Arrays
Arrays

- Arrays
- Defining and Initializing Arrays
- Array Example
- Subscript Out-of-Range Example
- Passing Arrays to Functions
  - Call by Reference
- Multiple-Subscripted Arrays
  - Double-Subscripted Array Example
- `enum` and `switch`
Arrays:: Structures of related data items

- Static entities, namely, the size of an array remains constant throughout the program execution.
- A group of contiguous memory locations with the same name and type.
Arrays

- To refer to an array element, specify
  - the array name
  - the position number \{in C this is an offset\}
    • referred to as a **subscript**.

- **Format:**

  \texttt{arrayname[position number]}

- First element is at position 0
- \texttt{n} element array named \texttt{c}:
  \texttt{c[ 0 ], c[ 1 ]...c[ n - 1 ]}
**name of array** (Note that all elements of this array have the same name, c)

<table>
<thead>
<tr>
<th>c[0]</th>
<th>-45</th>
</tr>
</thead>
<tbody>
<tr>
<td>c[1]</td>
<td>6</td>
</tr>
<tr>
<td>c[2]</td>
<td>0</td>
</tr>
<tr>
<td>c[3]</td>
<td>72</td>
</tr>
<tr>
<td>c[4]</td>
<td>1543</td>
</tr>
<tr>
<td>c[5]</td>
<td>-89</td>
</tr>
<tr>
<td>c[6]</td>
<td>0</td>
</tr>
<tr>
<td>c[7]</td>
<td>62</td>
</tr>
<tr>
<td>c[8]</td>
<td>-3</td>
</tr>
<tr>
<td>c[9]</td>
<td>1</td>
</tr>
<tr>
<td>c[10]</td>
<td>6453</td>
</tr>
</tbody>
</table>

Position number of the element within array c
Arrays

- Array elements are like normal variables.
  
  ```c
  c[ 0 ] = 3;
  printf( "%d", c[ 0 ] );
  ```

- One can perform operations within the subscript which must be an integer or an integer expression.

Examples:

```c
  c[x - 2] = 6;
  c[i+j-4] = c[x-2];
  c[intfcn(i)] = 0;
```
6.3 Defining Arrays

- When defining arrays, specify
  - Name
  - Type of array
  - Number of elements
    
    ```
    arrayType arrayName[ numberOfElements ];
    ```
  - Examples:
    ```
    int c[ 10 ];
    float myArray[ 3284 ];
    ```

- Defining multiple arrays of same type
  - Format similar to regular variables
  - Example:
    ```
    int b[ 100 ], x[ 27 ];
    ```
Initializing Array

- `int n[ 5 ] = { 1, 2, 3, 4, 5 };`
  - If not enough initializers, rightmost elements become 0.
    ```
    int n[ 5 ] = { 0 } ;
    ```
  - All elements 0

- If too many initializers, a syntax error occurs!

- **C arrays have no bounds checking!!**

- If size omitted, initializers determine size.
  ```
  int n[ ] = { 1, 2, 3, 4, 5 };
  ```
  - 5 initializer elements, therefore `n` is a 5 element array.
/* Fig. 6.5: fig06_05.c */
Initialize the elements of array s to the even integers from 2 to 20 */

#include <stdio.h>

#define SIZE 10 /* 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 */
    int j; /* counter */

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

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

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

    return 0; /* indicates successful termination */
}

#define directive tells compiler to replace all instances of the word SIZE with 10.

SIZE is replaced with 10 by the compiler, so array s has 10 elements.
Fig. 6.5: fig06_05.c

// Initializing the elements of array s to the even integers from 2 to 20.
#include <stdio.h>
#define SIZE 10  // 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
  size_t j;  // counter

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

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

  // output contents of array s in tabular format
  for ( j = 0; j < SIZE; ++j ) {
    printf( "%lu%ls\n", j, s[ j ] );
  }  // end for

  return 0;
}  // end main

Fig. 6.5  |  Initialize the elements of array s to the even integers from 2 to 20. (Part 1 of 2.)
<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
</tr>
</tbody>
</table>

**Fig. 6.5** Initialize the elements of array `s` to the even integers from 2 to 20. (Part 2 of 2.)
Arrays are **dangerous in C** because:

- There is no default initialization.
- There is no bounds checking for subscripts out-of-range.
/* Bizarre Example of subscripting out of range */
int main ()
{
    int i, m, n, j;
    int a[100], k, p;

    j = 77; k = 88; p = 99;

    for (i = -1; i <= 103; i++)
    {
        a[i] = 2*i;
        if (i > 98) printf("i =%d, a[i] = %d\n", i, a[i]);
    }

    printf("j = %d, n = %d, m = %d, i = %d, k = %d, p = %d\n",
        j, n, m, i, k, p);
    printf("%d %d %d %d %u\n",
        a[-1], a[99], a[100], a[102], a[i]);

    return 0;
}
6.5 Passing Arrays to Functions

- To pass an array argument to a function, specify the name of the array without any brackets.
- The array size is usually passed to the function.

```c
int myArray[24];
myFunction(myArray, 24);
```

- Arrays are passed by-reference.
  - The name of the array is associated with the address of the first array element.
  - The function knows where the array is stored and it can modify the original memory locations.
6.5 Passing Arrays to Functions

- Individual array elements
  - Are passed **by value**.
  - Pass the subscripted name (i.e., \texttt{myArray[3]}) to function.

- Function prototype
  
  \texttt{void modifyArray( int b[], int arraySize );}
  
  - Parameter names are optional in prototype.
    - \texttt{int b[]} could be written \texttt{int []}
    - \texttt{int arraySize} could be simply \texttt{int}
/* Arrays are passed using Call by Reference */
#include <math.h>
#define SIZE 6

void flip (float fray [], int fsize)
{
    float temp;
    int i,j;

    i = fsize - 1;
    for (j = 0; j < fsize/2 ; j++)
    {
        temp = fray[j];
        fray[j] = fray[i];
        fray[i] = temp;
        i--;
    }
    return;
}
```c
int main ()
{
    float var[SIZE];
    int i,j;
    for (i=0; i < SIZE; i++)
    {
        var[i] = 1.0/pow (2.0,i);
        printf(" %5.3f", var[i]);
    }
    printf("\n");

    for (j=0; j < 2; j++)
    {
        flip (var, SIZE);
        for (i=0; i < SIZE; i++)
            printf(" %5.3f", var[i]);
        printf("\n");
    }
}
```

```
$ ./passray
  1.000  0.500  0.250  0.125  0.062  0.031
  0.031  0.062  0.125  0.250  0.500  1.000
  1.000  0.500  0.250  0.125  0.062  0.031
```
6.9 Multiple-Subscripted Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (m by n array)
  - Like matrices: specify row, then column
- Initialization
  - `int b[2][2] = {{1, 2}, {3, 4}};`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero
    `int b[2][2] = {{1}, {3, 4}};`
- Referencing elements
  - Specify row, then column
    `printf("%d", b[0][1]);`
Fig. 6.20 Double-Subscripted Array

<table>
<thead>
<tr>
<th>Row</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
</tr>
<tr>
<td>1</td>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
</tr>
</tbody>
</table>

- Three rows and four columns.
/* Fig. 6.22: fig06_22.c 
   Double-subscripted array example */ 
#include <stdio.h> 
#define STUDENTS 3 
#define EXAMS 4 

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

/* function main begins program execution */ 
int main( void ) 
{
    int student; /* student counter */ 

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

    /* output array studentGrades */ 
    printf( "The array is:\n" ); 
    printArray( studentGrades, STUDENTS, EXAMS ); 

    Notice “const” 

Each row in the array corresponds to a single student’s set of grades
/* determine smallest and largest grade values */
printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
    minimum( studentGrades, STUDENTS, EXAMS ),
    maximum( studentGrades, STUDENTS, EXAMS ) );

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

return 0; /* indicates successful termination */

} /* end main */
/* Find the minimum grade */

int minimum( const int grades[][ EXAMS ], int pupils, int tests )
{
    int i; /* student counter */
    int j; /* exam counter */
    int lowGrade = 100; /* initialize to highest possible grade */

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

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

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

        } /* end inner for */

    } /* end outer for */

    return lowGrade; /* return minimum grade */

} /* end function minimum */
/* Find the maximum grade */
int maximum( const int grades[][ EXAMS ], int pupils, int tests )
{
    int i; /* student counter */
    int j; /* exam counter */
    int highGrade = 0; /* initialize to lowest possible grade */

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

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

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

        } /* end inner for */

    } /* end outer for */

    return highGrade; /* return maximum grade */
} /* end function maximum */
/* Determine the average grade for a particular student */

double average( const int setOfGrades[], int tests )
{
    int i; /* exam counter */
    int total = 0; /* sum of test grades */

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

    return ( double ) total / tests; /* average */
} /* end function average */

/* Print the array */

void printArray( const int grades[][ EXAMS ], int pupils, int tests )
{
    int i; /* student counter */
    int j; /* exam counter */

    /* output column heads */
    printf( " [0] [1] [2] [3] " );

/* output grades in tabular format */
for ( i = 0; i < pupils; i++ ) {
    /* output label for row */
    printf( "\n\nstudentGrades[%d] " , i );

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

} /* end outer for */

} /* end function printArray */

The array is:

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>studentGrades[0]</td>
<td>77</td>
<td>68</td>
<td>86</td>
<td>73</td>
</tr>
<tr>
<td>studentGrades[1]</td>
<td>96</td>
<td>87</td>
<td>89</td>
<td>78</td>
</tr>
<tr>
<td>studentGrades[2]</td>
<td>70</td>
<td>90</td>
<td>86</td>
<td>81</td>
</tr>
</tbody>
</table>

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
An `enum` and a `switch` Example

```c
/* A program that uses enumerated types, switch and a sentinel to terminate input */

#define SENTINEL 10
int main ()
{
    int day;

    /* enum starts assigning positional integers beginning with 0 */

days {SUN, MON, TUES, WED, THUR, FRI, SAT};

    scanf("%d", &day);
    while( day != SENTINEL)
    {
        switch (day)
        {
            case MON:
            case WED:
            case FRI:
            printf("%d - Go to class\n", day);
            break;
        }
    }
```
case TUES:
    printf("%d - Sleep in until 10\n", day);
    break;

case THUR:
    printf("%d - Do laundry\n", day);
    break;

case SAT:
    printf("%d - Go to gym. ", day);
    printf("Go out to a movie\n");
    break;

case SUN:
    printf("%d - Study lots!\n", day);
    break;

default:
    printf("%d - This is invalid input. Try again.\n", day);
    break;
}
scanf("%d", &day);
}
printf("Sentinal encountered.\n");
return 0;
Review of Arrays

- Arrays
- Defining and Initializing Arrays
- Array Example
- Subscript Out-of-Range Example
- Passing Arrays to Functions
  - Call by Reference
- Multiple-Subscripted Arrays
  - Double-Subscripted Array Example
- Enum and Switch Example