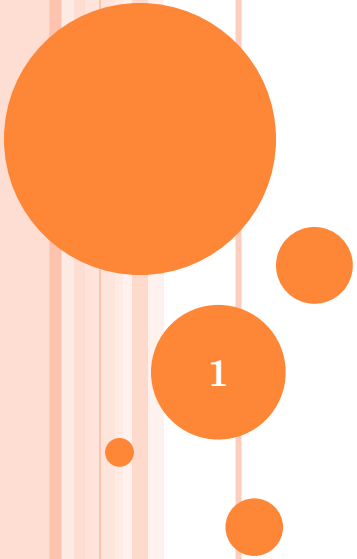


Arrays

CS A150 - C++ Programming 1

Arrays

- **Arrays**
  - Are built-in homogeneous containers in C/C++
  - Collect elements of the **same type**
- ***Lower-level abstraction***
- ***Fast*** and ***efficient***
- Allow ***random access***

Defining and Using Arrays

- An array of five integers:

```
int score[5];
```

- An array can be initialized when it is defined:

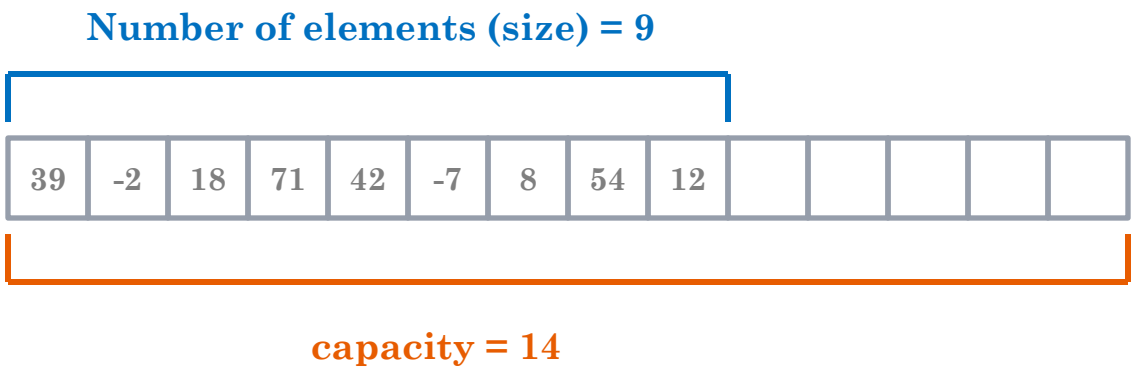
```
int score[]= { 100, 98, 100, 87, 92 };
```

  - Above, compiler determines the size by counting the values

- Elements in the array are numbered from **0** to (**capacity – 1)**

Capacity and Number of Elements

- Differentiate:
  - **Capacity**
    - How large the array is
  - **Number of elements**
    - Often defined as "**size**" of the array



Initializing Arrays

- As simple variables can be initialized at declaration:

```
int number = 3;
```

- Arrays can as well:

```
int scores[3] = { 100, 98, 87 };
```

  - Equivalent to the following:

```
int scores[3];
scores [0] = 100;
scores [1] = 98;
scores [2] = 87;
```

Array Capacity

- **Array capacity** has to be known ***at compile time***
- Size of a **statically-allocated array** **cannot** change
- ***Always*** use a defined/named **constant** for array capacity
  - Example:

```
const int NUMBER_OF_STUDENTS = 5;
int scores[NUMBER_OF_STUDENTS];
```

  - Improves *readability*, *versatility*, and *maintainability*

Accessing Arrays

- Access using **index/subscript**

```
cout << scores[3];
```

  - Will access the ***fourth*** element
- Note two uses of brackets:
  - In **declaration**, specifies **CAPACITY** of array
  - Anywhere else, specifies a ***subscript***

Loops

- **Loops** are extensively used when dealing with arrays:
  - **for** loops when traversing the entire array is needed.
    - Note that "entire" means only up to the last element and not up to the last index.

```
for (int idx = 0; idx < numOfElements; ++idx)
```

  - **while** loops when traversing might be cut off before reaching the last element.
    - If you are searching an element, there is no reason to continue searching after you found the element.

```
while(!found && idx < numOfElements)
```

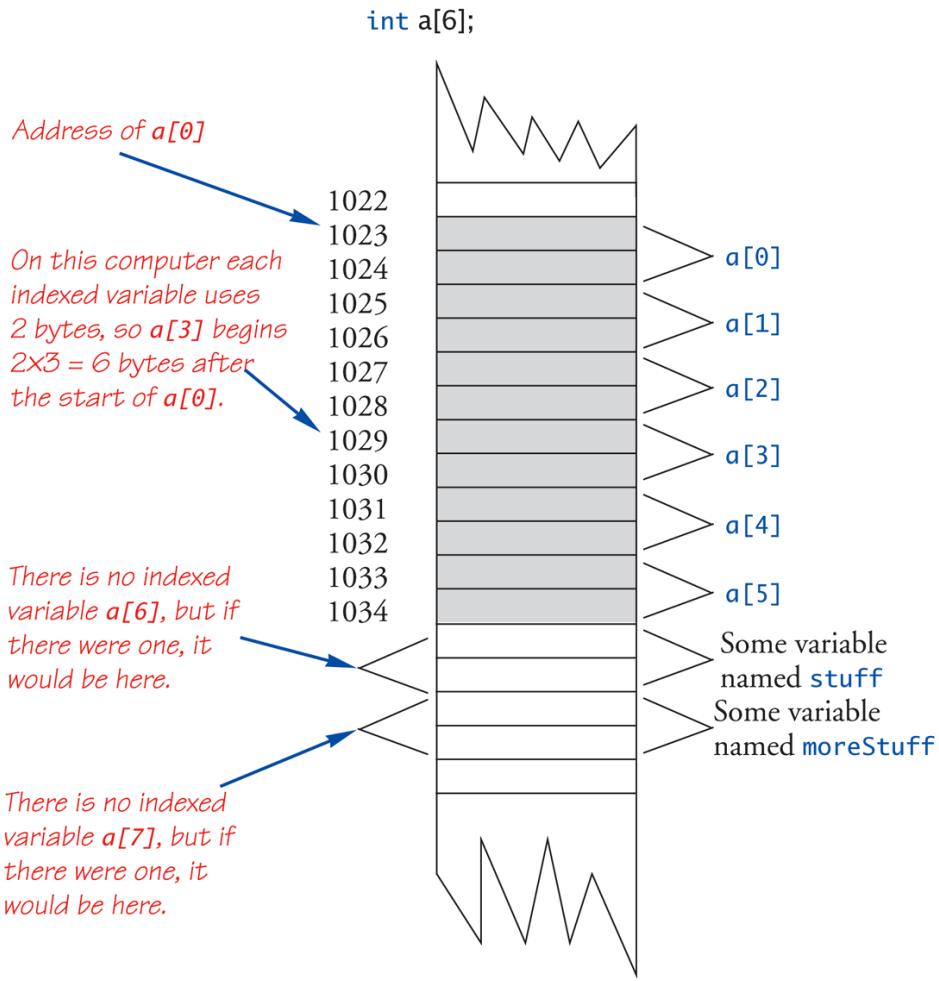
Book Example

- Display 5.1: Program Using an Array

Major Array Pitfall

- Array indices ***always*** **start with zero**
- Zero is "first" number to computer scientists
- C++ will "let" you go beyond range
  - Unpredictable results
  - Compiler will not detect these errors!
- Up to programmer to "**stay in range**"

Array in Memory



Arrays in Functions

- As **arguments** to **functions**
  - You can **pass indexed variables**
    - An individual "element" of an array can be function parameter
  - And you **can pass entire arrays**
    - All array elements can be passed as "one entity“
    - Called “**array parameter**”
    - **Send number of elements** as well

Array Parameters

- What's really passed when you are passing an array?
- Think of array as 3 "pieces"
  - **Address** of first indexed variable (arrName[0])
  - Array base **type**
  - **Number of elements** in the array
- But only 1st piece is passed!
  - Just the beginning address of array
    - You need to provide the rest
  - Very similar to "pass-by-reference"

Array Parameters (cont.)

- When passing arrays
  - **No** brackets in array argument
  - Must send **number of elements** separately
- One nice property:
  - Can use SAME function to fill any size array
  - Exemplifies "re-use" properties of functions
  - Example:

```
int score[5],
    time[10];
fillUp(score, 5);
fillUp(time, 10);
```

Book Example

- Display 5.3: Function with Array Parameter

Using the `const` Modifier

- **Recall:** **Array parameter** actually passes address of 1st element
  - ***Similar*** to **pass-by-reference**
- Function can then modify array!
  - Often desirable, sometimes not
- Protect array contents from modification
  - Use **`const`** modifier **before** array parameter
    - Called "**constant array parameter**"
    - Tells compiler to "**not allow**" modifications

THE `const` MODIFIER - EXAMPLE

- The function print will output all the items in the array without modifying the array

  → **will NOT modify** the array

```
void print(const int a[], int numOfElem);
```

- The function replace will overwrite every even number in the array with a zero

  → **will modify** the array

```
void replace(int a[], int numOfElem);
```

Functions that Return an Array

- Functions **cannot** return arrays same way simple types are returned
- Requires use of a "**pointer**"
- Will be studied later (***dynamic arrays***)

Function Comments

- When writing comments for your function, you will include **both**
  - the **const** modifier and
  - the ampersand **&**

```
int someFunction( const int a[], int & numOfElem);
//someFunction - (description)
//@param const int[] - (description)
//@param int& - (description)
//@return int – (description)
```

Programming with Arrays

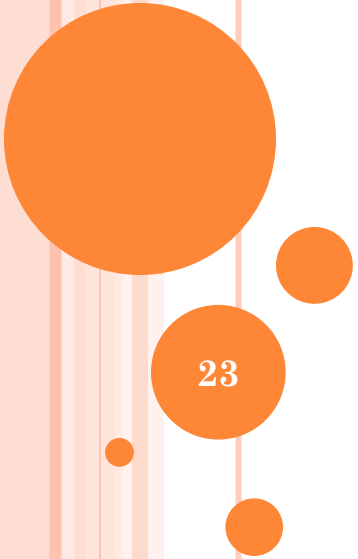
- Plenty of uses
  - **Partially-filled arrays**
    - Must be declared some "**max number of elements**"
    - **Number of elements**: sometimes called ***size***
    - **Capacity**: length of array
  - Sorting
  - Searching

Example

- Project: Searching an Array

Questions?

(Arrays)



(Arrays)