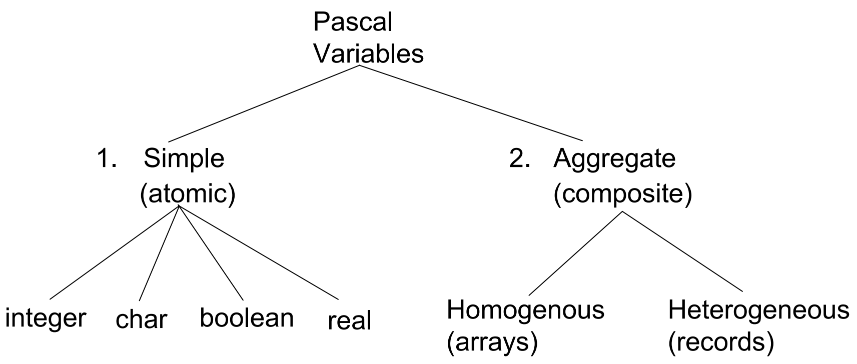
Arrays

In this section of notes you will be introduced to a homogeneous composite type, one-dimensional arrays

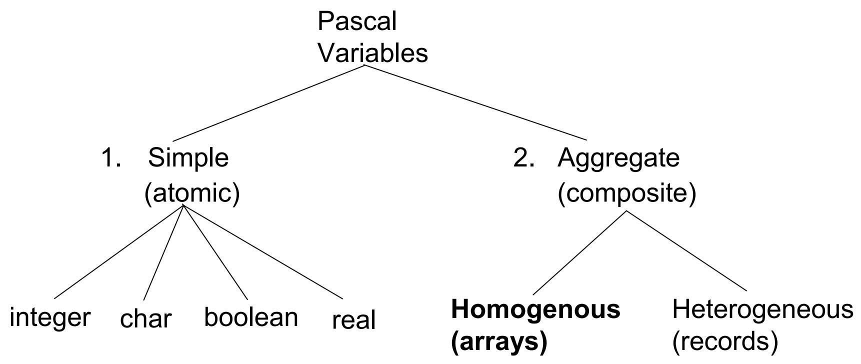
James Tam

Types Of Variables



James Tam

Types Of Variables



Why Bother With Composite Types?

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

```
const

   CLASSSIZE =  5;

begin

   var stu1, stu2, stu3, stu4, stu5, total, average : real;

   write('Enter grade for student number 1: ');

   readln(stu1);

   write('Enter grade for student number 2: ');

   readln(stu2);
```

Why Bother With Composite Types? (2)

```
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 / CLASSSIZE;
writeln('The average grade is ', average:6:2, '%');
```

With Bother With Composite Types? (3)

```
(* Printing the grades for the class. *)
writeln('Student: ', 1, stu1);
writeln('Student: ', 2, stu2);
writeln('Student: ', 3, stu3);
writeln('Student: ', 4, stu4);
writeln('Student: ', 5, stu5);
```

With Bother With Composite Types? (3)

```
(* Printing the grades for the class. *)
writeln('Student: ', 1, stu1);
writeln('Student: ', 2, stu2);
writeln('Student: ', 3, stu3);
writeln('Student: ', 4, stu4);
writeln('Student: ', 5, stu5);
```

NO!

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.

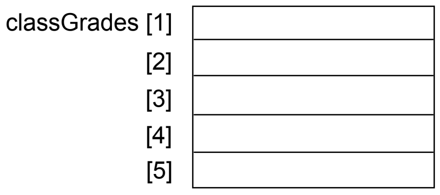
Declaring Arrays

Format:

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

Example:

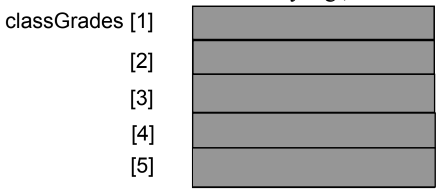
classGrades : array [1..CLASSSIZE] of real;



Accessing Data In The Array

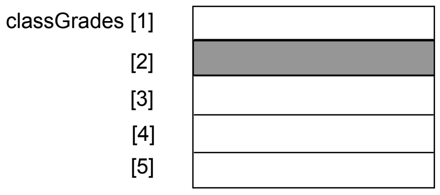
First you need to indicate which array is being accessed

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



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]"



Assigning Data To The Array

Format:

(Whole array)	(One element)
name of array	name of array [index]

Examples (assignment via the assignment operator):

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

James Tam

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

James Tam

Assigning Data To The Array (3)

(Whole array – all elements: Character arrays only)

```
var charArray : array [1..5] of char;
readln(charArray);
```

Accessing Data In The Array

Examples (displaying information):

(Single element)

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

(Whole array – all elements)

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

(Whole array – all elements: Character arrays only)

```
var charArray : array [1..5] of char;
write(charArray);
```

Revised Version Using An Array

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

```
const
  CLASSSIZE =  5;
begin
  var classGrades    : array [1..CLASSSIZE] of real;
  var i              :  integer;
  var total, average : real;
  total := 0;
```

Class Example Using An Array (2)

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

for i := 1 to CLASSSIZE 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.

   ```
   type
       Grades = array [1..CLASSSIZE] of real;
   ```

2. Declare an instance of this type.

   e.g.,

   ```
   var L01 : Grades;
   ```

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

   ```
   (Function/procedure call)
    displayGrades (L01, average);

   (Function/procedure definition)
   procedure displayGrades (L01       : Grades;
                            average : real);
   ```

Passing Arrays As Parameters: An Example

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

```
program classList (input, output);
const
  CLASSSIZE =  5;
type
  Grades = array [1..CLASSSIZE] of real;
procedure tabulateGrades (var L01       : Grades;
                          var average : real);
var
  i      : integer;
  total : real;
```

Passing Arrays As Parameters: An Example (2)

```
begin      (* tabulateGrades *)
  total := 0;
  for i := 1 to CLASSSIZE do
  begin
    write('Enter grade for student no. ', i, ': ');
    readln(L01[i]);
    total := total + L01[i];
  end;
  average := total / CLASSSIZE;
  writeln;
end;       (* tabulateGrades *)
```

James Tam

Passing Arrays As Parameters: An Example (3)

```
procedure displayGrades (L01     : Grades;
                         average : real);
var
  i : integer;
begin
  writeln('Grades for the class...');
  for i := 1 to CLASSSIZE do
    writeln('Grade for student no. ', i, ' is ', L01[i]:6:2, '%');
  writeln('The average grade is ', average:6:2, '%');
  writeln;
end;
```

James Tam

Passing Arrays As Parameters: An Example (4)

```
begin
  var L01      : Grades;
  var average : real;
  tabulateGrades (L01, average);
  displayGrades (L01, average);
end.
```

Returning Arrays From Functions

1. Declare a type for the array.

   e.g.
   ```
   type
       Grades = array [1..CLASSSIZE] of real;
   ```

2. Declare an instance of this type.

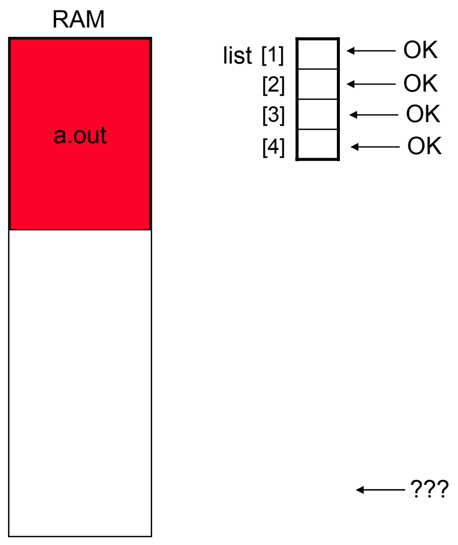
   e.g.,
   ```
   var L01 : Grades;
   ```

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

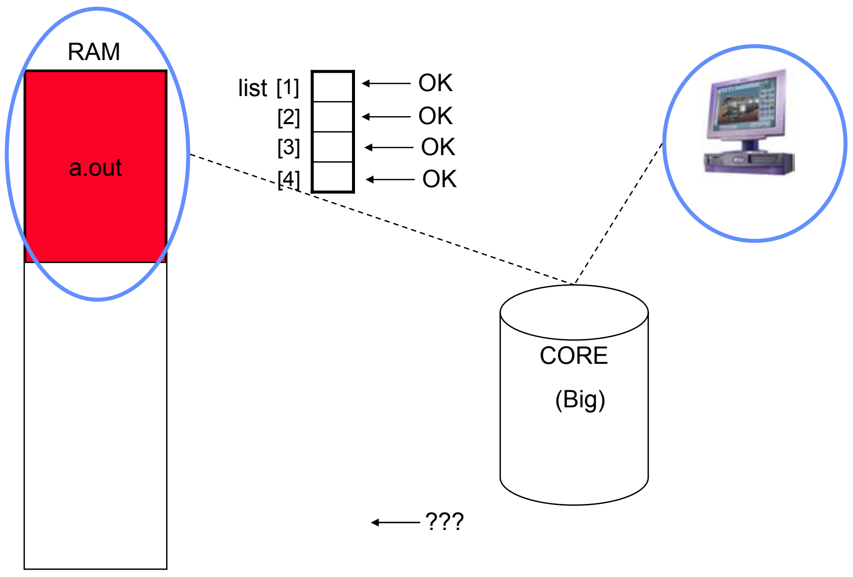
   (Function/procedure call)
   ```
   L01 := fun (L01);
   ```

   (Function/procedure definition)
   ```
   function fun (L01 : Grades ): Grades;
   ```

Segmentation Faults And Arrays



Segmentation Faults And Arrays



Wav file from “The SImpsons”

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
- How to pass instances of arrays into methods and how to return an array from a function.
- What is a segmentation fault and core dump file.