CSE 1062 **Fundamentals of Programming**

Lecture #9

Spring 2016

Computer Science & Engineering Program
The School of EE & Computing
Adama Science & Technology University

Outline

ASTU

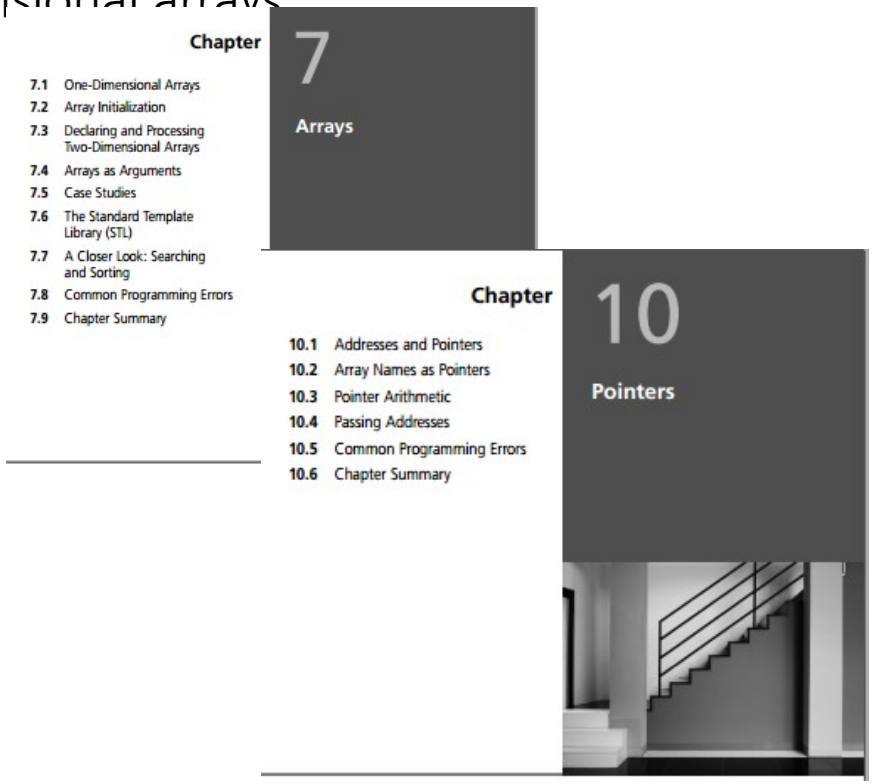
- Arrays
  - One-dimensional arrays
  - Array initialization
  - Declaring and processing two-dimensional arrays
  - Arrays as arguments
- Pointers
  - Addresses and pointers
  - Array names as pointers
  - Pointer arithmetic
  - Passing addresses

Case Studies: Statistical Analysis

[Data Processing] Students' Grading

Reading assignment

- **Chapter 7 of the textbook**
- **Chapter 10 of the textbook**
- Read about <cstring> library
- Read about Standard Template Library and Vectors



One-Dimensional Arrays

- One-dimensional array: A list of related values with the same data type, stored using a single group name (called the array name)
- Syntax:

```
dataType arrayName[number-of-items]
```

- By convention, the number of items is first declared as a constant, and the constant is used in the array declaration

One-Dimensional Arrays (continued)

ASTU

```
const int NUMELS = 6;
int volts[NUMELS];

const int ARRAYSIZE = 4;
char code[ARRAYSIZE];

const int SIZE = 100;
double amount[SIZE];
```

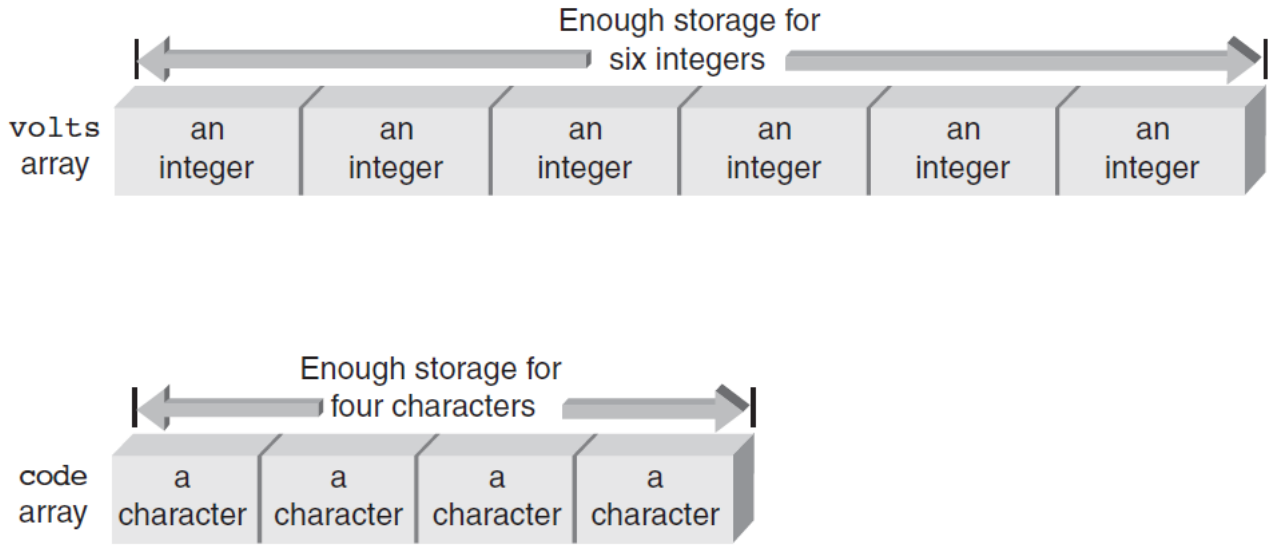
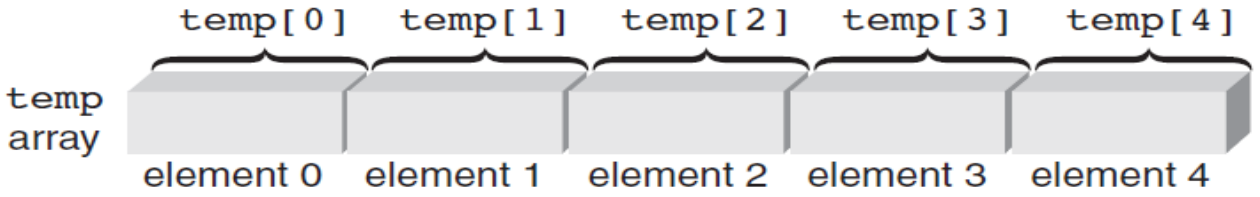


Figure 7.1 The `volts` and `code` arrays in memory

One-Dimensional Arrays (continued)

ASTU

- **Element:** An item in the array
  - Array storage of elements is contiguous
- **Index (or subscript)** of an element: The position of the element within the array
  - Indexes are zero-relative
- To reference an element, use the array name and the index of the element



One-Dimensional Arrays (continued)

- Index represents the offset from the start of the array
- Element is also called **indexed variable** or **subscripted variable**
- Subscripted variable can be used anywhere that a variable can be used
- Expressions can be used within the brackets if the value of the expression
- Yields an integer value
- Is within the valid range of subscripts

One-Dimensional Arrays (continued)

ASTU

- All of the elements of an array can be pr ocessed by using a loop
- The loop counter is used as the array ind ex to specify the element
- Example:

```
sum = 0;
for (i=0; i<5; i++)
   sum = sum + temp[i];
```

Input and Output of Array Values

- Array elements can be assigned values in teractively using a cin stream object
- Out of range array indexes are not check ed at compile-time
  - May produce run-time errors
  - May overwrite a value in the referenced me mory location and cause other errors
- Array elements can be displayed using th e cout stream object

Input and Output of Array Values

ASTU

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      const int MAXTEMPS = 5;
6      int i, temp[MAXTEMPS], total = 0;
7      for (i = 0; i < MAXTEMPS; i++) // enter the temperatures
8      {
9          cout << "Enter a temperature: ";
10         cin >> temp[i];
11     }
12     cout << "\nThe total of the temperatures";
13     for (i = 0; i < MAXTEMPS; i++) // display and total the temperatures
14     {
15         cout << " " << temp[i];
16         total = total + temp[i];
17     }
18     cout << " is " << total << endl;
19     return 0;
20 }
```

```
C:\Users\Tinsae\Desktop\nm.exe
Enter a temperature: 19
Enter a temperature: 22
Enter a temperature: 16
Enter a temperature: 29
Enter a temperature: 25

The total of the temperatures 19 22 16 29 25 is 111

Process returned 0 (0x0)   execution time : 15.259 s
Press any key to continue.
```

Array Initialization

- Array elements can be initialized in the array declaration statement
- Example:

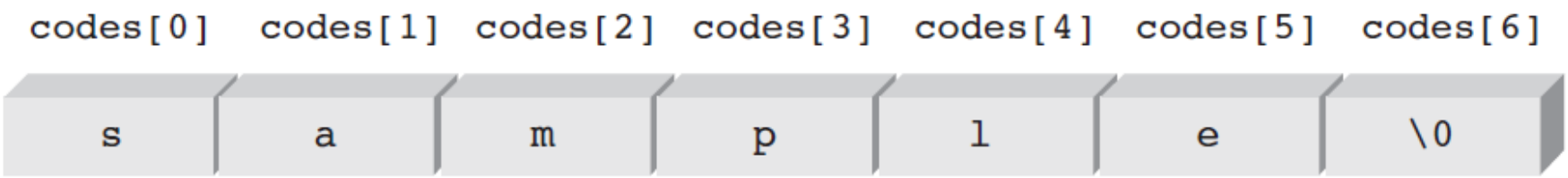
  int temp[5] = {98, 87, 92, 79, 85};
- Initialization:
  - Can span multiple lines, because white space is ignored in C++
  - Starts with array element 0 if an insufficient number of values is specified
- If initializing in the declaration, the size may be omitted

Array Initialization

ASTU

- char array will contain an extra null char acter at the end of the string
- Example:

  char codes[] = “sample”;



Array Initialization

ASTU

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      const int MAXELS = 5;
6      int i, max, nums[MAXELS] = {2, 18, 1, 27, 16};
7      max = nums[0];
8      for (i = 1; i < MAXELS; i++)
9          if (max < nums[i])
10             max = nums[i];
11     cout << "The maximum value is " << max << endl;
12     return 0;
13 }
```

```
C:\Users\Tinsae\Desktop\nm.exe
The maximum value is 27

Process returned 0 (0x0)   execution time : 0.023 s
Press any key to continue.
```

Two-Dimensional Arrays

ASTU

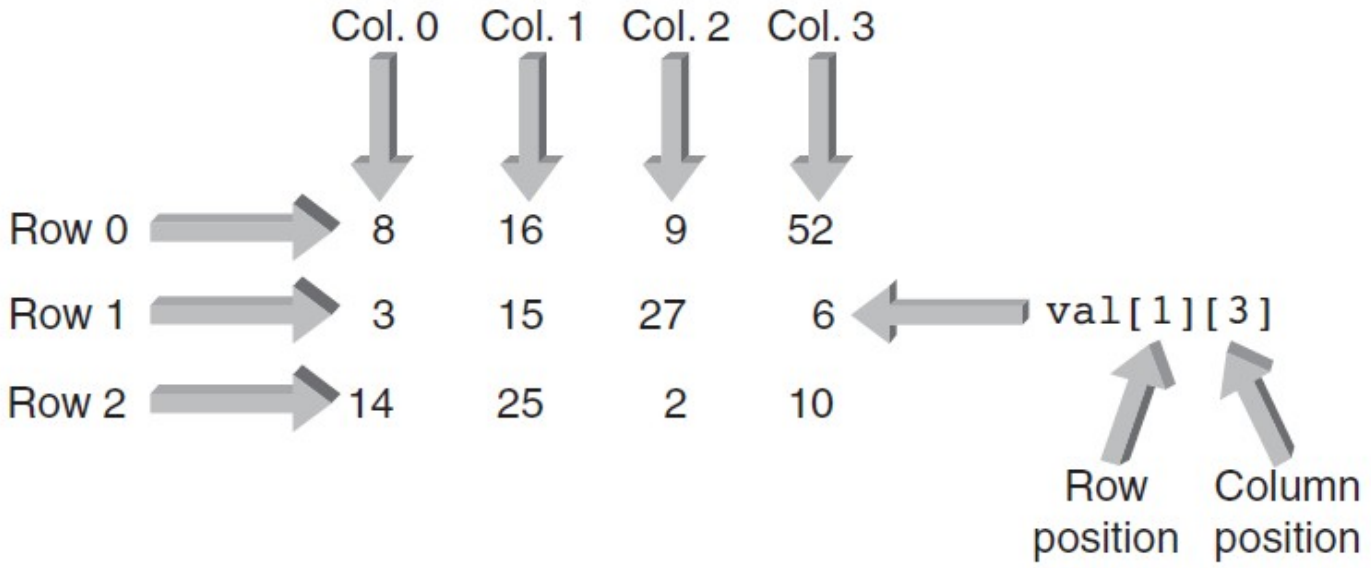
- Two-dimensional array: Has both rows a nd columns
- Also called a table
- Both dimensions must be specified in th e array declaration
- Row is specified first, then column
- Both dimensions must be specified when referencing an array element

Two-Dimensional Arrays

ASTU

- Example:

  int val[1][3];



Two-Dimensional Arrays

- Two-dimensional arrays can be initialized in the declaration by listing values within braces, separated by commas
- Braces can be used to distinguish rows, but are not required
- Nested for loops are used to process two -dimensional arrays
  - Outer loop controls the rows
  - Inner loop controls the columns

Two-Dimensional Arrays: Displaying

ASTU

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
     const int NUMROWS = 3;
     const int NUMCOLS = 4;
     int i, j;
     int val[NUMROWS][NUMCOLS] = {8,16,9,52,3,15,27,6,14,25,2,10};
     cout << "\nDisplay of val array by explicit element"<< endl
          << setw(4) << val[0][0] << setw(4) << val[0][1]
          << setw(4) << val[0][2] << setw(4) << val[0][3]<< endl
          << setw(4) << val[1][0] << setw(4) << val[1][1]
          << setw(4) << val[1][2] << setw(4) << val[1][3]<< endl
          << setw(4) << val[2][0] << setw(4) << val[2][1]
          << setw(4) << val[2][2] << setw(4) << val[2][3];
```

Two-Dimensional Arrays: Displaying

ASTU

```
17      cout << "\n\nDisplay of val array using a nested for loop";
18      for (i = 0; i < NUMROWS; i++)
19      {
20          cout << endl; // print a new line for each row
21          for (j = 0; j < NUMCOLS; j++)
22              cout << setw(4) << val[i][j];
23      }
24      cout << endl;
25      return 0;
26  }
27
```

```
C:\Users\Tinsae\Desktop\nm.exe

Display of val array by explicit element
   8  16   9  52
   3  15  27   6
  14  25   2  10

Display of val array using a nested for loop
   8  16   9  52
   3  15  27   6
  14  25   2  10

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Two-Dimensional Arrays: Multiplying Elements

ASTU

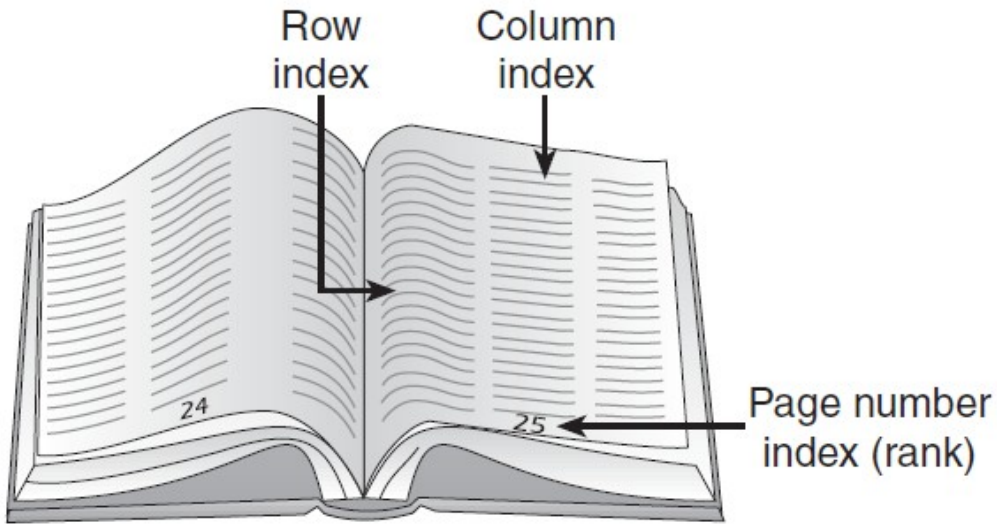
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    const int NUMROWS = 3;
    const int NUMCOLS = 4;
    int i, j;
    int val[NUMROWS][NUMCOLS] = {8,16,9,52,
                                 3,15,27,6,
                                 14,25,2,10
                                };
// Multiply each element by 10 and display it
    cout << "\nDisplay of multiplied elements";
    for (i = 0; i < NUMROWS; i++)
    {
        cout << endl; // start each row on a new line
        for (j = 0; j < NUMCOLS; j++)
        {
            val[i][j] = val[i][j] * 10;
            cout << setw(5) << val[i][j];
        } // end of inner loop
    } // end of outer loop
    cout << endl;
    return 0;
}
```

```
C:\Users\Tinsae\Desktop\nm.exe

Display of multiplied elements
   80  160   90  520
   30  150  270   60
  140  250   20  100

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Larger Dimensional Arrays

ASTU

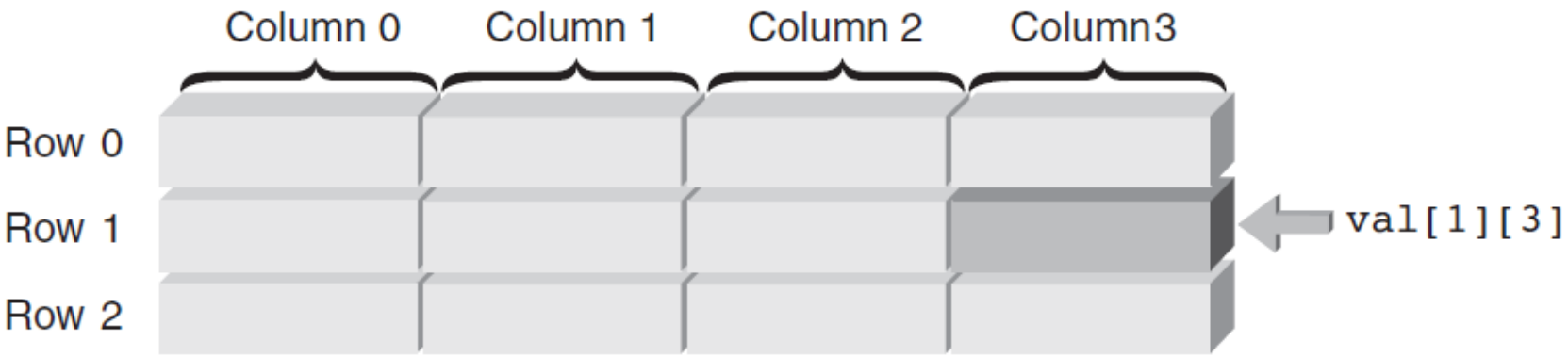
- Arrays with more than two dimensions c an be created, but are not commonly us ed
- Think of a three-dimensional array as a b ook of data tables



Arrays as Arguments

- An individual array element can be passed as a n argument just like any individual variable
- The called function receives a copy of the arra y element's value
- Passing an entire array to a function causes th e function to receive a reference to the array, not a copy of its element values
- The function must be declared with an array a s the argument
- Single element of array is obtained by adding an offset to the array's starting location

Arrays as Arguments



No. of bytes in a complete row

Offset = [(3 × 4) + [1 × (4 × 4)] = 28 bytes

- Bytes per integer
- Column specification
- Row index
- Column index

Arrays as Arguments: Finding Maximum

ASTU

```
1  #include <iostream>
2  using namespace std;
3  int findMax(int [], int); // function prototype
4  int main()
5  {
6      const int MAXELS = 5;
7      int nums[MAXELS] = {2, 18, 1, 27, 16};
8      cout << "The maximum value is "
9           << findMax(nums, MAXELS) << endl;
10     return 0;
11 }
12 // Find the maximum value
13 int findMax(int vals[], int numels)
14 {
15     int i, max = vals[0];
16     for (i = 1; i < numels; i++)
17         if (max < vals[i]) max = vals[i];
18     return max;
19 }
```

```
C:\Users\Tinsae\Desktop\nm.exe
The maximum value is 27

Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```

Arrays as Arguments: Using 2D arrays

ASTU

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  const int ROWS = 3;
5  const int COLS = 4;
6  void display(int [ROWS][COLS]); // function prototype
7  int main()
8  {
9      int val[ROWS][COLS] = {8,16,9,52,
10                            3,15,27,6,
11                            14,25,2,10
12                           };
13     display(val);
14     return 0;
15 }
16 void display(int nums[ROWS][COLS])
17 {
18     int rownum, colnum;
19     for (rownum = 0; rownum < ROWS; rownum++)
20     {
21         for (colnum = 0; colnum < COLS; colnum++)
22             cout << setw(4) << nums[rownum][colnum];
23         cout << endl;
24     }
25     return;
26 }
```

Case Study 1: Statistical Analysis

ASTU

- A program is to be developed that accep ts a list of a maximum of 100 numbers a s input, determines both the average an d standard deviation of the numbers, an d then displays the result.

Step 1: Analyze the Problem

- The statement of the problem indicates that
  - Two output values are required:
    - an average and a standard deviation.
  - The input item is a list of integer numbers.
  - The problem statement doesn't specify the list size
    - make the application's functions as general as possible, bot h functions will be designed to handle any size list passed t o them.
    - This design also requires passing the exact number of array elements to each function at the time of the function call.
    - This capability means each function must be capable of rec eiving at least two input items as parameters
      - an array of arbitrary size and an integer number corresponding to t he number of elements in the passed array.

Step 2: Develop a Solution

- The I/O specifications determined from the problem analysis imply that each function's parameter list must be capable of receiving at least two items
  - one parameter to accommodate the integer array
  - the second parameter to accept an integer.
- The first function returns the average of the numbers in the passed array
- The second function returns the standard deviation. These items are determined as follows:

Step 2: Develop a Solution

ASTU

Calculate the average by adding the grades and dividing by the number of grades added

Determine the standard deviation by:

- ***Subtracting the average from each grade (results in a set of new numbers, each called a deviation)***
- ***Squaring each deviation found in the previous step***
- ***Adding the squared deviations and dividing the sum by the number of deviations***

Step 3, 4: Code, Test and Correct the Solution

ASTU

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
double findAvg(int [], int); // function prototype
double stdDev(int [], int, double); // function prototype
int main()
{
    const int NUMELS = 10;
    int values[NUMELS] = {98, 82, 67, 54, 78, 83, 95, 76, 68, 63};
    double average, sDev;
    average = findAvg(values, NUMELS); // call the function
    sDev = stdDev(values, NUMELS, average);  // call the function
    cout << "The average of the numbers is "
         << setw(5) << setiosflags(ios::showpoint)
         << setprecision(4) << average << endl;
    cout << "The standard deviation of the numbers is "
         << setw(5) << setiosflags(ios::showpoint)
         << setprecision(4) << sDev << endl;
    return 0;
}
```

Step 3, 4: Code, Test and Correct the Solution

ASTU

```
22  double findAvg(int nums[], int numel)
23  {
24      int i;
25      double sumnums = 0.0;
26      for (i = 0; i < numel; i++) // calculate the sum of the grades
27          sumnums = sumnums + nums[i];
28      return (sumnums / numel);  // calculate and return the average
29  }
30  double stdDev(int nums[], int numel, double av)
31  {
32      int i;
33      double sumdevs = 0.0;
34      for (i=0; i<numel; i++)
35          sumdevs=sumdevs + pow((nums[i]-av),2);
36      return(sqrt(sumdevs/numel));
37  }
```

```
C:\Users\Tinsae\Desktop\nm.exe
The average of the numbers is 76.40
The standard deviation of the numbers is 13.15

Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```

Case Study 2: Students' Grading Program

ASTU

- A program that uses a two-dimensional a rray named grade to store and then displa y the grade records for a small class.
- The class has four students, and the recor ds include three quizzes.
  - Reads quiz scores for each student into the t wo-dimensional array grade
  - Compute the average score for each student
  - Compute the average score for each quiz.
  - Displays the quiz scores and the averages.

Case Study 2: Students' Grading Program

ASTU

```
#include <iostream>
#include <iomanip>
using namespace std;
const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;
void computeStAve(int [][NUMBER_QUIZZES], double []);
void computeQuizAve(int [][NUMBER_QUIZZES], double[]);
void display(int[][NUMBER_QUIZZES],double [], double[]);
int main( )
{
    int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
    double stAve[NUMBER_STUDENTS];
    double quizAve[NUMBER_QUIZZES];
    //fill the array
    for(int i=0; i<NUMBER_STUDENTS; i++)
    {
        cout<<"Enter Student "<<i+1<<" Scores"<<endl;
        for(int j=0; j<NUMBER_QUIZZES; j++)
        {
            cout<<"Quiz "<<j+1<<" ? ";
            cin>>grade[i][j];
        }
    }
    computeStAve(grade, stAve);
    computeQuizAve(grade, quizAve);
    display(grade, stAve, quizAve);
    return 0;
}
```

Case Study 2: Students' Grading Program

ASTU

```
28  void computeStAve(int grade[][NUMBER_QUIZZES], double stAve[])
29  {
30      for (int stNum = 1; stNum <= NUMBER_STUDENTS; stNum++)
31      {
32          //Process one stNum:
33          double sum = 0;
34          for(int quizNum = 1; quizNum <= NUMBER_QUIZZES; quizNum++)
35              sum = sum + grade[stNum-1][quizNum-1];
36          //sum contains the sum of the quiz scores for student number stNum.
37          stAve[stNum-1] = sum/NUMBER_QUIZZES;
38          //Average for student stNum is the value of stAve[stNum-1]
39      }
40  }
41  void computeQuizAve(int grade[][NUMBER_QUIZZES], double quizAve[])
42  {
43      for (int quizNum = 1; quizNum <= NUMBER_QUIZZES; quizNum++)
44      {
45          //Process one quiz (for all students):
46          double sum = 0;
47          for(int stNum = 1; stNum <= NUMBER_STUDENTS; stNum++)
48              sum = sum + grade[stNum-1][quizNum-1];
49          //sum contains the sum of all student scores on quiz number quizNum.
50          quizAve[quizNum-1] = sum/NUMBER_STUDENTS;
51          //Average for quiz quizNum is the value of quizAve[quizNum-1]
52      }
53  }
```

Case Study 2: Students' Grading Program

ASTU

```
54  void display(int grade[][NUMBER_QUIZZES],double stAve[], double quizAve[])
55  {
56      cout.setf(ios::fixed);
57      cout.setf(ios::showpoint);
58      cout.precision(1);
59      cout << setw(10) << "Student"
60           << setw(5) << "Ave"
61           << setw(15) << "Quizzes\n";
62      for (int stNum = 1; stNum <= NUMBER_STUDENTS; stNum++)
63      {
64          //Display for one stNum:
65          cout << setw(10) << stNum
66               << setw(5) << stAve[stNum-1] << " ";
67          for (int quizNum = 1; quizNum <= NUMBER_QUIZZES; quizNum++)
68              cout << setw(5) << grade[stNum-1][quizNum-1];
69          cout << endl;
70      }
71
72      cout << "Quiz averages = ";
73      for (int quizNum = 1; quizNum <= NUMBER_QUIZZES; quizNum++)
74          cout << setw(5) << quizAve[quizNum-1];
75      cout << endl;
76  }
```

Testing the Program

ASTU

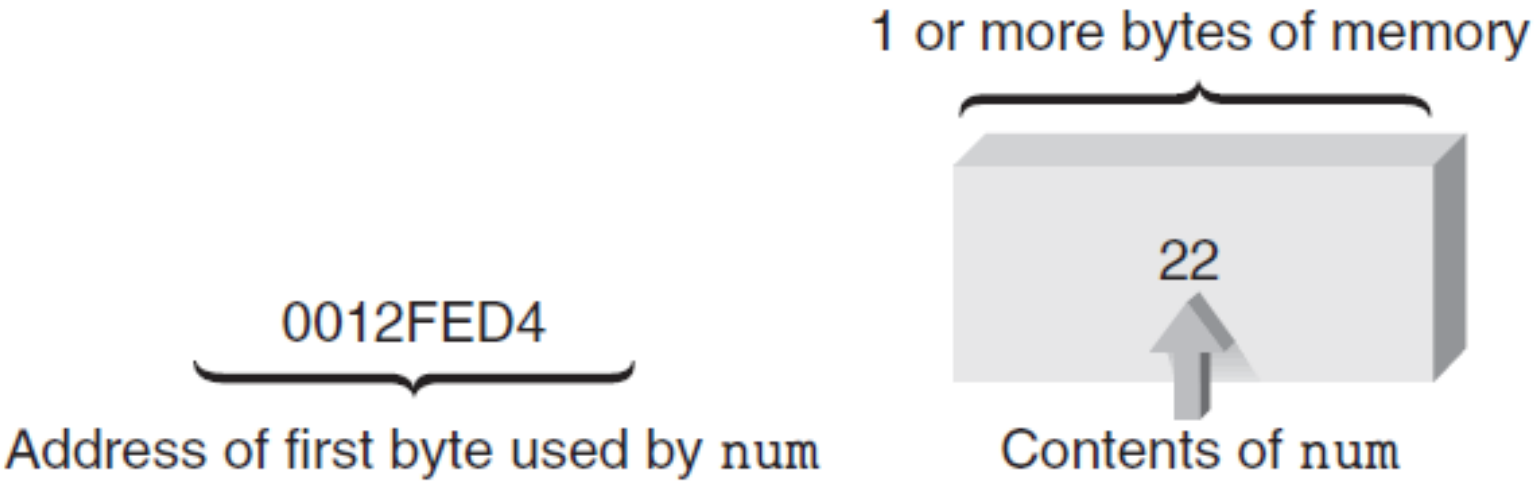
```
C:\Users\Tinsae\Documents\gradecasestudy.exe
Enter Student 1 Scores
Quiz 1 ? 9
Quiz 2 ? 9
Quiz 3 ? 7
Enter Student 2 Scores
Quiz 1 ? 2
Quiz 2 ? 9
Quiz 3 ? 7
Enter Student 3 Scores
Quiz 1 ? 10
Quiz 2 ? 10
Quiz 3 ? 9
Enter Student 4 Scores
Quiz 1 ? 8
Quiz 2 ? 9
Quiz 3 ? 9
   Student  Ave        Quizzes
         1  8.3      9     9     7
         2  6.0      2     9     7
         3  9.7     10    10     9
         4  8.7      8     9     9
Quiz averages =    7.2   9.2   8.0
```

Addresses and Pointers

- The address operator, &, accesses a variable's address in memory
- The address operator placed in front of a variable's name refers to the address of the variable

  `&num` means the address of `num`

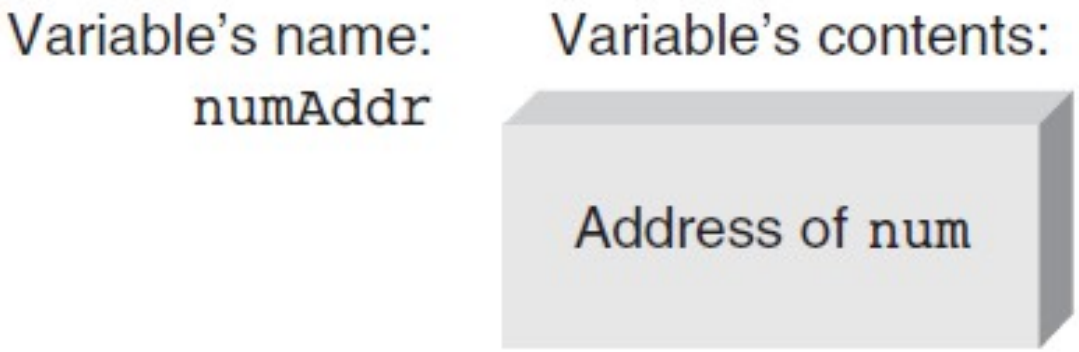
Addresses and Pointers



Storing Addresses

ASTU

- Addresses can be stored in a suitably dec lared variable



Storing Addresses

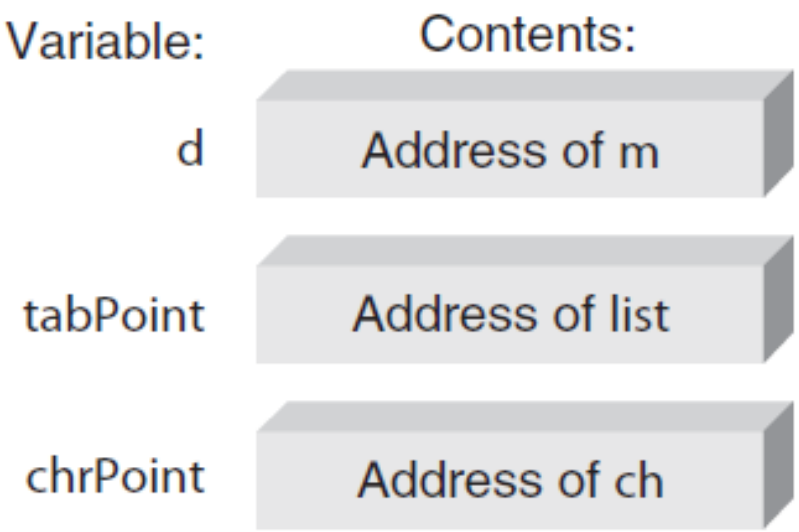
ASTU

- Example statements store addresses of t he variable `m`, `list`, and `ch` in the variab les `d`, `tabPoint`, and `chrPoint`

```
d = &m;
tabPoint = &list;
chrPoint = &ch;
```

- `d`, `tabPoint`, and `chrPoint` are called pointer variables or pointers

Storing Addresses



Using Addresses

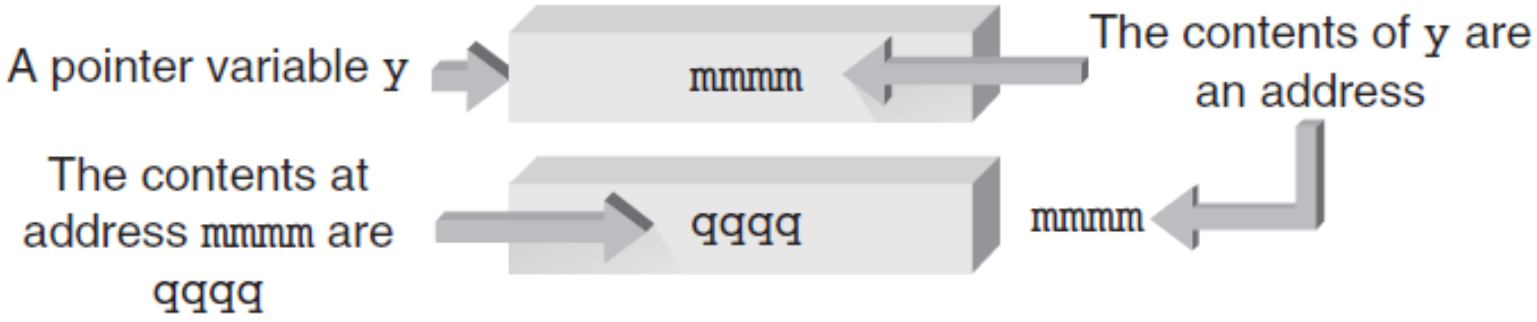
ASTU

- To use a stored address, C++ provides th e **indirection operator**, *
- The * symbol, when followed by a point er, means “the variable whose address is stored in”

  `*numAddr` means the variable whose addres s is stored in `numAddr`

Using Addresses

ASTU



Using Addresses

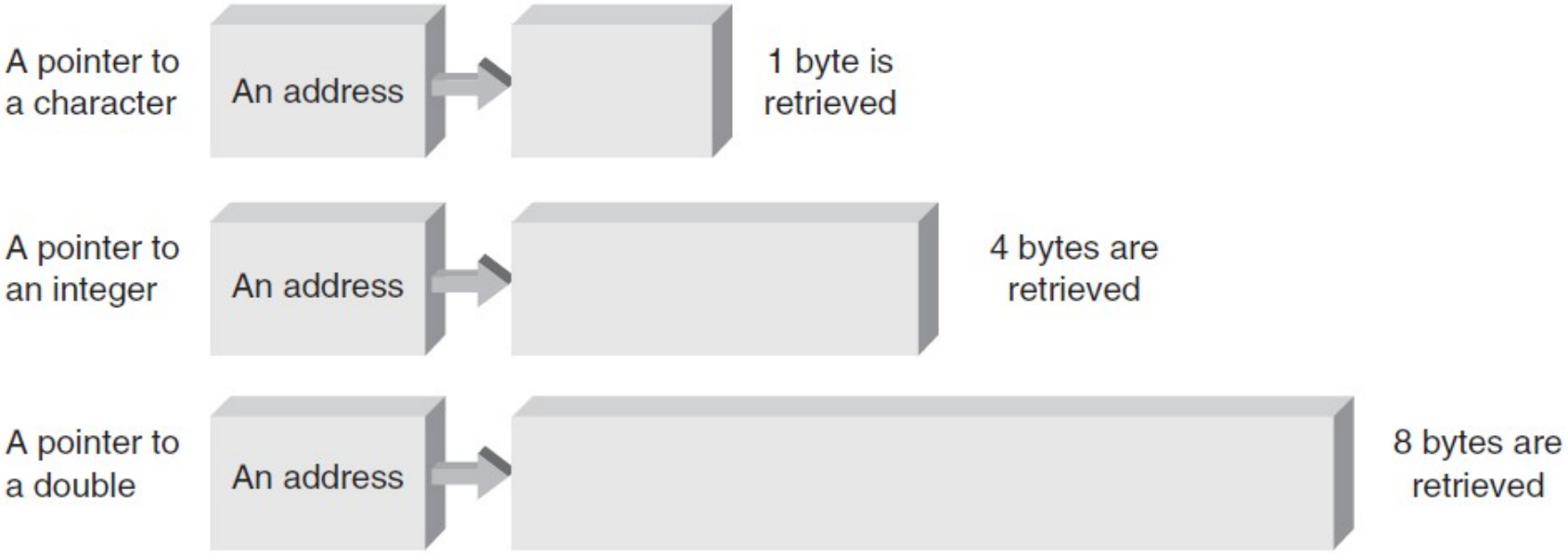
- When using a pointer variable, the value that is finally obtained is always found b y first going to the pointer for an addres s
- The address contained in the pointer is t hen used to get the variable's contents
- Since this is an indirect way of getting to the final value, the term **indirect addressi ng** is used to describe it

Declaring Pointers

- Like all variables, pointers must be declared before they can be used to store an address
- When declaring a pointer variable, C++ req uires specifying the type of the variable that is pointed to
  - Example: `int *numAddr;`
- To understand pointer declarations, reading them “backward” is helpful
  - Start with the indirection operator, *, and transl ate it as “the variable whose address is stored in ” or “the variable pointed to by”

Declaring Pointers

ASTU



Pointers: Example

ASTU

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int *numAddr; // declare a pointer to an int
6      int miles, dist; // declare two integer variables
7      dist = 158; // store the number 158 in dist
8      miles = 22; // store the number 22 in miles
9      numAddr = &miles;  // store the address of miles in numAddr
10     cout << "The address stored in numAddr is " << numAddr << endl;
11     cout << "The value pointed to by numAddr is " << *numAddr << "\n\n";
12     numAddr = &dist; // now store the address of dist in numAddr
13     cout << "The address now stored in numAddr is " << numAddr << endl;
14     cout << "The value now pointed to by numAddr is " << *numAddr << endl;
15     return 0;
16 }
17
```

```
C:\Users\Tinsae\Documents\bs.exe
The address stored in numAddr is 0x22fef8
The value pointed to by numAddr is 22

The address now stored in numAddr is 0x22fef4
The value now pointed to by numAddr is 158

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

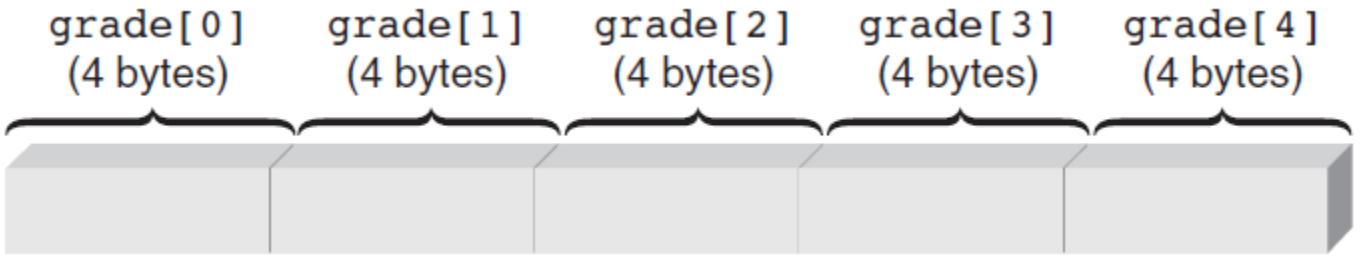
References and Pointers

- A **reference** is a named constant for an address
  - The address named as a constant cannot be changed
- A pointer variable's value address can be change d
- For most applications, using references rather th an pointers as arguments to functions is preferre d
  - Simpler notation for locating a reference parameter
  - Eliminates address (&) and indirection operator (*) req uired for pointers
- References are **automatically dereferenced**, also c alled **implicitly dereferenced**

Array Names as Pointers

ASTU

- There is a direct and simple relationship between array names and pointers



- Using subscripts, the fourth element in grade is referred to as grade[3], address calculated as:

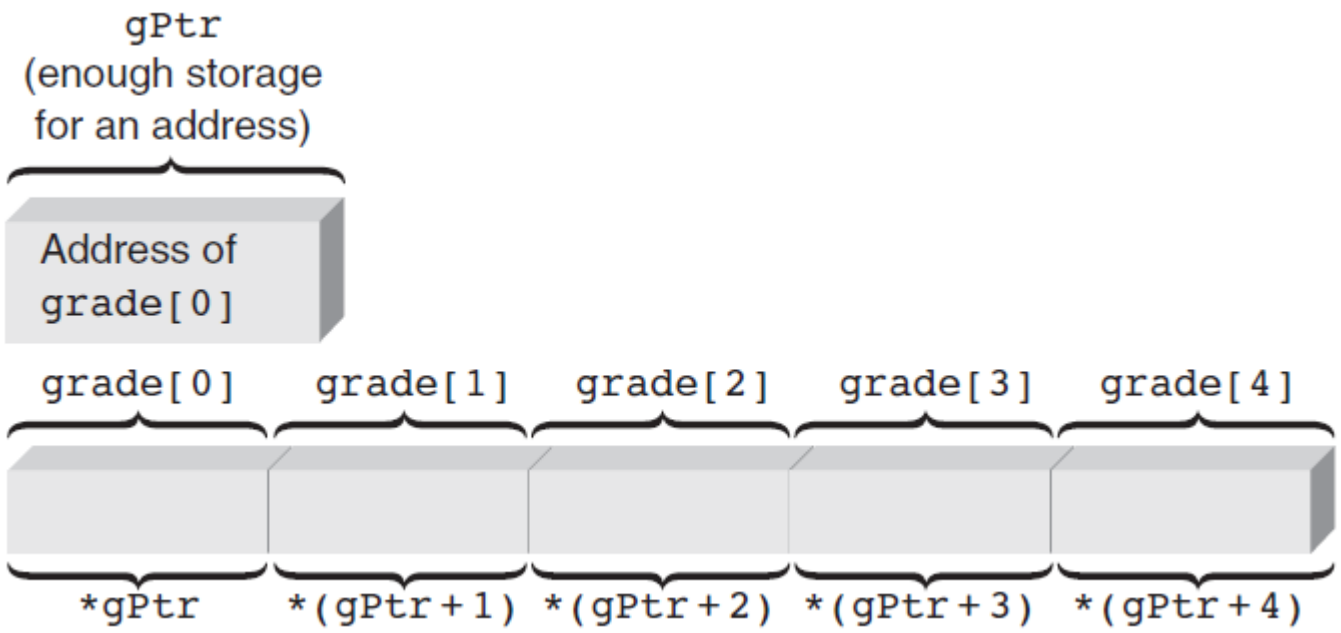
```
&grade[3] = &grade[0] + (3 * sizeof(int))
```

Array Names as Pointers

Array Element	Subscript Notation	Pointer Notation
Element 0	`grade[0]`	`*gPtr` or `(gPtr + 0)`
Element 1	`grade[1]`	`*(gPtr + 1)`
Element 2	`grade[2]`	`*(gPtr + 2)`
Element 3	`grade[3]`	`*(gPtr + 3)`
Element 4	`grade[4]`	`*(gPtr + 4)`

Array Names as Pointers

ASTU



Array Names as Pointers: Example

ASTU

```
1  #include <iostream>
2  using namespace std;
3  void withArray();
4  void arrayWithPointer();
5  int main(){
6      withArray();
7      arrayWithPointer();
8      return 0;
9  }
10 void withArray()
11 {
12     const int ARRAYSIZE = 5;
13     int i, grade[ARRAYSIZE] = {98, 87, 92, 79, 85};
14     for (i = 0; i < ARRAYSIZE; i++)
15         cout << "\nElement " << i << " is " << grade[i];
16     cout << endl;
17 }
18 void arrayWithPointer()
19 {
20     const int ARRAYSIZE = 5;
21     int *gPtr; // declare a pointer to an int
22     int i, grade[ARRAYSIZE] = {98, 87, 92, 79, 85};
23     gPtr = &grade[0];   // store the starting array address
24     for (i = 0; i < ARRAYSIZE; i++)
25         cout << "\nElement " << i << " is " << *(gPtr + i);
26     cout << endl;
27 }
```

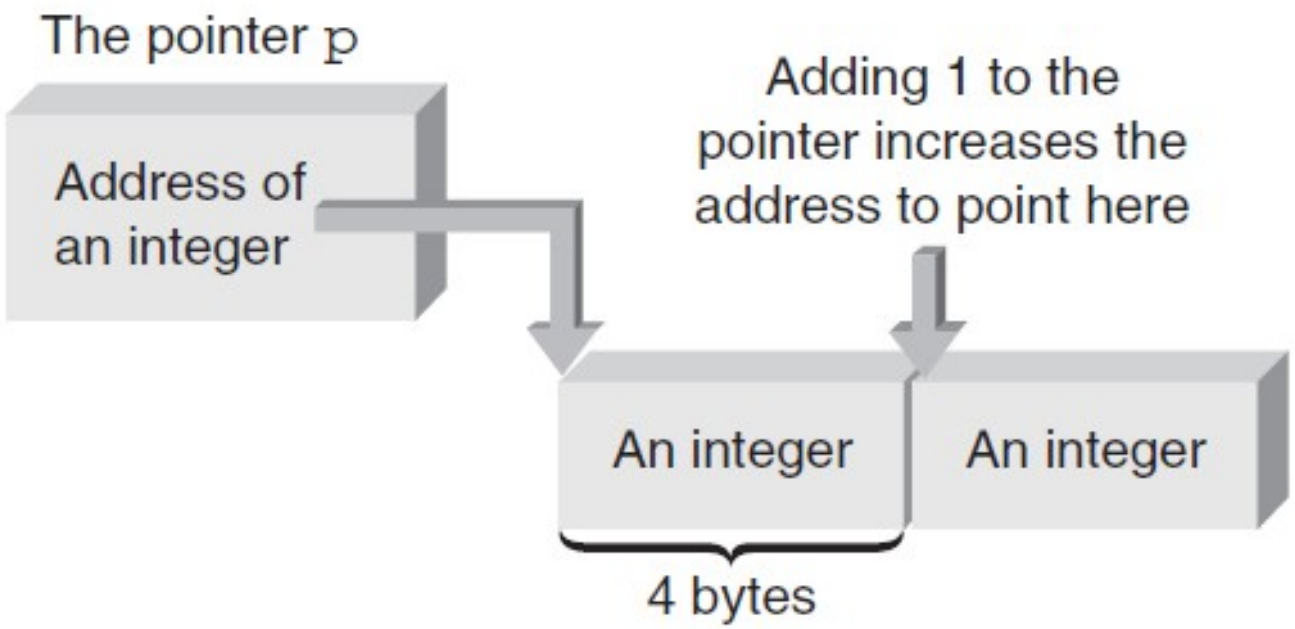
```
C:\Users\Tinsae\Documents\bs.exe

Element 0 is 98
Element 1 is 87
Element 2 is 92
Element 3 is 79
Element 4 is 85

Element 0 is 98
Element 1 is 87
Element 2 is 92
Element 3 is 79
Element 4 is 85
```

Pointer Arithmetic

- Pointer variables, like all variables, contai n values
- The value stored in a pointer is a memor y address
- By adding or subtracting numbers to poi nters you can obtain different addresses
- Pointer values can be compared using rel ational operators (==, <, >, etc.)

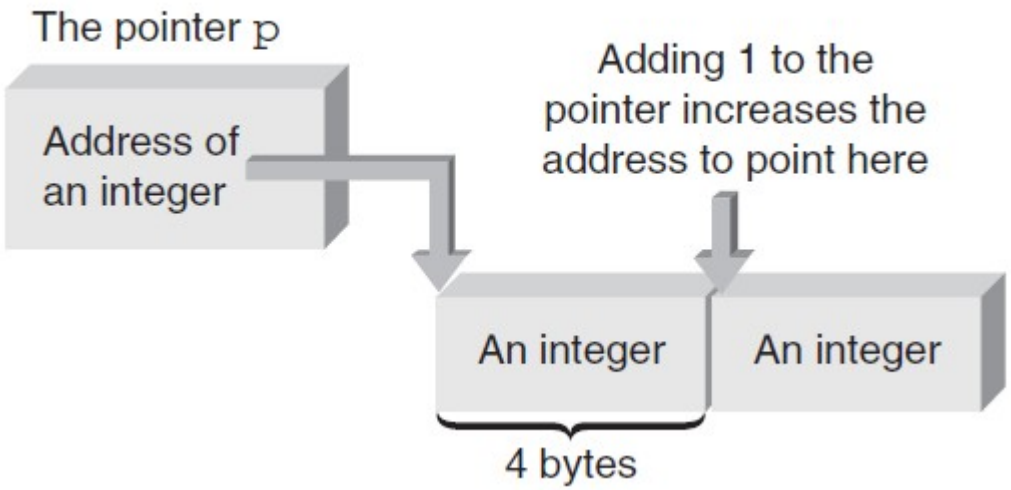
Pointer Arithmetic

ASTU



Pointer Arithmetic

ASTU



Pointer Arithmetic: Example

ASTU

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      const int NUMS = 5;
6      int nums[NUMS] = {16, 54, 7, 43, -5};
7      int total = 0, *nPt;
8      nPt = nums; // store address of nums[0] in nPt
9      while (nPt < nums + NUMS)
10         total += *nPt++;
11     cout << "The total of the array elements is " << total << endl;
12     return 0;
13 }
```

```
C:\Users\Tinsae\Documents\a.exe
The total of the array elements is 115

Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```

Pointer Initialization

ASTU

- Increment and decrement operators can be applied as both prefix and postfix op erators

```
*ptNum++     // use the pointer and then increment it
*++ptNum     // increment the pointer before using it
*ptNum--     // use the pointer and then decrement it
*--ptNum     // decrement the pointer before using it
```

- Of the four possible forms, `*ptNum++` is most common
  - Allows accessing each array element as the address is “marched along” from starting address to address of last array element

Pointer Initialization

- Pointers can be initialized with they are d eclared

```
int *ptNum = &miles;
```

- Pointers to arrays can also be initialized when they are declared

```
double *zing = &volts[0];
```

Passing Addresses

- Reference pointers can be used to pass addresses through reference parameters
  - Implied use of an address
- Pointers can be used explicitly to pass addresses with references
  - Explicitly passing references with the address operator is called pass by reference
  - Called function can reference, or access, variables in the calling function by using the passed addresses

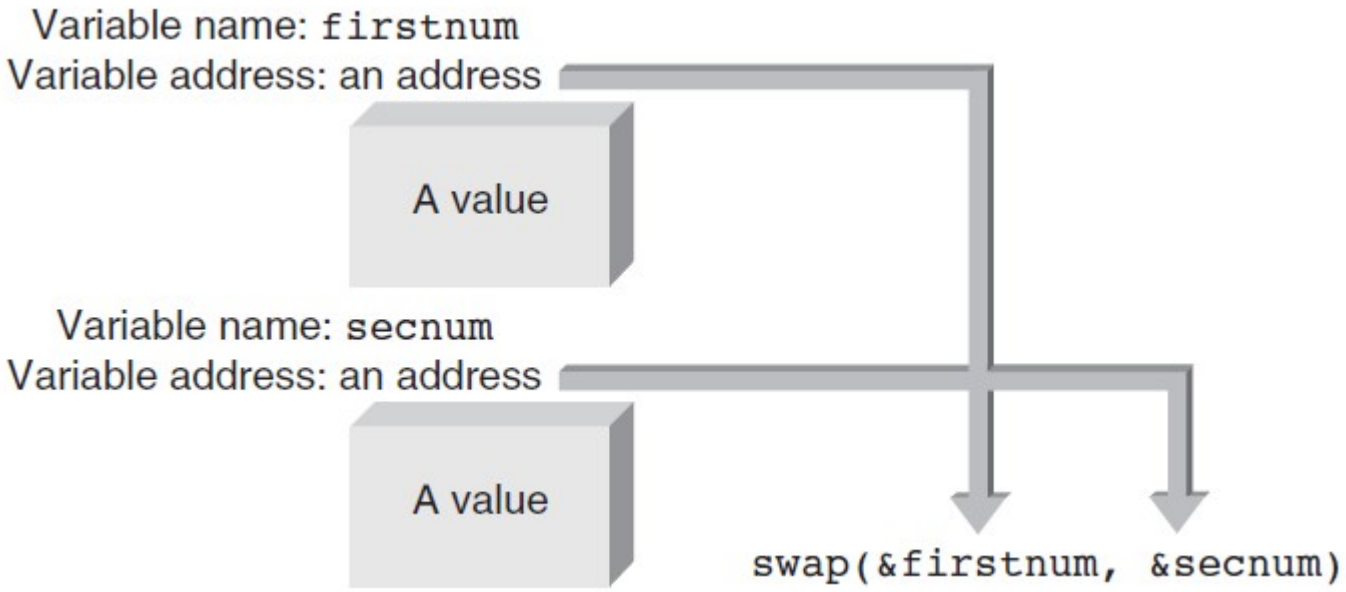
Passing Addresses: Example

ASTU

```
1  #include <iostream>
2  using namespace std;
3  void swap(double *, double *); // function prototype
4  int main()
5  {
6      double firstnum = 20.5, secnum = 6.25;
7      cout << "The value stored in firstnum is: " << firstnum << endl;
8      cout << "The value stored in secnum is: " << secnum << "\n\n";
9      swap(&firstnum, &secnum); // call swap
10     cout << "The value stored in firstnum is now: "
11          << firstnum << endl;
12     cout << "The value stored in secnum is now: "
13          << secnum << endl;
14     return 0;
15 }
16 // This function swaps the values in its two arguments
17 void swap(double *nm1Addr, double *nm2Addr)
18 {
19     double temp;
20     temp = *nm1Addr; // save firstnum's value
21     *nm1Addr = *nm2Addr;  // move secnum's value into firstnum
22     *nm2Addr = temp; // change secnum's value
23     return;
24 }
25
```

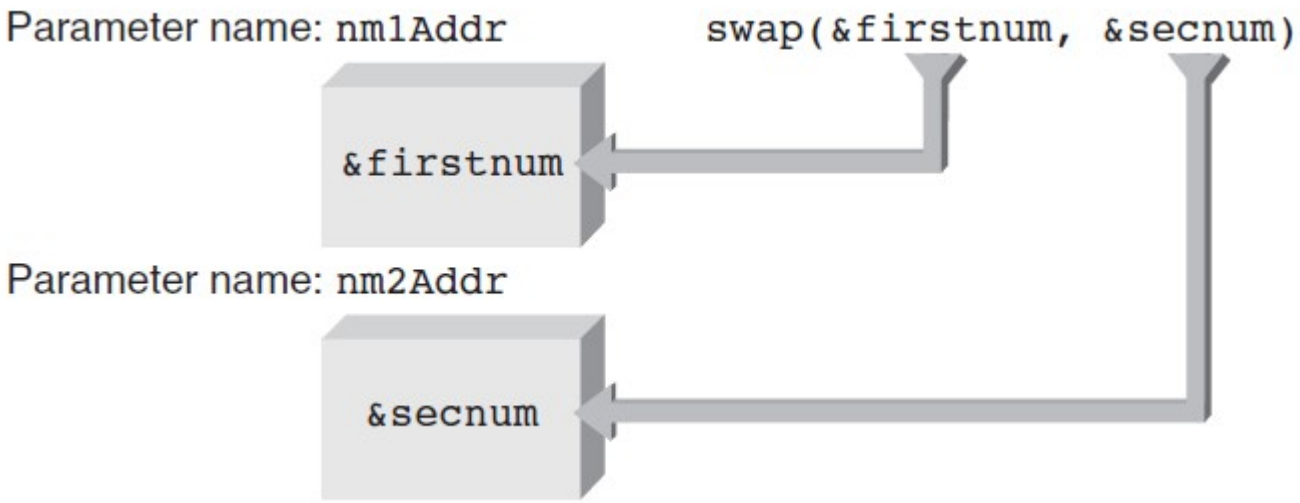
Passing Addresses

ASTU



Passing Addresses

ASTU



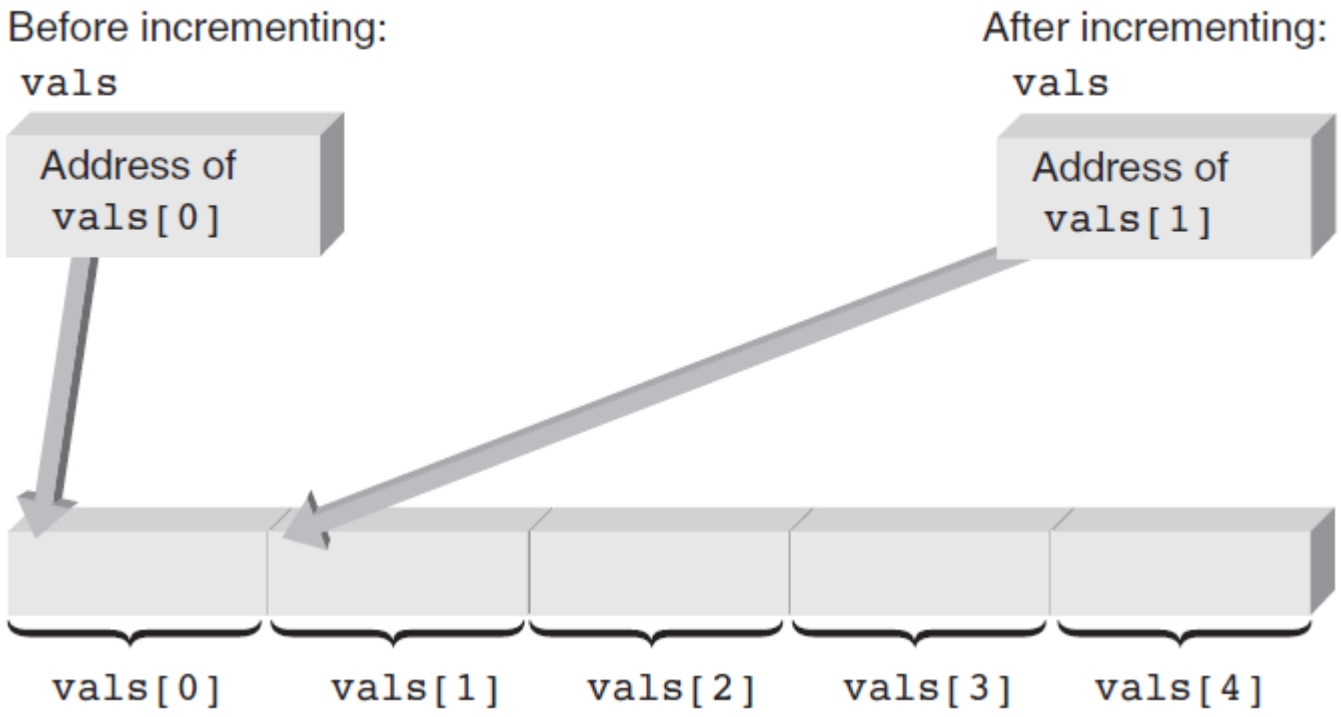
Passing Arrays

ASTU

- When an array is passed to a function, it s address is the only item actually passed
  - “Address” means the address of the first loca tion used to store the array
  - First location is always element zero of the a rray

Passing Arrays

ASTU



Passing Arrays: Example

ASTU

```
1  #include <iostream>
2  using namespace std;
3  int findMax(int [], int); // function prototype
4  int main()
5  {
6      const int NUMPTS = 5;
7      int nums[NUMPTS] = {2, 18, 1, 27, 16};
8      cout << "\nThe maximum value is "
9           << findMax(nums,NUMPTS) << endl;
10     return 0;
11 }
12 // This function returns the maximum value in an array of ints
13 int findMax(int vals[], int numels)
14 {
15     int i, max = vals[0];
16     for (i = 1; i < numels; i++)
17         if (max < vals[i])
18             max = vals[i];
19     return max;
20 }
```