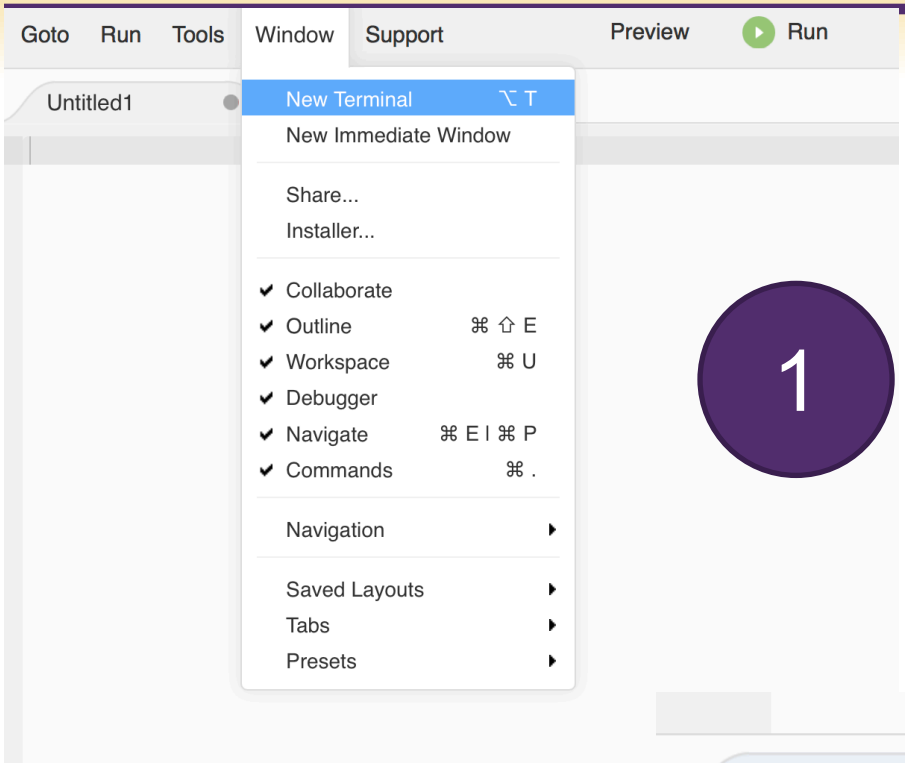
C Programming for Engineers

Arrays

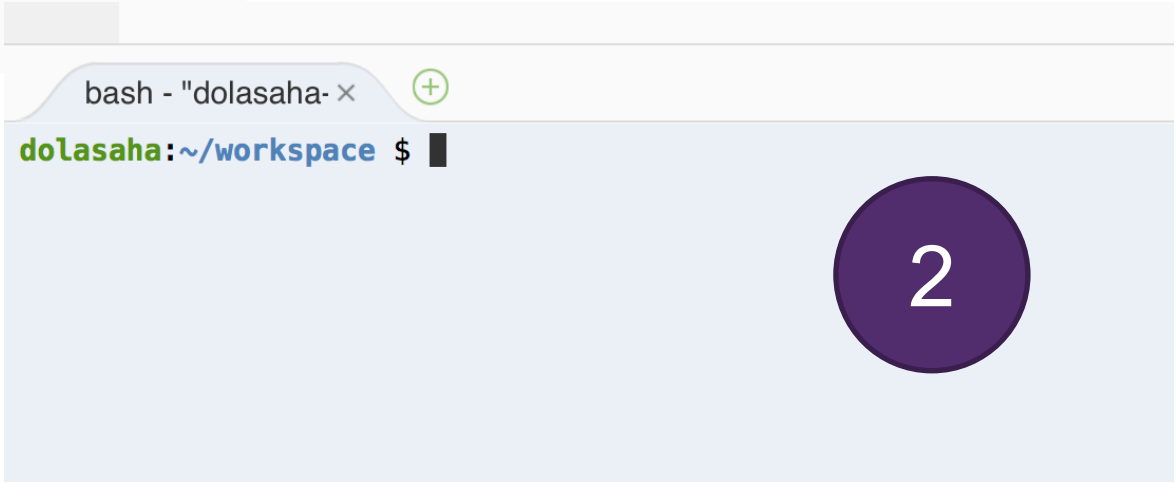
ICEN 360– Spring 2017

Prof. Dola Saha

Compiling your own code



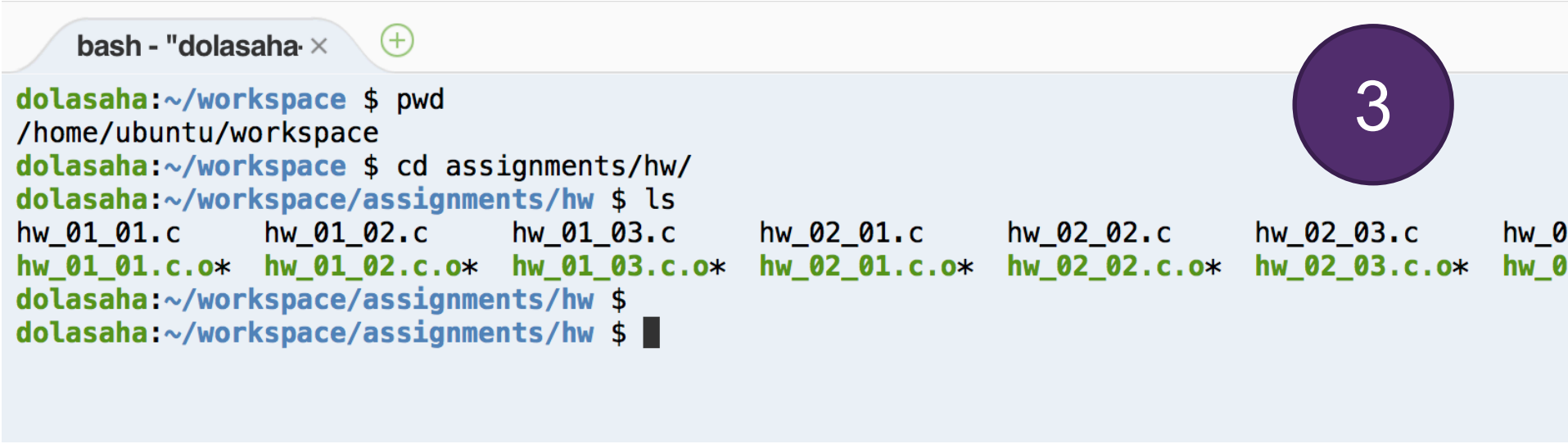
1



2

Compiling your own code

- `pwd` – print work directory
- `cd directory_name` – change directory
- `ls` – list the content of current directory



Linking with Math Library

- `gcc –o object_filename c_file.c –lm`
  - `-l` link to the library
  - `-lm` is specific for math
- Run the object file
  - `./object_filename`

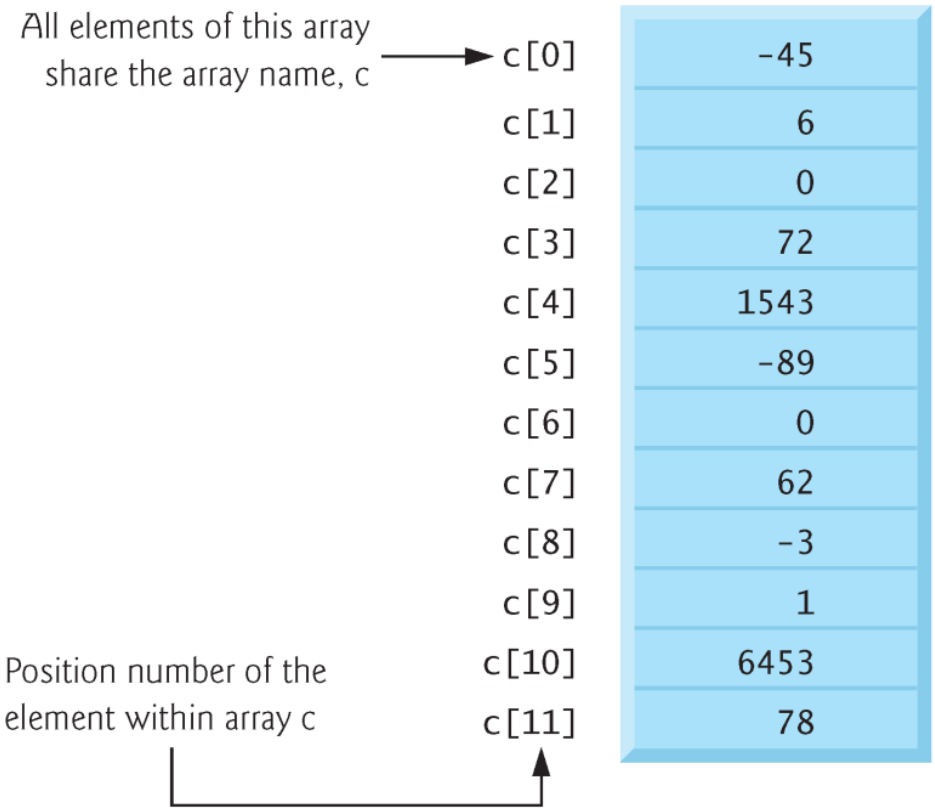
```
dolasaha:~/workspace/assignments/hw $ gcc -o convertCoordinate hw_03_01.c -lm
dolasaha:~/workspace/assignments/hw $ ls
convertCoordinate*  hw_01_01.c.o*  hw_01_02.c.o*  hw_01_03.c     hw_02_01.c     hw_02_02.c     hw_02_03.c
hw_01_01.c          hw_01_02.c     hw_01_03*      hw_01_03.c.o*  hw_02_01.c.o*  hw_02_02.c.o*  hw_02_03.c.o*
dolasaha:~/workspace/assignments/hw $
dolasaha:~/workspace/assignments/hw $ ./convertCoordinate
Enter P for Polar coordinate or C for Cartesian Coordinate: c
Enter Cartesian coordinate (x,y) with space: 5 5
The Polar Coordinate for (x=5.000000, y=5.000000) is r=7.071068, theta=45.000000 degrees
dolasaha:~/workspace/assignments/hw $
```

4

Array

- Arrays are data structures consisting of related data items of the same type.
- A group of *contiguous* memory locations that all have the *same type*.
- To refer to a particular location or element in the array
  - Array's name
  - Position number of the particular element in the array

Example Array



Array indexing

- The first element in every array is the zeroth element.
- An array name, like other identifiers, can contain only letters, digits and underscores and cannot begin with a digit.
- The position number within square brackets is called an index or subscript.
- An index must be an integer or an integer expression
  - array_name[x], array_name[x+y], etc.
- For example, if `a = 5` and `b = 6`, then the statement
  - `c[a + b] += 2;`

    adds 2 to array element `c[11]`.

Array in memory

- Array occupies contiguous space in memory
- The following definition reserves 12 elements for integer array c, which has indices in the range 0-11.
  - `int c[12];`
- The definition
  - `int b[100]; double x[27];`

  reserves 100 elements for integer array b and 27 elements for double array x.
- Like any other variables, uninitialized array elements contain garbage values.

Initializing array

```
1  // Fig. 6.3: fig06_03.c
2  // Initializing the elements of an array to zeros.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     int n[5]; // n is an array of five integers
9
10    // set elements of array n to 0
11    for (size_t i = 0; i < 5; ++i) {
12       n[i] = 0; // set element at location i to 0
13    }
14
15    printf("%s%13s\n", "Element", "Value");
16
17    // output contents of array n in tabular format
18    for (size_t i = 0; i < 5; ++i) {
19       printf("%7u%13d\n", i, n[i]);
20    }
21 }
```

Output

```
Element          Value
      0              0
      1              0
      2              0
      3              0
      4              0
```

Use of `size_t`

- Notice that the variable `i` is declared to be of type `size_t`, which according to the C standard represents an unsigned integral type.
- This type is recommended for any variable that represents an array's size or an array's indices.
- Type `size_t` is defined in header `<stddef.h>`, which is often included by other headers (such as `<stdio.h>`).
- [Note: If you attempt to compile Fig. 6.3 and receive errors, simply include `<stddef.h>` in your program.]

Initializing with initializer list

```
1  // Fig. 6.4: fig06_04.c
2  // Initializing the elements of an array with an initializer list.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     // use initializer list to initialize array n
9     int n[5] = {32, 27, 64, 18, 95};
10
11    printf("%s%13s\n", "Element", "Value");
12
13    // output contents of array in tabular format
14    for (size_t i = 0; i < 5; ++i) {
15       printf("%7u%13d\n", i, n[i]);
16    }
17 }
```

Output

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
```

Initializing with fewer initializers

- If there are *fewer* initializers than elements in the array, the remaining elements are initialized to zero.
- Example:

```
// initializes entire array to zeros
int n[10] = {0};
```

- The array definition
  - `int n[5] = {32, 27, 64, 18, 95, 14};`

  causes a syntax error because there are six initializers and *only* five array elements.

Initializing without array size

- If the array size is *omitted* from a definition with an initializer list, the number of elements in the array will be the number of elements in the initializer list.
- For example,
  - `int n[] = {1, 2, 3, 4, 5};`

  would create a five-element array initialized with the indicated values.

Initializing to even list

```
1  // Fig. 6.5: fig06_05.c
2  // Initializing the elements of array s to the even integers from 2 to 10.
3  #include <stdio.h>
4  #define SIZE 5 // maximum size of array
5
6  // function main begins program execution
7  int main(void)
8  {
9     // symbolic constant SIZE can be used to specify array size
10    int s[SIZE]; // array s has SIZE elements
11
12    for (size_t j = 0; j < SIZE; ++j) { // set the values
13       s[j] = 2 + 2 * j;
14    }
15
16    printf("%s%13s\n", "Element", "Value");
17
18    // output contents of array s in tabular format
19    for (size_t j = 0; j < SIZE; ++j) {
20       printf("%7u%13d\n", j, s[j]);
21    }
22 }
```

Output

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
```

Preprocessor

- The `#define` preprocessor directive is introduced in this program.
- **`#define SIZE 5`**
  - defines a symbolic constant `SIZE` whose value is `5`.
- A symbolic constant is an identifier that's replaced with replacement text by the C preprocessor before the program is compiled.
- Using symbolic constants to specify array sizes makes programs more modifiable.

Common Programming Error 6.3

Ending a `#define` *or* `#include` *preprocessor directive with a semicolon. Remember that preprocessor directives are* not *C statements.*

Adding elements of an array

```
1  // Fig. 6.6: fig06_06.c
2  // Computing the sum of the elements of an array.
3  #include <stdio.h>
4  #define SIZE 12
5
6  // function main begins program execution
7  int main(void)
8  {
9     // use an initializer list to initialize the array
10    int a[SIZE] = {1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45};
11    int total = 0; // sum of array
12
13    // sum contents of array a
14    for (size_t i = 0; i < SIZE; ++i) {
15       total += a[i];
16    }
17
18    printf("Total of array element values is %d\n", total);
19 }
```

```
Total of array element values is 383
```

Classwork Assignment

- Initialize an array of size with an initializer list and find the maximum element.

Using Arrays to Summarize Poll (1)

```
1  // Fig. 6.7: fig06_07.c
2  // Analyzing a student poll.
3  #include <stdio.h>
4  #define RESPONSES_SIZE 40 // define array sizes
5  #define FREQUENCY_SIZE 11
6
7  // function main begins program execution
8  int main(void)
9  {
10    // initialize frequency counters to 0
11    int frequency[FREQUENCY_SIZE] = {0};
12
13    // place the survey responses in the responses array
14    int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
15       1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
16       5, 6, 7, 5, 6, 4, 8, 6, 8, 10};
17
18    // for each answer, select value of an element of array responses
19    // and use that value as an index in array frequency to
20    // determine element to increment
21    for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
22       ++frequency[responses[answer]];
23    }
24
```

Using Arrays to Summarize Poll (2)

```
25      // display results
26      printf("%s%17s\n", "Rating", "Frequency");
27
28      // output the frequencies in a tabular format
29      for (size_t rating = 1; rating < FREQUENCY_SIZE; ++rating) {
30         printf("%6d%17d\n", rating, frequency[rating]);
31      }
32   }
```

```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```

Histogram with Array elements (1)

```
1  // Fig. 6.8: fig06_08.c
2  // Displaying a histogram.
3  #include <stdio.h>
4  #define SIZE 5
5
6  // function main begins program execution
7  int main(void)
8  {
9     // use initializer list to initialize array n
10    int n[SIZE] = {19, 3, 15, 7, 11};
11
12    printf("%s%13s%17s\n", "Element", "Value", "Histogram");
13
14    // for each element of array n, output a bar of the histogram
15    for (size_t i = 0; i < SIZE; ++i) {
16       printf("%7u%13d          ", i, n[i]);
17
18       for (int j = 1; j <= n[i]; ++j) { // print one bar
19          printf("%c", '*');
20       }
21
22       puts(""); // end a histogram bar with a newline
23    }
24 }
```

Histogram with Array elements (1)

```
Element         Value       Histogram
      0            19       ********************
      1             3       ***
      2            15       ***************
      3             7       *******
      4            11       ***********
```

Character Arrays & String Representation

- Store *strings* in character arrays.
- So far, the only string-processing capability we have is outputting a string with `printf`.
- A string such as `"hello"` is really an array of individual characters in C.
- A character array can be initialized using a string literal.
- For example,
  - `char string1[] = "first";`

  initializes the elements of array `string1` to the individual characters in the string literal `"first"`.

Size of Character Array

- In this case, the size of array `string1` is determined by the compiler based on the length of the string.
- The string `"first"` contains five characters *plus* a special *string-termination character* called the null character.
- Thus, array `string1` actually contains six elements.
- The character constant representing the null character is `'\0'`.
- All strings in C end with this character.

Character Array Indexing

- The preceding definition is equivalent to
  - ```
    char string1[] =
        {'f', 'i', 'r', 's', 't', '\0'};
    ```
- Because a string is really an array of characters, we can access individual characters in a string directly using array index notation.
- For example, `string1[0]` is the character `'f'` and `string1[3]` is the character `'s'`.
    ```

Scanning string

- We also can input a string directly into a character array from the keyboard using scanf and the conversion specifier `%s`.
- For example,
  - `char string2[20];`

  creates a character array capable of storing a string of *at most 19 characters* and a *terminating null character*.
- The statement
  - `scanf("%19s", string2);`

  reads a string from the keyboard into `string2`.
- The name of the array is passed to `scanf` without the preceding & used with nonstring variables.
- The & is normally used to provide `scanf` with a variable's *location* in memory so that a value can be stored there.

Scanning string

- Function `scanf` will read characters until a *space*, *tab*, *newline* or *end-of-file indicator* is encountered.
- The `string2` should be no longer than 19 characters to leave room for the terminating null character.
- If the user types 20 or more characters, your program may crash or create a security vulerability.
- For this reason, we used the conversion specifier `%19s` so that `scanf` reads a maximum of 19 characters and does not write characters into memory beyond the end of the array `string2`.

Memory Management in Scanning String

- It's your responsibility to ensure that the array into which the string is read is capable of holding any string that the user types at the keyboard.
- Function `scanf` does *not* check how large the array is.
- Thus, `scanf` can write beyond the end of the array.
- You can use `gets(text)` to get the text from user.

Printing String

- A character array representing a string can be output with `printf` and the `%s` conversion specifier.
-
- The array `string2` is printed with the statement
  - `printf("%s\n", string2);`
- Function `printf`, like `scanf`, does not check how large the character array is.
- The characters of the string are printed until a terminating null character is encountered.

Treating Character Arrays as String (1)

```
1  // Fig. 6.10: fig06_10.c
2  // Treating character arrays as strings.
3  #include <stdio.h>
4  #define SIZE 20
5
6  // function main begins program execution
7  int main(void)
8  {
9     char string1[SIZE]; // reserves 20 characters
10    char string2[] = "string literal"; // reserves 15 characters
11
12    // read string from user into array string1
13    printf("%s", "Enter a string (no longer than 19 characters): ");
14    scanf("%19s", string1); // input no more than 19 characters
15
16    // output strings
17    printf("string1 is: %s\nstring2 is: %s\n"
18           "string1 with spaces between characters is:\n",
19           string1, string2);
```

Treating Character Arrays as String (2)

```
20
21      // output characters until null character is reached
22      for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
23         printf("%c ", string1[i]);
24      }
25
26      puts("");
27   }
```

```
Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

Self Review Assignment

- String Comparison: Write a program to get `string1` and `string2` from user. Then compare each element iteratively to find if they are same or different. Finally, display if the two strings matched or not.

Passing Arrays to Functions

- To pass an array argument to a function, specify the array's name without any brackets.
- For example,

```
int hourlyTemperatures[HOURS_IN_A_DAY];
modifyArray(hourlyTemperatures, HOURS_IN_A_DAY);
```

  the function call passes array `hourlyTemperatures` and its size to function `modifyArray`.
- The name of the array evaluates to the address of the first element of the array.
- The called function *can modify* the element values in the callers' original arrays.

Passing Array to Functions (1)

```c
// Fig. 6.13: fig06_13.c
// Passing arrays and individual array elements to functions.
#include <stdio.h>
#define SIZE 5

// function prototypes
void modifyArray(int b[], size_t size);
void modifyElement(int e);

// function main begins program execution
int main(void)
{
   int a[SIZE] = {0, 1, 2, 3, 4}; // initialize array a

   puts("Effects of passing entire array by reference:\n\nThe "
      "values of the original array are:");

   // output original array
   for (size_t i = 0; i < SIZE; ++i) {
      printf("%3d", a[i]);
   }

   puts(""); // outputs a newline

```

Passing Array to Functions (2)

```
25      modifyArray(a, SIZE); // pass array a to modifyArray by reference
26      puts("The values of the modified array are:");
27
28      // output modified array
29      for (size_t i = 0; i < SIZE; ++i) {
30         printf("%3d", a[i]);
31      }
32
33      // output value of a[3]
34      printf("\n\n\nEffects of passing array element "
35         "by value:\n\nThe value of a[3] is %d\n", a[3]);
36
37      modifyElement(a[3]); // pass array element a[3] by value
38
39      // output value of a[3]
40      printf("The value of a[3] is %d\n", a[3]);
41   }
42
```

Passing Array to Functions (3)

```
43  // in function modifyArray, "b" points to the original array "a"
44  // in memory
45  void modifyArray(int b[], size_t size)
46  {
47     // multiply each array element by 2
48     for (size_t j = 0; j < size; ++j) {
49        b[j] *= 2; // actually modifies original array
50     }
51  }
52
53  // in function modifyElement, "e" is a local copy of array element
54  // a[3] passed from main
55  void modifyElement(int e)
56  {
57     // multiply parameter by 2
58     printf("Value in modifyElement is %d\n", e *= 2);
59  }
```

Passing Array to Functions (4)

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

Protecting Array Elements

- Function `tryToModifyArray` is defined with parameter `const int b[]`, which specifies that array `b` is constant and cannot be modified.
- The output shows the error messages produced by the compiler—the errors may be different for your compiler.

```
1  // in function tryToModifyArray, array b is const, so it cannot be
2  // used to modify its array argument in the caller
3  void tryToModifyArray(const int b[])
4  {
5     b[0] /= 2; // error
6     b[1] /= 2; // error
7     b[2] /= 2; // error
8  }
```