системні рішення було зроблено висновок, що на сьогоднішній день в Україні не існує такої системи, яка б на основі однієї програмної розробки виконувала б всі поставленні завдання. Так в Санаторії для задоволення потреб всіх користувачів використовується ціла низка програмного забезпечення, а в деяких випадках працівники змушені обходитись без нього. Таким чином кожен з них, для здійснення оптимальної роботи повинен розібратися як працює не одна, а одразу декілька програм, а це є досить проблематично.

Для ведення обліку депутатів у фракціях, більшості і комітетів, в яких депутати працюють використовуються різні системи управління базами даних (СУБД) і зокрема Microsoft Office Access, з пакету Microsoft Office.

Складання списку працівників комплексів і процедур в яких можуть відпочивати люди і досі виконуються спочатку без програмного забезпечення, лише після його повної розробки, він переноситься в електронний вигляд для розповсюдження. Для цього також використовується найрізноманітніше програмне забезпечення.

Для розроблення програмних реклам і візиток також використовується найрізноманітніше програмне забезпечення, починаючи від стандартних програм Windows, і закінчуючи Adobe Photoshop та іншими спеціалізованими програмами.

Для обміну даними з міжнародними організаціями, іншими санаторіями, також використовуються різноманітні програмні засоби для обробки електронної пошти.

Система, яка є розроблена під час виконання даного курсового проекту є оптимальною для організації роботи Санаторію, адже вона поєднує в одному модулі вирішення всіх поставлених задач і є достойною альтернативою для програмних систем, які зараз використовують працівники Санаторію.

**3. Моделювання системної архітектури засобами UML**

**3.1. Аналіз предметної галузі**

Під час розробки системи для роботи Санаторію спочатку було визначено всіх потенційних користувачів даної системи і визначено всі їх вимоги до даної системи. Таким чином серед прецедентів, які мають відношення до санаторію можна виділити:

* ***Відпочиваючий***
* ***Лікар***
* ***Медсестра***
* ***Кухар***
* ***Масажист***

Далі було проаналізовано всі їх вимоги від системи, які можна структуризувати наступним чином:



Рис.2- Ієрархія точок зору усіх користувачів системи.

Для графічного відображення вимог кожного користувача системи було використано уніфіковану мову моделювання UML. За допомогою якої було створено діаграми прецедентів, що і відображає їх вимоги.

а)

б)
Рис.3 – Діаграми прецедентів: пацієнта-відпочиваючого (а) і лікаря (б).

а)

б)

в)

Рис.4 – Діаграми прецедентів: Медсестри (А), кухаря (Б) і масажиста (В).

На першому періоді розробки системи було визначено місце системи в середовищі:

Система для роботи Санаторію

Інтернет

Користувачі

Підпрограма Організації екскурсій

Підпрограма формування документів

Підпрограма обробки електронної пошти

Підпрограма обробки баз даних

Підпрограма розкладу спортивних секцій

Рис.5. - Структурна схема моделі системного середовища.



Рис.6 - Модель процесу оформлення в санаторій.



Рис.7 – Процес проходження консультації в лікаря

При реалізації даної системи використовується ряд баз даних:

Кожна база даних зберігає відповідну інформацію, і містить поля за допомогою яких вона зв’язується з іншими базами даних, що забезпечує можливість реалізації системи.

 Для відображення сутності та зв’язків між базами даних можна використати наступну діаграму:



Рис.9. – Модель сутності баз даних, та зв’язків між ними.

**3.2. Розробка архітектури програмного забезпечення**

Наступним етапом в розробці системи є розробка архітектури програмного забезпечення. На цьому етапі визначається структуру системи, та структуру керування та взаємодії складових системи.

Для реалізації структури системи було використано одну із стандартних моделей – модель репозиторій. Яка забезпечує доступ до потрібної інформації кожному користувачу. Всі дані зберігаються на одному більш-менш потужному комп’ютері, що дозволяє централізувати роботу, і економити при виборі параметрів машин клієнтів, що здешевлює реалізацію системи.

Система для роботи Санаторію

Підпрограма Організації екскурсій

Підпрограма формування документів

Підпрограма обробки електронної пошти

Підпрограма обробки баз даних

Підпрограма розкладу спортивних секцій

 Рис.10. Структура системи для роботи музичної школи (модель репозиторій)



Рис.11 - Модель керування.

**4. Тестування програмного забезпечення**

**4.1. Теоретична частина**

Тестування - процес виконання програми з метою виявлення помилок. Кроки процесу задаються тестами. Кожен тест визначає:

* свій набір вихідних даних і умов для запуску програми;
* набір очікуваних результатів роботи програми.

Інша назва тесту - тестовий варіант. Повну перевірку програми гарантує вичерпне тестування. Воно вимагає перевірити всі набори вихідних даних, всі варіанти їх обробки і включає велику кількість тестових варіантів. На жаль, але вичерпне тестування в багатьох випадках залишається тільки мрією - спрацьовують ресурсні обмеження (перш за все, обмеження за часом).

Хорошим вважають тестовий варіант з високою ймовірністю виявлення ще не розкритою помилки. Успішним називають тест, який виявляє до цих пір не розкриту помилку.

Метою проектування тестових варіантів є систематичне виявлення різних класів помилок при мінімальних витратах часу і вартості. Важлива відповідь на питання: що може тестування? Тестування забезпечує:

* виявлення помилок;
* демонстрацію відповідності функцій програми її призначенням;
* демонстрацію реалізації вимог до характеристик програми;
* відображення надійності як індикатора якості програми.

А чого не може тестування? Тестування не може показати відсутність дефектів (воно може показувати тільки присутність дефектів).

Розглянемо інформаційні потоки процесу тестування. Вони показані на рис. 12.

 Рис.12- Інформаційні потоки процесу тестування.

На вході процесу тестування три потоки:

* + текст програми;
	+ вихідні дані для запуску програми;
	+ очікувані результати.

Виконуються тести, всі отримані результати оцінюються. Це означає, що реальні результати тестів порівнюються з очікуваними результатами. Коли виявляється розбіжність, фіксується помилка - починається налагодження. Процес налагодження непередбачуваний за часом. На пошук місця дефекту та виправлення до однієї години, день, місяць. Невизначеність у налагодженні призводить до великих труднощів в плануванні дій.

Після збирання та оцінювання результатів тестування починається відображення якості та надійності ПЗ. Якщо регулярно зустрічаються серйозні помилки, що вимагають проектних змін, то якість і надійність ПЗ підозрілими, констатується необхідність посилення тестування. З іншого боку, якщо функції ПО реалізовані правильно, а виявлені помилки легко виправляються, може бути зроблений один з двох висновків:

* + якість і надійність ПЗ задовільні;
	+ тести не здатні виявляти серйозні помилки.

У кінцевому рахунку, якщо тести не виявляють помилок, з'являється сумнів у тому, що тестові варіанти досить продумані і що в програмне забезпечення немає прихованих помилок. Такі помилки будуть, в кінцевому підсумку, виявлятися користувачами і коректуватися розробником на етапі супроводу (коли вартість виправлення зростає в 60-100 разів у порівнянні з етапом розробки).

Результати, накопичені в ході тестування, можуть оцінюватися й більш формальним способом. Для цього використовують моделі надійності ПЗ, що виконують прогноз надійності за реальними даними про інтенсивність помилок.

Існують 2 принципу тестування програми:

* + функціональне тестування (тестування «чорного ящика»);
	+ структурний тестування (тестування «білого ящика»).
	+ Тестування на відмову
	+ Перевірка на модель
	+ Та ін.

**4.2. Тестування "чорної скриньки"**

Тестування «чорного ящика»

Відомі: функції програми.

Досліджується: робота кожної функції на всій області визначення.

Як показано на рис. 13, основне місце програми тестів «чорного ящика» - інтерфейс ПЗ.



Рис. 13. Тестування «чорного ящика»

Ці тести демонструють:

* + як виконуються функції програм;
	+ як приймаються вихідні дані;
	+ як виробляються результати;
	+ як зберігається цілісність зовнішньої інформації.

При тестуванні «чорного ящика» розглядаються системні характеристики програм, ігнорується їх внутрішня логічна структура. Повне тестування, як правило, неможливо. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то буде потрібно 1010 тестових варіантів. Відзначимо також, що тестування «чорного ящика» не реагує на багато особливостей програмних помилок.

 

Як видно з зображень екрану, інтерфейс програми є простим, зручним і зрозумілим. Всі інформаційні повідомлення виводяться на українській мові. Організоване зручне і зрозуміле меню, яке дає доступ до 3-х реалізованих в програмі функцій, кожна функція має своє додаткове і зрозуміле підменю.

**4.3. Тестування "білої скриньки"**

Відома: внутрішня структура програми.

Досліджуються: внутрішні елементи програми і зв'язку між ними (рис. 14).



Рис. 14. Тестування «білого ящика»

Об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їх взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше - інформаційні зв'язки. Тестування за принципом «білого ящика» характеризується ступенем, за яким тести виконують або покривають логіку (вихідний текст) програми. Повне тестування провести важко. Особливості цього принципу тестування розглянемо окремо.

Для проведення даного тестування було обрано клас який зв’язується з базою даних.

Критерії тестування:

1. Наявність змістовних ”шапок” до файлів
2. Наявність змістовних ”шапок” до функцій
3. Відповідність імен функції Венгерській нотації
4. Відповідність імен змінних Венгерській нотації
5. Наявність пояснюючих коментарів до коду
6. Дотримання форматування тексту
7. Мобільність
8. Оптимальне функціонування окремих блоків програми.

В процесі тестування «білої скриньки» був використаний наступний код:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FILE..........: main.cpp

DESCRIPTION...: System of sanatoriy

FUNCTIONS.....: void input\_file(fstream\* file), void output\_file(fstream\* file), void show\_procedure();

SWITCHES......: WIN32 - if defined, 32-bit version is

compiled, otherwise 16-bit edition is compiled.

COPYRIGHT.....: Copyright (c) 2010.

HISTORY.......: DATE COMMENT

-------- --------------------------------------

03-17-10 Created

03-31-10 Modified

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*===============================[ PUBLIC DECLARATIONS ]========\*/

#include<iostream>

#include<cstdio>

#include<fstream>

#include<clocale>

#include <cstdlib>

#include <conio.h>

#include "main.h"

using namespace std;

int main()

{

 /\*===============================[ OBJECT DECLARATIONS ]========\*/

 setlocale(2,"");

 int choice;

 fstream\* file;

 file = new fstream("DB.txt",ios::app);

 file->close();

 bool bExit = true;

 while (bExit)

 {

 system("cls");

 cout <<"Система Санаторiй (1-3):"<< endl;

 cout <<"1. Вивести список вiдпочиваючих"<< endl;

 cout <<"2. Додати нового вiдпочиваючого"<< endl;

 cout <<"3. Список процедур" << endl;

 cout <<"Виберiть пункт,який ВАС зацiкавив(1-3): ";

 cin >>choice;

 cout<<endl;

 switch(choice){

 /\*===============================[ Choice ]===================\*/

 case 1:

 input\_file(file);

 break;

 case 2:

 output\_file(file);

 break;

 case 3:

 show\_procedure();

 system("pause");

 break;

 /\*===============================[ END Chice1 ]================\*/

 default:

 bExit = false;

 }

 }

 return 0;

}

Як видно з вище приведеного тексту, код оформлений за вимогами Венгерської нотації, присутня шапка початку файлу, шапка класу, процедур, присутні доцільні коментарі до тексту програми.

Функціонування усіх блоків є оптимальним і доцільним. Також всі блоки працюють правильно і збоїв в роботі окремо в’зятого блоку не виявлено.

**4.4. Перевірка на модель**

Метою даного виду тестування є перевірка на правильність виконання поставленої задачі за допомогою додаткової моделі, яка вже відлагоджена і працює правильно. Дане тестування має наступний структурний вигляд:



Рис. 15 – Схема процесу перевірки на модель.

Для проведення тестування в якості системи було використано задачу №1. Завданням якої є знайти суму наступного ряду sum=1/0!+1/1!...1/n!.

В якості моделі для перевірки було використано програму Wise Calculator, яка дозволяє вирішувати різноманітні математичні задачі.

В процесі тестування було здійснено перевірку для 15 різних n, і в 15 з 15 випадків результати виконання і системи і моделі були однаковими, тобто правильність виконання програми становить 100 %.

 

Рис. 16 – Результат виконання програми - системи.



Рис. 17 – Результат виконання програми - моделі.

**4.5. Тестування на відмову**

Для проведення тестування на відмову взята програма, яка була розроблена для курсової роботи. Повний текст даної програми наведений в додатку А.

Основними критеріями тестування було дослідження, поведінку як окремих задач програми так і модуль в цілому, при введені різноманітних вхідних даних.

Аналізуючи отриманні результати можна зробити наступний висновок: що як окремі підпрограми так і модуль в цілому є середньоспеціалізовані, тобто їх не так просто вивести з ладу, передбачені основні невірні дані. Програма видає повідомлення про помилку якщо така трапиться, а не просто закривається і якщо помилка не критична продовжує свою роботу.

**Висновок**

Під час виконання даного курсового проекту було досягнуто основну його мету, тобто було розроблено програмну системи Санаторій.

При проектуванні архітектури системи було використано мову UML. Перевага її в тому що за допомогою діаграм можна представити практично всі основні моделі, які потрібні для розробки системи – при цьому вони є досить зрозумілими для програміста або менеджера проекту. Хоча також є певні недоліки – нема жорсткої стандартизації при розробці певної системи. Це може призвести до того що різними особами окремі моменти на схемі можуть інтерпретуватися по різному. Також відсутність чіткої стандартизації ускладнює вивчення. Але попри все це використання даної мови все ж полегшує процес розробки.

При розробці системи головний акцент робився на автоматизацію процесу роботи Санаторію. Хоча до досконалості ще дуже далеко, оскільки потрібно проводити комплексні дослідження і спостереження, консультуватися у спеціалістів. Тому перед впровадженням потрібно провести детальне тестування.

В процесі виконання другої частини курсового проекту – проведення тестування програмного забезпечення було досліджено як окремі підсистеми так і модуль в цілому на відповідність їх поставлених вимогах. Для цього було розроблено ряд тестів різноманітного характеру. І було визначено, що дана система, яка була розроблена під час виконання курсового проекту є достатньо стабільною у використанні і є дуже спеціалізована і прив’язана до вирішення конкретно поставленого завдання. Хоча вона повністю задовольняє вимоги навчального процесу, адже вона написана відповідно до вимог Венгерської нотації і виконує поставленні завдання.

**Список літератури**

1. Орлов С.А., «Технологія розробки програмного забезпечення» Питер., 2002 – 322с
2. Фаулер М., «UML. Основы, 3-е издание / Пер. с англ.» - СПб.: Символ-плюс, 1999. - 192 с.
3. Дудзяний І.М., «Обєктно-орієнтоване моделювання програмних систем»- Львів, Видавничий центр ЛНУ ім. Івана Франка, 2007.
4. Коротун Т.М., «Моделі і методи тестування програмних систем», Київ, 2006.
5. Андон Ф.И. , Коваль Г.И. ,Коротун Т.М, Сус­лов В.Ю. «*Основы* инженерии качества программных систем» . – Киев: Академпериодика, 2002. – 504 с.
6. [http://www.uk.wikipedia.org](http://www.uk.wikipedia.org/)
7. http://msdn.microsoft.com/ru/

**Додаток А**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FILE..........: main.h

DESCRIPTION...: System of sanatoriy

FUNCTIONS.....: void input\_file(fstream\* file), void output\_file(fstream\* file), void show\_procedure();

SWITCHES......: WIN32 - if defined, 32-bit version is

compiled, otherwise 16-bit edition is compiled.

COPYRIGHT.....: Copyright (c) 2010.

HISTORY.......: DATE COMMENT

-------- --------------------------------------

03-17-10 Created

03-31-10 Modified

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <fstream>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FUNCTION......: input\_file

DESCRIPTION...: Input text of file

ATTRIBUTES....: Public

ARGUMENTS.....:

 file - pointer to file

RETURNS.......: void

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void input\_file(fstream\* file);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FUNCTION......: output\_file

DESCRIPTION...: Output text of file

ATTRIBUTES....: Public

ARGUMENTS.....:

 file - pointer to file

RETURNS.......: void

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void output\_file(fstream\* file);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FUNCTION......: show\_procedure

DESCRIPTION...: Output information

ATTRIBUTES....: Public

ARGUMENTS.....: none

RETURNS.......: void

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show\_procedure();

/\*\* (END OF FILE : main.h)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FILE..........: main.cpp

DESCRIPTION...: System of sanatoriy

FUNCTIONS.....: void input\_file(fstream\* file), void output\_file(fstream\* file), void show\_procedure();

SWITCHES......: WIN32 - if defined, 32-bit version is

compiled, otherwise 16-bit edition is compiled.

COPYRIGHT.....: Copyright (c) 2010.

HISTORY.......: DATE COMMENT

-------- --------------------------------------

03-17-10 Created

03-31-10 Modified

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*===============================[ PUBLIC DECLARATIONS ]========\*/

#include<iostream>

#include<cstdio>

#include<fstream>

#include<clocale>

#include <cstdlib>

#include <conio.h>

#include "main.h"

using namespace std;

int main()

{

 /\*===============================[ OBJECT DECLARATIONS ]========\*/

 setlocale(2,"");

 int choice;

 fstream\* file;

 file = new fstream("DB.txt",ios::app);

 file->close();

 bool bExit = true;

 while (bExit)

 {

 system("cls");

 cout <<"Система Санаторiй (1-3):"<< endl;

 cout <<"1. Вивести список вiдпочиваючих"<< endl;

 cout <<"2. Додати нового вiдпочиваючого"<< endl;

 cout <<"3. Список процедур" << endl;

 cout <<"Виберiть пункт,який ВАС зацiкавив(1-3): ";

 cin >>choice;

 cout<<endl;

 switch(choice){

 /\*===============================[ Choice ]===================\*/

 case 1:

 input\_file(file);

 break;

 case 2:

 output\_file(file);

 break;

 case 3:

 show\_procedure();

 system("pause");

 break;

 /\*===============================[ END Chice1 ]================\*/

 default:

 bExit = false;

 }

 }

 return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FUNCTION......: input\_file

DESCRIPTION...: Input text of file

ATTRIBUTES....: Public

ARGUMENTS.....:

 file - pointer to file

RETURNS.......: void

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void input\_file(fstream\* file)

{

 int i;

 char str[500];

 file = new fstream("DB.txt",ios::in);

 i=1;

 while (!file->eof())

 {

 file->getline(str,500);

 if (str[0]!=0)

 {

 cout << i << ". ";

 cout << str << endl;

 i++;

 }

 }

 file->close();

 system("pause");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FUNCTION......: output\_file

DESCRIPTION...: Output text of file

ATTRIBUTES....: Public

ARGUMENTS.....:

 file - pointer to file

RETURNS.......: void

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void output\_file(fstream\* file)

{

 char \*p;

 char str[500];

 char str1[100];

 char str2[100];

 char str3[100];

 char str4[100];

 cout << "Всi даннi вводити латинськими лiтерами" << endl;

 cout <<"Введiть данi про Вiдпочиваючого:\n\n";

 cout <<"Прiзвище: ";

 cin >>str;

 cout <<"Iм`я: ";

 cin >>str1;

 cout <<"Побатьковi: ";

 cin >>str2;

 cout <<"Номер кiмнати: ";

 cin >>str3;

 cout <<"Лiкар: ";

 cin >>str4;

 strcat(str," ");

 strcat(str,str1);

 strcat(str," ");

 strcat(str,str2);

 strcat(str," ");

 strcat(str,str3);

 strcat(str," ");

 strcat(str,str4);

 cout << endl;

 cout <<"Вiтаємо, Ви успiшно пройшли реєстрацiю" << endl;

 p=str;

 strcat(p,"\n");

 file = new fstream("DB.txt",ios::app);

 file->write(p, strlen(p));

 file->close();

 system("pause");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

FUNCTION......: show\_procedure

DESCRIPTION...: Output information

ATTRIBUTES....: Public

ARGUMENTS.....: none

RETURNS.......: void

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show\_procedure()

{

 cout << "Наш Санаторiй пропонує такi процедури:" << endl;

 cout << "\t- Масаж" << endl;

 cout << "\t- Лiкувальнi ванни" << endl;

 cout << "\t- Солянi печери" << endl;

 cout << "\t- Мiнеральнi води" << endl;

}

/\*\* (END OF FILE : main.cpp)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/