УДК 681.3.062

DOI: 10.21779/2542-0321-2019-34-3-73-78

А.М. Магомедов¹, А.З. Якубов¹, С.А. Лавренченко²

К вопросу о представлении в памяти числовых переменных разных типов

¹ Дагестанский государственный университет; Россия, 367001, г. Махачкала, ул. М. Гаджиева, 43 а; tagomedtagir1@yandex.ru, yakubovaz@mail.ru;

² Российский государственный университет туризма и сервиса; Россия, 141221, Московская обл., Пушкинский р-н, д/п Черкизово, ул. Главная, 99; lawrencenko@hotmail.com

В программах, написанных на языках высокого уровня, при вычислении простых выражений нередко возникают коллизии, причиной которых является недостаточное внимание к деталям представления в оперативной памяти (ОП) переменных, объявленных в программе. Это относится к переменным различных типов, в особенности – к переменным различных вещественных типов. Последние имеют одну и ту же структуру представления, включающую три поля: знак (*s*), порядок (*E*) и мантиссу (*M*). При этом алгоритм вычисления значений переменной определяется не только типом переменной, но и текущей величиной в поле *E*.

Вопросы вывода двоичного представления в ОП переменных целочисленных типов широко освещены в литературе, а вопросам вывода представления переменных вещественного типа уделено гораздо меньше внимания.

В статье изложены подходы к выводу двоичного представления в ОП переменных вещественного типа средствами языков Delphi и C#. В каждом из этих языков (дополнительно к средствам, присущим и другим алгоритмическим языкам) предусмотрены средства, специфические только для данного языка. На их основе в статье объяснены некоторые характерные коллизии, возникающие в программах при вычислении арифметических выражений; при этом существенную роль играет не только принципиальная возможность точного представления того или иного вещественного числа в виде конечной двоичной дроби, но и достижимость такого представления для конкретного вещественного типа. Все объяснения сопровождаются подробными примерами.

В качестве прикладного применения изложенных результатов построены алгоритмы порождения всех подмножеств заданного начального множества натуральных чисел.

Ключевые слова: *память, программа, переменная, тип, алгоритм.*

Введение

Процесс преподавания компьютерных наук нередко инспирирует исследование проблем, относящихся к области на стыке дискретной математики и программирования [1–3]. Настоящая статья посвящена одной из таких проблем.

В разделе 2 приводится структура представления вещественных переменных, знакомство с которой необходимо для понимания причин возникновения в программах различного рода коллизий.

Пример 1. Почему следующая программа на языке Delphi выводит значение True?

```
var  
  r: Single=0.1; t: Single=260.0;  
begin
```

```
writeln ((r<>0.1) and (t=260.0));
readln;
end.
```

Замечание. Здесь и в дальнейшем из соображений компактности текста принято размещение листинга программы без соблюдения рекомендации «одна строка – один оператор».

Пример 2. Как представляется в оперативной памяти (ОП) переменная a , объявленная в программе на языке C# следующим образом: `double a=260.0`?

В разделе 3 указаны способы организации просмотра битовых представлений вещественных переменных разных типов. В качестве приложения в разделе 4 рассмотрены алгоритмы порождения булеана.

Представление вещественных переменных в ОП

Адресация байтов в ОП начинается, как известно, с нуля: 0, 1, 2, ... Ответ на вопрос о многобайтовом представлении переменной вещественного типа в ОП (для определенности будем подразумевать типы Single, Double/Real и Extended языка Delphi 7.0) должен включать разъяснения по общему для всех вещественных типов формату (рис. 1), а также информацию о порядке байтов и вычислении значения числа по заданным s , E и M .

Рис. 1. Поле s содержит один бит: если число отрицательно, то 1, иначе – 0; а поля E и M соответственно 8 и 23 (Single), 11 и 52 (Double, Real), 15 и 64 (Extended) битов

Направление от мантиссы M к знаку s – от младших байтов к старшим («интеловский» порядок); отметим, что для ряда процессоров порядок байтов выбирается программно во время инициализации операционной системы, но может быть выбран и аппаратно переключками на материнской плате [4].

В зависимости от типа числа меняется не только число битов, отводимых под E и M , но и алгоритм вычисления значения числа. В напоминании известного алгоритма вычисления значений ограничимся случаем, когда поле E содержит как биты, равные 0, так и биты, равные 1:

$$(-1)^s 2^{E-127} \cdot 1.M \text{ (для Single),}$$

$$(-1)^s 2^{E-1023} \cdot 1.M \text{ (для Double и Real),}$$

$(-1)^s 2^{E-16383} \cdot M_1.M'$, где M_1 – первый разряд мантиссы, M' – запись остальной части мантиссы (для Extended); для единообразной записи второго сомножителя иногда используют 2^{E-e} , где $e = 2^{|E|-1} - 1$.

Замечание. В случае типа Single при $E = 0$ используется формула

$$(-1)^s \cdot 2^{126} \cdot 0.M,$$

при $E = 255$, $M = 0$ значение принимается равным бесконечности, а ситуация $E = 255$, $M \neq 0$ рассматривается как ошибка.

Стремление сформулировать алгоритм вычисления значения числа по заданным s , E и M , обеспечивающий в общем случае однозначность, максимальную точность и производительность, приводит к неожиданным следствиям: арифметические операции не обладают свойствами коммутативности и ассоциативности, разность двух различных чисел может равняться нулю, а нуль может иметь знак и т. д. [5].

Пример 3. Переменная q типа Single представлена в памяти следующим образом:

01000011100000100000000000000000. Найти значение q .

Решение. $0100001110000010...0 = (-1)^0 \cdot 2^{10000111_2 - 127} \cdot 1.0000010... = 2^{135-127} \cdot \left(1 + \frac{1}{64}\right) = 2^8 \cdot \left(1 + \frac{1}{64}\right) = 256 + 4 = 260.0$.

Побитовый вывод значения вещественной переменной

Вывод двоичного представления в ОП значения q -байтной целочисленной переменной k не вызывает затруднений:

for i:=8*q-1 downto 0 do write (($k \text{ shr } i$) and 1). (1)

Лемма. Цикл (1), где операция $k \text{ shr } i$ означает сдвиг двоичного представления переменной k вправо на i разрядов, а через and обозначена операция поразрядного логического умножения, осуществляет вывод двоичного представления значения q -байтной переменной k (в порядке «от старших битов к младшим» справа налево).

Доказательство. В результате выполнения операции $k \text{ shr } i$ в младший разряд поля ОП, занимаемого k , будет записан разряд с позицией i в двоичном представлении переменной k (обозначим этот разряд через k_i). Тогда в результате поразрядного умножения на $00...001$ получим $00...00k_i$; следовательно, будет выведено 0 или 1, соответствующее k_i . Доказательство закончено.

В случае многобайтовой целочисленной переменной достаточно применить этот вывод к каждому байту с учетом порядка «от младших к старшим». Но в случае вещественной переменной операции сдвига и логического умножения неприменимы. Одним из подходов к решению в этом случае является предварительное преобразование к типу равновеликого (с тем же количеством байтов) байтового массива.

Пример 4. Вывести представление в ОП заданного в программе Delphi значения переменной L типа Single.

Решение:

```
type arrayb=array [0..3] of Byte;
var i, j: Integer; L: Single=260.0; b: arrayb;
begin
  b:=arrayb(L);
  for i:=sizeof(L)-1 downto 0 do for j:=7 downto 0 do write ( (b[i] shr j) and 1);
  readln;
end.
```

В результате будет выведен набор битов (см. пример 3):

01000011100000100000000000000000.

Имеются и иные способы сведения вывода битового представления вещественной переменной L к выводу равновеликой многобайтовой целочисленной переменной b . Например, в программе на языке Delphi размещение L и b можно совместить в ОП с помощью директивы компилятора Absolute.

Приведем соответствующее решение для примера 4:

```
var L: Single=260.0; b: LongInt Absolute L; i: Byte;
begin
  for i:=31 downto 0 do write ( (b shr i) and 1);
  readln;
end.
```

Во всех приведенных выше примерах для вывода двоичных разрядов использовались операции сдвига и логического умножения. Разумеется, в разных языках кроме них (или других привычных операций, например операций целочисленного деления) существуют и другие, более «специализированные» средства. Так, в программе C# для решения аналогичной задачи мы рекомендуем использовать коллекцию `BitArray` и метод `GetBytes()` статического класса `BitConverter`, который может принимать любой числовой тип: `short`, `int`, `long`, их беззнаковые аналоги, типы с плавающей запятой (`float` и `double`), а также логический тип [3].

```
using System;
using System.Collections;
class Program
{
    static void Main()
    {
        BitArray b = new BitArray(BitConverter.GetBytes(260.0F));
        for (int i = b.Length - 1; i >= 0; i--) Console.Write(b[i] ? 1 : 0);
    }
}
```

В результате, как и ранее, получим 01000011100000100000000000000000.

Замечание. Если в приведенном решении заменить оператор цикла `for` на `foreach`: `foreach (bool bit in b) Console. Write (bit ? 1 : 0)`, то решение остается верным, однако направление вывода изменится на обратное: 0000000000000000000100000111000010.

Вернемся к примеру 1. Процессоры Intel оперируют с типом `Extended`. Поэтому число 0.1 перед присвоением переменной r сначала преобразуется в 80-битовое число типа `Extended` (обозначим его $0.1'$); легко показать, что точное представление числа 0.1 в виде конечной двоичной дроби невозможно.

При записи в «прокрустово ложе» переменной r типа `Single` (32 бита) младшие биты мантиссы числа $0.1'$ будут потеряны, поэтому в r получим некоторое $0.1''$, отличное от $0.1'$.

Перед выполнением сравнения $r < 0.1$ и левая, и правая части снова преобразуются в тип `Extended`. При этом значение $0.1''$ типа `Single` переменной r при преобразовании в тип `Extended` не изменится (добавятся лишь незначимые нулевые биты), но правая часть сравнения преобразуется в $0.1'$; таким образом, результатом сравнения $r < 0.1$ явится `True`.

Остается добавить, что результатом сравнения $t = 260.0$ будет `True`: как видно из вывода программы в примере 4, число 260.0 не только допускает точное представление в виде конечной двоичной дроби, но допускает точное представление в формате `Single` (и `Extended`); отсюда следует, что $260.0'$ равно $260.0''$ (обозначения аналогичны $0.1'$ и $0.1''$).

Алгоритмы вывода всех двоичных последовательностей заданной длины

Рассмотрим два способа порождения *булеана* – всех подмножеств множества $N = \{0, 1, \dots, n-1\}$, где n – заданное натуральное число. Поскольку каждое подмножество s множества N может быть задано двоичной последовательностью длины n , где разряд в позиции i равен 1, только тогда, когда $i \in s$, то нам предстоит рассмотреть алгоритмы порождения всех двоичных последовательностей длины n .

Алгоритм 4.1

Для $k=0, 1, \dots, 2^n-1$ выполнить:

begin

для $i=n-1, \dots, 1, 0$ выполнить вывод числа $((k \text{ shr } i) \text{ and } 1)$

end.

Алгоритм 4.2

Используется битовый массив $b[n], b[n-1], \dots, b[0]$, все элементы которого вначале равны нулю. Для записи порожденной последовательности служат $b[n-1], \dots, b[0]$.

Пока $b[n] = 0$, выполнить:

begin

1) вывод последовательности $b[n-1], \dots, b[0]$;

2) первый справа налево элемент $b[i]$, равный 0, заменить на 1, а все элементы правее него обнулить: $b[0]:=0; \dots, b[i-1]:=0$;

end.

Теорема. Как алгоритм 4.1, так и алгоритм 4.2 – алгоритмы порождения всех двоичных последовательностей длины n .

Доказательство. Для алгоритма 4.1 утверждение теоремы немедленно следует из леммы раздела 3. Для алгоритма 4.2 достаточно заметить, что каждая его итерация заключается в сложении «столбиком» единицы и предыдущего n -разрядного двоичного представления $b[n-1] \dots b[0]$ числа. Доказательство закончено.

Заключение

Хотя сложность алгоритма 4.1 превышает сложность алгоритма 4.2, трудно указать адекватные реализации алгоритмов на каком-либо языке программирования, чтобы программа, соответствующая алгоритму 4.2, выигрывала в быстродействии. Эта задача остается открытой.

Наконец, отметим, что все не приведенные здесь сведения по языкам Delphi и C#, необходимые для ознакомления с текстом статьи, можно найти в [6–8] и [9–11] соответственно.

Работа выполнена при поддержке Отдела математики и информатики ДНЦ РАН.

Литература

1. Магомедов А.М. Цепочечные структуры в задачах о расписаниях // Прикладная дискретная математика. – 2016. – № 3 (33). – С. 67–77.
2. Шихиев Ш.Б., Мирзабеков Я.М. Дискретный анализ в синтаксическом анализе // Информатика и её применения. – 2018. – Т. 12, вып. 2. – С. 98–104.
3. Шихиев Ш.Б., Мирзабеков Я.М. Формальная грамматика русского языка в примерах // Прикладная дискретная математика. – 2018. – № 40. – С. 114–126.
4. Порядок байтов [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki>.
5. Что нужно знать про арифметику с плавающей запятой [Электронный ресурс]. – URL: <https://habr.com/ru/post/112953>.
6. Архангельский А.Я. Программирование в Delphi: учебник по классическим версиям Delphi. – М.: Бином, 2017. – 583 с.

7. Бобровский С. Delphi 7: учебный курс. – СПб.: Питер, 2018. – 736 с.
8. Григорьев А.Б. О чем не пишут в книгах по Delphi. – БХВ-Петербург, 2016. – 576 с.
9. Вагнер Б. С#. Эффективное программирование. – М.: ЛОРИ, 2017. – 320 с.
10. Кариев Ч.А. Разработка Windows-приложений на основе Visual C#. – М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2015. – 501 с.
11. Фленов М.Е. Библия C#. – М.: БХВ-Петербург, 2015. – 532 с.

Поступила в редакцию 26 июня 2019 г.

UDC 681.3.062

DOI: 10.21779/2542-0321-2019-34-3-73–78

To the question of representing numeric variables of different types in memory

A.M. Magomedov¹, A.Z. Yakubov¹, S.A. Lawrencenko²

¹ Dagestan State University; Russia, 367001, Makhachkala, M. Gadzhiev st., 43a; magomedtagirl@yandex.ru, yakubovaz@mail.ru;

² Russian State University of Tourism and Service; Russia, Moscow region, 141221, Pushkino district, Cherkizovo, Glavnaya st., 99; lawrencenko@hotmail.com

When calculating simple expressions in programs written in high-level languages collisions often occur caused by insufficient attention to the details of the representation in RAM of variables declared in the program. This applies to variables of different types, in particular to variables of different material types. The latter have the same representation structure including three fields: sign (*s*), order (*E*) and mantissa (*M*). The algorithm for calculating the values of the variable is determined not only by the type of variable, but also by the current value in the *E* field.

While issues of deriving the binary representation of RAM variables in integer types are widely covered in the literature, much less attention is paid to the issues of deriving the representation of variables of a real type.

The article presents approaches to the derivation of the binary representation in RAM of variables of a real type by means of the Delphi and C# languages. Each of these languages (in addition to the tools inherent in other algorithmic languages) provides tools specific to that language only. On their basis the article explains some typical conflicts that arise in programs when evaluating arithmetic expressions; the essential role is played not only by the fundamental possibility of the accurate representation of a real number in the form of a finite binary fraction, but also the attainability of such representation specifically for the real type. All explanations are accompanied by detailed examples.

As an application of the presented results, algorithms for generating all subsets of a given initial set of natural numbers are constructed.

Keywords: memory, program, variable, type, algorithm.

Received 26 June, 2019