Unit 2c – Arrays

Mark Redekopp
Unit 2

- **Unit 1**: Scalar processing  
  – aka IPO=Input-Process-Output Programs

- **Unit 2**: Linear (1D) Processing

- **Unit 3**: Multidimensional Processing

- **Unit 4**: Divide & Conquer  
  (Functional Decomposition)

How do we store these data sets
Motivating Example

Suppose I need to store the grades for all students so I can then compute statistics, sort them, print them, etc.

I would need to store them in variables that I could access and use

– This is easy if I have 3 or 4 students
– This is painful if I have many students

```cpp
int main()
{
    int score1, score2, score3;
    cin >> score1 >> score2 >> score3;

    // output scores in sorted order
    if(score1 < score2 &&
        score1 < score3)
    {
        /* score 1 is smallest */
    }
    /* more */
}
```

```cpp
int main()
{
    int score1, score2, score3,
        score4, score5, score6,
        score7, score8, score9,
        score10, score11, score12,
        score13, score14, score15,
    /* ... */
        score139, score140;
    cin >> score1 >> score2 >> score3
        >> score4 >> score5 >> score6
    /* ... */
}```
Control vs. Data Structures

• Language constructs that allow us to make decisions are referred to as **control structures**
  – The common ones are: *if statements, while loops, for loops*

• We also need ways to store our data so we can access it easily and efficiently

• Arrays are the simplest data structure and the only one that C/C++ supports natively
  – Other data structures are available through other library code (but arrays need no additional code included)
Array Basics

• An array are a **named collection** of ordered variables of the **same type** that are accessed with an **index** and stored **contiguously** in memory
  – **Named collection**: One name to refer to the collection of variables
  – **Ordered**: There is a first and a last and one comes before another
  – Accessed with an **index**: Each variable is accessed with its position/index (using [] brackets)
  – **Same Type**: Variables in one array must all be the same type (one array can't store doubles and ints)

```cpp
int main()
{
    int scores[140];
    // allocates 140 integers
    // with garbage values

    for(int i=0; i < 140; i++){
        cin >> scores[i];
    }
}
```

<table>
<thead>
<tr>
<th>Addr:</th>
<th>520</th>
<th>524</th>
<th>528</th>
<th>...</th>
<th>1076</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td></td>
<td>[139]</td>
</tr>
<tr>
<td>scores:</td>
<td>96</td>
<td>84</td>
<td>93</td>
<td>...</td>
<td>90</td>
</tr>
</tbody>
</table>

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.
Accessing An Element

- Once an array is declared, there is nothing "special" about it. Each variable must be initialized and accessed 1 at a time.
- To access an individual variable/element of an array of size $n$, use the name of the array followed by square brackets containing ANY expression (constant, variable, arithmetic) that will evaluate to an index from 0 to $n-1$
  
  — Note: Indexing starts at 0

```c
int main()
{
    int x = 1, myval = 5;
    int scores[10];
    // allocates 10 integers

    scores[4] = 73;
    scores[x] = 82;
    // sets scores[1]

    scores[2*x + 1] = 93;
    // sets scores[3]

    scores[1+max(x,myval)] = 88;
    // sets scores[6]
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scores:</td>
<td>??</td>
<td>82</td>
<td>??</td>
<td>93</td>
<td>73</td>
<td>??</td>
<td>88 ...</td>
</tr>
</tbody>
</table>
Loops and Arrays (1)

• The real power of arrays is found when you combine them with loops

• Use the loop control variable (int i) to serve as the index of the array entry to be modified or accessed
  – Whether the array has 1 or 1,000,000 elements, our code size does not grow

```cpp
int main()
{
    int x = 1, myval = 5;
    int scores[100];
    // allocates 100 integers

    // initialize all to 0
    for(int i = 0; i < 100; i++){
        scores[i] = 0;
    }

    // OR.. read in all entries
    for(int i = 0; i < 100; i++){
        cin >> scores[i];
    }
}
```

Computer Memory

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scores:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Loops and Arrays (2)

• How could we determine the average score?

```cpp
int main()
{
    int scores[100];
    /* ... fill in the data ... */

    // Average all values
    for(int i=0; i < 100; i++){
        // ... 
    }
    cout << _______________ << endl;
    return 0;
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scores:</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Computer Memory
Loops and Arrays (3)

• How could we determine the max score?

```cpp
int main()
{
    int scores[100];
    /* ... fill in the data ... */

    // Find maximum

    for(int i=0; i < 100; i++){
    }
    cout << "Max: " << _____ << endl;
    return 0;
}
```
When Do We Need Arrays?

- Arrays **can** store many related data items of the **same type**
- A better question is when do we **need** to store these related data items in an array?
- We need arrays when we need to **revisit the data more than once**
  - If we just want to find the min/max or average we could just get the data from the user and update the sum or min/max as we go and not need to store each data item
  - Don't introduce arrays where they are not needed

```cpp
int main()
{
    int scores[100];
    // Get the data
    for(int i=0; i < 100; i++){
        cin >> scores[i];
    }
    // Average all values
    int sum = 0;
    for(int i=0; i < 100; i++){
        sum += scores[i];
    }
    cout << sum / 100.0 << endl;
    return 0;
}
```
```cpp
int main()
{
    int val, sum = 0;
    // Get the data & average it
    // at the same time
    for(int i=0; i < 100; i++){
        cin >> val;
        sum += val;
    }
    cout << sum / 100.0 << endl;
    return 0;
}
```
Index vs. Value

- The expression in the square brackets is an index.
- Using `array[index]` yields the data/value in the array at that index.
- An index can be ANY EXPRESSION, even the value from an array or the return value from a function.
- For an array declared to be size n, only indices 0 to n-1 are legal.

```c
int main()
{
    int scores[20];
    /* ... fill in the data ... */
    int i = 1;
    int x = scores[2*i + 1]; // x=_
    int y = scores[ scores[1] ]; // y=_
    int z = scores[ max(4,2) ]; // z=_
    return 0;
}
```

Computer Memory

<table>
<thead>
<tr>
<th>Addr:</th>
<th>520</th>
<th>524</th>
<th>528</th>
<th>532</th>
<th>536</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
| scores: (values) | 9 | 0 | 7 | 8 | 6 | ...

Value/data

scores[2*i+1]
A Common Error

- Care must be taken to ensure no index is used that will lead to an out-of-bounds access
  - Such an access will either corrupt other data or cause the program to crash!
  - These are often known as segmentation faults. When you see one, your first thought should be to check for a bad array index!

```c
int main()
{
    int scores[20];
    /* ... init in the data ... */
    int i;
    for(i=0; i <= 20; i++){
        // wrong?
        scores[i] = 0;
    }
    cin >> i;
    // what could happen here..not safe
    scores[i] = 100;

    // safe
    if( i >= 0 && i < 20 ){
        scores[i] = 100;
    }
    return 0;
}
```

### Computer Memory

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scores: (values)</td>
<td>9 0 7 ... 6 ?</td>
</tr>
</tbody>
</table>
Important C/C++ Rule: Array Size

• C/C++ needs to know the **SIZE of the array** when the program is **compiled**, not when it is **run**.
  – This implies the size of the array must be **ONE, FIXED (or constant) size** everytime the program is run.

• For this course, we will just allocate a **LARGE array** of the maximum size potentially needed and then **use only a portion of it as the program runs**
  – Future courses will teach you how to deal with this correctly and not waste array space.

```cpp
int main()
{
    // GOOD!!
    int data[24]; // 24 known at compile time

    // BAD!!
    int n;
    cin >> n;
    int data[n]; // n not known at compile time
}
```

```cpp
int main()
{
    int data[100]; // max needed
    int n;
    cin >> n;

    for(int i=0; i < n; i++)
    {
        cin >> data[i];
    }
}
```
Exercises 1

• cpp/arrays/fibonacci
• cpp/arrays/sorted
ARRAY DETAILS
Character Arrays

- C-Strings are stored as character arrays
  - Each character consumes 1 element in the array
  - Ends with the null character (e.g. 0 decimal or '\0' ASCII)
- Can use `cin` and `cout` with a character array to get a string from the keyboard or output a string
  - `cin` and `cout` will loop over the array inputting or printing one character at a time

```cpp
int main()
{
    char str1[7] = "CS 102";
    /* Initializes the array to "CS 102"*/
    str1[5] = '3'; // now "CS 103"
    cout << str1 << endl; // prints "CS 103"
    cin >> str1; // get a new string from the user (suppose user types "hello")
    cout << str1;
}
```

Program Output:

```
CS 103
hello
```
Initializing Arrays With Constants

- Arrays can be initialized with **constants** when they are declared.
- To do so, use an **initialization list** which is a comma separated list of constants in {...}.
  - **Exception:** If **fewer** values are provided than the size of the array, remaining elements will be filled with 0s.
- If an initialization list is provided you need not specify the size in the square brackets (i.e. just use empty []) as the compiler can figure out what size the array must be by counting the initial values.

```c
int main()
{
    int data[5] = {9, 7, 8, 9, 5};
    double dec[4] = {0.25, 0.3};
    char str1[3] = {'C', 'S', '\0'};
    // For char arrays easier to use ""
    char str2[3] = "CS";
    // str2 initialization is same as str1
}
```

```c
int main()
{
    int data[] = {9, 7, 8, 9, 5};
    // allocates array of size 5

    double dec[] = {0.25, 0.3, 0.18, 0.2};
    // allocates array of size 4

    char str2[] = "CS";
    // allocates array of size 3
}
```
Exercises 2

• cpp/arrays/sumpairs
  – Given an array of size n (n is even), output the sum of the
  • first and last
  • 2\textsuperscript{nd} and 2\textsuperscript{nd} to last
  • 3\textsuperscript{rd} and 3\textsuperscript{rd} to last

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scores:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Computer Memory
SOLUTIONS
Loops and Arrays (2)

• How could we determine the average score?

```cpp
int main()
{
    int scores[100];
    /* ... fill in the data ... */

    // Average all values
    int sum = 0;
    for(int i=0; i < 100; i++){
        sum += scores[i];
    }
    cout << (double)sum / 100 << endl;

    // Find maximum
    int max = 0;
    for(int i=0; i < 100; i++){
        if( scores[i] > max)
            max = scores[i]
    }
    cout << "Max: " << max << endl;
    return 0;
}
```

Loops and Arrays (3)

• How could we determine the max score?

```cpp
int main()
{
    int scores[100];
    /* ... fill in the data ... */

    // Find maximum
    int max = -1;
    for(int i=0; i < 100; i++) {
        if( scores[i] > max)
            max = scores[i]
    }
    cout << "Max: " << max << endl;
    return 0;
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scores:</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Computer Memory
Index vs. Value

• The expression in the square brackets is an index
• Using `array[index]` yields the data/value in the array at that index
• An index can be **ANY EXPRESSION**, even the value from an array or the return value from a function
• For an array declared to be size `n`, only indices 0 to n-1 are legal

```c
int main()
{
    int scores[20];
    /* ... fill in the data ... */

    int i = 1;
    int x = scores[2*i + 1];    // x=8
    int y = scores[ scores[1] ]; // y=9
    int z = scores[ max(4,2) ]; // z=6
    return 0;
}
```

Computer Memory

<table>
<thead>
<tr>
<th>Addr:</th>
<th>520</th>
<th>524</th>
<th>528</th>
<th>532</th>
<th>536</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>scores: (values)</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>