Data Structures

- Holds a collection of data
- several individual variables
  - each stores its own value
  - each is independent of the other
  - but the group can be allocated as a whole

Many Data Structures

- arrays, stacks, queues
- structures
- linked list, tree, hash table, graph, ...

Classifications

- Homogeneous data structure vs Heterogeneous data structure
- Statically sized or Dynamically sized
- Single Unit or Linked Data Structures

Array Indexing

C provides a special syntax for accessing cells in an array
- Allocation of space for array named a:
  ```
  int a[5];
  ```
  This creates five int variables "named" a[0], a[1], a[2], a[3], a[4]
- To modify contents of cell #2 to 6 and cell #1 to 74:
  ```
  a[2] = 6;
  a[1] = 74;
  ```
- To use the contents of cell #2 and cell #1:
  ```
  printf("value = %d\n", a[1]-a[2]);
  ```
This access mechanism to the individual elements is called array indexing
- In Java / C / C++, array cells are indexed beginning at 0 and going up to n-1 (n is number of cells)
- Beware: start at 0! and end at one less than the size!!
Square Brackets: [ ] and specifying the length

- Two independent uses in C:
  - Array object creation
    ```
    int a[10];
    ```
  - Array indexing
    ```
    a[0]
    ```
  - Nothing in the array or about the array to indicate size of the array after it is created
  - Often we use a symbolic constant to specify the size of the array
    ```
    #define SIZE 10
    ```
    ```
    int main(){
        int a[SIZE];
    }
    ```

Summary of Arrays

- Arrays are:
  - Sequences of cells holding values of the same type ("base type")
  - All arrays are declared and access based on that base type — all arrays are of a type
- To define an array variable:
  ```
  int a[SIZE]; //an array with base type int
  ```
  Assuming SIZE was previously #defined as 10
- To access individual array cells: use indexing
  ```
  a[0], a[1], ..., a[9]
  ```
  - Cells are just like variables:
    - They may be read:
      ```
      x = a[3];
      ```
    - They may be written:
      ```
      a[2] = 7;
      ```

A Common Programming Idiom

To process all elements in array a...

- Do following:
  ```
  int j = 0;
  for (j = 0; j < SIZE; j++){
      //process the one element at a[j]...
  }
  ```
- Remember:
  - The lowest index is always 0
  - So use $i < \text{SIZE}$, not $i <= \text{SIZE}$
Examples

- filling an integer array using scanf
- counting elements in a character array
- adding and averaging values in a floating point array

Array Initializers

- Use curly braces and commas to list elements the array contains at startup:

```c
#include <stdio.h>

#define SIZE 10

int main(){
    int arr1[5] = {3,2,1,8,4};
    char arr2[3] = {'J','a','n'};
    float arr3[4] = {2.4,3.6,1,5};
```

- If there are too many values given, compilation error is given.
- If there are too few values given, the array elements for which no value is given are assigned 0.
- If no array size is specified, it will assume you meant the exact number of the number of initializing values.

Passing Arrays as arguments

- The name of an array is a reference to (address of) the space where the array is stored
- the [] dereference that address and add the offset determined by the value inside the brackets
- an array argument is always a copy of that address
- so when you modify the array argument, you do modify the original array
- An array contains elements of a base type
- The element of the array is treated like that base type if passed
Parallel Arrays

- Multiple arrays used to store related data
  
  ```
  int id[MAXSIZE];
  float score[MAXSIZE];
  char grade[MAXSIZE];
  ```

- Index determines which person's information:
  
  - Student with id 91 has an 82.22 average which is calculated to be a B

2-Dimensional Arrays

- Declared with two dimensions of size
  - Rows and columns
  - Two pairs of square brackets
  - Indexed with two individual indexes

  ```
  #define ROWMAX 5
  #define COLMAX 6
  int main()
  
  int matrix[ROWMAX][COLMAX];

  for (row = 0; row < ROWMAX; row++)
      for (col = 0; col < COLMAX; col++)
          matrix[row][col] = row + col;

  ```

Array Initializers

- Two methods
  - Indicating rows and columns
  - As one list of values

  - Like single dimensional arrays — uninitialized values set to 0 when an initializer is used
n-Dimensional Arrays

- Three or more dimensions
- #define SIZE1 5
- #define SIZE2 3
- #define SIZE3 4

... int arr[SIZE1][SIZE2][SIZE3];
for (row = 0; row < SIZE1; row++) {
    for (col = 0; col < SIZE2; col++) {
        for (depth = 0; depth < SIZE3; depth++) {
            printf("%.2f  ", arr[row][col][depth]);
        }
        printf("\n");
    }
    printf("----------------------------------------\n");
}

Passing to Functions

- In the Parameter List
  - For a single dimensional array – the size does not need to be indicated in the parameter list.
  - For a multi dimensional array – only the first dimension of size can be omitted in the parameter list.

- In the Argument List
  - For the argument when passing a whole array – no indices are given at all.
  - For the argument when passing a single element – all indices need to be indicated (in the correct range).

- Can not be used as the return type of a function.

In class Example

- Fill a SIZExSIZE matrix
- Print a SIZExSIZE matrix
- multiply two SIZExSIZE matrices
- in both orders to show that it isn’t commutative