Arrays in C

Based on slides © Pat Palmer
Modified by Diana Palsetia

Changeable Data - Variables

One variable holds one value
- It occupies storage in memory
- The contents of the storage = the value of the variable
- There is a symbol (name) associated with its address
- The value (contents) may be changed by the program instructions

To keep track of many things, you need many variables
- E.g. int data1; int data2; int data3;
- All of these variables need to have names

What if you need to keep track of hundreds or thousands of values?

Multiple Changeable Values

Array
- Lets you associate one name with a lot of variables of same type

Example
- Locate by numerical index between 0 and array size minus 1 below
  is an array of type int, declared as:

    int randoms[10];

    0 1 2 3 4 5 6 7 8 9
    randoms 12 43 6 83 14 -57 109 12 0 6

Indexing into Arrays

To reference a single array element use arrayname [ index ]
for example:
    randoms[4] contains the value 14
    randoms[9] contains the value 6

Indexed elements can be used just like simple variables
- you can access their values
- you can modify their values

An array index is sometimes called a subscript

    0 1 2 3 4 5 6 7 8 9
    randoms 12 43 6 83 14 -57 109 12 0 6
Using Array Elements

randoms
0 1 2 3 4 5 6 7 8 9
12 43 6 83 14 -57 109 12 0 6

code examples:
• x = randoms[1]; /* sets x to 43 */
• randoms[4] = 99; /* replaces 14 with 99 */
• m = 5;
  y = randoms[m]; /* sets y to -57 */
• z = randoms[randoms[9]]; /* sets z to 109 */

Declaring Array Size

When you declare an array, its maximum size must be known

For example, you can declare:
char columns[80];

It's better to avoid using a "magic number":
#define MAXCOLUMNS 80
char columns[MAXCOLUMNS];

Every Array has One Type

An array may hold any type of value

All values in an array must be the same type

For example, you can have:
int week_days[7];
char small_letters[26];
float real_nums[10];

You can also have an array of arrays:
int matrix[2][3];

A program using an array

#define MAX_ELEMS 5
int main()
{
    float randoms[MAX_ELEMS];
    float total;
    float average;
    int ind; /* loop counter */
    /* initialize the array */
    randoms[0] = 34.0;
    randoms[1] = 27.0;
    randoms[2] = 45.0;
    randoms[3] = 82.0;
    randoms[4] = 22.0;
    total = 0.0;
    for (ind=0; ind<MAX_ELEMS; ind++)
    {
        total = total + randoms[ind];
    } /* loop counter */
    average = total / (float)MAX_ELEMS;
    printf("Total %f, Average %f\n",
           total, average);
    return 0;
}

Total 210.000000, Average 42.000000
Array Assignment?

Many languages let you assign the contents of one array to another.
- But C does not allow this.

You have to use:
- A for loop and go through the array yourself, explicitly copying each element between two arrays.

Arrays Length

No run-time length information

- C does not track length of arrays
- No Java-like `values.length` construct
- Thus, you need to pass length or use a sentinel.

```c
int average(int values[], int size)
{
    int index, sum;
    for (index = 0; index < size; index++) {
        sum = sum + values[index];
    }
    return (sum / size);
}
```

Passing

C passes arrays by address

- The address of the array (i.e., of the first element) is written to the function’s activation record.
- Otherwise, would have to copy each element.

```c
type main()
{
    int numbers[MAX_NUMS];
    ...
    mean = average(numbers, MAX_NUMS);
    ...
}
type average(int values[], int size)
{
    int index, sum = 0;
    for (index = 0; index < size; index++) {
        sum = sum + values[index];
    }
    return (sum / size);
}
```

Strings

C does not have a native string type

- Strings are faked in C using `arrays of char`
- String literals (constants) are stored as arrays.
- The last used element in a string literal (array) is always \0 (NUL).
- The \0 value marks the end of a string within the full array.

```
char mychars[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```c
0 1 2 3 4 5 6 7 8 9
72 101 108 108 111 0 ? ? ? ?
```

`'H' 'e' 'l' 'l' 'o' '\0'`
## Storage of a String Literal

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>101</td>
<td>108</td>
<td>108</td>
<td>111</td>
<td>32</td>
<td>87</td>
<td>111</td>
<td>114</td>
<td>108</td>
<td>100</td>
<td>10</td>
<td>0</td>
<td>?</td>
</tr>
</tbody>
</table>

'H' 'e' 'l' 'l' 'o' ' ' 'W' 'o' 'r' 'l' 'd' 'n' 'D'  

- This control character (newline) was part of the string to be printed out.
- NUL marks the end of the string in storage.