

Arrays

Pointers and Dynamic Arrays

Learning Objectives

- Static Arrays
 - Introduction to Arrays
 - Arrays in Functions
 - Programming with Arrays
 - Multidimensional Arrays
- Pointers
 - Pointer variables
 - Memory management
- Dynamic Arrays
 - Creating and using
 - Pointer arithmetic

Introduction to Arrays

- Array definition:
 - A collection of data of same type
- First "aggregate" data type
 - Means "grouping"
 - int, float, double, char are simple data types
- Used for lists of like items
 - Test scores, temperatures, names, etc.
 - Avoids declaring multiple simple variables
 - Can manipulate "list" as one entity

Declaring Arrays

- Declare the array → allocates memory
int score[5];
 - Declares array of 5 integers named "score"
 - Similar to declaring five variables:
int score_0, score_1, score_2, score_3, score_4
- Individual parts called many things:
 - Indexed or subscripted variables
 - "Elements" of the array
 - Value in brackets called index or subscript
 - Numbered from 0 to size - 1

Accessing Arrays

- Access using index/subscript
 - `cout << score[3];`
- Note two uses of brackets:
 - In declaration, specifies SIZE of array
 - Anywhere else, specifies a subscript
- Size, subscript need not be literal
 - `int score[MAX_SCORES];`
 - `score[n+1] = 99;`
 - If n is 2, identical to: `score[3]`

Array Usage

- Powerful storage mechanism
- Can issue command like:
 - "Do this to i^{th} indexed variable"
where i is computed by program
 - "Display all elements of array score"
 - "Fill elements of array score from user input"
 - "Find highest value in array score"
 - "Find lowest value in array score"

Array Program Example:

Display 5.1 Program Using an Array (1 of 2)

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:

Display 5.1 Program Using an Array (2 of 2)

```
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

for-loops with Arrays

- Natural counting loop
 - Naturally works well "counting through" elements of an array

- Example:

```
for (int idx = 0; idx<5; idx++)  
{  
    cout << score[idx] << "off by "  
        << max - score[idx] << endl;  
}
```

- Loop control variable (idx) counts from 0 – 5

Major Array Pitfall

- Array indexes 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"

Defined Constant as Array Size

- Always use defined/named constant for array size
- Example:

```
const int NUMBER_OF_STUDENTS = 5;  
int score[NUMBER_OF_STUDENTS];
```
- Improves readability
- Improves versatility
- Improves maintainability

Uses of Defined Constant

- Use everywhere size of array is needed
 - In for-loop for traversal:

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



```
    // Manipulate array
```

```
}
```
 - In calculations involving size:

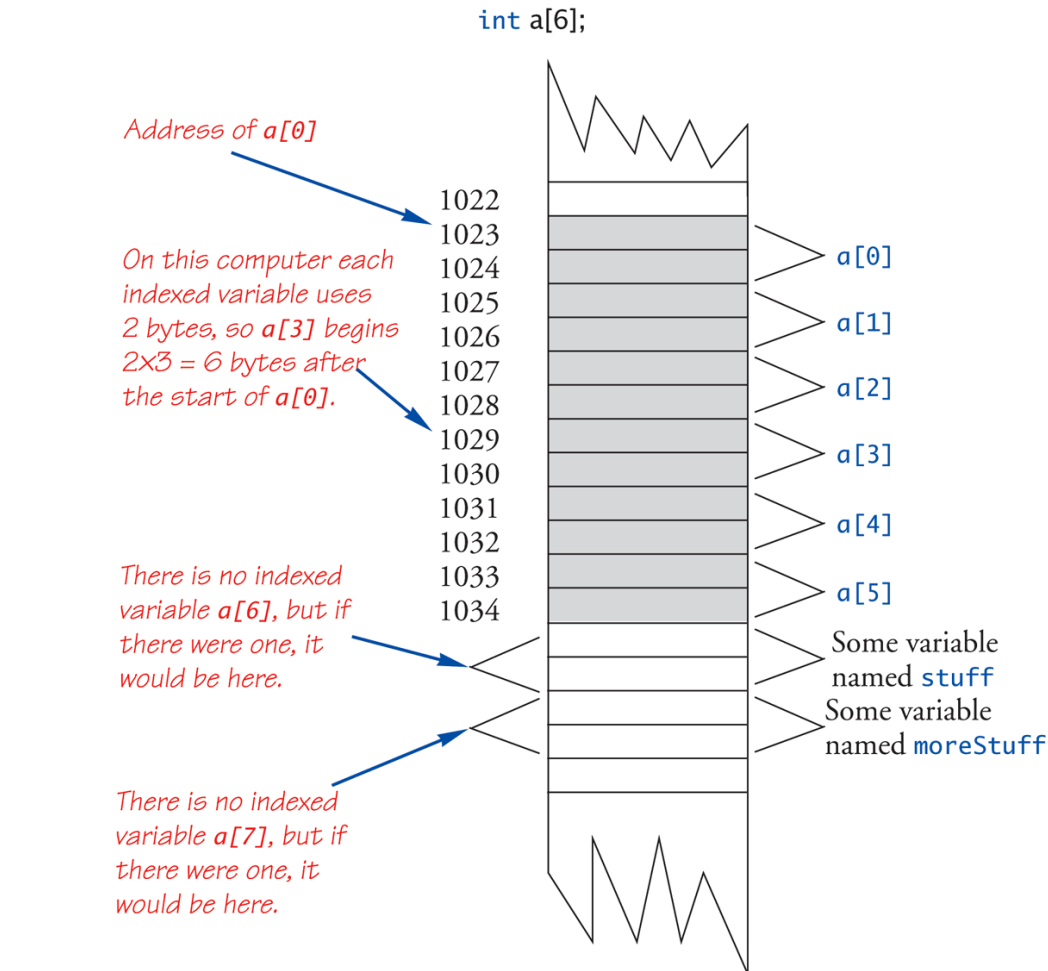
```
lastIndex = (NUMBER_OF_STUDENTS - 1);
```
 - When passing array to functions (later)
- If size changes → requires only ONE change in program!

Arrays in Memory

- Recall simple variables:
 - Allocated memory in an "address"
- Array declarations allocate memory for entire array
- Sequentially-allocated
 - Means addresses allocated "back-to-back"
 - Allows indexing calculations
 - Simple "addition" from array beginning (index 0)

An Array in Memory

Display 5.2 An Array in Memory



Initializing Arrays

- As simple variables can be initialized at declaration:

```
int price = 0;    // 0 is initial value
```

- Arrays can as well:

```
int children[3] = {2, 12, 1};
```

- Equivalent to following:

```
int children[3];  
children[0] = 2;  
children[1] = 12;  
children[2] = 1;
```

Auto-Initializing Arrays

- If fewer values than size supplied:
 - Fills from beginning
 - Fills "rest" with zero of array base type
 - `int b[4] = {5, 12, 11};`
 - Whats stored in `b[3]` ?
- If array-size is left out
 - Declares array with size required based on number of initialization values
 - Example:
`int b[] = {5, 12, 11};`
 - Allocates array `b` to size 3

Arrays in Functions

- As arguments to functions
 - Indexed variables
 - An individual "element" of an array can be function parameter
 - Entire arrays
 - All array elements can be passed as "one entity"
- As return value from function
 - Can be done → chapter 10

Indexed Variables as Arguments

- Indexed variable handled same as simple variable of array base type
- Given this function declaration:
`void myFunction(double par1);`
- And these declarations:
`int i; double n, a[10];`
- Can make these function calls:
`myFunction(i); // i is converted to double`
`myFunction(a[3]); // a[3] is double`
`myFunction(n); // n is double`

Entire Arrays as Arguments

- Formal parameter can be entire array
 - Argument then passed in function call is array name
 - Called "array parameter"
- Send size of array as well
 - Typically done as second parameter
 - Simple int type formal parameter

Entire Array as Argument Example:

Display 5.3 Function with an Array Parameter

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;  
}
```

Entire Array as Argument Example

- Given previous example:
- In some main() function definition, consider this calls:

```
int score[5], numberOfScores = 5;  
fillup(score, numberOfScores);
```

- 1st argument is entire array
- 2nd argument is integer value
- Note no brackets in array argument!

Array Parameters

- May seem strange
 - No brackets in array argument
 - Must send size 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);
```

Functions that Return an Array

- Functions cannot return arrays same way simple types are returned
- Requires use of a "pointer"
- Will be discussed later

Programming with Arrays

- Plenty of uses
 - Partially-filled arrays
 - Must be declared some "max size"
 - Sorting
 - Searching

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]
- C++ allows any number of indexes
 - Typically no more than two

Multidimensional Array Parameters

- Similar to one-dimensional array

- 1st dimension size not given
 - Provided as second parameter
- 2nd dimension size IS given

- Example:

```
void DisplayPage(const char p[][100], int sizeDimension1)
{
    for (int index1=0; index1<sizeDimension1; index1++)
    {
        for (int index2=0; index2 < 100; index2++)
            cout << p[index1][index2];
        cout << endl;
    }
}
```

Arrays Example: arrayExample.cpp

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    int age[4];
    age[0]=23;
    age[1]=34;
    age[2]=65;
    age[3]=74;
    for(i=0; i<4; i++)
    {
        cout <<"Element: "<< i <<" Value of age: "<< age[i] <<"\n";
    }
    return 0;
}
```

declare an integer array containing 4 elements

Note: The number in the square brackets [] is the position number of a particular array element. The position numbers begins at 0

Output:

```
Element: 0 Value of age: 23
Element: 1 Value of age: 34
Element: 2 Value of age: 65
Element: 3 Value of age: 74
```