Lecture 5. Arrays

• An array is a named collection of variables all of the same type
  Each variable in the collection is an element
  Elements are known by their integer positions or indices

    int count[11];

defines an array named count that has 11 elements with indices 0..10

• Array elements are accessed by subscripting

  count[ expression ]

  expression is any expression whose value is an integer between 0 and 10 inclusive

  Subscript expressions are variables, a.k.a. lvalues

  No bounds checking — effect of out-of-bound subscripts is undefined

• Array elements occupy successive locations in memory

• Array elements are uninitialized; use loops to initialize them

    int i, count[11];

    for (i = 0; i < 11; i++)
      count[i] = 0;
Printing a Histogram

- `scores` contains 115 exam scores between 0 and 100

```c
#include <stdio.h>

int main(void) {
    int i, counts[11], score;
    for (i = 0; i < 11; i++)
        counts[i] = 0;
    while (scanf("%d", &score) != EOF)
        counts[score/10]++;
    for (i = 10; i >= 0; i--) {
        int n = counts[i];
        printf("%3d ", 10*i);
        while (n-- > 0)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

- Use an array to hold the number of scores in each 10-point range

/*
Print a histogram of scores from 0..100 in groups of 10.
*/

```c
% lcc hist.c
% a.out <scores
100 **
  90 ********
  80 ****************
  70 ************************
  60 ****************
  50 ****************
  40 ********
  30 ***
  20 **
  10 *
  0 *
```
Dissecting hist.c

```c
int i, counts[11], score;
for (i = 0; i < 11; i++)
    counts[i] = 0;

Declares counts and initializes each of its 11 elements to 0
```

```c
while (scanf("%d", &score) != EOF)
    counts[score/10]++;

Reads the scores and increments the appropriate element of counts; scanf returns the value EOF at the end-of-file is reached (EOF is defined in stdio.h)
```

```c
for (i = 10; i >= 0; i--) {
    int n = counts[i];
    printf("%3d ", 10*i);
    while (n-- > 0)
        printf("*");  
    printf("\n");
}

Loops from counts[10] down to counts[0] printing each line of the histogram
```
Multidimensional Arrays

• Multidimensional arrays have two or more indices

```c
int x[3][5];
```
defines an 2-dimensional array x that has $3 \times 5 = 15$ elements

```
x[0][0]   x[0][1]   x[0][2]   x[0][3]   x[0][4]
\hline
x[1][0]   x[1][1]   x[1][2]   x[1][3]   x[1][4]
\hline
```

• Array *rows* occupy successive locations in memory — *row-major order*

```
x[0][0]   x[0][1]   x[0][2]   x[0][3]   x[0][4]
\hline
x[1][0]   x[1][1]   x[1][2]   x[1][3]   x[1][4]
\hline
```
Printing a Stem-and-Leaf Plot

- A **stem-and-leaf plot** displays the data values themselves in a histogram

```
% lcc stem.c
% a.out <scores
10 00
  9 97765510
  8 977655533100
  7 99877666555544433211100
  6 9887654433221000
  5 44322111111000
  4 88844444311
  3 7655211
  2 865221
  1 65311
  0 8850
```

- Use a 2-dimensional array to hold the number of times each score occurs
  
  \[
  \text{counts}[i][j] \text{ is the number of times the score } 10 \times i + j \text{ occurs}
  \]

  Each row of \text{counts} is a row in the stem plot

```c
#include <stdio.h>

int main(void) {
    int i, j, counts[11][10], score;
    for (i = 0; i < 11; i++)
        for (j = 0; j < 10; j++)
            counts[i][j] = 0;
    while (scanf("%d", &score) != EOF)
        counts[score/10][score%10]++;
    for (i = 10; i >= 0; i--) {
        printf("%2d ", i);
        for (j = 9; j >= 0; j--)
            printf("%d", j);
        printf("\n");
    }
    return 0;
}
```
Dissecting stem.c

```c
int i, j, counts[11][10], score;
for (i = 0; i < 11; i++)
    for (j = 0; j < 10; j++)
        counts[i][j] = 0;

Declares counts as a 11-by-10 array and initializes each of its 110 elements to 0

counts[score/10][score%10]++;

Increments the element of counts that holds the number of times score occurs

for (i = 10; i >= 0; i--)
    {printf("%2d ", i);
      ...
      printf("\n");
    }

Loops down the rows of counts, printing each ‘leaf’ and a new-line character

for (j = 9; j >= 0; j--)
    {int n = counts[i][j];
      while (n-- > 0)
          printf("%d", j);
    }

Loops down the i-th column in counts printing j = score%10 for each occurrence of score
```
Passing Arrays to Functions

• Array parameters are declared by omitting the array size

```c
void record(int score, int counts[]) {
    counts[score/10]++;
}
```

• Arrays are passed to functions by giving just the array name

```c
int main(void) {
    int i, counts[11], score;
    for (i = 0; i < 11; i++)
        counts[i] = 0;
    while (scanf("%d", &score) != EOF)
        record(score, counts);
    for (i = 10; i >= 0; i--) {
        printf("%3d ", 10*i);
        printf("%d ", printhist(counts[i]));
        printf("\n");
    }
    return 0;
}
```

• Arrays — and only arrays — are passed in a way that simulates call-by-reference

The callee can change elements in the caller’s array argument

An element is passed by value — the callee cannot change the caller’s element
Passing Arrays to Functions, cont’d

• Declare multidimensional array parameters by omitting only the number of rows

```c
void printstem(int counts[][10], int nrows) {
    while (--nrows >= 0) {
        int j;
        printf("%2d ", nrows);
        for (j = 9; j >= 0; j--) {
            int n = counts[nrows][j];
            while (n-- > 0)
                printf("%d", j);
        }
        printf("\n");
    }
}

int main(void) {
    int i, j, counts[11][10], score;
    for (i = 0; i < 11; i++)
        for (j = 0; j < 10; j++)
            counts[i][j] = 0;
    while (scanf("%d", &score) != EOF)
        counts[score/10][score%10]++;
    printf(counts, 11);
    return 0;
}
```

• Passing the number of rows, or array size, to functions helps avoid indexing bugs