Composite Data Types
3/12/01 Lecture #14 16.070

• Many programs operate on large amounts of data
• Useful to store/manipulate data in logical manner
• Two techniques for representing collection of related data types
  ➢ Arrays -- to handle collection of data of same data type
  ➢ Structures -- to handle collection of data of different data types
Arrays

- An array is a collection of identically typed variables stored contiguously in memory.
- An array is made up of a fixed number of elements, which are referenced by subscripts.
- The basic purpose of an array is to store large amounts of related data that share the same data type.
  - Example: Analyze temperature fluctuations over the course of the year
    - Store average temperature for each day
    - Requires 365 memory locations
    - Declare 365 variables, each with a unique name?
    - In linear algebra, use a one-dimensional matrix (vector): t(1), t(2), …, t(365)
Array Declaration in C

- Array Declaration defines a template that describes the number of elements, and the type of the elements:
  
  Format: `<type> <array-name>[<num-of-elements>]`;

  ➢ Example: `int daily_temp[365];`

  Memory now is allocated – storage space for 365 integer values for Array daily_temp

  ➢ Example: `float sc_torque[3];`

  Memory now is allocated – storage space for 3 float values for Array sc_torque

- Memory size is determined by array type and number of elements

- Array Declaration defines a variable object, and allocates memory of the specified range and of the specified type for the elements
Array Subscripts

- Array elements are referenced by subscript, also known as index
  - \texttt{sc\_torque[0]} – the first element in array \texttt{sc\_torque}
  - \texttt{sc\_torque[1]} – the second
  - \texttt{sc\_torque[2]} – the third

- Subscripts must be discrete (integers) and within the range defined in the Array Declaration Statement
  
  ➢ Example: \texttt{sc\_torque[1] = 1.1;}
  
  Sets the second element of \texttt{sc\_torque} equal to 1.1
  
  ➢ Example: \texttt{x = sc\_torque[0] + 1.0;}
  
  Reads the value stored in first element of \texttt{sc\_torque}, adds 1 to it, and stores result into variable \texttt{x}

- Subscripts can be any legal C expression of integer type
  
  ➢ \texttt{sc\_torque[y+1] = sc\_torque[y] + 2.0;}
  
  ➢ Remember: C provides zero-based subscripting! First subscript is 0, not 1.
Initializing Arrays

- Arrays, just like variables, should be initialized

- Without initialization, array elements contain random data - garbage
  - Initialization can be defined at the time of declaration, and will be performed at compile time:
    ```
    float sc_torque[3] = {0.0, 0.0, 0.0};
    ```
  - Initializing more values than elements will cause compiler error
  - Initializing fewer values than elements, remaining elements are initialized to zero
  - If no array size is specified, compiler defines array size based on number of initial values
  - Initialization can also be done at run time, e.g., with a loop:
    ```
    for (j = 0; j < 3; j++)
        sc_torque[j] = 0.0;
    ```

- Consequences for not initializing arrays?
How Arrays are Stored in Memory

• Consider the 3-element float array sc_torque

Example 1:
```c
float sc_torque[3] = {1.0, 1.5};
```

Example 2:
```c
float sc_torque[3];
sc_torque[0] = 1.0;
sc_torque[1] = 1.5;
```

<table>
<thead>
<tr>
<th>Element</th>
<th>Address (in hex)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc_torque[0]</td>
<td>100</td>
<td>1.0</td>
</tr>
<tr>
<td>sc_torque[1]</td>
<td>108</td>
<td>1.5</td>
</tr>
<tr>
<td>sc_torque[2]</td>
<td>110</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>118</td>
<td></td>
</tr>
</tbody>
</table>
Array Attributes

- C defines attribute function `sizeof` that can be used to determine the bounds of an array

  - `sizeof(<array-name>)` returns the size of the array in bytes
    - `sizeof(sc_torque)` evaluates to 24 - three 8-byte floats

  - `sizeof(<array-name>[<element number>])` returns the size of the element in bytes
    - `sizeof(sc_torque[0])` evaluates to 8

- Can you use `sizeof` to determine the number of elements in an array?
  - `num_elements = (sizeof(sc_torque)) / (sizeof(float))`
Operations on Arrays

- Operations performed on arrays are ones that can be performed on any
  variable whose type is the same as the array

/* print # of days for each month for non-leap years */

#include <stdio.h>

int main (void)
{
    const int months = 12;
    int days[months] = {31,28,31,30,31,30,31,31,30,31,30,31};
    int index;

    for (index = 0; index < months; index++)
        printf ("Month %d has %d days.\n", index+1, days[index]);
    /* end for */

    return 0;
}
Example of Array Operations

- Add two vectors, represented by arrays, by adding corresponding elements from each array to calculate the sum

```c
#define Vector_size 6

int main (void)
{
    int i;
    int j;
    int VectorA[Vector_size];
    int VectorB[Vector_size];
    int VectorSum[Vector_size];
    ...
    /* input Vectors A and B */
    ...
    for (i = 0; i < Vector_size; i++)
        Vector_Sum[i] = VectorA[i] + VectorB[i];
    }
    ...
```
Exceeding Array Boundaries

- It is the responsibility of the programmer to make sure that array subscripts are within bounds!

- Exceeding array subscript bounds will overwrite other memory locations, often producing garbage output -- a difficult error to debug
  - Compiler does no checking to determine if a subscript is within the bounds of an array
  - Exception: Compiler will report an error if number of values in a compile-time initialization statement exceeds the number of elements in the array

```c
float sc_torque[3] = {1.0, 1.5, 2.0, 2.5}; /*compiler-detected error*/
```

- Example of error not detected by compiler

```c
int sc_torque[3];
for (j = 1; j <= 3; j++)
    sc_torque[j] = 0.0;
```
Structures

• A **structure** is a collection of related data values that can be of different types
• A structure is made up of **components** or **members**, which can be of different types
• Components have names instead of subscript values
• A structure is a programmer-defined data type (unlike arrays); the programmer defines names of components, types of components, and order of components

➤ Example: Characterize reaction wheel behavior
  – Determine power state: 0 = off, 1 = on
  – Identify torque for each axis: roll, pitch, yaw
  – Read speed command: 0 = 0 speed, 255 = full speed (e.g., 2000rpm)
Declaring Structure Data Types

• Declaring a structure defines a template that describes the format of the structure, and the name and format of each component:

  Format: struct <struct_name>
  {
    <type component_name1>
    <type component_name2>
    ...
  };

  Example: struct rw_struct
  {
    int power;
    float roll_torque;
    float pitch_torque;
    float yaw_torque;
    int cmd;
  };

• A structure declaration is the master plan that describes how the structure is put together
Declaring Structure Variables

- Structure Declaration does not allocate memory or create variables
- Only provides template to specify name of each field and type of information stored in each field
- Structures are created by declaring variables of type `struct`:

  ```
  struct rw_struct rw1;
  ```

- Memory now is allocated – storage space for 5 distinct components for structure `rw1`:
  - `power`: 4 bytes allocated
  - `roll_torque`: 8 bytes allocated
  - `pitch_torque`: 8 bytes allocated
  - `yaw_torque`: 8 bytes allocated
  - `cmd`: 4 bytes allocated
Assigning Values to Members

- Individual members can be assigned a number of different ways

- Initialize during compile time
  ```c
  struct rw_struct rw1 = {0, 0.0, 5.1, 0.0, 0};
  ```

- Assign during run time using the structure member operator '.
  ```c
  rw1.power = 1;
  rw1.roll_torque = 0.0;
  rw1.pitch_torque = 5.1;
  rw1.yaw_torque = 0.0;
  rw1.cmd = 255;
  ```
Assigning Values to Members – cont.

- Structures can be manipulated as whole objects

  ➢ If two structures are of the same type, the entire contents of one structure can be copied into the other record. For example, declare

    ```
    struct rw_struct rw1, rw2;
    
rw1.power = 1;
    rw1.roll_torque = 0.0;
    rw1.pitch_torque = 5.1;
    rw1.yaw_torque = 0.0;
    rw1.cmd = 255;
    
rw2 = rw1;  // rw1 and rw2 now both contain the same data
    ```
Structures Used in Functions

- Structures can be passed as arguments to functions
  ```c
  float total_torque (struct rw_struct rw1,
                      struct rw_struct rw2)
  {
    float roll_torque = 0.0;
    ...
    roll_torque = rw1.roll_torque + rw2.roll_torque;
    ...
  }
  ```

- Structures can also be returned from functions
  ```c
  struct rw_struct wheel1 (void)
  {
    struct rw_struct rw1;

    /* update values in rw1 */
    return rw1;
  } /* end function wheel1 */
  ```
Review

• Arrays and structures can be used to store, organize and manipulate large amounts of data

<table>
<thead>
<tr>
<th>Feature</th>
<th>Array</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-parts</td>
<td>Elements</td>
<td>Components/members</td>
</tr>
<tr>
<td>Type of subparts</td>
<td>Homogeneous</td>
<td>Heterogeneous</td>
</tr>
<tr>
<td>How subpart referenced</td>
<td>Subscript number(s)</td>
<td>Component Name</td>
</tr>
</tbody>
</table>

• Readings
  ➢ Review today's lecture with read sections C11.1-11.4 and chapter C12
  ➢ For Wednesday, read sections C10.1-10.5 on Pointers
  ➢ For Friday, read sections C11.8-11.10