Arrays

Today

- What are arrays?
- Initializing arrays
- Midterm review

Arrays

Using Arrays

Think of a list of values - sales per month in 2021

```
32 54 67.5 29 35 80.3 115 98 100 65 210.5 140
```

(all of the same type, of course)

(storable as **`double`**s)

Using Arrays

```
32 54 67.5 29 35 80.3 115 98 100 65 210.5 140
```

Which is the largest value in this set?

(You must look at every single value to decide.)

Using Arrays

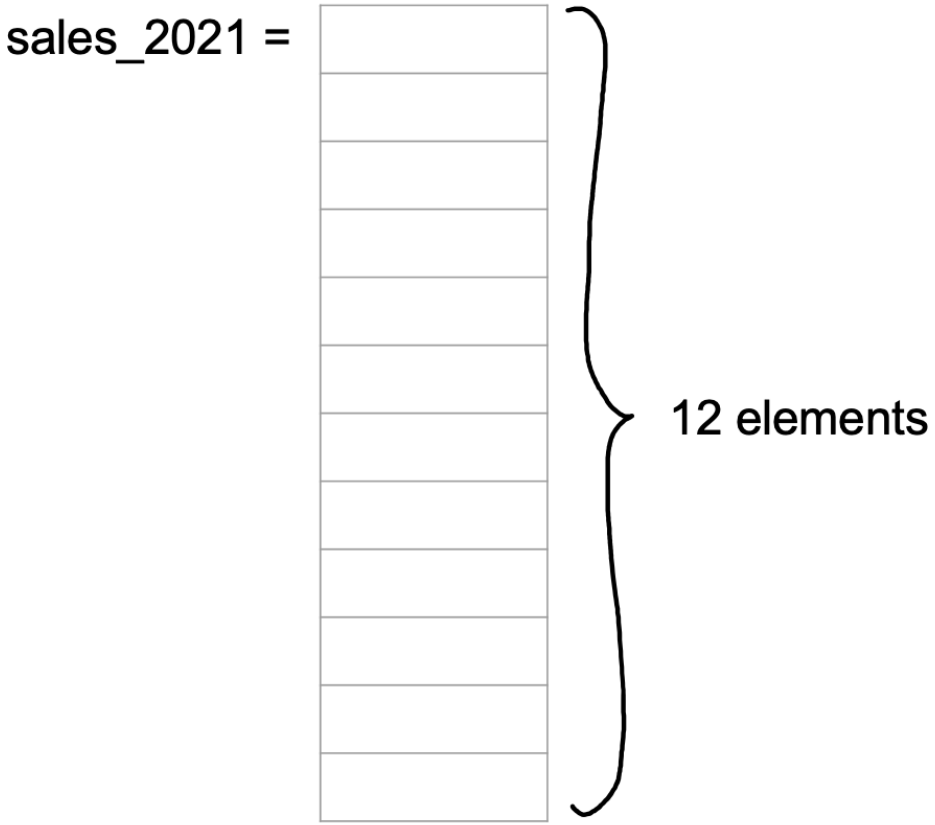
```
32 54 67.5 29 35 80.3 115 98 100 65 210.5 140
```

- So you would create a variable for each, of course!

```
double jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec;
```

Then what ???

Defining Arrays



An “array of double”

Twelve elements of **double** type can be stored under one name as an array.

```
double sales_2021[12];
```

type of each element

number of elements – the “size” of the array, must be a constant

Introduction to Arrays

Definition: An array is a collection of data of the same type, referenced as different elements of the same name.

- First "aggregate" data type
  - Means "grouping"
  - *int, float, double, char* are simple data types
- Used for lists of like items
  - Test scores, temperatures, names, etc.
  - Avoids declaring multiple simple variables
  - Can manipulate "list" as one entity

Declaring Arrays

Declare the array → allocates memory

```
int score[5];
```

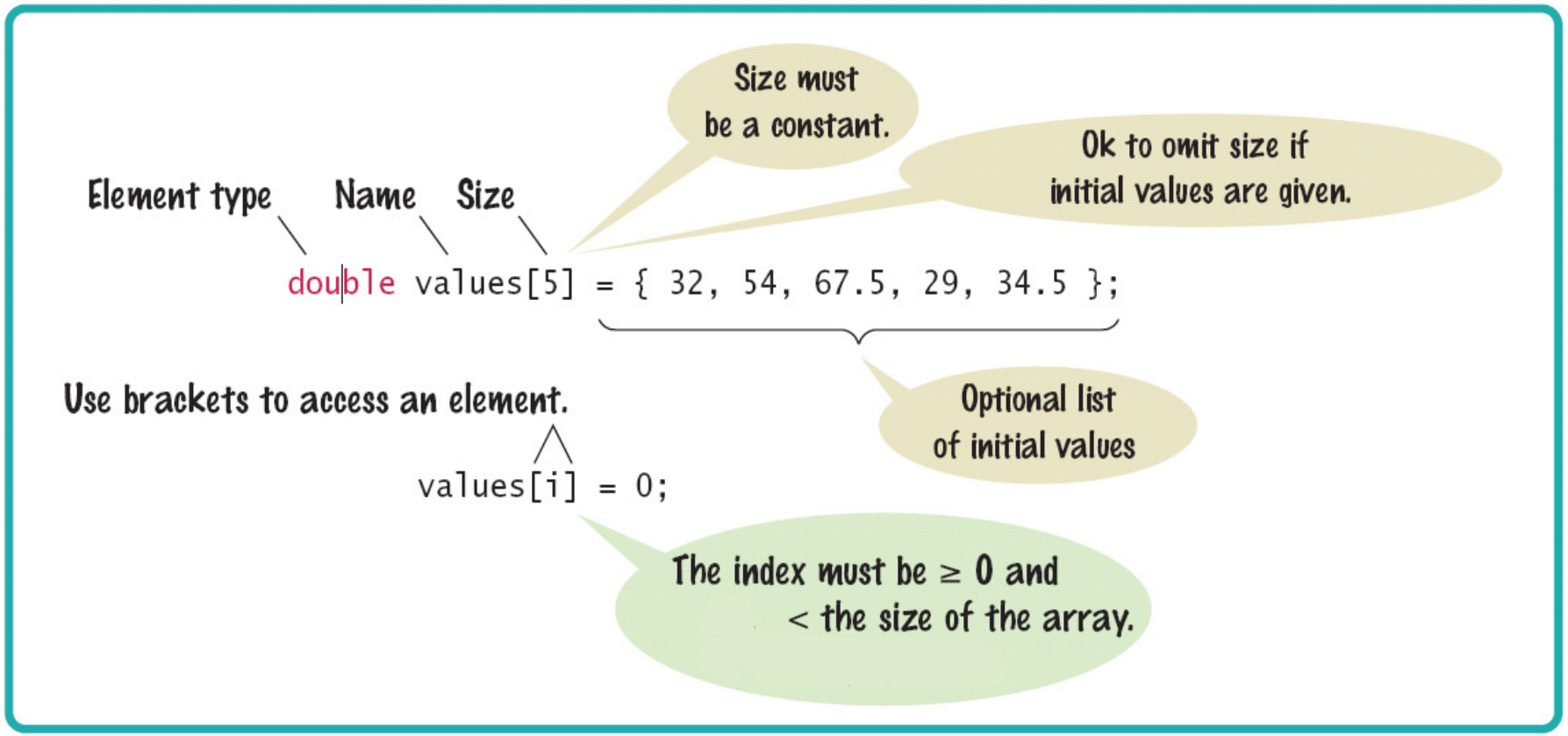
- Declares array of 5 integers named "score"
- Similar to declaring five variables:

```
int score[0], score[1], score[2], score[3], score[4];
```

- Individual parts can be called many things:
  - Indexed or subscripted variables
  - "Elements" of the array
  - Value in brackets is called index or subscript
  - Numbered from 0 to (size – 1)

Array Syntax

Defining an Array



Accessing Arrays

- Access using index/subscript

```
cout << score[3];
```

- Note two uses of brackets:
  - In declaration, specifies SIZE of array
  - Anywhere else, specifies a subscript
- Size, subscript need not be literal

```
int score[MAX_SCORES];
score[n+1] = 99;
```

--> If n is 2, identical to: score[3]

Accessing Arrays

sales_2021 =

32.0
54.0
67.5
29.0
35.0
80.0
115.0
98.0
100.0
65.0
210.5
140

12 elements

Accessing an Array Element

sales_2021 =

32.0
54.0
67.5
29.0
35.0
80.0
115.0
98.0
100.0
65.0
210.5
140

To access the element at index 5 using this notation:

```
sales_2021[5]
```

5 is the *index*.

```
double sales_2021[12];
...
cout << sales_2021[5] << endl;
```

The output will be **80**

Accessing an Array Element

sales_2021 =

32.0
54.0
67.5
29.0
35.0
~~80.0~~ 17.7
115.0
98.0
100.0
65.0
210.5
140

To access the element at index 5 using this notation:

```
sales_2021[5]
```

5 is the *index*.

```
sales_2021[5] = 17.7;
...
cout << sales_2021[5] << endl;
```

The output will be **17.7**

Accessing an Array Element

- That is, the legal elements for the **sales_2021** array are:
- sales_2021[0] , the first element
- sales_2021[1] , the second element
- sales_2021[2] , the third element
- sales_2021[3] , the fourth element
- sales_2021[4] , the fifth element
- …
- sales_2021[11] , the twelfth and last legal element
- recall: double sales_2021[12];

- The index must be >= 0 and <= 11.

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 is ... 12 numbers.

Array Usage

- Powerful storage mechanism
- Can issue commands like:
  - "Do this to i^{th} indexed variable”, where i is computed by program
  - "Display all elements of array score"
  - "Fill elements of array score from user input"
  - "Find highest value in array score"
  - "Find lowest value in array score”
- Disadvantages: size MUST BE KNOWN at declaration