L#6

Основы алгоритмизации и программирования. Массивы

Педагогическое образование, 3 семестр

Mayer Svetlana Fyodorovna

Одномерные массивы

Объявление массива

Массив - это набор элементов одного и того же типа, последовательно расположенных в памяти. Каждый элемент имеет свой собственный индекс. Мы будем использовать так называемые динамические массивы. Это означает, что память выделяется динамически - во время выполнения программы

```
begin
  var a: array of integer;
  a := new integer[3];
  a[0] := 5;
  a[1] := 2;
  a[2] := 3;
end.
```

Мы можем объединить определение и выделение памяти

```
var a := new integer[3] (5,2,3);
```

Вывод:

```
begin
  var a: array of integer;
  a := new integer[5];
  a[0] := 1;  a[1] := 3; a[2] := 5; a[2] := 7; a[2] := 9;
```

```
Println(a); // [1,3,5,7,9]
a.Println;  // 1 3 5 7 9
a.Println(';'); // 1;3;5;7;9
Println(a[2]); // 5
```

Массивы как ссылочные типы

Мы говорим, что переменная массива ссылается на память, выделенную новой операцией

```
var a := new integer[3];
```

Чтобы вернуть память, выделенную массивом, мы можем присвоить переменной массива специальное нулевое значение :

```
var a := nil;
```

Проход по элементам массива

Для перебора элементов массива по их индексам мы можем использовать цикл **for**:

```
for var i:=0 to a.Length-1 do
  a[i] += 1;
```

Для доступа только для чтения (без изменения значений) мы можем использовать цикл **foreach** :

```
foreach var x in a do
  Print(x)
```

Универсальный тип (Generic)

- В определении **универсального типа** или функции/процедуры параметр типа является заполнителем для определенного типа, который клиент указывает при создании экземпляра универсального типа.

<T> означает любой тип

```
procedure printArray<T>(a: array of T);
begin
  for var i:=0 to a.Length-1 do
  begin
    print(a[i]);
  end;
  println
end;
```

Многофайловая компоновка

- Чтобы создать многофайловое приложение, сначала вам нужно создать файл модуля (.pas). Такие файлы состоят из следующих строк :

```
unit DynArrs;
```

- **Interface section (блок интерфейса):** определяет интерфейс к функциям, то есть их объявления: (объявления) of the function(s) or procedures:

```
// ====================== interface section
Interface
procedure PrintIntArr(a: array of integer);
function ArraySum(a: array of integer): integer;
```

- **Раздел реализации:** содержит всю логику функций или процедур.

```
Implementation
// распечатывает целочисленный массив
procedure PrintIntArr(a: array of integer);
begin
  for var i := 0 to a.High do
    Print($'{a[i]} ');
end;
// выводит сумму массива
function ArraySum(a: array of integer): integer;
begin
  // …
end;
```

Многофайловая компоновка

DynArrs.pas

```
unit DynArrs;
// ========================================= interface section
interface
/// <summary>
/// prints integer array
/// </summary>
procedure PrintIntArr(a: array of integer);
/// <summary>
/// outputs a sum of array
/// </summary>
function ArraySum(a: array of integer): integer;
// ========================================= implementation section
implementation
// prints integer array
procedure PrintIntArr(a: array of integer);
begin
  Assert(a <> nil);
  for var i := 0 to a.High do
    Print($'{a[i] }');
end;

// outputs a sum of array
function ArraySum(a: array of integer): integer;
begin
  Assert(a <> nil, 'ArraySum: a <> nil');
  Result := 0;
  for var i := 0 to a.High do
    Result += a[i];
end;
end.
```

Task-01.pas

```
uses DynArrs;

begin
  var a := new integer[3](1, 2, 3);
  ArraySum(a).Println;
end.
```

Q & A