**Создание буферов фиксированного размера**

Ключевое слово fixed находит еще одно применение при создании одномерных

массивов фиксированного размера. В документации на C# такие массивы называются

*буферами фиксированного размера.* Такие буферы всегда являются членами структуры.

Они предназначены для создания структуры, в которой содержатся элементы массива,

образующие буфер. Когда элемент массива включается в состав структуры, в ней, как

правило, хранится лишь ссылка на этот массив. Используя буфер фиксированного размера,

в структуре можно разместить весь массив. В итоге получается структура, пригодная

в тех случаях, когда важен ее размер, как, например, в многоязыковом программировании,

при согласовании данных, созданных вне программы на С#, или же когда

требуется неуправляемая структура, содержащая массив. Но буферы фиксированного

размера можно использовать только в небезопасном коде.

Для создания буфера фиксированного размера служит следующая общая форма:

fixed тип имя\_буфера[размер];

где тип обозначает тип данных массива; *имя\_буфера* — конкретное имя буфера фиксированного

размера; *размер* — количество элементов, образующих буфер. Буферы

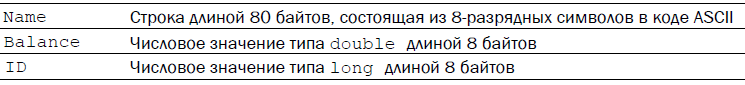
фиксированного размера могут указываться только в структуре.

Для того чтобы стала очевиднее польза от буферов фиксированного размера, рассмотрим

ситуацию, в которой программе ведения счетов, написанной на C++, требуется

передать информацию о банковском счете. Допустим также, что учетная запись

каждого счета организована так, как показано ниже.



В программе на C++ каждая структура содержит массив Name, тогда как в программе

на C# в такой структуре хранится лишь ссылка на массив. Поэтому для правильного

представления данных из этой структуры в C# требуется буфер фиксированного размера,

как показано ниже.

// Использовать буфер фиксированного размера.

unsafe struct FixedBankRecord

{

public fixed byte Name[80]; // создать буфер фиксированного размера

public double Balance;

public long ID;

}

Когда буфер фиксированного размера используется вместо массива Name, каждый

экземпляр структуры FixedBankRecord будет содержать все 80 байтов массива Name.

Именно таким образом структура и организована в программе на C++. Следовательно,

общий размер структуры FixedBankRecord окажется равным 96, т.е. сумме ее членов.

Ниже приведена программа, демонстрирующая этот факт.

(***glava20\_9***)

//bofer fized size

unsafe struct FixedBankRecord

{

public fixed byte Name[80];

public double Balance;

public long ID;

}

class FixedSizeBuffer

{

unsafe static void Main()

{

Console.WriteLine("Size of FixedBankRecord: " + sizeof(FixedBankRecord));

}

}

Размер структуры FixedBankRecord оказывается в точности равным сумме ее членов,

но так бывает далеко не всегда со структурами, содержащими буферы фиксированного

размера. Ради повышения эффективности кода общая длина структуры может

быть увеличена для выравнивания по четной границе, например по границе слова.

Поэтому общая длина структуры может оказаться на несколько байтов больше, чем

сумма ее членов, даже если в ней содержатся буферы фиксированного размера. Как

правило, аналогичное выравнивание длины структуры происходит и в C++. Следует,

однако, иметь в виду возможные отличия в этом отношении.

И наконец, обратите внимание на то, как в данной программе создается буфер фиксированного

размера вместо массива Name.

public fixed byte Name[80];

Как видите, размер массива указывается после его имени. Такое обозначение обычно

принято в C++ и отличается в объявлениях массивов в С#. В данном операторе распределяется

по 80 байтов памяти в пределах каждого объекта типа FixedBankRecord.

**Обнуляемые типы**

Обнуляемый тип — это особый вариант типа значения, представленный структурой.

Помимо значений, определяемых базовым типом, обнуляемый тип позволяет

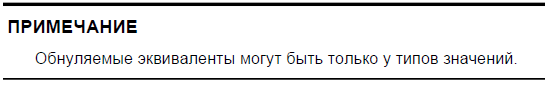
хранить пустые значения (null). Следовательно, обнуляемый тип имеет такой же диапазон

представления чисел и характеристики, как и его базовый тип. Он предоставляет

дополнительную возможность обозначить значение, указывающее на то, что переменная

данного типа не инициализирована. Обнуляемые типы являются объектами типа

System.Nullаble<Т>, где Т — тип значения, которое не должно быть обнуляемым.



Обнуляемый тип может быть указан двумя способами. Во-первых, объекты типа

Nullable<T>, определенного в пространстве имен System, могут быть объявлены явным

образом. Так, в приведенном ниже примере создаются обнуляемые переменные

типа int и bool.

System.Nullable<int> count;

System.Nullable<bool> done;

И во-вторых, обнуляемый тип объявляется более кратким и поэтому чаще используемым

способом с указанием знака ? после имени базового типа. В приведенном

ниже примере демонстрируется более распространенный способ объявления обнуляемых

переменных типа int и bool.

int? count;

bool? done;

Когда в коде применяются обнуляемые типы, создаваемый обнуляемый объект

обычно выглядит следующим образом.

int? count = null;

В данной строке кода переменная count явно инициализируется пустым значением

(null). Это вполне соответствует принятому правилу: прежде чем использовать

переменную, ей нужно присвоить значение. В данном случае присваиваемое значение

означает, что переменная не определена.

Во-первых, можно проверить переменную на пустое значение.

Так, если переменная count объявлена так, как показано выше, то в следующей

строке определяется, имеет ли эта переменная конкретное значение.

if (count != null) // переменная имеет значение

Если переменная count не является пустой, то она содержит конкретное значение.

И во-вторых, можно воспользоваться доступным только для чтения свойством

HasValue типа Nullable<T>, чтобы определить, содержит ли переменная обнуляемого

типа конкретное значение. Это свойство показано ниже.

bool HasValue

Свойство HasValue возвращает логическое значение true, если экземпляр объекта,

для которого оно вызывается, содержит конкретное значение, а иначе оно возвращает

логическое значение false. Ниже приведен пример, в котором конкретное значение

обнуляемого объекта count определяется вторым способом с помощью свойства

HasValue.

if(count.HasValue) // переменная имеет значение

Если обнуляемый объект содержит конкретное значение, то получить это значение

можно с помощью доступного только для чтения свойства Value типа Nullable<T>.

Т Value

Свойство Value возвращает экземпляр обнуляемого объекта, для которого

оно вызывается. Если же попытаться получить с помощью этого свойства значение

пустой переменной, то в итоге будет сгенерировано исключение System.

InvalidOperationException. Кроме того, значение экземпляра обнуляемого объекта

можно получить путем приведения к его базовому типу.

В следующей программе демонстрируется основной механизм обращения с обнуляемым

типом.

(***glava20\_10***).

class NullableDemo

{

static void Main()

{

int? count = null;

if (count.HasValue)

Console.WriteLine("Count has value: " + count.Value);

else

Console.WriteLine("Count has not value.");

count = 100;

if (count.HasValue)

Console.WriteLine("Count has value: " + count.Value);

else

Console.WriteLine("Count has not value.");

}

}

**Применение обнуляемых объектов в выражениях.**

Обнуляемый объект может использоваться в тех выражениях, которые являются

действительными для его базового типа. Более того, обнуляемые объекты могут сочетаться

с необнуляемыми объектами в одном выражении. И это вполне допустимо

благодаря предопределенному преобразованию базового типа в обнуляемый. Когда

обнуляемые и необнуляемые типы сочетаются в одной операции, ее результатом становится

значение обнуляемого типа.

В приведенной ниже программе демонстрируется применение обнуляемых типов

в выражениях.

(***glava20\_11***)

class NullableDemo

{

static void Main()

{

int? count = null;

int? result = null;

int incr = 10;

//result has null value

//variable count is empty

result = count + incr;

if (result.HasValue)

Console.WriteLine("Result has value: " + result.Value);

else

Console.WriteLine("Result has no value.");

//now count has its value

//thas why result will be with value

count = 100;

result = count + incr;

if (result.HasValue)

Console.WriteLine("Result has value: " + result.Value);

else

Console.WriteLine("Result has no value.");

}

}

**Оператор ??**

Попытка преобразовать обнуляемый объект в его базовый тип путем приведения

типов обычно приводит к генерированию исключения System.

InvalidOperationException, если обнуляемый объект содержит пустое значение.

Это может произойти, например, в том случае, если значение обнуляемого объекта

присваивается переменной его базового типа с помощью приведения типов. Появления

данного исключения можно избежать, если воспользоваться оператором ??, называемым

*нулеобъединяющим оператором.* Этот оператор позволяет указать значение,

которое будет использоваться по умолчанию, если обнуляемый объект содержит пустое

значение. Он также исключает потребность в приведении типов.

Ниже приведена общая форма оператора ??.

**обнуляемый\_объект ?? значение\_по\_умолчанию**

Если *обнуляемый\_объект* содержит конкретное значение, то результатом операции

?? будет именно это значение. В противном случае результатом операции ?? окажется

*значение\_по\_умолчанию.*

Например, в приведенном ниже фрагменте кода переменная balance содержит

пустое значение. Вследствие этого переменной currentBalance присваивается значение

0.0, используемое по умолчанию, и тем самым устраняется причина для генерирования

исключения.

double? balance = null;

double currentBalance;

currentBalance = balance ?? 0.0;

В следующем фрагменте кода переменной balance присваивается значение

123.75.

double? balance = 123.75;

double currentBalance;

currentBalance = balance ?? 0.0;

Теперь переменная currentBalance содержит значение 123.75 переменной

balance.

И еще одно замечание: выражение в правой части оператора ?? вычисляется только

в том случае, если выражение в левой его части не содержит значение. Этот факт

демонстрируется в приведенной ниже программе.

(это что то типа условного оператора, если значение есть, то переменная будет равна значению до вопросительных знаков, иначе используется значение после вопросительных знаков.)

(***glava20\_12***)

class NullableDemo2

{

//return null rest

static double GetZeroBall()

{

Console.WriteLine("In method GetZeroBall.");

return 0.0;

}

static void Main()

{

double? balance = 123.75;

double currentBalance;

//method GetZeroBall() wount call,

//because balance has value

currentBalance = balance ?? GetZeroBall();

Console.WriteLine(currentBalance);

}

}

В этой программе метод GetZeroBal() не вызывается, поскольку переменная

balance содержит конкретное значение. Как пояснялось выше, если выражение в левой

части оператора ?? содержит конкретное значение, то выражение в правой его

части не вычисляется.

**Обнуляемые объекты, операторы отношения и логические операторы**

Обнуляемые объекты могут использоваться в выражениях отношения таким же

образом, как и соответствующие объекты необнуляемого типа. Но они должны подчиняться

следующему дополнительному правилу: когда два обнуляемых объекта сравниваются

в операциях сравнения <, >, <= или >=, то их результат ***будет ложным***, если

***любой*** из обнуляемых объектов оказывается пустым, т.е. содержит значение null.

В качестве примера рассмотрим следующий фрагмент кода.

byte? lower = 16;

byte? upper = null;

// Здесь переменная lower определена, а переменная upper не определена.

if(lower<upper) // ложно

В данном случае проверка того, что значение одной переменой меньше значения

другой, дает ложный результат.

Следовательно, если один или оба сравниваемых обнуляемых объекта оказываются

пустыми, то результат их сравнения всегда будет ложным. Это фактически означает,

что пустое значение (null) не участвует в отношении порядка.

Тем не менее с помощью операторов == и != можно проверить, содержит ли обнуляемый

объект пустое значение. Например, следующая проверка вполне допустима и

дает истинный результат.

if(upper == null) // ...

Если в логическом выражении участвуют два объекта типа bool?, то его результат

может иметь одно из трех следующих значений: true (истинное), false (ложное) или

null (неопределенное). Ниже приведены результаты применения логических операторов

& и | к объектам типа bool?.

