Arrays

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:
  \[ \text{arrayname}[\text{position number}] \]
  - First element at position 0
  - \( n \) element array named \( \text{c} \):
    \[ \text{c}[0], \text{c}[1], ..., \text{c}[n-1] \]

Array Elements

- Array elements are like normal variables
  \[ \text{c}[0] = 3; \]
  \[ \text{printf}( \text{"%d", c[0]} ); \]
- We can perform operations in subscript.
  e.g. if \( \text{x} \) equals 3
  \[ \text{c}[\text{x}-2] = \text{c}[3] = \text{c}[\text{x}] \]
  \[ \text{c}[\text{x}+1] = \text{c}[4] \]
  \[ \text{c}[\text{x}-1] = \text{c}[2] \]
Declaring Arrays

- When declaring arrays, specify
  - Name
  - Type of array
  - Number of elements
    arrayType arrayName[ numberOfElements ];
  - Examples:
    int c[10];
    float myArray[3284];
- Declaring multiple arrays of same type
  - Format similar to regular variables
  - Example:
    int b[100], x[27];

Examples Using Arrays

- Initializers
  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 a syntax error is produced syntax error
  - C arrays have no bounds checking
  - If size omitted, initializers determine it
    int n[ ] = { 1, 2, 3, 4, 5 };
    - 5 initializers, therefore 5 element array

Initializing an Array

```
#include <stdio.h>

int main(void)
{
    int n[100], i;
    for (i=0; i < 100; i++)
        n[i] = i;
    for (i=0; i < 100; i++)
        printf("Element %d has value %d. \n", i, n[i]);
    return 1;
}
```

Examples

- Reading values into an array
  int i, x[100];
  for (i=0; i < 100; i=i+1) {
    printf("Enter an integer: ");
    scanf("%d", &x[i]);
  }

- Summing up all elements in an array
  int sum = 0;
  for (i=0; i<=99; i=i+1)
      sum = sum + x[i];
  printf("Sum = %d\n", sum);
Examples (contd.)

- Shifting the elements of an array to the left.

```c
/* store the value of the first element in a temporary variable */
temp = x[0];
for (i=0; i < 99; i=i+1)
x[i] = x[i+1];
//The value stored in temp is going to be the value of the last element:
x[99] = temp;
```

Examples

- Finding the location of a given value (item) in an array.

```c
i = 0;
while ((i<100) && (x[i] != item))
i = i + 1;
if (i == 100)
    loc = -1; // not found
else
    loc = i; // found in location i
```

Program Output

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>*************</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>***</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>*************</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>*************</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>*************</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>*************</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>*************</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>****</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>*************</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>*</td>
</tr>
</tbody>
</table>
Passing Arrays to Functions

• Passing arrays
  – To pass an array argument to a function, specify the name of the array without any brackets
    
    ```c
    int myArray[24]; //declaration in main
    myFunction(myArray, 24); //calling the function
    ```
  • Array size is usually passed to function
  – Arrays passed call-by-reference
  – Name of array is address of first element
  – Function knows where the array is stored
  – Modifies original memory locations

• Passing array elements
  – Passed by call-by-value
  – Pass subscripted name (i.e., `myArray[3]`) to function

Example

```c
int sum(int a[], int n)
{
    int s = 0;
    for (j=0; j < n; j++)
        s = s + a[j];
    return s;
}
```

```c
int main()
{
    int a[SIZE] = { 0, 1, 2, 3, 4 }, i;
    printf( "Effects of passing entire array call by reference:
    The values of the original array are:
    " );
    for ( i = 0; i <= SIZE - 1; i++ )
        printf( "%3d", a[i] );
    printf( "\n" );
    modifyArray( a, SIZE );
    printf( "The values of the modified array are:
    " );
    for ( i = 0; i <= SIZE - 1; i++ )
        printf( "%3d", a[i] );
    printf( \n "
    
    Effects of passing array element call by value:
    The value of a[3] is %d
    
    ``
    ```
    modifyElement( a[3] );
    printf( "The value of a[ 3 ] is %d\n", a[3] );
    return 0;
}
```

• Function prototype
  ```c
  void modifyArray( int b[], int arraySize );
  ```

• Parameter names optional in prototype
  ```c
  int b[] could be written int []
  int arraySize could be simply int
  ```

Note: `a[]` is a notational convenience. In fact

```c
    int a[] = int *a
```
```c
void modifyArray(int b[], int size)
{
    int j;
    for (j = 0; j <= size - 1; j++)
        b[j] *= 2;
}

void modifyElement(int e)
{
    printf("Value in modifyElement is %d\n", e *= 2);
}
```

Effects of passing entire array call 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 call by value:
The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6

---

Two Dimensional Arrays

- Multiple subscripts arrays
  - Tables with rows and columns (m by n array)
  - Like matrices: specify row, then column
  ```plaintext
  Row 0  | Column 1 | Column 2 | Column 3 |
  --- | --- | --- | --- |
  1  | 2  | 3  | 4  |
  5  | 6  | 7  | 8  |
  9  | 10 | 11 | 12 |
  Column subscript
  Array name
  Row subscript
  ```

- Initialization
  - `int b[2][2] = {{ 1,2 },{ 3,4 }};`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero
  ```plaintext
  int b[2][2] = {{1}, {3, 4}};
  ```

- Referencing elements
  - Specify row, then column
  ```plaintext
  printf("Enter a number: ");
  scanf("%d", &a[row][col]);
  ```

Examples

- Reading values into a two-dimensional array:
  ```c
  int a[10][20];
  for (row=0; row < 10; row = row+1){
    for(col=0; col < 20; col = col+1) {
      printf("Enter a number: ");
      scanf("%d", &a[row][col]);
    }
  }
  ```
```
```c
#include <stdio.h>

#define STUDENTS 3
#define EXAMS 4

int minimum( int studentGrades[][ EXAMS ], int, int );
int maximum( int studentGrades[][ EXAMS ], int, int );
double average( int studentGrades[], int );
void printArray( int studentGrades[][ EXAMS ], int, int );

int main() {
    int student;
    int studentGrades[ STUDENTS ][ EXAMS ] = {
        { 77, 68, 86, 73 },
        { 96, 97, 99, 74 },
        { 70, 90, 86, 81 }
    };

    printf( "The array is:
" );
    printArray( studentGrades, STUDENTS, EXAMS );
    printf( "Lowest grade: %d
Highest grade: %d
", minimum( studentGrades, STUDENTS, EXAMS ),
maximum( studentGrades, STUDENTS, EXAMS ) );

    for ( student = 0; student <= STUDENTS - 1; student++ )
        printf( "The average grade for student %d is %.2f
", student,
average( studentGrades[ student ], EXAMS ) );

    return 0;
}

int minimum( int grades[][ EXAMS ], int pupils, int tests ) {
    int i, j, lowGrade = 100;
    for ( i = 0; i <= pupils - 1; i++ )
        for ( j = 0; j <= tests - 1; j++ )
            if ( grades[ i ][ j ] < lowGrade )
                lowGrade = grades[ i ][ j ];
    return lowGrade;
}

int maximum( int grades[][ EXAMS ], int pupils, int tests ) {
    int i, j, highGrade = 0;
    for ( i = 0; i <= pupils - 1; i++ )
        for ( j = 0; j <= tests - 1; j++ )
            if ( grades[ i ][ j ] > highGrade )
                highGrade = grades[ i ][ j ];
    return highGrade;
}

double average( int setOfGrades[], int tests ) {
    int i, total = 0;
    for ( i = 0; i <= tests - 1; i++ )
        total += setOfGrades[ i ];
    return ( double ) total / tests;
}

void printArray( int grades[][ EXAMS ], int pupils, int tests ) {
    int i, j;
    printf( "                 [0]  [1]  [2]  [3] 
" );
    for ( i = 0; i <= pupils - 1; i++ ) {
        printf( "studentGrades[%d]  ", i );
        for ( j = 0; j <= tests - 1; j++ )
            printf( "%-5d", grades[ i ][ j ] );
        printf( "\n" );
    }
}
```

Program Output

The array:

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td>68</td>
<td>86</td>
<td>73</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>99</td>
<td>74</td>
</tr>
<tr>
<td>70</td>
<td>90</td>
<td>86</td>
<td>81</td>
</tr>
</tbody>
</table>

Lowest grade: 68
Highest grade: 97
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

Each row is a particular student, each column is the grades on the exam.