Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №1 по теме: «Встроенные структуры данных»

Выполнил:  
студент группы ПВ-21  
Адаменко И. И.

Проверил:  
профессор  
Синюк В. Г.

Белгород  
2013

Цель работы: изучение базовых типов данных языка Pascal как структур данных.

## Задания

1. Для заданных типов данных определить:
   1. Абстрактный уровень представления СД:
      1. Характер организованности и изменчивости.
      2. Набор допустимых операций.
   2. Физический уровень представления СД:
      1. Схему хранения.
      2. Объем памяти, занимаемый экземпляром СД.
      3. Формат внутреннего представления СД и способ его интерпретации.
      4. Характеристику допустимых значений.
      5. Тип доступа к элементам.
   3. Логический уровень представления СД. Способ описания СД и ее экземпляра на языке программирования.
2. Для заданных типов данных определить набор значений, необходимый для изучения физического уровня представления СД.
3. Преобразовать значения в двоичный код.
4. Преобразовать двоичный код в значение.
5. Разработать и отладить программу, выдающую двоичное представление значений заданных СД. В программе использовать процедуры *PrintByte* и *PrintVar.*
   1. Спецификация процедуры *PrintByte:*
      1. Заголовок: *procedure PrintByte(a:byte);*
      2. Назначение: выводит на экран монитора двоичное представление переменной *a* типа *byte.*
      3. Входные параметры: *a.*
      4. Выходные параметры: нет.
      5. Рекомендации: использовать побитовые операции сдвига и логического умножения.
   2. Спецификация процедуры *PrintVar:*
      1. Заголовок: *procedure PrintVar(var a; size: word);*
      2. Назначение: выводит на экран двоичное представление переменной *a* произвольного типа размером *size* байт.
      3. Входные параметры: *a* — переменная произвольного типа, значение которой выводится на экран в двоичном представлении (нетипизированный параметр), *size* — объем памяти (в байтах), занимаемый переменной *a.*
      4. Выходные параметры: нет.
      5. Рекомендации: нетипизированную переменную *a* привести к типу «массив байт», значение каждого элемента которого выводить на экран в двоичном представлении процедурой *PrintByte.*
6. Обработать программой значения, полученные в результате выполнения п. 3 задания. Сделать выводы.
7. Разработать и отладить программу, определяющую значение переменной по ее двоичному представлению по следующему алгоритму:
   1. Ввести двоичный код в переменную *S* строкового типа.
   2. Преобразовать *S* в вектор *B* типа «массив байт».
   3. Привести *B* к заданному типу. Вывести значение.
   4. Конец.
8. Обработать программой значения, полученные в результате выполнения п. 4 задания. Сделать выводы.

### Задание для I варианта

|  |  |  |
| --- | --- | --- |
| Тип 1 | Тип 2 | Тип 3 |
| integer | real | (red, yellow, green) |

## Характеристика типов данных как СД

### Integer

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип упорядоченный и статический.
   2. К типу применимы такие операции, как: арифметические, битовые, операция доступа и инициализации и другие.
2. Физический уровень представления СД:
   1. Структура данных хранится последовательно.
   2. Каждый экземпляр СД занимает, в зависимости от архитектуры процессора, 2 или 4 байта.
   3. Значение переменной этого типа хранится в памяти как последовательность бит, интерпретируемая в число в двоичной системе.
   4. Допустимые значения для данной СД, в зависимости от архитектуры процессора, лежат в следующих диапазонах: или .
   5. Тип доступа к элементам данной СД прямой.
3. На логическом уровне представления СД, то есть, на языке программирования Pascal экземпляр СД описывается следующим образом: var n: integer;

### Real

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип неупорядоченный и статический.
   2. К типу применимы такие операции, как: арифметические, битовые, операция доступа и инициализации и другие.
2. Физический уровень представления СД:
   1. Структура данных хранится последовательно.
   2. Каждый экземпляр СД занимает 6 байт.
   3. Значение переменной этого типа хранится в памяти как 6-байтовое (48-битовое) число со знаком, которое условно делится на три поля: 1-битовое поле знака, 39-битовое поле мантиссы и 8-битовое поле порядка, в котором также задается и знак порядка. Мантисса — это число, которое всегда меньше единицы, у которой первый разряд содержит отличную от нуля цифру. Такое представление числа называется нормализованным.
   4. Допустимые значения для данной СД лежат в следующем диапазоне: .
   5. Тип доступа к элементам данной СД прямой.
3. На логическом уровне представления СД, то есть, на языке программирования Pascal экземпляр СД описывается следующим образом: var n: real;

### (red, yellow, green)

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип упорядоченный и статический.
   2. К типу применима операция доступа, операция инициализации и другие.
2. Физический уровень представления СД:
   1. Структура данных хранится последовательно.
   2. Каждый экземпляр СД занимает 2 байта.
   3. Значение переменной этого типа хранится в памяти как последовательность бит, интерпретируемая в порядковый в двоичной системе, а каждому порядковому номеру соответствует свое значение (по порядку перечисления).
   4. Допустимые значения порядковых номеров для данной СД лежат в следующем диапазоне: .
   5. Тип доступа к элементам данной СД прямой.
3. На логическом уровне представления СД, то есть, на языке программирования Pascal экземпляр СД описывается следующим образом: type colors = (red, yellow, green); var color: colors;

## Набор значений, необходимых для изучения физического уровня представления СД

* Integer: 17449;
* Real: 99.35591;
* (red, yellow, green): yellow.

## Значения, преобразованные в двоичный код

* Integer: 0100 0100 0010 1001;
* Real: 0100 0000 0101 1000 1101 0110 1100 0111 0011 1010 1011 1100 1001 0100 0111 0000;
* (red, yellow, green): 0000 0000 0000 0000 0000 0000 0000 0001.

## Двоичный код, преобразованный в значения

* Integer: 1100 0011 1010 0101;
* Real: 1000 0100 0001 1100 1001 0010 1000 0011 1111 1110 1111 1000 1101 0000 0011 0010;
* (red, yellow, green): 0000 0000 0000 0000 0000 0000 0000 0010.
* Integer: −15451;
* Real: −7.32976037648828E−289;
* (red, yellow, green): green.

## Программа, выдающая двоичное представления значений заданных СД

### Модуль SD

|  |  |
| --- | --- |
| 001 | **UNIT** SD; |
| 002 |  |
| 003 | {-----------------------------------------------------------------} |
| 004 | **INTERFACE** |
| 005 | {-----------------------------------------------------------------} |
| 006 |  |
| 007 | **uses** CRT; |
| 008 |  |
| 009 | **type** |
| 010 | colors = (red, yellow, green); |
| 011 |  |
| 012 | **procedure** PrintByte(a: **byte**); |
| 013 | **procedure** PrintVar(**var** a; size: **word**); |
| 014 | **function** BinToInt(**const** s: **string**): **integer**; |
| 015 | **function** BinToReal(**const** s: **string**): **real**; |
| 016 | **function** BinToColor(**const** s: **string**): **word**; |
| 017 |  |
| 018 | {-----------------------------------------------------------------} |
| 019 | **IMPLEMENTATION** |
| 020 | {-----------------------------------------------------------------} |
| 021 |  |
| 022 | **procedure** PrintByte(a: **byte**); |
| 023 | **var** |
| 024 | i, x: **byte**; |
| 025 |  |
| 026 | **begin** |
| 027 |  |
| 028 | **for** i := 7 **downto** 0 **do** |
| 029 | **begin** |
| 030 | x := (a **shr** i) **and** 1; |
| 031 | **write**(x); |
| 032 | **if** (i **mod** 4 = 0) **then** |
| 033 | **begin** |
| 034 | **case** (i) **of** |
| 035 | 4: textcolor(8); |
| 036 | 0: textcolor(7); |
| 037 | **end**; |
| 038 | **write**(' '); |
| 039 | **end**; |
| 040 | **end**; |
| 041 |  |
| 042 | **end**; |
| 043 |  |
| 044 | **procedure** PrintVar(**var** a; size: **word**); |
| 045 | **var** |
| 046 | i: **byte**; |
| 047 | byteArray: **array** [1..65520] **of** **byte** **absolute** a; |
| 048 |  |
| 049 | **begin** |
| 050 |  |
| 051 | **for** i := size **downto** 1 **do** |
| 052 | PrintByte(byteArray[i]); |
| 053 |  |
| 054 | **end**; |
| 055 |  |
| 056 | **procedure** AddOne(**var** b: **byte**; a: **word**); |
| 057 | **begin** |
| 058 | b := b **or** (1 **shl** (a **mod** 8)); |
| 059 | **end**; |
| 060 |  |
| 061 | **function** BinToInt(**const** s: **string**): **integer**; |
| 062 | **var** |
| 063 | byteArray: **array** [1..2] **of** **byte**; |
| 064 | i, j: **word**; |
| 065 | x: **integer** **absolute** byteArray; |
| 066 |  |
| 067 | **begin** |
| 068 |  |
| 069 | **for** i := 1 **to** 2 **do** |
| 070 | byteArray[i] := 0; |
| 071 |  |
| 072 | j := 0; |
| 073 | **for** i := length(s) **downto** 1 **do** |
| 074 | **begin** |
| 075 | **if** s[i] = '1' **then** |
| 076 | AddOne(byteArray[(j **div** 8) + 1], j); |
| 077 |  |
| 078 | **inc**(j); |
| 079 | **end**; |
| 080 |  |
| 081 | BinToInt := x; |
| 082 |  |
| 083 | **end**; |
| 084 |  |
| 085 | **function** BinToReal(**const** s: **string**): **real**; |
| 086 | **var** |
| 087 | byteArray: **array** [1..8] **of** **byte**; |
| 088 | i, j: **word**; |
| 089 | x: **real** **absolute** byteArray; |
| 090 |  |
| 091 | **begin** |
| 092 |  |
| 093 | **for** i := 1 **to** 8 **do** |
| 094 | byteArray[i] := 0; |
| 095 |  |
| 096 | j := 0; |
| 097 | **for** i := length(s) **downto** 1 **do** |
| 098 | **begin** |
| 099 | **if** s[i] = '1' **then** |
| 100 | AddOne(byteArray[(j **div** 8) + 1], j); |
| 101 |  |
| 102 | **inc**(j); |
| 103 | **end**; |
| 104 |  |
| 105 | BinToReal := x; |
| 106 |  |
| 107 | **end**; |
| 108 |  |
| 109 | **function** BinToColor(**const** s: **string**): **word**; |
| 110 | **var** |
| 111 | byteArray: **array** [1..4] **of** **byte**; |
| 112 | i, j: **word**; |
| 113 | x: **word** **absolute** byteArray; |
| 114 |  |
| 115 | **begin** |
| 116 |  |
| 117 | **for** i := 1 **to** 4 **do** |
| 118 | byteArray[i] := 0; |
| 119 |  |
| 120 | j := 0; |
| 121 | **for** i := length(s) **downto** 1 **do** |
| 122 | **begin** |
| 123 | **if** s[i] = '1' **then** |
| 124 | AddOne(byteArray[(j **div** 8) + 1], j); |
| 125 |  |
| 126 | **inc**(j); |
| 127 | **end**; |
| 128 |  |
| 129 | BinToColor := x; |
| 130 |  |
| 131 | **end**; |
| 132 |  |
| 133 | **begin** |
| 134 |  |
| 135 | **end**. |

### Основная программа

|  |  |
| --- | --- |
| 01 | **program** lab1; |
| 02 |  |
| 03 | **uses** SD; |
| 04 |  |
| 05 | **var** |
| 06 | first: **integer**; |
| 07 | second: **real**; |
| 08 | third: colors; |
| 09 | s: **string**; |
| 10 |  |
| 11 | **begin** |
| 12 |  |
| 13 | **write**(*'Enter the integer-number, please: '*); |
| 14 | **readln**(first); |
| 15 | **writeln**(*'Binary code of it: '*); |
| 16 | PrintVar(first, **sizeof**(first)); |
| 17 | **writeln**; |
| 18 | **writeln**; |
| 19 |  |
| 20 | **write**(*'Enter the real-number, please: '*); |
| 21 | **readln**(second); |
| 22 | **writeln**(*'Binary code of it: '*); |
| 23 | PrintVar(second, **sizeof**(second)); |
| 24 | **writeln**; |
| 25 |  |
| 26 | **write**(*'Enter the color, please: '*); |
| 27 | **readln**(s); |
| 28 | first := 1; |
| 29 | **case** (s) **of** |
| 30 | 'red': third := red; |
| 31 | 'green': third := green; |
| 32 | 'yellow': third := yellow; |
| 33 | **else** |
| 34 | first := 0; |
| 35 | **end**; |
| 36 | **if** (first = 1) **then** |
| 37 | **begin** |
| 38 | **writeln**(*'Binary code of it: '*); |
| 39 | PrintVar(third, **sizeof**(third)); |
| 40 | **writeln**; |
| 41 | **end** |
| 42 | **else** |
| 43 | **writeln**(*'Error! You enter not color!'*); |
| 44 | **writeln**; |
| 45 |  |
| 46 | **writeln**(*'Now, enter the binary code of integer: '*); |
| 47 | **readln**(s); |
| 48 | first := BinToInt(s); |
| 49 | **writeln**(*'Integer: '*, first); |
| 50 | **writeln**; |
| 51 |  |
| 52 | **writeln**(*'Enter the binary code of real: '*); |
| 53 | **readln**(s); |
| 54 | second := BinToReal(s); |
| 55 | **writeln**(*'Real: '*, second); |
| 56 | **writeln**; |
| 57 |  |
| 58 | **writeln**(*'Now, enter the binary code of color: '*); |
| 59 | **readln**(s); |
| 60 | **case** (BinToColor(s)) **of** |
| 61 | 0: **writeln**(*'Color: red'*); |
| 62 | 1: **writeln**(*'Color: yellow'*); |
| 63 | 2: **writeln**(*'Color: green'*); |
| 64 | **else** |
| 65 | **writeln**(*'Error! Incorrect binary code for color!'*); |
| 66 | **end**; |
| 67 |  |
| 68 | **readln**; |
| 69 |  |
| 70 | **end**. |

### Заключение

Я обработал с помощью программы тестовые данные и получил те же самые значения. Следовательно, программа работает верно, сбоев не обнаружено.

Однако, я заметил, что на Free Pascal на 64-х битной системе тип Real представляется не шестью байтами, а восемью. Перечисляемый же тип требует для хранения четыре байта, вместо двух.