Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Абстрактне програмування в C++

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи
Кандидат фізико-математичних наук,
доцент
Бублик В.В.
_____
(підпис)
“____” _____ 2020 р.
Виконала студентка ІПЗ-3:
Скирта М.І.
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Кандидат фізико-математичних наук,

доцент

_____Жежерун О.П.

(підпис)

„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту 3-го курсу, факультету інформатики

Скирті Марії Ігорівні

Реалізувати алгоритми розв’язання систем лінійних рівнянь з використанням абстрактного програмування в C++

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Поняття абстракції в програмування

2 Огляд засобів абстрактного програмування в C++

3 Реалізація практичної частини з використанням абстрактного програмування та демонстрація переваг абстрактного підходу до програмування алгоритмів

Висновки

Список використаних джерел

Дата видачі „___” _____ 2020 р. Керівник _____

Завдання отримав _____

Календарний план виконання роботи

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової	14.10.2019	
2.	Пошук тематичної наукової літератури	20.10.2019	
3.	Ознайомлення з науковою літературою	01.12.2019	
4.	Визначення структури програми	04.02.2020	
5.	Повторення синтаксису С++	10.02.2020	
6.	Реалізація практичної частини	01.03.2020	
7.	Написання першої частини курсової роботи	01.04.2020	
8.	Написання другої частини курсової роботи	01.05.2020	
9.	Написання висновків курсової роботи	09.05.2020	
10.	Перегляд змісту роботи з керівником	10.05.2020	
11.	Внесення змін до роботи	10.05.2020	
12.	Завантаження курсової роботи	11.05.2020	

Студент _____

Керівник _____

“_____” _____

Зміст

Вступ 4

Розділ 1. Абстрактне програмування 6

1.1 Поняття абстракції в програмуванні 6

1.2 Абстракція даних 6

1.3 Абстракція керування 7

Висновки до розділу 1 8

Розділ 2. Засоби абстрактного програмування в C++ 9

2.1 Класи 9

2.2 Абстрактні класи 10

2.3 Узагальнене програмування як засіб абстракції 11

2.4 Концепції 12

2.5 Абстракція керування 13

Висновки до розділу 2 14

Розділ 3. Реалізація алгоритмів лінійної алгебри з використанням абстракції 15

3.1 Існуючі рішення 15

3.2 Реалізація з низьким рівнем абстракції 16

3.3 Реалізація з високим рівнем абстракції 20

Висновки до розділу 3 24

Висновки 25

Список використаних джерел 26

Вступ

Абстракція у програмуванні з’явилась ще з одними з перших мов програмування і стала одним з найпотужніших його засобів. Засоби та методи абстракції вивчали і висвітлювали у своїх роботах такі науковці як Барбара Лісков, яка ввела поняття абстрактного типу даних, Джон Гуттаг, Олександр Степанов та багато інших.

Що далі від програмування машинного коду ставали мови програмування, то більше засобів абстракції з’являлось у їх синтаксисі. У сучасних високорівневих мовах програмування абстракція реалізована на багатьох рівнях та у багатьох формах.

Мова C++ є у цьому сенсі унікальною, адже підтримує як низькорівневий, так і високорівневий код, і тим самим надає багато різних способів реалізувати одну й ту саму задачу.

Ця робота присвячена вивченню переваг використання для реалізації алгоритмів коду з високим рівнем абстракції, розгляду засобів, що дозволяють це реалізувати в C++ та демонстрації на прикладі реалізації алгоритмів розв’язку систем лінійних рівнянь в C++.

Об’єктом дослідження є абстрактне програмування в C++.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу абстракції як принципу програмування. Наведено огляд типів абстракції в програмування та переваги використання в програмному коді кожного них.

В другому розділі проаналізовано засоби абстракції, доступні у мові С++. Розглянуто можливі застосування кожного з них та завдання, у вирішенні яких є корисними дані засоби, наведено приклади реалізації у С++.

Третій розділ присвячено реалізації алгоритмів розв’язку систем лінійних рівнянь мовою С++. На прикладі реалізації алгоритму продемонстровано два підходи: з низьким та високим рівнем абстракції, проаналізовано переваги та недоліки кожного варіанту.

Постановка задачі

1. Проаналізувати абстракцію як підхід до програмування

2. Оглянути та описати засоби абстрактного програмування в С++

3. Продемонструвати переваги використання високого рівня абстракції для реалізації алгоритмів на прикладі реалізації алгоритмів розв’язання систем лінійних рівнянь в С++.

Розділ 1. Абстрактне програмування

1.1 Поняття абстракції в програмуванні

У програмуванні абстракція – це навмисне приховування деяких деталей процесу або сутності з метою виділення важливих аспектів, деталей або структури. [1]

Абстрагування – один з основних принципів об’єктно-орієнтованого програмування, хоча його реалізація не обмежується тільки мовами, що підтримують об’єктно-орієнтовану парадигму. У тій чи іншій формі абстракція реалізована у багатьох мовах програмування.

Абстракція в програмуванні поділяється на два основні типи: абстракція даних та абстракція керування.

1.2 Абстракція даних

Принцип абстракції даних полягає у тому, щоб приховати деталі реалізації певного типу даних. Це досягається відокремленням інтерфейсу – набору операцій, визначених для даної структури від його реалізації.

До переваг використання абстракції даних в коді належать:

- Можливість легко змінити реалізацію операції не змінюючи поведінку типу(при зміні реалізації не змінюється код, який використовує цей тип даних);
- Відсутність потреби знати про конкретну реалізацію та низькорівневі операції – достатньо знати лише поведінку типу.

Основним підходом до реалізації абстракції даних в програмуванні стала концепція абстрактних типів даних. Абстрактний тип даних – це клас абстрактних об’єктів, які повністю характеризуються визначеними на них

операціями[2]. Абстрактний тип даних визначає набір функцій, незалежних від конкретної реалізації типу, для оперування його значеннями.

Використовуючи абстрактні типи даних в своєму коді користувач не повинен знати, як саме вони реалізовані(як зберігаються в пам’яті тощо), йому достатньо знати лише поведінку(семантику, властивості) цих типів.

Абстрактні типи даних у мовах програмування часто реалізуються за допомогою модулів. Модуль – це набір зв’язаних процедур разом з даними, що піддаються обробці.

У мовах, що підтримують об’єктно-орієнтовану парадигму, однією з форм реалізації абстрактних типів даних є визначені користувачем типи – класи[3].

1.3Абстракція керування

Принцип абстракції керування полягає в приховуванні деталей виконання певної операції. Реалізація абстракції керування в мовах програмування зазвичай передбачає надання користувачу можливості визначення нових контролюючих структур.

Абстракція керування в програмуванні здійснюється за допомогою підпрограм(subroutines). Підпрограми зазвичай реалізовуються у вигляді процедур(не повертають результату) або функцій(повертають результат).

Для програміста абстракція керування має такі переваги:

- Відсутність потреби писати код на низькому(машинному) рівні;
- Можливість уникати дублювання коду за рахунок винесення повторюваних операцій у окремі підпрограми.

Мови програмування як засіб абстракції керування також надають перевагу відсутності необхідності написання різного коду для різних

комп’ютерних систем та архітектур. Задача формування машинного коду(або проміжного коду для інтерпретовних мов) для конкретної системи покладається на компілятор. Завдяки цьому користувач може розраховувати на те, що результат інтерпретації написаного ним коду вищого рівня буде однаковим для різних систем.

Висновки до розділу 1

Абстракція є одним з основних принципів програмування та реалізована у багатьох мовах програмування та мовах. Особливо важливою є абстракція для парадигми об’єктно-орієнтованого програмування. Абстракція в програмування поділяється на абстракцію даних та абстракцію керування.

Розділ 2. Засоби абстрактного програмування в C++

C++ є мультипарадигмальною мовою з підтримкою таких парадигм як процедурна, об'єктно-орієнтована та узагальнена і підтримує реалізацію багатьох різних форм абстракції.

У C++ реалізовано абстракцію в тому числі за допомогою заголовних файлів. Сигнатури методів та класів визначаються у заголовних файлах з розширенням .h, в той час як їх реалізація визначається вже в іншому файлі, який має розширення cpp. Таке розділення створює зручний механізм абстракції для відокремлення інтерфейсів від реалізацій.

2.1 Класи

Як зазначалось у попередньому розділі, в об’єктно-орієнтованій парадигмі класи є формою реалізації абстрактних типів даних.

Класи описують множини об’єктів, схожих за властивостями та внутрішньою структурою. Класи об’єднують типи даних та визначені на них операції. Назовні надаються публічні методи для маніпулювання даними об’єктів класу, в той час як самі дані класу приховані від користувачів і напряму до них доступитись не можна(за винятком окремих випадків). Таке відокремлення дозволяє легко за необхідності змінювати реалізацію методів класу, при цьому не вимагаючи ніяких змін в коді користувачів класу.

Так, наприклад матриця, представлена класом Matrix може зберігати елементи у вигляді одновимірного масиву, двовимірного масиву, зв’язного списку або інших структур. Усі ці структури потребують різних реалізацій методу ітерації елементами матриці. Для користувача важливий результат методу – послідовне проходження по всіх елементах, а не його реалізація. Отже, клас

Matrix має метод ітерації з ідентифікатором доступу public – доступний за межами класу. При цьому реалізація методу, як і внутрішня організація зберігання елементів матриці приховані від користувача.

2.2 Абстрактні класи

Вищий рівень абстракції у C++ надають абстрактні класи. Їх використання дозволяє об’єднувати різні класи в певні логічні групи з спільними властивостями.

Метою абстрактного класу є визначення базового інтерфейсу для похідних від нього класів. Такі класи обов’язково містять віртуальні функції, тобто ті, що не мають реалізації в даному класі, а будуть реалізовані в похідних класах. Віртуальні функції визначаються за допомогою ключового слова virtual.

Об’єкти абстрактного класу можуть бути створені тільки через похідні класи – спроба створити об’єкт абстрактного класу викличе помилку компіляції. Похідні класи, які реалізують усі віртуальні функції базового класу називаються конкретними. Зв’язок між абстрактними та конкретними класами здійснюється за допомогою успадкування.

Абстрактний клас може містити реалізацію методів.

Наприклад, клас Matrix може слугувати базовим класом для конкретних реалізацій Matrix1dArray – матриці, що базується на одновимірному масиві та Matrix2dArray – матриці, що базується на двовимірному масиві. Клас Matrix визначає спільний інтерфейс для цих окремих реалізацій.

2.3 Узагальнене програмування як засіб абстракції

Завдяки підтримці парадигми узагальненого програмування мова C++ дозволяє визначити додатковий рівень абстракції. Центральною ідеєю узагальненого програмування є використання параметризованих операцій, які є повністю незалежними від конкретних реалізацій структур даних та представляють конкретні та ефективні алгоритми[6].

У C++ реалізація узагальненого підходу здійснюється за допомогою шаблонів. Існують такі основні типи шаблонів:

- Шаблони класів – дозволяють визначати параметризовані класи, реалізація яких не залежить від конкретного типу даних. Шаблонні класи часто використовуються для реалізації класів-контейнерів, таких як стек, черга тощо. Також їх зручно використовувати для математичних структур, таких як матриці та вектори, оскільки їх елементами можуть бути різні числові типи – цілі, дійсні, комплексні числа, яким відповідають різні типи даних.

- Шаблони функцій – дозволяють визначати параметризовані функції. Шаблонні функції доцільно використовувати для програмування алгоритмів, які не залежать від конкретного типу даних. До таких алгоритмів належать, наприклад, функції сортування та перебору елементів певної структури даних. Також шаблонні функції доцільно використовувати для програмування математичних алгоритмів та операцій.

- Шаблони змінних – дозволяють визначати змінні без визначення їх конкретного типу. Шаблони змінних були додані у стандарті C++14. Основна мета шаблонних змінних полягає у більш зручному поєднанні з шаблонними алгоритмами.

Для визначення шаблонного класу, функції або змінної використовується однаковий синтаксис. Перед визначенням відповідного шаблону записується ключове слово template і після цього у кутових дужках перераховуються параметри шаблону. Параметри-типи шаблону визначаються за допомогою ключових слів class або typename. Також параметрами шаблону можуть бути параметри звичайних типів та параметри-шаблони.

Приклад визначення шаблону класу матриці, параметризованого типом елементів:

```
template<typename Elem>
class Matrix
{
};
```

Приклад визначення шаблону функції множення матриці на вектор:

```
template<typename Matrix, typename Vector>
Matrix& multiply(const Matrix&, const Vector&);
```

2.4 Концепції

У C++20, останньому затвердженому на час написання роботи стандарті, були додані концепції(concepts). До введення на рівні стандарту мови вони були реалізовані у бібліотеці Boost.

Концепції – це предикати часу виконання[Good concepts]. Концепції допомагають встановити відповідність використовуваних в шаблоні параметрів набору певних критеріїв. Вони можуть використовуватись з класами та функціями шаблонами.

Концепції є дуже корисними доповненнями до узагальненого програмування, оскільки дозволяють визначати властивості типів даних, необхідні для коректного виконання методів шаблонного класу або шаблонних функцій.

Приклад визначення концепції для перевірки, чи можуть елементи різних типів порівнюватись між собою на рівність:

```
template<typename A, typename B>
concept bool = comparable
requires (A a, B b) {
    { a == b } -> bool;
    { a != b } -> bool; };
```

2.5 Абстракція керування

В C++ абстракція керування реалізується за допомогою функцій. Функції можуть бути об’єднані з даними, якими вони маніпулюють в класи, тоді вони називаються методами цього класу[3].

Також у C++ є функтори – функціональні об’єкти. Завдяки ним можна виконувати об’єкт класу як функцію. Для створення такого об’єкта потрібно перевизначити в його класі оператор (). У стандарті C++ з’явились лямбда-вирази – короткий запис анонімних функторів.

Висновки до розділу 2

Як мультипарадигменна мова програмування C++ містить реалізацію різних абстракцій, в тому числі вищого рівня. Засобами абстракції даних у C++ є класи, а на вищому рівні – абстрактні класи. Завдяки підтримці парадигми узагальненого програмування з використанням шаблонів функцій, шаблонів класів та концепцій можна створити абстракції досить високого рівня.

Розділ 3. Реалізація алгоритмів лінійної алгебри з використанням абстракції

Практична частина роботи полягає в реалізації алгоритмів лінійної алгебри для розв’язку систем лінійних алгебраїчних рівнянь з використанням засобів абстрактного програмування в C++, демонстрації недоліків використання низькорівневих абстракцій та переваг високорівневих абстракцій для програмування алгоритмів.

3.1 Існуючі рішення

Мова C++ часто використовується для реалізації складних математичних обчислень, оскільки має широкий функціонал та механізми ефективного керування пам’яттю, що підвищують ефективність обчислювальних методів. Мовою C++ реалізовані декілька бібліотек для вирішення задач лінійної алгебри. Зокрема, до бібліотек, у яких використовуються найновіші технології високого рівня абстракції належать такі:

- Blitz++ — високоефективна бібліотека функцій векторної математики. Перша бібліотека, у якій почали використовувати шаблони C++, завдяки чому досягається висока швидкість виконання обчислень. Має підтримку операцій з матрицями, векторами і тензорами.

- Iterative Template Library (ITL) — бібліотека, призначена для застосування ітеративних числових методів до об'єктів лінійної алгебри. Містить реалізацію складних ітеративних методів для різних реалізацій структур даних лінійної алгебри та алгоритми передумовлення для систем лінійних рівнянь. На даний час активно не підтримується.

- uBLAS — бібліотека шаблонів заголовних файлів, що реалізує всі три рівні стандарту BLAS для розріджених, щільних та компактних(packed) матриць. Використовує перевантаження операторів та шаблони для ефективного обчислення.[14]

- Eigen — бібліотека шаблонів заголовних файлів з відкритим вихідним кодом для матрично-векторних обчислень. Містить операції для різних форматів матриць, в тому числі для щільних та розріджених матриць.[15]

- MTL4 — бібліотека лінійної алгебри. Підтримує декілька реалізацій розріджених та щільних матриць. Завдяки використанню узагальненого програмування дозволяє виконувати обчислення з різними числовими типами, в тому числі з високою точністю. Поточна остання версія бібліотеки, розроблена Ендрю Лумсдейном та Пітером Готтшлінгом, використовує такі нові можливості мови C++ як семантика переміщень, ознаки типів(type traits) тощо.[16]

Стандартна бібліотека шаблонів(STL) також містить реалізацію таких алгебраїчних структур як матриці та вектори, проте вони не є оптимізованими для задач розв’язання систем лінійних рівнянь.

3.2Реалізація з низьким рівнем абстракції

Для прикладу розглянемо алгоритм розв’язання систем лінійних рівнянь Гаусса-Зейделя. Він належить до ітераційних алгоритмів розв’язання, тобто, є

наближеним методом розв’язку, що обчислює результат із заданою точністю у формі границі послідовності деяких векторів.

Принцип ітераційних методів полягає у обчисленні наближеного розв’язку на кожній ітерації на основі результатів попередньої ітерації. Алгоритм зупиняється, якщо досягнуто заданої точності розв’язку.

Значною перевагою ітераційних над прямими є те, що при збіжності алгоритму вони дозволяють швидше знайти розв’язок системи з заданою точністю, а також можуть використовуватись для великих систем лінійних рівнянь.

Алгоритм знаходження розв’язку системи лінійних рівнянь методом Гаусса-Зейделя:

ALGORITHM **4.2** *General Block Gauss-Seidel Iteration*

1. *Until convergence Do:*
2. *For* $i = 1, 2, \ldots, p$ *Do:*
3. *Solve* $A_{ii}\delta_i = W_i^T(b - Ax)$
4. *Set* $x := x + V_i\delta_i$
5. *EndDo*
6. *EndDo*

Рис 1. Алгоритм Гаусса-Зейделя[9]

Найпростіша реалізація цього алгоритму мовою C++ виглядає так:

Лістинг 1. Реалізація методу Гаусса-Зейделя з низьким рівнем абстракції

```
bool convergence(double* x, double* p, int n, double eps)
{
	double conv = 0;
	for (int i = 0; i < n; i++)
		conv += (x[i] - p[i]) * (x[i] - p[i]);
	return (sqrt(conv) < eps);
}


void GaussSeidel(double eps, int size, double** a, double* b,
double* x)
{
	double* p = new double[];
	for (int i = 0; i < size; i++)
	{
		p[i] = x[i];
	}
	int i, j, n = 0;
	while (!convergence(x, p, size, eps))
	{
		for (int i = 0; i < size; i++)
			p[i] = x[i];
		for (int i = 0; i < size; i++)
		{
			double cur = 0;
			for (int j = 0; j < i; j++)
				cur += (a[i][j] * x[j]);
			for (int j = i + 1; j < size; j++)
				cur += (a[i][j] * p[j]);
			x[i] = (b[i] - cur) / a[i][i];
		}
		n++;
	}
	return;
}
```

Надана вище реалізація написана на доволі низькому рівні абстракції. Вона має ряд проблем та недоліків.

По-перше, розглянемо сигнатуру даних методів convergence та GaussSeidel

```
bool convergence(double* x, double* p, int n, double eps);
```

```
void GaussSeidel(double eps, int size, double** a, double* b, double* x);
```

В даній реалізації для зберігання елементів векторів та матриць використовуються масиви. Це робить неможливим використання даного методу для інших форм реалізації матриць та векторів. Для того, щоб використати цей алгоритм для іншої реалізації цих типів даних довелось би повністю переписати його. Отже, ця реалізація непридатна для повторного використання, що суперечить принципу написання коду з якомога більшим потенціалом для «reusability».

До того ж, виконання цього методу може призвести до помилок управління пам’яттю, оскільки неможливо перевірити чи коректно були ініціалізовані користувачем масиви елементів та чи відповідає розмір, переданий як параметр дійсному розміру матриці та векторів.

По-друге, усі числові типи явно вказані в даній реалізації. Вектори та масиви містять елементи типу double. Якщо розв’язати рівняння даним методом необхідно було б для іншого числового типу, наприклад, для цілих або комплексних чисел або для типів з більшою точністю, то довелось би повністю переписати даний метод, аналогічно до першої проблеми цієї реалізації.

Параметр заданої точності eps також має тип double. Для деяких застосувань точність розв’язку системи лінійних рівнянь може виявитись критичною, а тому для неї потрібно буде застосувати числовий тип з більшою точністю, наприклад, з спеціальних бібліотек.

Такий підхід до реалізації до реалізації скоріше характерний для мови С, оскільки вона не має таких засобів абстракції як в С++ та використовується переважно для написання коду низького рівня.

3.3 Реалізація з високим рівнем абстракції

Очевидно, що версія реалізації методу Гаусса-Зейделя з низьким рівнем абстракції, продемонстрована в попередньому підрозділі, має багато аспектів, які можна вдосконалити. Розглянемо інший підхід до реалізації цього методу в С++.

Лістинг 2. Реалізація методу Гаусса-Зейделя з високим рівнем абстракції

```
template <typename Matrix, typename Vector, typename Precision>
void GaussSeidel(const Matrix& m, const Vector& b, Vector& x, const
Precision& eps)
{
	Vector* p = x;
	while(!convergence(x, p, eps))
	{
		precision(m, b, x);
		p = x;
	}
	return;
}
```

У новому варіанті реалізації виправлено вади попереднього. Розглянемо детальніше сигнатуру нового методу

```
template <typename Matrix, typename Vector, typename Precision>
void GaussSeidel(const Matrix& m, const Vector& b, Vector& x, const
Precision& eps);
```

Тепер у визначенні методу використовуються шаблони – засоби узагальненого програмування C++. Це дає нам ряд переваг.

По-перше, тепер реалізації абстрактних типів масиву та матриці, для яких визначено цей метод, не обмежуються тільки масивами, як у попередньому лістингу. Алгоритм є абсолютно незалежним від формату зберігання елементів цих структур. Єдині вимоги, що висуваються зараз до параметрів функції Matrix, Vector та Precision визначаються описом алгоритму.

Можливість використання різних реалізацій матриці в алгоритмі є особливо важливою для наукового програмного забезпечення. У обчисленнях часто використовуються розріджені матриці – це матриці, більша частина елементів яких є нулями. Зберігання матриць такого виду у вигляді масивів є не ефективним, оскільки зберігання нульових елементів не має сенсу. Для таких матриць існують окремі формати зберігання такі як:

- DOK(Dictionary of keys) –словник ключів, зберігаються пари ключ-значення, де ключем є пара значень ряд-стовпчик, а значенням –ненульовий елемент на їх перетині.
- LIL(List of lists) – зберігається список списків рядів матриці, у якому міститься номер стовпчика та значення ненульового елемента на перетині ряда та стовпчика. Зазвичай зберігається відсортованим за індексом стовпчика.
- COO(Coordinate list) – зберігається список кортежів (рядок, стовпчик, значення). Зазвичай зберігається відсортованим за індеском рядка та стовпчика.
- CSR(Compressed sparse row) – зберігається у вигляді трьох одновимірних масивів: з індексами рядків, індексами стовпчиків та значеннями ненульових елементів. Цей метод зберігання є ефективним для швидкого доступу до рядків і множення матриці на вектор.

- CSC(Compressed sparse column) – так само зберігаються дані в трьох масивах, але в першому масиві зберігаються стовпчики. Цей формат зберігання є ефективним для множення матриці на вектор та зберігання розріджених матриць.[11]

Завдяки тому, що функцію оголошено як шаблонну, розріджена матриця, яка зберігається у будь-якому з наведених вище форматів може бути використана для розв'язання системи рівнянь.

Розглянемо вимоги, які висуваються описом алгоритму до типів даних матриця та масив. У рядку

```
Vector* p = x;
```

відбувається виклик копіювального конструктора класу Vector, а в рядку

```
p = x;
```

копіювальне присвоєння. Отже, відповідні операції мають бути визначені у реалізації класу Vector для того, щоб він міг бути параметром цієї шаблонної функції. Для того, щоб до конкретної реалізації класу Matrix застосувати цей метод для її типу повинен бути визначений метод `precision`.Для конкретної реалізації класу Vector також має бути визначений метод `convergence`.

Методи `precision` та `convergence` не для будь-яких реалізацій типів даних матриці та вектора можуть бути визначені однакового, оскільки вони вимагають можливості доступатись до конкретних елементів, а методи доступу до них можуть визначатись по різному. Наприклад, для матриці доступ до елементів може визначатись як для двовимірного масиву – за допомогою двох пар квадратних дужок на кшталт

```
matrix[i][j];
```

або за допомогою окремого методу класу

```
matrix.get(size_t row, size_t col);
```

Алгоритм перевірки, чи виконується умова зупинки ітерацій також може бути реалізовано у різні способи[9]. Наприклад, для розріджених матриць для більшої точності обчислень доцільніше використовувати формулу

$$||Ax^{(k)} - b|| \leq \varepsilon$$

Вирішення цієї проблеми відбувається шляхом визначення двох окремих методів з різними варіантами доступу до елементів. Якщо для конкретної реалізації не існує якогось методу – його потрібно визначити для цієї реалізації.

Таким чином виконується принцип програмування для повторного використання – «programming for reuse». Він полягає в тому, що код можна використовувати за нових умов за допомогою дописування нового коду, а не переписування старого.

Другою проблемою попереднього варіанту реалізації методу була залежність від конкретного числового типу. У методі, визначеному у лістингу 2, не вказано жодного числового типу. Матриця та вектор тепер можуть містити елементи будь-якого числового формату: int, double, float тощо.

У варіанті реалізації з високим рівнем абстракції також відсутня конкретизація типу параметра початкової точності. Тепер він передається як параметр шаблону `Precision`.

Нова версія розглянутого алгоритму має ще одну перевагу – зрозумілість. Якщо у першому варіанті реалізації доволі складно зрозуміти, які саме дії виконувались над масивами елементів та якому кроку алгоритму вони відповідали, то в більш абстрактній версії реалізації, за виключенням ініціалізації вектору p, кожна стрічка коду відповідає кроку алгоритму(див. рис1).

Отже, за допомогою використання засобів абстракції C++ високого рівня нова реалізація методу Гаусса-Зейделя не тільки не має недоліків попередньої, але й має значні переваги: незалежність від числового типу елементів та від реалізацій абстрактних типів даних.

Висновки до розділу 3

На прикладі двох реалізацій розглянутого алгоритму Гаусса-Зейделя було з’ясовано, що реалізація з низьким рівнем абстракції має такі недоліки:

- Низький потенціал для повторного використання коду через залежність від конкретних реалізацій типів даних матриці та вектора, числового типу елементів;
- Високу схильність до помилок;
- Незрозумілість – погану «readability» коду.

Було продемонстровано, що описані вище помилки виправляються за допомого використання конструкцій вищого рівня абстракції мови C++, таких як класи та шаблони.

Висновки

У даній роботі було проаналізовано абстракцію в програмуванні, засоби її реалізації в C++ та її використання в програмуванні алгоритмів лінійної алгебри.

В результаті проведеного дослідження було з’ясовано, що використання засобів високого рівня абстракції в C++, таких як класи та шаблони має значні переваги в порівнянні з використанням коду з низьким рівнем абстракції. Їх використання дозволяє програмувати алгоритми, які не залежать від конкретних реалізацій та числових типів, що створює значний потенціал для повторного використання коду алгоритму. Також перевагою такого підходу є більша зрозумілість коду, що дозволяє відтворити запис дуже близько до покрокового визначення алгоритму.

У майбутньому результати цієї роботи можуть бути використані для реалізації алгоритмів та структур лінійної алгебри засобами мови C++. Також робота може бути продовжена в напрямку збільшення ефективності виконання обчислень в алгоритмах засобами C++. До таких засобів належить запровадження паралельних обчислень з використанням багатопоточності та використання мета-програмування для оптимізації операцій з структурами лінійної алгебри з використанням шаблонів виразів.

Список використаних джерел

1. Budd T. An Introduction to Object-Oriented Programming, 3e / Timothy Budd., 2002. – 611 с.
2. Liskov B. Programming with Abstract Data Types / B. Liskov, S. Zilles. // ACM SIGPLAN Notices. – 1974. – Volume 9, Issue 4.
3. Roberts E. S. Programming Abstractions in C++ / E. S. Roberts, J. Zelenski., 1997. – 682 с.
4. Vandevoorde D. C++ Templates: The Complete Guide / D. Vandevoorde, N. M. Josuttis., 2002. – 552 с.
5. Gottschling P. Discovering Modern C++: An Intensive Course for Scientists, Engineers, and Programmers / Peter Gottschling. – New York: Addison-Wesley, 2015. – 474 с.
6. Musser D. Generic programming/ David R. Musser, Alexander A. Stepanov. – November, 1994.
7. Stroustrup B. Concepts: The Future of Generic Programming/ Bjarne Stroustrup - http://www.w.stroustrup.com/good_concepts.pdf
8. Бублик В.В. Об’єктно-орієнтоване програмування/ В.В. Бублик. – К.: IT-книга - 2015. – 624 с.
9. Saad Y. Iterative methods for sparse linear systems/ Yousef Saad. - SIAM, 2003. — 528 с.
10. George A. Computer Solution of Large Sparse Positive Definite Systems / A. George, J. Liu. – New Jersey: Prentice-Hall, 1981. – 332 с.
11. Писсанецки С. Технология разрежённых матриц = Sparse Matrix Technology/ С. Писсанецки. – 1988. – М.: Мир. — 410 с.
12. Баландин М.Ю. Методы решения СЛАУ большой размерности/ М. Ю. Баландин, Э. П. Шурина. – Новосибирск: Изд-тво НГТУ. – 2000. – 70 с.

13. Meyers S. Effective Modern C++. 42 Specific Ways to Improve Your Use of C++ 11 and C++ 14/ Scott Mayers. — O'Reilly. — 2014. — 334 c.
14. https://www.boost.org/doc/libs/1_65_1/libs/numeric/ublas/doc/index.html
15. http://eigen.tuxfamily.org/index.php?title=Main_Page#Overview
16. http://old.simunova.com/docs/mtl4/html/index.html