

Topic 7: Arrays and collections

1. Defining and populating arrays
2. Array operations and boundary errors
3. Two-dimensional arrays
4. Collections

Objectives

18. Define and manipulate an array
19. Manage a collection

1. Defining and populating arrays

- *Array*: An indexed sequence of storage locations for data items all of the same type
- Arrays implement mathematical sequences, e.g., a_1, a_2, a_3, \dots
- Elements are accessed by *subscript*, ranging from 0 to (#elements - 1):

```
score[0] = 75;
out.print(score[0]);
```
- Memory address of an element of an array is calculated from its offset from the first element

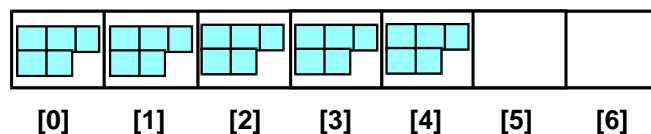
Declaring arrays

- Arrays are declared in Java using brackets, *new* operator, and the number of elements:

```
int[] score = new int[6];
```
- Arrays are *objects* with the special constant public data member *length*:

```
score.length
```

 above is 6
- Array may have *occupancy* smaller than length



Array initialization

```
int[] days_in_month =
{
    31, 28, 31, 30, 31, 30,
    31, 31, 30, 31, 30, 31
};
out.print("February has " +
    days_in_month[1] + " days");
```

Array of integers

- When array is initialized in this way, memory allocation by use of *new* is implicit and number of elements in array is number of initial values

Building an array

```
final int MAX=3;
int[] income = new int[MAX];
int sum=0;

income[0] = 258;
income[1] = 192;
income[2] = 467;
for (int i=0; i < MAX; ++i)
{
    out.print(income[i]);
    sum += income[i];
}
out.print("Total: "+sum);
```

Reading file into array

```
final int MAX_SZ = 50;
String file_name = "Readfile.txt";
int[] A = new int[MAX_SZ];

System.out.println("Reading "+file_name);
FileReader reader = new FileReader(file_name);
Scanner fin = new Scanner(reader);

int i = 0;
while (fin.hasNextInt() && i < MAX_SZ)
{
    A[i] = fin.nextInt(); // Read
    i++;
}
fin.close();

for (int j=0; j < i; j++)
    System.out.print(A[j] + " "); // Display
```

2. Array operations and boundary errors

Loops may be written to:

- Search or search and replace
- Sort
- Calculate statistics such as sum, average, maximum, minimum, median, variance, mode

Example: Calculating variance

```
int[] score = {90,75,84,94,89,97,81},
    i,j,total,avg,
    num_scores = sizeof score / sizeof(int);

// Find average:
for (i=0,total=0; i < num_scores; ++i)
    total += score[i];
avg = total / num_scores;

// Display variances from average:
out.printf("Average = %d\n",avg);
out.printf("Score    Variance\n");
for (j=0; j < num_scores; ++j)
    out.printf("%4d%12d\n",score[j],score[j] - avg);
```

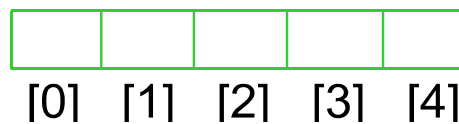
Output:

Score	Variance
90	3
75	-12
84	-3
94	7
89	2
97	10
81	-6

*Variance is difference between
one item's value and average value*

Observing array boundaries

- `int[] n = new int[5];` allocates five *ints*



- Semantically valid subscripts here: 0 to 4
- Boundary violation throws exception
- `"n[5]"` is valid syntax, but refers to memory that is outside the bounds of the declared array

Boundary errors

- Consider

```
int[] A = new int[5];
A[-1] = 2;
A[5] = 6;
```
- Both assignments are *bounds errors* that Java run-time environment will flag by throwing an exception
- Range of valid array subscripts is 0 .. (A.length-1)
- Any access to an uninitialized array is an error:

```
int[]A; A[0] = 1; // error
```

Copying arrays

- Assigning an array reference as the value of an array object results in two references to the same object:

```
int[] A = new int[8];
int[] B = A;
```
- Deep copying makes a separate copy:

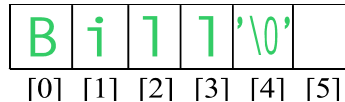
```
int[] B = (int[])A.clone();
```
- `System.arraycopy(A, 2, B, 0, 3)` copies three elements of *A* to *B* starting at subscript 2

A small pink square containing the text "pic" in a teal font.

C-style strings use null terminator

- The C string library and I/O functions treat the ASCII value 0 as a sentinel that terminates a string

```
char name[6] = "Bill";
```



- Assigning '\0' to a character element of a string may shorten the string

```
name[3] = '\0';
out.printf("%s", name);
```

Loops and associative operators

- Expressions with associative operators may be evaluated using loops or induction (recursion)
- Examples:* +, ×, ∧, ∨, ∩, ∪
- Other operations on *sequences* may be computed using loops:
 - Inversion (\neg)
 - Is-ascending*
 - All-identical*
 - Count
 - Search
 - Replace

Recurrences for arrays

Function definition:

$$\text{Sum}(A) = \begin{cases} 0 & \text{if } |A| = 0 \\ A[1] & \text{if } |A| = 1 \\ A[1] + \text{Sum}(A[2 .. |A|]) & \text{otherwise} \end{cases}$$

Algorithm suggested by definition:

```

sum ← 0
for i ← 1 to |A|
    sum ← sum + A[ i ]
  
```

Similar recurrences exist for other functions

3. Two-dimensional arrays

3552.2	3560.0	3540.0	← High
3539.2	3544.8	3530.6	← Low
3550.5	3557.7	3530.6	← Close

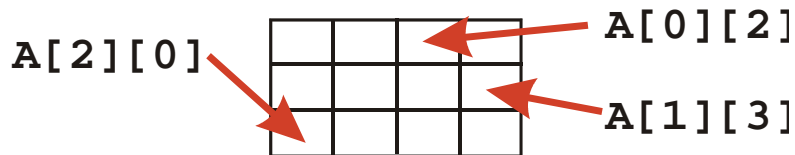
```

Final int MAX_DAYS = 30;
enum PRICE_CATEGORY {HIGH,LOW,CLOSE};
Double[][] daily_average = new double
    [CLOSE+1][MAX_DAYS];
  
```

Rows *Columns*

Rows and columns in 2D array

- First subscript represents *row* (up-down)



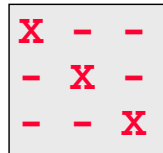
- Second subscript represents *column* (across)
- This is called *row-major ordering*

Initializing a 2D array (C++)

```
const int NUM_DAYS = 5;
enum CATEGORIES {HIGH,LOW,CLOSE};
const char* CATEGORY_NAME[CLOSE+1] =
    {"HIGH", "LOW", "CLOSE"};
void main()
{
    double price[CLOSE+1][NUM_DAYS] =
    {
        {76.2,81.3,78.5,79.2,80.7},
        {70.9,75.4,71.3,71.8,74.1},
        {74.0,73.6,78.2,76.6,79.5}
    };
};
```

Example: Tic-tac-toe board

```
char board[3][3] = {'-'};
board[0][0] = 'X';
board[1][1] = 'X';
board[2][2] = 'X';
for(int row = 0; row < 3; ++row)
{
    for(int col = 0; col < 3; ++col)
        out(board[row][col]);
}
```



```
X - -
- X -
- - X
```

String arrays

```
// array of 100 strings,
String[] s = new String[100];

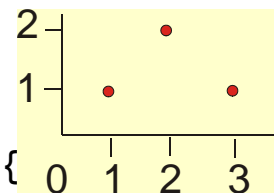
// Read words from file into array 's':
for (int i=0; ! in.hasNext(); i++)
{
    s[i] = in.nextLine();
}
```

4. Collections

- A collection is an array of objects
- Collections correspond to database tables, where columns correspond to data members of objects
- The *ArrayList* generic class stores collections of arbitrary objects

An array of objects

```
public class Point {
    int x,y;
    void display()
    {
        out.print("(" + x + "," + y + "),");
    }
}
public void main()
{
    Points triangle[3] = {{1,1},{
    for (int i=0; i < 3; ++i)
        triangle[i].display();
}
```



A collection of employees

```

class Roster
{
  public Roster()
  { num_recs = 0; }
  private Employee emp[100];
  private int num_recs;
};

```

A collection class
[payroll.cpp]

```

class Employee
{
  char name[40];
  int salary;
};

```

An instance of collection class rosters

num_recs

4

emp

David Keil Computer Science I 7. Arrays 6/09 23

A relation associates set elements

- An array of structures implements a *relation*
- A relation represents an *association* between elements of two or more sets
- *Examples:*
 - product names and prices (price list)
 - teams, opponent teams, and dates (game schedule)
 - swimmers and elapsed times
- The term “relation” is from set theory and database theory, not C or C++ terminology

A location on a coordinate axis

```
public class Point
{
    public Point()
        { x = y = 0; }
    public Point(int x_init,int y_init)
        { x = x_init; y = y_init; }
    public void input(Scanner in)
        {x = in.nextInt(); y = in.nextInt(); }
    public void display()
        { out.print("(" + x + "," + y + ") "); }
    private int x,y; // data members
};
```

[point.cpp]

A collection of point objects

```
public class point_collections
{
    public point_collections() { num_points = 0; }
    public void retrieve(String name) {...};
    public void display() {...};
    enum { MAX_POINTS = 100 };
    private Point point[MAX_POINTS];
    private int num_points;
};

public static void main()
{
    point_collections list = new point_collections();
    list.retrieve("point.txt");
    list.display();
}
```

ArrayLists are expandable

- Example declaration:
`ArrayList <Integer> A = new ArrayList<Integer>;`
- Array lists expand or contract as needed
- *ArrayList* is called a *generic class* because it supports collections of objects of any uniform type
- Methods: *add(x)*, *remove(i)*, *set(i, x)*, *get(i)*, *add(i,x)*
- To use: *import java.util.ArrayList;*
- Compare with C++ *vectors*

ArrayList example

- Example declaration with operations:
`ArrayList <Integer> A =
 new ArrayList<Integer>;
A.add(2);
A.add(5);
A.remove(2);
A.set(1,3);
out(A.get(0));`
- Note that a *class*, not a primitive type, must appear in `< >` after *ArrayList*

Wrapper classes

- Wrapper classes convert items of primitive types into objects, e.g., for use with *ArrayList*
- Wrapper classes: *Integer*, *Character*, *Boolean*, *Byte*, *Double*, *Float*, *Long*, *Short*
- Instances of wrapper classes are automatically unwrapped when used, avoiding need to use methods:

```
Integer x = 4;  
Integer y = 2 + x;
```

Traversing a collection with *for*

- Enhanced *for* statement iterates through an *ArrayList*, without using an index variable (usually *i* with *for*)
- *Example:*

```
ArrayList <Integer> A =  
    new ArrayList<Integer>;  
A.add(2); ...  
for(Integer a : A)  
{ out(a+" "); }
```

Array problems

- Given a set of tasks, each with start and finish times, find the smallest number of non-overlapping tasks that fill the day, from 1:00 to 6:00. *Example:* { (A,1,3), (B,2,4), (C,3,5), (D,4,6), (E, 5,6), (F,1,4), (G,1,2) }
- Given user inputs of a set of 4 quarterly sales figures for each of 3 regions, calculate totals by quarter and by region. *Example:*

	Q1	Q2	Q3	Q4
R1	236	311	398	892
R2	128	232	109	673
R3	776	897	349	1031

References

- Cay Horstmann. *Big Java*, 3rd ed. Wiley, 2008, Ch. 7.
- D. Keil. Displaying an array. Classroom handout.