Лабораторна робота № 5

ШАБЛОНИ ФУНКЦІЙ І КЛАСІВ. ОБРОБКА ВИНЯТКІВ.

Мета. Отримати практичні навички створення шаблонів і використання їх у програмах С++.

Основний зміст роботи. Створити шаблон заданого класу і використовувати його для даних різних типів.

Короткі теоретичні відомості

Шаблон функції.

Шаблон функції (інакше параметризована функція) визначає загальний набір операцій (алгоритм), які будуть застосовуватися до даних різних типів. При цьому тип даних, над якими функція повинна виконувати операції, передається їй у вигляді параметра на стадії компіляції.

В С++ параметризована функція створюється за допомогою ключового слова *template.* Формат шаблону функції:

template <class тип_даних> тип_поверт_значення назва_функції (список_параметрів) {тіло_функції}

```
#include <iostream>
template<typename T, typename U = int>
T add(T x, U y) {
    return x + y;
}
int main() {
    std::cout << add(5, 10) << std::endl;  // виведе 15
    std::cout << add(3.14, 2.5) << std::endl;  // виведе 5.64
    std::cout << add('a', 3) << std::endl;  // виведе 'd'
    std::cout << add(2, 2.5) << std::endl;  // виведе 4 (компілятор приведе 2.5 до int)
    std::cout << add(2.5, 2) << std::endl;  // виведе 4.5 (компілятор приведе 2 до double)
```

```
    std::cout << add(5) << std::endl;  // виведе 5 (застосується значення за замовчуванням)
    return 0;
}
```

Основні властивості параметрів шаблону функції:

– імена параметрів шаблону повинні бути унікальними в усьому визначеному шаблоні;

– список параметрів шаблону не може бути порожнім;

– у списку параметрів шаблону може бути кілька параметрів, і кожному з них повинно передувати ключове слово class або typename;

– назва параметру шаблону має всі права імені типу у визначеній шаблоном функції;

– певна функція на основі шаблону може мати будь-яку кількість непараметризованих формальних параметрів; може бути непараметризоване значення, що повертається функцією;

– в списку параметрів сигнатури шаблону назви можуть не збігатися з назвами тих же параметрів у визначенні шаблону;

– при конкретизації параметризованої функції необхідно, щоб при виклику функції типи фактичних параметрів, відповідали параметризованим формальним параметрам.

Спеціалізації ШФ:

- звичайна:

```
template <typename T1, typename T2,>
T1 add(T1 a, T2 b) {
    return a + (T1)b;
}
```

- явна:

```
template <>
std::string add<std::string>(std::string a, std::string b) {
    return a + " " + b;
}
```

- змішана:

```
template <class T1>
    string add (T1 a, const char* b) {
    string result = to_string(a)+" " +string(b) ;
   return  result;
}
```

Шаблон класу.

Шаблон класу (інакше параметризований клас) використовується для побудови родового класу. Створюючи родовий клас, ви створюєте ціле сімейство споріднених класів, які можна застосовувати до будь-якого типу даних. Таким чином, тип даних, яким оперує клас, вказується як параметр при створенні об'єкту, що належить до цього класу. Подібно до того, як клас визначає правила побудови і формат окремих об'єктів, шаблон класу визначає спосіб побудови окремих класів. У визначенні класу, що входить в шаблон, назва класу є не назвою окремого класу, а параметризованою назвою сімейства класів.

Загальна форма оголошення параметризованого класу:

template <class тип_даних> class назва_класу {… };

Приклад шаблону класу для опису контейнера на основі динамічного масиву:

```
template <typename T, int size = 10>
class DynamicArray {
private:
    T* data;
    int array_size;

public:
    DynamicArray() : data(new T[size]), array_size(size) {}

    ~DynamicArray() {
        delete[] data;
```

```
        }

        // Копіювання
        DynamicArray(const DynamicArray& other) : data(new
T[other.array_size]), array_size(other.array_size) {
            std::copy(other.data, other.data + array_size, data);
        }

        // Присвоєння
        DynamicArray& operator=(const DynamicArray& other) {
            if (this != &other) {
                T* new_data = new T[other.array_size];
                std::copy(other.data, other.data + other.array_size,
new_data);
                delete[] data;
                data = new_data;
                array_size = other.array_size;
            }
            return *this;
        }

        // Доступ до елементів за індексом
        T& operator[](size_t index) {
            return data[index];
        }

        const T& operator[](size_t index) const {
            return data[index];
        }

        // Додавання елемента в кінець масиву
        void push_back(const T& value) {
            T* new_data = new T[array_size + 1];
            std::copy(data, data + array_size, new_data);
```

```
            new_data[array_size] = value;
            delete[] data;
            data = new_data;
            ++array_size;
        }

        // Видалення останнього елемента масиву
        void pop_back() {
            if (array_size > 0) {
                T* new_data = new T[array_size - 1];
                std::copy(data, data + array_size - 1, new_data);
                delete[] data;
                data = new_data;
                --array_size;
            }
        }

        // Отримання кількості елементів у масиві
        size_t getSize() const {
            return array_size;
        }
    };
```

У цьому випадку ми використовуємо шаблонний параметр size, який за замовчуванням має значення 10. У конструкторі за замовчуванням ми виділяємо пам'ять розміром size.

```
    #include <iostream>
    #include "dynamic_array.h"

    int main() {
        // Створюємо масив зі стандартним розміром 10
        DynamicArray<int> arr1;
        for (int i = 0; i < arr1.getSize(); ++i) {
```

```
        arr1[i] = i * 2;
    }
    std::cout << "arr1: ";
    for (int i = 0; i < arr1.getSize(); ++i) {
        std::cout << arr1[i] << " ";
    }
    std::cout << std::endl;

    // Створюємо масив з розміром 5
    DynamicArray<int, 5> arr2;
    for (int i = 0; i < arr2.getSize(); ++i) {
        arr2[i] = i * 3;
    }
    std::cout << "arr2: ";
    for (int i = 0; i < arr2.getSize(); ++i) {
        std::cout << arr2[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

У цьому прикладі ми створюємо два масиви з використанням класу DynamicArray. Перший масив arr1 створюється за допомогою конструктора за замовчуванням, тобто зі стандартним розміром 10. Ми заповнюємо його елементи і виводимо на екран. Другий масив arr2 створюється з розміром 5, вказаним явно. Ми також заповнюємо його елементи і виводимо на екран.

Основні властивості шаблонів класів:

– методи параметризованого класу автоматично є параметризованими. Їх не обов'язково оголошувати за допомогою *template*.

– дружні функції, які описуються в параметризованому класі, не є автоматично параметризованими функціями, тобто за замовчуванням такі функції є дружніми для всіх класів, які організуються з даного шаблону;

– якщо friend-функція містить у своєму описі параметр типу параметризованого класу, то для кожного створеного за даним шаблоном класу є власна *friend*-функція;

– в рамках параметризованого класу можна визначити *friend*-шаблони (дружні параметризовані класи);

– з одного боку, шаблони можуть бути похідними (унаслідуватися) як від шаблонів, так і від звичайних класів, з іншого боку, вони можуть використовуватися в якості базових для інших шаблонів або класів;

– шаблони методів, не можна описувати як *virtual*;

– локальні класи не можуть містити шаблони в якості своїх елементів.

Методи параметризованих класів. Реалізація методу шаблону класу поза визначення класу, повинна включати такі додаткові два елементи:

– визначення повинно починатися з ключового слова *template,* за яким слідує такий же *список_параметрів_типів* в кутових дужках, який використано у визначенні шаблону класу.

– за *назвою_класу,* перед операцією дозволу доступу **::,** повинен слідувати *список_імен_параметрів* шаблону.

template <список_типів> тип_поверт_значення назва_класу <список_імен_параметрів>:: назва_функції (список_параметрів) {… }.

Щоб реалізувати метод push_back за межами класу, спочатку потрібно визначити його прототип.

```
template<typename T, int size>
class DynamicArray {
public:
    // ...

    void push_back(const T& element);

    // ...
};
```

Тоді визначення методу за межами класу може виглядати так:

```
template<typename T, int size>
void DynamicArray<T, size>::push_back(const T& element) {
    // Реалізація методу
    // ...
}
```

Обробка винятків

Обробка винятків - це механізм в C++, який дозволяє виявляти та реагувати на помилки під час виконання програми. Винятки можуть бути викликані під час виконання програми з різних причин, таких як неправильне введення користувачем даних або помилки в логіці програми.

У C++ для обробки винятків використовують ключові слова try, catch та throw. Ключове слово try використовується для вказівки блоку коду, де можуть виникнути винятки. Ключове слово catch використовується для вказівки блоку коду, який буде виконаний, якщо виняток виникає у блоку try. Ключове слово throw використовується для генерації винятку в будь-якому місці програми.

Приклад коду, який демонструє обробку винятків:

```
#include <iostream>
#include <string>

int main() {
    try {
        int x;
        std::cout << "Enter a positive integer: ";
        std::cin >> x;
        if (x <= 0) {
            throw std::string("The number must be positive.");
        }
        std::cout << "The square of " << x << " is " << x * x << 
std::endl;
    }
    catch (std::string e) {
        std::cout << "Error: " << e << std::endl;
```

```
    }
    catch (...) {
        std::cout << "Unknown error." << std::endl;
    }
    return 0;
}
```

Клас exception

Ієрархія класів exception у C++ включає базовий клас std::exception, який є базовим для багатьох інших класів винятків, що використовуються в стандартній бібліотеці C++.

Він містить лише один метод:

```
class exception {
public:
  exception() noexcept;
  exception(const exception&) noexcept;
  exception& operator=(const exception&) noexcept;
  virtual ~exception();
  virtual const char* what() const noexcept;
};
```

Конструктор за замовчуванням exception() та конструктор копіювання exception(const exception&) копіюють об'єкт.

Оператор присвоєння operator=(const exception&) присвоює значення одного об'єкта іншому об'єкту.

Віртуальний деструктор ~exception() забезпечує правильне звільнення пам'яті.

Віртуальний метод what() повертає рядок, який описує виняткову ситуацію.

Клас std::exception є базовим для всіх класів винятків, і може бути використаний для обробки винятків, які не пов'язані з певним конкретним типом винятку. Якщо виняток, що виник, можна представити як об'єкт класу std::exception, то його можна обробити як виняток загального типу.

Похідні класи:

- std::logic_error використовується для представлення виключень, що виникають при неправильній логіці програми,
- std::runtime_error використовується для представлення виключень, що виникають в результаті помилок під час виконання програми.
- std::bad_alloc, який виникає, коли запит на виділення пам'яті не вдається,
- std::out_of_range, який використовується для представлення виключень, що виникають при спробі отримати доступ до неприпустимого індексу в масиві або векторі,
- std::invalid_argument, який використовується для представлення виключень, що виникають при передачі неправильних аргументів у функцію.

Загалом, класи винятків у C++ дозволяють програмістам легко відловлювати та обробляти винятки, що виникають під час виконання програми, та надають зручний інтерфейс для представлення різних типів винятків.

У C++ можна створювати власні класи винятків, щоб представляти специфічні для вашої програми виняткові ситуації. Для цього потрібно створити клас, який наслідується від класу std::exception або його похідних, та визначити конструктори, які передають потрібну інформацію про виняток.

Приклад створення власного класу винятку:

```
#include <exception>
#include <string>

class MyException : public std::exception {
public:
    MyException(const std::string& message) : message_(message) {}

    const char* what() const noexcept override {
        return message_.c_str();
```

```
    }

private:
    std::string message_;
};
```

У цьому прикладі ми створюємо клас MyException, який наслідується від класу std::exception. Клас містить приватне поле message_, яке містить повідомлення про виняток, та конструктор, який приймає рядок як параметр і присвоює його полю message_. Крім того, ми перевизначаємо віртуальний метод what(), який повертає рядок, який описує виняток.

Отже, при виникненні виняткової ситуації в програмі, можна створити об'єкт класу MyException та передати йому необхідну інформацію про виняток. Далі, у відповідному місці програми можна обробити виняток, перехопивши об'єкт класу MyException або його базового класу std::exception.

Порядок виконання роботи.

1. Створити шаблон заданого класу. Визначити конструктори, деструктор, перевантажену операцію присвоювання ("=") і операції, задані в варіанті завдання (лабораторна 2).

2. Написати програму тестування, в якій перевіряється використання шаблону для стандартних типів даних. Використовувати технологію обробки винятків. Створити власний виняток для шаблону.

3. Виконати тестування.

4. Визначити клас, який буде використовуватися як параметр шаблону. Визначити в класі необхідні функції і перевантажені операції. Використовувати технологію обробки винятків. Створити власний виняток для класу.

5. Написати програму тестування, в якій перевіряється використання шаблону для типу визначеного класу.

6. Виконати тестування.

Методичні вказівки.

1. Для шаблонів списків в якості стандартних типів використовувати символьні, цілі і дійсні типи. Для користувацького типу взяти клас з лабораторної роботи № 1.

2. Для шаблонів масивів в якості стандартних типів використовувати цілі і дійсні типи. Для користувацького типу взяти клас "Комплексне число" *complex.*

```
class complex {
    int re; // дійсна частина
    int im; // уявна частина
public;
    // Необхідні методи і перевантажені операції
};
```

3. Реалізацію шаблону слід розмістити разом з визначенням в заголовному файлі.

4. Тестування повинне бути виконане для всіх типів даних і для всіх операцій.

Зміст звіту.

1. Титульний лист: назва дисципліни, номер і назва роботи, прізвище, назва, по батькові студента, дата виконання.

2. Постановка завдання. Слід дати постановку, тобто вказати шаблон якого класу повинен бути створений, які повинні бути в ньому конструктори, методи, перевантажені операції і т.д. Те ж саме слід вказати для користувацького класу.

3. Визначення шаблону класу з коментарями.

4. Визначення користувацького класу з коментарями.

5. Реалізація конструкторів, деструктора, операції присвоювання і операцій, які задані в варіанті завдання.

6. Те ж саме для призначеного для класу користувача.

7. Результати тестування. Слід вказати для яких типів і які операції перевірені і які виявлені помилки (або не виявлені)

Питання для самоконтролю.

1. У чому сенс використання шаблонів?

2. Які синтаксис/семантика шаблонів функцій?

3. Які синтаксис/семантика шаблонів класів?

4. Напишіть параметризовану функцію сортування масиву методом обміну.

5. Визначте шаблон класу "вектор" – одновимірний масив.

6. Що таке параметри шаблону функції?

7. Перерахуйте основні властивості параметрів шаблону функції.

8. Як записувати параметр шаблону?

9. Чи можна перевантажувати параметризовані функції?

10. Перерахуйте основні властивості параметризованих класів.

11. Чи може бути порожнім список параметрів шаблону? Поясніть.

12. Як викликати параметризовану функцію без параметрів?

13. Чи всі методи параметризованого класу є параметризованими?

14. Чи є дружні функції, описані в параметризованому класі, параметризованими?

15. Чи можуть шаблони класів містити віртуальні методи?

16. Як визначаються методи параметризованих класів поза визначення шаблону класу?