

# Working With Arrays of Data

- Consider a program that:
  - Gets 5 numbers from the user
  - Outputs the average
- What variables are needed?
- What are the steps the program should take?

# Working With Arrays of Data

- Consider a program that:
  - Gets 5 numbers from the user
  - Outputs the average
- What variables are needed?
- What are the steps the program should take?
- But what if it takes 100 numbers instead of 5? 1000?

# Working With Arrays of Data

- Using arrays, we can:
  - Allocate all 5 integers at once
  - Give them a single name
  - Access them by *index*

# Naming Arrays of Data

- When you declare a variable the computer:
  - Allocates space for it `int num;`
  - Gives it a name
- The space allocation is based on the type of the variable `bytes of each element for the corresponding time * length = 400 bytes`  
`4 bytes for first element, 4 bytes for the second element.`
  - Main memory is one long sequence of bytes
  - An integer (int) takes 4 bytes on most systems
- So why not allocate multiple ints and give them one name?
  - Requires a new syntax for allocation
  - Requires a way to specify which int you want to work with

# Arrays

- Array: a collection of a fixed number of components wherein all of the components have the same data type
- In a one-dimensional array, the components are arranged in a list form
- Syntax for declaring a one-dimensional array:

```
dataType arrayName[intExp];
```

`intExp` evaluates to a positive integer

# Arrays (continued)

- Example:

```
int num[5];
```

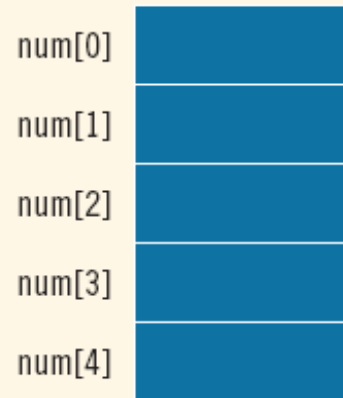


FIGURE 9-1 Array num

# Accessing Array Components

- General syntax:

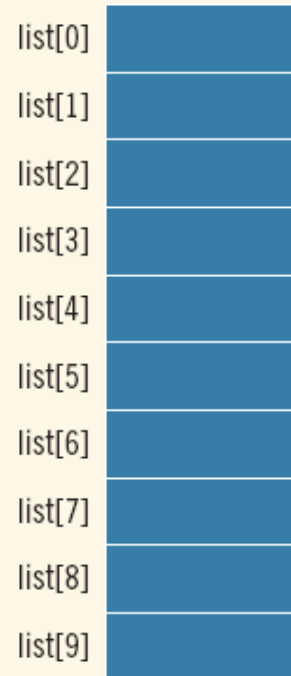
`array_name[index]`

is any expression whose value is a nonnegative integer

- Index value specifies the position of the component in the array
- `[]` is the **array subscripting operator**
- The array index always starts at 0

# Accessing Array Components (continued)

```
int list[10];
```

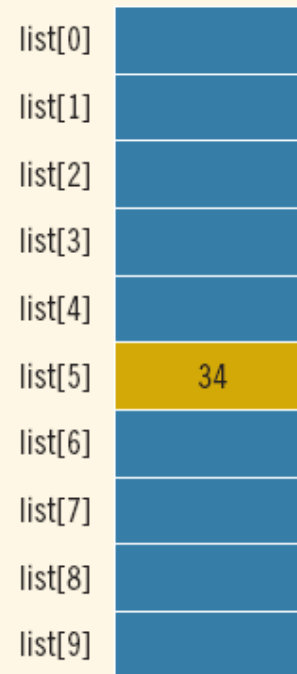


**FIGURE 9-2** Array `list`



# Accessing Array Components (continued)

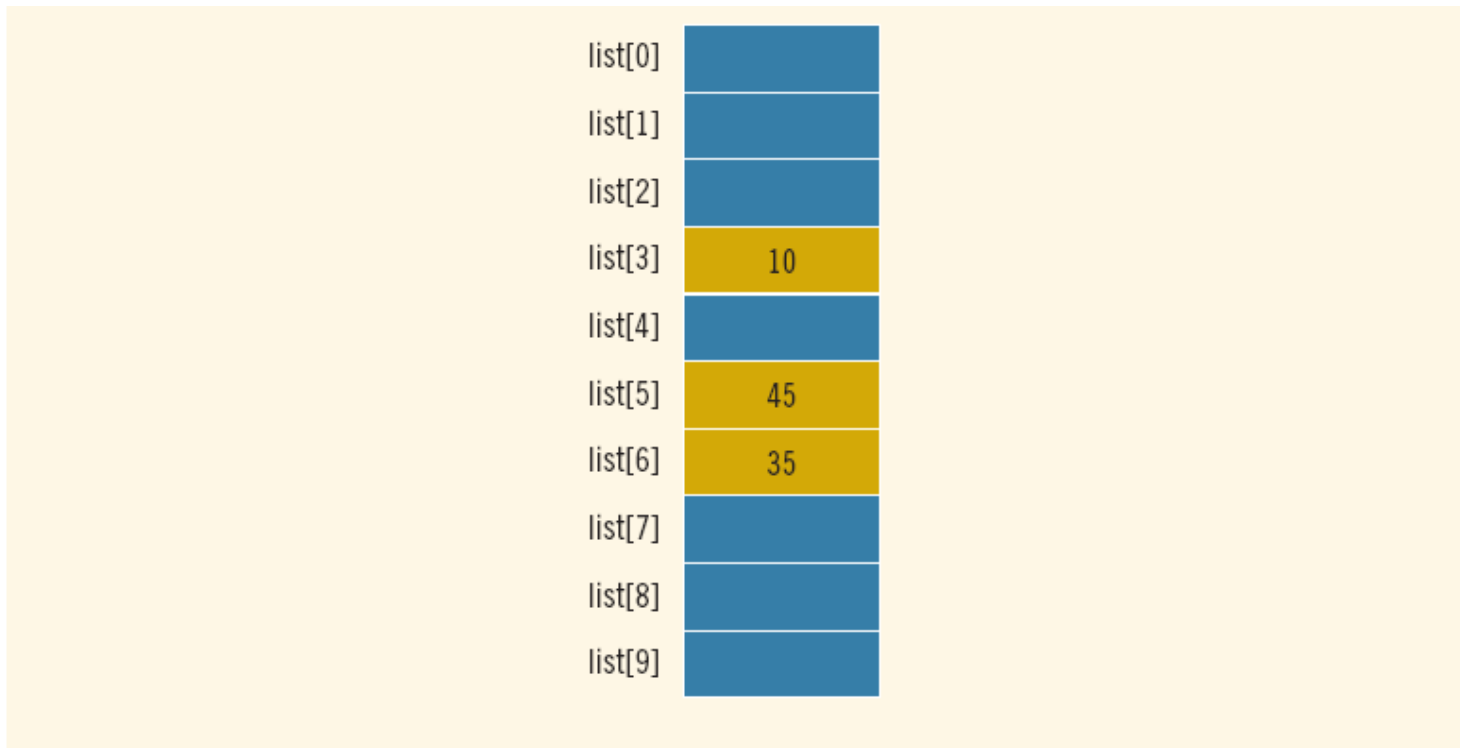
```
list[5] = 34;
```



**FIGURE 9-3** Array `list` after execution of the statement `list[5] = 34;`

# Accessing Array Components (continued)

```
list[3] = 10;  
list[6] = 35;  
list[5] = list[3] + list[6];
```



**FIGURE 9-4** Array list after execution of the statements `list[3] = 10;`, `list[6] = 35;`, and `list[5] = list[3] + list[6];`

# Accessing Array Components (continued)

---

## EXAMPLE 9-2

You can also declare arrays as follows:

```
const int ARRAY_SIZE = 10;  
int list[ARRAY_SIZE];
```

That is, you can first declare a named constant and then use the value of the named constant to declare an array and specify its size.

---

# Working With Arrays of Data

- Using arrays, we can:
  - Allocate all 5 integers at once
  - Give them a single name
  - Access them by *index*
- Using arrays together with for loops, we can:
  - Allocate any number of integers
  - Give them a single name
  - Access them by index
  - Repeat instructions over any number of integers
    - Usually by using a `for` loop counter

# Processing arrays

- For loop is almost always the answer
  - How do you print an array?
  - How do you search an array?
  - How do you copy an array?
  - Etc...
- The for loop counter is used as the array index
  - To access each element in the array sequentially

```
for( int i=0; i<length; i++ )  
{  
    cout << some_array[i] << endl;  
}
```
- Notice that you have to know the length of the array!

# Size, length, count, etc.

- Every array has 2 critical numbers associated with it
  - Maximum Size: how many elements can it store?
    - Also called size some times
  - Actual Count: how many valid pieces of data are in it
    - Also called length, size, count, etc.
    - (not technical terms)
- Every array element *always* has a value
  - You can't really delete anything from an array, only overwrite things
- Data is *always* stored in contiguous elements!
  - Starting from element 0, no empty spaces
  - The last valid index is always at array length-1

# Summary

- Array variables allow you to allocate and name a sequence of values
  - The elements are accessed by index
  - Data is stored from element 0 to element length-1
  - This works really well with for loops
    - If the question involves an array, the answer is usually a for loop!
- Computers are really good at counting and repetitive tasks
  - Arrays allow you, the programmer, to specify things once and allow the computer to do it ten times, a hundred times, a million times...

# Exercises

- Declare an array of 150 doubles
  - Declare a constant SIZE and use it in the array declaration
- Syntax checks!
  - (you rarely access individual array elements like this)
  - Set the 10<sup>th</sup> element in the array of doubles to 5.6
  - Print the 10<sup>th</sup> element
    - e.g. “The 10<sup>th</sup> element is 5.6”
  - Set the 72<sup>nd</sup> element to the value of the 12<sup>th</sup> element
  - Ask the user to enter a value and store it in the 113<sup>th</sup> element
  - Set the 43<sup>rd</sup> – 46<sup>th</sup> elements to the values 7, 8, 9, 10
    - Use a for loop!



# Exercises

- Store 100 copies of the number 50 in your array
  - Declare an integer *length*, set it to 100
  - Set the first *length* elements to 50
- Print all the valid elements in the array
  - Your answer should use *length*
- Store the numbers 1 through 100 in your array
  - e.g. first element is 1, second element is 2, etc.
- Using a for loop, set the first 10 elements in the array to the value of the last 10 valid elements (in reverse)
  - e.g. first element ends up with 100, second with 99, etc.