CS135; 1D Arrays and Strings

Gael Zarco

April 18, 2025

Structured Data Types contain data where each item is a collection of other data items.

- Simple data structures are the building blocks of structured data types.

1 Arrays

An **Array** is a collection of a fixed number of components (also called elements) all of the same data type and in contigous (adjacent) memory space. A **One-Dimensional Array** is an array in which the components are arranged in a list form.

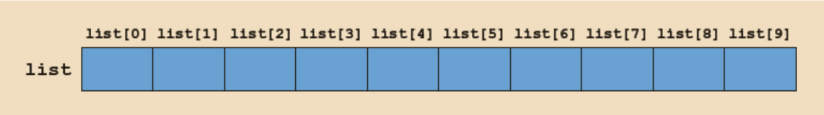
Listing 1: 1D Array Syntax

```
dataType arrayName[intExp];

// Example
int list[5];    // Declared array 'list' of 10 elements
```

`intExp` specifies the number of components in the array and can be any constant expression that evaluates to a positive integer. The Example above delcares an array `list` of *10* components.

- The components are `list[0]`, `list[1]`, ..., `list[9]`.
- Declares a total of 10 variables

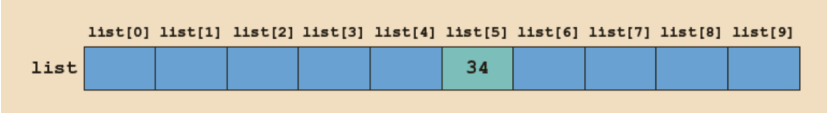


Listing 2: 1D Array Assignment

```
list[5] = 34;
```

This expression stores *34* in `list[5]`, which is the *sixth* component of the array `list`.

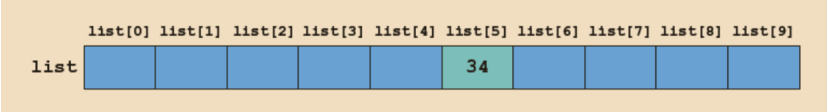
- You can use `i` to index into an array.
  - `list[i]`



Listing 3: 1D Array Assignment Cont

```
list[3] = 10;
list[6] = 35;

// Add the contents of list[3] and list[6] and store in list[5]
list[5] = list[3] + list[6];
```



1.1 Processing One-Dimensional Arrays

Listing 4: Read Data Into 1D Array

```
for (i = 0; i < 10; i++)
  cin >> list[i];
```

Listing 5: Print Data From 1D Array

```
for (i = 0; i < 10; i++)
  cout << list[i];
```

Listing 6: Find Sum & Avg From 1D Array

```
int sum = 0;
int avg;

for (i = 0; i < 10; i++)
  sum = sum + list[i];

avg = sum / 10;
```

Listing 7: Find Largest Element in 1D Array

```
int maxIdx = 0;

for (int i = 1; i < 10; i++)
```

```
4       if (list[maxIdx] < list[i])
5         maxIdx = i;
6
7     int largestInt = list[maxIdx];
```

1.2 Array Index Out of Bounds

Listing 8: Array Index Example

```
1     double num[10];
2     int i;
```

The component `num[i]` is *valid* or **In Bounds** if `index`:

- $0 \leq$ index $\leq$ ARRAY_SIZE $- 1$.
- `index` is not negative or greater than ARRAY_SIZE $- 1$.
  - It is **Out of Bounds** in this event.
  - C++ does not check whether the index value is within range; this is the programmer's responsibility.

1.3 Array Initialization During Declaration

An array can be initialized while being declared

Listing 9: Array Initialization Example

```
1     double sales[5] = {12.25, 32.50, 16.90, 23, 45.68};
2
3     // Not necessary to specify the size when initializing
4     double sales[] = {12.25, 32.50, 16.90, 23, 45.68};
```

1.4 Partial Initialization of Arrays During Declaration

Listing 10: Partial Array Initialization

```
1     int list[10] = {5, 6, 3};
```

The first three components of `list` are `list[0] = 5, list[1] = 6, list[2] = 3`, and the rest are set to the default of 0.

1.5 Restrictions on Array Processing

C++ does not allow **Aggregate Operations** on an array. Aggregate operations on an array are any operations that manipulate the entire array as a single unit.

Listing 11: Illegal Aggregate Operation on Array

```
int myList[5] = {0, 4, 8, 12, 16};
int yourList[5];

// illegal
yourList = myList;
```

1.6 Arrays as Parameters to Functions

In C++, arrays are passed as parameters to functions by **Reference Only**. You do **not** use the `&` symbol when declaring an array as a formal parameter.

Listing 12: Arrays as Formal Parameters

```
void initialize(int list[], int listSize);
```

1.7 Constant Arrays as Formal Parameters

You can use `const` keyword in the declaration of a formal param to prevent the function from changing the actual param.

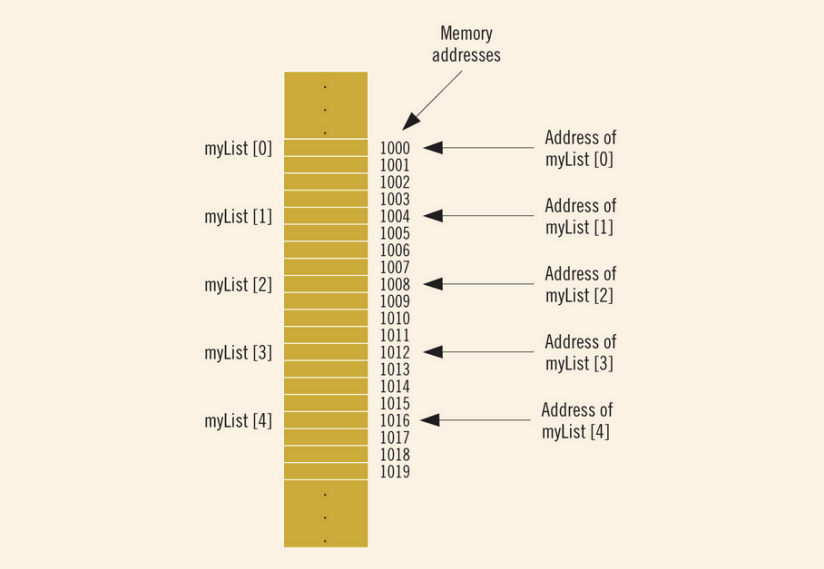
Listing 13: Constant Arrays as Formal Parameters

```
void foo(int x[], const int y[], int sizeX, int sizeY);
```

1.8 Base Address of an Array and Array in Computer Memory

The **Base Address** of an array is the address (memory location) of the first array component.

- In 1D arrays, the base address is list[0];



1.9 Functions Cannot Return a Value of the Type Array

C++ does not allow functions to return a value of type array.

1.10 Integral Data Type and Array Indices

C++ allows any integral type to be used as an array index.

Listing 14: Improved Code Readability

```
enum paintType { GREEN, RED, BLUE, BROWN, WHITE, ORANGE, YELLOW };
double paintSale[7];

paintSale[RED] = paintSale[RED] + 75.69;
```

1.11 Other Ways to Declare Arrays

Listing 15: Declaration Using Existing Variable

```
const int NO_OF_STUDENTS = 20;
int testScores[NO_OF_STUDENTS];
```

Listing 16: Declaration with `using`

```
const int SIZE = 50;          // Line 1
using list = double[SIZE];    // Line 2
list yourList;                // Line 3
list myList;                  // Line 4
```

2 Searching an Array for a Specific Item

Sequential Search (*linear search*):

1. Searching a list for a given item, starting from the first element.
2. Compare each element in the array with the value being searched.
3. Continue to search until item is found or no more data is left.

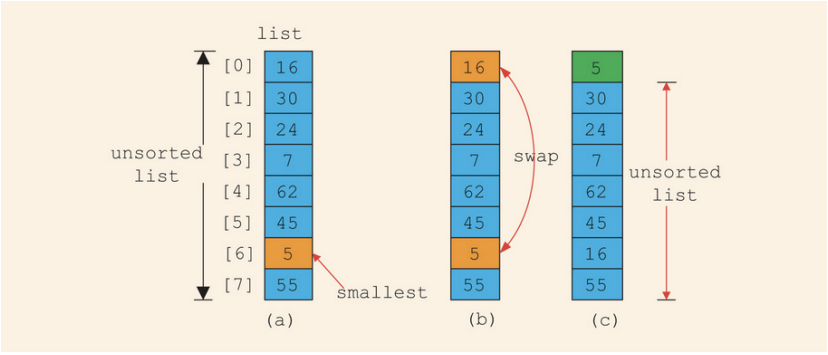
Listing 17: Simple Array Search

```
1   int seqSearch(const int list[], int listLength, int searchItem) {
2     int  loc   = 0;
3     bool found = false;
4
5     while (loc < listLength && !found) {
6       if (list[loc] == searchItem) {
7         found = true;
8       } else {
9         ++loc;
10      }
11    }
12
13    return found ? loc : -1;
14  }
```

3 Sorting

Selection Sort is rearranging the list by selecting an element and moving it to its proper position.

1. Find the smallest element in the unsorted portion of the list.
2. Move it to the top of the unsorted portion by swapping with current element.
3. Start again with the rest of the list.



4 Auto Declaration and Range-Based `for` Loops

Modern C++ allows for **Auto** declaration of variables

- Data type does not need to be specified.

```
auto num = 15;
```

The compiler deduces `num` to be of type `int`.

Listing 18: Range-Based `for` Loop

```
double list[25];
double sum = 0;

for (double num : list) {  // read as "for each num in list"
    sum += num;
}
```

5 C-Strings (Character Arrays)

A **Character Array** is an array whose components are of type `char`.

- C-strings are *null-terminated* (`'\0'`) character arrays.
- Examples
  - 'A' is the character A
  - "A" is the C-string A
  - *Note*: "A" represents two characters. 'A' and `'\0'`.

Listing 19: C-String Declaration

```
char name[16];
```

C-strings are null-terminated and `name` has 16 components, the largest string it can store has 15 characters. If you store a string whose length is less than the array size, the last components are unused.

Listing 20: Omitting Size of Array During Initialization

```
char name[] = "John";
```

Declares an array of length *5* and stores the C-string "John" in the array. Useful `string` functions:

- `strcopy`

- `strncpy`
- `strcmp`
- `strlen`

5.1 String Comparison

C-Strings are compared character by character using the collating system sequence. Use the `strcmp` function to compare strings.

If using ASCII char set:

- "Air" ¡ "Boat"
- "Air" ¡ "An"
- "Bill" ¡ "Billy"
- "Hello" ¡ "hello"

5.2 Reading and Writing Strings

Most array rules apply to C-strings (which are `char` arrays). However, C++ **DOES** allow aggregate ops for the input and output of C-strings.

5.3 String Input

Listing 21: String Input Example

```
1    cin >> name;
```

Stores the next input C-string into `name`.

Listing 22: Read Strings with Blanks with `get`

```
1    cin.get(str, m + 1);
```

- When executed, stores the next `m` characters into `str`, but the newline character is not stored.
- If input string has fewer than `m` characters, reading stops at newline character.

5.4 String Output

Listing 23: String Output Example

```
cout << name;
```

- `<<` continues to write the contents of `name` until it finds a `null` character.
- If `name` does not contain a `null` character, then strange output may occur as it will continue to output data from memory until a `null` character is found.

5.5 `string` Type and Input/Output Files

Argument to open function must be a null-terminated string (a C-string).

- If using a string var for the name of an I/O file, the value must first be converted to a C-string before calling `open`.
- Use the `c_str` function to convert.

Listing 24: `c_str` Syntax

```
strVar.c\_str();
```

Where `strVar` is a variable of type `string`.