Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

## Кафедра прикладной математики и кибернетики

Современные технологии программирования

Практическая работа №7

«Параметризованный абстрактный тип данных «Память»»

Выполнил: студент 4 курса

группы ИП-111 Кузьменок Денис Витальевич

Проверил преподаватель: Зайцев Михаил Георгиевич

Новосибирск, 2024 г.

# Цель

Сформировать практические навыки реализации параметризованного абстрактного типа данных с помощью шаблона классов С++.

# Задание

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «память», для хранения одного числа – объекта типа T, используя шаблон классов С++.
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).

# Реализация:

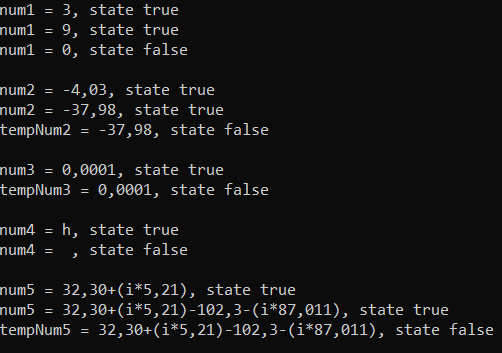


Рис. 1 – Результат проверки работоспособности программы.

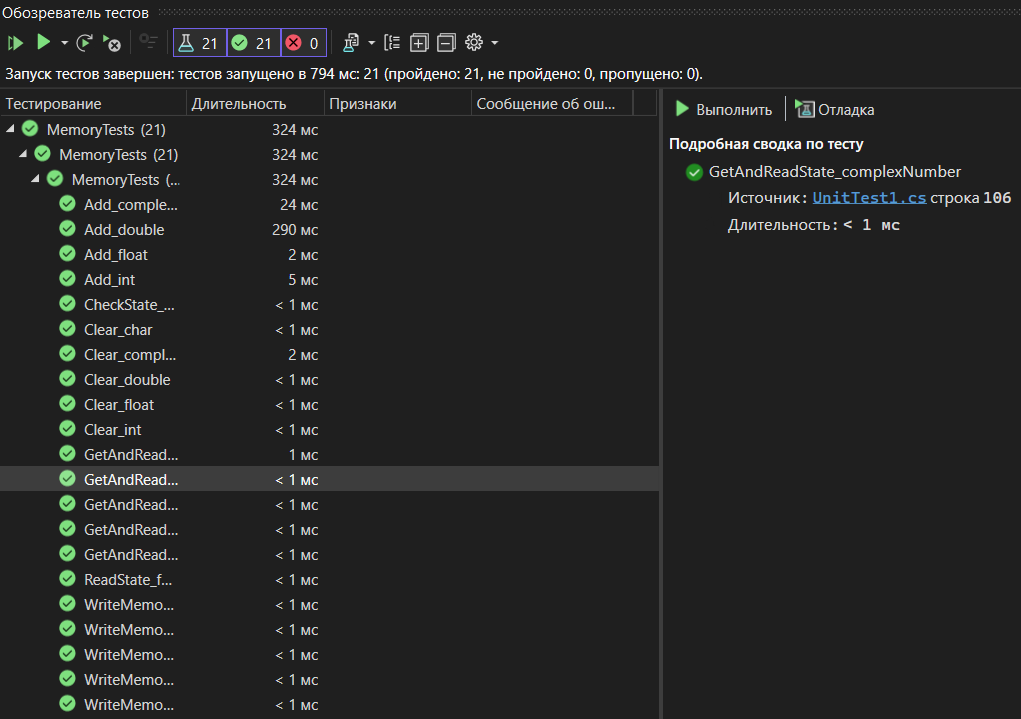


Рис. 2 – Результат выполнения модульных тестов.

## WriteMemory\_int

* + **Что проверяет:** проверяет, что запись числа в память корректно сохраняется и может быть считано.

## Входные значения:

* + - TMemory<int> num1.WriteMemory(3) — создаётся объект числа со значением 3.
  + **Ожидаемое значение:** 3 — число, сохранённое в памяти.

## WriteMemory\_float

* + **Что проверяет:** проверяет, что запись числа в память корректно сохраняется и может быть считано.

## Входные значения:

* + - TMemory<float> num1.WriteMemory(0.08f) — создаётся объект числа со значением 0.08.
  + **Ожидаемое значение:** 0.08 — число, сохранённое в памяти после перезаписи.

## WriteMemory\_NegativeDouble

* + **Что проверяет:** проверяет, что запись числа в память корректно сохраняется и может быть считано.

## Входные значения:

* + - TMemory<double> num1.WriteMemory(-7.92) — создаётся объект числа со значением -7.92.
  + **Ожидаемое значение:** -7.92 — число, сохранённое в памяти после перезаписи.

## WriteMemory\_char

* + **Что проверяет:** проверяет, что запись числа в память корректно сохраняется и может быть считано.

## Входные значения:

* + - TMemory<char> num1.WriteMemory(‘b’) — создаётся объект числа со значением ‘b’.

## Ожидаемое значения:

* + - ‘b’ — число, сохранённое в памяти после перезаписи.

## WriteMemory\_complexNumber

* + **Что проверяет:** проверяет, что запись числа в память корректно сохраняется и может быть считано.

## Входные значения:

## TEditor complexNumber = new TEditor();

## complexNumber.WriteNumber(“72,9+(i\*0,62)

* + - TMemory<TEditor> num1.WriteMemory(complexNumber) — создаётся объект числа со значением “72,9+(i\*0,62)”.

## Ожидаемое значения:

* + - “72,9+(i\*0,62)” — число, сохранённое в памяти после перезаписи.

## GetAndReadState\_int

* + **Что проверяет:** достает из памяти сохраненное значение.

## Входные значения:

* + - TMemory<int> num1.WriteMemory(5); — создаётся объект памяти.
    - num1.Get()
  + **Ожидаемое значение:** 5 — получение числа, сохраненного в памяти.

## GetAndReadState\_float

* + **Что проверяет:** достает из памяти сохраненное значение.

## Входные значения:

* + - TMemory<float> num1.WriteMemory(82.125f); — создаётся объект памяти.
    - num1.Get()
  + **Ожидаемое значение:** 82.125f — получение числа, сохраненного в памяти.

## GetAndReadState\_double

* + **Что проверяет:** достает из памяти сохраненное значение.

## Входные значения:

* + - TMemory<double> num1.WriteMemory(44.32); — создаётся объект памяти.
    - num1.Get()
  + **Ожидаемое значение:** 44.32 — получение числа, сохраненного в памяти.

## GetAndReadState\_char

* + **Что проверяет:** достает из памяти сохраненное значение.

## Входные значения:

* + - TMemory<char> num1.WriteMemory(‘s’); — создаётся объект памяти.
    - num1.Get()
  + **Ожидаемое значение:** ‘s’ — получение числа, сохраненного в памяти.

## GetAndReadState\_complexNumber

* + **Что проверяет:** достает из памяти сохраненное значение.

## Входные значения:

## TEditor complexNumber = new TEditor();

## complexNumber.WriteNumber(“83,3-(i\*0,29)

* + - TMemory<TEditor> num1.WriteMemory(complexNumber) — создаётся объект числа со значением “83,3-(i\*0,29)”.
    - num1.Get()
  + **Ожидаемое значение:** “83,3-(i\*0,29)” — получение числа, сохраненного в памяти.

## Add\_int

* + **Что проверяет:** проверяет, что состояние памяти обновляется при добавлении данных.

## Входные значения:

* + - TMemory<int> num1.WriteMemory(10); — создаётся объект памяти.
    - Добавляется 5.
  + **Ожидаемое значение:** 15 — состояние памяти должно отражать наличие данных.

## Add\_float

* + **Что проверяет:** проверяет, что состояние памяти обновляется при добавлении данных.

## Входные значения:

* + - TMemory<float> num1.WriteMemory(21.6652f); — создаётся объект памяти.
    - Добавляется 1.92.
  + **Ожидаемое значение:** 23.5852f — состояние памяти должно отражать наличие данных.

## Add\_double

* + **Что проверяет:** проверяет, что состояние памяти обновляется при добавлении данных.

## Входные значения:

* + - TMemory<double> num1.WriteMemory(-38.81); — создаётся объект памяти.
    - Добавляется -72.9015.
  + **Ожидаемое значение:** -111.7115 — состояние памяти должно отражать наличие данных.

## Add\_complexNumber

* + **Что проверяет:** проверяет, что состояние памяти обновляется при добавлении данных.

## Входные значения:

## TEditor complexNumber = new TEditor();

## complexNumber.WriteNumber(“-102,3-(i\*87,011)

* + - TMemory<TEditor> num1.WriteMemory(complexNumber) — создаётся объект числа со значением “-102,3-(i\*87,011)”.
    - Добавляется “32,30+(i\*5,21)”.
  + **Ожидаемое значение:** “32,30+(i\*5,21)-102,3-(i\*87,011)”— состояние памяти должно отражать наличие данных.

## CheckState\_true

* + **Что проверяет:** проверяет, что состояние памяти изменяется при добавлении в неё числа.

## Входные значения:

* + - TMemory<int> num1.WriteMemory(2); — создаётся объект памяти.
    - num1.ReadState().
  + **Ожидаемое значение:** true — т.к. в памяти есть данные.

## ReadState\_false

* + **Что проверяет:** проверяет, что состояние памяти изменяется при добавлении в неё числа.

## Входные значения:

* + - TMemory<int> num1.WriteMemory(-22); — создаётся объект памяти.
    - int number = num1.Get()
    - num1.ReadState().
  + **Ожидаемое значение:** false — т.к. данные из памяти были считаны.

## Clear\_int

* + **Что проверяет:** проверяет, что для данного типа данных устанавливается значение по умолчанию.

## Входные значения:

* + - TMemory<int> num1.WriteMemory(90); — создаётся объект памяти.
    - num1.Clear()
  + **Ожидаемое значение:** 0 — значение по умолчанию для int.

## Clear\_float

* + **Что проверяет:** проверяет, что для данного типа данных устанавливается значение по умолчанию.

## Входные значения:

* + - TMemory<float> num1.WriteMemory(51.92f); — создаётся объект памяти.
    - num1.Clear()
  + **Ожидаемое значение:** 0.0f — значение по умолчанию для float.

## Clear\_double

* + **Что проверяет:** проверяет, что для данного типа данных устанавливается значение по умолчанию.

## Входные значения:

* + - TMemory<double> num1.WriteMemory(-0.67); — создаётся объект памяти.
    - num1.Clear()
  + **Ожидаемое значение:** 0.0 — значение по умолчанию для double.

## Clear\_char

* + **Что проверяет:** проверяет, что для данного типа данных устанавливается значение по умолчанию.

## Входные значения:

* + - TMemory<char> num1.WriteMemory(‘c’); — создаётся объект памяти.
    - num1.Clear()
  + **Ожидаемое значение:** ‘\0’ — значение по умолчанию для char.

## Clear\_complexNumber

* + **Что проверяет:** проверяет, что для данного типа данных устанавливается значение по умолчанию.

## Входные значения:

* + - TMemory<TEditor> num1.WriteMemory(complexNumber); — создаётся объект памяти.
    - num1.Clear()
  + **Ожидаемое значение:** “0,+(i\*0,)” — значение по умолчанию для TEditor.

# Вывод

В результате работы над лабораторной работой были сформированы практические навыки реализации параметризованного абстрактного типа данных с помощью шаблона классов С#, разработки функций классов на языке С#, разработка модульных тестов для тестирования функций классов и выполнения модульного тестирования на языке С# с помощью средств автоматизации Visual Studio.

# Листинг программы:

**Program.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**using** **lab6**;

**namespace** **lab7**

{

**class** **Program**

{

**static** **void** **Main**(**string**[] args)

{

TEditor editor = **new** TEditor();

editor.WriteNumber("-62,6+(i\*52,9)");

TMemory<**int**> num1 = **new** TMemory<**int**>();

TMemory<**float**> num2 = **new** TMemory<**float**>();

TMemory<**double**> num3 = **new** TMemory<**double**>();

TMemory<**char**> num4 = **new** TMemory<**char**>();

TMemory<TEditor> num5 = **new** TMemory<TEditor>(editor);

num1.WriteMemory(**3**);

Console.WriteLine($"num1 = {num1.ReadNumber()}, " + (num1.ReadState() ? "state true" : "state false"));

num1.Add(**6**);

Console.WriteLine($"num1 = {num1.ReadNumber()}, " + (num1.ReadState() ? "state true" : "state false"));

num1.Clear();

Console.WriteLine($"num1 = {num1.ReadNumber()}, " + (num1.ReadState() ? "state true\n" : "state false\n"));

num2.WriteMemory(-**4.03f**);

Console.WriteLine($"num2 = {num2.ReadNumber()}, " + (num2.ReadState() ? "state true" : "state false"));

num2.Add(-**33.95f**);

Console.WriteLine($"num2 = {num2.ReadNumber()}, " + (num2.ReadState() ? "state true" : "state false"));

**float** tempNum2 = num2.Get();

Console.WriteLine($"tempNum2 = {tempNum2}, " + (num2.ReadState() ? "state true\n" : "state false\n"));

num3.WriteMemory(**0.0001**);

Console.WriteLine($"num3 = {num3.ReadNumber()}, " + (num3.ReadState() ? "state true" : "state false"));

**double** tempNum3 = num3.Get();

Console.WriteLine($"tempNum3 = {tempNum3}, " + (num3.ReadState() ? "state true\n" : "state false\n"));

num4.WriteMemory('h');

Console.WriteLine($"num4 = {num4.ReadNumber()}, " + (num4.ReadState() ? "state true" : "state false"));

num4.Clear();

Console.WriteLine($"num4 = {num4.ReadNumber()}, " + (num4.ReadState() ? "state true\n" : "state false\n"));

num5.WriteMemory(editor);

Console.WriteLine($"num5 = {num5.ReadNumber()}, " + (num5.ReadState() ? "state true" : "state false"));

//num5.Add("-102,3-(i\*87,011)");

Console.WriteLine($"num5 = {num5.ReadNumber()}, " + (num5.ReadState() ? "state true" : "state false"));

TEditor tempNum5 = num5.Get();

Console.WriteLine($"tempNum5 = {tempNum5.ReadNumber()}, " + (num5.ReadState() ? "state true\n" : "state false\n"));

}

}

}

**TMemory.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **lab7**

{

**public** **interface** IWritable<T>

{

**void** **WriteMemory**(T **value**);

}

**public** **interface** IReadable<T>

{

T **ReadNumber**();

}

**public** **interface** IClearable

{

**void** **Clear**();

}

**public** **interface** IStateTrackable

{

**bool** **ReadState**();

}

**public** **class** **TMemory**<T> : IWritable<T>, IReadable<T>, IClearable, IStateTrackable

{

**public** T data;

**private** **bool** \_state = **false**;

**public** **TMemory**()

{

data = **default**(T);

\_state = **false**;

}

**public** **TMemory**(T initData)

{

data = initData;

\_state = **false**;

}

**public** **void** **WriteMemory**(T number)

{

data = number;

\_state = **true**;

}

**public** T **Get**()

{

**if** (!\_state)

{

**throw** **new** **InvalidOperationException**("Memory has not been written to.");

}

\_state = **false**;

**return** data;

}

**public** **void** **Add**(T addComplex)

{

data = (**dynamic**)data + (**dynamic**)addComplex;

\_state = **true**;

}

**public** **void** **Clear**()

{

data = **default**(T);

\_state = **false**;

}

**public** **bool** **ReadState**()

{

**return** \_state;

}

**public** T **ReadNumber**()

{

**return** data;

}

}

}

**UnitTests1.cs**

**using** **Microsoft.VisualStudio.TestTools.UnitTesting**;

**using** **System**;

**using** **lab7**;

**using** **lab6**;

**namespace** **MemoryTests**

{

[TestClass]

**public** **class** **MemoryTests**

{

[TestMethod]

**public** **void** **WriteMemory\_int**()

{

TMemory<**int**> num1 = **new** TMemory<**int**>();

num1.WriteMemory(**3**);

**int** actual = **3**;

Assert.AreEqual(actual, num1.ReadNumber());

}

[TestMethod]

**public** **void** **WriteMemory\_float**()

{

TMemory<**float**> num1 = **new** TMemory<**float**>();

num1.WriteMemory(**0.08f**);

**float** actual = **0.08f**;

Assert.AreEqual(actual, num1.ReadNumber());

}

[TestMethod]

**public** **void** **WriteMemory\_NegativeDouble**()

{

TMemory<**double**> num1 = **new** TMemory<**double**>();

num1.WriteMemory(-**7.92**);

**double** actual = -**7.92**;

Assert.AreEqual(actual, num1.ReadNumber());

}

[TestMethod]

**public** **void** **WriteMemory\_char**()

{

TMemory<**char**> num1 = **new** TMemory<**char**>();

num1.WriteMemory('b');

**char** actual = 'b';

Assert.AreEqual(actual, num1.ReadNumber());

}

[TestMethod]

**public** **void** **WriteMemory\_complexNumber**()

{

TEditor complexNumber = **new** TEditor();

complexNumber.WriteNumber("-0,3+(i\*5,09)");

TMemory<**string**> complexNumberMemory = **new** TMemory<**string**>("-0,3+(i\*5,09)");

Assert.AreEqual(complexNumber.ReadNumber(), complexNumberMemory.ReadNumber());

}

[TestMethod]

**public** **void** **GetAndReadState\_int**()

{

TMemory<**int**> num1 = **new** TMemory<**int**>();

num1.WriteMemory(**5**);

**int** temp = num1.Get();

Assert.AreEqual(temp, num1.ReadNumber());

Assert.IsFalse(num1.ReadState());

}

[TestMethod]

**public** **void** **GetAndReadState\_float**()

{

TMemory<**float**> num1 = **new** TMemory<**float**>();

num1.WriteMemory(**82.125f**);

**float** temp = num1.Get();

Assert.AreEqual(temp, num1.ReadNumber());

Assert.IsFalse(num1.ReadState());

}

[TestMethod]

**public** **void** **GetAndReadState\_double**()

{

TMemory<**double**> num1 = **new** TMemory<**double**>();

num1.WriteMemory(**44.32**);

**double** temp = num1.Get();

Assert.AreEqual(temp, num1.ReadNumber());

Assert.IsFalse(num1.ReadState());

}

[TestMethod]

**public** **void** **GetAndReadState\_char**()

{

TMemory<**char**> num1 = **new** TMemory<**char**>();

num1.WriteMemory('s');

**char** temp = num1.Get();

Assert.AreEqual(temp, num1.ReadNumber());

Assert.IsFalse(num1.ReadState());

}

[TestMethod]

**public** **void** **GetAndReadState\_complexNumber**()

{

TEditor complexNumber = **new** TEditor();

complexNumber.WriteNumber("82,3-(i\*0,29)");

TMemory<TEditor> complexNumberMemory = **new** TMemory<TEditor>(complexNumber);

complexNumberMemory.WriteMemory(complexNumber);

TEditor actual = complexNumberMemory.ReadNumber();

Assert.AreEqual(complexNumberMemory.Get(), actual);

Assert.IsFalse(complexNumberMemory.ReadState());

}

[TestMethod]

**public** **void** **Add\_int**()

{

TMemory<**int**> num1 = **new** TMemory<**int**>();

num1.WriteMemory(**10**);

num1.Add(**5**);

**int** actual = **15**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Add\_float**()

{

TMemory<**float**> num1 = **new** TMemory<**float**>();

num1.WriteMemory(**21.6652f**);

num1.Add(**1.92f**);

**float** actual = **23.5852f**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Add\_double**()

{

TMemory<**double**> num1 = **new** TMemory<**double**>();

num1.WriteMemory(-**38.81**);

num1.Add(-**72.9015**);

**double** actual = -**111.7115**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Add\_complexNumber**()

{

TEditor complexNumber = **new** TEditor();

complexNumber.WriteNumber("-102,3-(i\*87,011)");

TMemory<**string**> complexNumberMemory = **new** TMemory<**string**>("32,30+(i\*5,21)");

complexNumberMemory.Add(complexNumber.ReadNumber());

**string** actual = "32,30+(i\*5,21)-102,3-(i\*87,011)";

Assert.AreEqual(complexNumberMemory.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **CheckState\_true**()

{

TMemory<**int**> num1 = **new** TMemory<**int**>();

num1.WriteMemory(**2**);

Assert.IsTrue(num1.ReadState());

}

[TestMethod]

**public** **void** **ReadState\_false**()

{

TMemory<**int**> num1 = **new** TMemory<**int**>();

num1.WriteMemory(-**22**);

**int** number = num1.Get();

Assert.IsFalse(num1.ReadState());

}

[TestMethod]

**public** **void** **Clear\_int**()

{

TMemory<**int**> num1 = **new** TMemory<**int**>();

num1.WriteMemory(**90**);

num1.Clear();

**int** actual = **0**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Clear\_float**()

{

TMemory<**float**> num1 = **new** TMemory<**float**>();

num1.WriteMemory(**51.92f**);

num1.Clear();

**float** actual = **0.0f**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Clear\_double**()

{

TMemory<**double**> num1 = **new** TMemory<**double**>();

num1.WriteMemory(-**0.67**);

num1.Clear();

**double** actual = **0.0**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Clear\_char**()

{

TMemory<**char**> num1 = **new** TMemory<**char**>();

num1.WriteMemory('c');

num1.Clear();

**char** actual = '\0';

Assert.AreEqual(num1.ReadNumber(), actual);

}

[TestMethod]

**public** **void** **Clear\_complexNumber**()

{

TEditor editor = **new** TEditor();

editor.WriteNumber("0.93+(i\*0,)");

TMemory<**string**> num1 = **new** TMemory<**string**>();

num1.WriteMemory(editor.ReadNumber());

num1.Clear();

**string** actual = **null**;

Assert.AreEqual(num1.ReadNumber(), actual);

}

}

}