#### МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ПВНЗ ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ**

**ЧЕРКАСЬКА ФІЛІЯ**

Кафедра математичних і комп’ютерних дисциплін

**КУРСОВА РОБОТА**

з дисципліни: **“Об’єктно-орієнтована розробка програмних систем”**

на тему: **“Об’єктно-орієнтована програма обліку продуктового магазину”**

Черкаси 2008

**Вступ**

Успішна розробка і впровадження складних програмних систем - це щось більше, ніж просто програмування. Існує багато прийомів традиційних програмних розробок, наприклад, перегляд, що застосовні і в об'єктно-орієнтованій технології. У стабільному стані об'єктно-орієнтовані проекти вимагають менших ресурсів, а методи, необхідні для управління цими ресурсами, декілька відрізняються від традиційних, тому в процесі об'єктно-орієнтованої розробки не можна проводити інтеграцію всього відразу і за один раз; структурними одиницями управління для релізів повинні бути категорії класів і підсистеми, а не окремі файли і класи.

До позитивних якостей об'єктно-орієнтованих програм відносяться - графік числа виявлених помилок за певний час і щільність помилок. Існує ряд позитивних характеристик, орієнтованих на класи. Об'єктно-орієнтована розробка вимагає іншого інструментарію в порівнянні з традиційними методами. Перехід організації на об'єктно-орієнтовані технології - це зміна світогляду, а не просто вивчення нової мови програмування.

Завданням курсової роботи є вивчення предметної області «Продуктового магазину» та програмна реалізація даної області за допомогою об’єктно-орієнтованого напрямку програмування.

Програма реалізована в середовищі об’єктно-орієнтованого програмування С++ Builder 6.0, на основі мови програмування С++. Це пов’язано з тим, що даний напрям програмування має ряд переваг, необхідних для ведення обліку меблевого складу. Одна з переваг пов’язана з використанням віртуальних функцій, завдяки яким можна заощадити на перевірці типів і опустити багато конструкцій, що здійснюють управління. Згідно досвіду, код об'єктно-орієнтованої програми зазвичай коротше, ніж інших.

Мета курсової роботи – створення об’єктно-орієнтованої програми по веденню обліку продуктового магазину.

Об’єктом курсової роботи є продуктовий магазин, де здійснюється роздрібна торгівля продовольчих товарів, певної кількості та вартості.

Курсова робота розділена на два розділи:

* в Розділі 1 проведено дослідження предметної області «Продуктового магазину» та визначено головні напрямки і дані, що необхідно відобразити в програмі по веденню обліку продуктового магазину.
* в Розділі 2 детально розкрито поняття моделі в об’єктно-орієнтованих мовах програмування, реалізована побудова програмної моделі «Обліку продуктового магазину» та надано обґрунтування вибору мови програмування для створення програми.

**1. Аналіз предметної області**

Продуктовий магазин є кінцевим пунктом збуту товару. Кожний продуктовий магазин має свій асортимент товару для роздрібної торгівлі. В магазині присутні продукти загалом різного виду. До основних належать:

* молочні продукти (молоко, сметана, сир, ряжанка, кефір);
* хлібобулочні (хліб, батон, булочки);
* горілчані (горілка, вино, коняк, портвейн, пиво, слабоалкогольні напої);
* тютюнові вироби;
* кондитерські вироби (печиво, торти, цукерки, то що);
* консервні вироби;
* продукти швидкого приготування;
* інші продукти широкого вжитку.

Продукт циркулює наступним чином:

1. приход товару в магазин – продукти поступають на склад за накладними по приходу;

2. продаж з магазину – відбувається за видатковими чеками.

Продукти зберігаються до їх продажу, чи списання (останнє трапляється відносно періодично, тому що закінчився термін вжитку товару).

Облік продуктового магазину передбачає наступні етапи:

* додавання даних про товар в магазині;
* видалення даних з магазину;
* зберігання даних;
* сортування даних;
* перегляд даних;
* редагування даних.

Кожен з цих етапів є досить важливим і необхідним для ведення обліку. Також актуальним було б передбачати при внесенні даних реєстрацію дати приходу товару на магазин: день, місяць, рік. Для реалізації задуманого необхідним є створення програми, що буде в повному обсязі забезпечувати виконання цих етапів та відображати наступну інформацію:

* вид продукції;
* назва товару;
* ціна (за одиницю товару);
* виробник;
* число, місяць, рік приходу на склад;
* сума всього товару в магазині.

**2. Модель в об’єктно-орієнтованих мовах програмування**

Об'єктно-орієнтована технологія ґрунтується на так званій об'єктній моделі. Основними її принципами є:

* абстрагування;
* інкапсуляція;
* модульність;
* ієрархічність;
* типізація;
* паралелізм і збереження.

Кожен з цих принципів сам по собі не новий, але в об'єктній моделі вони вперше застосовані в сукупності.

Принципи об'єктно-орієнтованого програмування створені, щоб допомогти розробникам застосовувати могутні виразні засоби об'єктного і об'єктно-орієнтованого програмування, що використовує як блоки класи і об'єкти.

Оскільки побудова моделей украй важлива при проектуванні складних систем, об'єктно-орієнтоване проектування пропонує багатий вибір моделей. Об'єктно-орієнтовані моделі проектування відображають ієрархію і класів, і об'єктів системи. Ці моделі покривають весь спектр найважливіших конструкторських рішень, які необхідно розглядати при розробці складної системи, і таким чином надихають на створення проектів, що володіють всіма п'ятьма атрибутами добре організованих складних систем.

Об'єктна модель принципово відрізняється від моделей, які пов'язані з більш традиційними методами структурного аналізу, проектування і програмування. Це не означає, що об'єктна модель вимагає відмови від всіх раніше знайдених і випробуваних часом методів і прийомів. Швидше, вона вносить деякі нові елементи, які додаються до попереднього досвіду. Об'єктний підхід забезпечує ряд істотних переваг, які іншими моделями не передбачалися. Найбільш важливо, що об'єктний підхід дозволяє створювати системи, які задовольняють п'яти ознакам добре структурованих складних систем. Згідно досвіду, є ще п'ять переваг, які дає об'єктна модель.

По-перше, об'єктна модель дозволяє повною мірою використовувати виразні можливості об'єктних і об'єктно-орієнтованих мов програмування. Істотно підвищити ефективність і якість коду можна просто за рахунок використання C++ як "покращуваного C" з елементами абстракції даних. Проте набагато значнішим досягненням є введення ієрархії класів в процесі проектування.

По-друге, використання об'єктного підходу істотно підвищує рівень уніфікації розробки і придатність для повторного використання не тільки програм, але і проектів, що врешті-решт веде до створення середовища розробки. Об'єктно-орієнтовані системи часто виходять компактнішими, ніж їх не об'єктно-орієнтовані еквіваленти. А це означає не тільки зменшення об'єму коду програм, але і здешевлення проекту за рахунок використання попередніх розробок, що дає виграш у вартості і часі.

По-третє, використання об'єктної моделі приводить до побудови систем на основі стабільних проміжних описів, що спрощує процес внесення змін. Це дає системі можливість розвиватися поступово і не приводить до повної її переробки навіть у разі істотних змін початкових вимог.

По-четверте, об'єктна модель зменшує ризик розробки складних систем, перш за все тому, що процес інтеграції розтягується на весь час розробки, а не перетворюється на одноразову подію. Об'єктний підхід складається з ряду добре продуманих етапів проектування, що також зменшує ступінь риску і підвищує упевненість в правильності схвалюваних рішень

Кожна модель при реалізації використовує наступні методи розподілу пам’яті:

* за допомогою визначення змінних стандартних типів (цілі, дійсні та ін);
* за допомогою визначення змінних, визначених користувачем;
* за допомогою списків, стеків, черг, масивів, дерев тощо.

Існують ще декілька способів, наприклад, створення нового екземпляра форми чи елемента управління, але вони не дають великих можливостей при створенні складних структур даних.

Використовуючи ці методи, можна легко будувати статичні структури даних, такі як великі масиви визначених типів. Проте перерозподіл даних може бути досить складним. Наприклад, для того, щоб перемістити елемент з одного кінця масиву на інший, потрібно пере впорядкувати весь масив, зсунувши всі елементи на одну позицію, аж потім можна помістити елемент на нове місце. Крім того, масиви мають фіксований розмір, тому коли наперед не відомо про розмір створюваної програми, їх використання не доречне.

Динамічні структури даних дозволяють швидко і легко виконати такого роду зміни. Всього за кілька кроків можливо перемістити будь-який елемент в структурі даних в інше місце. До таких структур даних належать списки.

Найпростіша форма писку - це група об’єктів, до яких можна звернутись.

Зв’язане зберігання лінійного списку являється списком з двома зв’язками або двухзв’язним списком, якщо кожен елемент, що зберігається має два компонента вказівника (на попередній і на наступний елементи лінійного списку). Графічне зображення двухзв’язного списку представлене на мал.1.



Мал. 1 - Схема зберігання двухзв’язного списку

В залежності від методу доступу до елементів лінійного списку виділяють різновиди лінійних списків - стеки, черги, двохсторонні черги.

Стек являє собою кінцеву послідовність деяких однотипних елементів - скалярних змінних, масивів, структур або об’єднань, серед яких можуть бути й однакові. Він відноситься до динамічних структур даних, його кількість елементів наперед не вказується і в процесі роботи, як правило змінюється.

Допустимими операціями над стеком являються:

* перевірка стека на порожність;
* додавання нового елемента в кінець стека;
* видалення останнього елемента зі стека;
* доступ до останнього елемента.

Таким чином, операції додавання і видалення елемента виконується тільки в кінці списку.

Черга ж являє собою лінійний список, у якого елементи додаються і видаляються з початку списку (як черга в магазині).

Двохстороння черга - лінійний список, у якого операції додавання і видалення елементів та доступ до них можливий як на початку так і в кінці списку.

Використовуючи вказівники можливо побудувати багато інших корисних різновидів зв’язних структур даних, таких як дерева, графи, мережі.

В реалізованій програмі використовуються списки, а точніше - двухзв’язні списки з фіктивним елементом. Їх використання є доцільним, тому що створювана програма не має визначеного розміру, що, наприклад, виключає використання в програмі масивів.

За допомогою списків в програмі проводиться сортування, додавання, видалення та деякі інші операції, що так необхідні при веденні обліку меблевого складу.

**2.1 Програмна модель**

Програма, створена на мові програмування С++, в середовищі програмування Borland С++ Builder 6.0. При розробці моделі програми було створено наступні класи:

* 1. Table
  2. Data
  3. Record
  4. Iterator
  5. Number
  6. MyTable<Number>
  7. MyData
  8. Main

Кожин з цих класів має свою будову. Наприклад, будова класу Table представлена на мал.2.

class Table

protected:

Iterator the\_first()

Iterator the\_last()

Iterator record()

virtual Data\* create()

public:

Table();

virtual ~Table();

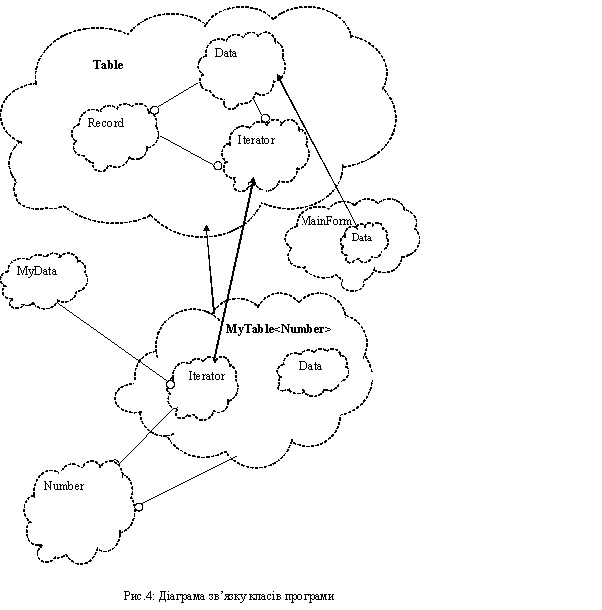
void clear

void save()

void load();

Мал. 2 - Клас Table

Крім цього, кожен клас так чи інакше взаємодіє з іншими класами. Зв’язок класів представлений на мал. 3 у вигляді діаграм Буча.



Мал. 3

З діаграми видно, що кілька класів використовуються одним класом. Наприклад класи Data, Record, Iterator використовуються класом Table, і крім того, використовуються між собою. Стрілочки на діаграмі означають відношення “наслідування” класів, а зв’язок через рисочку, що виходить з кружечка, означає використання одного класу іншим.

Клас зазвичай представляють аморфним об'єктом, на зразок хмарки. Хмарку було взято з матеріалів корпорації Intel, що документувала свою оригінальну об'єктно-орієнтовану архітектуру iAPX432. Форма цього образу натякає на розпливчатість меж абстракції, від яких не очікується гладкості і простоти. Пунктирний контур символізує те, що клієнти оперують зазвичай з екземплярами цього класу, а не з самим класом.

Крім класів, програма складається з об’єктів, а разом вони утворюють модулі. Діаграма модулів показує розподіл класів і об'єктів по модулях у фізичному проектуванні системи.

Кожна окрема діаграма модулів представляє деякий ракурс структури модулів системи.

Діаграма модулів використовується при розробці для того, щоб показати фізичне ділення архітектури програми по шарах і розділах.

Кожен модуль програми по обліку товарів магазину має ім'я - це ім'я відповідного фізичного файлу в каталозі проекту (Main.h і Main.cpp, Data.h i Data.cpp, Number.h i Number.cpp, Table.h та ін.).

Кожне ім'я файлу унікальне в програмі. Сам модуль містить або опис, або визначення класів і об'єктів, а також інші конструкції мови. "Розкривши" значок будь-якого модуля з діаграми, потрапляємо всередину відповідного файлу. Розглядаючи структуру програми з іншого боку, маємо наступну будову: програма містить в собі три форми.

Перша форма програми (MаіnForm) - це форма, що відкривається зразу ж при запуску програми – головна, відображає всі дані про продуктовий магазин. Крім відомостей про товар вона також має в розпорядженні кнопки:

* Додати
* Видалити
* Очистити
* Змінити
* Сортувати
* Зберегти
* Відкрити
* Вихід

Код обробника подій кожної з них наведений в додатку, в лістингу програми.

**2.2 Вибір засобів реалізації програми**

Програма реалізована в середовищі об’єктно-орієнтованого програмування Borland С++ Builder 6.0, на основі мови програмування С++. Це пов’язано з тим, що даний напрям програмування має ряд переваг, необхідних для ведення обліку меблевого складу.

Відомо декілька версій C++. У версії 1.0 реалізовані основні механізми об'єктно-орієнтованого програмування, такі як одиночне наслідування і поліморфізм, перевірка типів і перевантаження функцій. У створеній в 1989 році версії 2.0 знайшли віддзеркалення багато додаткових властивостей (наприклад, множинне спадкоємство), що виникли на базі широкого досвіду застосування мови численним співтовариством користувачів. У версії 3.0 (1990) з'явилися шаблони (класи, що параметризуються) і обробка виключень. Комітет ANSI з C++ (X3J16) недавно схвалив пропозиції по введенню просторів імен (що відповідає нашому позначенню категорій класів) і перевірки типів під час виконання.

Основні характеристики C++ :

* абстракції: змінні екземпляра, методи екземпляра, змінні класу, методи класу;
* інкапсуляція: змінних, методів;
* модульність: різновиди модулів;
* ієрархії: наслідування, шаблони, мета класи;
* типізація: сильна типізація, поліморфізм;
* паралельність: багатозадачність;
* збереженість: довго живучі об’єкти.

Абстрагування є одним з основних методів, використовуваних для вирішення складних завдань. Абстракція виділяє істотні характеристики деякого об'єкту, що відрізняють його від всіх інших видів об'єктів і, таким чином, чітко визначає його концептуальні межі з погляду спостерігача. Абстрагування концентрує увагу на зовнішніх особливостях об'єкту і дозволяє відокремити найістотніші особливості поведінки від неістотних. Вибір правильного набору абстракцій для заданої наочної області є головним завданням об'єктно-орієнтованого проектування.

Абстракція і інкапсуляція доповнюють один одного: абстрагування направлене на спостережувану поведінку об'єкту, а інкапсуляція займається внутрішнім устроєм. Найчастіше інкапсуляція виконується за допомогою утаєння інформації, тобто маскуванням всіх внутрішніх деталей, що не впливають на зовнішню поведінку. Зазвичай ховаються і внутрішня структура об'єкту і реалізація його методів. Інкапсуляція, таким чином, визначає чіткі межі між різними абстракціями.

Інкапсуляція виступає як процес відділення один від одного елементів об'єкту, що визначають його структуру і поведінку; служить для того, щоб ізолювати зобов'язання абстракції від їх реалізації.

Розділення програми на модулі до деякої міри дозволяє зменшити її складність. Правильне розділення програми на модулі є майже таким же складним завданням, як вибір правильного набору абстракцій. Модулі виконують роль фізичних контейнерів, в які поміщаються визначення класів і об'єктів при логічному проектуванні системи. Для невеликих завдань допустимий опис всіх класів і об'єктів в одному модулі. Проте для більшості програм (окрім найтривіальніших) кращим рішенням буде згрупувати в окремий модуль логічно зв'язані класи і об'єкти, залишивши відкритим ті елементи, які абсолютно необхідно бачити іншим модулям.

Ієрархія виступає, як процес впорядкування абстракцій, розташування їх по рівнях. Основними видами ієрархічних структур стосовно складних систем є структура класів (ієрархія "is-a") і структура об'єктів (ієрархія "part of"). Прикладом ієрархії є одиночне наслідування. Іншими словами, наслідування створює таку ієрархію абстракцій, в якій підкласи успадковують будову від одного або декількох батьківських класів.

Типізація являє собою спосіб захиститися від використання об'єктів одного класу замість іншого, або принаймні управляти таким використанням. Типізація примушує виражати абстракції так, щоб мова програмування, використовувана в реалізації, підтримувала дотримання ухвалених проектних рішень. Ідея узгодження типів займає в понятті типізації центральне місце.

В той час, як об'єктно-орієнтоване програмування засноване на абстракції, інкапсуляції і наслідуванні, паралелізм головну увагу приділяє абстрагуванню і синхронізації процесів.

Для об'єктно-орієнтованої розробки виділяють сім різних видів інструментів.

Перший інструмент - система з графічним інтерфейсом, що підтримує об'єктно-орієнтовану систему позначень. Такий інструмент може бути використаний при аналізі, щоб зафіксувати семантику сценаріїв, на ранніх стадіях розробки, щоб передати стратегічні і тактичні рішення, прийняті при проектуванні, а також для координування дій проектувальників. Подібний інструмент буде корисний впродовж всього життєвого циклу і при супроводі системи.

Другий, важливий для об'єктно-орієнтованої розробки інструмент, - браузер, який показує структуру класів і архітектуру модулів системи. Ієрархія класів може зробитися настільки складною, що важко навіть відшукати всі абстракції, які були введені при проектуванні. При вивченні фрагмента програми розробникові може знадобитися подивитися визначення класу деякого об'єкту. Знайшовши цей клас, йому ймовірно доведеться заглянути в опис якого-небудь з його суперкласів. З цієї причини браузер виявляється дуже важливим інструментом об'єктно-орієнтованого аналізу і проектування.

Третій вид інструментів, який дуже важливий - інкрементний компілятор. Метод еволюційної розробки, який застосовується в об'єктно-орієнтованому програмуванні, потребує компілятора, який міг би компілювати окремі оголошення і операторів. Для швидкої відладки методи і визначення полів повинні компілюватися інкрементно.

У категорію налагоджувальних засобів включається і такі інструменти, як стресові тестери, що випробовують програми в критичних умовах обмеженості ресурсів, і інструменти для аналізу пам'яті, що розпізнають порушення доступу до пам'яті (запис в недозволені ділянки пам'яті, читання з неініціалізованих ділянок, читання або запис за межами масиву).

Для великих проектів потрібні інструменти управління конфігурацією і контролю версій. Для управління конфігурацією кращими одиницями є категорії класів і підсистеми.

Інший інструмент, який є також важливим для об'єктно-орієнтованої розробки, - це бібліотекар класів. Бібліотека в С++ швидко розростається до таких розмірів, що розробники не можуть відшукати клас, що знадобився. Одна з причин швидкого зростання бібліотеки полягає в тому, що клас може мати декілька реалізацій з різними тимчасовими і просторовими семантиками.

Ще один тип інструменту - генератор графічного інтерфейсу користувача. Для систем, в яких великий об'єм взаємодії з користувачем, краще мати спеціальний інструмент для інтерактивного створення діалогів і вікон, чим програмувати все з нуля. Код, що з генерований такою системою, може бути потім пов'язаний з рештою об'єктно-орієнтованої системи і, якщо необхідно, вручну відкоригований.

Вибір даної мови програмування зумовлений через її переваги:

* велика конкурентоспроможність завдяки передбаченості, скороченню часу на розробку і великої гнучкості продукту;
* завдання, що розробляються, можуть бути настільки складними, що не залишається альтернативних рішень.

Вивчення численних випадків з практики підкріплює ці висновки; особливо часто указується на те, що об'єктний підхід може скоротити час розробки і розмір коду.

Дивлячись на тіньову сторону об'єктно-орієнтованої технології (недоліки), потрібно розглядати два питання: продуктивність і початкові витрати. В порівнянні з процедурними мовами, об'єктно-орієнтовані мови ясно вносять додаткові накладні витрати на пересилку повідомлення від одного об'єкту іншому. У мовах, що строго типізуються, компілятор часто може визначати, які виклики можуть бути зв'язані статично і згенерувати для них виклик процедури замість динамічного пошуку.

Інша причина зниження продуктивності криється не стільки в природі об'єктно-орієнтованих мов, скільки в способі їх використання в процесі об'єктно-орієнтованої розробки. Об'єктно-орієнтована технологія породжує багатошарові системи абстракцій. Один із наслідків цього розшарування в тому, що кожен метод виявляється дуже маленьким, оскільки він будується на методах нижнього рівня.

Інший наслідок розшарування: іноді методи служать лише для того, щоб дістати доступ до захищених атрибутів об'єкту. В результаті відбуваються дуже багато викликів. З позитивного боку таке шарування сприяє розумінню системи; до деяких складних систем неможливо навіть підступитися, якщо не почати з проектування шарів.

**Висновки**

В процесі роботи було створено програму - «Облік продуктового магазину». Програма реалізована в середовищі об’єктно-орієнтованого програмування Borland С++ Builder 6.0, на основі мови програмування С++.

Було досліджено предметну область - «Продуктовий магазин», що внесло багато ясності для розробки програми.

Детальний розгляд та створення програмної моделі дало можливість більш детально спроектувати створену програму, розбивши її на модулі так, що прискорило швидкість її створення та внесло ясність на деякі використовувані аспекти програми.

Створена програма «Облік продуктового магазину» в процесі доопрацювання може бути використана за основу для створення більш складної та великої бази даних по роботі в даній предметній області.

**Список використаних джерел**

1. Гради Буч Объектно-ориентированный анализ и проектированиес примерами приложений на С++. Второе издание Rational Санта-Клара, Калифорния, перевод с английского под редакцией И. Романовского и Ф. Андреева.
2. Павловская Т.А. С++. Объектно-ориентированное программирование: практикум. СПб: Питер, 2005.
3. Б.С. Хусаинов. Структуры и алгоритмы обработки данных. – Примеры на языке Си: Учеб. Пособие, 2004.
4. Кнут, Дональд, Эрвин. Искуство програмирования в 3-х т. М.: Вильянс, 2003.

5. Катренко А.В. Системний аналіз об’єктів та процесів комп’ютеризації. Львів: Новий світ-2000, 2007.

6. Б.С. Хусаинов. Структуры и алгоритмы обработки данных. – Примеры на языке Си: Учеб. Пособие. Финансы и статистика, 2004.

7. Гайдаржи В.І. Основи проектування та використання баз даних. К.: Політехніка, ТОВ фірма Періодика, 2004.

8. Малевич Б.Я. Теория алгоритмов. М.: Машиностроение, 2004.

9. Прокудин Г.С. Основы программирования и алгоритмические язики. К.: Изд-во Европейського университета, 2003.

10. Озеров В. Советы по Delphi. Советы програмистов. СПб.: Символ-плюс,2002.

**Додаток А**

**Результат роботи програми «Облік продуктового магазину»**

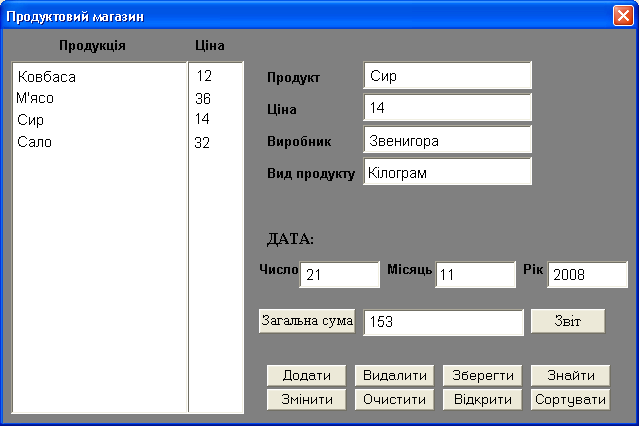


Рис. 1 - Форма обліку продуктового магазину

**Додаток Б**

**Лістинг програми**

1. **Table.cpp**

#include <vcl.h>

#pragma hdrstop

#include "Table.h"

Table::Table()

{

list = new Record;

list->next = list->prev = list;

list->id = 0;

ID = 0;

}

Table::~Table()

{

Record \*t,\*q;

for(t=list->next;t!=list;t=q)

{

q = t->next;

delete t->data;

delete t;

}

delete list;

}

Table: Iterator Table: record()

{

Record \*r = new Record;

ID++;

r->id = ID;

r->data = create();

r->next = list->next;

r->prev = list;

list->next->prev = r;

list->next = r;

return Iterator(r,this);

}

void Table::clear()

{

Record \*t,\*q;

for(t=list->next;t!=list;t=q)

{

q = t->next;

delete t->data;

delete t;

}

list->next = list->prev = list;

ID = 0;

}

void Table::save(FILE \*f)

{

Fprintf (f,"%d\n", ID);

for(Record \*t=list->next;t!=list;t=t->next)

{

fprintf(f,"%7d ",t->id);

t->data->save(f);

fprintf(f,"\n");

}

Fprintf (f,"END");

}

void Table::zvit(FILE \*f)

{

fprintf(f,"Ціна Назва Виробник Вид Дата\n");

fprintf(f,"-----------------------------------------------\n");

for(Record \*t=list->next;t!=list;t=t->next)

{

t->data->zvit(f);

}

}

void Table::load(FILE \*f)

{

int id;

Record \*t;

int rez = fscanf(f,"%d\n",&ID);

if((rez!=EOF)&&(rez!=0))

{

while(true)

{

t = new Record;

rez = fscanf(f,"%d",&t->id);

if((rez==EOF)||(rez==0))

{

delete t;

break;

}

t->data = create();

rez = t->data->load(f);

if((rez==EOF)||(rez==0))

{

delete t;

break;

}

t->next = list;

t->prev = list->prev;

list->prev->next = t;

list->prev = t;

}

}

}

void Table::Iterator::remove()

{

r->next->prev = r->prev;

r->prev->next = r->next;

delete r->data;

delete r;

}

void Table::Iterator::operator >>(Iterator &p)

{

if((t==p.t)&&(p.r!=r)&&(p.r!=r->next)&&(p.r!=t->list))

{

p.r->next->prev = p.r->prev;

p.r->prev->next = p.r->next;

p.r->next = r->next;

p.r->prev = r;

r->next->prev = p.r;

r->next = p.r;

}

}

void Table::Iterator::operator <<(Iterator &p)

{

if((t==p.t)&&(p.r!=r)&&(p.r!=r->prev)&&(p.r!=t->list))

{

p.r->next->prev = p.r->prev;

p.r->prev->next = p.r->next;

p.r->next = r;

p.r->prev = r->prev;

r->prev->next = p.r;

r->prev = p.r;

}

}

1. **Table.h**

#ifndef TableH

#define TableH

#include <stdlib.h>

#include <stdio.h>

#include "Date.h"

#include "My\_date.h"

class Table

{

public:

class Data

{

public: virtual ~Data(){}

virtual void save(FILE \*f){}

virtual void zvit(FILE \*f) {}

virtual int load(FILE \*f){return 1;}

};

private:

struct Record

{

int id;

Data \*data;

Record \*next;

Record \*prev;

} \*list;

int ID;

public:

class Iterator;

friend class Iterator;

class Iterator

{

friend class Table;

Record \*r;

Table \*t;

protected:

Iterator(Record \*R,Table \*T):r(R),t(T){}

Data\* data() {return r->data;}

Iterator the\_next() {return Iterator(r->next,t);}

Iterator the\_prev() {return Iterator(r->prev,t);}

public: Iterator(){}

int id(){return r->id;}// повертає номер запису в таблиці

void operator++() {r = r->next;}// переміщує ітератор на наступний запис

void operator--() {r = r->prev;}// переміщує ітератор на попередній запис

bool operator==(const Iterator &p){return r==p.r;}// порівнює з іншим ітератором

bool operator!=(const Iterator &p){return r!=p.r;}// порівнює з іншим ітератором

bool valid() {return r!=t->list;}// true - вказує на дійсний елемент;false - вказує на нульовий елемент

void operator>>(Iterator &p);// перемістити p після даного ітератора

void operator<<(Iterator &p);// перемістити p до даного ітератора

void remove();// видалити запис з таблиці

};

protected:

Iterator the\_first() {return Iterator(list->next,this);}

Iterator the\_last() {return Iterator(list->prev,this);}

Iterator record();

virtual Data\* create(){return new Data;};

public: Table();

virtual ~Table();

void clear();// очистити таблицю

void save(FILE \*f);// зберегти таблицю

void zvit(FILE \*f);

void load(FILE \*f);// завантажити таблицю

};

template<class MyData> class MyTable : public Table

{

public:

class Iterator : public Table::Iterator

{

friend class MyTable;

Iterator(const Table::Iterator &p):Table::Iterator(p){}

public: Iterator(){}

MyData& operator\*() {return \*(MyData\*)data();}// доступ до полів MyData

MyData\* operator->() {return (MyData\*)data();}// доступ до полів MyData

Iterator next() {return Iterator(the\_next());}// повертає ітератор на наступний елемент

Iterator prev() {return Iterator(the\_prev());}// повертає ітератор на попередній елемент

};

protected:

virtual Data\* create(){return new MyData;}

public: Iterator record() {return Iterator(Table::record());}// створює новий запис і повертає ітератор на нього

Iterator first() {return Iterator(the\_first());}// повертає ітератор на перший запис в таблиці

Iterator last() {return Iterator(the\_last());}// повертає ітератор на останній запис в таблиці

};

1. **Number.cpp**

#include <vcl.h>

#pragma hdrstop

#include "Number.h"

void Number::save(FILE \*f)

{char s[1000];

char s1[1000];

char s2[1000];

sprintf(s,"%s",str.c\_str());

sprintf(s1,"%s",str1.c\_str());

sprintf(s2,"%s",str2.c\_str());

for(int i=0;s[i];i++) if(s[i]==' ') s[i]='\_';

fprintf(f,"%d %s %s %s %d %d %d ",d,s,s1,s2,day,month,year);

}

void Number::zvit(FILE \*f)

{

char s[1000];

char s1[1000];

char s2[1000];

sprintf(s,"%s",str.c\_str());

sprintf(s1,"%s",str1.c\_str());

sprintf(s2,"%s",str2.c\_str());

fprintf(f,"%5d %10s %10s %10s %2d %2d %4d ",d,s,s1,s2,day,month,year);

fprintf(f,"\n");

}

int Number::load(FILE \*f)

{ char s[1000];

char s1[1000];

char s2[1000];

int rez = fscanf(f,"%d %s %s %s %d %d %d",&d,s,s1,s2,&day,&month,&year);

if(rez==7)

{ for(int i=0;s[i];i++) if(s[i]=='\_') s[i]=' ';

for(int i=0;s1[i];i++) if(s1[i]=='\_') s1[i]=' ';

for(int i=0;s2[i];i++) if(s2[i]=='\_') s2[i]=' ';

str = AnsiString(s);

str1 = AnsiString(s1);

str2 = AnsiString(s2);

}

return rez;

}

1. **Number.h**

#ifndef NumberH

#define NumberH

#include "Table.h"

#include "Date.h"

#include "My\_date.h"

class Number : public Table::Data

{

public: int d;

int day; // число випуска

int year; // год випуска

int month; // месяц випуска

AnsiString str; // название мебели

AnsiString str1; // производитель

AnsiString str2; // вид мебели

void save(FILE \*f);

void zvit(FILE \*f);

int load(FILE \*f);// 0 - помилка завантаження; EOF - кінець файла; >0 - завантаження успішне

};

**5. Date.cpp**

#include <vcl.h>

#pragma hdrstop

#include "Date.h"

bool Date::load(int d,int m,int y)

{ if(y>=1900)

{ switch(m)

{

case 1:// січень 31 день

case 3:// березень 31 день

case 5:// травень 31 день

case 7:// липень 31 день

case 8:// серпень 31 день

case 10:// жовтень 31 день

case 12:// грудень 31 день

if((d<1)||(d>31)) return false;

break;

case 2:// лютий 28-29 день

if(y%4)

{

if((d<1)||(d>29)) return false;

}

else

{

if((d<1)||(d>28)) return false;

}

break;

case 4:// квітень 30 день

case 6:// червень 30 день

case 9:// вересень 30 день

case 11:// листопад 30 день

if((d<1)||(d>30)) return false;

break;

default: return false;

}

Day = d;

Month = m;

Year = y;

X = d + m\*100 + year\*10000;

Name = StrToInt(d)+"."+StrToInt(m)+"."+StrToInt(y);

}

else return false;

}

bool Date::load(const AnsiString &s)

{

int x = s.Pos(".");

if(x)

{

AnsiString str = s.SubString(1,x-1);

int d = StrToInt(str);

str = s.SubString(x+1,s.Lenth-x);

int y = str.Pos(".");

if(y)

{

str = s.SubString(x+1,y+x-1);

int m = StrToInt(str);

str = s.SubString(y+x+1,s.Lenth-x-y);

int y = StrToInt(str);

return load(d,m,y);

}

else return false;

}

else return false;

}

**6. Date.h**

#ifndef DateH

#define DateH

class Date

{ int Day;

int Month;

int Year;

long X;

AnsiString Name;

public: bool load(int d,int m,int y);

bool load(const AnsiString &s);

int day(){return Day;}

int month(){return Month;}

int year(){return Year;}

const char& str(){return Str.c\_str();}

const Date& operator=(const Date &D){Day = D.Day; Month = D.Month; Year = D.Year;}

bool operator>(const Date &D){return X>D.X;}

bool operator>=(const Date &D){return X>=D.X;}

bool operator<(const Date &D){return X<D.X;}

bool operator<=(const Date &D){return X<=D.X;}

bool operator==(const Date &D){return X==D.X;}

bool operator!=(const Date &D){return X==D.X;}

};

**7. Main.cpp**

#include <vcl.h>

#pragma hdrstop

#include "Main.h"

#include "My\_date.h"

#include "Date.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TMainForm \*MainForm;

\_\_fastcall TMainForm::TMainForm(TComponent\* Owner)

: TForm(Owner)

{

}

//------------------------------------------------------------------------void \_\_fastcall TMainForm::Add(TObject \*Sender)

{

Ptr p = list.record();

p->d = StrToInt(NewInt->Text);

p->day = StrToInt(Day->Text);

p->year = StrToInt(Year->Text);

p->month = StrToInt(Month->Text);

//p->x = StrToFloat(NewDouble->Text);

p->str = String->Text;

p->str1 = FirmaString->Text;

p->str2 = WidString->Text;

redraw();

}

void TMainForm::redraw()

{ List->Clear();

StrList->Clear();

for(Ptr p = list.first();p.valid();++p)

{

List->Items->Add(IntToStr(p->d));

StrList->Items->Add(p->str);

}

}

//------------------------------------------------------------------------void \_\_fastcall TMainForm::Del(TObject \*Sender)

{ int n = List->ItemIndex;

int i;

Ptr p;

for(i=0,p=list.first();p.valid()&&(i<n);++i,++p);

if((i==n)&&p.valid())

{

p.remove();

redraw();

}

}

//------------------------------------------------------------------------void \_\_fastcall TMainForm::Find(TObject \*Sender)

{ int x = StrToInt(NewInt->Text);

Ptr p;

int i;

for(i=0,p=list.first();p.valid()&&(p->d!=x);++p,++i);

if(p.valid())

{ List->ItemIndex = i;

StrList->ItemIndex = i;

NewInt->Text = IntToStr(p->d);

Day->Text = IntToStr(p->day);

Year->Text = IntToStr(p->year);

Month->Text = IntToStr(p->month);

//NewDouble->Text = FloatToStr(p->x);

String->Text = p->str;

FirmaString->Text = p->str1;

WidString->Text = p->str2;

}

else List->ItemIndex = -1;

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::Sort(TObject \*Sender)

{

Ptr t,p,q;

for(t=list.first().next();t.valid();t=q)

{

q = t.next();

for(p=t.prev();p.valid()&&(p->d>t->d);--p);

p>>t;

}

redraw();

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::Save(TObject \*Sender)

{

FILE \*f;

if(SaveDialog->Execute())

{

f = fopen(SaveDialog->FileName.c\_str(),"w");

if(f!=NULL)

{

list.save(f);

fclose(f);

}

}

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::ZvitClick(TObject \*Sender)

{

FILE \*f;

if(SaveDialog->Execute())

{

f= fopen(SaveDialog->FileName.c\_str(),"w");

if(f!=NULL)

{

list.zvit(f);

fclose(f);

}

}

}

void \_\_fastcall TMainForm::Open(TObject \*Sender)

{

FILE \*f;

if(OpenDialog->Execute())

{

f = fopen(OpenDialog->FileName.c\_str(),"r");

if(f!=NULL)

{

list.clear();

String->Text="";

NewInt->Text="";

list.load(f);

redraw();

fclose(f);

}

}

}

//------------------------------------------------------------------------void \_\_fastcall TMainForm::Clear(TObject \*Sender)

{

list.clear();

redraw();

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::Select(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

TPoint point;

point.x = X;

point.y = Y;

int sel = List->ItemAtPos(point,true);

int i;

Ptr t;

if(sel>=0)

{

List->ItemIndex = sel;

StrList->ItemIndex = sel;

for(t=list.first(),i=0;t.valid()&&(i<sel);++i,++t);

if(t.valid())

{

NewInt->Text = IntToStr(t->d);

Day->Text = IntToStr(t->day);

Year->Text = IntToStr(t->year);

Month->Text = IntToStr(t->month);

//NewDouble->Text = FloatToStr(t->x);

String->Text = t->str;

FirmaString->Text = t->str1;

WidString->Text = t->str2;

}

}

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::Update(TObject \*Sender)

{

Ptr t;

int i;

int sel;

sel = List->ItemIndex;

if(sel>=0)

{ for(t=list.first(),i=0;t.valid()&&(i<sel);++i,++t);

if(t.valid())

{

t->d = StrToInt(NewInt->Text);

t->day = StrToInt(Day->Text);

t->year = StrToInt(Year->Text);

t->month = StrToInt(Month->Text);

t->str = String->Text;

t->str1 = FirmaString->Text;

t->str2 = WidString->Text;

//t->x = StrToFloat(NewDouble->Text);

redraw();

}

}

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::SelectStr(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

TPoint point;

point.x = X;

point.y = Y;

int sel = StrList->ItemAtPos(point,true);

int i;

Ptr t;

if(sel>=0)

{

List->ItemIndex = sel;

StrList->ItemIndex = sel;

or(t=list.first(),i=0;t.valid()&&(i<sel);++i,++t);

if(t.valid())

{

NewInt->Text = IntToStr(t->d);

Day->Text = IntToStr(t->day);

Year->Text = IntToStr(t->year);

Month->Text = IntToStr(t->month);

String->Text = t->str;

FirmaString->Text = t->str1;

WidString->Text = t->str2;

}

}

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::SortNum(TObject \*Sender)

{

Ptr t,p,q;

for(t=list.first().next();t.valid();t=q)

{

q = t.next();

for(p=t.prev();p.valid()&&(p->d>t->d);--p);

p>>t;

}

redraw();

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::SortStr(TObject \*Sender)

{

Ptr t,p,q;

for(t=list.first().next();t.valid();t=q)

{

q = t.next();

for(p=t.prev();p.valid()&&(p->str>t->str);--p);

p>>t;

}

redraw();

}

//------------------------------------------------------------------------

void \_\_fastcall TMainForm::Button1Click(TObject \*Sender)

{

Close();

}

1. **Main.h**

#ifndef MainH

#define MainH

#include <Classes.hpp>

#include <Controls.hpp>

#include <StdCtrls.hpp>

#include <Forms.hpp>

#include "Number.h"

#include <Dialogs.hpp>

#include <ComCtrls.hpp>

#include <ExtCtrls.hpp>

#include <jpeg.hpp>

#include "Date.h"

#include "My\_date.h"

//------------------------------------------------------------------------

typedef MyTable<Number> Data;

typedef MyTable<Number>::Iterator Ptr;

class TMainForm : public TForm

{

\_\_published:// IDE-managed Components

TEdit \*NewInt;

TLabel \*NewIntLabel;

TButton \*AddButton;

TListBox \*List;

TButton \*DelButton;

TButton \*FindButton;

TButton \*SortButton;

TButton \*SaveButton;

TButton \*OpenButton;

TButton \*Button3;

TButton \*Zvit;

TSaveDialog \*SaveDialog;

TOpenDialog \*OpenDialog;

TEdit \*String;

TLabel \*Label1;

TButton \*ChangeButton;

TListBox \*StrList;

TLabel \*NumLabel;

TLabel \*StrLabel;

TEdit \*FirmaString;

TEdit \*WidString;

TLabel \*Label3;

TLabel \*Label4;

TLabel \*Label5;

TLabel \*Label6;

TLabel \*Label7;

TLabel \*Label8;

TEdit \*Day;

TEdit \*Month;

TEdit \*Year;

TButton \*Button1;

TImage \*Image1;

void \_\_fastcall Add(TObject \*Sender);

void \_\_fastcall Del(TObject \*Sender);

void \_\_fastcall Find(TObject \*Sender);

void \_\_fastcall Sort(TObject \*Sender);

void \_\_fastcall Save(TObject \*Sender);

oid \_\_fastcall Open(TObject \*Sender);

void \_\_fastcall Clear(TObject \*Sender);

void \_\_fastcall Select(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y);

void \_\_fastcall Update(TObject \*Sender);

void \_\_fastcall SelectStr(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y);

void \_\_fastcall SortNum(TObject \*Sender);

void \_\_fastcall SortStr(TObject \*Sender);

void \_\_fastcall Button1Click(TObject \*Sender);

private:// User declarations

Data list;

void redraw();

public:// User declarations

\_\_fastcall TMainForm(TComponent\* Owner);

};

**9. Magazin.cpp**

#include <vcl.h>

#include "Date.h"

#pragma hdrstop

USERES("Magazin.res");

USEFORM("Main.cpp", MainForm);

USEUNIT("Table.cpp");

USEUNIT("Number.cpp");

USEUNIT("Date.cpp");

USEFILE("Date.h");

USEUNIT("My\_date.cpp");

USEUNIT("data.cpp");

//------------------------------------------------------------------------WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

try

{

Application->Initialize();

Application->CreateForm(\_\_classid(TMainForm), &MainForm);

&ProgramForm);

Application->Run();

}

catch (Exception &exception)

{

Application->ShowException(&exception);

}

return 0;

}