Arrays

• We Typically Encounter Groups Of Like-Minded Objects
  – eggs in an egg carton
  – apartments in an apartment building

Agenda

• Arrays
  – Array Parameters
  – Typical Array Operations
• Multidimensional Arrays
Arrays

• We Typically Encounter Groups Of Like-Minded Objects
  – eggs in an egg carton
  – apartments in an apartment building
• Each Object In The Set Is The Same
• The Overall Set Has A Size

Arrays

• We Typically Encounter Groups Of Like-Minded Objects
  – eggs in an egg carton
  – apartments in an apartment building
• Each Object In The Set Is The Same
• The Overall Set Has A Size
• C++ Has A Similar Construct
  – arrays

Arrays

• An Array Is A Collection Of Values Of All Identical Type
  – classes also contain collections of values, but these values are of different types
• The Collection Has A Variable Name
• Each Item In The Collection Has A Subscript That Defines Its Position
Array Declaration

- Syntax:
  ```
  type arrayname[ size ];
  ```
- type referred to as the base type for all array elements
- arrayname is the variable name for the entire collection
- size is the number of elements allowed in the collection
  - indexes from 0 to size-1

Arrays

- Example:
  ```
  int grades[ 5 ];
  ```
Arrays

- Example:
  ```c
  int grades[ 5 ];
  grades[ 0 ]
  grades[ 1 ]
  grades[ 2 ]
  grades[ 3 ]
  grades[ 4 ]
  ```

Each Indexed Element Is An int

Indexes Start At Zero
Arrays

- Arrays Are An Ordered List
- Arrays Are Stored Contiguously In One Block
- Each Index Is An lvalue In Its Own Right
- [ ] Is Used To Declare And Access Arrays

Arrays

- Example:
  ```
  int grades[3];
  grades[0] = 1;
  grades[1] = 10;
  grades[2] = 100;
  ```

Arrays

- Example:
  ```
  int grades[3];
  grades[0] = 1;
  grades[1] = 10;
  grades[2] = 100;
  ```
Arrays

- Example:
  ```c
  int grades[3];
  grades[0] = 1;
  grades[1] = 10;
  grades[2] = 100;
  ```

Note the Two Uses of `[]`'s
Don’t Confuse Declaration Syntax with Element Access

---

Array Program Example

```c
//finds five scores and shows how much each
//score differs from the highest score.
#include <stdio.h>

int main()
{
  int score[5];
  int max;
  int i;
  cin >> score[0];
  max = score[0];
  for (i = 1; i < 5; i++)
  {
    cin >> score[i];
    if (score[i] > max)
    {
      max = score[i];
    }
  }
  //max is the largest of the values score[0].... score[4].
}
```
Major Array Pitfalls!

- Array Indexes Always Start With Zero!
  - Zero is “first” number to computer scientists
- C++ Will “Let” You Go Off The Edge…
  - Leads To Unpredictable Results
  - Even Worse, The Compiler Cannot Detect It
- It Is The Programmer’s Responsibility To “Stay Within The Bounds” Of The Array

Indexes range from 0 to (array_size – 1)
Example:
```cpp
double temperature[24];
They are indexed as: temperature[0], temperature[1] ... temperature[23]
```
- Common mistake:
  ```cpp
temperature[24] = 5;
  ```
  - Index 24 is "out of range"!
  - No warning, possibly disastrous results

Defined Constant As Array Size

- Recommendation: Always Use A Constant To Set An Array Size
  Example:
  ```cpp
  const int NUMBER_OF_STUDENTS = 5;
  int score[NUMBER_OF_STUDENTS];
  ```
- Improves Readability, Versatility And Maintainability
- Use That Constant Everywhere You Loop Over The Array
Defined Constant As Array Size

- Recommendation: Always Use A Constant To Set An Array Size
  Example:
  ```
  const int NUMBER_OF_STUDENTS = 5;
  int score[NUMBER_OF_STUDENTS];
  ```
- Improves Readability, Versatility And Maintainability
- Use That Constant Everywhere You Loop Over The Array

Array Initialization

- Like Other Variables, Arrays Can Be Initialized When They Are Declared
- Generally, It’s A Good Idea To Define Constants For Array Size

```
const int SIZE=3;
int grades[SIZE];
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```
Like Other Variables, Arrays Can Be Initialized When They Are Declared

Generally, It’s A Good Idea To Define Constants For Array Size

```cpp
class MyClass {
    private:
        const int SIZE=3; // Define constant for array size
        int grades[SIZE]; // Declare array
    public:
        grades[0] = 1; // Initialize array
        grades[1] = 10;
        grades[2] = 100;
};
```

If You Supply Fewer Values Than The Full Array, Elements Are Filled From The Front And Elements Lacking A Value Will Get Filled With Zero Of The Base Type

If You Leave Off The Array Size, You Must Supply An Initializer And C++ Will Create An Array For Just These Elements
Array Iteration

- for Loops Are Often Used With Arrays
  - array index need not be a fixed constant

```cpp
const int SIZE=3;
int a[SIZE]={1,10,100};
for (int i=0; i<SIZE; i++) {
  cout << "a[" << i << "]=" <<a[i]<< endl;
}
```

Important Considerations

- Don’t Exceed Array Bounds
  - OutOfBounds Errors Cause Problems
- Typically, Bounds Errors Come On The Last Iteration Going Over The Edge
- You Are Forewarned!

Time For Our First Demo!

- ArrayCode.cpp

(See Handout For Example 1)
Summarizing Our First Demo!

- Arrays Let You Work With Groups Of Data
- Carefully Track Array Size!

Arrays As Function Parameters

- Like Any Other lvalue, Array Elements Can Be Passed To Functions

```c
void print_value(int i);
const int SIZE=3;
int a[SIZE]={1,10,100};
for (int i=0; i<SIZE; i++) {
    print_value(a[i]);
}
```

Arrays As Function Parameters

Display 3.1: Function with an Array Parameter

```c
Sample Dialogue
Declaration
void fill(int a[], int size);
//Precondition: size is the declared size of the array a.
//Postcondition: The array a is filled with size integers

Sample Dialogue
Definition
void fill(int a[], int size) {
    cout << "Enter " << size << " numbers:"
    for (int i = 0; i < size; i++)
        cin >> a[i];
    cout << "The last array index used is: " << (size - 1) << endl;
}
```
Arrays As Function Parameters

• The Whole Array Can Also Be A Parameter To A Function
• Arrays Are Passed To Functions As Array Parameters
  – neither pass-by-value or pass-by-reference
  – closely mimics pass-by-reference
• If A Function Changes Element Value, These Changes Will Be Seen By The Caller

void fill_up(int items[], int length);
const int SIZE=3;
int a[SIZE]={1,10,100};
fill_up(a, SIZE);
Arrays As Function Parameters

- Formal Parameter Syntax: `type name[]`
- Actual Parameter Syntax: `name`

```c
void fill_up( int items[], int length );
const int SIZE=3;
int a[SIZE]={1,10,100};
fill_up( a, SIZE );
```

Observations

- Since The Array Parameter Definition Lacks Array Size Value, It Is Always A Good Idea To Pass The Size Of The Array As An Extra Argument

```c
void fill_up( int items[], int length );
int a[5], b[10];
fill_up( a, 5 );
fill_up( b, 10 );
```
Observations

• Since The Array Parameter Definition Lacks Array Size Value, It Is Always A Good Idea To Pass The Size Of The Array As An Extra Argument

```c
void fill_up( int items[], int length );
```

```c
int a[5], b[10];
fill_up( a, 5 );
fill_up( b, 10 );
```

Observations

• When Arrays Are Passed To Functions, Elements Changed By The Function Are Visible To The Caller
• Array Parameters Are Kinda Pass-by-Reference
  – No copies of the individual elements are made
  – Changes to any elements will be seen by the caller

Array As Function Parameters

• What Is Really Passed?
• Think Of Array As 3 "Parts"
  – Address Of First Element (arrName[0])
  – Array Base Type
  – Size Of Array
• Only 1st Piece Is Passed!
  – Just The Beginning Address Of The Array
  – Very Similar To “Pass By Reference”
**const Array Arguments**

- If You Know The Function Will Not Change The Array Values, Use `const` Modifier

```c
void print(const int items[], int length);
```

**Typical Array Operations**

- Searching
- Sorting
Searching An Array

Display 5.6 Searching an Array

1 //Search a partially filled array of nonnegative integers.
2 #include <iostream>
3 using namespace std;
4 const int DECLARED_SIZE = 20;
5 void FillArray(int &arr, int size, bool numbered);
6 //Precondition: size is the declared size of the array a.
7 //Postcondition: numbered is the number of values stored in a.
8 //Any element of a[0..numbered-1] have been filled with
9 //nonnegative integers read from the keyboard.
10 int Search(int &arr, int size, int target);
11 //Precondition: numbered is <= the declared size of a.
12 //If size <= numbered, then arr[0..numbered-1] have values.
13 //Returns the first index such that a[index] == target.
14 //Otherwise, returns -1.

Searching An Array

int main()
{
    int arr[DECLARED_SIZE], listSize, target;
    FillArray(arr, DECLARED_SIZE, listSize);
    cout << "Enter a number to search for: ";
    cin >> target;
    int result = Searcher(arr, listSize, target);
    if (result == -3)
        cout << "-3 is not on the list."
    else
        cout << "The target is stored in position ", result,
        " (Remember: The first position is 0)\n";

Searching An Array

int Search(int &arr, int size, int target)
{
    int element = arr[0];
    while (element != target) // (first index is "0")
    {---
        int index = Search(arr, size, target);
        if (index == -1)
            return -1;
        if (target == arr[index])
            return index;
        else
            element = arr[index];
    }
    return element;
}
Searching An Array

```java
49  if (head == null)
50     return -1;
51  else
52      return -1;
53 }
```

Sorted Array

Enter an integer greater than zero.

Visit the end of the list with a negative number.

Enter a number in the list in the list.

Enter a number to search for.

If found, return the position.

Else, return -1.

Searching for a negative number is handled by returning -1.

Note: Remember: The first position is 0.

Sorting An Array

Selection Sort

Display 3.7 Selection Sort

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Display 3.8 Sorting an Array

```java
1. //Tests the procedure sort.
2. #include <iostream>
3. using namespace std;
4. void FillArray(int &A, int size, int numberToBeFilled)
5. { //Precondition: size is the declared size of the array A.
6.     //Postcondition: numberToBeFilled is the number of values stored in A.
7.     for (int i = 0; i < numberToBeFilled; i++)
8.         cin >> A[i];
9.     } //End procedure FillArray.
10. //Precondition: numberToBeFilled == declared size of the array A.
11. //Postcondition: A is sorted.
```

```c++
void FillArray(int &A, int size, int numberToBeFilled)
{
    for (int i = 0; i < numberToBeFilled; i++)
        cin >> A[i];
}
```

```c++
//Precondition: numberToBeFilled == declared size of the array A.
//Postcondition: A is sorted.
```
Sorting an Array

```java
void fillArray(int[] array) { 
    System.out.println("Enter the numbers:");
    for (int i = 0; i < array.length; i++) { 
        System.out.print("Enter number "+i+": ");
        array[i] = input.nextInt();
    }
}
```

Sorting an Array

```java
void sortArray(int[] array) { 
    for (int i = 0; i < array.length; i++) { 
        System.out.print(array[i]+" ");
    }
}
```

Sorting an Array

```java
void swapArray(int[] array, int index1, int index2) { 
    int temp = array[index1];
    array[index1] = array[index2];
    array[index2] = temp;
}
```
Multidimensional Arrays

• C++ allows any number of indexes
  – Typically no more than two

Multidimensional Arrays

• Arrays with more than one index
  ```
  char page[30][100];
  ```
  • Two indexes: An "array of arrays"
  • Visualize as:
    ```
    page[0][0], page[0][1], ..., page[0][99]
    page[1][0], page[1][1], ..., page[1][99]
    ...
    page[29][0], page[29][1], ..., page[29][99]
    ```

Multidimensional Array Parameters

• Similar to one-dimensional array
  – 1st dimension size not given
    • Provided as second parameter
  – 2nd dimension size IS given
Multidimensional Array Parameters

- Example:
  ```cpp
def display(const char p[][100], int size1)
  {
    ...
  }
```

Multidimensional Array Parameters

- Example:
  ```cpp
  for (int i=0; i<size1; i++){
    for (int j=0; j<100; j++)
      cout << p[i][j];
    cout << endl;
  }
  ```

Summary

- Arrays
  - Array Parameters
  - Typical Array Operations
- Multidimensional Arrays