Arrays

An array is a list containing

- a fixed number of entries AND
- entries of all the same type

```plaintext
int a[5]; - declares an array of 5 ints
float c[8]; - declares an array of 8 floats
```

Zero-indexing

- An array with \( n \) elements is accessed with indices 0 through \( n-1 \)

  ```plaintext
dailyTemps[4] - accesses fifth element of the dailyTemps array
```

Memory allocation

Declaration of array with \( n \) elements takes contiguous chunks of memory to hold each of the \( n \) elements

```plaintext
int scores[3];
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>04902340</td>
<td>12</td>
</tr>
<tr>
<td>04902348</td>
<td>89</td>
</tr>
<tr>
<td>04902356</td>
<td>543</td>
</tr>
<tr>
<td>04902364</td>
<td></td>
</tr>
<tr>
<td>04902372</td>
<td></td>
</tr>
<tr>
<td>04902380</td>
<td></td>
</tr>
<tr>
<td>04902388</td>
<td></td>
</tr>
<tr>
<td>04902396</td>
<td></td>
</tr>
<tr>
<td>04902404</td>
<td></td>
</tr>
<tr>
<td>04902412</td>
<td></td>
</tr>
<tr>
<td>04902420</td>
<td></td>
</tr>
<tr>
<td>04902428</td>
<td></td>
</tr>
</tbody>
</table>
Declaration

Array must be declared with constant number of entries

```cpp
const int gradeSize=26;
char grades[gradeSize];
float heights[26];
```

Initialization

- Entries of array can be initialized with bracketed list
- Un-filled slots will default to zero after initialization

```cpp
float heights[26]={5.5, 4.9, 6, 4.5};
cout << heights[1] << " " << heights[6]; // Outputs: 4.9 0
```

Arrays and loops

for loops are well-structured to handle arrays

```cpp
const int gradeSize=26;
char grades[gradeSize];
for(int i=0; i<gradeSize; i++) {
    cout << grades[i] << endl;
}
```

What does this code do?

```cpp
int a[5]={1,3,6,4,2};
cout << a[3] << endl;
```

```cpp
int i=1;
while(i<4) {
    cout << a[i+1]-a[i] << endl;
    i++;
}
```

What does this code do?

```cpp
int b[5], size_b=0;
int i=0;
while(i<4) {
    if (a[i]>3) {
        b[size_b]=a[i];
        size_b++;
    }
    i++;
}
```

Out-of-range indexing

- An index value not allowed by array declaration is “out of range”
  ```cpp
  char a[10];
cin >> a[10]; // out of range!
  ```

- Out-of-range index produces no compiler error, but can cause serious program problems
  - Reading/writing incorrect spots in memory
Out-of-range indexing

```c
int scores[4]={1,2}, idNum;
idNum=34253;
scores[5]=12;
cout << idNum;
```

```c
1 2 0 0
idNum 12
??? 34253 ???
```

Array elements in functions

- Array element accepted as normal function argument
- If
  ```c
  int my_function(int n);
  int a[10], b;
  Then can execute:
  b=my_function(a[2]);
  b=my_function(a[5]);
  ```

Arrays in functions

We can pass full array to a function
- Function declaration
  ```c
  void printList(int list[], int size);
  ```
- Call
  ```c
  int list[5], size=5;
  printList(list, size);
  ```

Roughly “pass by reference”

- By default, elements of input array can be changed by function
  ```c
  void getList(char a[], int size);
  ```
  // Precondition: Receives blank list
  // of chars and size of list
  // Postcondition: list of chars is
  // filled by user

“Variable” array size

Can simulate a user-selected array size
- Define max_size of array
- Define user-selected array_size
  ```c
  const int max_size=500;
  int array_size, scores[max_size];
  cout << "What is our array size? ";
  cin >> array_size;
  ```
Programming with arrays

• Search – is number x in my array?
• Sort – arrange numbers from small to large

Sorting method

Start with:
• Unsorted list of numbers U
• Empty list E
Method:
• Find smallest number in U
• Place smallest number in E (E[0]=smallest(U);
• Find second-smallest number in U
• Place second smallest number in E
• ...keep going

Functions: const arrays

• Can insist array values remain unchanged:
  • Function declaration
    void showAll(const int list[], int size);
• Call
  int list[5], size=5;
  showAll(list, size);

More on const arrays

• If formal parameter is const array, cannot input to another function as non-const
  void showAll(const int list[], int size);
  void getAll(int list[], int size);
  ...
  void showAll(const int list[], int size) {
    getAll(list, size); // ERROR!
    ...
    // Display list elements
    return;
  }

Selection sort

• Formal parameters: list of numbers list and its size list_size
• Extra functions:
  - min_index(int a[], int start_index, int size); - finds index of minimum value in array a from start_index to the end of the array
  - swap(int& num1, int& num2); - swaps the values in num1 and num2

Selection sort: sorting strategy

Start with element n=0 of list (first element)
• Find indexSmall = min_index(list, n+1, list_size)
• If list[indexSmall] < list[n], swap the two elements
• Proceed to next list element, n++
Starting list:
2 0 8 5
n=0: indexSmall=1, list[indexSmall]==0 (<2), Swap!
0 2 8 5
n=1: indexSmall=3, list[indexSmall]==5 (>2), No swap
0 2 8 5
n=2: indexSmall=3, list[indexSmall]==5 (<8), Swap!
0 2 5 8

Multi-dimensional arrays
- Storing a table of data
const int numStudents=5, numTests=3;
char grades[numStudents][numTests];
grades[2][0] = 'A';
grades[3][0] = 'B';

```
   ???  ???  ???
   ???  ???  ???
   A     ???  ???
  B     ???  ???
  ???  ???  ???
```

“Array of arrays”

char grades[5][3] treated as array with 5 entries – each “entry” is a 3-element char array

```
grades[0]  A  0  1  2
          B  A
grades[1]  C  0  1  2
          C  B
grades[2]  A  0  1
          B
```

Passing multi-dimensional arrays

```c
void print_list(const char list[][3],
               int num_rows);
```

Size of inner array must be specified