UCLA

CS 31:
Introduction To Computer Science I

Howard A. Stahl

UCLA

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

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

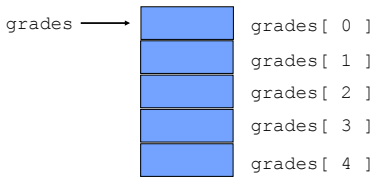
- Syntax:
  `type arrayname[ size ];`
- `type` referred to as the base type for all array elements
- `arrayname` is the variable name for the entire collection
- `size` is the number of elements allowed in the collection
  - indexes from `0` to `size-1`

Arrays

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

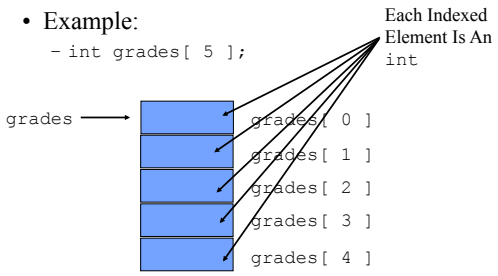
Arrays

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



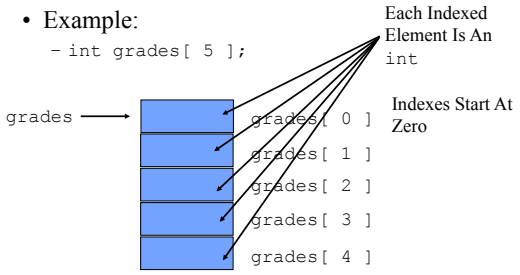
Arrays

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



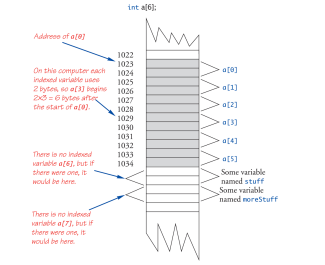
Arrays

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



Arrays In Memory

Display 5.2 **An Array in Memory**



Arrays

- Arrays Are An Ordered List
  - `grades[ 1 ]` precedes `grades[ 10 ]`
  - `grades[ 10 ]` precedes `grades[ 11 ]`
- Arrays Are Stored Contiguously In One Block
- Each Index Is An *lvalue* In Its Own Right
- `[ ]` Is Used To Declare And Access Arrays

Arrays

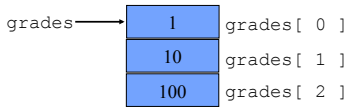
- Example:

```
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;
```

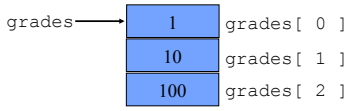


Arrays

- Example:

```
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

Display 5.1 Program Using an Array

```
1  //Reads in five scores and shows how much each
2  //score differs from the highest score.
3  #include <iostream>
4  using namespace std;
5  int main( )
6  {
7      int i, score[5], max;
8      cout << "Enter 5 scores:\n";
9      cin >> score[0];
10     max = score[0];
11     for (i = 1; i < 5; i++)
12     {
13         cin >> score[i];
14         if (score[i] > max)
15             max = score[i];
16         //max is the largest of the values score[0],..., score[i].
17     }
```

Array Program Example

```
18     cout << "The highest score is " << max << endl
19          << "The scores and their\n"
20          << "differences from the highest are:\n";
21     for (i = 0; i < 5; i++)
22         cout << score[i] << " off by "
23              << (max - score[i]) << endl;
24     return 0;
25 }
```

Sample Dialogue

Enter 5 scores:
5 9 2 10 6
The highest score is 10
The scores and their
differences from the highest are:
5 off by 5
9 off by 1
2 off by 8
10 off by 0
6 off by 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

Major Array Pitfalls!

- Indexes range from 0 to (array_size – 1)
- Example:
  `double temperature[24];`
  They are indexed as:temperature[0], temperature[1] …temperature[23]
- Common mistake:
  `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:
  ```
  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 A Maintainability
- Use That Constant Everywhere Y The Array

Only 1 Thing To Change If The Size Changes…

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;
```

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;
```

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;
```

```
const int SIZE=3;
int grades[SIZE]={1,10,100};
```

Array Initialization

- 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 Initialization

- 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

```
int grades[ ]={1,10,100};
```

Array Iteration

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

```
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

```
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 5.3 Function with an Array Parameter

Sample DialogueFunction Declaration

```
void fillUp(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

Sample DialogueFunction Definition

```
void fillUp(int a[], int size)
{
    cout << "Enter " << size << " numbers:\n";
    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

Arrays As Function Parameters

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

Arrays As Function Parameters

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

```
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*

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

const int SIZE=3;
int a[SIZE]={1,10,100};

fill_up( a, SIZE );
```

No Index Value

Observations

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

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

```
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

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

int a[5], b[10];

fill_up( a, 5 );
fill_up( b, 10 );
```

Function Can Be Called With Arrays Of Various Sizes

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

`const` Array Arguments

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

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

Typical Array Operations

- Searching
- Sorting

Searching An Array

Display 5.6 Searching an Array

```
//Searches a partially filled array of nonnegative integers.
#include <iostream>
using namespace std;
const int DECLARED_SIZE = 20;

void fillArray(int a[], int size, int& numberUsed);
//Precondition: size is the declared size of the array a.
//Postcondition: numberUsed is the number of values stored in a.
//a[0] through a[numberUsed-1] have been filled with
//nonnegative integers read from the keyboard.

int search(const int a[], int numberUsed, int target);
//Precondition: numberUsed is <= the declared size of a.
//Also, a[0] through a[numberUsed -1] have values.
//Returns the first index such that a[index] == target,
//provided there is such an index; otherwise, returns -1.
```

Searching An Array

```
int main( )
{
    int arr[DECLARED_SIZE], listSize, target;

    fillArray(arr, DECLARED_SIZE, listSize);

    char ans;
    int result;
    do
    {
        cout << "Enter a number to search for: ";
        cin >> target;

        result = search(arr, listSize, target);
        if (result == -1)
            cout << target << " is not on the list.\n";
        else
            cout << target << " is stored in array position "
                 << result << endl
                 << "(Remember: The first position is 0.)\n";
```

Searching An Array

Display 5.6 Searching an Array

```
        cout << "Search again?(y/n followed by Return): ";
        cin >> ans;
    } while ((ans != 'n') && (ans != 'N'));
    cout << "End of program.\n";
    return 0;
}

void fillArray(int a[], int size, int& numberUsed)
<The rest of the definition of fillArray is given in Display 5.5>
int search(const int a[], int numberUsed, int target)
{
    int index = 0;
    bool found = false;
    while ((!found) && (index < numberUsed))
    if (target == a[index])
        found = true;
    else
        index++;
```

Searching An Array

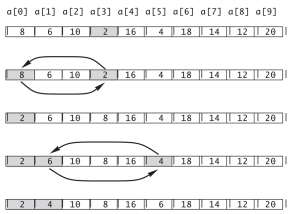
```
    if (found)
        return index;
    else
        return -1;
}
```

Sample Dialogue

```
Enter up to 20 nonnegative whole numbers.
Mark the end of the list with a negative number.
10 20 30 40 50 60 70 80 -1
Enter a number to search for: 10
10 is stored in array position 0
(Remember: The first position is 0.)
Search again?(y/n followed by Return): y
Enter a number to search for: 40
40 is stored in array position 3
(Remember: The first position is 0.)
Search again?(y/n followed by Return): y
Enter a number to search for: 42
42 is not on the list.
Search again?(y/n followed by Return): n
End of program.
```

Sorting An Array

Display 5.7 Selection Sort



Sorting An Array

Display 5.8 Sorting an Array

```
//Tests the procedure sort.
#include <iostream>
using namespace std;

void fillArray(int a[], int size, int& numberUsed);
//Precondition: size is the declared size of the array a.
//Postcondition: numberUsed is the number of values stored in a.
//a[0] through a[numberUsed - 1] have been filled with
//nonnegative integers read from the keyboard.
void sort(int a[], int numberUsed);
//Precondition: numberUsed <= declared size of the array a.
```

(continued)

Sorting An Array

Display 5.8 Sorting an Array

```
//The array elements a[0] through a[numberUsed - 1] have values.
//Postcondition: The values of a[0] through a[numberUsed - 1] have
//been rearranged so that a[0] <= a[1] <= ... <= a[numberUsed - 1].

void swapValues(int& v1, int& v2);
//Interchanges the values of v1 and v2.

int indexOfSmallest(const int a[], int startIndex, int numberUsed);
//Precondition: 0 <= startIndex < numberUsed. Reference array elements
//have values. Returns the index i such that a[i] is the smallest of the
//values a[startIndex], a[startIndex + 1], ..., a[numberUsed - 1].

int main( )
{
    cout << "This program sorts numbers from lowest to highest.\n";

    int sampleArray[10], numberUsed;
    fillArray(sampleArray, 10, numberUsed);
    sort(sampleArray, numberUsed);

    cout << "In sorted order the numbers are:\n";
    for (int index = 0; index < numberUsed; index++)
        cout << sampleArray[index] << " ";
    cout << endl;

    return 0;
}
```

Sorting An Array

```
void fillArray(int a[], int size, int& numberUsed)
        <The rest of the definition of fillArray is given in Display 5.5.>

void sort(int a[], int numberUsed)
{
    int indexOfNextSmallest;
    for (int index = 0; index < numberUsed - 1; index++)
    {//Place the correct value in a[index]:
        indexOfNextSmallest =
                    indexOfSmallest(a, index, numberUsed);
        swapValues(a[index], a[indexOfNextSmallest]);
        //a[0] <= a[1] <=...<= a[index] are the smallest of the original array
        //elements. The rest of the elements are in the remaining positions.
    }
}

void swapValues(int& v1, int& v2)
{
    int temp;
    temp = v1;
    v1 = v2;
```

Sorting An Array

Display 5.8 Sorting an Array

```
    v2 = temp;
}

int indexOfSmallest(const int a[], int startIndex, int numberUsed)
{
    int min = a[startIndex],
        indexOfMin = startIndex;
    for (int index = startIndex + 1; index < numberUsed; index++)
        if (a[index] < min)
        {
            min = a[index];
            indexOfMin = index;
            //min is the smallest of a[startIndex] through a[index]
        }

    return indexOfMin;
}
```

Sample Dialogue

```
This program sorts numbers from lowest to highest.
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.
80 30 50 70 60 90 20 30 40 -1
In sorted order the numbers are:
20 30 30 40 50 60 70 80 90
```

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:

```
void display(
 const char p[][100],
 int size1)
{
 …

}
```

Multidimensional Array Parameters

- Example:

```
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