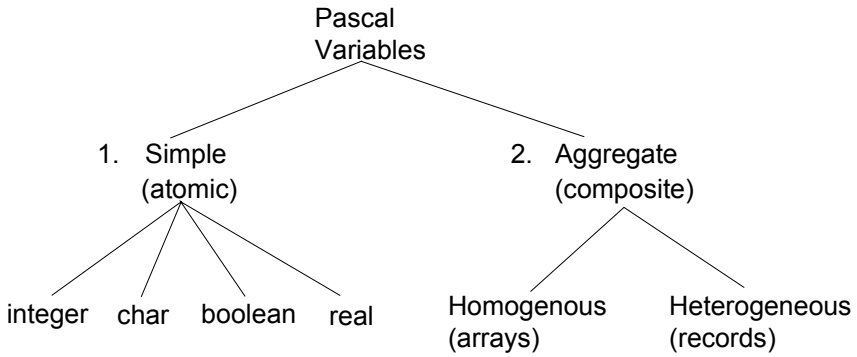
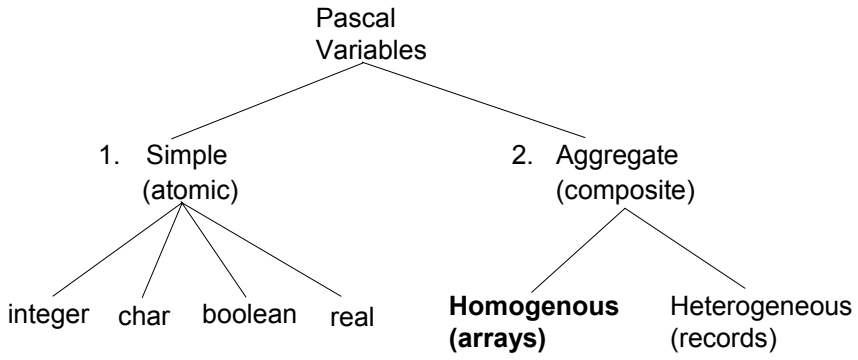
Arrays

In this section of notes you will be introduced to a composite type where all elements must be of the same type (homogeneous): arrays

Types Of Variables



Types Of Variables



Example Problem

Write a program that will track the percentage grades for a class of students. The program should allow the user to enter the grade for each student. Then it will display the grades for the whole class along with the average.

Why Bother With Composite Types?

For a compilable example look in Unix under:
/home/231/tamj/examples/arrays/classList1.p

```
const
  CLASS_SIZE =  5;
begin
  var stu1     : real;
  var stu2     : real;
  var stu3     : real;
  var stu4     : real;
  var stu5     : real;
  var total    : real;
  var average : real;
```

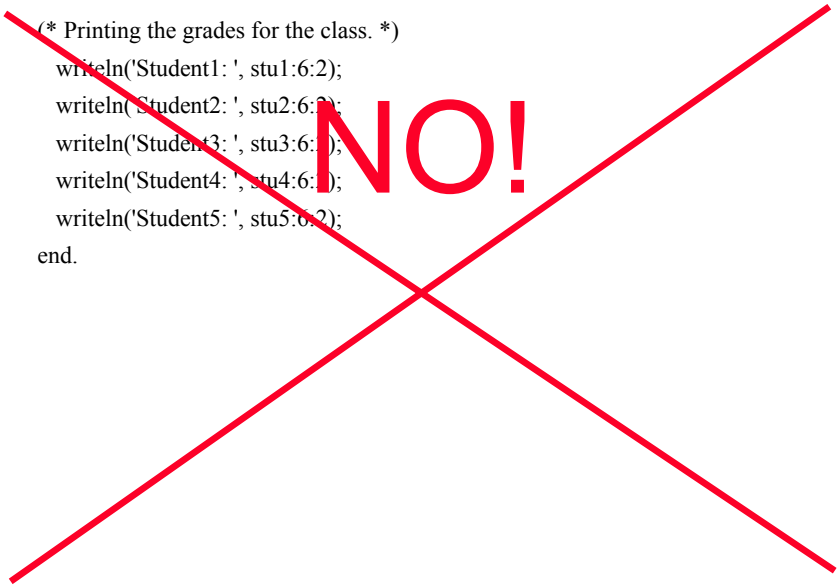

Why Bother With Composite Types? (2)

```
  write('Enter grade for student number 1: ');
  readln(stu1);
  write('Enter grade for student number 2: ');
  readln(stu2);
  write('Enter grade for student number 3: ');
  readln(stu3);
  write('Enter grade for student number 4: ');
  readln(stu4);
  write('Enter grade for student number 5: ');
  readln(stu5);
  total := stu1 + stu2 + stu3 + stu4 + stu5;
  average := total / CLASS_SIZE;
  writeln('The average grade is ', average:6:2, '%');
```

With Bother With Composite Types? (3)

```
 (* Printing the grades for the class. *)
  writeln('Student1: ', stu1:6:2);
  writeln('Student2: ', stu2:6:2);
  writeln('Student3: ', stu3:6:2);
  writeln('Student4: ', stu4:6:2);
  writeln('Student5: ', stu5:6:2);
end.
```


With Bother With Composite Types? (3)



What's Needed

- A composite variable that is a collection of another type.
  - The composite variable can be manipulated and passed throughout the program as a single entity.
  - At the same time each element can be accessed individually.
- What's needed…an array!

Declaring Arrays

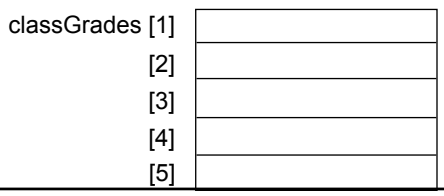
As with any other variable, you must first create an array in memory by declaring an instance.

Format:

name: array [*low index..high index*] of *element type*;

Example:

```
const
   CLASS_SIZE = 5;
                        :    :
   var classGrades : array [1..CLASS_SIZE] of real;
```



Accessing Data In The Array

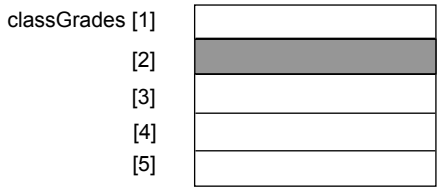
To manipulate an array you need to first indicate which array is being accessed

- Done via the name of the array e.g., "classGrades"

Using only the **name of the array** refers to the whole array

If you are accessing a single element, you need to indicate which element that you wish to access.

- Done via the array index e.g., "classGrades[2]"



Use the **array name and a subscript** (the 'index') refers to a single element

Assigning Data To The Array

Format:

(Whole array)	(One element)
name of array := value;	name of array [index] := value;

Examples (assignment via the assignment operator):

(Whole array)	(One element)
firstArray := secondArray;	classGrades [1] := 100;

Assigning Data To The Array (2)

Examples (assigning values via read or readln):

(Single element)

```
readln(classGrades[1]);
```

(Whole array – all elements)

```
for i: = 1 to CLASS_SIZE do
begin
    write('Input grade for student No. ', i, ': ');
    readln(classGrades[i]);
end;
```


Assigning Data To The Array (3)

Example: (Whole array – all elements: Character arrays only)

```
var charArray : array [1..SIZE] of char;
readln(charArray);
```

Important note: arrays cannot be passed as a parameters to read or readln (except for one-dimensional character arrays)

Accessing The Data In The Array

Examples (displaying information):

(Single element)

```
writeln(classGrades[1]);
```

(Whole array – all elements)

```
for i := 1 to CLASS_SIZE do
    writeln('Grade for student No. ', i:2, ' ', classGrades[i]:6:2);
```


Accessing The Data In The Array (2)

Example: (Whole array – all elements: Character arrays only)

```
var charArray : array [1..SIZE] of char;
write(charArray);
```

Important note: arrays cannot be passed as a parameters to write or writeln (except for one-dimensional character arrays)

Revised Version Using An Array

For a compilable example look in Unix under:
/home/231/tamj/examples/arrays/classList2.p

```
const
  CLASS_SIZE =  5;
begin
  var classGrades  : array [1..CLASS_SIZE] of real;
  var i            : integer;
  var total        : real;
  var average      : real;

  total := 0;
```


Class Example Using An Array (2)

```
for i := 1 to CLASS_SIZE do
begin
    write('Enter grade for student no. ', i, ': ');
    readln (classGrades[i]);
    total := total + classGrades[i];
end;
average := total / CLASS_SIZE;
writeln;
writeln('The average grade is ', average:6:2, '%');

for i := 1 to CLASS_SIZE do
   writeln('Grade for student no. ', i, ' is ', classGrades[i]:6:2, '%');
```

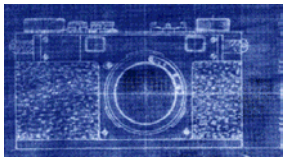
Passing Arrays As Parameters

1. Declare a type for the array.

   e.g.

   ```
   const
       CLASS_SIZE = 5;
   type
     Grades = array [1..CLASS_SIZE] of real;
   ```

   - Declaring a type does not create an instance
     - A type only describes the attributes of a new kind of variable that can be created and used.
     - No memory is allocated.



Passing Arrays As Parameters (2)

2. Declare an instance of this type.

   e.g., var lecture01 : Grades;
   - Memory is allocated!

3. Pass the instance to functions/procedures as you would any other parameter.

   ```
   (Function/procedure call)
   displayGrades (lecture01, average);

   (Function/procedure definition)
   procedure displayGrades (lecture01  : Grades;
                            average    : real);
   ```

Passing Arrays As Parameters: An Example

The full example can be found in Unix under /home/231/tamj/examples/classList3.p):

```
program classList (input, output);

const
  CLASS_SIZE =  5;

type
  Grades = array [1..CLASS_SIZE] of real;

procedure tabulateGrades (var lecture01  : Grades;
                          var average    : real);
var
  i      : integer;
  total : real;
```


Passing Arrays As Parameters: An Example (2)

```
begin      (* tabulateGrades *)
  total := 0;
  for i := 1 to CLASS_SIZE do
  begin
    write('Enter grade for student no. ', i, ': ');
    readln(lecture01[i]);
    total := total + lecture01[i];
  end;
  average := total / CLASS_SIZE;
  writeln;
end;       (* tabulateGrades *)
```

Passing Arrays As Parameters: An Example (3)

```
procedure displayGrades (lecture01 : Grades;
                         average   : real);
var
   i : integer;
begin
  writeln('Grades for the class...');
  for i := 1 to CLASS_SIZE do
    writeln('Grade for student no. ', i, ' is ', lecture01[i]:6:2, '%');
  writeln('The average grade is ', average:6:2, '%');
  writeln;
end;
```


Passing Arrays As Parameters: An Example (4)

```
begin
  var lecture01 : Grades;
  var average   : real;
  tabulateGrades (lecture01, average);
  displayGrades (lecture01, average);
end.
```

Returning Arrays From Functions

1. Declare a type for the array.

   e.g.

   ```
   const
       CLASS_SIZE = 5;
   type
       Grades = array [1..CLASS_SIZE] of real;
   ```

2. Declare an instance of this type.

   e.g.,

   ```
   var lecture01 : Grades;
   ```

3. Return the instance of the array as you would any other return value.

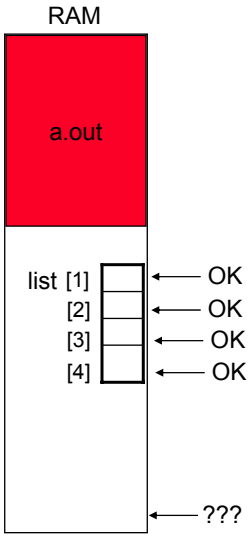
   (Function call)

   ```
   lecture01 := fun (lecture01);
   ```

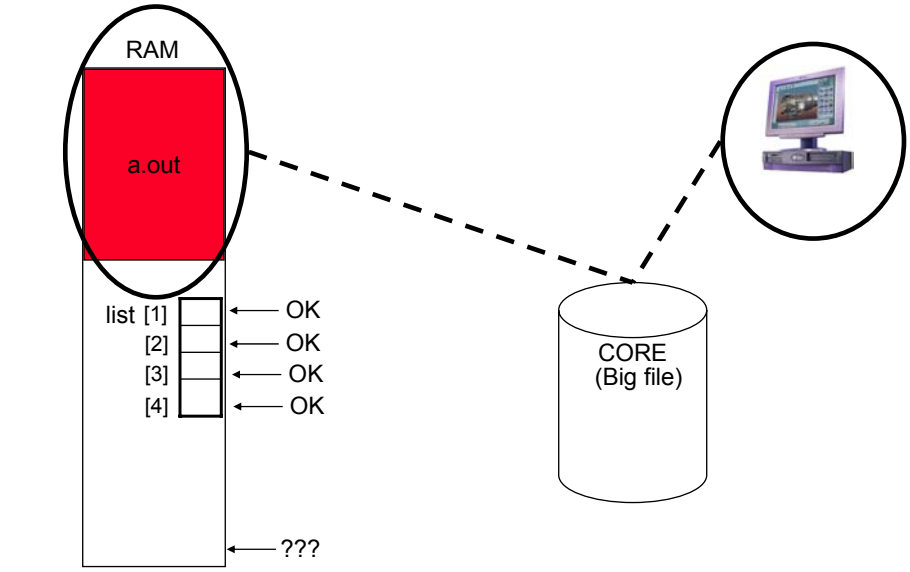
   (Function definition)

   ```
   function fun (lecture01 : Grades ): Grades;
   ```


Segmentation Faults And The Array Bounds (2)



Segmentation Faults And The Array Bounds (2)



Wav file from "The SImpsons"

Segmentation Faults And The Array Bounds (3)

- When using an array take care not to exceed the bounds.
- Ways of reducing the likelihood of exceeding the bounds of the array:

1. Use a constant in conjunction with arrays e.g.,

```
const
    MAX = 5;
```

2. Refer to the constant when declaring an array:

```
var aList : array [1..MAX] of integer;
```

3. Refer to the constant when declaring the type for the array:

```
type
   List = array [1..MAX] of integer;
```

4. Refer to the constant when iterating/traversing through the array:

```
for i := 1 to MAX do
    writeln('Grade for student no. ', i, ' is ', lecture01[i]:6:2, '%');
```

Segmentation Faults And The Array Bounds (4)

5. Make sure that array indices are properly initialized.
   - You may need to verify this assumption with debugging statements.

Incorrect ☹: What is the current value of index 'i'?

```
program array1 (output);
begin
  var i    : integer;
  var list : array [1..2] of integer;
  list [i] := i;
  writeln (list[i]);
end.
```

Correct ☺: Always initialize your variables before using them: in this case the index 'i' is set to a value within the bounds of the array before it's used.

```
program array2 (output);
begin
  var i    : integer;
  var list : array [1..2] of integer;
  i := 2;
  list [i] := i;
  writeln (list[i]);
end.
```


The String Type

It is a special type of character array.

Format for declaration:

var *name* : string [*No of elements*];

Example declaration:

var firstName : string [MAX];

Benefits Of The String Type

1. The end of array is marked.
2. Many operations have already been implemented.

Marking The End Of The Array

The full example can be found in Unix under the path:
/home/231/tamj/examples/arrays/stringExample.p

```
program stringExample (output);
const
  MAX = 8;
begin
  var list1 : array [1..MAX] of char;
  var list2 : string[MAX];
  list1 := 'abcdefg';
  list2 := 'abcdefg';
  writeln('-', list1, '-');
  writeln('-', list2, '-');
end.
```

The Contents Of The String "List2"

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
'a'	'b'	'c'	'd'	'e'	'f'	'g'	END

Strings Are A Built-In Type[1]

- This means that they can be passed as parameter in the same fashion as other built in types, no type needs to be defined beforehand.

Format:

procedure *procedureName* (*stringName* : string);
OR
procedure *procedureName* (var *stringName* : string);

Examples:

procedure proc1 (list : string);
OR
procedure proc2 (var list : string);

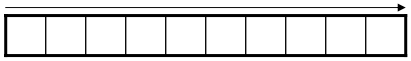
1 For many programming languages and some versions of Pascal

When To Use Arrays Of Different Dimensions

- Determined by the data – the number of categories of information determines the number of dimensions to use.

  Examples:
- (1D array)
  - Tracking grades for a class
  - Each cell contains the grade for a student i.e., grades[i]
  - There is one dimension that specifies which student's grades are being accessed

  One dimension (which student)



- (2D array)
  - Expanded grades program
  - Again there is one dimension that specifies which student's grades are being accessed
  - The other dimension can be used to specify the lecture section

When To Use Arrays Of Different Dimensions (2)

- (2D array continued)

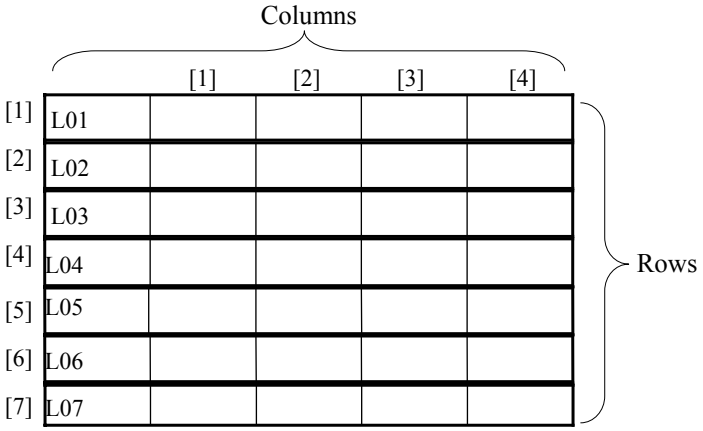
Student

Lecture section

	First student	Second student	Third student	…
L01				
L02				
L03				
L04				
L05				
:				
L0N				

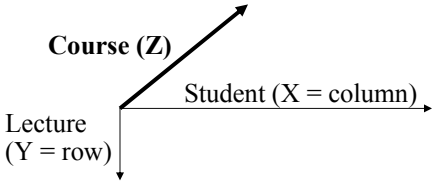
When To Use Arrays Of Different Dimensions (3)

- (2D array continued)
- Notice that each row is merely a 1D array
- (A 2D array is an array containing rows of 1D arrays)



When To Use Arrays Of Different Dimensions (4)

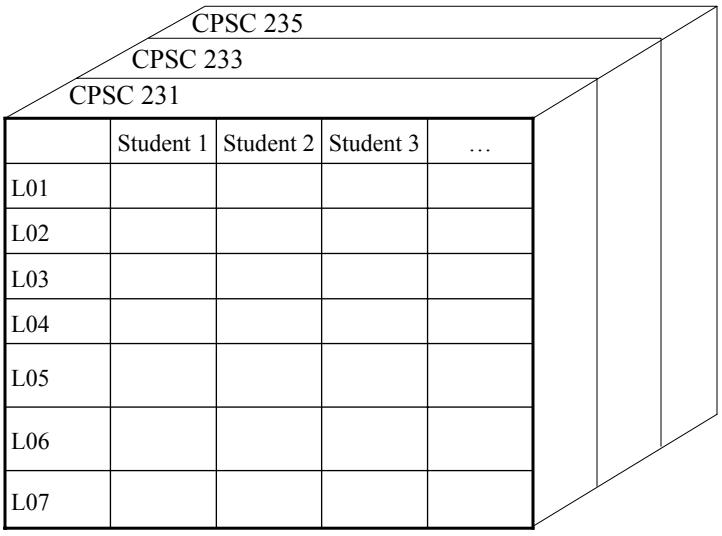
- (3D array – take the 2D array but allow for multiple courses).
- The third dimension specifies which course grades are being tracked.



Note:

1. The standard approach for specifying the dimensions is to specify the row coordinate (Y) *and then* the column coordinate (X).
2. The size of a dimension must be the same for all elements along that dimension e.g., all rows must be of the same size

When To Use Arrays Of Different Dimensions (5)



	Student 1	Student 2	Student 3	…
L01				
L02				
L03				
L04				
L05				
L06				
L07				

Declaring Multi-Dimensional Arrays

Format:

(Two dimensional arrays)

Name : array [*min..max*, *min..max*] of *type*;

Rows Columns

(Three dimensional arrays)

Name : array [*min..max*, *min..max, min..max*] of *type*;

Examples:

var johnFinances : array [1..3, 1..7] of real;

var cube : array[1..6, 1..6, 1..6] of char;

Declaring Multi-Dimensional Arrays As A Type

Format:

Type declaration

Type name = array [*min*..*max*, *min*..*max*] of *element type*;

Type name = array [*min*..*max*, *min*..*max, min*..*max*] of *element type*;

Variable declaration

array name : *Type name*;

Declaring Multi-Dimensional Arrays As A Type (2)

Example:

Type declaration

```
Finances = array [1..3, 1..7] of real;
Cube = array [1..6, 1..6, 1..6] of char;
```

Variable declaration

```
var johnFinances : Finances;
var aCube        : Cube;
```

Accessing / Assigning Values To Elements

Format:

name [row][column] := name [row][column];

Example:

```
finances [1][1] := 4500;
writeln (finances[1][1]);
```


Multi-Dimensional Arrays And Input/Output

- Arrays of more than one dimension (including multi-dimensional character arrays) cannot be passed as parameters to: read, readln, write, writeln.
- Only one-dimensional character arrays can be passed as parameters to these procedures.

Example 2D Array Program: A Character-Based Grid

You can find the full program in Unix under:
/home/231/tamj/examples/arrays/grid.p

A Character-Based Grid

```
program gridExample (input, output);

const
  MAX_ROWS        = 4;
  MAX_COLUMNS     = 4;
  NO_COMBINATIONS = 10;

type
  Grid =   array[1..MAX_ROWS, 1..MAX_COLUMNS] of char;
```

A Character-Based Grid (2)

```
function generateElement (temp : integer) : char;
var
  anElement : char;
begin
  case (temp) of
    1, 2, 3, 4, 5, 6 :
    anElement := ' ';

    7, 8, 9:
    anElement := '*';

    10:
    anElement := '.';
```


A Character-Based Grid (3)

```
    else
    begin
      writeln('<< Error with the random no. generator.>>');
      writeln('<< Value should be 1-10 but random value is ', temp);
      anElement := '!';
    end;
  end;
  generateElement := anElement;
end;
```

A Character-Based Grid (4)

```
procedure initialize (var aGrid :  Grid);
var
  r       : integer;
  c       : integer;
  temp : integer;
begin
  for r := 1 to MAX_ROWS do
  begin
    for c := 1 to MAX_COLUMNS do
    begin
      temp := random(NO_COMBINATIONS) + 1;
      aGrid[r][c] := generateElement(temp);
    end;
  end;
end;
```


A Character-Based Grid (5)

```
procedure display (aGrid : Grid);
var
  r : integer;
  c : integer;
begin
  for r := 1 to MAX_ROWS do
  begin
    for c := 1 to MAX_COLUMNS do
    begin
      write(aGrid[r][c]);
    end;
    writeln;
  end;
end;
```

A Character-Based Grid (6)

```
procedure displayLines (aGrid : Grid);
var
  r : integer;
  c : integer;
begin
  for r := 1 to MAX_ROWS do
  begin
    writeln(' - - - -');
    for c := 1 to MAX_COLUMNS do
    begin
      write('|', aGrid[r][c]);
    end;
    writeln('|');
  end;
  writeln(' - - - -');
end;
```


A Character-Based Grid (7)

```
begin
  var aGrid : Grid;
  initialize(aGrid);
  writeln('Displaying grid');
  writeln('================');
  display(aGrid);
  writeln;
  writeln('Displaying grid with bounding lines');
  writeln('============================');
  displayLines(aGrid);
end.
```

Valid Operators: 1D Character Arrays And Strings

- The relational operators will work with the String type and 1-dimensional character arrays.
- They will not work with other type of arrays.

You Should Now Know

- What is the difference between simple types (atomic) and composite types (aggregate).
- What is the benefit of using homogeneous composite types (arrays)
- How to declare arrays.
- How to access or assign values to array elements.
- How to work with an entire array (e.g., access or assign values to different parts).
- How to pass instances of arrays into functions and procedures and how to return an array from a function.
- What is a segmentation fault and what is a core dump file.
- How to declare and to use instances of a string type.
- The number of dimensions to declare for an array.
- How to declare and traverse arrays of multiple dimensions.
- How to display "bounding lines" around array elements as a formatting technique.