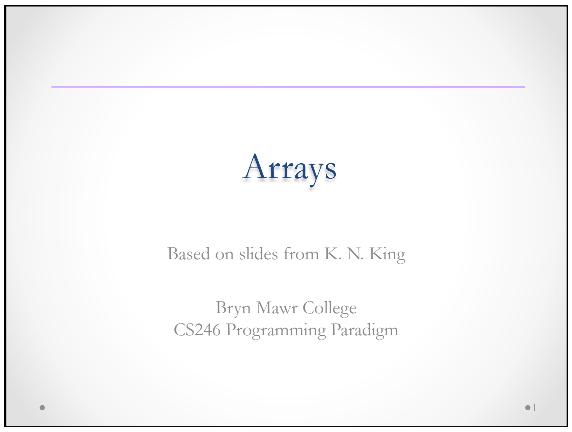
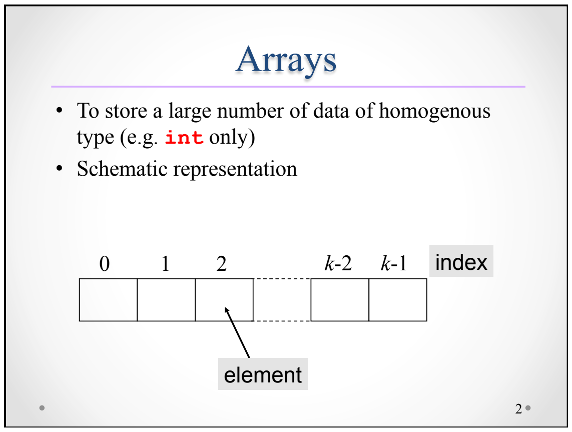


Arrays

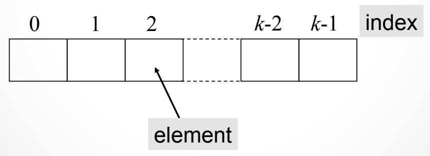
Based on slides from K. N. King

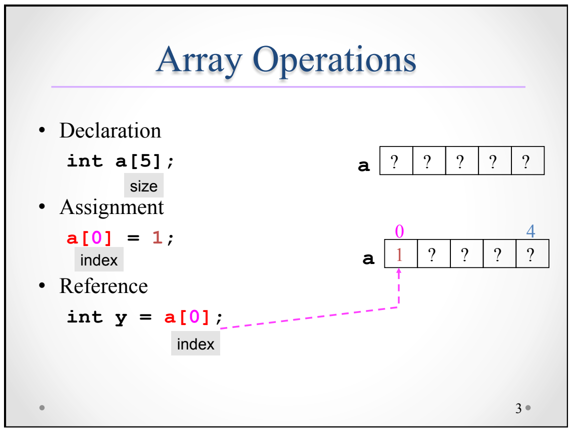
Bryn Mawr College
CS246 Programming Paradigm



Arrays

- To store a large number of data of homogenous type (e.g. `int` only)
- Schematic representation





Array Operations

- Declaration

  `int a[5];`

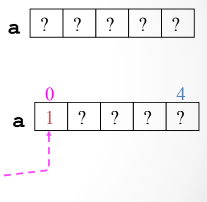
  size
- Assignment

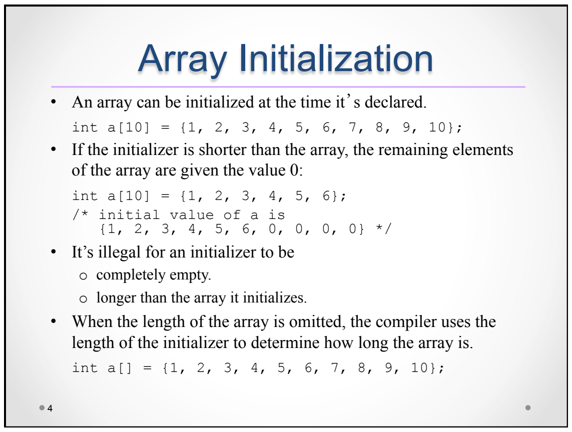
  `a[0] = 1;`

  index
- Reference

  `int y = a[0];`

  index





Array Initialization

- An array can be initialized at the time it's declared.

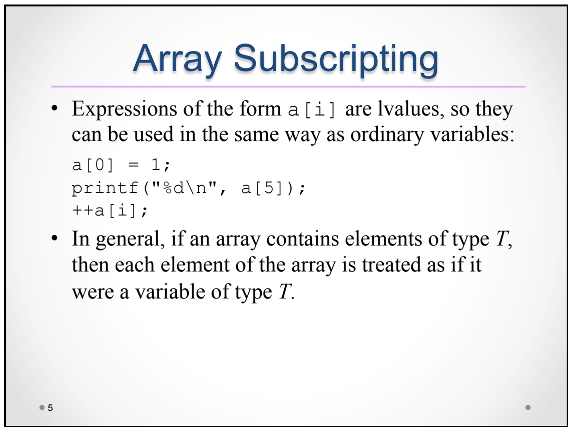
```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- 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} */
```

- It's illegal for an initializer to be
  - completely empty.
  - longer than the array it initializes.
- When the length of the array is omitted, the compiler uses the length of the initializer to determine how long the array is.

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

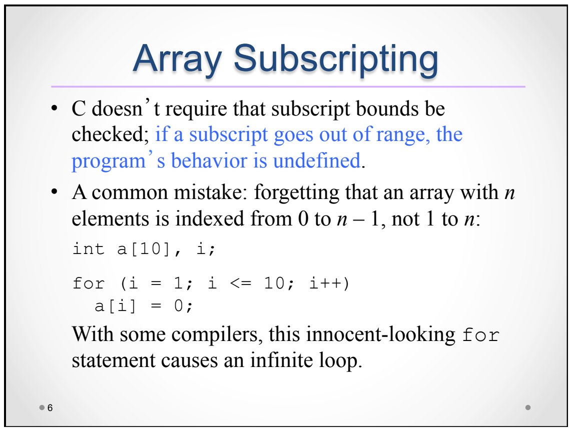


Array Subscripting

- Expressions of the form `a[i]` are lvalues, so they can be used in the same way as ordinary variables:

```
a[0] = 1;
printf("%d\n", a[5]);
++a[i];
```

- In general, if an array contains elements of type *T*, then each element of the array is treated as if it were a variable of type *T*.



Array Subscripting

- C doesn't require that subscript bounds be checked; if a subscript goes out of range, the program's behavior is undefined.
- A common mistake: forgetting that an array with *n* elements is indexed from 0 to *n* – 1, not 1 to *n*:

```
int a[10], i;

for (i = 1; i <= 10; i++)
  a[i] = 0;
```

With some compilers, this innocent-looking `for` statement causes an infinite loop.

Array Subscripting

- An array subscript may be any integer expression:

```
a[i+j*10] = 0;
```

- The expression can even have side effects:

```
i = 0;
while (i < N)
  a[i++] = 0;
```

- Be careful when an array subscript has a side effect:

```
i = 0;
while (i < N)
  a[i] = b[i++];
```

- The expression `a[i] = b[i++]` accesses the value of `i` and also modifies `i`, causing undefined behavior.

7

Arrays and Characters

```
int main() {
  int digits[10] = {0}, i; char c;

  while((c = getchar()) != EOF) {
    if (c >= '0' && c <= '9')
      digits[c-'0']++;
  }

  return 0;
}
```

8

Program: Checking a Number for Repeated Digits

- The 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.

9

repdigit.c

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

#include <stdio.h>
#define FALSE 0
#define TRUE 1

int main(void){
  int 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;
}
```

10

`sizeof` and 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 4 bytes).
- Use `sizeof` to test the length of an array:

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

- Defining a macro for the size calculation:

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

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

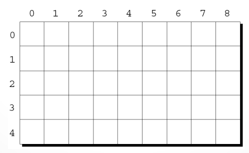
11

Multidimensional Arrays

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

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

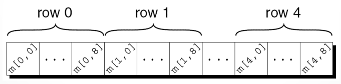
- `m` has 5 rows and 9 columns. Both rows and columns are indexed from 0:



12

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:



Initialization

- 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.
- If an initializer isn't large enough to fill a multidimensional array, the remaining elements are given the value 0.

```
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}};
```

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.

Program: Dealing a Hand of Cards

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

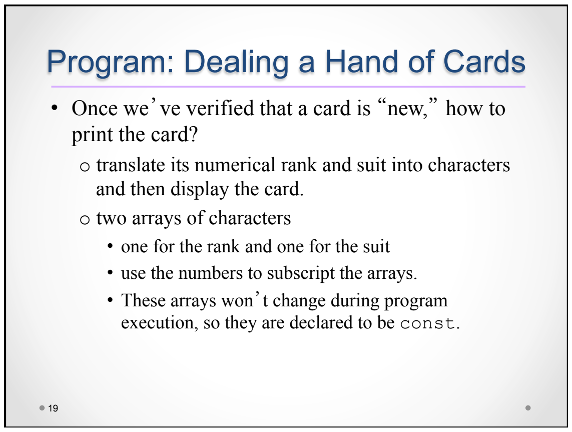
```
Enter number of cards in hand: 5
Your hand: 7c 2s 5d as 2h
```

Program: Dealing a Hand of Cards

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

Program: Dealing a Hand of Cards

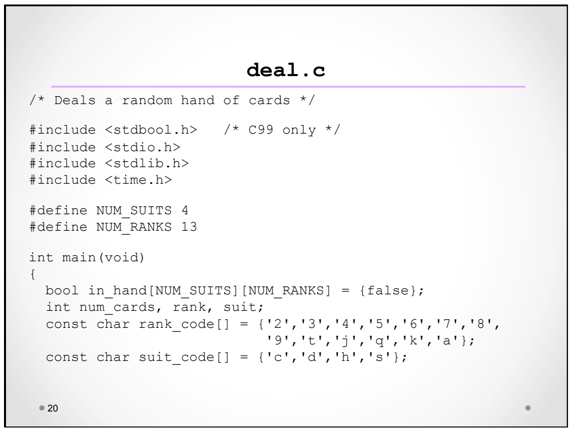
- How do we keep track of which cards have already been chosen?
  - The `in_hand` array with 4 rows and 13 columns;
  - All elements of the array will be false to start with.
  - Each time we pick a card at random, we'll check whether the element of `in_hand` corresponding to that card is true or false.
    - 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've verified that a card is "new," how to print the card?
  - translate its numerical rank and suit into characters and then display the card.
  - two arrays of characters
    - one for the rank and one for the suit
    - use the numbers to subscript the arrays.
    - These arrays won't change during program execution, so they are declared to be `const`.

19



deal.c

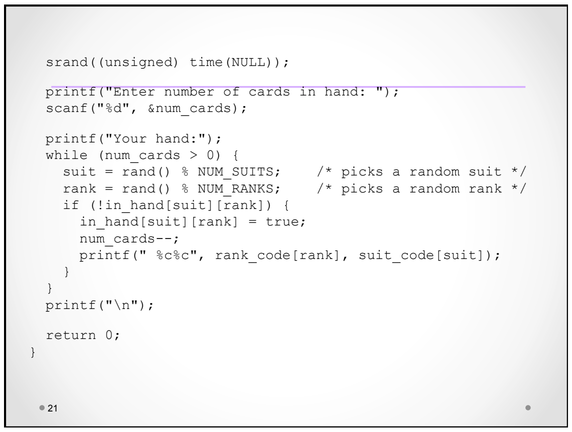
```
/* Deals a random hand of cards */

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

#define NUM_SUITS 4
#define 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'};
```

20



```
  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;
}
```

21