Faculty of Computer Science, Dalhousie University *12-Oct-2018*

CSCI 2132 — Software Development

Lecture 16: Multidimensional Arrays

Location: Chemistry 125 Instructor: Vlado Keselj
Time: 12:35 – 13:25

Previous Lecture

- Software testing
- Software debugging, gdb
- Arrays in C

Example: `binary.c`

We will not look at an example of C code for binary search on a sorted sequence of numbers. We assume that the user will be asked to enter 10 integers in the ascending order. After that, we ask user to enter the number to be searched. If the number is found in the sequence, we will report which in order this number was entered, e.g., first (1.), second (2.) etc, or if it was not found, we will report that the number was not found.

The following is the corresponding code, and your task is to fill in the blanks. (This code is available in the file `~prof2132/public/binary.c`, on the `bluenose` system.)

```
/* CSCI 2132 Sample "fill-in blanks" code: binary.c */
#include <stdio.h>

#define LEN 10

int main() {
  int array[LEN], lower, upper, middle, key, i;

  printf("Enter %d numbers in ascending order:\n", LEN);

  for (i = 0; i < LEN; i++)
    scanf("%d", ________ );

  printf("Enter the number to be searched for: ");
  scanf("%d", &key);

  lower  = 0;
  upper  = LEN - 1;
  middle = (lower + upper) / 2;

  while (array[middle] != key && lower < upper) {

    if      (lower+1 == upper)    lower = __________ ;
    else if (key < array[middle]) upper = __________ ;
    else                          lower = __________ ;
    middle = (lower + upper) / 2;
  }

  if ( ____________ )
    printf("%d is the %d-th number you entered.\n", key, middle+1);
  else
    printf("Not found.\n");

  return 0;
}
```

15.2 Multidimensional Arrays

We discussed so far one-dimensional arrays. Arrays can have two, three or more dimensions, in which case we call them *multidimensional arrays.* To declare a multidimensional array, we generally need to specify the length of each of its dimensions. For example, the following line of code defines a 2D array of 5 rows and 9 columns:

```
int m[5][9];
```

2D arrays can be used to represent a matrix or a table. For example, the array `m` declared above can be used to represent a 5×9 table, whose rows are numbered 0, 1, 2, 3, and 4; and whose columns are numbered 0, 1, 2, 3, 4, 5, 6, 7, and 8.

We still use array subscripting to access an element of a multidimensional array. For example, `m[1][4]` is stored in row 1 and column 4 (recall that row and column numbers start with 0).

How is a 2D array stored in memory? We know that most computer systems use linear memory, i.e., data in memory are stored as a sequence of bytes. Thus we need to store the content of a 2D array as a sequence. The elements in a 2D array are stored in *row-major order,* which means that we store first the elements of the first row, then the second row, and so on. Take the array m for example. Its elements will be stored physically in memory in this order:

m[0][0]	m[0][1]	...	m[0][8]	m[1][0]	m[1][1]	...	m[1][8]	...	m[4][0]	m[4][1]	...	m[4][8]

The elements are stored in continuous way, without any gaps. To initialize multidimensional arrays when declaring them, we could use nested one-dimensional initializers. The following code snippet defines and initializes a 2D array to be a 3×3 identity matrix:

```
int t[3][3] = {{1, 0, 0},
               {0, 1, 0},
               {0, 0, 1}};
```

Inner bracers could be omitted. An example:

```
int z[3][3] = {0};
```

initializes the whole matrix to a zero matrix.

Variable-Length Arrays (C99)

So far we required array lengths to be constants. However, sometimes we do not know the array length in advance. In C99, variable-length arrays are introduced for this (there are no variable-length arrays in C89 or earlier). The length of a VLA is a non-constant expression. The following code snippet declares a VLA and assigns values to it:

```
int len, i;
printf("Enter the number of integers: ");
scanf("%d", &len);

int array[len];
printf("Enter %d numbers: ", len);
for (i = 0; i < len; i++)
   scanf("%d", &array[i]);
```

It is necessary to assign a value to len before we use it as the length when declaring a VLA.

A programming exercise: Rewrite the binary search program using a VLA so that the user can first enter the array length and then the values stored in the array.

VLAs can be multidimensional, but they cannot have initializers.

15.3 Latin Square Example (`latin.c`)

Let us see an example that makes use of both variable-length arrays and 2D arrays. The example is to test whether a matrix is a Latin square. An n by n matrix is a Latin square if each row is a permutation of numbers 1, 2, ..., n and each column is a permutation of the same numbers 1, 2, ..., n. Some example are:

```
1 2 3 is Latin   1 2 3 is not Latin
2 3 1 square     3 1 2 square
3 1 2            1 3 2
```

For example, if you are familiar with Sudoku puzzles, a completed Sudoku puzzle is a 9 by 9 Latin square, with some additional constraints regarding 3×3 regions of the matrix.

The most interesting part of our program is how to check whether a row, or a column, is a permutation of numbers 1, 2, ..., n; i.e., whether each number from 1, 2, ... n appears in this row exactly once. We will use an array of counters `occurs`, such that `occurs[i]` is the number of times number `i` was seen in a row or column being examined. The following is the source code of the program `latin.c`, which needs to be completed: (The file is available as `~prof2132/public/latin.c-blanks` on the `bluenose` system.)

```
/* Program latin.c */
#include <stdio.h>

int main() {
  int size = 0, i, j;

  printf("Enter the size of the latin square: ");
  scanf("%d", &size);
  if (size < 1) return 1;

  int square[size][size], occurs[size+1];

  printf("Enter a matrix (%d by %d) of integers in the range [1-%d]: \n",
         size, size, size);
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++) {
      scanf("%d", ________________ );
      if (square[i][j] > size || square[i][j] <= 0) {
        printf("Error: element out of range.\n");
        return 1;
      }
    }

  for (i = 0; i < size; i++) {
    for (j = 1; j <= size; j++)
      occurs[j] = 0;

    for (j = 0; j < size; j++) {
      if ( _________________ > 0) {
        printf("This is not a Latin square.\n");
        return 1;
      }
      else
        occurs[ ______________ ] = 1;
    }
  }

  for (j = 0; j < size; j++) {
    for (i = 1; i <= size; i++)
      occurs[i] = 0;

    for (i = 0; i < size; i++) {
      if (_____________________ > 0) {
        printf("This is not a Latin square.\n");
        return 1;
      }
      else
        occurs[ _____________ ] = 1;
    }
  }

  printf("This is a Latin square.\n");
  return 0;
}
```