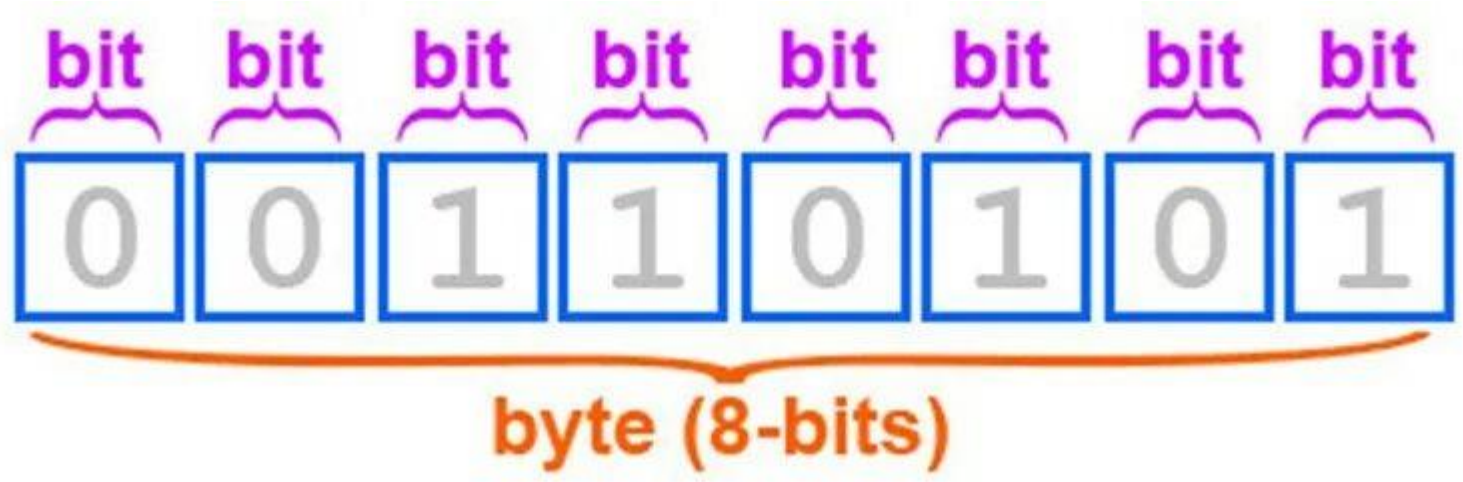
Переменные, типы данных и управление памятью в C#

Информатика

Фоменко М.Ю.

Физическая и логическая организация оперативной памяти

- **Бит** - **минимальная единица измерения информации** в цифровой технике. Может принимать одно из двух значений: 0 или 1.
- **Байт** - совокупность битов, обрабатываемая компьютером одновременно.



Физическая и логическая организация оперативной памяти

- **Ячейка памяти** в вычислительной технике - электронная схема, которая хранит один бит двоичной информации. Она должна быть настроена на сохранение логической 1 (высокий уровень напряжения) и сброшена для сохранения логического 0 (низкий уровень напряжения).
- Своими словами про 1 байт в соверменных

рис. 1

Физическая и логическая организация оперативной памяти

- Адрес памяти - уникальное целое число, которое идентифицирует конкретный байт внутри всего пространства оперативной памяти.
- Если объем ОЗУ составляет 4 гигабайта, это означает, что в этом массиве содержится 2^32 (более 4 миллиардов) байтов
- Адреса будут варьироваться от 0 до 2^32 – 1, представление 16ая система счисления

рис. 1

Физическая и логическая организация оперативной памяти

- Первый адрес тут – 2.147.478.016 в десятичной.
- Если процессору нужно сохранить чисто 1_000_000_000 в ячейку …EA00 – он запишет двоичное представление в блоки EA00, EA01, EA02, EA03
- 00111011_10011010_11001010_00000000
- Куда какой бит?

рис. 1

Физическая и логическая организация оперативной памяти

- Little-Endian и Big-Endian
- Little-Endian - младший байт (little end) записывается в ячейку с младшим адресом.
- Современные ПК, включая процессоры Intel и AMD (архитектура x86, x86-64).
- Big-Endian - старший байт (big end) записывается в ячейку с младшим адресом, старые архитектуры (IBM mainframes, PowerPC, SPARC)

рис. 1

Физическая и логическая организация оперативной памяти

- Процессор запишет байты в следующем порядке:
- В ячейку 0xEA00 запишется байт 0x00 (8 нулей).
- В ячейку 0xEA01 запишется байт 0xCA (11001010).
- В ячейку 0xEA02 запишется байт 0x9A (10011010).
- В ячейку 0xEA03 запишется байт 0x3B (00111011).

рис. 1

Данные в вычислительной системе

- Переменная - именованная область памяти, в которой хранится некоторое значение.
- С формальной точки зрения, переменная в языке программирования представляет собой триаду: **идентификатор (имя), тип и значение**, ассоциированное с определенной областью памяти.

Данные в вычислительной системе

int userAge;

1. Этап компиляции: Компилятор C# анализирует данную декларацию.
    1. Он распознает int как спецификацию типа данных (System.Int32).
    2. Он распознает userAge как идентификатор, который будет использоваться в коде для доступа к этой области памяти.
    3. Компилятор генерирует промежуточный код (Intermediate Language, IL), который содержит инструкцию для среды исполнения (.NET CLR) о необходимости выделить на стеке вызовов 4 байта памяти, когда исполнение дойдет до области видимости данной переменной.

Данные в вычислительной системе

int userAge;

1. Этап исполнения (Runtime):
    1. Когда поток исполнения входит в метод, где объявлена переменная userAge, среда CLR выполняет инструкцию, сгенерированную компилятором.
    2. Она резервирует в текущем фрейме стека непрерывный блок из 4 байтов.
    3. С этого момента и до конца области видимости переменной, идентификатор userAge символически связан с начальным адресом этого 4-байтового блока. Любая операция чтения или записи по имени userAge будет транслирована в операцию с соответствующей ячейкой памяти.

Типы данных

Тип данных - формальная спецификация, которая определяет два фундаментальных аспекта:

- Диапазон или набор значений, которые может принимать переменная (bool, int, string)
- Операции, применимые к переменным данного типа

(Попытка применить, скажем, операцию умножения к переменной типа bool приведет к ошибке на этапе компиляции, поскольку такая операция не определена для данного типа)

Типы данных

C# является **языком со статической типизацией**.

Это означает, что тип каждой переменной известен и проверяется на этапе компиляции, а не во время выполнения программы.

Типы данных

Функция типа данных на уровне работы компьютера - служить «инструкцией по интерпретации».

Одна и та же последовательность битов в памяти может иметь совершенно разный смысл в зависимости от того, какой тип к ней применяется.

Типы данных

Допустим, в 4-байтовой ячейке памяти хранится следующая битовая последовательность (для простоты показан только один байт):

- **0100 0001**
- Если INT – это 65
- Если CHAR – это ‘A’
- Если FLOAT or DOUBLE – среда исполнения бы формировала часть его мантиссы или экспоненты, представляя совершенно иное числовое значение.

Типы данных

Тип данных - фундаментальный контракт между программистом и компилятором/средой исполнения.

Программист объявляет о своих намерениях, а система гарантирует, что память будет выделена корректно и что биты в этой памяти будут последовательно и предсказуемо интерпретироваться в соответствии с объявленным типом.

Стек и Управляемая Куча

Очереди предполагают хранение операций или переменных и работу с ними в организованном порядке.

LIFO (Last In First Out) - «последним пришёл - первым ушёл». В такой очереди первым обрабатывается самый последний добавленный запрос.

FIFO (First In First Out) - «первым пришёл - первым ушёл». В такой структуре данные добавляются в конец очереди, а извлекаются из начала.

FIFO (First-In, First-Out) - Принцип Классической Очереди

Элемент, который был добавлен в коллекцию **раньше всех**, будет обработан **первым**. Это самый справедливый и интуитивно понятный порядок.

- Приходит первый покупатель (A). Он становится первым в очереди.
- Приходит второй покупатель (B). Он встает в конец очереди, за покупателем A.
- Приходит третий покупатель (C). Он встает за покупателем B.

[Касса] <-- (A) <-- (B) <-- (C)

Обслужат (A), после (B), а затем (C). Порядок поступления полностью совпадает с порядком обслуживания.

FIFO (First-In, First-Out) - Принцип Классической Очереди

Где это используется?

- Кнопка "Назад" в браузере: Она возвращает вас на последнюю посещенную страницу.
- Функция "Отменить" (Ctrl+Z): Отменяется последнее выполненное действие.
- Стек вызовов функций (Call Stack): Когда одна функция вызывает другую, новая "кладется наверх" стека. Завершается всегда та, что наверху (та, что была вызвана последней).

LIFO (Last-In, First-Out) - Принцип Стека

Элемент, который был добавлен в коллекцию **последним**, будет обработан **первым**.

- Вы моете первую тарелку (A) и кладете ее на полку.
- Моете вторую тарелку (B) и кладете ее наверх тарелки A.
- Моете третью тарелку (C) и кладете ее наверх тарелки B.

Какую вы возьмете? Естественно, верхнюю — (C), потому что до нижних добраться неудобно и опасно.

После (C) вы возьмете (B), и только в самом конце - (A). Порядок извлечения является обратным порядку добавления.

LIFO (Last-In, First-Out) - Принцип Стека

Где это используется?

- Очередь на печать: Документы печатаются в том порядке, в котором вы их отправили.
- Обработка запросов на сервере: Сервер обрабатывает запросы от пользователей в порядке их поступления, чтобы все были обслужены честно.
- Сообщения в чате: Вы видите сообщения в том порядке, в котором они были отправлены.

Стек

Стек (The Stack) - высокоорганизованная, строго структурированная область памяти, функционирующая по принципу LIFO (Last-In, First-Out).

Стек предназначен для управления исполнением программного кода на уровне методов (в C# их принято называть методами, в других языках — функциями или процедурами). Когда вызывается некий метод, для него на вершине стека выделяется блок памяти, называемый **фреймом стека** (stack frame).

Стек

Содержимое фрейма стека:

- Параметры, переданные в метод - значения аргументов, с которыми был вызван метод, копируются в его фрейм.
- Локальные переменные - память для всех переменных, объявленных внутри метода, резервируется здесь же.
- Адрес возврата - адрес в коде вызвавшего метода, куда должен вернуться поток исполнения после завершения текущего метода.

Стек

Вы можете добавить новую тарелку (операция push) только наверх стопки. Снять тарелку (операция pop) вы также можете только сверху. Доступ к тарелкам в середине стопки невозможен без снятия всех вышележащих.

Когда метод завершает свою работу (через оператор return или достигнув конца своего блока), его фрейм "снимается" (pop) с вершины стека. Вся память, занимаемая этим фреймом, мгновенно считается свободной. Управление возвращается по сохраненному адресу возврата.

Стек

Характеристика стека:

- Управление жизненным циклом детерминировано. Память выделяется при входе в метод и гарантированно освобождается при выходе.
- Операции выделения и освобождения памяти на стеке сводятся к инкременту или декременту одного регистра процессора - указателя стека
- Размер стека для каждого потока исполнения фиксирован и относительно невелик (обычно порядка 1-2 мегабайт). Попытка разместить на стеке больше данных, чем он может вместить приводит к критической ошибке времени выполнения **StackOverflowException**

Управляемая Куча

Управляемая куча (The Managed Heap) в .NET — это область памяти, выделенная средой CLR (Common Language Runtime) для хранения объектов, созданных приложением.

В отличие от стека, момент освобождения памяти в куче не является детерминированным. Объект, созданный в одном методе, может продолжать использоваться в других частях программы еще долго после того, как создавший его метод завершил работу.

Управляемая Куча

Память в куче выделяется всякий раз, когда в коде используется оператор new. Этот оператор сигнализирует среде CLR о необходимости выполнить следующие действия:

1. Найти в куче непрерывный блок свободной памяти, достаточный для размещения нового объекта.
2. Инициализировать этот блок памяти (обнулить его).
3. Вызвать конструктор объекта для его окончательной настройки.
4. Вернуть адрес (ссылку) на начало выделенного блока памяти. Этот адрес и будет значением переменной, которой присваивается результат new.

Управляемая Куча

Так как переменная может занимать память и после выполнения метода, для определения момента «когда можно очистить память от переменной» используется **Сборщик мусора.**

Периодически Garbage Collector приостанавливает выполнение программы и производит «сборку мусора».

Любой объект в куче, до которого невозможно добраться, следуя по ссылкам от "корней", считается "мусором" — то есть, более не используемым.

Управляемая Куча

Характеристики кучи:

- Объекты в куче существуют до тех пор, пока на них существует хотя бы одна достижимая ссылка.
- Аллокация памяти в куче требует поиска подходящего блока и может быть на порядки медленнее стековой аллокации.
- Память освобождается не в предсказуемый момент, а тогда, когда GC сочтет это необходимым.
- Куча может занимать гигабайты памяти, ограничиваясь лишь объемом ОЗУ и архитектурой ОС.
- *GC избавляет программиста от необходимости вручную освобождать память, предотвращая целый класс ошибок, таких как утечки памяти (memory leaks) и висячие указатели (dangling pointers), характерных для неуправляемых языков вроде C++*

Куча Стеков

Признак	**Стек (Stack)**	**Управляемая Куча (Managed Heap)**
Назначение	Управление потоком исполнения, локальные данные	Хранение объектов с динамическим временем жизни
Структура	LIFO (Last-In, First-Out)	Неструктурированный пул памяти
Скорость выделения	Чрезвычайно высокая (сдвиг указателя)	Относительно медленная (поиск свободного блока)
Скорость освобождения	Чрезвычайно высокая (сдвиг указателя)	Недетерминированная, выполняется Сборщиком Мусора
Время жизни данных	Ограничено временем жизни вызова метода	От создания до тех пор, пока существуют ссылки
Управление	Автоматическое, компилятором и CLR	Автоматическое, Сборщиком Мусора (GC)
Размер	Малый, фиксированный	Большой, динамически растет
Риск ошибки	`StackOverflowException` (переполнение)	Паузы на сборку мусора, фрагментация памяти

Значимый тип данных

Значимый тип - такой тип данных, переменная которого хранит свое значение непосредственно внутри области памяти, выделенной для самой переменной.

Ключевые представители:

- Все числовые примитивы: int, double, float, decimal, byte и т.д.
- Логический тип: bool
- Символьный тип: char
- Все структуры, объявленные с помощью ключевого слова struct.
- Все перечисления, объявленные с помощью enum.

Значимый тип данных

Когда значение одной переменной значимого типа присваивается другой, происходит полное побитовое копирование значения.

```
void ValueTypeSemantics()
{
    int a = 42;  // На стеке выделяется 4 байта, в них записывается значение 42.
    int b = a;   // На стеке выделяется ЕЩЕ 4 байта. Содержимое 'a' (42) копируется в 'b'.
    b = 100;     // Значение в ячейке 'b' изменяется на 100. Ячейка 'a' НЕ затронута.
    // По завершении этого кода, 'a' по-прежнему равно 42.
}
```

Ссылочные типы данных

Ссылочный тип - такой тип данных, переменная которого хранит не сам объект, а ссылку (адрес) на место в управляемой куче, где располагается этот объект.

Ключевые представители:

- Все классы, объявленные с помощью ключевого слова class.
- Тип string (со своими особенностями, о которых позже).
- Все массивы (например, int[], string[]).
- Делегаты, интерфейсы и тип object.

Ссылочные типы данных

Локация в памяти (двухкомпонентная модель):

- Сам объект (его поля и служебные данные) всегда создается в куче при помощи оператора new.
- Переменная, которая ссылается на этот объект, хранит лишь его адрес. Сама эта переменная-ссылка, если она является локальной, располагается на стеке.

При присваивании одной переменной ссылочного типа другой, **копируется только значение ссылки (адреса)**, а не сам объект в куче.

Ссылочные типы данных

```
void ReferenceTypeSemantics()
{
    int[] arr1 = new int[2]; // 'new': В КУЧЕ выделяется память под массив.
    // 'arr1': На СТЕКЕ создается переменная, хранящая адрес этого массива.
    arr1[0] = 42;

    int[] arr2 = arr1;       // На СТЕКЕ создается переменная 'arr2'.
    // В 'arr2' КОПИРУЕТСЯ ЗНАЧЕНИЕ 'arr1' - то есть, АДРЕС массива в куче.
    // Сам массив НЕ копируется.

    arr2[0] = 100;           // Мы обращаемся по адресу в 'arr2', находим объект в куче и меняем его.
    // Поскольку 'arr1' хранит тот же адрес, он указывает на тот же, измененный объект.
    // По завершении этого кода, arr1[0] будет равно 100.
}
```

null

Для ссылочных типов существует специальное значение — null. null — это ссылка, которая не указывает ни на один объект. Это нулевой указатель.

int a = 0; — a хранит валидное числовое значение «ноль».

int[] arr = null; — переменная arr существует на стеке, но она не содержит адреса какого-либо объекта в куче. Она «пуста»

Попытка обратиться к членам объекта через null-ссылку (например, arr.Length) приведет к одной из самых распространенных ошибок времени выполнения — System.NullReferenceException. Это означает, что среда исполнения попыталась перейти по адресу, хранящемуся в переменной, но обнаружила, что этот адрес "нулевой" и никакого объекта там нет.

Сводная таблица фундаментальных различий

Характеристика	**Значимые типы (Value Types)**	**Ссылочные типы (Reference Types)**
Что хранится в переменной	Непосредственно само значение.	Адрес (ссылка) на объект в куче.
Основное местоположение	**Стек** (для локальных переменных).	**Куча** (для данных объекта).
Присваивание (=)	**Копирование значения**. Создаются две независимые копии.	**Копирование ссылки**. Обе переменные указывают на один объект.
Сравнение (==)	Сравнивает **значения**.	Сравнивает **ссылки** (адреса). `true`, если указывают на 1 объект.
Значение по умолчанию	`0`, `false` и т.п. (не может быть `null`).	`null` (ссылка в никуда).
Управление памятью	Автоматически, при выходе из области видимости (со стека).	Управляется **Сборщиком Мусора (GC)**.
Примеры	`int`, `bool`, `struct`, `enum`	`string`, `Array`, `class`, `object`

Массивы, Строки и Механизмы вызова функций

```
void ArrayMemoryAllocation()
{
    // Декларация массива из 100 целых чисел
    int[] integerArray = new int[100];
    // На стеке будет выделено 8 байт т.к. x64
    // (4 байта для x86 архитектуры, 8 для x64)
    // Эти 8 байт - ссылка на область в куче
    // Вот в куче будет выделено 400 байт + 12-24 байта
    // где хранится длина массива, информация о типе и т.д.
}
```

Массивы, Строки и Механизмы вызова функций

```
Ссылок: 1
static void ModifyArray(int[] arr)
{
    // 'arr' на стеке метода ModifyArray получает КОПИЮ АДРЕСА
    // и указывает на тот же объект в куче, что и 'integerArray'
    arr[0] = -1;
}

Ссылок: 0
static void Main(string[] args)
{

    int[] integerArray = new int[100];
    integerArray[0] = 42;
    ModifyArray(integerArray);
    // Здесь integerArray[0] будет равно -1,
    // так как оба 'arr' и 'integerArray'
    // оперировали одним и тем же блоком памяти в куче.

}
```

Массивы, Строки и Механизмы вызова функций

Тип string в C# является ссылочным типом, однако он обладает особым свойством — неизменяемости (immutability).

Это свойство делает его поведение в некоторых сценариях похожим на поведение значимых типов, что требует отдельного рассмотрения.

Как и любой другой ссылочный тип, переменная типа string на стеке хранит адрес объекта в куче. Сам объект в куче содержит последовательность символов (char) и служебную информацию.

Массивы, Строки и Механизмы вызова функций

```
string greeting = "Hello";
greeting = greeting + ", World!";
```

- Выделяет в куче новый блок памяти, достаточный для хранения строки "Hello, World!".
- Создает в этом новом месте новый строковый объект.
- Оператор присваивания (=) обновляет значение переменной greeting на стеке, записывая в нее адрес нового объекта.

Массивы, Строки и Механизмы вызова функций

В C# по умолчанию используется передача параметров по значению (pass-by-value). Однако интерпретация этого правила критически зависит от того, является ли передаваемый тип значимым или ссылочным.

"Передать по значению" означает, что для параметра метода в его стековом фрейме создается новая ячейка памяти, и в нее копируется значение переменной-аргумента.

Массивы, Строки и Механизмы вызова функций

Передача значимого типа (например, int):

- Что копируется: Само значение (например, число 42).
- Результат: Метод оперирует полной, независимой копией исходных данных. Любые изменения этой копии внутри метода никак не влияют на оригинальную переменную в вызывающем коде.

Массивы, Строки и Механизмы вызова функций

Передача ссылочного типа (например, int[]):

- Что копируется: Значение переменной-ссылки, то есть адрес объекта в куче.
- Результат: Внутри метода создается новая, локальная ссылка, но она указывает на тот же самый объект в куче, что и исходная ссылка. Метод не может изменить исходную ссылку (т.е. заставить ее указывать на другой массив), но он может через свою копию ссылки получить доступ к общему объекту в куче и изменить его состояние (например, поменять значения элементов массива).

Массивы, Строки и Механизмы вызова функций

Визуальная аналогия:

- Значимый тип: Вы даете коллеге ксерокопию документа. Он может рисовать на своей копии что угодно, ваш оригинал останется нетронутым.
- Ссылочный тип: Вы даете коллеге дубликат ключа от вашего офиса. Теперь у вас два физически разных ключа, но оба открывают одну и ту же дверь. Если коллега войдет в офис и переставит мебель, вы, войдя позже со своим ключом, увидите эти изменения.

Спасибо за внимание

- Спасибо за внимание