Arrays

Dr Deepayan Bhowmik Dr Andrea Bracciali

In this lecture …

- Arrays !

- Other online IDE
  - https://ide.geeksforgeeks.org/index.php
  - https://www.onlinegdb.com/

Array Initialization

Arrays exist in C like in most of other languages.

- An array, like any other variable, can be given an initial value at the time it's declared.

- The most common form of array initializer is a list of constant expressions (a 10-position integer array):

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Array Initialization

- If the initializer is shorter than the array, the remaining elements of the array are given the value 0:

```
int a[10] = {1, 2, 3, 4, 5, 6};
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

- An array of all zeros:

```
int a[10] = {0};
/* initial value of a is {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} */
```

There's a single 0 inside the braces because **it's illegal** for an initializer to be **completely empty**.

- It's also illegal for an initializer to be **longer than the array** it initializes.

Array Initialization

- If an initializer is present, the length of the array may be omitted:

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- The compiler uses the length of the initializer to determine how long the array is.

Designated Initializers (C99)

- Sparse arrays: only few elements, e.g.

```
int a[15] =
  {0, 0, 29, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 48};
```

- Error-prone and tedious for a large array!

- C99's designated initializers:

```
int a[15] =
    {[2] = 29, [9] = 7, [14] = 48};
```

- Each number in brackets is said to be a designator.

Designated Initializers (C99)

- Designated initializers are shorter and easier to read (at least for some arrays).
- Also, the order in which the elements are listed no longer matters.
- Another way to write the previous example:

```
int a[15] = {[14] = 48, [9] = 7, [2] = 29};
```

Designated Initializers (C99)

- If the array being initialized has length *n*, each designator must be between 0 and *n* – 1.
- If the length of the array is omitted, a designator can be any nonnegative integer.
  - The compiler will deduce the length of the array from the largest designator.
- The following array will have 24 elements:

```
int b[] = {[5] = 10, [23] = 13, [11] = 36, [15] = 29};
```

Designated Initializers (C99)

- An initializer may use both the older (element-by-element) technique and the newer (designated) technique:

```
int c[10] = {5, 1, 9, [4] = 3, 7, 2, [8] = 6};
```

- Output:

```
c[10] = {5, 1,9,0,3,7,2,0,6,0};
```

Program: Checking a Number for Repeated Digits

- The `repdigit.c` program checks whether any of the digits in a number appear more than once.
- After the user enters a number, the program prints either `Repeated digit` or `No repeated digit`:

```
Enter a number: 28212
Repeated digit
```

- The number 28212 has a repeated digit (2); a number like 9357 doesn't.

Program: Checking a Number for Repeated Digits

- The program uses an array of 10 Boolean values to keep track of which digits appear in a number.
- Initially, every element of the `digit_seen` array is false.
- When given a number n, the program examines n's digits one at a time, storing the current digit in a variable named `digit`.
  - If `digit_seen[digit]` is true, then `digit` appears at least twice in n.
  - If `digit_seen[digit]` is false, then digit has not been seen before, so the program sets `digit_seen[digit]` to true and keeps going.

repdigit.c

```
/* Checks numbers for repeated digits */

#include <stdbool.h>   /* C99 only */
#include <stdio.h>

int main(void)
{
  bool digit_seen[10] = {false};
  int digit;
  long n;

  printf("Enter a number: ");
  scanf("%ld", &n);

  while (n > 0) {
    digit = n % 10;
    if (digit_seen[digit])
      break;
    digit_seen[digit] = true;
    n /= 10;
  }

  if (n > 0)
    printf("Repeated digit\n");
  else
    printf("No repeated digit\n");

  return 0;
}
```

Using the `sizeof` Operator with Arrays

- The `sizeof` operator can determine the size of an array (in bytes).
- If `a` is an array of 10 integers, then `sizeof(a)` is typically 40 (assuming that each integer requires four bytes).
- We can also use `sizeof` to measure the size of an array element, such as `a[0]`.
- Dividing the array size by the element size gives the length of the array:

```
sizeof(a) / sizeof(a[0])
```

Using the `sizeof` Operator with Arrays

- Some programmers use this expression when the length of the array is needed.
- A loop that clears the array `a`:

```
for (i = 0; i < sizeof(a) / sizeof(a[0]); i++)
  a[i] = 0;
```

Note that the loop doesn't have to be modified if the array length should change at a later date.

Using the `sizeof` Operator with Arrays

- Some compilers produce a warning message for the expression `i < sizeof(a) / sizeof(a[0])`.
- The variable `i` probably has type `int` (a signed type), whereas `sizeof` produces a value of type `size_t` (an unsigned type).
- Comparing a signed integer with an unsigned integer can be dangerous, but in this case it's safe…
- How to avoid this?

Using the `sizeof` Operator with Arrays

- To avoid a warning, we can add a cast that converts `sizeof(a) / sizeof(a[0])` to a signed integer:

```
for (i = 0; i < (int) (sizeof(a) / sizeof(a[0])); i++)
  a[i] = 0;
```

- Defining a macro for the size calculation can be helpful:

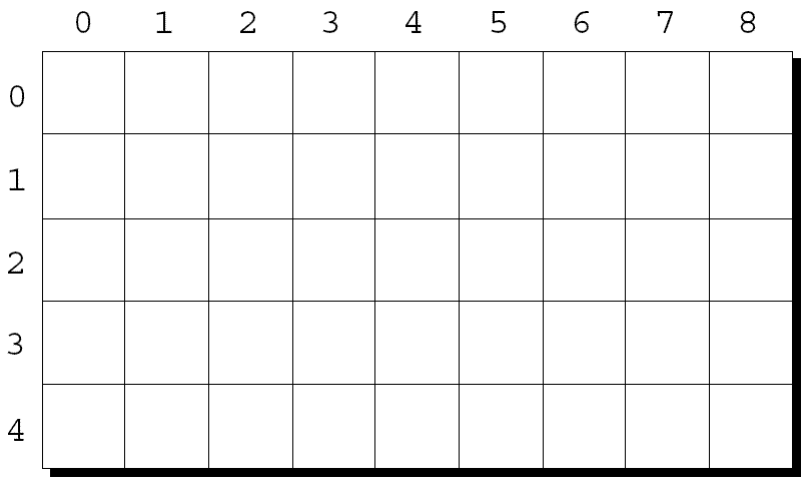
```
#define SIZE ((int) (sizeof(a) / sizeof(a[0])))

for (i = 0; i < SIZE; i++)
  a[i] = 0;
```

Multidimensional Arrays

- An array may have any number of dimensions.
- The following declaration creates a two-dimensional array (a *matrix,* in mathematical terminology):

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

- `m` has 5 rows and 9 columns. Both rows and columns are indexed from 0:

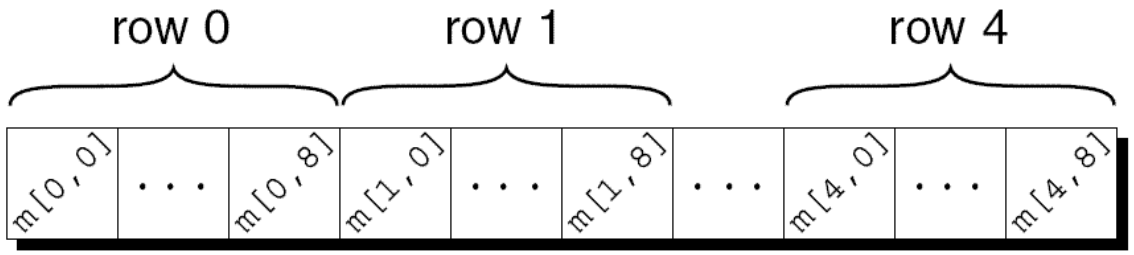


Multidimensional Arrays

- To access the element of `m` in row `i`, column `j`, we must write `m[i][j]`.
- The expression `m[i]` designates row `i` of `m`, and `m[i][j]` then selects element `j` in this row.
- Resist the temptation to write `m[i,j]` instead of `m[i][j]`.

Multidimensional Arrays

- Although we visualize two-dimensional arrays as tables, that's not the way they're actually stored in computer memory.
- C stores arrays in ***row-major order,*** with row 0 first, then row 1, and so forth.
- How the `m` array is stored:



Multidimensional Arrays

- Nested `for` loops are ideal for processing multidimensional arrays.
- Consider the problem of initializing an array for use as an identity matrix. A pair of nested `for` loops is perfect:

```
#define N 10 /* vs 'const int N = 10;'*/

int ident[N][N];
int row, col;

for (row = 0; row < N; row++)
  for (col = 0; col < N; col++){
    if (row == col)
      ident[row][col] = 1;
    else
      ident[row][col] = 0;
  }
```

Can this be made more efficient?

```
ident[N][N] = {0};
for (int i=0;i<N;i++)
ident[i][i] = 1;
```

Initializing a Multidimensional Array

- We can create an initializer for a two-dimensional array by nesting one-dimensional initializers:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

- Initializers for higher-dimensional arrays are constructed in a similar fashion.
- C provides a variety of ways to abbreviate initializers for multidimensional arrays

Initializing a Multidimensional Array

- If an initializer isn't large enough to fill a multidimensional array, the remaining elements are given the value 0.
- The following initializer fills only the first three rows of `m`; the last two rows will contain zeros:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0}};
```

Initializing a Multidimensional Array

- If an inner list isn't long enough to fill a row, the remaining elements in the row are initialized to 0:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1},
               {0, 1, 0, 1, 1, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

Initializing a Multidimensional Array

- We can even omit the inner braces:

```
int m[5][9] = {1, 1, 1, 1, 1, 0, 1, 1, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0,
               0, 1, 0, 1, 1, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 1, 1, 1};
```

  Once the compiler has seen enough values to fill one row, it begins filling the next.

- Omitting the inner braces can be risky, since an extra element (or even worse, a missing element) will affect the rest of the initializer.

Initializing a Multidimensional Array

- C99's designated initializers work with multidimensional arrays.
- How to create 2 × 2 identity matrix:

```
double ident[2][2] = {[0][0] = 1.0, [1][1] = 1.0};
```

As usual, all elements for which no value is specified will default to zero.

Constant Arrays

- An array can be made "constant" by starting its declaration with the word `const`:

```
const char hex_chars[] =
  {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
   'A', 'B', 'C', 'D', 'E', 'F'};
```

- An array that's been declared `const` should not be modified by the program.
- `const` isn't limited to arrays, but it's particularly useful in array declarations.

Program: Dealing a Hand of Cards

- The `deal.c` program illustrates both two-dimensional arrays and constant arrays.
- The program deals a random hand from a standard deck of playing cards.
- Each card in a standard deck has a *suit* (clubs, diamonds, hearts, or spades) and a *rank* (two, three, four, five, six, seven, eight, nine, ten, jack, queen, king, or ace).

Program: Dealing a Hand of Cards

- The user will specify how many cards should be in the hand:

```
Enter number of cards in hand: 5
Your hand: 7c 2s 5d as 2h
```

- Problems to be solved:
  - How do we pick cards randomly from the deck?
  - How do we avoid picking the same card twice?

Program: Dealing a Hand of Cards

- To pick cards randomly, we'll use several C library functions:
  - `time()` (from `<time.h>`) – returns the current time, encoded in a single number.
  - `srand()` (from `<stdlib.h>`) – initializes C's random number generator.
  - `rand()` (from `<stdlib.h>`) – produces an apparently random number each time it's called.
- By using the `%` operator, we can scale the return value from `rand` so that it falls between 0 and 3 (for suits) or between 0 and 12 (for ranks).

Program: Dealing a Hand of Cards

- The `in_hand` array is used to keep track of which cards have already been chosen.
  - The array has 4 rows and 13 columns; each element corresponds to one of the 52 cards in the deck.
  - All elements of the array will be false to start with.
- Each time we pick a card at random, we'll check whether the element of `in_hand` corresponding to that card is true or false.
  - If it's true, we'll have to pick another card.
  - If it's false, we'll store `true` in that element to remind us later that this card has already been picked.

Program: Dealing a Hand of Cards

- Once we have verified that a card is “new”, we will need to translate its numerical rank and suit into characters and then display the card.
- To translate the rank and suit to character form, we will set up two arrays of characters—one for the rank and one for the suit—and then use the numbers to subscript the arrays.
- These arrays won’t change during program execution, so they are declared to be `const`.

deal.c

```
/* Deals a random hand of cards */

#include <stdbool.h>   /* C99 only */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

const int NUM_SUITS 4;
const int NUM_RANKS 13;

int main(void)
{
  bool in_hand[NUM_SUITS][NUM_RANKS] = {false};
  int num_cards, rank, suit;
  const char rank_code[] = {'2','3','4','5','6','7','8',
                            '9','t','j','q','k','a'};
  const char suit_code[] = {'c','d','h','s'};
```

```
  srand((unsigned) time(NULL));

  printf("Enter number of cards in hand: ");
  scanf("%d", &num_cards);

  printf("Your hand:");
  while (num_cards > 0) {
    suit = rand() % NUM_SUITS;     /* picks a random suit */
    rank = rand() % NUM_RANKS;     /* picks a random rank */
    if (!in_hand[suit][rank]) {
      in_hand[suit][rank] = true;
      num_cards--;
      printf(" %c%c", rank_code[rank], suit_code[suit]);
    }
  }
  printf("\n");

  return 0;
}
```