Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Лабораторная работа №7

Выполнил:
студенты 4 курса
группы ИП-216
Андрущенко Ф.А.
Литвинов А. Е.
Русецкий А. С.

Проверил:
преподаватель кафедры ПМиК
Агалаков Антон Александрович

Новосибирск, 2025 г.

Задание

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «память», для хранения одного числа – объекта типа T, используя шаблон классов C++
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций

Спецификация типа данных «память».
ADT TMemory
Данные
Память (тип TMemory, в дальнейшем - память) - это память для хранения «числа» объекта типа T в поле FNumber, и значения «состояние памяти» в поле FState. Объект память - изменяемый. Он имеет два состояния, обозначаемых значениями: «Включена» (_On), «Выключена» (_Off). Её изменяют операции: Записать (Store), Добавить (Add), Очистить (Clear).

Операции

Конструктор	
Начальные значения:	Нет.
Процесс:	Инициализирует поле FNumber объекта «память» (тип TMemory) объектом «число» (тип T) со значением по умолчанию.

	Например для числа типа TFrac со значением 0/1. Память устанавливается в состояние «Выключена», в поле FState «состояние памяти» заносится значение (_Off).
Записать	
Вход:	(E) – объект тип T.
Предусловия:	Нет.
Процесс:	В объект «память» (тип TMemory) в поле FNumber записывается копия объекта E. Память устанавливается в состояние «Включена», в поле FState «состояние памяти» заносится значение (_On).
Выход:	Нет.
Постусловия:	Состояние памяти поле FState – «Включена» (_On).
Взять	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает копию объекта хранящегося в объекте «память» (тип TMemory) в поле FNumber.
Выход:	Объект типа T.
Постусловия:	Состояние памяти поле FState – «Включена» (_On).
Добавить	
Вход:	(E) – число объект типа T.
Предусловия:	Нет.
Процесс:	В поле FNumber объекта «память» (тип TMemory) записывается объект типа T, полученный в результате сложения числа (E) и числа, хранящегося в памяти в поле FNumber.
Выход:	Нет.
Постусловия:	Состояние памяти поле FState – «Включена» (_On).
Очистить	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	В поле числа (FNumber) объекта «память» (тип TMemory) записывается объект типа T со значением по умолчанию. Например, для простой дроби - 0/1. Память (поле FState) устанавливается в состояние «Выключена» (_Off).
Выход:	Нет.
Постусловия:	Состояние памяти поле FState – «Выключена» (_Off).
ЧитатьСостояниеПамяти	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Копирует и возвращает значение поля FState «состояние памяти» объекта «память» (тип TMemory) в формате строки.
Выход:	Значение поля «состояния памяти» (типа String).
Постусловия:	Нет.
ЧитатьЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Копирует и возвращает значение поля «число» (FNumber) объекта «память» (тип TMemory).
Выход:	Объект число (тип T).
Постусловия:	Нет.

Исходный код программы

```
from typing import TypeVar, Generic

T = TypeVar('T')


class TMemory(Generic[T]):
    """Параметризованный класс Память для хранения объекта типа T"""

    class MemoryState:
        """Внутренний класс для состояний памяти"""
        _On = "Включена"
        _Off = "Выключена"

    def __init__(self, default_value: T):
        """
        Конструктор
        Начальные значения:
        - FNumber: объект типа T со значением по умолчанию
        - FState: состояние "Выключена" (_Off)
        """
        self.FNumber = default_value
        self.FState = self.MemoryState._Off

    def Store(self, E: T) -> None:
        """
        Записать
        Вход: E - объект типа T
        Процесс: Записывает копию объекта E в память, устанавливает состояние
"Включена"
        """
        self.FNumber = E
        self.FState = self.MemoryState._On

    def Get(self) -> T:
        """
        Взять
        Выход: копия объекта, хранящегося в памяти
        Процесс: Возвращает копию объекта, устанавливает состояние "Включена"
        """
        self.FState = self.MemoryState._On
        return self.FNumber

    def Add(self, E: T) -> None:
        """
        Добавить
        Вход: E - объект типа T
        Процесс: Добавляет число E к числу в памяти (использует оператор +)
        """
        try:
```

```
            # Проверяем, поддерживает ли тип T операцию сложения
            self.FNumber = self.FNumber + E
            self.FState = self.MemoryState._On
        except TypeError as e:
            raise TypeError(
                f"Тип {type(self.FNumber).__name__} не поддерживает операцию сложения"
            ) from e

    def Clear(self, default_value: T) -> None:
        """
        Очистить
        Процесс: Устанавливает значение по умолчанию, состояние "Выключена"
        """
        self.FNumber = default_value
        self.FState = self.MemoryState._Off

    def ReadMemoryState(self) -> str:
        """
        ЧитатьСостояниеПамяти
        Выход: строковое представление состояния памяти
        """
        return self.FState

    def ReadNumber(self) -> T:
        """
        ЧитатьЧисло
        Выход: объект, хранящийся в памяти
        Процесс: Не изменяет состояние памяти
        """
        return self.FNumber

    def __str__(self) -> str:
        """Строковое представление объекта"""
        return f"Память[Состояние: {self.FState}, Значение: {self.FNumber}]"


# Пример использования с целыми числами
if __name__ == "__main__":
    # Создаем память для целых чисел с начальным значением 0
    memory_int = TMemory[int](0)
    print(f"Создана память: {memory_int}")

    # Записываем значение
    memory_int.Store(42)
    print(f"После записи 42: {memory_int}")

    # Получаем значение
    value = memory_int.Get()
    print(f"Получено значение: {value}")

    # Добавляем значение
    memory_int.Add(8)
```

```
    print(f"После добавления 8: {memory_int}")

    # Читаем состояние
    state = memory_int.ReadMemoryState()
    print(f"Состояние памяти: {state}")

    # Очищаем память
    memory_int.Clear(0)
    print(f"После очистки: {memory_int}")
```

Модульные тесты для тестирования класса

```
import unittest
from typing import List
from tmemory import TMemory


class TestTMemory(unittest.TestCase):
    """Тесты для класса TMemory"""

    def test_initialization(self):
        """Тест инициализации"""
        memory = TMemory[int](0)
        self.assertEqual(memory.FNumber, 0)
        self.assertEqual(memory.FState, "Выключена")
        self.assertEqual(memory.ReadMemoryState(), "Выключена")
        self.assertEqual(memory.ReadNumber(), 0)

    def test_store(self):
        """Тест операции Записать"""
        memory = TMemory[int](0)
        memory.Store(100)
        self.assertEqual(memory.FNumber, 100)
        self.assertEqual(memory.FState, "Включена")
        self.assertEqual(memory.ReadNumber(), 100)

    def test_get(self):
        """Тест операции Взять"""
        memory = TMemory[int](0)
        memory.Store(50)
        value = memory.Get()
        self.assertEqual(value, 50)
        self.assertEqual(memory.FState, "Включена")

    def test_add(self):
        """Тест операции Добавить"""
        memory = TMemory[int](10)
        memory.Store(20)  # Сначала записываем 20
        memory.Add(5)  # Добавляем 5 к 20 = 25
        self.assertEqual(memory.FNumber, 25)
```

```
        self.assertEqual(memory.FState, "Включена")

    def test_clear(self):
        """Тест операции Очистить"""
        memory = TMemory[int](0)
        memory.Store(999)
        memory.Clear(0)
        self.assertEqual(memory.FNumber, 0)
        self.assertEqual(memory.FState, "Выключена")

    def test_read_operations(self):
        """Тест операций чтения"""
        memory = TMemory[str]("default")
        memory.Store("test_value")
        state = memory.ReadMemoryState()
        number = memory.ReadNumber()
        self.assertEqual(state, "Включена")
        self.assertEqual(number, "test_value")

    def test_with_custom_type(self):
        """Тест с пользовательским типом (списком)"""
        memory = TMemory[List[int]]([])
        memory.Store([1, 2, 3])
        self.assertEqual(memory.Get(), [1, 2, 3])
        # Для списков операция Add будет конкатенацией
        memory.Add([4, 5])
        self.assertEqual(memory.Get(), [1, 2, 3, 4, 5])

    def test_add_unsupported_type(self):
        """Тест операции Добавить с неподдерживающим сложение типом"""
        memory = TMemory[dict]({})
        memory.Store({"a": 1})
        with self.assertRaises(TypeError):
            memory.Add({"b": 2})  # Словари не поддерживают операцию +


# Дополнительный пример с пользовательским классом
class Fraction:
    """Пример пользовательского класса для демонстрации"""

    def __init__(self, numerator: int = 0, denominator: int = 1):
        self.numerator = numerator
        self.denominator = denominator

    def __add__(self, other: 'Fraction') -> 'Fraction':
        # Простая реализация сложения дробей
        if self.denominator == other.denominator:
            return Fraction(self.numerator + other.numerator, self.denominator)
        return Fraction(
            self.numerator * other.denominator + other.numerator * self.denominator,
            self.denominator * other.denominator
        )
```

```
    def __eq__(self, other: 'Fraction') -> bool:
        return (self.numerator == other.numerator and
                self.denominator == other.denominator)

    def __str__(self) -> str:
        return f"{self.numerator}/{self.denominator}"


class TestTMemoryWithFraction(unittest.TestCase):
    """Тесты для TMemory с пользовательским типом Fraction"""

    def test_fraction_memory(self):
        """Тест памяти с дробями"""
        default_frac = Fraction(0, 1)
        memory = TMemory[Fraction](default_frac)

        # Записываем дробь
        frac1 = Fraction(1, 2)
        memory.Store(frac1)
        self.assertEqual(memory.Get().numerator, 1)
        self.assertEqual(memory.Get().denominator, 2)

        # Добавляем дробь
        frac2 = Fraction(1, 3)
        memory.Add(frac2)
        result = memory.Get()
        # 1/2 + 1/3 = 5/6
        self.assertEqual(result.numerator, 5)
        self.assertEqual(result.denominator, 6)

        # Очищаем
        memory.Clear(Fraction(0, 1))
        self.assertEqual(memory.ReadNumber().numerator, 0)  # Используем ReadNumber
вместо Get
        self.assertEqual(memory.FState, "Выключена")  # Теперь состояние останется
"Выключена"


if __name__ == "__main__":
    from rich import print
    from rich.panel import Panel
    from rich.console import Console
    from unittest import TextTestRunner, TestResult, TestSuite

    console = Console()

    class RichTestResult(TestResult):
        def __init__(self, stream, descriptions, verbosity):
            super().__init__(stream, descriptions, verbosity)

        def startTest(self, test):
```

```
            super().startTest(test)
            console.print(f"[cyan]Running:[/cyan] {test._testMethodName}")

        def addSuccess(self, test):
            super().addSuccess(test)
            console.print(f"[green]✓ PASS:[/green] {test._testMethodName}")

        def addFailure(self, test, err):
            super().addFailure(test, err)
            console.print(f"[red]✗ FAIL:[/red] {test._testMethodName}")

        def addError(self, test, err):
            super().addError(test, err)
            console.print(f"[magenta]💥 ERROR:[/magenta] {test._testMethodName}")

    loader = unittest.defaultTestLoader
    suite = TestSuite()
    suite.addTests(loader.loadTestsFromTestCase(TestTMemory))
    suite.addTests(loader.loadTestsFromTestCase(TestTMemoryWithFraction))

    runner = TextTestRunner(resultclass=RichTestResult, verbosity=0)
    result = runner.run(suite)

    console.print(
        Panel.fit(
            f"[green]Passed: {result.testsRun - len(result.failures) -
len(result.errors)}[/green]\n"
            f"[red]Failed: {len(result.failures)}[/red]\n"
            f"[magenta]Errors: {len(result.errors)}[/magenta]\n"
            f"[yellow]Total: {result.testsRun}[/yellow]",
            title="Test Results",
        )
    )
```

Результат тестирования методов класса