Arrays

CISC1600/1610
X. Zhang

Outline

- Function parameter passing
- Array recap
- Array as a container: partially filled array
- Passing array to function
- Some common functions on arrays
- 2D (two dimensional) arrays

Function call: recap

- **Arguments vs Parameters**
  - Arguments: what are being passed in a function call
  - Parameters: those listed in function declaration/definition
- Passing parameters: assigning actual arguments to corresponding parameters
  - pass-by-value: the value of actual argument is assigned to parameter
  - pass-by-reference: the address of actual argument (must be a variable) is assigned to parameter
    - the parameter refers to the argument

Function call: passing parameters

```
void borrow (int & x, int y, int amount);

int main ()
{
    int value1, value2;
    value1 = 2;
    value2 = 10;
    cout <<"value1=" << value1 << endl;
        << "value2=" << value2 <<endl;
    borrow (value1, value2, 3);  //Calling borro
value2
    cout <<"After function call: " ;
    cout <<"value1=" << value1 << endl;
        << "value2=" << value2 <<endl;

}

void borrow (int & x, int y, int amount)
{
    x-=amount;
    y+=amount;
}
```

- Which parameters are passed by value, by argument?
- Parameter y and amount are pass-by-value
  - int y = value2;
  - int amount = 3;
  - the value of actual argument is assigned to parameter
- Parameter x is pass-by-reference: address of actual argument, value1, is assigned to parameter
  - int & x = value1;// x refers to value1 in main
  - When x-=amount, it's modifying value1 in main!

Outline

- Function parameter passing
- Array recap
- Array as a container: partially filled array
- Passing array to function
- Some common functions on arrays
- 2D (two dimensional) arrays

Array Syntax

SYNTAX 6.1 Defining an Array

Element type   Name   Size

Size must be a constant.

Ok to omit size if initial values are given.

```
double data[5] = { 32, 54, 67.5, 29, 35 };
```

Optional list of initial values

Use brackets to access an element.

```
data[i] = 0;
```

The index must be ≥ 0 and $<$ the size of the array.

Define Arrays

data =

| ?? |
| ?? |
| ?? |
| ?? |
| ?? |
| ?? |
| ?? |
| ?? |
| ?? |
| ?? |

10

An "array of double"

Ten elements of **double** type can be stored in this array named data

```
double data[10];
```

type of each element

# of elements, i.e. the "size" of the array

Must be a constant
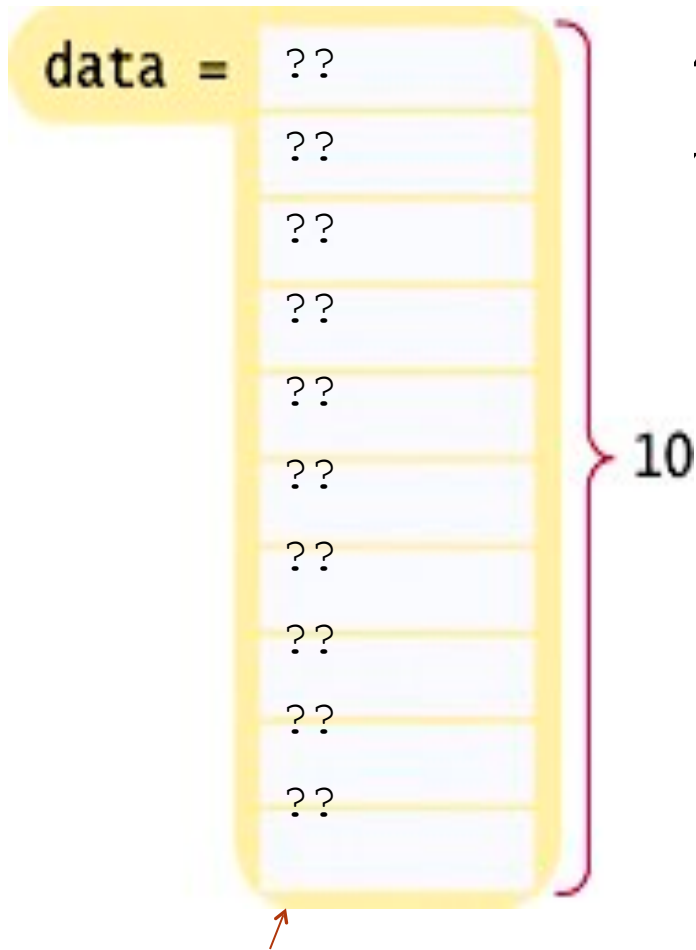
The elements have random values!
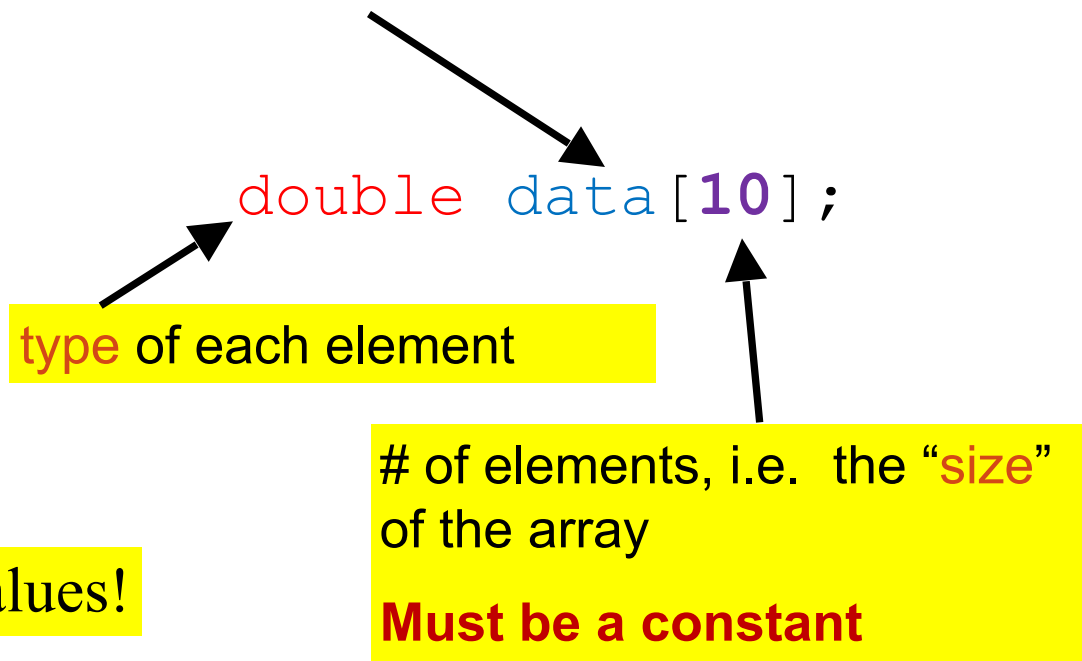
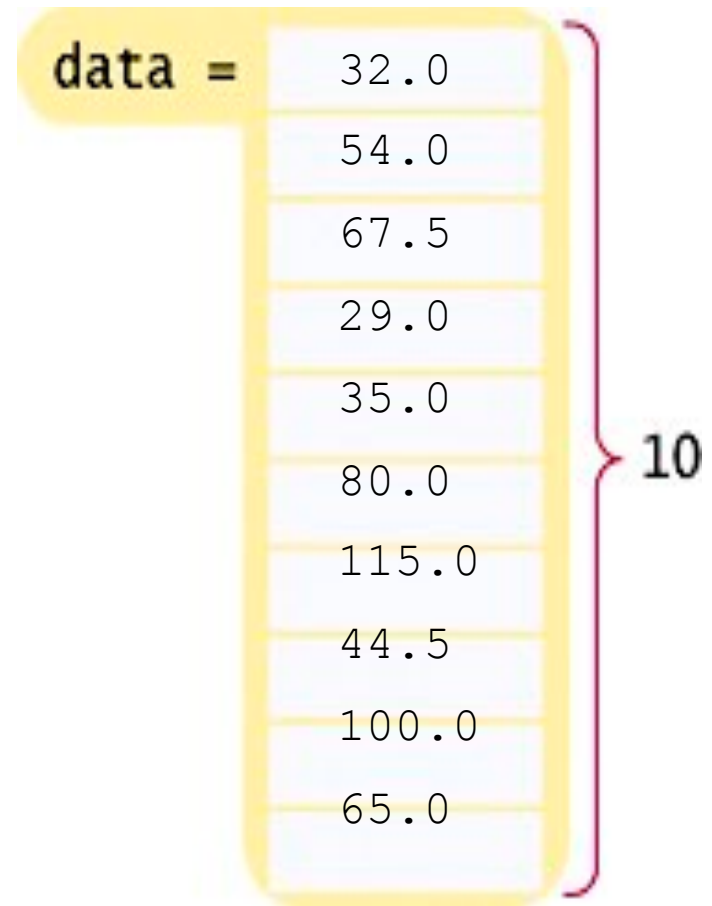# Define Arrays with initial values

define an array while specifying initial values:

```
double data[] = { 32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65 };
```

size is omitted, as
compiler can count...

Practice: declare an array and store
names of all US coins in the array

Practice: declare an array of char to store
characters 'a', 'b', 'c', 'd', 'e'.

data =

| 32.0 |
| 54.0 |
| 67.5 |
| 29.0 |
| 35.0 |
| 80.0 |
| 115.0 |
| 44.5 |
| 100.0 |
| 65.0 |

10

Access Array Element

```
double data[] = { 32, 54, 67.5, 29, 35, 80,
115, 44.5, 100, 65 };
```

data =

32.0	[0]
54.0	[1]
67.5	[2]
29.0	[3]
17.7	[4]
80.0	[5]
115.0	[6]
44.5	[7]
100.0	[8]
65.0	[9]

data[4]

❖ To access an array element:

data[i]

❖ where **i** is the *index*.

❖ In C++ and most computer languages, index starts with **0**.

Visiting All Elements

To visit all elements of an array, use a variable for index.

```cpp
double data[] = { 32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65 };

for (int i = 0; i < 10; i++)
{
    cout << data[i] << endl;

}
```

When i is 0, data[i] is data[0], the first element.

When i is 1, data[i] is data[1], the second element.

…When i is 9, data[i] is data[9], the *last legal* element.

Bounds Error

A *bounds* error occurs when you access
an element outside the legal set of indices:

```
cout << data[10];
```

Doing this can corrupt data
or cause your program to terminate.

DANGER!!!

DANGER!!!

DANGER!!!

Table 1 Defining Arrays

`int numbers[10];`	An array of ten integers.
`const int SIZE = 10;` `int numbers[SIZE];`	It is a good idea to use a named constant for the size.
🚫 `int size = 10;` `int numbers[size];`	**Error:** The size must be a constant.
`int squares[5] = { 0, 1, 4, 9, 16 };`	An array of five integers, with initial values.
`int squares[] = { 0, 1, 4, 9, 16 };`	You can omit the array size if you supply initial values. The size is set to the number of initial values.
`int squares[5] = { 0, 1, 4 };`	If you supply fewer initial values than the size, the remaining values are set to 0. This array contains 0, 1, 4, 0, 0.
`string names[3];`	An array of three strings.

Use array to build a table

- How to display a date using English name for month?

```
int month;
…
if (month==1)
    cout <<"Jan";
else if (month==2)                    very long code!
        cout <<"Feb";
else if (month==3)
        cout << "Mar";
...
```

Usage of array

- We can simplify the code using a table that maps 1 to Jan, 2 to Feb, 3 to Mar, …
- How ?
  - String month_names[12]={"Jan", "Feb", "Mar", "Apr", "May", "Jun", "July", "Aug", "Sep", "Oct", "Nov", "Dec"};
  - cout << month_names[month];
  - Not quite, this maps 0 to Jan, 1 to Feb, ..
  - To fix: cout << month_names[month-1];

Usage of array

- We can simplify the code using a table that maps 1 to Jan, 2 to Feb, 3 to Mar, …
- Alternatively:
  - String month_names[13]={"","Jan", "Feb", "Mar", "Apr", "May", "Jun", "July", "Aug", "Sep", "Oct", "Nov", "Dec"};
  - cout << month_names[month];

- one liner versus if/elseif statement with 12 cases !

DataChecker function

- Create and use a table that maps month to the number of days in the month
  - 1: 31
  - 2: 28 or 29
  - 3: 31
  - 4: 30
  - 5: 31
  - …

  How to convert to using vector?

- What to do with Feb?

```
int DaysInMonth[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
if (leap_year(yr))
      DaysInMonth[2]=29;
```

Outline

- Function parameter passing
- Array recap
- Array as a container: partially filled array
- Passing array to function
- Some common functions on arrays
- 2D (two dimensional) arrays

Partially-Filled Arrays

- How to choose size for an array ?
  - pick a reasonable maximum number of elements
 - We call this quantity **capacity**, e.g.,

```
const int CAPACITY = 100;
double data[CAPACITY];
```

- This array usually has less than CAPACITY elements in it
  - array is **partially filled**
  - use a companion variable to keep track # of elements in array

```
int size = 0; // array is empty initially
```

Partially-Filled Arrays

If only four elements have been stored in the array:


```
for (int i = 0; i < size; i++)
{
    cout << data[i] << endl;
}
```
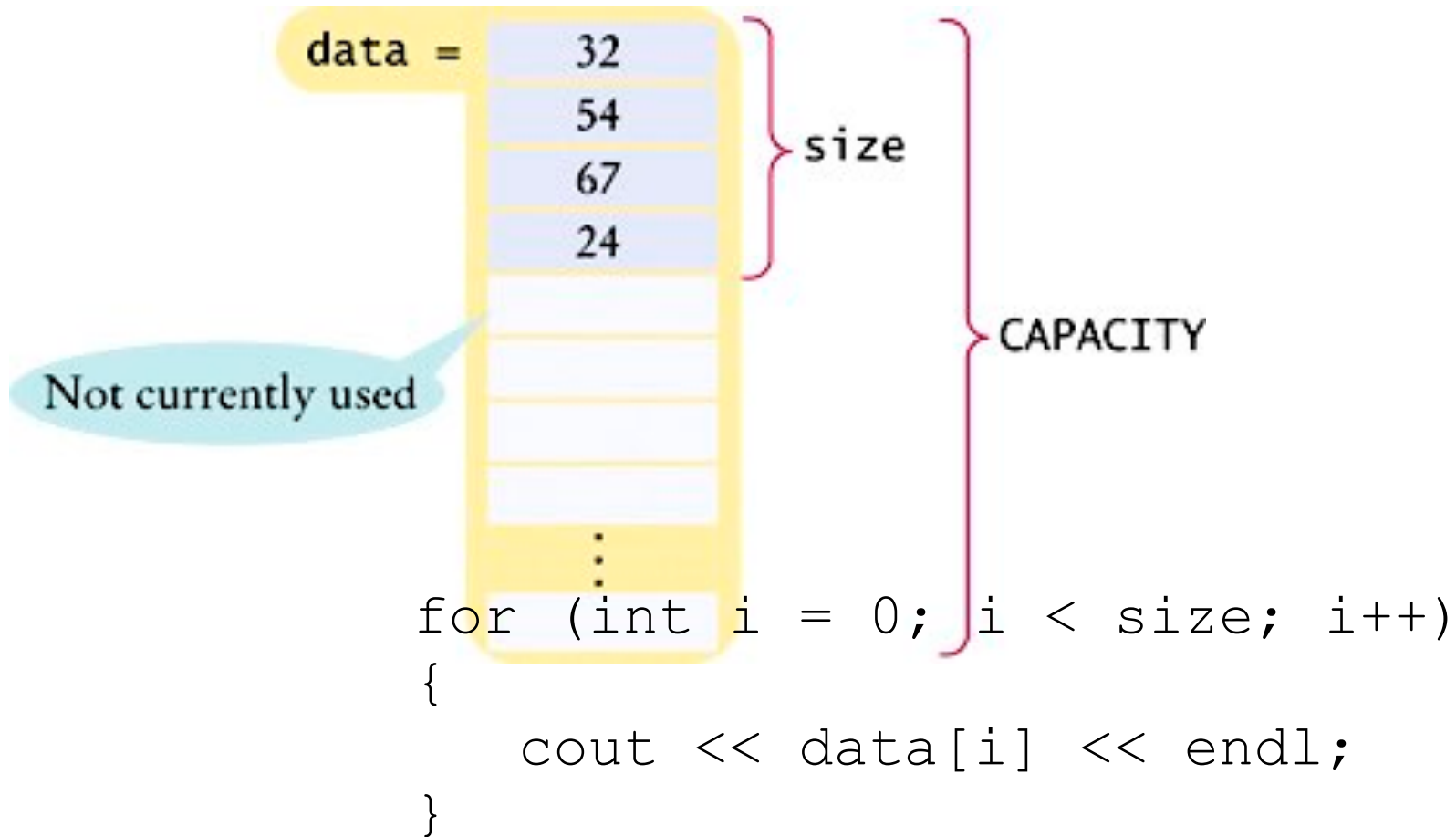
Partially-Filled Array

```cpp
const int CAPACITY = 100;
double data[CAPACITY];
int size = 0;

double input;
while (cin >> input)
{
    if (size < CAPACITY)
    {
        data[size] = x;
        size++;
    }
}
```

Whenever size of the array changes we update this variable

Outline

- Function parameter passing
- Array recap
- Array as a container: partially filled array
- Passing array to function
- Some common functions on arrays
- 2D (two dimensional) arrays

Arrays

values

- a lower level structure
  - contiguous memory that holds multiple elements of same type
  - actually pointer variable, storing address of first (0th) element
- Programmers
  - use companion variables to keep track part of array is filled
    - If array is always filled from top, size
    - Sometimes, use two indices, start, end

0

1

Array Parameter

❑ When passing array to a function ...
- use an empty pair of square brackets
  after parameter name to indicate it's an array.
- Pass size of array (# of elements)

```
 // calculate the sum of elements in an array
double sum(double data[], int size)
 {
    double total = 0;
    for (int i = 0; i < size; i++)
    {
        total = total + data[i];
    }
    return total;
 }
```

Exercise

- Write a function that calculate the average of values in the array of int

Array Parameters

❏ changes to array parameter are made to actual parameter, i.e., the array the caller passed to function:

```
void Initialize(double data[], int size)
{
    for (int i = 0; i < size; i++)
    {
        data[i] = 0;
    }
}
```

This works! But why?

```
 in caller,

  double my_array[20];
  Initialize (my_array, 20);
```

Array Parameters

❑ **data** is passed by value,
- At function call, parameter variable (data) is initialized with value of actual parameter (my_array)
- Key: recall array variable is a pointer (addr of first element of array)

```
void Initialize(double data[], int size)
{
    for (int i = 0; i < size; i++)
    {
        data[i] = 0;
    }
}
…
double my_array[20];
Initialize (my_array, 20);
```
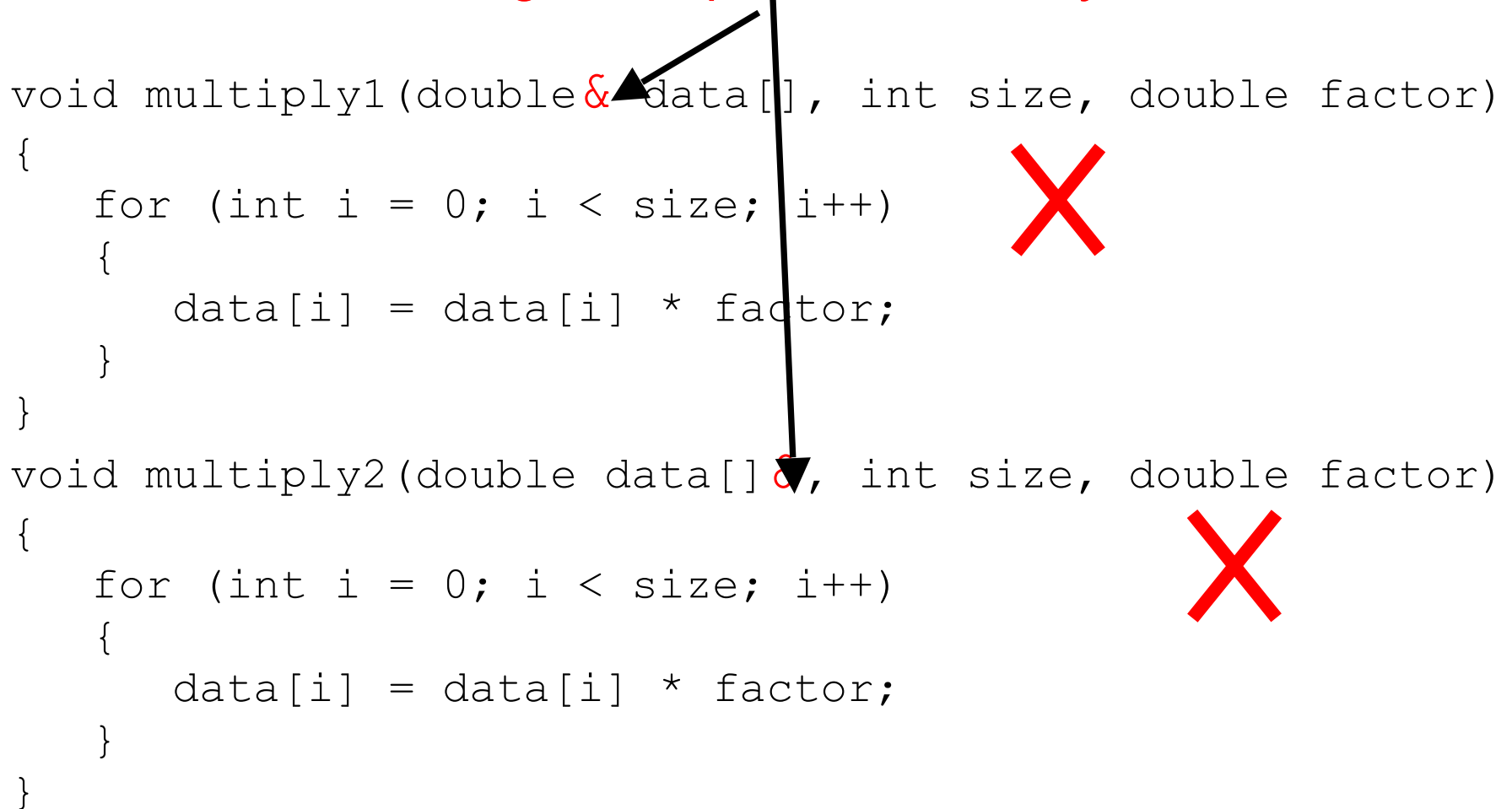
Array Parameter

And writing an ampersand is *always* an error:

```
void multiply1(double& data[], int size, double factor)
{
    for (int i = 0; i < size; i++)
    {
        data[i] = data[i] * factor;
    }
}
void multiply2(double data[]&, int size, double factor)
{
    for (int i = 0; i < size; i++)
    {
        data[i] = data[i] * factor;
    }
}
```

Outline

- Function parameter passing

- Array recap

- Array as a container: partially filled array

- Passing array to function

- Some common functions on arrays

- 2D (two dimensional) arrays

Example functions on array

Search array for an element

/** search for value x in an array

@param values: array of ints

@param size: length of values

@param x: the value to look for

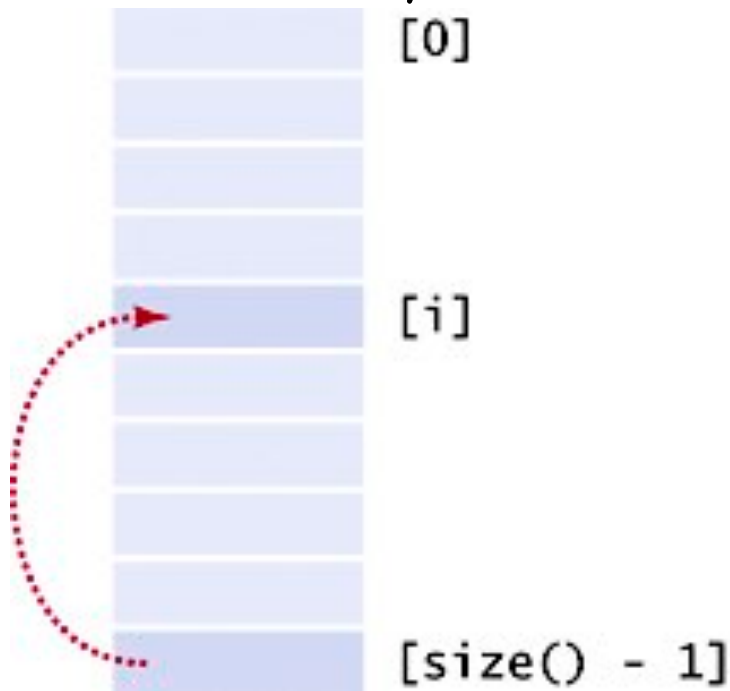@return -1 if x is not in the array, otherwise, return the index */

int search (int values[ ], int size, int x)

{

}

Removing an Element*

❖ Suppose you want to remove the element at index i.

❖ If elements in the array are not in any particular order

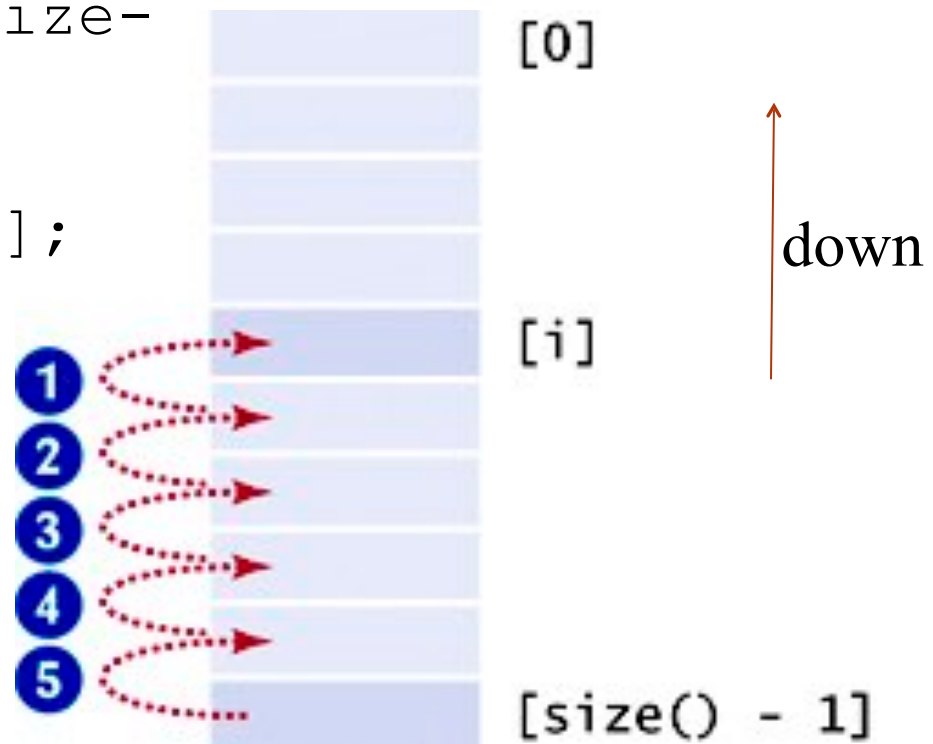   ❖ Overwrite the element to be removed with the last element, then shrink the size by one

```
[0]

[i]

[size() - 1]
```

```
int last_pos = size – 1;
data[i] = data[last_pos];
size --;
```

Removing an Element (cont'd)*

If array is sorted

move all elements following the i-th element "down" (to a lower index), and then shrink the size by one

```
for (int j = pos; j < size-
    1; j++)
{
    data[j] = data[j + 1];
}
size --;
```
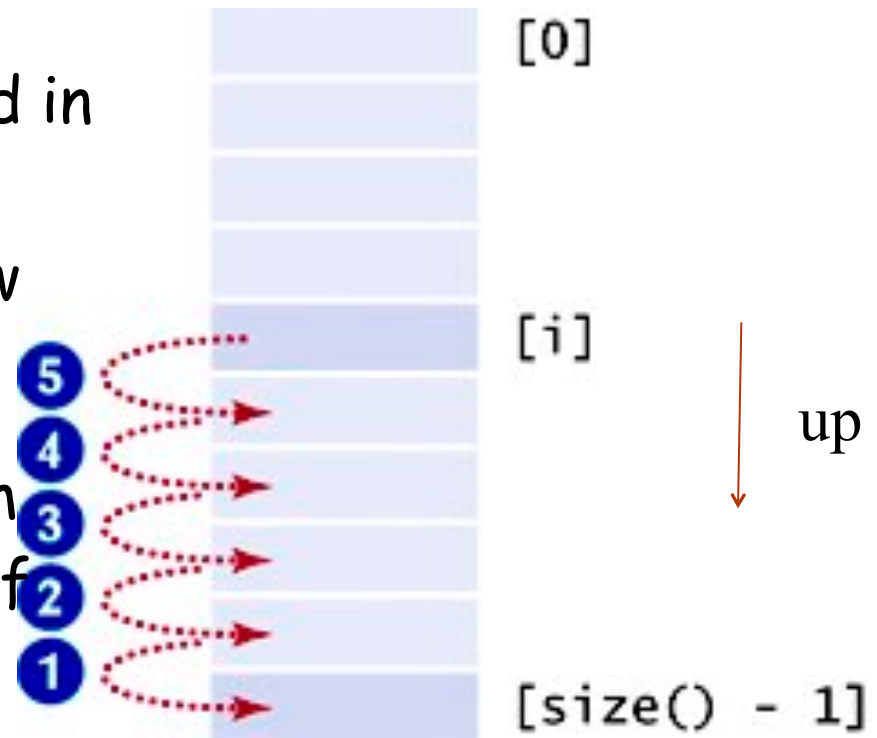
[0]

down

[i]

[size() - 1]

Inserting an Element*

❖ Insert a new element into array
❖   array[size] = data;
❖   size++;

❖ If elements need to be sorted in order
   ❖   find position to insert new value

   ❖   Move up all elements from that location to the end of array

   ❖   Insert the new element at now vacant position [i].

[0]

[i]

[size() - 1]

up

Outline

- Function parameter passing

- Array recap

- Array as a container: partially filled array

- Passing array to function

- Some common functions on arrays

- 2D (two dimensional) arrays

Two-Dimensional Arrays

- Collections of values that have a **two-dimensional** layout
    - E.g., multiplication table
    - E.g., configuration of game board (tic-tac-toe, chess…)
- Each element accessed with *rows and columns* index
    - ***two-dimensional array***, or a ***matrix***

TWO-DIMENSIONAL ARRAY

- Declare a 2D array:

const int ROWS_NUM = 10;
const int COLS_NUM = 10;
double multi_table[ROWS_NUM][COLS_NUM];

As with one-dimensional arrays, size in both dimension decided at compilation time, cannot be changed …

- Individual elements are accessed by double subscripts: multi_table[i] [j]
  - multi_table[3][4] = 3*4;
  - **Use separate brackets!** , not multi_table[3,4] (wrong)

FILLING IN DATA

- To fill a two-dimensional array, use nested loops:

```
for (int i = 0; i < ROWS_NUM; i++)
    for (int j = 0; j < COLS_NUM; j++)
        multi_table[i][j] = i*j;
```

Computing Row and Column Totals

- A common task is to compute row or column totals.
- Row total:

```
for (int i=0; i< ROWS_NUM;i++)
{
    total = 0;
    for (int j=0; j < COLS_NUM, j++)
        total += multi_table[i][j];
    cout << " sum of row " << i <<" is " << total;
}
```

How is 2D array stored?

- Like 1D array, 2D array is also a low-level construct
  - 2D array stored in a contiguous block of memory: first row 0 elements, then row 1 elements, …
- Compiler finds multi_table[i][j] by computing offset:
  - i * COLS_NUM + j // there are i rows before me
  - Compiler needs to know COLS_NUM => COLS_NUM must be a const

row 0   row 1   row 2   row 3

powers =

. . .

powers[3][4]
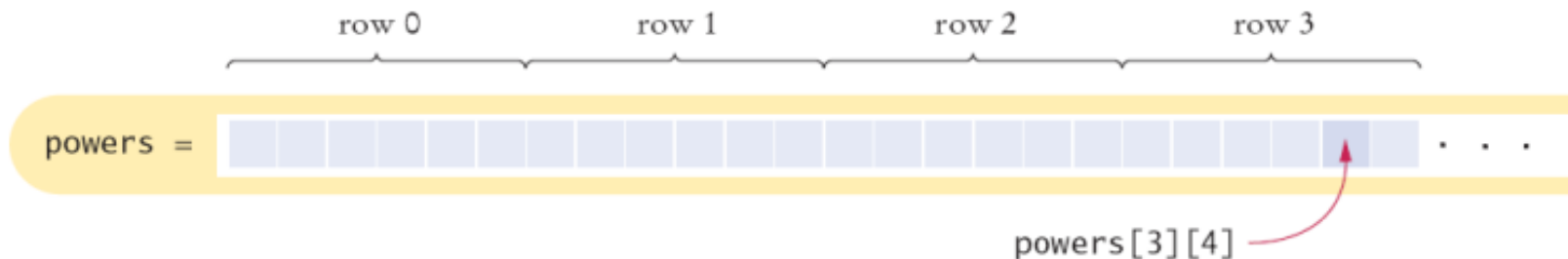
PASSING TO FUNCTIONS

- Specify number of columns when declaring 2D array parameter

```
void print_table(double table[][COLS_NUM], int table_rows)
{
  const int WIDTH = 10;
  cout << fixed << setprecision(2);
  for (int i = 0; i < table_rows; i++)
  {
     for (int j = 0; j < COLS_NUM; j++)
        cout << setw(WIDTH) << table[i][j];
     cout << "\n";
  }
}
```

When calling this function, passing the name of the array `print_table (power_table, 4);`

Common Error

Leaving out the columns value is a very common error.

```
int row_total(int table[][], int row)
...
```

The compiler doesn't know how "long" each row is!

Example

Putting a value for the rows is not an error.

```
int row_total(int table[17][COLUMNS], int row)
...
```

The compiler just ignores whatever you place there.

2D Array Parameters

- Following function works only for 2D arrays with 6 columns.

   const int POWERS_COL=6;

   void print_table(const double table[][POWERS_COLS],
      int table_rows)

- To process 2D array with different column numbers:
  - A different function:

```
const int COLUMNS_2 = 4;
int print_table_4(int table[][COLUMNS_2], int row)
{
 ….
}
```

SUMMARY

- Array
  - Declaration, fixed size at compilation time
  - Use of an int variable to store its size
  - Array assignment
  - Use array in function
    - Return an array
    - As parameter , need to pass the size …
- 2D array accessed with two subscripts: row and column
  - Stored as 1D array
  - Compiler calculate index from row and column indices