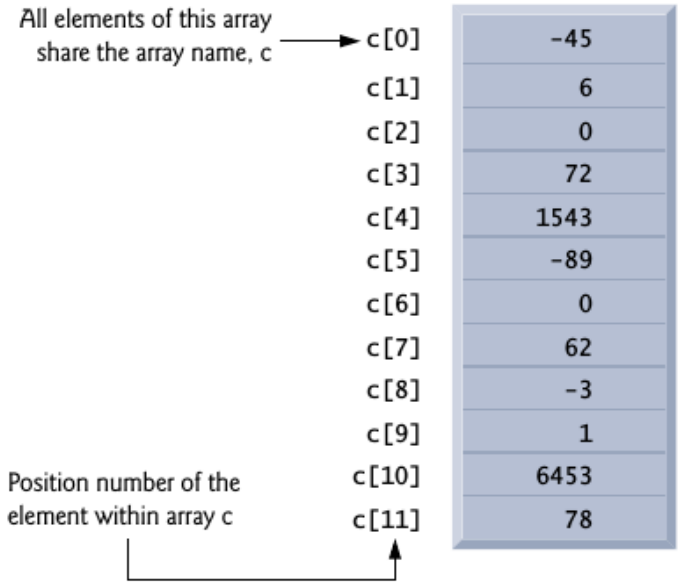
CC-112 Programming Fundamentals

Arrays

Nazar Khan
Department of Computer Science
University of the Punjab

Arrays

- An array is a group of *contiguous* memory locations that all have the *same type*.
- It is a *data structure*.
- Data structures are different from *data types* such as `int`, `float`, etc.
- Data structures usually contain multiple data elements.
- How the elements are *arranged* and *manipulated* is what defines a data structure. For example, array vs. stack vs. queue.



Initializing an array

```c
// Initializing the elements of an array to zeros.
#include <stdio.h>

// function main begins program execution
int main(void)
{
   int n[5]; // n is an array of five integers

   // set elements of array n to 0
   for (size_t i = 0; i < 5; ++i) {
      n[i] = 0; // set element at location i to 0
   }

   printf("%s%13s\n", "Element", "Value");

   // output contents of array n in tabular format
   for (size_t i = 0; i < 5; ++i) {
      printf("%7u%13d\n", i, n[i]);
   }
}
```

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

`size_t` is defined in stddef.h which is included by stdio.h already. It represents an unsigned integer type.

Initializing an array with initializer list

```c
// Initializing the elements of an array with an initializer list.
#include <stdio.h>

// function main begins program execution
int main(void)
{
   // use initializer list to initialize array n
   int n[5] = {32, 27, 64, 18, 95};

   printf("%s%13s\n", "Element", "Value");

   // output contents of array in tabular format
   for (size_t i = 0; i < 5; ++i) {
      printf("%7u%13d\n", i, n[i]);
   }
}
```

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

Customized initialization

```c
// Initializing the elements of array s to the even integers from 2 to 20.
#include <stdio.h>
#define SIZE 5 // maximum size of array

// function main begins program execution
int main(void)
{
   // symbolic constant SIZE can be used to specify array size
   int s[SIZE]; // array s has SIZE elements

   for (size_t j = 0; j < SIZE; ++j) { // set the values
      s[j] = 2 + 2 * j;
   }

   printf("%s%13s\n", "Element", "Value");

   // output contents of array s in tabular format
   for (size_t j = 0; j < SIZE; ++j) {
      printf("%7u%13d\n", j, s[j]);
   }
}
```

The `#define` preprocessor directive

- `#define SIZE 5` defines a *symbolic constant* SIZE whose value is 5.
- SIZE will be replaced everywhere by the *replacement text* 5 by the C preprocessor before compilation.
- Using symbolic constants makes programs more modifiable. Without SIZE, changing array from 5 to 10 elements would have required 3 changes in the code.
- As programs get larger, this technique becomes more useful for writing clear, easy to read, maintainable programs – a symbolic constant (like SIZE) is easier to understand than the numeric value 5, which could have different meanings throughout the code.

Sum of an array

```c
// Computing the sum of the elements of an array.
#include <stdio.h>
#define SIZE 12

// function main begins program execution
int main(void)
{
   // use an initializer list to initialize the array
   int a[SIZE] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
   int total = 0; // sum of array

   // sum contents of array a
   for (size_t i = 0; i < SIZE; ++i) {
      total += a[i];
   }

   printf("Total of array element values is %d\n", total);
}
```

Computing a histogram

Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.

```
// Analyzing a student poll.
#include <stdio.h>
#define RESPONSES_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 11

// function main begins program execution
int main(void)
{
   // initialize frequency counters to 0
   int frequency[FREQUENCY_SIZE] = {0};

   // place the survey responses in the responses array
   int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10};

   // for each answer, select value of an element of array responses
```

Computing a histogram

```
   // and use that value as an index in array frequency to
   // determine element to increment
   for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
      ++frequency[responses[answer]];
   }

   // display results
   printf("%s%17s\n", "Rating", "Frequency");

   // output the frequencies in a tabular format
   for (size_t rating = 1; rating < FREQUENCY_SIZE; ++rating) {
      printf("%6d%17d\n", rating, frequency[rating]);
   }
}
```

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

Displaying a histogram

```
// Displaying a histogram.
#include <stdio.h>
#define SIZE 5

// function main begins program execution
int main(void)
{
   // use initializer list to initialize array n
   int n[SIZE] = {19, 3, 15, 7, 11};

   printf("%s%13s%17s\n", "Element", "Value", "Histogram");

   // for each element of array n, output a bar of the histogram
   for (size_t i = 0; i < SIZE; ++i) {
      printf("%7u%13d          ", i, n[i]) ;

      for (int j = 1; j <= n[i]; ++j) { // print one bar
         printf("%c", '*');
           }

      puts(""); // end a histogram bar with a newline
   }
}
```

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

Out-of-bounds access

- C has no array bounds checking to prevent the program from referring to an element that does not exist.
- For example, `n[SIZE+1]` and `n[-10]` are legal!
- Thus, an executing program can "walk off" either end of an array without warning – a security problem!
- You should ensure that all array references remain within the bounds of the array.

Law of large numbers

```c
// Roll a six-sided die 60,000,000 times
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 7

// function main begins program execution
int main(void)
{
   unsigned int frequency[SIZE] = {0}; // clear counts

   srand(time(NULL)); // seed random number generator

   // roll die 60,000,000 times
   for (unsigned int roll = 1; roll <= 60000000; ++roll) {
      size_t face = 1 + rand() % 6;
      ++frequency[face]; // replaces entire switch of Fig. 5.12
   }

   printf("%s%17s\n", "Face", "Frequency");

   // output frequency elements 1-6 in tabular format
   for (size_t face = 1; face < SIZE; ++face) {
      printf("%4d%17d\n", face, frequency[face]);
   }
}
```

String is an array of characters

```
//initializing a character array with a string
char string1[] = "first";
printf("%s\n",string1);
for (int i=0; i<6; i++)
   printf("%c ",string1[i]);
```

- The string “first” contains five characters plus a special *string-termination character* called the *null character*.
- The escape sequence representing the null character is '\0'.
- All strings in C end with this character.

```
//initializing with list
char string2[] = {'f', 'i', 'r', 's', 't', '\0'};
for (int i=0; i<6; i++)
   printf("%c(%d) ",string2[i], string2[i]);
```

Inputting into a character array

```
char string2[20];
```

creates a character array capable of storing a string of at most 19 characters and a terminating null character.

```
char string2[20];
scanf("%19s", string2);
```

reads a string from the keyboard into `string2`.

> Why no & before second argument of `scanf`? Because *an array name* ***is*** *the address of the first element of the array*.

`scanf` will read characters until a *space, tab, newline* or *end-of-file indicator* is encountered.

String is an array of characters

```c
// Treating character arrays as strings.
#include <stdio.h>
#define SIZE 20

// function main begins program execution
int main(void)
{
   char string1[SIZE]; // reserves 20 characters
   char string2[] = "string literal"; // reserves 15 characters

   // read string from user into array string1
   printf("%s", "Enter a string (no longer than 19 characters): ");
   scanf("%19s", string1); // input no more than 19 characters

   // output strings
   printf("string1 is: %s\nstring2 is: %s\n"
          "string1 with spaces between characters is:\n",
          string1, string2);

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

```
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
```

Static arrays are initialized to 0 by default

```c
// Static arrays are initialized to zero if not explicitly initialized.
#include <stdio.h>

void staticArrayInit(void); // function prototype
void automaticArrayInit(void); // function prototype

// function main begins program execution
int main(void)
{
   puts("First call to each function:");
   staticArrayInit();
   automaticArrayInit();

   puts("\n\nSecond call to each function:");
   staticArrayInit();
   automaticArrayInit();
}

// function to demonstrate a static local array
void staticArrayInit(void)
{
   // initializes elements to 0 before the function is called
   static int array1[3];

   puts("\nValues on entering staticArrayInit:");

   // output contents of array1
   for (size_t i = 0; i <= 2; ++i) {
```

Static arrays are initialized to 0 by default

```
      printf("array1[%u] = %d  ", i, array1[i]);
   }

   puts("\nValues on exiting staticArrayInit:");

   // modify and output contents of array1
   for (size_t i = 0; i <= 2; ++i) {
      printf("array1[%u] = %d  ", i, array1[i] += 5);
   }
}

// function to demonstrate an automatic local array
void automaticArrayInit(void)
{
   // initializes elements each time function is called
   int array2[3] = { 1, 2, 3 };

   puts("\n\nValues on entering automaticArrayInit:");

   // output contents of array2
   for (size_t i = 0; i <= 2; ++i) {
      printf("array2[%u] = %d  ", i, array2[i]);
   }

   puts("\nValues on exiting automaticArrayInit:");

   // modify and output contents of array2
   for (size_t i = 0; i <= 2; ++i) {
      printf("array2[%u] = %d  ", i, array2[i] += 5);
```

Static arrays are initialized to 0 by default

```
    }
}
```

```
First call to each function:

Values on entering staticArrayInit:
array1[0] = 0 array1[1] = 0 array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10 array1[1] = 10 array1[2] = 10

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8
```

Array name = address of first element of array

```c
// Array name is the same as the address of the arrays first element.
#include <stdio.h>

// function main begins program execution
int main(void)
{
   char array[5]; // define an array of size 5

   printf("    array = %p\n&array[0] = %p\n   &array = %p\n",
      array, &array[0], &array);
}
```

```
array = 0031F930
&array[0] = 0031F930
&array = 0031F930
```

Passing arrays as arguments to functions

```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

   modifyArray(a, SIZE); // pass array a to modifyArray by reference
   puts("The values of the modified array are:");

   // output modified array
   for (size_t i = 0; i < SIZE; ++i) {
```

Passing arrays as arguments to functions

```
      printf("%3d", a[i]);
   }

   // output value of a[3]
   printf("\n\n\nEffects of passing array element "
      "by value:\n\nThe value of a[3] is %d\n", a[3]);

   modifyElement(a[3]); // pass array element a[3] by value

   // output value of a[3]
   printf("The value of a[3] is %d\n", a[3]);
}

// in function modifyArray, "b" points to the original array "a"
// in memory
void modifyArray(int b[], size_t size)
{
   // multiply each array element by 2
   for (size_t j = 0; j < size; ++j) {
      b[j] *= 2; // actually modifies original array
   }
}

// in function modifyElement, "e" is a local copy of array element
// a[3] passed from main
void modifyElement(int e)
{
   // multiply parameter by 2
   printf("Value in modifyElement is %d\n", e *= 2);
```

Passing arrays as arguments to functions

```
}
```

```
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
```

The const type qualifier

```c
// Using the const type qualifier with arrays.
#include <stdio.h>

void tryToModifyArray(const int b[]); // function prototype

// function main begins program execution
int main(void)
{
   int a[] = { 10, 20, 30 }; // initialize array a

   tryToModifyArray(a);

   printf("%d %d %d\n", a[0], a[1], a[2]);
}

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

Sorting an array

```
// Sorting an array's values into ascending order.
#include <stdio.h>
#define SIZE 10

// function main begins program execution
int main(void)
{
   // initialize a
   int a[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37};

   puts("Data items in original order");

   // output original array
   for (size_t i = 0; i < SIZE; ++i) {
      printf("%4d", a[i]);
   }

   // bubble sort
   // loop to control number of passes
   for (unsigned int pass = 1; pass < SIZE; ++pass) {

      // loop to control number of comparisons per pass
      for (size_t i = 0; i < SIZE - 1; ++i) {

         // compare adjacent elements and swap them if first
         // element is greater than second element
         if (a[i] > a[i + 1]) {
            int hold = a[i];
```

Sorting an array

```
            a[i] = a[i + 1];
            a[i + 1] = hold;
         }
      }
   }

   puts("\nData items in ascending order");

   // output sorted array
   for (size_t i = 0; i < SIZE; ++i) {
      printf("%4d", a[i]);
   }

   puts("");
}
```

```
Data items in original order
   2 6 4 8 10 12 89 68 45 37
Data items in ascending order
   2 4 6 8 10 12 37 45 68 89
```

Computing mean, median, and mode

```
// Survey data analysis with arrays;
// computing the mean, median and mode of the data.
#include <stdio.h>
#define SIZE 99

// function prototypes
void mean(const unsigned int answer[]);
void median(unsigned int answer[]);
void mode(unsigned int freq[], const unsigned int answer[]) ;
void bubbleSort(unsigned int a[]);
void printArray(const unsigned int a[]);

// function main begins program execution
int main(void)
{
   unsigned int frequency[10] = {0}; // initialize array frequency

   // initialize array response
   unsigned int response[SIZE] =
      {6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
       7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
       6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
       7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
       6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
       7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
       5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
       7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
       7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
```

Computing mean, median, and mode

```
        4, 5, 6, 1, 6, 5, 7, 8, 7};

   // process responses
   mean(response);
   median(response);
   mode(frequency, response);
}

// calculate average of all response values
void mean(const unsigned int answer[])
{
   printf("%s\n%s\n%s\n", "********", "  Mean", "********");

   unsigned int total = 0; // variable to hold sum of array elements

   // total response values
   for (size_t j = 0; j < SIZE; ++j) {
      total += answer[j];
   }

   printf("The mean is the average value of the data\n"
          "items. The mean is equal to the total of\n"
          "all the data items divided by the number\n"
          "of data items (%u). The mean value for\n"
          "this run is: %u / %u = %.4f\n\n",
          SIZE, total, SIZE, (double) total / SIZE);
}

// sort array and determine median element's value
```

Computing mean, median, and mode

```
void median(unsigned int answer[])
{
   printf("\n%s\n%s\n%s\n%s",
          "********", " Median", "********",
          "The unsorted array of responses is");

   printArray(answer); // output unsorted array

   bubbleSort(answer); // sort array

   printf("%s", "\n\nThe sorted array is");
   printArray(answer); // output sorted array

   // display median element
   printf("\n\nThe median is element %u of\n"
          "the sorted %u element array.\n"
          "For this run the median is %u\n\n",
          SIZE / 2, SIZE, answer[SIZE / 2]);
}

// determine most frequent response
void mode(unsigned int freq[], const unsigned int answer[])
{
   printf("\n%s\n%s\n%s\n", "********", "  Mode", "********");

   // initialize frequencies to 0
   for (size_t rating = 1; rating <= 9; ++rating) {
      freq[rating] = 0;
   }
```

Computing mean, median, and mode

```
// summarize frequencies
for (size_t j = 0; j < SIZE; ++j) {
   ++freq[answer[j]];
}

// output headers for result columns
printf("%s%11s%19s\n\n%54s\n%54s\n\n",
        "Response", "Frequency", "Histogram",
        "1    1    2    2", "5    0    5    0    5");

// output results
unsigned int largest = 0; // represents largest frequency
unsigned int modeValue = 0; // represents most frequent response

for (size_t rating = 1; rating <= 9; ++rating) {
   printf("%8u%11u          ", rating, freq[rating]);

   // keep track of mode value and largest frequency value
   if (freq[rating] > largest) {
      largest = freq[rating];
      modeValue = rating;
   }

   // output histogram bar representing frequency value
   for (unsigned int h = 1; h <= freq[rating]; ++h) {
      printf("%s", "*");
   }
```

Computing mean, median, and mode

```
      puts(""); // being new line of output
   }

   // display the mode value
   printf("\nThe mode is the most frequent value.\n"
          "For this run the mode is %u which occurred"
          " %u times.\n", modeValue, largest);
}

// function that sorts an array with bubble sort algorithm
void bubbleSort(unsigned int a[])
{
   // loop to control number of passes
   for (unsigned int pass = 1; pass < SIZE; ++pass) {

      // loop to control number of comparisons per pass
      for (size_t j = 0; j < SIZE - 1; ++j) {

         // swap elements if out of order
         if (a[j] > a[j + 1]) {
            unsigned int hold = a[j];
            a[j] = a[j + 1];
            a[j + 1] = hold;
         }
      }
   }
}

// output array contents (20 values per row)
```

Computing mean, median, and mode

```
void printArray(const unsigned int a[])
{
   // output array contents
   for (size_t j = 0; j < SIZE; ++j) {

      if (j % 20 == 0) { // begin new line every 20 values
         puts("");
      }

      printf("%2u", a[j]);
   }
}
```

Linear search through an array

```c
// Linear search of an array.
#include <stdio.h>
#define SIZE 100

// function prototype
size_t linearSearch(const int array[], int key, size_t size);

// function main begins program execution
int main(void)
{
   int a[SIZE]; // create array a

   // create some data
   for (size_t x = 0; x < SIZE; ++x) {
      a[x] = 2 * x;
   }

   printf("Enter integer search key: ");
   int searchKey; // value to locate in array a
   scanf("%d", &searchKey);

   // attempt to locate searchKey in array a
   size_t index = linearSearch(a, searchKey, SIZE);

   // display results
   if (index != -1) {
      printf("Found value at index %lu\n", index);
   }
```

Linear search through an array

```
   else {
      puts("Value not found");
      printf("Expressed as unsigned long int, value of index is %lu\n", index);
      printf("But expressed as int, value of index is %d\n", (int)index);
   }
}

// compare key to every element of array until the location is found
// or until the end of array is reached; return index of element
// if key is found or -1 if key is not found
size_t linearSearch(const int array[], int key, size_t size)
{
   // loop through array
   for (size_t n = 0; n < size; ++n) {

      if (array[n] == key) {
         return n; // return location of key
      }
   }

   return -1; // key not found
}
```

Binary search through a sorted array

```
// Binary search of a sorted array.
#include <stdio.h>
#define SIZE 15

// function prototypes
size_t binarySearch(const int b[], int searchKey, size_t low, size_t high);
void printHeader(void);
void printRow(const int b[], size_t low, size_t mid, size_t high);

// function main begins program execution
int main(void)
{
   int a[SIZE]; // create array a

   // create data
   for (size_t i = 0; i < SIZE; ++i) {
      a[i] = 2 * i;
   }

   printf("%s", "Enter a number between 0 and 28: ");
   int key; // value to locate in array a
   scanf("%d", &key);

   printHeader();

   // search for key in array a
   size_t result = binarySearch(a, key, 0, SIZE - 1);
```

Binary search through a sorted array

```
   // display results
   if (result != -1) {
      printf("\n%d found at index %d\n", key, result);
   }
   else {
      printf("\n%d not found\n", key);
   }
}

// function to perform binary search of an array
size_t binarySearch(const int b[], int searchKey, size_t low, size_t high)
{
   // loop until low index is greater than high index
   while (low <= high) {

      // determine middle element of subarray being searched
      size_t middle = (low + high) / 2;

      // display subarray used in this loop iteration
      printRow(b, low, middle, high);

      // if searchKey matched middle element, return middle
      if (searchKey == b[middle]) {
         return middle;
      }

      // if searchKey is less than middle element, set new high
      else if (searchKey < b[middle]) {
         high = middle - 1; // search low end of array
```

Binary search through a sorted array

```
      }

      // if searchKey is greater than middle element, set new low
      else {
         low = middle + 1; // search high end of array
      }
   } // end while

   return -1; // searchKey not found
}

// Print a header for the output
void printHeader(void)
{
   puts("\nSubscripts:");

   // output column head
   for (unsigned int i = 0; i < SIZE; ++i) {
      printf("%3u ", i);
   }

   puts(""); // start new line of output

   // output line of - characters
   for (unsigned int i = 1; i <= 4 * SIZE; ++i) {
      printf("%s", "-");
   }

   puts(""); // start new line of output
```

Binary search through a sorted array

```
}

// Print one row of output showing the current
// part of the array being processed.
void printRow(const int b[], size_t low, size_t mid, size_t high)
{
   // loop through entire array
   for (size_t i = 0; i < SIZE; ++i) {

      // display spaces if outside current subarray range
      if (i < low || i > high) {
         printf("%s", "    ");
      }
      else if (i == mid) { // display middle element
         printf("%3d*", b[i]); // mark middle value
      }
      else { // display other elements in subarray
         printf("%3d ", b[i]);
      }
   }

   puts(""); // start new line of output
}
```

Binary search through a sorted array

First run

```
Enter a number between 0 and 28: 25

Indices:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                             24  26* 28
                                             24*

25 not found
```

Second run

```
Enter a number between 0 and 28: 8

Indices:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

8 found at index 4
```

Binary search through a sorted array

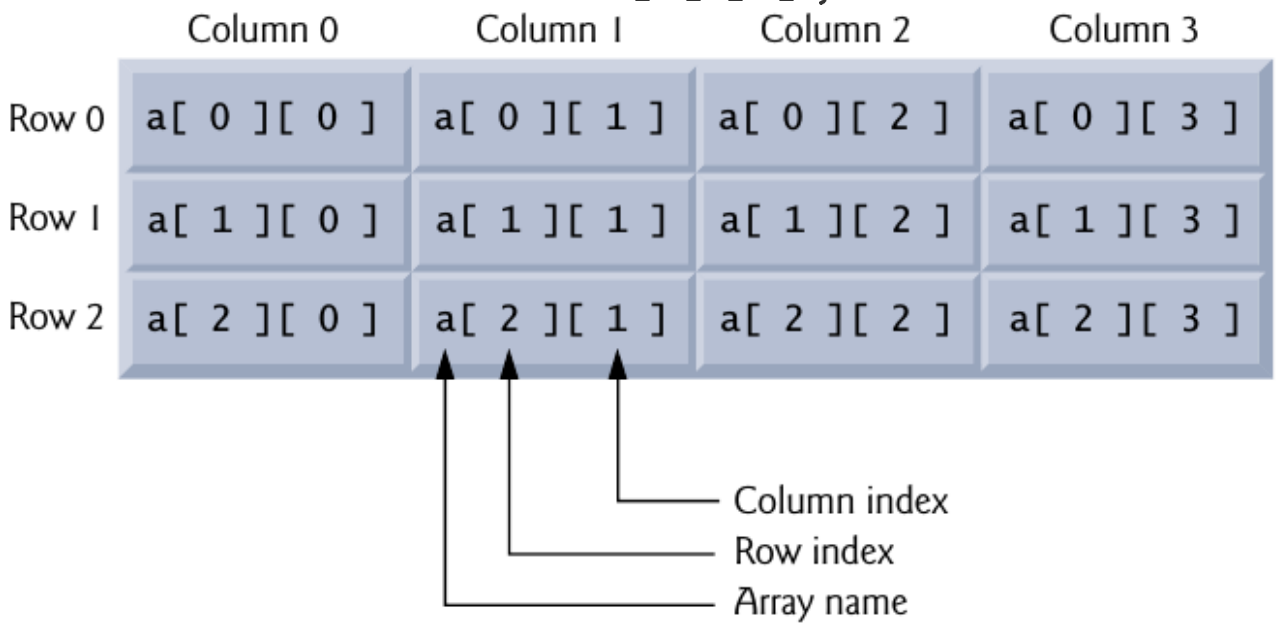
Third run

```
Enter a number between 0 and 28: 6

Indices:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
-------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found at index 3
```

Multidimensional arrays

- So far we have considered 1-dimensional array only. Require 1 index to access an element.
- But arrays can be 2-dimensional. Require 2 indices to access an element.
- A 2-dimensional array with 2 rows and 3 columns can be declared as

```
int a[2][3];
```



- Can be d-dimensional as well. Require d indices to access an element.

Multidimensional arrays

```
// Initializing multidimensional arrays.
#include <stdio.h>

void printArray(int a[][3]); // function prototype

// function main begins program execution
int main(void)
{
   int array1[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };
   puts("Values in array1 by row are:");
   printArray(array1);

   int array2[2][3] = { 1, 2, 3, 4, 5 };
   puts("Values in array2 by row are:");
   printArray(array2);

   int array3[2][3] = { { 1, 2 }, { 4 } };
   puts("Values in array3 by row are:");
   printArray(array3);
}

// function to output array with two rows and three columns
void printArray(int a[][3])
{
   // loop through rows
   for (size_t i = 0; i <= 1; ++i) {

      // output column values
```

Multidimensional arrays

```
      for (size_t j = 0; j <= 2; ++j) {
         printf("%d ", a[i][j]);
      }

      printf("\n"); // start new line of output
   }
}
```

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

Manipulating 2D arrays

```c
// Two-dimensional array manipulations.
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

// function prototypes
int minimum(const int grades[][EXAMS], size_t pupils, size_t tests);
int maximum(const int grades[][EXAMS], size_t pupils, size_t tests);
double average(const int setOfGrades[], size_t tests);
void printArray(const int grades[][EXAMS], size_t pupils, size_t tests);

// function main begins program execution
int main(void)
{
   // initialize student grades for three students (rows)
   int studentGrades[STUDENTS][EXAMS] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   // output array studentGrades
   puts("The array is:");
   printArray(studentGrades, STUDENTS, EXAMS);

   // determine smallest and largest grade values
   printf("\n\nLowest grade: %d\nHighest grade: %d\n",
      minimum(studentGrades, STUDENTS, EXAMS),
      maximum(studentGrades, STUDENTS, EXAMS));
```

Manipulating 2D arrays

```
   // calculate average grade for each student
   for (size_t student = 0; student < STUDENTS; ++student) {
      printf("The average grade for student %u is %.2f\n",
         student, average(studentGrades[student], EXAMS));
   }
}

// Find the minimum grade
int minimum(const int grades[][EXAMS], size_t pupils, size_t tests)
{
   int lowGrade = 100; // initialize to highest possible grade

   // loop through rows of grades
   for (size_t i = 0; i < pupils; ++i) {

      // loop through columns of grades
      for (size_t j = 0; j < tests; ++j) {

         if (grades[i][j] < lowGrade) {
            lowGrade = grades[i][j];
         }
      }
   }

   return lowGrade; // return minimum grade
}

// Find the maximum grade
```

Manipulating 2D arrays

```
int maximum(const int grades[][EXAMS], size_t pupils, size_t tests)
{
   int highGrade = 0; // initialize to lowest possible grade

   // loop through rows of grades
   for (size_t i = 0; i < pupils; ++i) {

      // loop through columns of grades
      for (size_t j = 0; j < tests; ++j) {

         if (grades[i][j] > highGrade) {
            highGrade = grades[i][j];
         }
      }
   }

   return highGrade; // return maximum grade
}

// Determine the average grade for a particular student
double average(const int setOfGrades[], size_t tests)
{
   int total = 0; // sum of test grades

   // total all grades for one student
   for (size_t i = 0; i < tests; ++i) {
      total += setOfGrades[i];
   }
```

Manipulating 2D arrays

```
   return (double) total / tests; // average
}

// Print the array
void printArray(const int grades[][EXAMS], size_t pupils, size_t tests)
{
   // output column heads
   printf("%s", "                  [0]   [1]   [2]   [3]");

   // output grades in tabular format
   for (size_t i = 0; i < pupils; ++i) {

      // output label for row
      printf("\nstudentGrades[%u] ", i);

      // output grades for one student
      for (size_t j = 0; j < tests; ++j) {
         printf("%-5d", grades[i][j]);
      }
   }
}
```

Manipulating 2D arrays

```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

Variable-length arrays

- Sometimes array size is not known at compilation time.
- It might only be known during program execution.
- A *variable-length array* is an array whose size is defined by an expression evaluated at execution time.

```
// Using variable-length arrays in C99
#include <stdio.h>

// function prototypes
void print1DArray(size_t size, int array[size]);
void print2DArray(size_t row, size_t col, int array[row][col]);

int main(void)
{
   printf("%s", "Enter size of a one-dimensional array: ");
   int arraySize; // size of 1-D array
   scanf("%d", &arraySize);

   int array[arraySize]; // declare 1-D variable-length array

   printf("%s", "Enter number of rows and columns in a 2-D array: ");
   int row1, col1; // number of rows and columns in a 2-D array
   scanf("%d %d", &row1, &col1);
```

Variable-length arrays

```
int array2D1[row1][col1]; // declare 2-D variable-length array

printf("%s",
   "Enter number of rows and columns in another 2-D array: ");
int row2, col2; // number of rows and columns in another 2-D array
scanf("%d %d", &row2, &col2);

int array2D2[row2][col2]; // declare 2-D variable-length array

// test sizeof operator on VLA
printf("\nsizeof(array) yields array size of %d bytes\n",
   sizeof(array));

// assign elements of 1-D VLA
for (size_t i = 0; i < arraySize; ++i) {
   array[i] = i * i;
}

// assign elements of first 2-D VLA
for (size_t i = 0; i < row1; ++i) {
   for (size_t j = 0; j < col1; ++j) {
      array2D1[i][j] = i + j;
   }
}

// assign elements of second 2-D VLA
for (size_t i = 0; i < row2; ++i) {
   for (size_t j = 0; j < col2; ++j) {
```

Variable-length arrays

```
            array2D2[i][j] = i + j;
         }
      }

      puts("\nOne-dimensional array:");
      print1DArray(arraySize, array); // pass 1-D VLA to function

      puts("\nFirst two-dimensional array:");
      print2DArray(row1, col1, array2D1); // pass 2-D VLA to function

      puts("\nSecond two-dimensional array:");
      print2DArray(row2, col2, array2D2); // pass other 2-D VLA to function
}

void print1DArray(size_t size, int array[size])
{
      // output contents of array
      for (size_t i = 0; i < size; i++) {
         printf("array[%d] = %d\n", i, array[i]);
      }
}

void print2DArray(size_t row, size_t col, int array[row][col])
{
      // output contents of array
      for (size_t i = 0; i < row; ++i) {
         for (size_t j = 0; j < col; ++j) {
            printf("%5d", array[i][j]);
         }
```

Variable-length arrays

```
        puts("");
    }
}
```

```
Enter size of a one-dimensional array: 6
Enter number of rows and columns in a 2-D array: 2 5
Enter number of rows and columns in another 2-D array: 4 3

sizeof(array) yields array size of 24 bytes

One-dimensional array:
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25

First two-dimensional array:
    0    1    2    3    4
    1    2    3    4    5

Second two-dimensional array:
    0    1    2
    1    2    3
    2    3    4
```

Variable-length arrays

3 4 5