"First solve the problem. Then write the code."

- J. Johnson

CSE102
Computer Programming with C

2017-2018 Spring Semester

Arrays

Largely adapted from J.R. Hanly, E.B. Koffman, F.E. Sevilgen, and others…

Declaring Arrays

- Simple memory types: single memory cell
- Group of related data items: adjacent memory cells
  - Array: uses consecutive area in memory
    - Can be referenced as a group
  - Array elements: each data item
    - Can be accessed individually

Example: `double x[8];`
  - Name of the array is "x"
  - There are eight elements (memory cells)
  - Each element is double

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Declaring Arrays

```
double x[8];
```
Declaration of an array with 8 elements of type double

```
x[0] = 1;
x[1] = 2;
x[2] = x[0] + x[1];
…
x[7] = x[5] + x[6];
```
Each element can be accessed individually

- x[5] is a subscripted variable
- 5 is a array subscript
  - Any integer
  - From 0 to 7 !!!

```
printf("%.2f", x[0]);
x[3] = 12.20;
sum = sum +x[5];
x[2] = 13 + x[0];
x[7] = pow(x[1],x[4]);
scanf("%lf", &x[0]);
```

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Example: Student Records

```
#define NUM_STUDENTS 50

int id[NUM_STUDENTS];
double gpa[NUM_STUDENTS];
```

- Parallel arrays
  - id[i] and gpa[i] are related
    - First student's ID is in id[0]
    - First student's GPA is in gpa[0]

id[0]	5503	gpa[0]	2.71
id[1]	4556	gpa[1]	3.09
id[2]	5691	gpa[2]	2.98
	. . .		. . .
id[49]	9146	gpa[49]	1.92

Example: Grading Program

```
#define NUM_QUEST 10
#define NUM_CLASS_DAYS 5

typedef enum
    {monday, tuesday, wednesday, thursday, friday}
class_days_t;

char answers[NUM_QUEST];
Int score[NUM_CLASS_DAYS];
```

answer[0]	T	score[monday]	9
answer[1]	F	score[tuesday]	7
answer[2]	F	score[wednesday]	5
	. . .	score[thursday]	3
answer[9]	T	score[friday]	1

Declaring Arrays

- More than one array can be declared at once

```
double bolts[20], needle, pins[10];
```

- An array can be initialized in declaration

```
int primes[5] = {2, 3, 5, 7, 11};
int primes[] = {2, 3, 5, 7, 11};
```

Syntax:

```
element_type array_name[size];
element_type array_name[size] = {initialization list};
```


Array Subscripts

- Subscript specifies array elements
  - Any expression if type int
  - Must be between 0 to size-1

x[10] may result in a run-time error, more likely to print incorrect results.

- Syntax

  array_name[subscript]

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

EX:

```
i = 5;
x[i-2] = x[i]-2;
x[2*i] = x[i--];
i = (int)x[(int)x[3+1]];
```

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Array Subscripts

TABLE 8.2 Code Fragment That Manipulates Array x

Statement	Explanation
`i = 5;`	
`printf("%d %.1f", 4, x[4]);`	Displays 4 and 2.5 (value of x[4])
`printf("%d %.1f", i, x[i]);`	Displays 5 and 12.0 (value of x[5])
`printf("%.1f", x[i] + 1);`	Displays 13.0 (value of x[5] plus 1)
`printf("%.1f", x[i] + i);`	Displays 17.0 (value of x[5] plus 5)
`printf("%.1f", x[i + 1]);`	Displays 14.0 (value of x[6])
`printf("%.1f", x[i + i]);`	Invalid. Attempt to display x[10]

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Array Subscripts

`printf("%.1f", x[2 * i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f", x[2 * i - 3]);`	Displays -54.5 (value of `x[7]`)
`printf("%.1f", x[(int)x[4]]);`	Displays 6.0 (value of `x[2]`)
`printf("%.1f", x[i++]);`	Displays 12.0 (value of `x[5]`); then assigns 6 to `i`
`printf("%.1f", x[--i]);`	Assigns 5 (6 - 1) to `i` and then displays 12.0 (value of `x[5]`)
`x[i - 1] = x[i];`	Assigns 12.0 (value of `x[5]`) to `x[4]`
`x[i] = x[i + 1];`	Assigns 14.0 (value of `x[6]`) to `x[5]`
`x[i] - 1 = x[i];`	Illegal assignment statement

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Using for loops to access arrays

- Processing elements of an array in sequence
- Ex: Array of squares

```
int square[11], i;
for (i = 0; i < 11; i++)
  square[i] = i * i;
```

Array square

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	1	4	9	16	25	36	49	64	81	100

- Ex: Sum of scores

```
sum_score = 0;
for(today = monday; today <= friday; ++today)
  scanf("%d", &score[today]);
for(today = monday; today <= friday; ++today)
  sum_score += score[today];
```


Program to Print a Table of Differences

```
1.  /*
2.   * Computes the mean and standard deviation of an array of data and displays
3.   * the difference between each value and the mean.
4.   */
5.
6.  #include <stdio.h>
7.  #include <math.h>
8.
9.  #define MAX_ITEM  8  /* maximum number of items in list of data              */
10.
11. int
12. main(void)
13. {
14.       double x[MAX_ITEM],     /* data list                                    */
15.              mean,            /* mean (average) of the data                   */
16.              st_dev,          /* standard deviation of the data               */
17.              sum,             /* sum of the data                              */
18.              sum_sqr;         /* sum of the squares of the data               */
19.       int    i;
20.
21.         /* Gets the data                                                      */
22.         printf("Enter %d numbers separated by blanks or <return>s\n> ",
23.                MAX_ITEM);
24.         for  (i = 0;  i < MAX_ITEM;  ++i)
25.             scanf("%lf", &x[i]);
```

(continued)

Program to Print a Table of Differences

```
26.       /* Computes the sum and the sum of the squares of all data            */
27.       sum = 0;
28.       sum_sqr = 0;
29.       for   (i = 0;  i < MAX_ITEM;  ++i) {
30.             sum += x[i];
31.             sum_sqr += x[i] * x[i];
32.       }
33.
34.       /* Computes and prints the mean and standard deviation                */
35.       mean = sum / MAX_ITEM;
36.       st_dev = sqrt(sum_sqr / MAX_ITEM - mean * mean);
37.       printf("The mean is %.2f.\n", mean);
38.       printf("The standard deviation is %.2f.\n", st_dev);
39.
40.       /* Displays the difference between each item and the mean             */
41.       printf("\nTable of differences between data values and mean\n");
42.       printf("Index       Item      Difference\n");
43.       for  (i = 0;  i < MAX_ITEM;  ++i)
44.           printf("%3d%4c%9.2f%5c%9.2f\n", i, ' ', x[i], ' ', x[i] - mean);
45.
46.       return (0);
47. }
```

Program to Print a Table of Differences

```
Enter 8 numbers separated by blanks or <return>s
> 16  12  6  8  2.5  12  14  -54.5
The mean is 2.00.
The standard deviation is 21.75.

Table of differences between data values and mean
Index       Item      Difference
  0        16.00          14.00
  1        12.00          10.00
  2         6.00           4.00
  3         8.00           6.00
  4         2.50           0.50
  5        12.00          10.00
  6        14.00          12.00
  7       -54.50         -56.50
```


Array Elements as Function Arguments

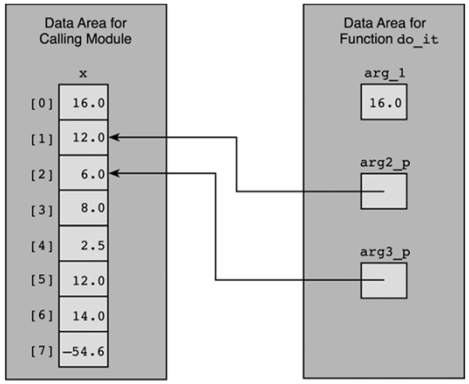
- Array elements can be arguments to functions
  - As other variables
  - Input argument

    printf("%d", a[1]);
  - Output argument

    scanf("%d", &a[1]);
  - Input/output argument

```
void do_it(double arg1, double *arg2_p , double *arg3_p);
do_it(p, &r, &s);
do_it(x[0], &x[1], &x[2]);
```


Data Area for Calling Module and do_it



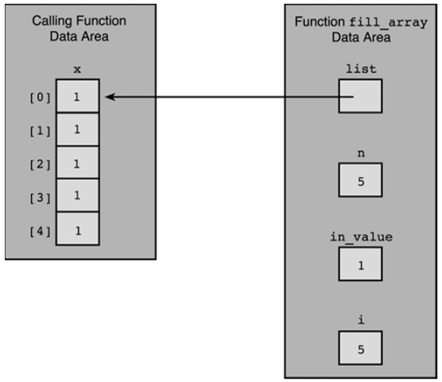
Array Arguments

- Passing whole arrays to functions
  - Array as a actual parameter
    - array name without subscript in the argument list
  - Formal parameter is the address of the first array element
    - Use subscript to access array's elements
    - Work on the original array not on a copy!...

Ex: Fill an array with the same value

```
void fill_array(int list[], int n, int in_value);
fill_array(x, 5, 1)
fill_array(&x[0], 5, 1)
```

Data Areas for fill_array (x, 5, 1);



Function fill_array

```
1.  /*
2.   * Sets all elements of its array parameter to in_value.
3.   * Pre:  n and in_value are defined.
4.   * Post: list[i] = in_value, for 0 <= i < n.
5.   */
6.  void
7.  fill_array (int list[],    /* output - list of n integers                */
8.              int n,         /* input - number of list elements            */
9.              int in_value)  /* input - initial value                      */
10. {
11.
12.       int i;            /* array subscript and loop control             */
13.
14.       for  (i = 0;  i < n;  ++i)
15.           list[i] = in_value;
16. }
```


Array Arguments

- You can use *list instead of list[] in a formal parameter list
  - Pass an array as a argument
  - int list[]; means parameter is an array
  - int *list; is correct as well
    - Array argument: passing the address of the first element
    - But, it does not show that the argument is an array!
    - You should remember that it is array not output parameter
  - What if the array is only input parameter
    - Use the const qualifier
    - You can not modify const parameters, otherwise the compiler will mark as an error

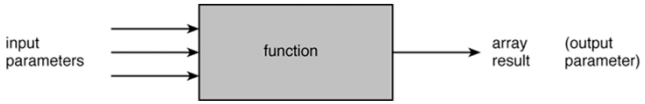
  Ex: Finding max element in an array
    - You do not need to modify array elements
    - It is safer to use const qualifier

Find the Largest Element

```
1.  /*
2.   *  Returns the largest of the first n values in array list
3.   *  Pre:  First n elements of array list are defined and n > 0
4.   */
5.  int
6.  get_max(const int list[], /* input - list of n integers                  */
7.          int       n)      /* input - number of list elements to examine */
8.  {
9.       int i,
10.          cur_large;      /* largest value so far                       */
11.
12.      /*  Initial array element is largest so far.                       */
13.      cur_large = list[0];
14.
15.      /*  Compare each remaining list element to the largest so far;
16.          save the larger                                                */
17.      for  (i = 1;  i < n;  ++i)
18.          if (list[i] > cur_large)
19.                cur_large = list[i];
20.
21.      return (cur_large);
22. }
```

Returning Array Result

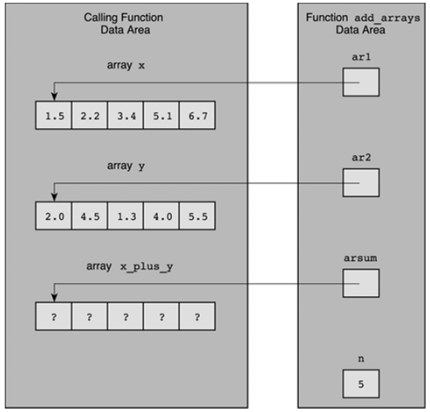
- You can not return an array as a function's return value
- You should define it as an output parameter



Ex: Adding two arrays

```
void add_arrays(const double ar1[], const double ar2[],
                                    doube arrsum[], int n);

add_arrays(x, y, x_plus_y, 5);
```


Data Areas for add_arrays(x, y, x_plus_y, 5);



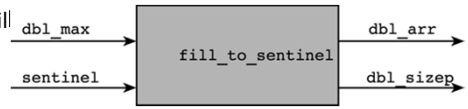
Function to Add Two Arrays

```
1.  /*
2.   *  Adds corresponding elements of arrays ar1 and ar2, storing  the result in
3.   *  arsum. Processes first n elements only.
4.   *  Pre:  First n elements of ar1 and ar2 are defined. arsum's corresponding
5.   *       actual argument has a declared size >= n (n >= 0)
6.   */
7.  void
8.  add_arrays(const double ar1[],   /* input -                                  */
9.             const double ar2[],   /*    arrays being added                    */
10.            double       arsum[], /* output - sum of corresponding
11.                                      elements of ar1 and ar2                */
12.            int          n)       /* input -  number of element
13.                                              pairs summed                   */
14. {
15.       int i;
16.
17.       /* Adds corresponding elements of ar1 and ar2                         */
18.       for  (i = 0;  i < n;  ++i)
19.           arsum[i] = ar1[i] + ar2[i];
20. }
```


Partially Filled Arrays

- Array is not completely used
  - Some part is reserved for later use
  - Need to reuse the same array for other purpose later
- Need to remember the actual number of elements in the array
  - Declared size should be larger than actual size!..

Ex: Fil



Filled Array

```
1.  /*
2.   *  Gets data to place in dbl_arr until value of sentinel is encountered in
3.   *  the input.
4.   *  Returns number of values stored through dbl_sizep.
5.   *  Stops input prematurely if there are more than dbl_max data values before
6.   *  the sentinel or if invalid data is encountered.
7.   *  Pre:  sentinel and dbl_max are defined and dbl_max is the declared size
8.   *        of dbl_arr
9.   */
10. void
11. fill_to_sentinel(int     dbl_max,    /* input - declared size of dbl_arr       */
12.                  double  sentinel,   /* input - end of data value in
13.                                                  input list                  */
14.                  double  dbl_arr[],  /* output - array of data                */
15.                  int    *dbl_sizep)  /* output - number of data values
16.                                                   stored in dbl_arr          */
17. {
18.       double data;
19.       int    i, status;
20.
```


Filled Array

```
21.       /* Sentinel input loop                                                 */
22.       i = 0;
23.       status = scanf("%lf", &data);
24.       while (status == 1  &&  data != sentinel  &&  i < dbl_max) {
25.           dbl_arr[i] = data;
26.           ++i;
27.           status = scanf("%lf", &data);
28.       }
29.
30.       /* Issues error message on premature exit                              */
31.       if (status != 1) {
32.            printf("\n*** Error in data format ***\n");
33.            printf("*** Using first %d data values ***\n", i);
34.       } else if (data != sentinel) {
35.            printf("\n*** Error: too much data before sentinel ***\n");
36.            printf("*** Using first %d data values ***\n", i);
37.       }
38.
39.       /* Sends back size of used portion of array                            */
40.       *dbl_sizep = i;
41. }
```


Driver for Testing fill_to_sentinel

```
1.  /* Driver to test fill_to_sentinel function */
2.
3.  #define A_SIZE  20
4.  #define SENT   -1.0
5.
6.  int
7.  main(void)
8.  {
9.        double arr[A_SIZE];
10.       int    in_use,     /* number of elements of arr in use */
11.              i;
12.
13.       fill_to_sentinel(A_SIZE, SENT, arr, &in_use);
14.
15.       printf("List of data values\n");
16.       for  (i = 0;  i < in_use;  ++i)
17.           printf("%13.3f\n", arr[i]);
18.
19.       return (0);
20. }
```


Stacks

- Remember stack?..
  - Only top element can be accessed
  - Operations
    - Push
    - Pop
  - Array as a stack
  - What should be the parameters to push and pop

Stacks

- Remember stack?..
  - Only top element can be accessed
  - Operations
    - Push
    - Pop
  - Array as a stack
  - What should be parameters to push and pop

    void push(char stack[], char item, int *top, int max_size);
    char pop(char stack[], int *top);

    push(s, '2', &s_top, STACK_SIZE);
    c = pop(s, &s_top);

Functions push and pop

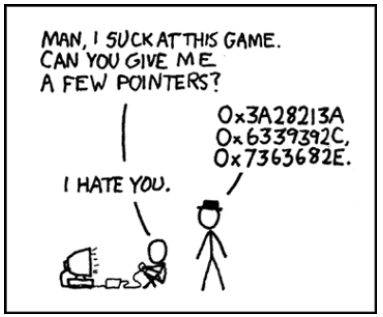
```
1.  void
2.  push(char stack[],   /* input/output - the stack */
3.       char item,      /* input - data being pushed onto the stack */
4.       int  *top,      /* input/output - pointer to top of stack */
5.       int  max_size)  /* input - maximum size of stack */
6.  {
7.       if (*top < max_size-1) {
8.           ++(*top);
9.           stack[*top] = item;
10.      }
11. }
```

(continued)

Functions push and pop

```
12. char
13. pop(char stack[],   /* input/output - the stack */
14.     int  *top)      /* input/output - pointer to top of stack */
15. {
16.      char item;     /* value popped off the stack */
17.
18.      if (*top >= 0) {
19.           item = stack[*top];
20.           --(*top);
21.      } else {
22.           item = STACK_EMPTY;
23.      }
24.
25.      return (item);
26. }
```


Pointers



http://www.i-programmer.info/component/content/article/942-cartoon.html

Searching an Array

- Two important problems in processing arrays
  - Searching: Locating a particular value
  - Sorting: Ordering the elements
- Searching: Linear search
  - Test each elements in the array one by one
  - Until the array is exhausted or the target is found

Linear Search Algorithm

1. Assume the target has not been found
2. Start with the initial array element
3. Repeat while the target is not found and there are more array elements
   4. if the current element matches the target
      5. set a flag to indicate that target found
   else
      6. Advance to the next array element
7. If the target was found
   8. return the target index as the search result

   else
   9. return -1 as the search result

Linear Search

```
1.  #define NOT_FOUND -1   /* Value returned by search function if target not
2.                           found                                              */
3.
4.  /*
5.   *  Searches for target item in first n elements of array arr
6.   *  Returns index of target or NOT_FOUND
7.   *  Pre:  target and first n elements of array arr are defined and n>=0
8.   */
9.  int
10. search(const int arr[],  /* input - array to search                         */
11.        int       target, /* input - value searched for                     */
12.        int       n)      /* input - number of elements to search           */
13. {
14.      int i,
15.          found = 0,     /*  whether or not target has been found          */
16.          where;         /*  index where target found or NOT_FOUND         */
17.
```


Linear Search

```
18.      /*  Compares each element to target                                   */
19.      i = 0;
20.      while (!found && i < n) {
21.          if (arr[i] == target)
22.                found = 1;
23.          else
24.                ++i;
25.      }
26.
27.      /* Returns index of element matching target or NOT_FOUND             */
28.      if (found)
29.            where = i;
30.      else
31.            where = NOT_FOUND;
32.
33.      return (where);
34. }
```

Sorting an Array

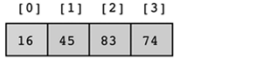
- Sorting is quite useful
  - Many operations implemented more efficiently if the data is sorted
  - Output is more understandable if the information is sorted
- Selection sort: Not very efficient but simple
  - Locate the smallest element and move it to location 0
  - Locate the smallest element in the remaining array starting with location 1 and move it to location 1
  - Locate the smallest element in the remaining array starting with location 2 and move it to location 2
  - Continue like this until location n-2

Selection Sort Algorithm

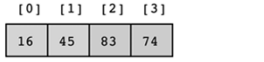
1. for each value of fill from 0 to n-2
   2. find index of the smallest element in the unsorted subarray list[fill] through list[n-1]
   3. if fill is not the position of the smallest element
      4. exchange the smallest element with the one at the position fill

Trace of Selection Sort

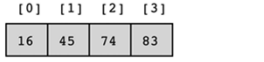
`fill` is 0. Find the smallest element in subarray `list[1]` through `list[3]` and swap it with `list[0]`.



`fill` is 1. Find the smallest element in subarray `list[1]` through `list[3]`—no exchange needed.



`fill` is 2. Find the smallest element in subarray `list[2]` through `list[3]` and swap it with `list[2]`.



Finding Minimum in a Range

```
1.  /*
2.   *  Finds the position of the smallest element in the subarray
3.   *  list[first] through list[last].
4.   *  Pre:  first < last and elements 0 through last of array list are defined.
5.   *  Post: Returns the subscript k of the smallest element in the subarray;
6.   *        i.e., list[k] <= list[i] for all i in the subarray
7.   */
8.  int get_min_range(int list[], int first, int last);
9.
10.
11. /*
12.  *  Sorts the data in array list
13.  *  Pre:  first n elements of list are defined and n >= 0
14.  */
15. void
16. select_sort(int list[],   /* input/output - array being sorted                */
17.             int n)        /* input - number of elements to sort               */
```

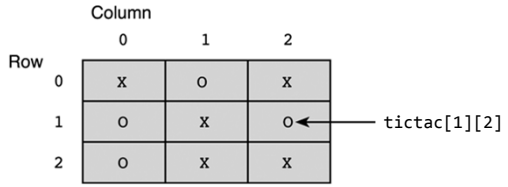
Finding Minimum in a Range

```
{
      int fill,           /* first element in unsorted subarray              */
          temp,           /* temporary storage                               */
          index_of_min;   /* subscript of next smallest element              */

      for (fill = 0; fill < n-1; ++fill) {
          /* Find position of smallest element in unsorted subarray */
          index_of_min = get_min_range(list, fill, n-1);

          /* Exchange elements at fill and index_of_min */
          if (fill != index_of_min) {
                temp = list[index_of_min];
                list[index_of_min] = list[fill];
                list[fill] = temp;
          }
      }
}
```


Multidimensional Arrays

- Array with two or more dimensions
  - Tables of data
  - Matrices
  - Tic-tac-toe board

char tictac[3][3];



Multidimensional Arrays

Syntax:

- Decleration:
  element-type aname[size1][size2]…[sizen];
- Parameter to a function
  element-type aname[][size2]…[sizen]

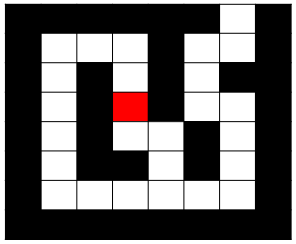
Ex:

```
#define NROWS 10
#define NCOLS  10
double table[NROWS][NCOLS];
int tt[7][5][6];

void process_matix(double table[][NCOLS], int nrows);
void process_t(int tt[][5][6], int nrows);
```


Check Whether Tic-tac-toe Board Is Filled

```
/* Checks whether a tic-tac-toe board is completely filled. */
int
filled(char ttt_brd[3][3])  /*  input - tic-tac-toe board                   */
{
      int r, c,    /* row and column subscripts    */
          ans;     /* whether or not board filled */

      /*  Assumes board is filled until blank is found                       */
      ans = 1;

      /*  Resets ans to zero if a blank is found                             */
      for  (r = 0;  r < 3;  ++r)
          for  (c = 0;  c < 3;  ++c)
             if (ttt_brd[r][c] == ' ')
                  ans = 0;

      return (ans);
}
```

2D Array for Maze Game



Initialization of Multidimensional Arrays

- Initialize like one dimensional arrays
  - Use group of values as rows

Example:

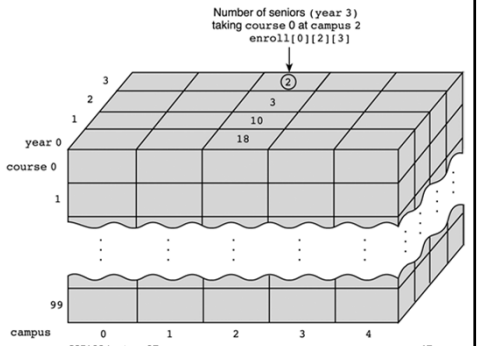
```
char tictac[3][3] = {{' ',' ',' '},{' ',' ',' '},{' ',' ',' '}};

char tictac[3][3] = {{' ',' ',' '},{' ','X',' '},{' ',' ',' '}};

char tictac[3][3] = {{'0','1','2'},{'3','4','5'},{'6','7','8'}};
```


Arrays with Several Dimentions

Three dimensional array for enrollment data

int enroll[MAXCRS][5][4];

courses:
  0 to MAXCRS-1
campuses:
  0 to 4
years:
  0 to 3



Three-Dimensional Array enroll

- Find and display the total number of students in each course
- Find and display the number of students at each campus

Case Study: Hospital Revenue

- Track revenue by unit and by quarter
  - Input: revenue transactions (in a file)
    - Unit number, quarter, revenue amount
  - Output: a table as follows

REVENUE SUMMARY

Unit	Summer	Fall	Winter	Spring	TOTAL*
Emerg	12701466.16	12663532.66	12673191.41	11965595.94	50004
Medic	12437354.59	11983744.61	12022200.48	11067640.00	47511
Oncol	16611825.25	16996019.70	15976592.83	15391817.42	64976
Ortho	16028467.82	15635498.54	15675941.06	15175890.29	62516
Psych	6589558.39	6356869.38	5860253.24	6196157.30	25003
TOTALS*	64369	63636	62208	59797	

*in thousands of dollars

Case Study: Hospital Revenue

- New types
  - quarter_t {fall, winter, spring, summer}
  - unit_t {emerg, medic, oncol, ortho, psych}
- Problem constants
  - NUM_UNITS 5
  - NUM_QUARTERS 4
- Problem inputs
  - Transaction file
  - double revenue[NUM_UNITS][NUM_QUARTERS]
- Problem outputs
  - double unit_totals[NUM_UNITS]
  - double quarter_totals[NUM_QUARTERS]

Case Study: Hospital Revenue

Algorithm:

1. Scan revenue data, posting by unit and quarter, returning a value to show success or failure of the data scan
2. It the data scan proceeded without error
   3. compute unit totals
   4. compute quarterly totals
   5. Display revenue table and row and column sums

Hospital Revenue

```
1.  /*
2.   *  Scans revenue figures for one year and stores them in a table organized
3.   *  by unit and quarter. Displays the table and the annual totals for each
4.   *  unit and the revenue totals for each quarter
5.   */
6.
7.  #include <stdio.h>
8.
9.  #define REVENUE_FILE "revenue.txt"   /* name of revenue data file    */
10. #define NUM_UNITS    5
11. #define NUM_QUARTERS 4
12.
13. typedef enum
14.       {summer, fall, winter, spring}
15. quarter_t;
16.
17. typedef enum
18.       {emerg, medic, oncol, ortho, psych}
19. unit_t;
20.
```

Hospital Revenue

```
21. int  scan_table(double revenue[][NUM_QUARTERS], int num_rows);
22. void sum_rows(double row_sum[], double revenue[][NUM_QUARTERS], int num_rows);
23. void sum_columns(double col_sum[], double revenue[][NUM_QUARTERS], int num_rows);
24. void display_table(double revenue[][NUM_QUARTERS], const double unit_totals[],
25.                    const double quarter_totals[], int num_rows);
26. /*  Insert function prototypes for any helper functions. */
27.
```


Hospital Revenue

```
28. int
29. main(void)
30. {
31.       double revenue[NUM_UNITS][NUM_QUARTERS]; /* table of revenue */
32.       double unit_totals[NUM_UNITS];           /* row totals */
33.       double quarter_totals[NUM_QUARTERS];     /* column totals */
34.       int    status;
35.
36.       status = scan_table(revenue, NUM_UNITS);
37.       if (status == 1) {
38.             sum_rows(unit_totals, revenue, NUM_UNITS);
39.             sum_columns(quarter_totals, revenue, NUM_UNITS);
40.             display_table(revenue, unit_totals, quarter_totals,
41.                           NUM_UNITS);
42.       }
43.       return (0);
44. }
```


Hospital Revenue

```
1.  /*
2.   *  Scans the revenue data from REVENUE_FILE and computes and stores the
3.   *  revenue results in the revenue table. Flags out-of-range data and data
4.   *  format errors.
5.   *  Post:  Each entry of revenue represents the revenue total for a
6.   *         particular unit and quarter.
7.   *         Returns 1 for successful table scan, 0 for error in scan.
8.   *  Calls: initialize to initialize table to all zeros
9.   */
10. int
11. scan_table(double revenue[][NUM_QUARTERS], /* output */
12.            int num_rows)                   /* input  */
13. {
14.       double    trans_amt;      /* transaction amount */
15.       int       trans_unit;     /* unit number        */
16.       int       quarter;        /* revenue quarter    */
17.       FILE     *revenue_filep;  /* file pointer to revenue file */
18.       int       valid_table = 1;/* data valid so far  */
```

(continued)

Hospital Revenue

```
19.       int       status;         /* input status       */
20.       char      ch;             /* one character in bad line */
21.
22.       /*  Initialize table to all zeros */
23.       initialize(revenue, num_rows, 0.0);
24.
```

Hospital Revenue

```
24.
25.       /*  Scan and store the valid revenue data */
26.       revenue_filep = fopen(REVENUE_FILE, "r");
27.       for  (status = fscanf(revenue_filep, "%d%d%lf", &trans_unit,
28.                             &quarter, &trans_amt);
29.             status == 3  &&  valid_table;
30.             status = fscanf(revenue_filep, "%d%d%lf", &trans_unit,
31.                             &quarter, &trans_amt)) {
32.           if (summer <= quarter  &&  quarter <= spring  &&
33.               trans_unit >= 0  &&  trans_unit < num_rows) {
34.                 revenue[trans_unit][quarter] += trans_amt;
35.           } else {
36.                 printf("Invalid unit or quarter -- \n");
37.                 printf("  unit is ");
38.                 display_unit(trans_unit);
39.                 printf(", quarter is ");
40.                 display_quarter(quarter);
41.                 printf("\n\n");
42.                 valid_table = 0;
43.           }
44.       }
45.
```


Hospital Revenue

```
46.       if (!valid_table) {          /* error already processed */
47.             status = 0;
48.       } else if (status == EOF) {  /* end of data without error */
49.             status = 1;
50.       } else {                     /* data format error */
51.             printf("Error in revenue data format. Revise data.\n");
52.             printf("ERROR HERE >>> ");
53.             for  (status = fscanf(revenue_filep, "%c", &ch);
54.                   status == 1  &&  ch != '\n';
55.                   status = fscanf(revenue_filep, "%c", &ch))
56.                 printf("%c", ch);
57.             printf(" <<<\n");
58.             status = 0;
59.       }
60.       return (status);
61. }
```

(continued)

Hospital Revenue

```
62. /*
63.  *  Stores value in all elements of revenue.
64.  *  Pre:  value is defined and num_rows is the number of rows in
65.  *        revenue.
66.  *  Post: All elements of revenue have the desired value.
67.  */
68. void
69. initialize(double revenue[][NUM_QUARTERS], /* output */
70.            int    num_rows,               /* input  */
71.            double value)                  /* input  */
72. {
73.       int       row;
74.       quarter_t quarter;
75.
76.       for  (row = 0;  row < num_rows;  ++row)
77.           for  (quarter = summer;  quarter <= spring;  ++quarter)
78               revenue[row][quarter] = value;
79. }
```


Hospital Revenue

```
8.  void
9.  display_table(double       revenue[][NUM_QUARTERS], /* input */
10.               const double unit_totals[],           /* input */
11.               const double quarter_totals[],        /* input */
12.               int          num_rows)                /* input */
13. {
14.       unit_t    unit;
15.       quarter_t quarter;
16.
17.       /* Display heading */
18.       printf("%34cREVENUE SUMMARY\n%34c---------------\n\n", ' ', ' ');
19.       printf("%4s%11c", "Unit", ' ');
20.       for  (quarter = summer;  quarter <= spring;  ++quarter){
21.             display_quarter(quarter);
22.             printf("%8c", ' ');
23.       }
24.       printf("TOTAL*\n");
25.       printf("----------------------------------------");
26.       printf("----------------------------------------\n");
27.
```

Hospital Revenue

```
28.       /* Display table */
29.       for  (unit = emerg;  unit <= psych;  ++unit) {
30.            display_unit(unit);
31.           printf("    ");
32.           for  (quarter = summer;  quarter <= spring;  ++quarter)
33.               printf("%14.2f", revenue[unit][quarter]);
34.           printf("%13d\n", whole_thousands(unit_totals[unit]));
35.       }
36.       printf("----------------------------------------");
37.       printf("----------------------------------------\n");
38.       printf("TOTALS*");
39.       for  (quarter = summer;  quarter <= spring;  ++quarter)
40.           printf("%14d", whole_thousands(quarter_totals[quarter]));
41.       printf("\n\n*in thousands of dollars\n");
42.  }
```


Hospital Revenue

```
43.  /*
44.   *  Display an enumeration constant of type quarter_t
45.   */
46.  void
47.  display_quarter(quarter_t quarter)
48.  {
49.       switch (quarter) {
50.       case summer:  printf("Summer");
51.                     break;
52.
53.       case fall:    printf("Fall ");
54.                     break;
55.
56.       case winter:  printf("Winter");
57.                     break;
58.
59.       case spring:  printf("Spring");
60.                     break;
61.
62.       default:      printf("Invalid quarter %d", quarter);
63.       }
64.  }
```


Hospital Revenue

```
65.
66.  /*
67.   *  Return how many thousands are in number
68.   */
69.  int whole_thousands(double number)
70.  {
71.      return (int)((number + 500)/1000.0);
72.  }
```