МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

Кафедра інформаційних технологій

**КУРСОВА РОБОТА**

з об’єктно-орієнтованого програмування

на тему: «Реалізувати класи РЯДОК СИМВОЛІВ, ВЕЛИКІ ЦІЛІ ЧИСЛА, МАСИВ ВЕЛИКИХ ЧИСЕЛ»

Студента 2-го курсу КН-2 групи

напряму підготовки*«Комп’ютерні науки»*

спеціальності\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_*Манжула Д.*\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник: \_\_\_*зав. каф. ІТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_\_\_\_\_\_\_\_\_\_*д.т.н., проф.СоколовськийЯ.І.\_*

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

м. Львів – 2018 рік

Зміст

[Вступ 5](#_Toc482829852)

[1 Специфікація роботи 6](#_Toc482829853)

[1.1. Основи ООП 6](#_Toc482829854)

[1.2. Створення класів: поля, методи 8](#_Toc482829855)

[1.1 Перевантаження операторів 9](#_Toc482829856)

[1.2 Контейнери 10](#_Toc482829857)

[1.3 Ітератори 11](#_Toc482829858)

[2 Програмна документація 12](#_Toc482829859)

[2.1 Проектування структури програми 12](#_Toc482829860)

[2.2 Проектування інтерфейсу користувача 17](#_Toc482829861)

[2.2.1 Консольний інтерфейс 17](#_Toc482829862)

[2.2.2 Графічний інтерфейс з використанням Windows Forms 18](#_Toc482829863)

[2.3 Розроблення консольної версії програми на мові C++ 19](#_Toc482829864)

[2.4 Розроблення програми з графічним інтерфейсом на C# 24](#_Toc482829865)

[2.5 Приклад та аналіз результатів виконання програми з консольним інтерфейсом 29](#_Toc482829866)

[2.6 Приклад та аналіз результатів виконання програми з графічним інтерфейсом на Windows Forms 36](#_Toc482829867)

[Висновки 40](#_Toc482829868)

[Список використаних джерел 41](#_Toc482829869)

[Додатки 42](#_Toc482829870)

[Додаток А. Діаграми класів 42](#_Toc482829871)

[Додаток Б. Код консольної реалізації програми на мові C++ 46](#_Toc482829872)

[Додаток В. Код реалізації програми з графічним інтерфейсом з використанням Windows Forms на мові C# 75](#_Toc482829873)

# Вступ

Об’єктно-орієнтоване програмування (ООП) – це еволюційний крок, який випливає із розвитку програмування. ООП дає нам можливість відчути себе не тільки програмістом, а й архітектором, проектуючи структуру програми, створюючи красиві форми. Створення класів за допомогою ООП можна порівняти з будуванням будинку. Спочатку ми створюємо план – описуємо клас, а пізніше будуємо будинок – створюємо об’єкт, тобто сутність класу.

Ціль курсової роботи продемонструвати основи ООП на прикладі програми розробки класу МАСИВ ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ, що містить масив об'єктів класу ВЕЛИКІ ЦІЛІ ЧИСЛА.

Без комп’ютерних технологій неможливе сьогодення. З інтенсивним розвитком програмування постає задача структурування інформації і при потребі її швидкого пошуку і редагування.

У програмі створено клас РЯДОК СИМВОЛІВ з полями даних: вказівник на початок рядка у динамічній пам’яті, максимальна довжина рядка, який є базовим для класу ВЕЛИКІ ЦІЛІ ЧИСЛА. Створено контейнерний клас МАСИВ ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ, що містить масив об’єктів класу ВЕЛИКІ ЦІЛІ ЧИСЛА. В контейнерному класі реалізовано метод знаходження суми елементів масиву. Створено два програмних застосунки, що реалізують поставлене завдання. Перший програмний застосунок має консольний інтерфейс і розроблений на мові програмування C++. Другий програмний застосунок має графічний інтерфейс на мові програмування C# з використанням Windows Forms засобами Visual Studio 2015.

# Специфікація роботи

Для розробки програмного забезпечення відповідно до поставленого завдання необхідно знати основи об’єктно-орієнтованого програмування, його основні парадигми (наслідування, поліморфізм та інкапсуляція) та володіти такими поняттями, як об’єкт, клас, конструктори та деструктори, перевантаження методів та операторів, віртуальність, контейнерні класи, ітератори та ін. Коротко опишемо ці поняття та методи їх застосування.

## Основи ООП

З розвитком програмування виникла ідея поєднати в межах однієї сутності дані і код, що безпосередньо опрацьовує ці дані. Така сутність отримала назву об’єкт, а відповідний підхід до створення програм називають об’єктно-орієнтованимпрограмуванням.

Об’єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка розглядає програму як сукупність гнучко пов’язаних між собою об’єктів. Кожен об’єкт має суттєві характеристики, які відрізняють його від усіх інших об’єктів. Сукупність таких характеристик називається абстракцією. Розрізняють абстракції стану та поведінки об’єкта.

Стан (дані об’єкта) характеризується переліком та значенням певних ознак. Поведінка (функціонал об’єкта) визначається набором операцій, які виконуються об’єктом, або над об’єктом.

Кожен об’єкт є екземпляром (представником) певного класу. Відповідно, клас – це відповідна абстракція об’єктів.

Основні переваги концепції ООП:

* моделювання предметів та явищ реального світу;
* можливість створювати користувацькі типи даних (класи);
* приховування деталей реалізації (інкапсуляція);
* можливість повторного використання коду (наслідування);
* інтерпретація викликів процедур та функцій на етапі виконання (поліморфізм).

**Інкапсулювання**– це механізм в програмуванні, який пов’язує в одне ціле функції і дані, якими вони маніпулюють, а також захищає їх від зовнішнього доступу і неправильного застосування. В об’єктно-орієнтованій мові функції і всі необхідні дані можуть пов’язуватись таким способом, що створюється автономна структура – об’єкт. Іншими словами, об’єктом є структура, яка підтримує інкапсулювання. В межах об’єкта функції, дані або і функції і дані можуть бути або закритими для інших об’єктів (***private***), або відкритими (***public*).**

Модифікатори доступу:

Модифікатор ***private*–** забороняє доступ до полів і методів класу ззовні самого класу. Поля і методи, оголошені з модифікатором ***private***, будуть доступні тільки в межах самого класу.

Якщо поле і метод класу визначені з модифікатором ***public*,** вони доступні ззовні оголошення базового класу чи похідних класів.Це в деякій мірі означає, що метод оголошений із зовнішнього класу може вільно звертатись до таких полів і методів. З допомогою модифікатора ***protected*** можна доступитись до полів і методів базового класу тільки для похідних класів, але не для зовнішніх по відношенню до класу методів.

**Успадкування** – це властивість, з допомогою якої один об’єкт може набувати властивостей іншого. При цьому підтримується концепція ієрархічної класифікації. Без використання успадкування кожний об’єкт повинен явно визначати всі свої характеристики; використовуючи наслідування, об’єкт повинен визначати тільки ті якості, які роблять його унікальним в межах свого класу. Визначення нового класу (породжений клас,підклас,[англ.](http://uk.wikipedia.org/wiki/%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)derived class,[англ.](http://uk.wikipedia.org/wiki/%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)subclass) може базуватись на визначенні вже існуючого (базовий клас,[англ.](http://uk.wikipedia.org/wiki/%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)base class,[англ.](http://uk.wikipedia.org/wiki/%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)superclass). В такому випадку, новий клас отримає властивості та поведінку базового класу, та доповнить їх своїми власними. У випадку одиничного успадкування, у кожного класу може бути лише один безпосередній базовий клас. У випадку множинного успадкування, дозволяється існування декількох безпосередніх надкласів. Застосування методів успадкування дозволяє покращити повторне використання коду шляхом використання вже визначених [методів](http://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%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))(поведінки) базових класів.

Разом з інкапсуляцією і успадкуванням **поліморфізм** також являє собою одну із важливих концепцій ООП. Застосування цієї концепції дозволяє значно полегшити розробку складних програм.

Термін поліморфізм має грецьке походження і означає «наявність багатьох форм». З поліморфізмом тісно пов’язані такі поняття,як абстрактні класи, віртуальні методи, перевантаження методів і властивостей.

Віртуальні методи – один із найважливіших прийомів реалізації поліморфізму. Вони дозволяють створювати загальний код, який може працювати разом з об’єктами базового класу, так і з об’єктами будь-якого його класу-нащадка. При цьому базовий клас визначає спосіб роботи з об’єктами і будь-які його нащадки можуть представляти конкретну реалізацію цього способу. Базовий клас може і не представляти реалізацію віртуального методу, а тільки оголошувати про його наявність. Такі методи без реалізації називаються абстрактними. Клас, який містить хоча б один такий метод називається абстрактним. Об’єкт такого класу створювати не можна. Нащадки абстрактного класу повинні обов’язково представити реалізацію для всіх його абстрактних методів, інакше, вони в свою чергу будуть абстрактними. Можливість присвоювати різну функціональність одному методу називається перевантаженням методу.

## Створення класів: поля, методи

Клас є основою для створення об’єктів. В класі визначаються дані і функціонал, який працює з цими даними. Об’єкти являються екземплярами класу. Ініціалізація змінних екземпляру відбувається в конструкторі. В класі можуть бути декілька конструкторів,тобто клас являється набором проектів, які визначають як будувати об’єкт. Дуже важливо розуміти різницю між класом і об’єктом: клас являється логічною абстракцією до тих пір,поки не буде створений об’єкт і не появиться фізична реалізація цього класу в пам’яті комп’ютера. Методи і змінні, які складають клас називаються членами класу. При визначенні класу об’являються дані, які він містить, і функціонал працюючий з цими даними. При створенні класу спочатку вказуємо ключове слово class. Найпростіші класи можуть мати тільки функціонал або тільки дані, але в реальних програмах класи включають в себе обидві складові. Дані містяться в змінних екземпляра, які визначені класом, а функціонал міститься в методах. Важливо відзначити, що в С++ визначені декілька специфічних різновидів членів класу: змінні екземпляра, статичні змінні, константи, методи, конструктори, деструктори, події, оператори і властивості.

**Методи** – це підпрограми, які управляють даними, визначені в класі, і в багатьох випадках забезпечують доступ до даних. За допомогою методів здійснюється взаємодія зовнішнього середовища з класом. Метод містить один і більше операторів. В професійно написаному С++-коді кожний метод виконує тільки одну задачу. Назвою метода може бути будь-який існуючий ідентифікатор. Ключові слова не можуть бути іменами методів. При оголошенні кожної змінної і методу можна встановлювати певний модифікатор доступу (за замовчуванням private). **Поле** – це член даних, який містить екземпляр значущого типу або посилання на вказівниковий тип. Загальномовне середовище (CLR) підтримує поля, які належать, як класу (статичні), так і екземпляру (нестатичні). Динамічна пам'ять для зберігання статичного поля виділяється в межах об’єкта класу, який створюється при його завантаженні в домен, що відбувається при компілюванні будь-якого методу, який посилається на цей тип. Динамічна пам'ять для збереження екземплярних полів виділяється при створенні екземпляра даного типу.

Одним із «спеціальних типів» членів класу є методи доступу, що забезпечують доступ до поля. В багатьох програмах виникає необхідність створення поля, хоч і доступного із інших об’єктів, але такого, для якого виконувався б контроль над операціями з його значенням. Наприклад, може бути потрібно обмеження діапазону значень, які присвоюються цьому полю. Для досягнення даної цілі ми використаємо закриту змінну в поєднанні з методом, який забезпечує доступ до її значення. Методи доступу використовуються для отримання значення і присвоєння його змінній.

## Перевантаження операторів

У мові C++ можна перенавантажувати більшість операторів, налаштувавши їх на конкретний клас. Наприклад, в класі, що підтримує стек, оператор "+" можна перевантажити для заштовхування елементів в стек, а оператор "-" — для виштовхування елементів. Перевантажений оператор зберігає своє первинне призначення. Просто набір типів, до яких його можна застосовувати,розширюється.

Перевантаження операторів — одна з найефективніших можливостей мови С++. Вона дозволяє повністю інтегрувати нові класи в існуюче програмне середовище. Після перевантаження операції над об'єктами нових класів виглядають точно так само, як операції над змінними вбудованих типів. Крім того, перевантаження операторів лежить в основі системи введення висновку вмові C++.

Перевантаження операторів здійснюється за допомогою операторних функцій (operator function), які визначають дії перевантажених операторів застосовні до відповідного класу. Операторні функції створюються за допомогою ключового слова operator. Операторні функції можуть бути як членами класу, так і звичайними функціями. Проте звичайні операторні функції, як правило, об’являються дружніми по відношенню до класу, для якого вони перевантажуються. У кожному з цих випадків операторна функція оголошується по-різному.Операторна функція-член має наступний вигляд:

типім’я\_класу::operator#{список-аргументів){...//Операції}

Зазвичай операторна функція повертає об'єкт класу, з яким вона працює проте тип значення, що повертається, може бути будь-яким. Символ # замінюється необхідним оператором. Наприклад, якщо в класі перевантажується оператор ділення «/» операторна функція-член називається operator/. При перевантаженні унарного оператора список аргументів залишається порожнім. При перевантаженні бінарного оператора список аргументів містить один параметр.

## Контейнери

Як випливає з назви, контейнер - це об'єкт, який може містити в собі інші об'єкти. Існує кілька різних типів контейнерів. Наприклад, клас **vector** визначає динамічний масив, **deque** створює двонаправлену чергу, а **list** являє собою зв'язний список. Ці контейнери називаються послідовними контейнерми (**sequence containers**), тому що в термінології STL послідовність - це лінійний список. STL також визначає асоціативні контейнери (**associative containers)**, які забезпечують ефективний витяг значень на основі ключів.

Таким чином, асоціативні контейнери зберігають пари "**ключ/значення**". Прикладом може служити **map**. Цей контейнер зберігає пари "ключ/значення", в яких кожен ключ є унікальним. Це полегшує вилучення значення по заданому ключу.

## Ітератори

Ітератори - це об'єкти, які ведуть себе більш-менш подібно вказівниками.

Вони надають можливість виконувати циклічну обробку елементів контейнера - подібно до того, як використовується покажчик для організації циклу по масиву.

Існує п'ять типів ітераторів.

|  |  |
| --- | --- |
| **Ітератор** | **Тип доступу** |
| Довільний доступ  BiIter | Зберігає і витягує значення. Доступ до елементів - в довільному порядку |
| Двонаправлений  ForIter | Зберігає і витягує значення. Допускає переміщення вперед і назад |
| Прямий  InIter | Зберігає і витягує значення. Переміщення тільки вперед |
| Вхідний  OutIter | Витягує, але не зберігає значення. Переміщення тільки вперед |
| Вихідний  RandIter | Зберігає, але не витягує значення. Переміщення тільки вперед |

# Програмна документація

## Проектування структури програми

Проектування структури програми ґрунтується на визначенні структур класів, операцій, даних та методів їх опрацювання, семантики потоків повідомлень.

Розв’язання задачі почнемо з виявлення понять класів і їх фундаментальних взаємозв'язків.

Основним завданням є створення класу МАСИВ ВЕЛИКИХ ЧИСЕЛ (у програмному коді - BigIntArray), що містить масив об’єктів класу ВЕЛИКІ ЦІЛІ ЧИСЛА (BigInt). Тобто об’єкт типу МАСИВ ВЕЛИКИХ ЧИСЕЛ повинен містити деяку сукупність або колекцію інших об’єктів. Очевидно, що взаємовідношення між вказаними класами відноситься до типу “МАСИВ ВЕЛИКИХ ЧИСЕЛ – ВЕЛИКІ ЦІЛІ ЧИСЛА”. З цього випливає, що МАСИВ ВЕЛИКИХ ЧИСЕЛ – це контейнерний клас, і для роботи з цим класом необхідно створити клас-ітератор (MyIterator). Цей клас має забезпечувати проходження по масиву об’єктів класу ВЕЛИКІ ЦІЛІ ЧИСЛА.

Очевидно, що в конструкторі класу МАСИВ ВЕЛИКИХ ЧИСЕЛ необхідно передбачити динамічне виділення пам’яті для необхідної кількості об’єктів типу ВЕЛИКІ ЦІЛІ ЧИСЛА, а в деструкторі – звільнення цієї пам’яті. Адреса початку масиву об’єктів буде представлено закритим полем CEmployee\* pEmployee.

З завдання з’ясовуємо також, що у класі необхідно мати метод Input() для первинного введення інформації в МАСИВ ВЕЛИКИХ ЧИСЕЛ. Для контролю правильності введення початкових даних нам стане в нагоді ще один метод – Show(), що забезпечує виведення на екран вміст контейнерного класу. Також необхідними є ще два методи – begin() та end(). Вони відповідно мають встановлювати вказівники на початок і кінець масиву об’єктів типу ВЕЛИКІ ЦІЛІ ЧИСЛА. Доцільно створити ще один метод BigInt\* GetBigInts(), який буде повертати вказівник на елемент масиву.

Повернемося до класу MyIterator. Отже, цей клас має забезпечувати роботу з масивом об’єктів, а саме забезпечувати прохід по ньому. Тому доцільно оголосити закрите поле T\* m\_pBuffer. Також варто знати кількість елементів масиву (поле size\_t m\_iCount) та позицію, на яку вказує ітератор (поле size\_t m\_iPosition). Щоб забезпечити виконання поставленого перед нами завдання, необхідно створити ітератор введення. Ітератор введення підтримує операції рівності, розіменування й інкрементації, а саме =, ->,\*i, i++ , !=, ++i, ,== \*i++. У нашому випадку буде достатньо забезпечити перші п’ять:

public:

T\* operator++();

T\* operator->();

T\* operator=(T\* lvalue);

bool operator!=(T\* lvalue);

T operator\*();

Тепер розберемося з класом ВЕЛИКІ ЦІЛІ ЧИСЛА (BigInt).У завданні вказано: «Розробити клас РЯДОК СИМВОЛІВ, який містить вказівник на початок рядка). Створити похідний від нього клас ВЕЛИКІ ЦІЛІ ЧИСЛА з полями кількість використаних байт в рядку і знак».Отже, клас ВЕЛИКІ ЦІЛІ ЧИСЛА (BigInt) має батьківський клас РЯДОК СИМВОЛІВ (String, - і має наслідувати весь його функціонал. Тому спочатку визначимося з цим класом.

Як вказано в завданні цей клас має мати відповідно такі поля: прізвище, ім’я, по-батькові, дата народження, стать. Доцільно, щоб ці поля мали типи char\*, char\*, char\*, char\*, char відповідно і мали модифікатор доступу protected, щоб бути доступними у похідному класі:

protected:

char \*p; // вказівник на початок рядка

int capacity; // розмір рядка

Також необхідно визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів цього класу і перевантажити операцію для встановлення значень полів даних, операцію присвоєння об'єктів =, потокові операції введення » та виведення « об'єктів. В результаті отримаємо такі методи:

public:

String();

String(int);

String(char\*);

String(string);

String(String&);

virtual ~String();

void setString(char \*);

char \*getString();

virtual String &operator+ (String&); // віртуальний оператор конкатенації рядків

String operator- (String&);

bool operator\* (String&);

virtual String& operator= (const String&);

virtual void print(ostream &os);

friend ostream& operator<< (ostream &output, String&);

friend istream& operator >> (istream &input, String&);

Перевантаження операції витягання несе в собі проблему, яку необхідно розглянути і вирішити. Під час вводу у поле int програма очікуватиме ціле число; якщо буде введена довільна стрічка, це викличе помилку часу виконання . Для вирішення цієї проблеми в програмі буде використана функція GetInt(), що забезпечує дуже надійне введення цілих чисел. Реалізацію цієї функції ми подамо у додатку В. Оскільки функція GetInt() - універсальна і позакласова, то її код розмістимо в окремому модулі.

Повернемося до класу ВЕЛИКІ ЦІЛІ ЧИСЛА (BigInt). Для зберігання інформації, що відноситься до службовця, будуть потрібні наступні поля:

private:

int size; // к-іть використаних байт (масиву p)

int sign; // -1, 0 або +1

Конструктор класу повинен виділяти пам'ять для зберігання вказаного рядка, а деструктор – звільняти цю пам'ять. Також необхідно передбачити конструктор копіювання, який виконуватимете «глибоке копіювання».Оскільки для збереження цілісності даних вказані поля будуть оголошені у закритій частині, то необхідно передбачити методи доступу до них. І нарешті, для класу ВЕЛИКІ ЦІЛІ ЧИСЛА потрібно передбачити перевантаження операції витягання, щоб забезпечити первинне введення інформації з клавіатури в методі Input() класу МАСИВ ВЕЛИКИХ ЧИСЕЛ, і операцію вставлення, яка буде використана в методі Show() класу МАСИВ ВЕЛИКИХ ЧИСЕЛ. Обидві операції будуть реалізовані як зовнішні дружні функції.

Окремим пунктом завдання є: «У межах ієрархії класів побудувати поліморфічний кластер на основі віртуального оператора +. Продемонструвати механізм пізнього зв’язування.»

Оскільки у С++ пізнє зв’язування охоплює ряд функцій-членів (методів) або операторів, званих віртуальними функціями, оперторами. Віртуальна функція або оператор оголошуються в базовому або в похідному класіі, потім перевизначається в успадкованих класах. Тому у класі РЯДОК СИМВОЛІВ ми оголосили віртуальний оператор virtual BigInt& operator+ (String&) і перевизначили його у класі ВЕЛИКІ ЦІЛІ ЧИСЛА. Сукупність класів (підкласів), в яких визначається і перевизначається віртуальний оператор або метод, називається поліморфічним кластером, асоційованим з деяким віртуальним оператором або методом. Тому класи String (РЯДОК СИМВОЛІВ) та BigInt (ВЕЛИКІ ЦІЛІ ЧИСЛА) утворюють поліморфічний кластер на базі віртуального оператора + virtual BigInt& operator+ (String&). У межах поліморфічного кластера повідомлення зв'язується з конкретною віртуальною функцією-членом під час виконання програми.

Пізніше, під час уточнення реалізації методів, буде необхідно визначити специфічні дії, які повинен виконувати об’єкт, отримавши повідомлення. Під час виконання програма інтерпретуватиме ці діїі пов'язуватиме повідомлення з відповідним методом. Перегляд списку методів і вибір потрібного покладається на програму.

Треба внести ще одне уточнення стосовно реалізації віртуального кластера: для повного звільнення виділеної програмою пам’яті для об’єктів, що є членами віртуального кластера, необхідно оголосити деструктори цих класів віртуальними.

Отже, на основі викладеного матеріалу передбачимо такі конструктори, деструктор, методи і оператори для класу ВЕЛИКІ ЦІЛІ ЧИСЛА:

public:

BigInt(int n, int cap);

BigInt(int n);

BigInt(long double d);

BigInt();

BigInt(string s);

BigInt(const char s[]);

BigInt(const BigInt &n);

const BigInt &operator=(const BigInt &n);

const BigInt &operator=(int n);

virtual ~BigInt();

/\*\* Перевантаження арифметичних операторів \*\*/

BigInt operator++();

BigInt operator++(int);

BigInt operator--();

BigInt operator--(int);

BigInt operator-();

BigInt operator+ (int n);

BigInt operator+ (BigInt n);

virtual BigInt& operator+ (String&); // віртуальний оператор додавання великих цілих чисел

BigInt&operator+=(int n);

BigInt&operator+=(BigInt n);

BigInt operator- (int n);

BigInt operator- (BigInt n);

BigInt&operator-=(int n);

BigInt&operator-=(BigInt n);

BigInt operator\* (int n);

BigInt operator\* (BigInt n);

void operator\*=(int n);

void operator\*=(BigInt n);

BigInt operator/ (int n);

BigInt operator/ (BigInt n);

void operator/=(int n);

void operator/=(BigInt n);

int operator% (int n);

BigInt operator% (BigInt n);

void operator%=(int n);

void operator%=(BigInt n);

int divide(int n);

BigInt divide(BigInt n);

BigInt operator\* (long double n);

void operator\*=(long double n);

/\*\* Оператори побітової арифметики \*\*/

BigInt operator<< (int n);

void operator<<=(int n);

BigInt operator >> (int n);

void operator>>=(int n);

На останок необхідно передбачити засіб для оброблення помилок, які можуть виникнути у ході виконання програми. Ймовірно, тип класу найбільше підходить для опису помилки, яка потенційно може виникнути у програмі. Інформація, що міститься в об'єкті класу винятків, дає змогу спростити процес оброблення винятків. Тому, для оброблення помилок створено клас Error, який у закритій частині містить масив символів, в який записується уточнена інформація про помилку. Ця інформація виводиться на екран за допомогою методу What().

Діаграма класів, яка служить для представлення статичної структури моделі і яка є логічною абсорбцією цього розділу, представлена у Додатку А.

## Проектування інтерфейсу користувача

### Консольний інтерфейс

Розглянемо спочатку проектування програми з консольним інтерфейсом. У завданні вказано, що мають бути перевантажені оператори потокового вводу і виводу для певних класів, а це можливо лише в тому випадку для консольних програм на C++. Отже, необхідно передбачити зручний діалог при роботі у консолі з користувачем. На початку необхідно вивести дані про назву програми і її автора.

Оскільки програма передбачає роботу з контейнерним класом, то необхідно передбачити зручне та зрозуміле введення інформації. Цю задачу ми частково вирішили на попередньому етапі проектування, а саме за допомогою методу Input() класу BigIntArray (МАСИВ ВЕЛИКИХ ЧИСЕЛ). Цей метод варто зреалізувати за допомогою перевизначеної операцій вставки класу ВЕЛИКЕ ЦІЛЕ ЧИСЛО (BigInt).

На далі діалог з користувачем зручно організувати у вигляді меню. Пункти цього меню мають надавати певний функціонал користувачу і водночас забезпечити демонстрацію поставлених перед нами підзадач. Отже, доцільно визначити такі пункти меню:

\*\*\* ДІЇ ТА АРИФМЕТИЧНІ ОПЕРАЦІЇ над ВЕЛИКИМИ ЦІЛИМИ ЧИСЛАМИ \*\*\*

1. Додавання ВЕЛИКИХ ЦІДИХ ЧИСЕЛ;

2. Віднімання ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

3. Множення ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

4. Ділення ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

5. Конкатенація ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

6. Відображення ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

7. Побітові (порозрядні) операції над ВЕЛИКИМИ ЦІЛИМИ ЧИСЛАМИ;

8. Перебір ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

9. Логарифми з ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

10. Вивід МАСИВУ ВЕЛИКИХ ЧИСЕЛ;

11. Знайти суму елементів масиву ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ;

12. Демо механізму пізнього зв'язування МАСИВ ВЕЛИКИХ ЧИСЕЛ = РЯДОК;

13. Демо механізму пізнього зв'язування МАСИВ ВЕЛИКИХ ЧИСЕЛ = ВЕЛИКЕ ЦІЛЕ ЧИСЛО;

14. Вихiд;

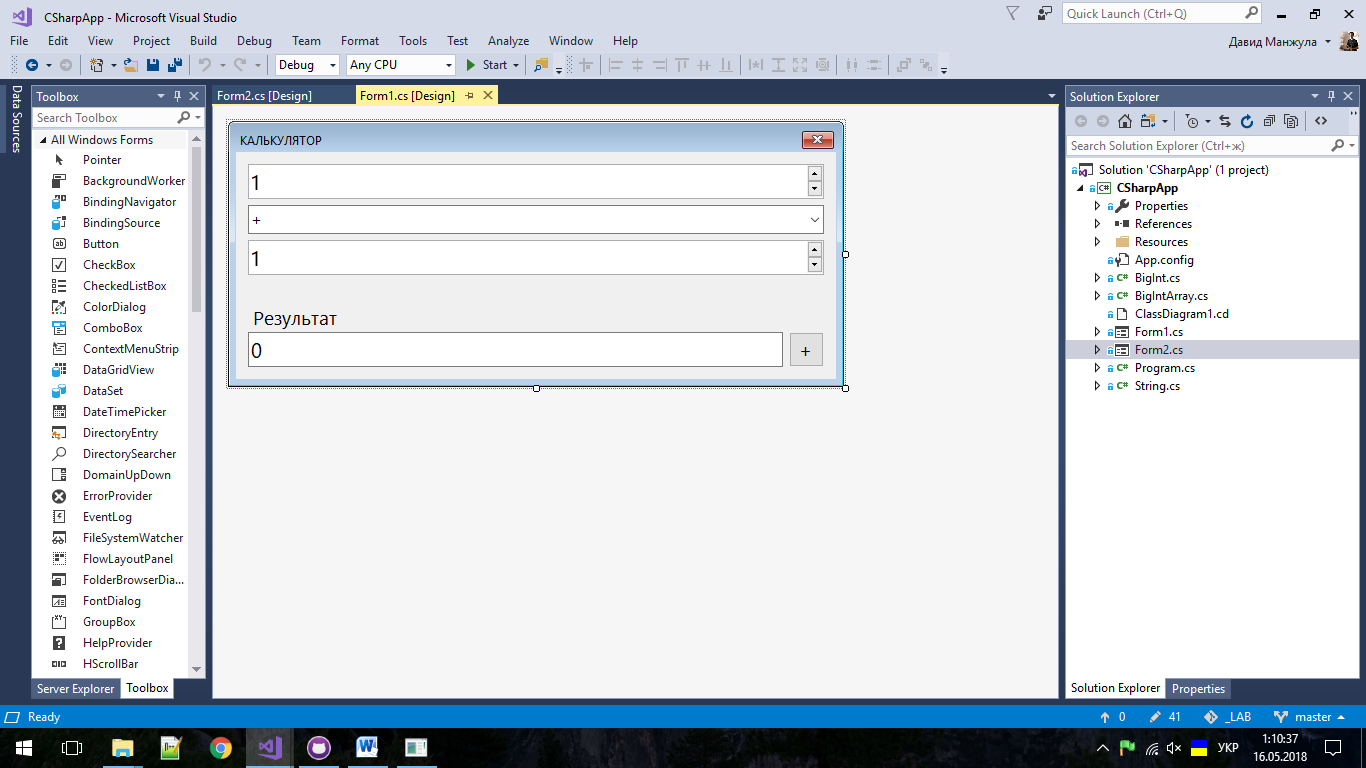
Виберiть будь-ласка пункт з меню: ;

Пункти з 1-9 демонструюють виконання арифметичних операцій і різних дій над великими цілими числами. Десятий пункт меню виводить вміст масиву великих чисел на екран. Одинадцятий обчислює суму всіх елементів масиву. Пункти 12, 13 демонструюють механізм пізнього зв’язування. Пункти передбачають організацію роботи з контейнерним класом МАСИВ ВЕЛИКИХ ЧИСЕЛ, а отже для її реалізації необхідно застосувати клас - ітератор MyIterator.

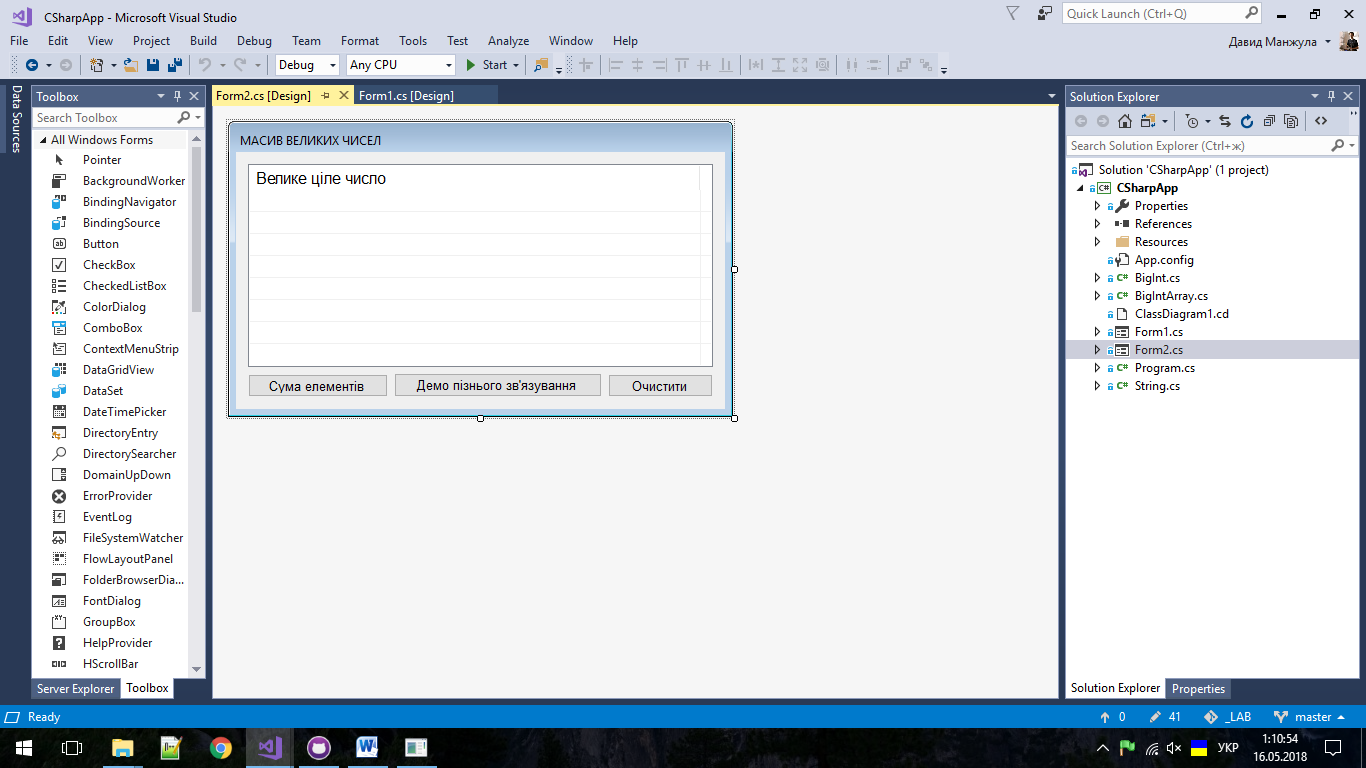
Отже, ці пункти меню надають користувачу необхідний функціонал (можливість виконувати різні дії над великими цілими числами, знаходження суми всіх введених великих цілих чисел у масиві та їх перегляд), і водночас задовольняють поставлені перед нами завдання про демонстрацію певних механізмів реалізації програмного коду.

### Графічний інтерфейс з використанням Windows Forms

Друга версія програми повинна мати графічний інтерфейс (ГІ), який будемо проекувати з використанням Windows Forms. Програма з ГІ складається з двох форм. На пешій формі “Калькулятор великих цілих чисел (ВЦЧ)” розміщені такі елементи керування ГІ як десять цифрових кнопок Button від 0-9, шість кнопок виконання арифметичних операцій, чотири кнопки викоання побітових операцій, кнопка зміни знаку +/-, кнопка <C> очищення, кнопка <- видалення останньої введеної цифри. три кнопки для роботи з масивом чисел (MC - очистка, MR - читання, MS - збереження), одна мітка Label і одне поле вводу TextBox. На рисунку нижче показано спроектований графічний інтерфейс програми.



На другій формі розміщено три кнопки “Сума елементів”, “Демо пізнього зв’язування”, “Очистити” і елемент список ListView.



## Розроблення консольної версії програми на мові C++

Після планування структури програми, розроблення її екранних форм, елементів діалогу, графічних схем класів та відношень між ними здійснимо деталізацію структури складових частин програми, які загалом повинні виконувати розв'язування поставленої задачі та забезпечувати взаємодію з користувачем через спроектований інтерфейс. Розроблення складових частин вимагає знання мов програмування С++. C#, підходів до проектування ПЗ з графічним інтерфейсом з використанням Windows Form і середовища розробки програмного забезпечення Visual Studio 2015. В роботі розроблено дві програми, які розв’язують поставлене завдання. Перша версія програма розроблена на мові C++ і має консольний інтерфейс. Друга версія програми з графічним інтерфейсом реалізована на мові C# з використанням Windows Form.

Оскільки описувати всі реалізовані у програмі методи – це занадто об’ємна робота для курсового проекту, то представимо реалізацію найбільш цікавих методів.

Розглянемо реалізацію на мові C++ першої версії програми з консольним інтерфейсом.

По – перше, в консольній версії програми маємо перевантажити оператори +, - і \* для класу РЯДОК СИМВОЛІВ (String).

String& String::operator+(String &source)

{

long cap = strlen(source.getString()) + strlen(p) + 1;

char \*dest = new char[cap];

strcpy(dest, p);

strcat(dest, source.getString());

delete[] p;

p = dest;

/\*String temp(dest);

\*this = temp;

delete[] dest;\*/

return \*this; //temp;

}

String String::operator-(String &source)

{

char \*pch, \*substr = source.getString();

char \*dest = new char[strlen(p)+1];

strcpy(dest, p);

while ((pch = strstr(dest, substr)) != NULL)

strcpy(pch, pch + strlen(substr));

String temp(dest);

delete[] dest;

return temp;

}

bool String::operator\*(String &source)

{

char \*pch = strstr(p, source.getString());

if (pch != NULL)

return true;

else

return false;

}

String& String::operator=(const String &Source)

{

// копіювання

if (&Source != this)

{

if (p != NULL) delete[] p;

p = new char[strlen(Source.p) + 1];

strcpy(p, Source.p);

}

return \*this;

}

По-друге, розглянемо перевантаження операторів потокового вводу >> і виводу << класу ВЕЛИКЕ ЦІЛЕ ЧИСЛО:

/\*\* Input/Output \*\*/

istream &operator >> (istream &in, BigInt &n)

{

n.size = 0;

n.sign = 1;

int sign = 1;

int c;

while ((c = in.peek()) >= 0 &&

(c == ' ' || c == '\t' || c == '\r' || c == '\n'))

in.get();

if (c < 0 || (c != (int)'-' && !isDigit(c)))

{

in >> c;

return in;

}

if (c == (int)'-') { sign = -1; in.get(); }

while ((c = in.peek()) >= 0 && isDigit(c))

{

in.get();

n \*= 10;

n += (c - (int)'0');

}

n.sign = sign;

n.normalize();

return in;

}

ostream &operator<<(ostream &out, BigInt n)

{

return out << n.toString();

}

Спочатку метод виводить інформацію з проханням ввести необхідні дані у певному форматі. Далі у захищеній частині коду відбувається читання інформації та подальше її присвоєння відповідним членам класу. Присвоєння реалізоване через відповідні методи доступу до полів класу.

По-третє, варто розглянути клас – ітератор, який застосовується для роботи з контейнерним класом МАСИВ ВЕЛИКИХ ЧИСЕЛ.

#pragma once

template <class T> class MyIterator

{

public:

MyIterator(T\* pElement, size\_t iCount);

~MyIterator();

public:

T\* operator++();

T\* operator->();

T\* operator=(T\* lvalue);

bool operator!=(T\* lvalue);

T operator\*();

protected:

T\* m\_pBuffer;

size\_t m\_iCount;

size\_t m\_iPosition;

};

template <class T> MyIterator<T>::MyIterator(T \*pElement, size\_t iCount) :

m\_pBuffer(pElement), m\_iCount(iCount)

{

m\_iPosition = 0;

}

template <class T> MyIterator<T>::~MyIterator()

{}

template <class T> T\* MyIterator<T>::operator++()

{

if (m\_pBuffer == NULL || m\_iCount <= 0)

{

throw runtime\_error("Bad pointer");

}

if (m\_iPosition > m\_iCount)

{

throw runtime\_error("Out of range");

}

m\_iPosition++;

T\* pData = m\_pBuffer + m\_iPosition;

return pData;

}

template <class T> T\* MyIterator<T>::operator->()

{

T\* pData = m\_pBuffer + m\_iPosition;

return pData;

}

template <class T> T\* MyIterator<T>::operator=(T\* lvalue)

{

return m\_pBuffer = lvalue;

}

template <class T> T MyIterator<T>::operator\*()

{

return \*(m\_pBuffer + m\_iPosition);

}

template <class T> bool MyIterator<T>::operator!=(T\* lvalue)

{

bool pr;

if ((m\_pBuffer + m\_iPosition) != lvalue) pr = true;

else pr = false;

return pr;

}

Для реалізації поставленого перед нами завдання достатньо було організувати ітератор введення, і перевантажити для нього операції \*і, і++,->, =,!=. Однак для даного завдання доцільніше було б організовувати прохід по масиву об’єктів у класі МАСИВ ВЕЛИКИХ ЧИСЕЛ, використовуючи вказівники на елементи масиву. Існування класу - ітератора у даному програмному коді призводить до його надлишковості. Механізм пізнього зв’язування реалізований у даному фрагменті коду:

case 12: // виклик віртуального оператора конкатенації рядків

{

String \*pObj = new String();

for (p\_BigInt = bigIntArr.begin(); p\_BigInt != bigIntArr.end(); ++p\_BigInt)

{

String \*temp = new String(p\_BigInt->toString());

\*pObj = \*pObj + \*temp;

delete temp;

}

cout << \*pObj << endl;

delete pObj;

}

break;

case 13: // виклик віртуального оператора додавання великих цілих чисел

{

String \*pObj = new BigInt();

for (p\_BigInt = bigIntArr.begin(); p\_BigInt != bigIntArr.end(); ++p\_BigInt)

{

\*pObj = \*pObj + \*p\_BigInt;

}

cout << \*pObj << endl;

delete pObj;

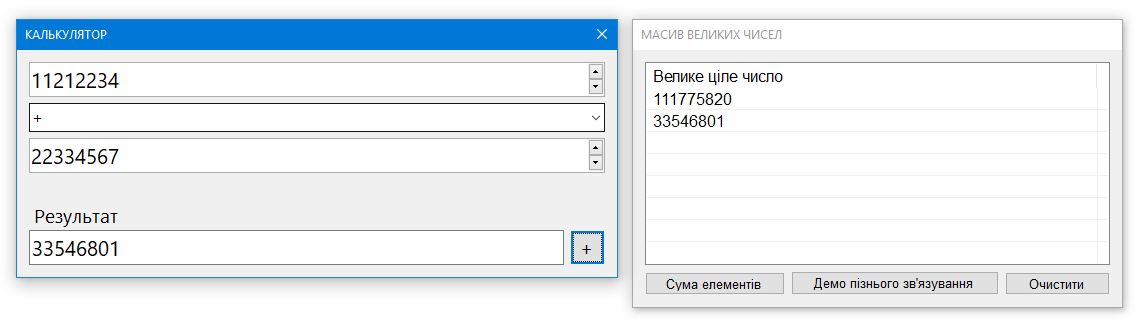
}

break;

Розв’язання задачі, в якій реалізовані розглянуті концепції, є багатофайловий проект Task. Лістинг програми розміщено у додатку Б.

## Розроблення програми з графічним інтерфейсом на C#

Тепер розглянемо розробку програми на мові C# з використанням Windows Forms. На рисунку нижче зображено графічний інтерфейс програми спроектований засобами Visual Studio 2015. Кожний елемент керування ГІ може генерувати відповідні події залежно від дій користувача над ним. Ці згенеровані події можуть бути зв’язані з відповідними методами для їх обробки, які реалізовані у відповідних класах.



Проте, перед тим як перейти до огляду реалізації методів обробки подій від елементів керування ГІ розглянемо реалізацію класів РЯДОК СИМВОЛІВ, ВЕЛИКІ ЦІЛІ ЧИСЛА і МАСИВ ВЕЛИКИХ ЧИСЕЛ на C#.

Cпочатку, розглянемо опис базового класу String (РЯДОК СИМВОЛІВ) на C#. Він містить методи для встановлення і читання рядка символів:

/\* Set method \*/

public void setString(char[] \_data)

{

this.data = \_data;

}

/\* Get methods \*/

char[] getString()

{

return data;

}

В похідному класі BigInt від String є наступні поля:

private uint[] data = null; // stores bytes from the Big Integer

public int dataLength; // number of actual chars used

Тепер перейдемо до реалізації методів обробки подій, які генерують елементи керування ГІ при зміні свого стану або при роботі користувача з ними. Ці методи, реалізовані в класі форми public partial class Form1 : Form. Розглянемо метод обробки натиснення цифрової кнопки 0-9:

private void btn1\_Click(object sender, EventArgs e)

{

if (Display.Text == "0") Display.Text = "";

Display.Text = Display.Text + 1;

}

private void btnAdd\_Click(object sender, EventArgs e)

{

a = new BigInt(Display.Text, 10);

Display.Clear();

operation = 1;

label1.Text = a.ToString() + "+";

sign = true;

}

private void btnEqu\_Click(object sender, EventArgs e)

{

calculate();

label1.Text = "";

}

Метод виконання арифметичних, побітових операцій та порівняння великих цілих чисел має вигляд:

private void calculate()

{

switch (operation)

{

case 1:

b = a + new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 2:

b = a - new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 3:

b = a \* new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 4:

b = a / new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 5:

b = a % new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 6:

b = a & new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 7:

b = a | new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 8:

b = a ^ new BigInt(Display.Text, 10);

Display.Text = b.ToString();

break;

case 9:

bool result = a < new BigInt(Display.Text, 10);

if (result) Display.Text = "TRUE";

else Display.Text = "FALSE";

break;

case 10:

if (a > new BigInt(Display.Text, 10)) Display.Text = "TRUE";

else Display.Text = "FALSE";

break;

default:

break;

}

}

В методі btnMS\_Click, який зв’язаний з кнопкою MS, відбувається зберігання введеного числа у МАСИВІ ЧИСЕЛ.

private void btnMS\_Click(object sender, EventArgs e)

{

BigInt n = new BigInt(Display.Text, 10);

bigIntArr.AddBigIntNumber(n.ToString());

}

Метод listView1\_SelectedIndexChanged виликається при виборі рядка списку ListView. В якому можна визначити,які рядки і списку ListView вибрано:

В методі btnSum\_Click, який зв’язаний з кнопкою “Сума елементів”, відбувається виклик методу bigIntArr.Sum(), який обчислює суму елементів масиву.

private void btnSum\_Click(object sender, EventArgs e)

{

string str = "";

str += "Сума всіх елементів масиву складає: " + bigIntArr.Sum();

MessageBox.Show(str, "Сума елементів масиву");

}

public BigInt Sum()

{

BigInt sum = new BigInt();

for (int i = 0; i < BigInts.Count; i++)

sum = sum + BigInts[i];

return sum;

}

В методі button2\_Click, який викликається при натисненні кнопки “Очистити”, виконується очистка елемента ГІ список bigIntsListView.Items.Clear() і видалення всіх чисел зі списку виклик методу bigIntArr.GetBigInts().Clear().

private void button2\_Click(object sender, EventArgs e)

{

bigIntsListView.Items.Clear();

bigIntArr.GetBigInts().Clear();

}

В методі UpdateData() виконується обновлення даних в списку елементів масиву.

public void UpdateData()

{

bigIntsListView.Items.Clear();

int counterOfArraylist = bigIntArr.GetCount();

string str;

foreach (BigInt number in bigIntArr.GetBigInts())

{

str = number.ToString();

ListViewItem li = new ListViewItem(str);

li.UseItemStyleForSubItems = false;

//li.SubItems[0].BackColor = Color.Aqua;

bigIntsListView.Items.Add(li);

}

}

Метод textBoxCharIsDigit\_KeyPress, який викликається при натисненні клавіші в полі TextBox, допускає ввід лише цілих чисел.

private void textBoxCharIsDigit\_KeyPress(object sender, KeyPressEventArgs e)

{

int isNumber = 0;

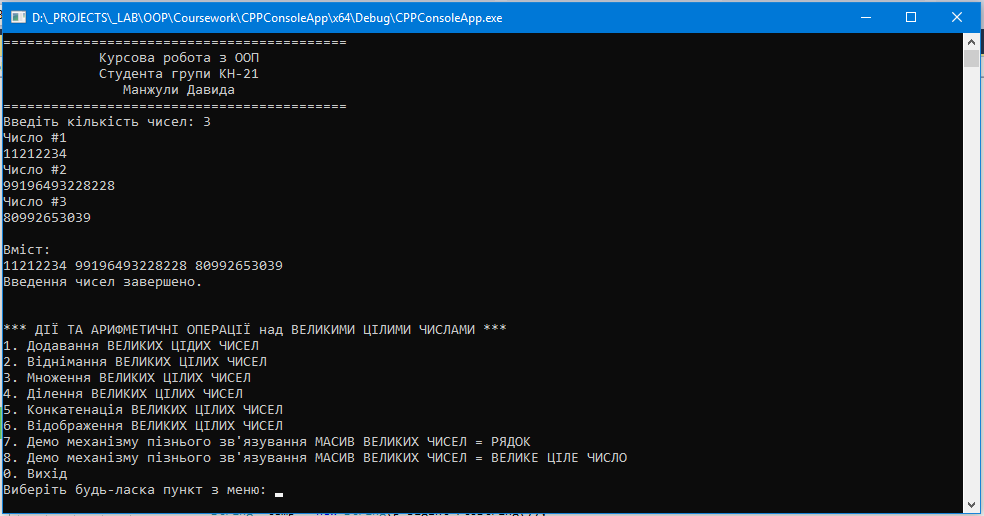
e.Handled = !int.TryParse(e.KeyChar.ToString(), out isNumber);

}

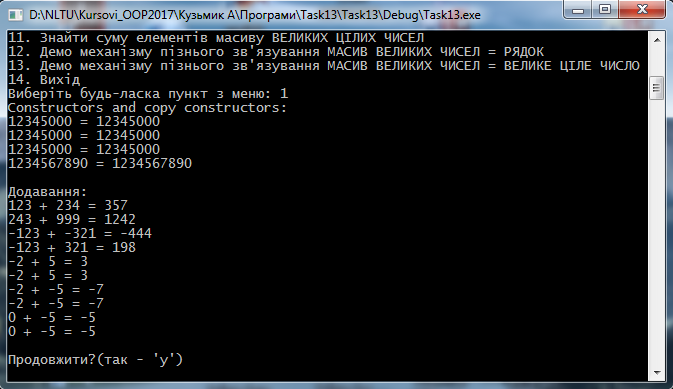
## Приклад та аналіз результатів виконання програми з консольним інтерфейсом

Консольна реалізація на мові C++.

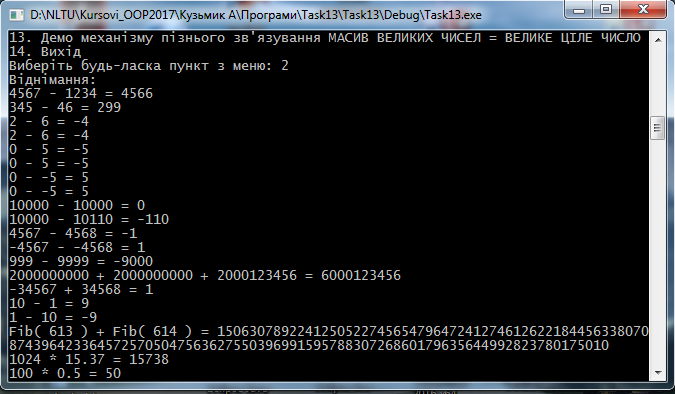
Ввід даних:



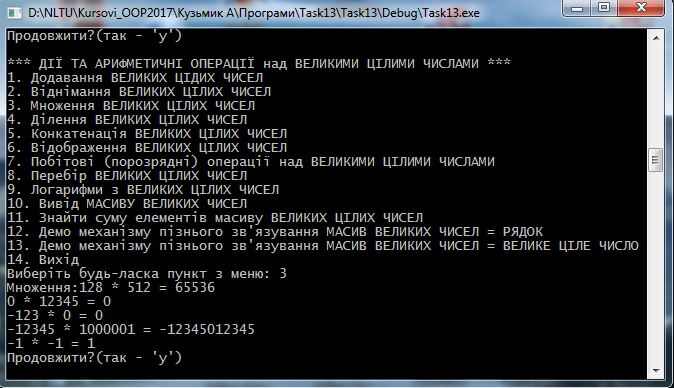
1. Додавання ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ:



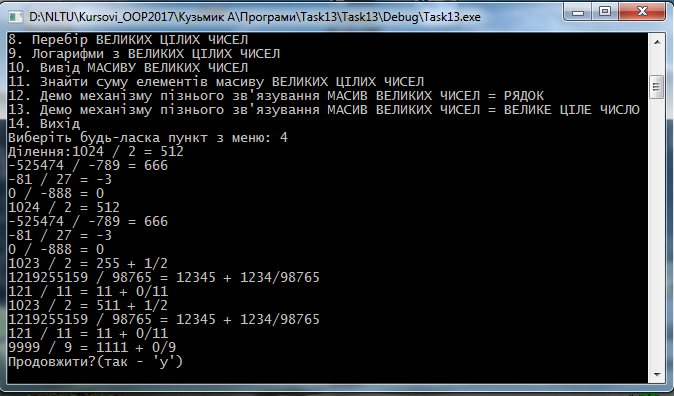
1. Віднімання ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ:



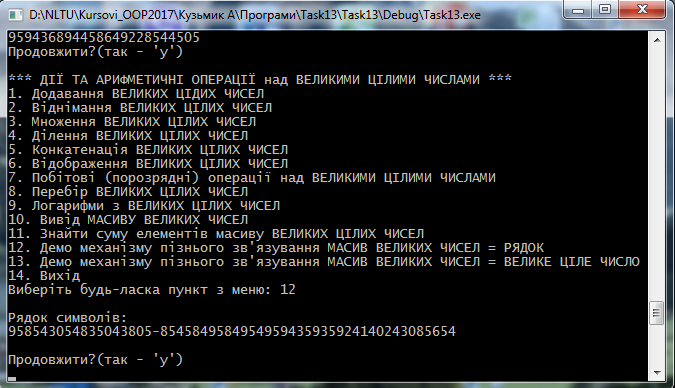
1. Множення ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ:



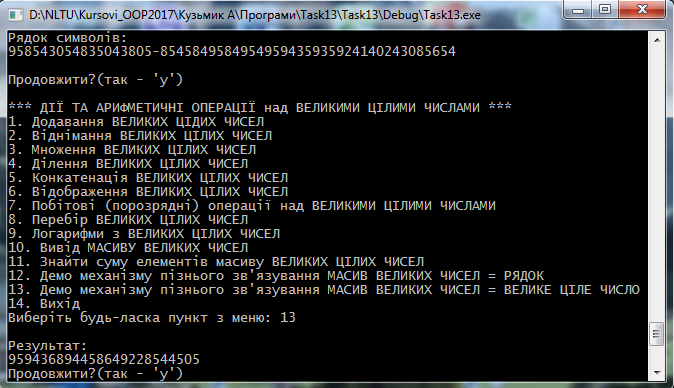
1. Ділення ВЕЛИКИХ ЦІЛИХ ЧИСЕЛ:



1. Демо механізму пізнього зв’язування МАСИВ ВЕЛИКИХ ЧИСЕЛ = РЯДОК:



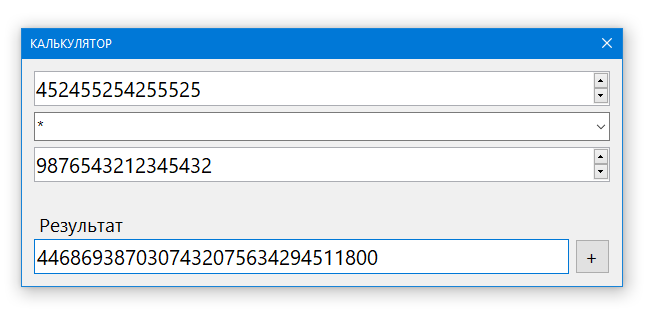
1. Демо механізму пізнього зв’язування МАСИВ ВЕЛИКИХ ЧИСЕЛ = ВЕЛИКЕ ЦІЛЕ ЧИСЛО:



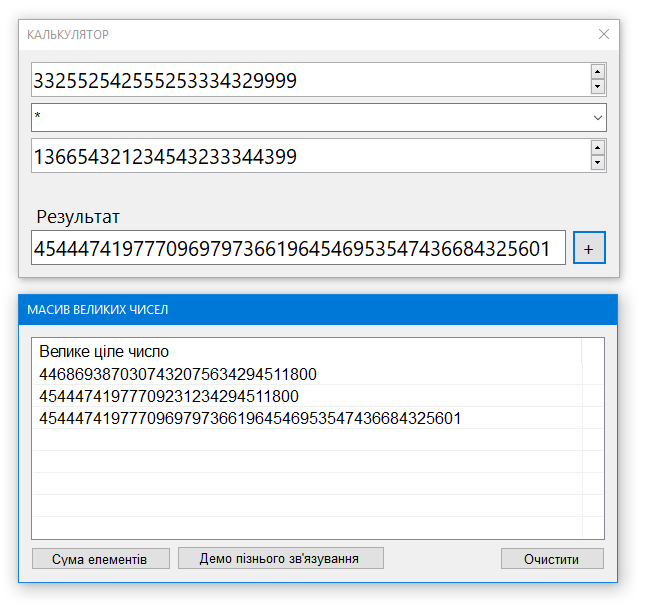
## Приклад та аналіз результатів виконання програми з графічним інтерфейсом на Windows Forms

Реалізація програми з ГІ на мові C# з використанням Windows Forms.

Виконання арифметичних операцій:



Внесення даних в МАСИВ ВЕЛИКИХ ЧИСЕЛ виконується натисненням кнопки “+”



# Висновки

Розглянуто теоретичний та практичний матеріал, призначений для набуття досвіду роботи з технологіями ООП. В ході виконання роботи реалізовано дві версії програми в середовищі Visual Studio 2015. Перша версія є консольна і реалізована на мові програмування С++, а друга версія має графічний інтерфейс і реалізована но мові C# з використанням Windows Forms. В програмах створено клас РЯДОК СИМВОЛІВ з полями даних: вказівник на початок рядка у динамічній пам’яті, максимальна довжина рядка та похідний від нього клас ВЕЛИКІ ЦІЛІ ЧИСЛА. Реалізовано операторні методи введення, виведення, виконання арифметичних оперцій та порівняння великих чисел. Розроблено клас-контейнер МАСИВ ВЕЛИКИХ ЧИСЕЛ, що містить масив об’єктів класу ВЕЛИКІ ЦІЛІ ЧИСЛА. Для роботи з масивом об’єктів розроблений клас-ітератор. Також продемонстровано на прикладі можливості об’єктно-орієнтованого програмування.

Загалом, створений програмний продукт дозволяє виконувати арифметичні операції та порівняння великих чисел, вносити їх у масив великих чисел.

В подальшому необхідно розширювати функціональність програми та забезпечувати кращий захист від несподіваних помилок.

Варто відзначити, що об’єктно-орієнтоване програмування дозволяє створювати програму, як набір користувацьких типів даних (класів), приховувати деталі реалізації, використовувати повторний код, інтерпретувати виклики процедур та функцій на етапі виконання (втілюючи парадигми ООП - інкапсулювання, поліморфізм, успадкування).

# Список використаних джерел

1. Соколовський Я.І., Коширець С.І., Сало М.Ф. Методичні вказівки до курсової роботи з дисципліни «об’єктно-орієнтоване програмування» для студентів денної та заочної форми навчання напряму 6.050101 «комп’ютерні науки». Рік випуску 2012. (ел.версія) удк 004.415.2045 (075.8)
2. Коширець С.І.Об'єктно-орієнтоване програмування мовою С++: Лаб. практ. – Львів: Вид-во НЛТУ України, 2012. – 126 с.: іл. 15, табл. 18.ISВN 966-0000-00-0
3. Грицюк Ю.І. Об'єктно-орієнтоване програмування мовою С++: Навч. пос. – Київ: Вид-во "Юніор", 2006. – 464 с.: іл. 15, табл. 18.ISВN 966-0000-00-0
4. Самоучитель C++. Год выпуска: 2003. Автор: Герберт Шилдт. Издательство: СПб.: БХВ-Петербург, ISBN: 5-7791-0086-1.
5. Герберт Шилдт. Название: С++ Базовый курс. 3-е издание. Год: 2010. Издательство: Вильямс ISBN: 978-5-8459-0768-4
6. Объектно-ориентированное программирование в С++ (4-е изд.) Год: 2004. Автор: Лафоре Р. Издательство: ПИТЕР ISBN: 5-94723-302-9
7. Шилдт Г. C#: учебный курс. СПб.: Питер, 2002. — 512 с.: ил.
8. Петцольд Ч. Программирование с использованием Microsoft Windows Forms. Пер. с англ. - М.: Русская Редакция; СПб. : Питер, 2006. - 432 с. - ISBN 5-7502-0284-4, ISBN 5-91180-041-1.
9. Эндрю Троелсен. Язык программирования C# 2010 и платформа .NET 4. Изд. Вильямс, 2001 – 1392 с.
10. Електрониий ресурс по C#: <https://www.tutorialspoint.com/csharp/>
11. Електронний ресурс по C#: <https://www.javatpoint.com/c-sharp-tutorial>
12. Електронний ресурс по C#: <https://msdn.microsoft.com/uk-ua/library/aa288436(v=vs.71).aspx>