**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра ЕОМ**



**Курсова робота**

з предмету «Програмування, частина 2 (Об’єктно-орієнтоване програмування)»

на тему:

«**Базові принципи об’єктно-орієнтованого програмування**»

Завдання роботи: «Більярдна»

Виконав:

ст.гр. КІ-15

Нижньов М.Ю.

Прийняв:

Асистент

Козак Н.Б.

**Львів**

**2020**

**Анотація**

У цій курсовій роботі було виконано написання програмного забезпечення на мові програмування С++ з використанням основних принципів об’єктно-орієнтованого програмування, таких як одинарне та множинне наслідування, абстрагування, інкапсуляція. Програма являє собою спрощену модель більярдного клубу та включає у себе роботу з текстовими файлами, а саме: запис, зчитування, пошук, видалення та редагування вибраних даних, а також сортування на екрані за певною характеристикою.

**Зміст**

Завдання на курсову роботу……………………….………………..……………3

Вступ…………………………………………….….………………….…….…….4

1.Визначення поняття ООП (Об'єктно-орієнтоване програмування ), його основні принципи та обґрунтування вибору його технологій………….….......4

1.1.Огляд та обгрунтування вибору технологій об'єктно орієнтованого програмування………………………………………………………………….....4

1.2.Визначення ООП………….……………………………………………...…...5

1.2.1 Концепції………………………………………………………………….....8

1.2.2 Особливості реалізації…………………………………………………........9

1.2.3 Підходи до проектування програм в цілому………………………………11

1.3 Теорія про діаграми…………………………………………………………...12

2.1 Аналіз об’єктно-орієнтованої моделі об’єкту та алгоритмів його функціонування………………………………………………………………........15

2.1.1 Аналіз контексту проекту…………………………………………………...15

2.1.1. 1 Структурний аналіз предметної області…………………………………15

2.1.1. 2Аналіз функіонального наповнення проекту…………………………….16

2.1.1.3 Вибір інтерфейсу проекту із пунктів меню які він реалізовує……..…..16

2.1.2Аналіз об’єктно-орієнтованої моделі проекту……………………………..17

2.1.3 Аналіз алгоритмів функіонування проекту………………….…………….18

2.2 Програмна реалізація проекту…………………………………….…………..21

2.2.1 Програмна реалізація об’єктно-орієнтовної моделі проекту…………….21

2.2.2 Програмна реалізація алгоритмів функціонування проекту……….……..21

2.2.2.1 Опис реалізації основних методів…………………………………..........21

2.2.3 Програмна реалізація інтерфейсу проекту та його пунктів меню………24

2.2.3.1 Код, що реалізовує інтерфейс проекту та його пункти меню…………..24

3. Відлагодження………………………………………………………………….36

3.1 Відлагодження та тестування програми в цілому…………………………...36

Висновки……………………………………………………………………………39

Використана література…………………………………………………………..39

Додатки …………………………………………………………………………….40

Додаток А(Лістинг програмного коду проекту)…………………………………40

**Завдання на курсову роботу**

У даній курсовій роботі повинно бути реалізовано на мові С++ таку сферу розваг як більярдний клуб. У ній повинні бути використані основні принципи об’єктно-орієнтованого програмування.

**“Більярдна”**

**Вступ**

Об'єктно-орієнтоване програмування (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування та поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Однією з таких мов є С++. C++ (Сі-плюс-плюс) — мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом (англ. Bjarne Stroustrup) в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) 1979 року та початково отримала назву «Сі з класами». Згодом Страуструп перейменував мову на C++ у 1983 р. Базується на мові С. Вперше описана стандартом ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2014. У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відеоігри. С++ суттєво вплинула на інші, популярні сьогодні, мови програмування: С# та Java.

***Метою виконання*** цієї курсової роботи є вивчення об’єктно-орієнтованої мови С++ та практичне застосування здобутих моїх знань у цій сфері.[1]

**1.Визначення поняття ООП (Об'єктно-орієнтоване програмування ), його основні принципи та обґрунтування вибору його технологій**

**1.1.Огляд та обгрунтування вибору технологій об'єктно-орієнтованого програмування.**

C++ - універсальна мова програмування, яка дозволяє зробити програмування приємнішим для серйозного програміста. С++ було розроблено на основі мови програмування С. Крім можливостей, які дає С, С++ представляє свої гнучкі і ефективні засоби визначення нових типів, застосування яких значно полегшує та спрощує технологію програмування. Використовуючи визначення нових типів програміст може розділяти свою програму на простіші частини, які легше проконтролювати. Такий метод побудови програм часто називають абстракцією даних. Інформація про типи міститься в деяких обєктах типів, визначених користувачем. Все, чим ми маніпулюємо в програмі, може розглядатися як обєкт. Програмування із застосуванням таких обєктів називають обєктно-орієнтованим. В результаті раціонального використання цього методу одержуємо набагато коротші, доступніші для роззуміння та легше контрольовані прорами.

Ключевим поняттям C++ є клас. Клас - це структури, які містять не тільки обяви даних, а й функцій, що визначаються користувачем. Класи забезпечують приховування даних, гарантовану ініціалізацію даних, неявне перетворення типів для типів, визначених користувачем, динамічне задання типу, контрольоване користувачем управління памяттю і механізмом перегрузки операцій.

Основними поняттями обєктно-орієнтованого програмування є:

інкапсуляція;

спадковість;

абстракція;

поліморфізм.

Інкапсуляція полягає утому, що кожний клас має три рівні доступу до елементів:- рівень, дані на якому можуть бути використані будь-якою функцією;- елементи цього рівня можуть бути використані функціями-членами даного класу або функціями-членами похідних класів;елементи цього рівня можуть бути використані функціями-членами даного класу.

Дані рівні класів називають полями

Спадковість полягає у тому, що деякі класи можуть бути створені на основі інших, шляхом введення додаткових елементів.

При цьому ті класи, на основі яких створені інші, називаються базовими, а ті, які будуються на основі базових - похідними, які створюються шляхом простого копіювання всіх елементів базових класів, тому при описі похідних класів вказуються тільки ті елементи, які є додатковими до базового класу. Похідний клас може мати декілька базових класів, а базовий - декілька похідних.

Абстракція полягає у використанні віртуальних функкцій(функції, які обявляються в базовому класі, але в похідному класі можуть мати зовсім інше значення ). Віртуальна функція в базовому класі може не мати ніякого тіла, але в похідному повинна бути визначеною. Таким чином абстракція дозволяє, звертаючись до однієї і тієї ж функції в різних обєктах, виконувати різні дії.

Поліморфізм - можливість перевизначення різних функцій та операцій. Поліморфізм полягає в тому, що, по-перше, будь-яку операцію або функцію, яка визначена стандартним чином, можна перевизначити всередині класу так, що вона виконуватиме зовсім інші дії, по-друге, може існувати одночасно багато різних функцій у класі з однаковим імям, але з різною кількістю або типами параметрів.

Оголошення класів

С++ підтримує обяву класів, які інкапсулюють члени. Члени класа - члени даних і функцфії-члени. В членах-даних зберігаються значення, які представляють стан екземплярів класу(тобто обєктів які належать класу). Функції - члени встановлюють і перевіряють стан екземплярів класа, керуючи значеннями в членах даних. Загальний синтаксис обяви базового класу в С++ має такий вигляд:

Оголошення класу починається з ключового слова class, після якого слідує імя класу. Декларація членів класа заключається в пару фігурних дужок і закінчуються значком «крапка з комою».[2]

**1.2.Визначення ООП**

**Основні поняття**

**Абстрагування** - це спосіб виділити набір значущих характеристик об'єкта, виключаючи з розгляду незначущі. Відповідно, абстракція - це набір всіх таких характеристик.

[Інкапсуляція](http://znaimo.com.ua/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))

**Інкапсуляція** - це властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі і приховати деталі реалізації від користувача.

[Спадкування](http://znaimo.com.ua/%D0%A1%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))

**Спадкування** - це властивість системи, що дозволяє описати новий клас на основі вже існуючого з частково або повністю позичає функціональністю. Клас, від якого здійснюється спадкування, називається базовим, батьківським або суперкласом. Новий клас - нащадком, спадкоємцем або похідним класом.

**Поліморфізм** - це властивість системи використовувати об'єкти з однаковим інтерфейсом без інформації про тип і внутрішній структурі об'єкта.

**Клас** є описуваної на мові термінології ( [простору імен](http://znaimo.com.ua/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%96%D1%80_%D1%96%D0%BC%D0%B5%D0%BD)) вихідного коду моделлю ще не існуючої сутності (об'єкта). Фактично він описує пристрій об'єкта, будучи свого роду кресленням. Кажуть, що об'єкт - це **екземпляр** класу. При цьому в деяких виконуючих системах клас також може представлятися деяким об'єктом при виконанні програми за допомогою [динамічної ідентифікації типу даних](http://znaimo.com.ua/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B0_%D1%96%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F_%D1%82%D0%B8%D0%BF%D1%83_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). Зазвичай класи розробляють таким чином, щоб їх об'єкти відповідали об'єктам предметної області.

[Об'єкт](http://znaimo.com.ua/%D0%9E%D0%B1%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))

Сутність в [адресному просторі](http://znaimo.com.ua/%D0%90%D0%B4%D1%80%D0%B5%D1%81%D0%B0%D1%86%D1%96%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%96) обчислювальної системи, що з'являється при створенні екземпляра класу або копіювання прототипу (наприклад, після запуску результатів [компіляції](http://znaimo.com.ua/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) та [зв'язування](http://znaimo.com.ua/%D0%9B%D1%96%D0%BD%D0%BA%D0%BE%D0%B2%D0%BA%D0%B0) вихідного коду на виконання).

[Прототип](http://znaimo.com.ua/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%B8%D1%85_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

Прототип - це об'єкт-зразок, за образом і подобою якого створюються інші об'єкти. Об'єкти-копії можуть зберігати зв'язок з батьківським об'єктом, автоматично успадковуючи зміни в прототипі; ця особливість визначається в рамках конкретного [мови](http://znaimo.com.ua/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).

**Визначення ООП і його основні концепції**

У центрі ООП перебуває поняття об'єкта. Об'єкт - це сутність, якою можна посилати повідомлення, і яка може на них реагувати, використовуючи свої дані. Дані об'єкта приховані від решти програми. Приховування даних називається [інкапсуляцією](http://znaimo.com.ua/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

Наявність інкапсуляції достатньо для об'єктності мови програмування, але ще не означає його об'єктної орієнтованості - для цього потрібна наявність [спадкування](http://znaimo.com.ua/%D0%A1%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

Але навіть наявність інкапсуляції і спадкування не робить мову програмування в повній мірі об'єктним з точки зору ООП. Основні переваги ООП проявляються тільки в тому випадку, коли в мові програмування реалізований [поліморфізм](http://znaimo.com.ua/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)); тобто можливість об'єктів з однаковою специфікацією мати різну реалізацію.

Мова [Self](http://znaimo.com.ua/Self), дотримуючись багато вихідні положення об'єктно-орієнтованого програмування, ввів альтернативне класам поняття *прототипу,* поклавши початок [прототипних програмуванню](http://znaimo.com.ua/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%B8%D1%85_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), считающемуся підвидом об'єктного.  
Складності визначення

ООП має вже більш ніж сорокарічну історію, але, незважаючи на це, досі не існує чіткого загальноприйнятого визначення даної технології [[1]](http://znaimo.com.ua/%D0%9E%D0%B1_%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F#link20). Основні принципи, закладені в перші об'єктні мови і системи, піддалися суттєвого зміни (або спотворення) і доповненню при численних реалізаціях наступного часу. Крім того, приблизно з середини 1980-х років термін "об'єктно-орієнтований" став [модним](http://znaimo.com.ua/%D0%9C%D0%BE%D0%B4%D0%B0), в результаті з ним сталося те саме, що дещо раніше з терміном "структурний" (що став модним після поширення технології [структурного програмування](http://znaimo.com.ua/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) - його стали штучно "прикріплювати" до будь-яких нових розробок, щоб забезпечити їм привабливість. [Бьерн Страуструп](http://znaimo.com.ua/%D0%91%D1%8C%D0%B5%D1%80%D0%BD_%D0%A1%D1%82%D1%80%D0%B0%D1%83%D1%81%D1%82%D1%80%D1%83%D0%BF) в 1988 році писав, що обгрунтування "об'єктної орієнтованості" чого-небудь, у більшості випадків, зводиться до помилкового [силлогизму](http://znaimo.com.ua/%D0%A1%D0%B8%D0%BB%D0%BE%D0%B3%D1%96%D0%B7%D0%BC) : "X - це добре. Об'єктна орієнтованість - це добре. *Отже,* X є об'єктно-орієнтованим".

Тімоті Бадд пише :

Роджер Кінг аргументовано наполягав, що його [кіт](http://znaimo.com.ua/%D0%9A%D1%96%D1%88%D0%BA%D0%B0) є об'єктно-орієнтованим. Крім інших своїх достоїнств, кіт демонструє характерне поведінка, реагує на повідомлення, наділений успадкованими реакціями і управляє своїм, цілком незалежним, внутрішнім станом.

На думку [Алана Кея](http://znaimo.com.ua/%D0%90%D0%BB%D0%B0%D0%BD_%D0%9A%D0%B5%D0%B9), творця мови [Smalltalk](http://znaimo.com.ua/Smalltalk), якого вважають одним із "батьків-засновників" ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів (цитується по вищезгаданій книзі Т. Бадда).

1. Все є об'єктом.
2. Обчислення здійснюються шляхом взаємодії (обміну даними) між об'єктами, при якому один об'єкт вимагає, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, посилаючи й одержуючи повідомлення. Повідомлення - це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії.
3. Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
4. Кожен об'єкт є представником класу, який висловлює загальні властивості об'єктів (таких, як цілі числа або списки).
5. У класі задається поведінка (функціональність) об'єкта. Тим самим всі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж дії.
6. Класи організовані в єдину деревовидну структуру із загальним коренем, звану ієрархією спадкоємства. Пам'ять і поведінка, пов'язана з екземплярами певного класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Таким чином, програма являє собою набір об'єктів, що мають стан і поведінку. Об'єкти взаємодіють за допомогою повідомлень. Природним чином вибудовується ієрархія об'єктів: програма в цілому - це об'єкт, для виконання своїх функцій вона звертається до входять до неї об'єктів, які, в свою чергу, виконують запитане шляхом звернення до інших об'єктів програми. Природно, щоб уникнути нескінченної рекурсії у зверненнях, на якомусь етапі об'єкт трансформує звернене до нього повідомлення в повідомлення до стандартних системних об'єктів, що надаються мовою і середовищем програмування.

Стійкість і керованість системи забезпечується за рахунок чіткого поділу відповідальності об'єктів (за кожну дію відповідає певний об'єкт), однозначного визначення інтерфейсів межоб'ектного взаємодії та повної ізольованості внутрішньої структури об'єкта від зовнішнього середовища (інкапсуляції).

**Визначити ООП можна і багатьма іншими способами**.

**1.2.1 Концепції**

Поява в ООП окремого поняття **класу** закономірно випливає з бажання мати безліч об'єктів з подібним поведінкою. Клас в ООП - це в чистому вигляді [абстрактний тип даних](http://znaimo.com.ua/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), створюваний програмістом. З цієї точки зору об'єкти є значеннями даного абстрактного типу, а визначення класу задає внутрішню структуру значень і набір операцій, які над цими значеннями можуть бути виконані. Бажаність ієрархії класів (а значить, спадкування) випливає з вимог до повторного використання коду - якщо кілька класів мають подібну поведінку, немає сенсу дублювати їх опис, краще виділити загальну частину в загальний батьківський клас, а в описі самих цих класів залишити тільки розрізняються елементи.

Необхідність спільного використання об'єктів різних класів, здатних обробляти однотипні повідомлення, вимагає підтримки **поліморфізму** - можливості записувати різні об'єкти в змінні одного і того ж типу. У таких умовах об'єкт, відправляючи повідомлення, може не знати в точності, до якого класу належить адресат, і одні й ті ж повідомлення, відправлені змінним одного типу, що містить об'єкти різних класів, викличуть різну реакцію.

Окремого пояснення вимагає поняття обміну **повідомленнями**. Спочатку (наприклад, в тому ж Smalltalk) взаємодія об'єктів уявлялося як "справжній" обмін повідомленнями, тобто пересилання від одного об'єкта іншому спеціального об'єкта-повідомлення. Така модель є надзвичайно загальною. Вона прекрасно підходить, наприклад, для опису паралельних обчислень за допомогою *активних об'єктів,* кожен з яких має власний потік виконання і працює одночасно з іншими. Такі об'єкти можуть вести себе як окремі, абсолютно автономні обчислювальні одиниці. Посилка повідомлень природним чином вирішує питання обробки повідомлень об'єктами, присвоєними поліморфним змінним - незалежно від того, як оголошується змінна, повідомлення обробляє код класу, до якого належить присвоєний змінній об'єкт.

Однак спільність механізму обміну повідомленнями має й інший бік - "повноцінна" передача повідомлень вимагає додаткових накладних витрат, що не завжди прийнятно. Тому в більшості нині існуючих об'єктно-орієнтованих мов програмування використовується концепція **"відправка повідомлення як виклик методу"** - об'єкти мають доступні ззовні методи, викликами яких і забезпечується взаємодія об'єктів. Даний підхід реалізований у величезній кількості мов програмування, в тому числі [C + +](http://znaimo.com.ua/C), [Object Pascal](http://znaimo.com.ua/Object_Pascal), [Java](http://znaimo.com.ua/Java), [Oberon-2](http://znaimo.com.ua/Oberon-2). На даний момент саме він є найбільш поширеним в об'єктно-орієнтованих мовах.

Концепція **віртуальних методів,** підтримувана цими та іншими сучасними мовами, з'явилася як засіб забезпечити виконання потрібних методів при використанні поліморфних змінних, тобто, по суті, як спроба розширити можливості виклику методів для реалізації частини функціональності, забезпечувана механізмом обробки повідомлень.

**1.2.2 Особливості реалізації**

Як вже говорилося вище, в сучасних об'єктно-орієнтованих мовах програмування кожен об'єкт є значенням, що належать до певного [класу](http://znaimo.com.ua/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Клас являє собою оголошений програмістом складовою [тип даних](http://znaimo.com.ua/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), який має в складі:

Поля даних

Параметри об'єкту (звичайно, не всі, а тільки необхідні в програмі), що задають його стан (властивості об'єкта предметної області). Іноді поля даних об'єкта називають властивостями об'єкта, через що можлива плутанина. Фізично поля представляють собою значення (змінні, константи), оголошені як належать класу.

[Методи](http://znaimo.com.ua/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BC%D0%BE%D0%B2%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))

Процедури і функції, пов'язані з класом. Вони визначають дії, які можна виконувати над об'єктом такого типу, і які сам об'єкт може виконувати.

Класи можуть успадковуватися один від одного. Клас-нащадок отримує всі поля та методи класу-батька, але може доповнювати їх власними або перевизначати вже наявні. Більшість мов програмування підтримує тільки одиничне успадкування (клас може мати тільки один клас-батько), лише в деяких допускається множинне спадкування - породження класу від двох або більше класів-батьків. Множинне успадкування створює цілий ряд проблем, як логічних, так і чисто реалізаційних, тому в повному обсязі його підтримка не поширена. Замість цього в 1990-і роки з'явилося і стало активно вводитися в об'єктно-орієнтовані мови поняття [інтерфейсу](http://znaimo.com.ua/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Інтерфейс - це клас без полів і без реалізації, що включає тільки заголовки методів. Якщо якийсь клас успадковує (або, як кажуть, реалізує) інтерфейс, він повинен реалізувати всі вхідні в нього методи. Використання інтерфейсів надає відносно дешеву альтернативу множинного спадкоємства.

Взаємодія об'єктів в абсолютній більшості випадків забезпечується викликом ними методів один одного.

**Інкапсуляція забезпечується наступними засобами**

Контроль доступу

Оскільки методи класу можуть бути як суто внутрішніми, що забезпечують логіку функціонування об'єкта, так і зовнішніми, за допомогою яких взаємодіють об'єкти, необхідно забезпечити прихованість перших при доступності ззовні других. Для цього в язики вводяться спеціальні синтаксичні конструкції, явно задають область видимості кожного члена класу. Традиційно це модифікатори public, protected і private, що позначають, відповідно, відкриті члени класу, члени класу, доступні тільки з класів-нащадків і приховані, доступні тільки усередині класу. Конкретна номенклатура модифікаторів і їх точний зміст розрізняються в різних мовах.

Методи доступу

Поля класу, в загальному випадку, не повинні бути доступні ззовні, оскільки такий доступ дозволив би довільним чином змінювати внутрішній стан об'єктів. Тому поля зазвичай оголошуються прихованими (або мову в принципі не дозволяє звертатися до полів класу ззовні), а для доступу до знаходяться в полях даними використовуються спеціальні методи, звані методами доступу. Такі методи або повертають значення того чи іншого поля, або роблять запис в це поле нового значення. При записі метод доступу може проконтролювати допустимість записуваного значення і, при необхідності, провести інші маніпуляції з даними об'єкта, щоб вони залишилися коректними (внутрішньо погодженими). Методи доступу називають ще аксессор (від [англ.](http://znaimo.com.ua/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *access* - Доступ), а окремо - [Геттер](http://znaimo.com.ua/%D0%93%D0%B5%D1%82%D1%82%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) ( [англ.](http://znaimo.com.ua/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *get* - Читання) та [сетера](http://znaimo.com.ua/Setter) ( [англ.](http://znaimo.com.ua/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *set* - Запис) [[5]](http://znaimo.com.ua/%D0%9E%D0%B1_%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F#link24).

[Властивості об'єкта](http://znaimo.com.ua/%D0%92%D0%BB%D0%B0%D1%81%D1%82%D0%B8%D0%B2%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))

Псевдополя, доступні для читання і / або запису. Властивості зовні виглядають як поля і використовуються аналогічно доступним полях (з деякими виключеннями), однак фактично при зверненні до них відбувається виклик методів доступу. Таким чином, властивості можна розглядати як "розумні" поля даних, що супроводжують доступ до внутрішніх даних об'єкта-якими додатковими діями (наприклад, коли зміна координати об'єкта супроводжується його перемальовуванням на новому місці). Властивості, по суті - не більше ніж **синтаксичний цукор**, оскільки жодних нових можливостей вони не додають, а лише приховують виклик методів доступу. Конкретна мовна реалізація властивостей може бути різною. Наприклад, в [C #](http://znaimo.com.ua/C%20Sharp) оголошення властивості безпосередньо містить код методів доступу, який викликається тільки при роботі з властивостями, тобто не вимагає окремих методів доступу, доступних для безпосереднього виклику. У Delphi оголошення властивості містить лише імена методів доступу, які повинні викликатися при зверненні до поля. Самі методи доступу являють собою звичайні методи з деякими додатковими вимогами до сигнатурі.

Поліморфізм реалізується шляхом введення в мову правил, згідно з якими змінної типу "клас" може бути присвоєний об'єкт будь-якого класу-нащадка її класу.

**1.2.3 Підходи до проектування програм в цілому**

ООП орієнтоване на розробку великих програмних комплексів, що розробляються командою програмістів (можливо, досить великий). Проектування системи в цілому, створення окремих компонент і їх об'єднання в кінцевий продукт при цьому часто виконується різними людьми, і немає жодного фахівця, який знав би про проект все.

Об'єктно-орієнтоване проектування полягає в описі структури та поведінки проектованої системи, тобто, фактично, у відповіді на два основних питання:

* **З яких частин складається система.**
* **У чому полягає відповідальність кожної з частин.**

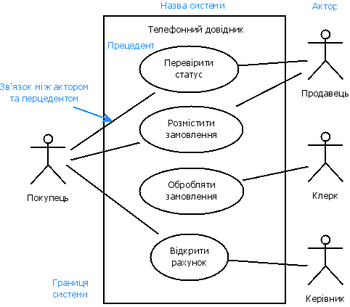
Виділення частин проводиться таким чином, щоб кожна мала мінімальний за обсягом і точно певний набір виконуваних функцій (обов'язків), і при цьому взаємодіяла з іншими частинами якомога менше.

Подальше уточнення призводить до виділення більш дрібних фрагментів опису. У міру деталізації опису та визначення відповідальності виявляються дані, які необхідно зберігати, наявність близьких по поведінці агентів, які стають кандидатами на реалізацію у вигляді класів з загальними предками. Після виділення компонентів і визначення інтерфейсів між ними реалізація кожного компонента може проводитися практично незалежно від інших (зрозуміло, при дотриманні відповідної технологічної дисципліни).

Велике значення має правильна побудова ієрархії класів. Одна з відомих проблем великих систем, побудованих по ООП-технології - так звана *проблема крихкості базового класу.* Вона полягає в тому, що на пізніх етапах розробки, коли ієрархія класів побудована і на її основі розроблено велику кількість коду, виявляється важко або навіть неможливо внести якісь зміни в код базових класів ієрархії (від яких породжені всі або багато працюють в системі класи ). Навіть якщо вносяться зміни не торкнуться інтерфейс базового класу, зміна його поведінки може непередбачуваним чином відбитися на класах-нащадках. У разі великої системи розробник базового класу просто не в змозі передбачити наслідки змін, він навіть не знає про те, як саме базовий клас використовується і від яких особливостей його поведінки залежить коректність роботи класів-нащадків. [3]

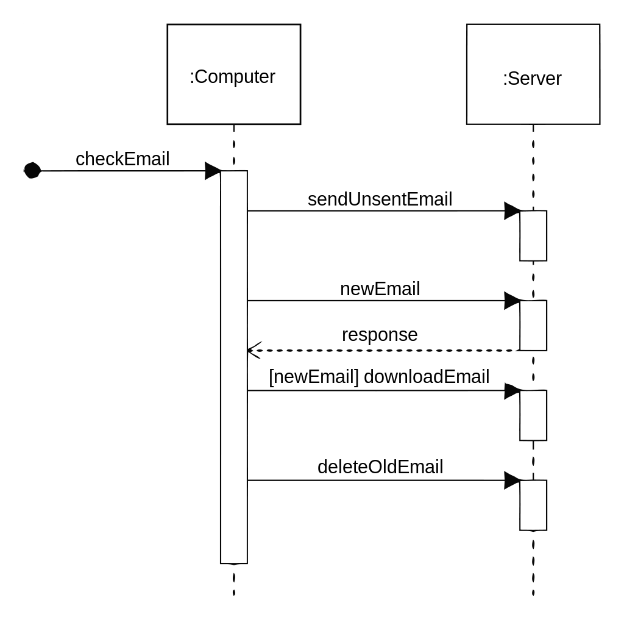
**1.3 Теорія про діаграми**

**Діаграма прецедентів** — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання. Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами.



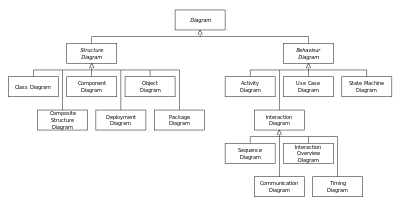
Приклад use case diagram

**Діаграма послідовності** (англ. sequence diagram) — різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення. Визначені стандартом UML 2.0 діаграми послідовностей мають ті ж можливості що і визначені стандартом UML 1.x, та підтримують додаткові можливості зміни стандартного порядку повідомлень.



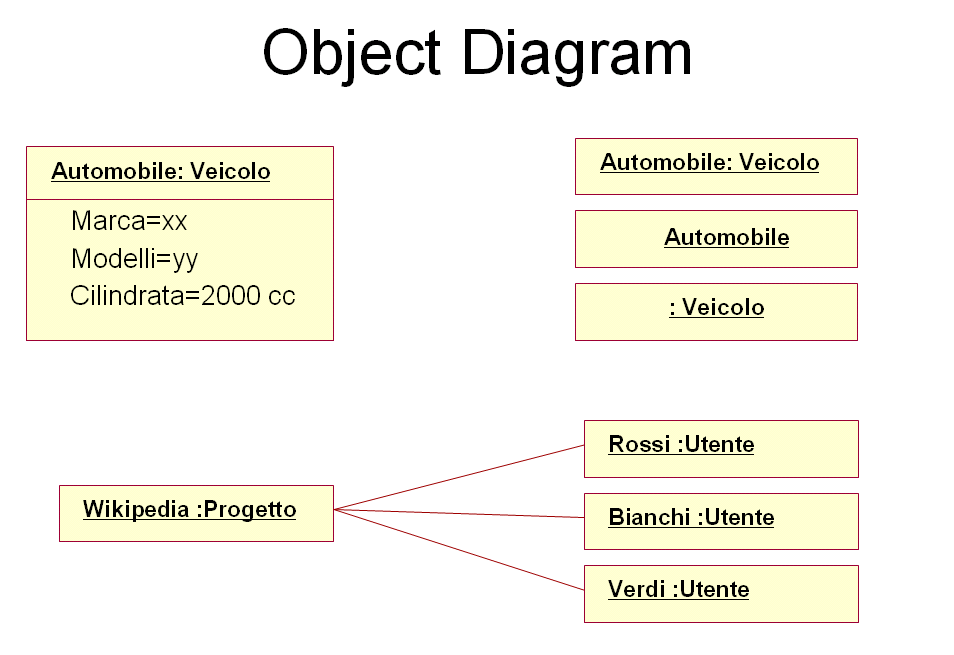
Приклад sequence diagram

**Діагра́ма кла́сів** (англ. class diagram) — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.



Приклад class diagram

**Діаграма об'єктів —** в UML, діаграма, що відображає об'єкти та їх зв'язки в певний момент часу. Діаграма об'єктів може розглядатись як окремий випадок діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (англ. collaboration diagram). Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.



Приклад object diagram

**2.Проектування**

**2.1.Аналіз об’єктно-орієнтованої моделі об’єкту та алгоритмів його функціонування**

**2.1.1 Аналіз контексту проекту**

**2.1.1.** **1**

**Структурний аналіз предметної області**

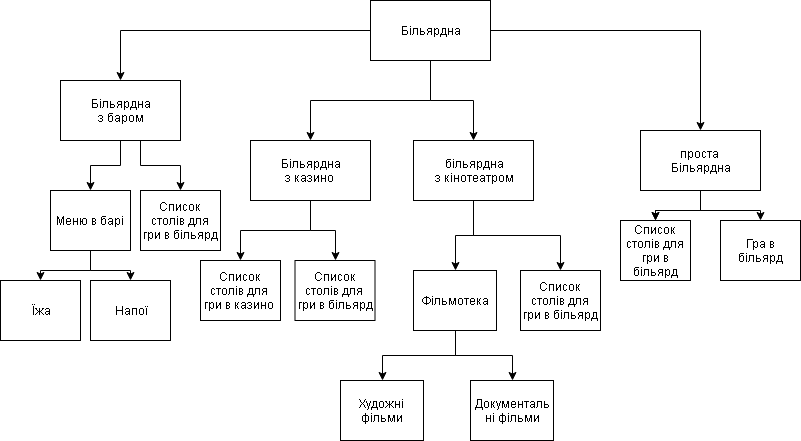


Рис.1. Вигляд предметної галузі

**2.1.1.** **2Аналіз функіонального наповнення проекту**

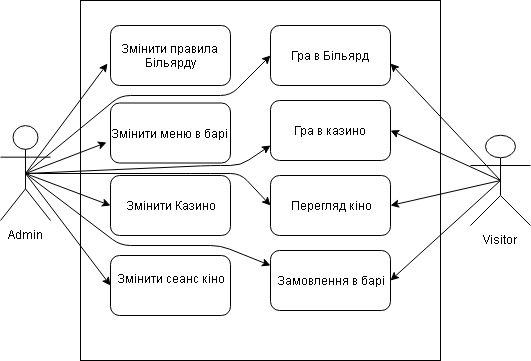
****

Рис. 2. Діаграма прецедентів

**2.1.1.3 Вибір інтерфейсу проекту із пунктів меню які він реалізовує**

Відкривши программу, нам пропонують зробити вибір який складається із 3 пунктів:

А)Режим Адміністратора

Б)Режим Користувача

В)Вихід із програми

Вибравши пункт “Режим Адміністратора” нам запропонують наступні 4 опції:

А)Змінити властивості більярду

Б)Змінити властивості казино

В)Змінити властивості кінотеатру

Г)Змінити властивості бару

Д)Вихід

Для кожної зміни, обов’язково треба ввести пароль, який знає тільки адміністратор.

Вибравши пункт “Режим Користувача” нам запропонують 5 пунктів меню,а саме:

А)Звичайний більярд

Б)Більярд з казино

В)Більярд з баром

Г)Більярд з кінотеатром

Д)Вихід

При виборі будь якого з пунктів меню, нам запропонують різні види більярду,а після,вибір доступного столу ,а в пунктах Б,В,Г різні додаткові послуги.

https://docs.google.com/drawings/d/sG-B9GFxqEKqrH1yesBJxGA/image?w=313&h=43&rev=1&ac=1

**2.1.2Аналіз об’єктно-орієнтованої моделі проекту**

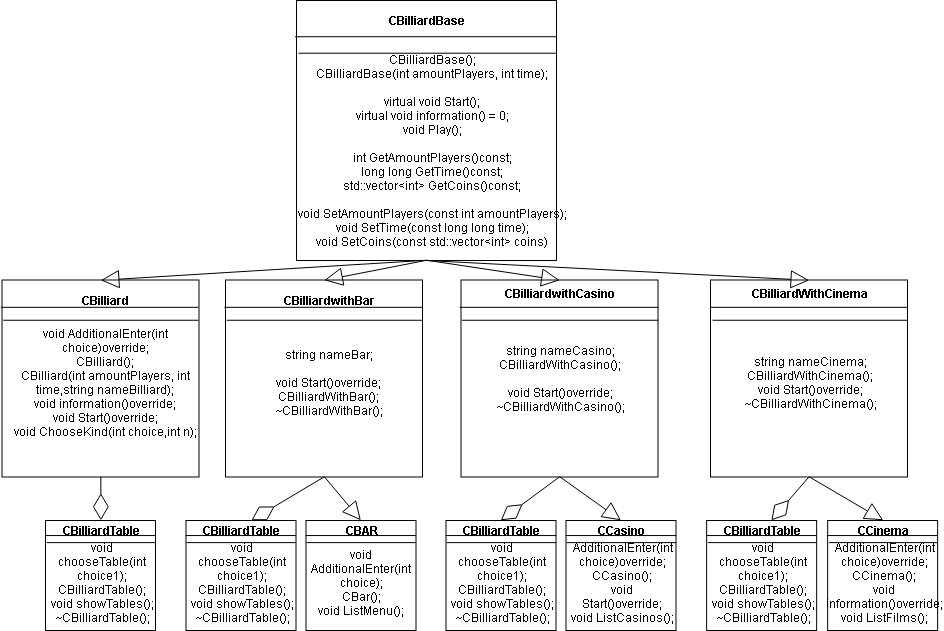
****

Рис. 3. Діаграма класів

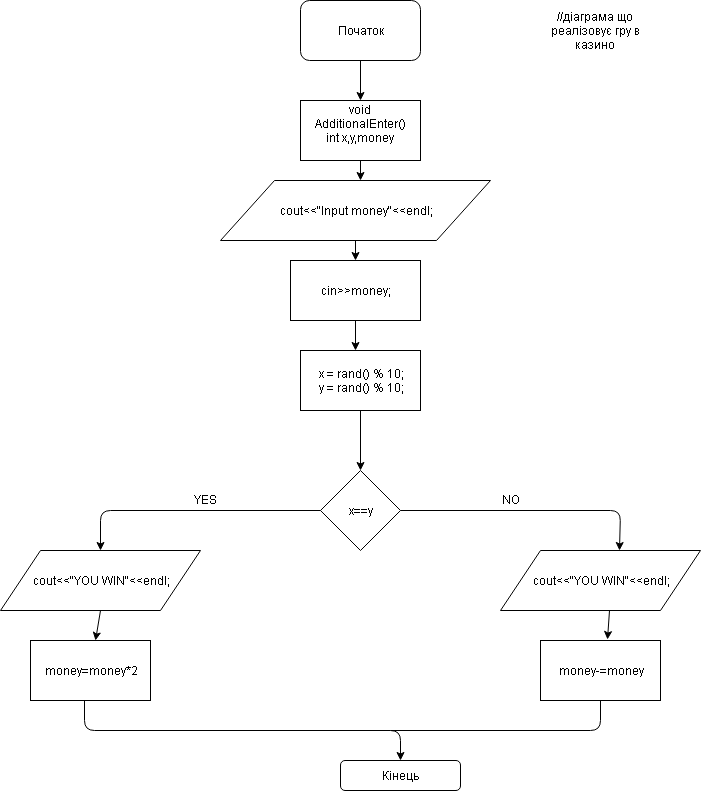
**2.1.3 Аналіз алгоритмів функіонування проекту**

Рис. 4. Блок-схема алгоритму що реалізовує гру в казино

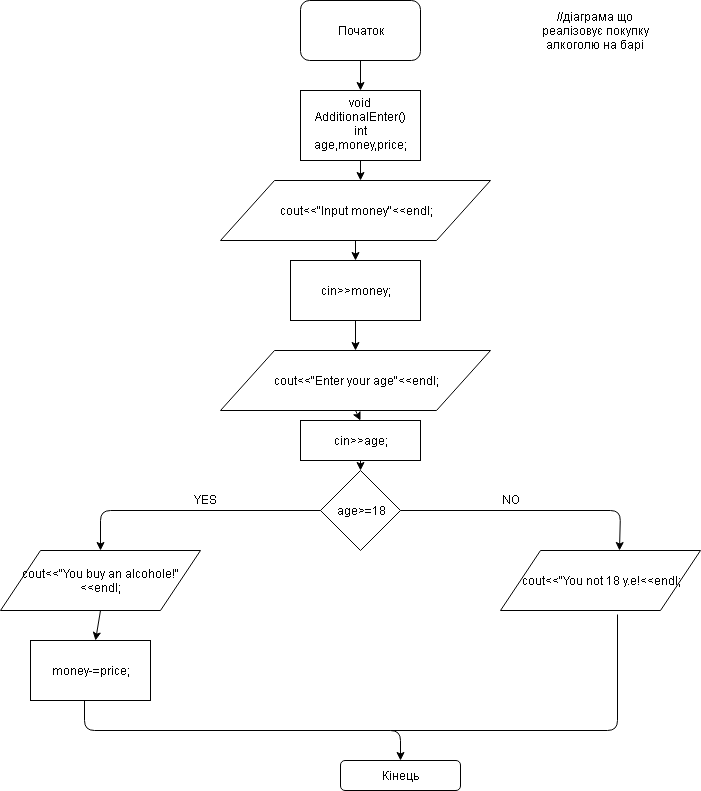
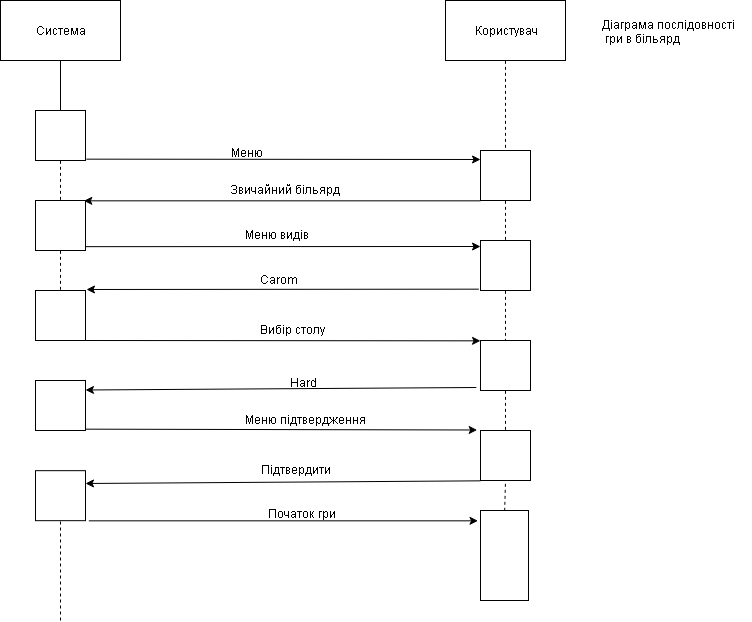
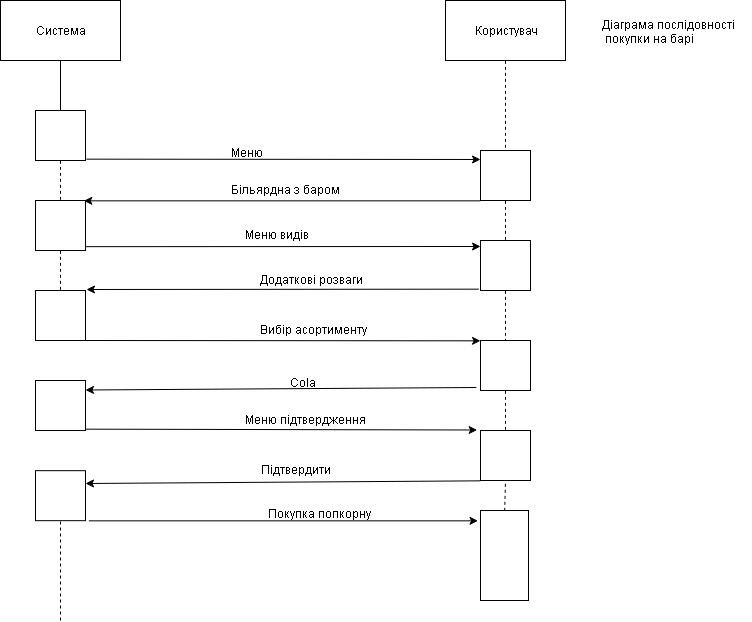


Рис. 5. Блок-схема алгоритму що реалізовує покупку алкоголю на барі

 Рис. 6. Діаграма послідовності гри в більярд

 Рис. 7. Діаграма послідовності покупки в барі

**2.2 Програмна реалізація проекту**

**2.2.1 Програмна реалізація об’єктно-орієнтовної моделі проекту**

Перша гілка класів – Cbilliard,CBilliardWithBar,CBilliardWithCasino,CBilliardWithCinema наслідуються від **CBilliardBase**

Друга гілка класів - Клас CBilliardTable наслідується від CBilliard.

Третя гілка класів – Клас CBar,CBilliardTable наслідуються від CBilliardWithBar.

Четверта гілка класів – Клас CCasino,CBilliardTable наслідуються від CBilliardWithCasino.

П’ята гілка класів – Клас CCinema,CBilliardTable наслідуються від CBilliardWithCinema.

**2.2.2 Програмна реалізація алгоритмів функціонування проекту**

**2.2.2.1 Опис реалізації основних методів.**

А)Метод гри в більярд

void CBilliardBase::Play() {

srand(std::time(NULL));

long long time = 0;

while (time < this->time) {

for (int i = 0; i < amountPlayers; ++i) {

int result = rand() % 2;

coins[i] += result;

if (result == 1) {

cout << i + 1 << " player scores in the hole(" << coins[i] << ")" << endl;

}

}

time += rand() % (this->time / 2);

if (time > this->time) {

time = this->time;

}

this\_thread::sleep\_for(chrono::milliseconds(1000));

cout << "Playing time: " << time << endl;

}

cout << "Scoreboard:" << endl;

for (int i = 0; i < amountPlayers; ++i) {

cout << i + 1 << " player has " << coins[i] << " score(s)" << endl;

}

}

Б)Метод гри в казино

void CCasino::AdditionalEnter(int choice) {

srand(std::time(NULL));

int money1;

while (true) {

if (listCasinos.empty()) {

return;

}

if (choice > 0 && choice <= listCasinos.size()) {

cout << "Input money:";

cin >> money1;

int x, y;

x = rand() % 10;

y = rand() % 10;

auto it = next(listCasinos.begin(), choice - 1);

cout << "You play " << \*it << ".";

if (x == y) {

cout << " Congratulation! You win " << money1 \* 2 << endl;

}

else {

cout << " You lose your money!" << endl;

}

break;

}

}

}

В)Метод перегляду фільму

void CCinema::AdditionalEnter(int choice) {

while (true) {

ListFilms();

if (listFilms.empty()) {

return;

}

if (choice > 0 && choice <= listFilms.size()) {

auto it = next(listFilms.begin(), choice - 1);

cout << "Watching film " << it->first << " with duration " << it->second << "(min.)" << " ..." << endl;

this\_thread::sleep\_for(chrono::milliseconds(5125));

cout << "Film ends. Leaving the cinema..." << endl;

break;

}

}

}

Г)Метод покупки в барі

void CBar::AdditionalEnter(int choice) {

while (true) {

if (menu.empty()) {

return;

}

if (choice > 0 && choice <= menu.size()) {

auto it = next(menu.begin(), choice - 1);

cout << "Eating/drinking " << \*it << "..." << endl;

this\_thread::sleep\_for(chrono::milliseconds(3125));

cout << "Leaving the bar..." << endl;

break;

}

}

}

Д)Метод вибору столу

void CBilliardTable::chooseTable(int choice1)

{

auto it = next(listtables.begin(), choice1);

cout << "You choose " << \*it << " Table" << endl;

cout << "Standard:" << endl;

}

Е)Методи показу та вибору видів більярду

void CBilliardBase::ShowKind() {

if (listKindPools.empty()) {

cerr << "There arent any kind of pools!Please add them!" << endl;

return;

}

int count = 1;

for (string kind : listKindPools) {

cout << count++ << " - " << kind << endl;

}

}

void CBilliardBase::ChooseKind(int choice,int n) {

char y;

auto it = next(listKindPools.begin(), choice);

cout << "You choose " << \*it << " mode!" << endl;

cout << "Standard:" << endl;

cout << "Amount players:" << amountPlayers << endl;

cout << "Playing time(sec.):" << time << endl;

if (n == 1 ){

Play();

}

else if (n==0){

int amountPlayers;

long long time;

cout << "Input amount players: ";

cin >> amountPlayers;

cout << "Input playing time(sec.): ";

cin >> time;

SetTime(time);

SetAmountPlayers(amountPlayers);

Play();

}

}

**2.2.3 Програмна реалізація інтерфейсу проекту та його пунктів меню**

**2.2.3.1 Код, що реалізовує інтерфейс проекту та його пункти меню**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

//#include "stdafx.h"

// from HKEY\_USERS\.DEFAULT\Keyboard Layout\Preload

#define ENG\_KEYBOARD\_LAYOUT "00000409"

#define UKR\_KEYBOARD\_LAYOUT "00000422"

#define RUS\_KEYBOARD\_LAYOUT "00000419"

#include "Admin.h"

#include "CHuman.h"

#include "CBilliardTable.h"

#include "CCinema.h"

#include "stdio.h"

#include "CCasino.h"

#include <stdlib.h>

#include <wchar.h>

#include <windows.h>

#include "CBilliardWithCinema.h"

#include "CBar.h"

#include "CBilliardWithBar.h"

#include "CBilliardWithCasino.h"

#include "CBilliard.h"

#include "CBilliardBase.h"

//#define DISABLE\_DEFAULT\_ACTION

#define MAX\_MENU\_SPACE 100

#define MAX\_LENGTH\_MENU\_NAME 100

#define MAX\_MENU\_ITEMS\_COUNT 100

void goToXY(int x, int y)

{

COORD p = { x, y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), p);

}

void clearGoToXY(int x, int y) {

COORD topLeft = { x, y };

HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_SCREEN\_BUFFER\_INFO screen;

DWORD written;

GetConsoleScreenBufferInfo(console, &screen);

FillConsoleOutputCharacterA(

console, ' ', screen.dwSize.X \* screen.dwSize.Y, topLeft, &written

);

FillConsoleOutputAttribute(

console, FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE,

screen.dwSize.X \* screen.dwSize.Y, topLeft, &written

);

SetConsoleCursorPosition(console, topLeft);

}

#define KEY\_UP 72

#define KEY\_DOWN 80

#define KEY\_LEFT 75

#define KEY\_RIGHT 77

#define KEY\_ENTER 13

#define KEY\_BACKSPACE 8

#define KEY\_ESCAPE 27

unsigned int runSpaceOp(int x, int y, int kolommen, char spaceOp[MAX\_MENU\_SPACE][MAX\_MENU\_SPACE], int rijen = MAX\_MENU\_SPACE)

{

int index = 0, indey = 0, keuze, lol = MAX\_MENU\_SPACE;

do {

goToXY(index + x, indey + y);

keuze = 0;

keuze = \_getch();

if (keuze == 0 || keuze == 0xE0 || keuze == 224)

{

keuze = \_getch();

goToXY(index + x, indey + y);

switch (keuze)

{

case KEY\_UP:indey--;

break;

case KEY\_LEFT: index--;

break;

case KEY\_DOWN:indey++;

break;

case KEY\_RIGHT: index++;

break;

}

indey = (indey <= 0) ? 0 : (indey > kolommen) ? kolommen : indey;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == 32 || (keuze >= 46 && keuze <= 57) || (keuze <= 64 && keuze >= 126))

{

spaceOp[index][indey] = (char)keuze;

printf("%c", spaceOp[index][indey]);

index++;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == KEY\_ENTER)

{

return indey;

}

} while (keuze != KEY\_ESCAPE);

return -1; //

}

void runTextOp(int x, int y, int kolommen, char spaceOp[MAX\_MENU\_SPACE][MAX\_MENU\_SPACE], int rijen = MAX\_MENU\_SPACE)

{

int index = 0, indey = 0, keuze, lol = MAX\_MENU\_SPACE;

do {

goToXY(index + x, indey + y);

keuze = 0;

keuze = \_getch();

if (keuze == 0 || keuze == 0xE0 || keuze == 224)

{

keuze = \_getch();

goToXY(index + x, indey + y);

switch (keuze)

{

case KEY\_UP:indey--;

break;

case KEY\_LEFT: index--;

break;

case KEY\_DOWN:indey++;

break;

case KEY\_RIGHT: index++;

break;

}

indey = (indey <= 0) ? 0 : (indey > kolommen) ? kolommen : indey;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == 32 || (keuze >= 46 && keuze <= 57) || (keuze <= 64 && keuze >= 126))

{

spaceOp[index][indey] = (char)keuze;

printf("%c", spaceOp[index][indey]);

index++;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == KEY\_BACKSPACE)

{

index = index--;

goToXY(index + x, indey + y);

printf(" ");

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == KEY\_ENTER)

{

index = 0;

indey++;

indey = (indey >= kolommen) ? kolommen : indey++;

}

} while (keuze != KEY\_ESCAPE);

}

CBilliard billard;

CBar billardbar;

CCasino billardcasino;

CCinema billardcinema;

CBilliardTable tolik;

Admin k;

void exitAction()

{

exit(0);

}

void defaultAction() {

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

printf("default action");

getchar();

#endif

}

void Starter()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(0);

billard.ChooseKind(0,1);

system("pause");

#endif

}

void Starter1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(1);

billard.ChooseKind(1,1);

system("pause");

#endif

}

void Starter2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(2);

billard.ChooseKind(2,1);

system("pause");

#endif

}

void Starter3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(3);

billard.ChooseKind(3,1);

system("pause");

#endif

}

void Starter4()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(4);

billard.ChooseKind(4,1);

system("pause");

#endif

}

void Starter5()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(5);

billard.ChooseKind(5,1);

system("pause");

#endif

}

void Starter0()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(0);

billard.ChooseKind(0,0);

system("pause");

#endif

}

void Starter11()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(1);

billard.ChooseKind(1, 0);

system("pause");

#endif

}

void readStoles()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(0);

system("pause");

#endif

}

void Starter22()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(2);

billard.ChooseKind(2, 0);

system("pause");

#endif

}

void Starter33()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(3);

billard.ChooseKind(3, 0);

system("pause");

#endif

}

void Starter44()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(4);

billard.ChooseKind(4, 0);

system("pause");

#endif

}

void Starter55()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(5);

billard.ChooseKind(5, 0);

system("pause");

#endif

}

void additionalbillard()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

cout << "There are any additional entertainments!" << endl;

\_getch();

#endif

}

void additionalbar()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(1);

system("pause");

#endif

}

void additionalbar1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(2);

system("pause");

#endif

}

void additionalbar2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(3);

system("pause");

#endif

}

void additionalbar3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(4);

system("pause");

#endif

}

void additionalbar4()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(5);

system("pause");

#endif

}

void additionalbar5()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(6);

system("pause");

#endif

}

void additionalbar6()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(1);

system("pause");

#endif

}

void additionalcasino()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(1);

system("pause");

#endif

}

void additionalcasino1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(2);

system("pause");

#endif

}

void additionalcasino2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(3);

system("pause");

#endif

}

void additionalcasino3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(4);

system("pause");

#endif

}

void additionalcasino4()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(5);

system("pause");

#endif

}

void additionalcasino5()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(6);

system("pause");

#endif

}

void additionalcasino6()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(7);

system("pause");

#endif

}

void additionalcasino7()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(8);

system("pause");

#endif

}

void AdminBar()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeBar();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void Admininforms()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.AdminInform();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void AdminCinema()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeCinema();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void AdminCasino()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeCasino();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void AdminBilliard()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeBilliard();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void additionalcinema()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcinema.AdditionalEnter(1);

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void exitfrom()

{

clearGoToXY(0, 0);

cout << "Goodbye!" << endl;

system("pause");

exit(0);

}

void information()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billard.information();

\_getch();

#endif

}

void information1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.information();

\_getch();

system("pause");

#endif

}

void information2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.information();

\_getch();

#endif

}

void kek()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcinema.information();

#endif

}

void password()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

AdminBilliard();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void password0()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

AdminBar();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void password1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

\_getch();

AdminCinema();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void password2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

AdminCasino();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void information3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcinema.information();

\_getch();

#endif

}

void keks()

{

billard.AdditionalEnter(1);

}

void infos()

{

cout << "Welcome to our netowork!" << endl;

cout << "You can choose a billiard!" << endl;

}

struct MenuElement {

char menuName[MAX\_LENGTH\_MENU\_NAME];

struct MenuElement \* SubMenuSet;

void(\*action)();

};

extern struct MenuElement Menu1[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE1[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE2[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE3[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE4[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE5[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE6[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement casino[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement cinema[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuK[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuA[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuB[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuC[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuD[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl2[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl3[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl4[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl5[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl6[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl7[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement Adminmenu[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement

baseMenu[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Billiard", menuA,(void(\*)())NULL},

{ "Billiard with Bar", menuB, (void(\*)())NULL },

{ "Billiard with Casino", menuC, (void(\*)())NULL },

{ "Billiard with Cinema", menuD, (void(\*)())NULL },

{"Exit",Menu1,(void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

Menu1[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Admin", Adminmenu,(void(\*)())NULL},

{ "Visitor", baseMenu, (void(\*)())NULL },

{"Exit",(struct MenuElement \*)NULL,exitfrom},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

Adminmenu[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Change Billiard",(struct MenuElement \*)NULL,password},

{ "Change Bar", (struct MenuElement \*)NULL,password0},

{ "Change Cinema",(struct MenuElement \*)NULL,password1},

{ "Change Casino",(struct MenuElement \*)NULL,password2},

{"Information",(struct MenuElement \*)NULL,Admininforms},

{"Exit",Menu1,(void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

menuTabl[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Easy", AGREE,(void(\*)())NULL},

{ "Main", AGREE1, (void(\*)())NULL },

{ "Hard", AGREE2, (void(\*)())NULL },

{ "Pog", AGREE3, (void(\*)())NULL },

{"Took",AGREE4,(void(\*)())NULL},

{"Frame",AGREE5,(void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

, menuA[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{"Information",(struct MenuElement \*)NULL,information},

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool",menuTabl,(void(\*)())NULL },

{ "Snooker",menuTabl,(void(\*)())NULL },

{ "English Billiards",menuTabl,(void(\*)())NULL },

{ "Carom", menuTabl,(void(\*)())NULL },

{ "Chicago", menuTabl,(void(\*)())NULL },

{"Additional entertainment",(struct MenuElement \*)NULL,defaultAction},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

AGREE[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter },

{"I Don't Agree with standarts ru les of billiard!",(struct MenuElement\*)NULL,Starter0 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE1[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter1 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter11 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE2[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter2 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter22 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE3[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter3 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter33 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE4[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter4 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter44 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE5[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter5 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter55 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

menuK[MAX\_MENU\_ITEMS\_COUNT] = {

{"Prev menu",menuC,(void(\*)())NULL},

{ "Beer",(struct MenuElement \*)NULL,additionalbar},

{"Vodka",(struct MenuElement\*)NULL,additionalbar1},

{"Cola",(struct MenuElement\*)NULL,additionalbar2},

{"Pizza",(struct MenuElement\*)NULL,additionalbar3 },

{"Sidr",(struct MenuElement\*)NULL,additionalbar4},

{"Kekos",(struct MenuElement\*)NULL,additionalbar5},

{"",(struct MenuElement \*)NULL,(void(\*)())NULL}

}

, menuC[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Information",(struct MenuElement \*)NULL,information2},

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool", menuTabl,(void(\*)())NULL },

{ "Snooker", menuTabl,(void(\*)())NULL },

{ "English Billiards",menuTabl,(void(\*)())NULL },

{ "Carom",menuTabl,(void(\*)())NULL },

{ "Chicago",menuTabl,(void(\*)())NULL },

{"Additional entertainment",casino,(void(\*)())NULL },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

casino[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuC, (void(\*)())NULL },

{ "Bingo",(struct MenuElement \*)NULL, additionalcasino },

{ "Baccart", (struct MenuElement \*)NULL,additionalcasino2 },

{ "Wheel of Fortune", (struct MenuElement \*)NULL,additionalcasino3 },

{ "Keno", (struct MenuElement \*)NULL, additionalcasino4 },

{ "Pai Gow poker", (struct MenuElement \*)NULL, additionalcasino5 },

{ "Slots", (struct MenuElement \*)NULL, additionalcasino6 },

{"Black Jack",(struct MenuElement\*)NULL,additionalcasino7},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

, menuB[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Information",(struct MenuElement \*)NULL,information1},

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool",menuTabl,(void(\*)())NULL },

{ "Snooker",menuTabl,(void(\*)())NULL },

{ "English Billiards",menuTabl,(void(\*)())NULL },

{ "Carom", menuTabl,(void(\*)())NULL },

{ "Chicago",menuTabl,(void(\*)())NULL },

{"Additional entertainment",menuK,(void(\*)())NULL },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

cinema[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuD, (void(\*)())NULL },

{ "Avengers",(struct MenuElement\*)NULL,additionalcinema },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuD[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Information",(struct MenuElement \*)NULL,information3},

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool", menuTabl,(void(\*)())NULL },

{ "Snooker", menuTabl,(void(\*)())NULL },

{ "English Billiards", menuTabl,(void(\*)())NULL },

{ "Carom",menuTabl,(void(\*)())NULL },

{ "Chicago", menuTabl,(void(\*)())NULL },

{"Additional entertainment",cinema,(void(\*)())NULL },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

};

unsigned int render(struct MenuElement \* menu) {

unsigned int index = 0;

clearGoToXY(0, 0);

if (menu) {

for (; menu[index].menuName[0]; ++index) {

printf(menu[index].menuName);

clearGoToXY(49, index);

printf("[ ]\r\n");

}

}

return --index;

}

int main() {

struct MenuElement \* menu =Menu1;

char spaceOp[MAX\_MENU\_SPACE][MAX\_MENU\_SPACE];

while (true) {

unsigned int lastIndex = render(menu);

if ((int)lastIndex == -1) {

return 0;

}

unsigned int selectedIndex = runSpaceOp(50, 0, lastIndex, spaceOp, 0);

if ((int)selectedIndex == -1) {

return 0;

}

if (menu[selectedIndex].action != NULL) {

menu[selectedIndex].action();

}

if (menu[selectedIndex].SubMenuSet != NULL) {

menu = menu[selectedIndex].SubMenuSet;

}

}

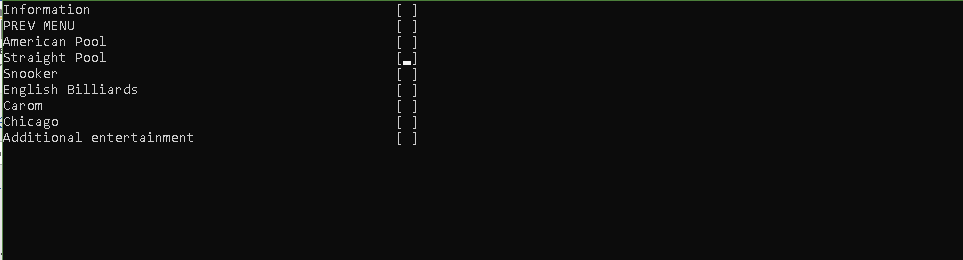
}

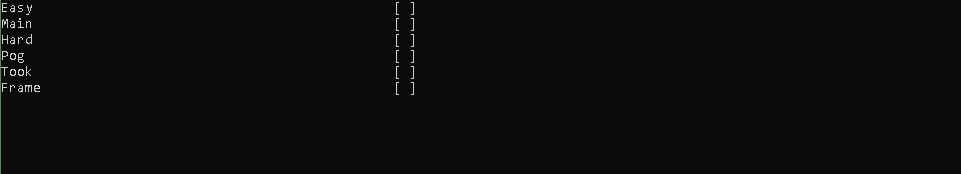
**3. Відлагодження**

**3.1 Відлагодження та тестування програми в цілому:**

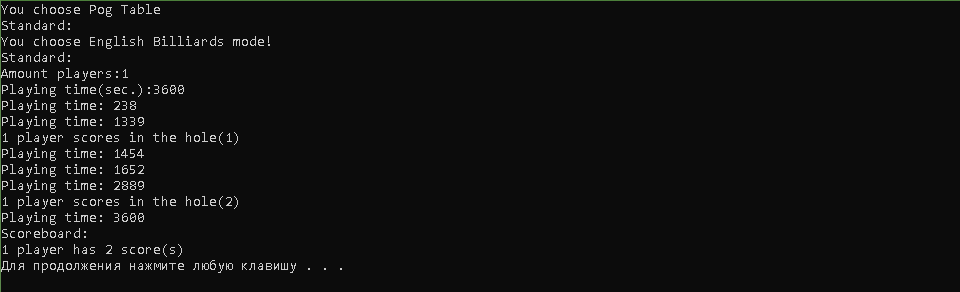
Подібні кроки були виконані і з перевіркою всієї програми. Нижче наведені скріншоти, які запевнюють в коректній роботі програми:

**1)**

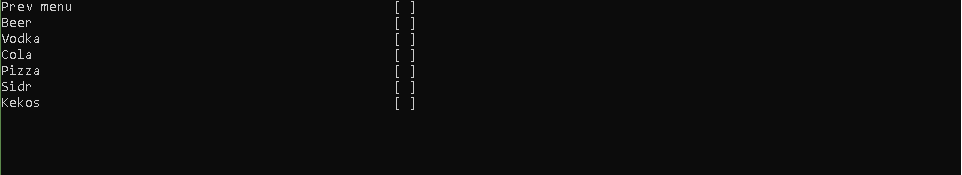
**2)**

**3)**

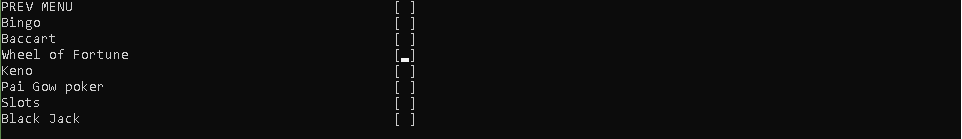
**4)**

**5)**

**6)**

****

**7)**

**8)**

**9)**

**10)**

**11)**

**3.2Аналіз часу виконання програми в залежності від об'єму вхідних даних**

Всі операції всередині програми виконуються миттєво, немає затримки між переходами по меню та виведенням інформації на консоль. Читання даних з файлу та виконання операцій з числами виконується бездоганно.

**Висновки**

Використання ООП для вирішення задач такого типу значно полегшує програмісту роботу, робить код більш структурованим та дозволяє легше його читати. Для себе можу зробити важливий висновок, що ООП — гнучкий інструмент який ідеально підходить для написання складних програм які потребують описання деяких об’єктів та їх поведінки.

**Використана література :** Design\_Patterns\_in\_Modern\_C, Зуев - Стандарт C++. Перевод, комментарии, примеры, Bjarne\_Stroustrup\_-\_The\_C\_\_Programming\_Languag, Параллельное программирование на С++ в действии, standartnaya-biblioteka-c-spravochnoe-rukovodstvo-2-e-izdanie, (-)Essential C++ - Stanley B. Lippman, Antony\_Polukhin\_-\_Boost\_C\_Application\_Development\_Cookbook\_-\_2013 .

**Додатки :**

**Додаток А(Лістинг програмного коду проекту)**

**Заголовкові файли :**

**CBilliardBase.h**

#pragma once

#include <iostream>

#include <string>

#include <conio.h>

#include <list>

#include <vector>

#include <fstream>

using namespace std;

#ifndef Billiard\_Base

#define Billiard\_Base

class CBilliardBase

{

private:

string Kinds = "Kindslist.txt";

protected :

string name;

list<string> listKindPools;

int amountPlayers;

std::vector<int> coins;

long long time;

void ShowKind();

void ChooseKind(int choice,int n);

virtual void AdditionalEnter(int choice) = 0;

public:

CBilliardBase();

CBilliardBase(int amountPlayers, int time);

virtual void Start();

virtual void information() = 0;

void Play();

int GetAmountPlayers()const;

long long GetTime()const;

std::vector<int> GetCoins()const;

void SetAmountPlayers(const int amountPlayers);

void SetTime(const long long time);

void SetCoins(const std::vector<int> coins);

};

**CBilliard.h**

#pragma once

#ifndef CBilliard\_hpp

#define CBilliard\_hpp

//#include "stdafx.h"

#include "CBilliardBase.h"

class CBilliard :public CBilliardBase {

private:

protected:

public:

string nameBilliard;

void AdditionalEnter(int choice)override;

CBilliard();

CBilliard(int amountPlayers, int time,string nameBilliard);

void information()override;

void Start()override;

void ChooseKind(int choice,int n);

};

#endif /\* CBilliard\_hpp \*/

**CBilliardWithBar.h**

#ifndef CBilliardWithBar\_hpp

#define CBilliardWithBar\_hpp

//#include "stdafx.h"

#include "CBilliardBase.h"

#include <vector>

#include <string>

class CBilliardWithBar : public CBilliardBase {

public:

string nameBar;

void Start()override;

CBilliardWithBar();

~CBilliardWithBar();

};

#endif /\* CBilliardWithBar\_hpp \*/

**CBilliardWithCinema.h**

#ifndef CBilliardWithCinema\_hpp

#define CBilliardWithCinema\_hpp

//#include "stdafx.h"

#include "CBilliardBase.h"

#include <vector>

#include <string>

class CBilliardWithCinema : public CBilliardBase {

private:

public:

string nameCinema;

CBilliardWithCinema();

void Start()override;

~CBilliardWithCinema();

};

#endif /\* CBilliardWithCinema\_hpp \*/

**CBilliardWithCasino.h**

#ifndef CBilliardWithCasino\_hpp

#define CBilliardWithCasino\_hpp

//#include "stdafx.h"

#include "CBilliardBase.h"

#include <vector>

#include <string>

class CBilliardWithCasino : public CBilliardBase {

private:

public:

string nameCasino;

CBilliardWithCasino();

void Start()override;

~CBilliardWithCasino();

};

#endif /\* CBilliardWithCasino\_hpp \*/

**CBilliardTable.h**

#pragma once

#include "CBilliardBase.h"

#include "CBilliard.h"

#ifndef Table

#define Table

class CBilliardTable:CBilliard

{

protected :

string tables = "tableslist.txt";

list<string> listtables;

public:

CBilliard a;

void chooseTable(int choice1);

CBilliardTable();

void showTables();

~CBilliardTable();

};

#endif // !Table

**CCasino.h**

#pragma once

#include "CBilliardWithCasino.h"

#ifndef CCasinoh

#define CCasinoh

class CCasino : public CBilliardWithCasino {

private:

public:

std::vector<std::string> listCasinos;

std::string casinoTxt = "casinolist.txt";

void AdditionalEnter(int choice)override;

CCasino();

void Start()override;

void ListCasinos();

void information()override;

void SetListCasinos(const std::vector<std::string> listCasinos);

std::vector<std::string> GetListCasinos()const;

};

#endif /\* CBilliardWithCasino\_hpp \*/

**CBar.h**

#pragma once

#ifndef CBAR

#define CBAR

#include "CBilliardWithBar.h"

class CBar: public CBilliardWithBar

{

public:

std::vector<std::string> menu;

std::string barTxt = "menuBar.txt";

void AdditionalEnter(int choice);

CBar();

void ListMenu();

void SetMenu(const std::vector<std::string> menu);

void information()override;

std::vector<std::string> GetMenu()const;

void AddDish();

void AddItsem(std::string item);

};

#endif /\* CBilliardWithBar\_hpp \*/

**CCinema.h**

#ifndef CCinema\_hpp

#define CCinema\_hpp

//#include "stdafx.h"

#include "CBilliardBase.h"

#include <vector>

#include <string>

#include "CBilliardWithCinema.h"

class CCinema : public CBilliardWithCinema {

private:

public:

std::vector< std::pair<std::string, int> > listFilms;

std::string filmTxt = "films.txt";

void AdditionalEnter(int choice)override;

CCinema();

void information()override;

void ListFilms();

void SetListCinema(const std::vector<std::pair<std::string, int> > listFilm);

std::vector<std::pair<std::string, int>> GetListFilm()const;

};

**Файли ресурсів :**

**CBilliardBase.cpp**

#include "CBilliardBase.h"

#include <iostream>

#include <vector>

#include <string>

#include <ctime>

#include <thread>

#include <chrono>

using namespace std;

CBilliardBase::CBilliardBase() : CBilliardBase(1, 3600) {}

CBilliardBase::CBilliardBase(int amountPlayers, int time) : amountPlayers(amountPlayers), time(time), coins(vector<int>(amountPlayers, 0)) {

ifstream in(Kinds);

if (in.is\_open()) {

string kind;

while (getline(in, kind)) {

listKindPools.push\_back(kind);

}

}

}

void CBilliardBase::ShowKind() {

if (listKindPools.empty()) {

cerr << "There arent any kind of pools!Please add them!" << endl;

return;

}

int count = 1;

for (string kind : listKindPools) {

cout << count++ << " - " << kind << endl;

}

}

void CBilliardBase::ChooseKind(int choice,int n) {

char y;

auto it = next(listKindPools.begin(), choice);

cout << "You choose " << \*it << " mode!" << endl;

cout << "Standard:" << endl;

cout << "Amount players:" << amountPlayers << endl;

cout << "Playing time(sec.):" << time << endl;

if (n == 1 ){

Play();

}

else if (n==0){

int amountPlayers;

long long time;

cout << "Input amount players: ";

cin >> amountPlayers;

cout << "Input playing time(sec.): ";

cin >> time;

SetTime(time);

SetAmountPlayers(amountPlayers);

Play();

}

}

void CBilliardBase::Start() {

#ifndef DISABLE\_DEFAULT\_ACTION

int choose;

auto count = listKindPools.size() + 1;

while (true) {

ShowKind();

cout << count << " - Additional entertainment" << endl;

cout << "0 - Back to main menu" << endl;

cin >> choose;

if (choose > 0 && choose < count) {

ChooseKind(choose - 1,1);

}

else if (choose == count) {

}

else if (choose == 0) {

break;

}

}

#endif

}

void CBilliardBase::Play() {

srand(std::time(NULL));

long long time = 0;

while (time < this->time) {

for (int i = 0; i < amountPlayers; ++i) {

int result = rand() % 2;

coins[i] += result;

if (result == 1) {

cout << i + 1 << " player scores in the hole(" << coins[i] << ")" << endl;

}

}

time += rand() % (this->time / 2);

if (time > this->time) {

time = this->time;

}

this\_thread::sleep\_for(chrono::milliseconds(1000));

cout << "Playing time: " << time << endl;

}

cout << "Scoreboard:" << endl;

for (int i = 0; i < amountPlayers; ++i) {

cout << i + 1 << " player has " << coins[i] << " score(s)" << endl;

}

}

int CBilliardBase::GetAmountPlayers()const {

return amountPlayers;

}

long long CBilliardBase::GetTime()const {

return time;

}

std::vector<int> CBilliardBase::GetCoins()const {

return coins;

}

void CBilliardBase::SetAmountPlayers(const int amountPlayers) {

this->amountPlayers = amountPlayers;

coins.resize(amountPlayers);

}

void CBilliardBase::SetTime(const long long time) {

this->time = time;

}

void CBilliardBase::SetCoins(const std::vector<int> coins) {

this->coins = coins;

}

**CBilliardWithBar.cpp**

#include "CBilliardWithBar.h"

CBilliardWithBar::CBilliardWithBar()

{

}

CBilliardWithBar::~CBilliardWithBar()

{

}

void CBilliardWithBar::Start() {

CBilliardBase::Start();

}

**CBilliardWithCinema.cpp**

#include "CBilliardWithCinema.h"

CBilliardWithCinema::CBilliardWithCinema()

{

}

void CBilliardWithCinema::Start() {

CBilliardBase::Start();

}

CBilliardWithCinema::~CBilliardWithCinema()

{

}

**CBilliard.cpp**

//#include "stdafx.h"

#include "CBilliard.h"

#include <iostream>

using namespace std;

void CBilliard::AdditionalEnter(int choice) {

cout << "There arent any entertaiments!" << endl;

}

CBilliard::CBilliard() : CBilliardBase() {}

CBilliard::CBilliard(int amountPlayers, int time,string nameBilliard) : CBilliardBase(amountPlayers, time) {

nameBilliard = "Billard POOL";

}

void CBilliard::information()

{

nameBilliard = "Billiard POOL";

cout << "HELLO,WELCOME TO OUR " << nameBilliard <<endl<< "We have standarts rules of billiard games" <<endl<<" Amount of players on game: " << amountPlayers <<endl<< " and time: " << time << endl;

}

void CBilliard::Start() {

CBilliardBase::Start();

}

void CBilliard::ChooseKind(int choice,int n)

{

CBilliardBase::ChooseKind(choice,n);

}

**CBilliardWithCasino.cpp**

#include "CBilliardBase.h"

#include "CBilliardWithCasino.h"

CBilliardWithCasino::CBilliardWithCasino()

{

}

void CBilliardWithCasino::Start() {

CBilliardBase::Start();

}

CBilliardWithCasino::~CBilliardWithCasino()

{

}

**CBilliardTable.cpp**

#pragma once

#include "CBilliardTable.h"

CBilliardTable::CBilliardTable()

{

ifstream ik(tables);

if (ik.is\_open()) {

string table;

while (getline(ik, table)) {

listtables.push\_back(table);

}

}

}

void CBilliardTable::showTables()

{

if (listtables.empty()) {

cerr << "There arent any kind of pools!Please add them!" << endl;

return;

}

int count = 1;

for (string table : listtables) {

cout << count++ << " - " << table << endl;

}

}

void CBilliardTable::chooseTable(int choice1)

{

auto it = next(listtables.begin(), choice1);

cout << "You choose " << \*it << " Table" << endl;

cout << "Standard:" << endl;

}

CBilliardTable::~CBilliardTable()

{

}

**CCasino.cpp**

#include "CCasino.h"

//#include "stdafx.h"

#include "CBilliardWithCasino.h"

#include "CHuman.h"

#include <string>

#include <fstream>

#include <iostream>

#include <ctime>

using namespace std;

void CCasino::AdditionalEnter(int choice) {

srand(std::time(NULL));

int money1;

while (true) {

if (listCasinos.empty()) {

return;

}

if (choice > 0 && choice <= listCasinos.size()) {

cout << "Input money:";

cin >> money1;

int x, y;

x = rand() % 10;

y = rand() % 10;

auto it = next(listCasinos.begin(), choice - 1);

cout << "You play " << \*it << ".";

if (x == y) {

cout << " Congratulation! You win " << money1 \* 2 << endl;

}

else {

cout << " You lose your money!" << endl;

}

break;

}

}

}

CCasino::CCasino() : CBilliardWithCasino() {

ifstream in(casinoTxt);

if (in.is\_open()) {

string casino;

while (getline(in, casino)) {

listCasinos.push\_back(casino);

}

}

nameCasino = "CBilliard With Casino";

}

void CCasino::information()

{

cout << "Hello,welcome to the" << nameCasino << endl << "There are has a list of casinos in Additional entertainments" << endl;

}

void CCasino::ListCasinos() {

if (listCasinos.empty()) {

cerr << "There arent any kind of casino!Please add them!" << endl;

return;

}

int count = 1;

cout << "List of casinos:" << endl;

for (string casino : listCasinos) {

cout << count++ << " - " << casino << endl;

}

}

void CCasino::Start() {

CBilliardBase::Start();

}

void CCasino::SetListCasinos(const std::vector<std::string> listCasinos) {

this->listCasinos = listCasinos;

}

std::vector<std::string> CCasino::GetListCasinos()const {

return listCasinos;

}

**CBar.cpp**

#include "CBar.h"

//#include "stdafx.h"

#include "CBilliardWithBar.h"

#include <iostream>

#include <string>

#include <fstream>

#include <chrono>

#include <thread>

using namespace std;

void CBar::AdditionalEnter(int choice) {

while (true) {

if (menu.empty()) {

return;

}

if (choice > 0 && choice <= menu.size()) {

auto it = next(menu.begin(), choice - 1);

cout << "Eating/drinking " << \*it << "..." << endl;

this\_thread::sleep\_for(chrono::milliseconds(3125));

cout << "Leaving the bar..." << endl;

break;

}

}

}

void CBar::ListMenu() {

if (menu.empty()) {

cerr << "There arent any dishes in menu!Please add them!" << endl;

return;

}

int count = 1;

cout << "Menu:" << endl;

for (auto item : menu) {

cout << count++ << " - " << item << endl;

}

}

CBar::CBar() : CBilliardWithBar() {

ifstream in(barTxt);

if (in.is\_open()) {

string menuItem;

while (getline(in, menuItem)) {

menu.push\_back(menuItem);

}

}

nameBar = "Billiard with bar";

}

void CBar::AddDish()

{

}

void CBar::information()

{

cout << "HELLO,Welcome to the " << nameBar << endl << "There are has a list of dishes in additional entertainments " << endl;

}

void CBar::SetMenu(const std::vector<std::string> menu) {

this->menu = menu;

}

std::vector<std::string> CBar::GetMenu()const {

return menu;

}

**CCinema.cpp**

#include "CCinema.h"

#include "CBilliardWithCinema.h"

//#include "stdafx.h"

#include <string>

#include <iostream>

#include <fstream>

#include <chrono>

#include <thread>

using namespace std;

void CCinema::AdditionalEnter(int choice) {

while (true) {

ListFilms();

if (listFilms.empty()) {

return;

}

if (choice > 0 && choice <= listFilms.size()) {

auto it = next(listFilms.begin(), choice - 1);

cout << "Watching film " << it->first << " with duration " << it->second << "(min.)" << " ..." << endl;

this\_thread::sleep\_for(chrono::milliseconds(5125));

cout << "Film ends. Leaving the cinema..." << endl;

break;

}

}

}

void CCinema::ListFilms() {

if (listFilms.empty()) {

cerr << "There arent any films!Please add them!" << endl;

return;

}

int count = 1;

cout << "List of films:" << endl;

for (auto film : listFilms) {

cout << count++ << " - " << film.first << "(" << film.second << " min.)" << endl;

}

}

CCinema::CCinema() : CBilliardWithCinema() {

ifstream in(filmTxt);

if (in.is\_open()) {

string parse;

string film;

int duration;

while (getline(in, parse)) {

auto it = parse.find(";");

film.append(parse, 0, it);

duration = stoi(parse.substr(it + 1, parse.size() - it));

listFilms.push\_back(make\_pair(film, duration));

}

}

nameCinema = "Billiard with cinema";

}

void CCinema::information()

{

cout << "Hello,Welcome to the " << nameCinema << endl << " There are list of films in additional entertainmetns!" << endl;

}

void CCinema::SetListCinema(const std::vector<pair<string, int> > listFilms) {

this->listFilms = listFilms;

}

std::vector<pair<string, int> > CCinema::GetListFilm()const {

return listFilms;

}

**Menu.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define WIN32\_LEAN\_AND\_MEAN

//#include "stdafx.h"

// from HKEY\_USERS\.DEFAULT\Keyboard Layout\Preload

#define ENG\_KEYBOARD\_LAYOUT "00000409"

#define UKR\_KEYBOARD\_LAYOUT "00000422"

#define RUS\_KEYBOARD\_LAYOUT "00000419"

#include "Admin.h"

#include "CBilliardTable.h"

#include "CCinema.h"

#include "stdio.h"

#include "CCasino.h"

#include <stdlib.h>

#include <wchar.h>

#include <windows.h>

#include "CBilliardWithCinema.h"

#include "CBar.h"

#include "CBilliardWithBar.h"

#include "CBilliardWithCasino.h"

#include "CBilliard.h"

#include "CBilliardBase.h"

//#define DISABLE\_DEFAULT\_ACTION

#define MAX\_MENU\_SPACE 100

#define MAX\_LENGTH\_MENU\_NAME 100

#define MAX\_MENU\_ITEMS\_COUNT 100

void goToXY(int x, int y)

{

COORD p = { x, y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), p);

}

void clearGoToXY(int x, int y) {

COORD topLeft = { x, y };

HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_SCREEN\_BUFFER\_INFO screen;

DWORD written;

GetConsoleScreenBufferInfo(console, &screen);

FillConsoleOutputCharacterA(

console, ' ', screen.dwSize.X \* screen.dwSize.Y, topLeft, &written

);

FillConsoleOutputAttribute(

console, FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE,

screen.dwSize.X \* screen.dwSize.Y, topLeft, &written

);

SetConsoleCursorPosition(console, topLeft);

}

#define KEY\_UP 72

#define KEY\_DOWN 80

#define KEY\_LEFT 75

#define KEY\_RIGHT 77

#define KEY\_ENTER 13

#define KEY\_BACKSPACE 8

#define KEY\_ESCAPE 27

unsigned int runSpaceOp(int x, int y, int kolommen, char spaceOp[MAX\_MENU\_SPACE][MAX\_MENU\_SPACE], int rijen = MAX\_MENU\_SPACE)

{

int index = 0, indey = 0, keuze, lol = MAX\_MENU\_SPACE;

do {

goToXY(index + x, indey + y);

keuze = 0;

keuze = \_getch();

if (keuze == 0 || keuze == 0xE0 || keuze == 224)

{

keuze = \_getch();

goToXY(index + x, indey + y);

switch (keuze)

{

case KEY\_UP:indey--;

break;

case KEY\_LEFT: index--;

break;

case KEY\_DOWN:indey++;

break;

case KEY\_RIGHT: index++;

break;

}

indey = (indey <= 0) ? 0 : (indey > kolommen) ? kolommen : indey;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == 32 || (keuze >= 46 && keuze <= 57) || (keuze <= 64 && keuze >= 126))

{

spaceOp[index][indey] = (char)keuze;

printf("%c", spaceOp[index][indey]);

index++;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == KEY\_ENTER)

{

return indey;

}

} while (keuze != KEY\_ESCAPE);

return -1; //

}

void runTextOp(int x, int y, int kolommen, char spaceOp[MAX\_MENU\_SPACE][MAX\_MENU\_SPACE], int rijen = MAX\_MENU\_SPACE)

{

int index = 0, indey = 0, keuze, lol = MAX\_MENU\_SPACE;

do {

goToXY(index + x, indey + y);

keuze = 0;

keuze = \_getch();

if (keuze == 0 || keuze == 0xE0 || keuze == 224)

{

keuze = \_getch();

goToXY(index + x, indey + y);

switch (keuze)

{

case KEY\_UP:indey--;

break;

case KEY\_LEFT: index--;

break;

case KEY\_DOWN:indey++;

break;

case KEY\_RIGHT: index++;

break;

}

indey = (indey <= 0) ? 0 : (indey > kolommen) ? kolommen : indey;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == 32 || (keuze >= 46 && keuze <= 57) || (keuze <= 64 && keuze >= 126))

{

spaceOp[index][indey] = (char)keuze;

printf("%c", spaceOp[index][indey]);

index++;

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == KEY\_BACKSPACE)

{

index = index--;

goToXY(index + x, indey + y);

printf(" ");

index = (index <= 0) ? 0 : (index > rijen) ? rijen : index;

}

if (keuze == KEY\_ENTER)

{

index = 0;

indey++;

indey = (indey >= kolommen) ? kolommen : indey++;

}

} while (keuze != KEY\_ESCAPE);

}

CBilliard billard;

CBar billardbar;

CCasino billardcasino;

CCinema billardcinema;

CBilliardTable tolik;

Admin k;

void exitAction()

{

exit(0);

}

void defaultAction() {

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

printf("default action");

getchar();

#endif

}

void Starter()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(0);

billard.ChooseKind(0,1);

system("pause");

#endif

}

void Starter1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(1);

billard.ChooseKind(1,1);

system("pause");

#endif

}

void Starter2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(2);

billard.ChooseKind(2,1);

system("pause");

#endif

}

void Starter3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(3);

billard.ChooseKind(3,1);

system("pause");

#endif

}

void Starter4()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(4);

billard.ChooseKind(4,1);

system("pause");

#endif

}

void Starter5()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(5);

billard.ChooseKind(5,1);

system("pause");

#endif

}

void Starter0()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(0);

billard.ChooseKind(0,0);

system("pause");

#endif

}

void Starter11()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(1);

billard.ChooseKind(1, 0);

system("pause");

#endif

}

void readStoles()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(0);

system("pause");

#endif

}

void Starter22()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(2);

billard.ChooseKind(2, 0);

system("pause");

#endif

}

void Starter33()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(3);

billard.ChooseKind(3, 0);

system("pause");

#endif

}

void Starter44()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(4);

billard.ChooseKind(4, 0);

system("pause");

#endif

}

void Starter55()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

tolik.chooseTable(5);

billard.ChooseKind(5, 0);

system("pause");

#endif

}

void additionalbillard()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

cout << "There are any additional entertainments!" << endl;

\_getch();

#endif

}

void additionalbar()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(1);

system("pause");

#endif

}

void additionalbar1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(2);

system("pause");

#endif

}

void additionalbar2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(3);

system("pause");

#endif

}

void additionalbar3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(4);

system("pause");

#endif

}

void additionalbar4()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(5);

system("pause");

#endif

}

void additionalbar5()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(6);

system("pause");

#endif

}

void additionalbar6()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.AdditionalEnter(1);

system("pause");

#endif

}

void additionalcasino()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(1);

system("pause");

#endif

}

void additionalcasino1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(2);

system("pause");

#endif

}

void additionalcasino2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(3);

system("pause");

#endif

}

void additionalcasino3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(4);

system("pause");

#endif

}

void additionalcasino4()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(5);

system("pause");

#endif

}

void additionalcasino5()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(6);

system("pause");

#endif

}

void additionalcasino6()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(7);

system("pause");

#endif

}

void additionalcasino7()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.AdditionalEnter(8);

system("pause");

#endif

}

void AdminBar()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeBar();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void Admininforms()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.AdminInform();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void AdminCinema()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeCinema();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void AdminCasino()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeCasino();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void AdminBilliard()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

k.ChangeBilliard();

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void additionalcinema()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcinema.AdditionalEnter(1);

system("pause");

#endif // !DISABLE\_DEFAULT\_ACTION

}

void exitfrom()

{

clearGoToXY(0, 0);

cout << "Goodbye!" << endl;

system("pause");

exit(0);

}

void information()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billard.information();

\_getch();

#endif

}

void information1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardbar.information();

\_getch();

system("pause");

#endif

}

void information2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcasino.information();

\_getch();

#endif

}

void kek()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcinema.information();

#endif

}

void password()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

AdminBilliard();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void password0()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

AdminBar();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void password1()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

\_getch();

AdminCinema();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void password2()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

while (true)

{

string pass;

pass = "12345";

cout << "Enter pass: ";

cin >> pass;

if (pass == "12345")

{

cout << "Succsesfull!" << endl;

AdminCasino();

break;

}

else

{

cout << "WRONG" << endl;

}

}

#endif

}

void information3()

{

#ifndef DISABLE\_DEFAULT\_ACTION

clearGoToXY(0, 0);

billardcinema.information();

\_getch();

#endif

}

void keks()

{

billard.AdditionalEnter(1);

}

void infos()

{

cout << "Welcome to our netowork!" << endl;

cout << "You can choose a billiard!" << endl;

}

struct MenuElement {

char menuName[MAX\_LENGTH\_MENU\_NAME];

struct MenuElement \* SubMenuSet;

void(\*action)();

};

extern struct MenuElement Menu1[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE1[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE2[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE3[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE4[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE5[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement AGREE6[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement casino[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement cinema[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuK[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuA[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuB[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuC[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuD[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl2[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl3[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl4[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl5[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl6[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTabl7[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement Adminmenu[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement

baseMenu[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Billiard", menuA,(void(\*)())NULL},

{ "Billiard with Bar", menuB, (void(\*)())NULL },

{ "Billiard with Casino", menuC, (void(\*)())NULL },

{ "Billiard with Cinema", menuD, (void(\*)())NULL },

{"Exit",Menu1,(void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

Menu1[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Admin", Adminmenu,(void(\*)())NULL},

{ "Visitor", baseMenu, (void(\*)())NULL },

{"Exit",(struct MenuElement \*)NULL,exitfrom},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

Adminmenu[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Change Billiard",(struct MenuElement \*)NULL,password},

{ "Change Bar", (struct MenuElement \*)NULL,password0},

{ "Change Cinema",(struct MenuElement \*)NULL,password1},

{ "Change Casino",(struct MenuElement \*)NULL,password2},

{"Information",(struct MenuElement \*)NULL,Admininforms},

{"Exit",Menu1,(void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

menuTabl[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Easy", AGREE,(void(\*)())NULL},

{ "Main", AGREE1, (void(\*)())NULL },

{ "Hard", AGREE2, (void(\*)())NULL },

{ "Pog", AGREE3, (void(\*)())NULL },

{"Took",AGREE4,(void(\*)())NULL},

{"Frame",AGREE5,(void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

, menuA[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{"Information",(struct MenuElement \*)NULL,information},

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool",menuTabl,(void(\*)())NULL },

{ "Snooker",menuTabl,(void(\*)())NULL },

{ "English Billiards",menuTabl,(void(\*)())NULL },

{ "Carom", menuTabl,(void(\*)())NULL },

{ "Chicago", menuTabl,(void(\*)())NULL },

{"Additional entertainment",(struct MenuElement \*)NULL,defaultAction},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

AGREE[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter },

{"I Don't Agree with standarts ru les of billiard!",(struct MenuElement\*)NULL,Starter0 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE1[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter1 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter11 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE2[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter2 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter22 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE3[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter3 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter33 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE4[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter4 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter44 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

AGREE5[MAX\_MENU\_ITEMS\_COUNT] = {

{"I Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter5 },

{"I Don't Agree with standarts rules of billiard!",(struct MenuElement\*)NULL,Starter55 },

{ "Cancel request",menuA,(void(\*)())NULL}

}

,

menuK[MAX\_MENU\_ITEMS\_COUNT] = {

{"Prev menu",menuC,(void(\*)())NULL},

{ "Beer",(struct MenuElement \*)NULL,additionalbar},

{"Vodka",(struct MenuElement\*)NULL,additionalbar1},

{"Cola",(struct MenuElement\*)NULL,additionalbar2},

{"Pizza",(struct MenuElement\*)NULL,additionalbar3 },

{"Sidr",(struct MenuElement\*)NULL,additionalbar4},

{"Kekos",(struct MenuElement\*)NULL,additionalbar5},

{"",(struct MenuElement \*)NULL,(void(\*)())NULL}

}

, menuC[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Information",(struct MenuElement \*)NULL,information2},

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool", menuTabl,(void(\*)())NULL },

{ "Snooker", menuTabl,(void(\*)())NULL },

{ "English Billiards",menuTabl,(void(\*)())NULL },

{ "Carom",menuTabl,(void(\*)())NULL },

{ "Chicago",menuTabl,(void(\*)())NULL },

{"Additional entertainment",casino,(void(\*)())NULL },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

},

casino[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuC, (void(\*)())NULL },

{ "Bingo",(struct MenuElement \*)NULL, additionalcasino },

{ "Baccart", (struct MenuElement \*)NULL,additionalcasino2 },

{ "Wheel of Fortune", (struct MenuElement \*)NULL,additionalcasino3 },

{ "Keno", (struct MenuElement \*)NULL, additionalcasino4 },

{ "Pai Gow poker", (struct MenuElement \*)NULL, additionalcasino5 },

{ "Slots", (struct MenuElement \*)NULL, additionalcasino6 },

{"Black Jack",(struct MenuElement\*)NULL,additionalcasino7},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

, menuB[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Information",(struct MenuElement \*)NULL,information1},

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool",menuTabl,(void(\*)())NULL },

{ "Snooker",menuTabl,(void(\*)())NULL },

{ "English Billiards",menuTabl,(void(\*)())NULL },

{ "Carom", menuTabl,(void(\*)())NULL },

{ "Chicago",menuTabl,(void(\*)())NULL },

{"Additional entertainment",menuK,(void(\*)())NULL },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

cinema[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuD, (void(\*)())NULL },

{ "Avengers",(struct MenuElement\*)NULL,additionalcinema },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuD[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Information",(struct MenuElement \*)NULL,information3},

{ "American Pool",menuTabl,(void(\*)())NULL },

{ "Straight Pool", menuTabl,(void(\*)())NULL },

{ "Snooker", menuTabl,(void(\*)())NULL },

{ "English Billiards", menuTabl,(void(\*)())NULL },

{ "Carom",menuTabl,(void(\*)())NULL },

{ "Chicago", menuTabl,(void(\*)())NULL },

{"Additional entertainment",cinema,(void(\*)())NULL },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

};

unsigned int render(struct MenuElement \* menu) {

unsigned int index = 0;

clearGoToXY(0, 0);

if (menu) {

for (; menu[index].menuName[0]; ++index) {

printf(menu[index].menuName);

clearGoToXY(49, index);

printf("[ ]\r\n");

}

}

return --index;

}

int main() {

struct MenuElement \* menu =Menu1;

char spaceOp[MAX\_MENU\_SPACE][MAX\_MENU\_SPACE];

while (true) {

unsigned int lastIndex = render(menu);

if ((int)lastIndex == -1) {

return 0;

}

unsigned int selectedIndex = runSpaceOp(50, 0, lastIndex, spaceOp, 0);

if ((int)selectedIndex == -1) {

return 0;

}

if (menu[selectedIndex].action != NULL) {

menu[selectedIndex].action();

}

if (menu[selectedIndex].SubMenuSet != NULL) {

menu = menu[selectedIndex].SubMenuSet;

}

}

}