CISC 1600/1610 Computer Science I

Arrays

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

Data types

Single pieces of information
- one integer – `int`
- one symbol – `char`
- one truth value – `bool`

Multiple pieces of information
- group of symbols – `string`
- group of anything – ***array***

An array is a list containing
- a fixed number of entries **AND**
- entries of all the same type

`int a[5];` - declares an array of 5 `int`s

`float c[8];` - declares an array of 8 `float`s

Array syntax

- Declaring an array:
```
char grades[4];
```
- Initializing an array:
```
char grades[4]={'A', 'B', 'A', 'C'};
```
- Accessing an array element:
```
cout << grades[2];
```

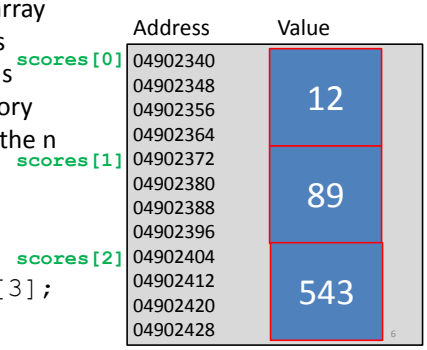

Zero-indexing

- An array with **n** elements is accessed with indices 0 through n-1
- `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

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



Declaration

Array must* be declared with constant number of entries

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

7

Initialization

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

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

8

Arrays and loops

for loops are well-structured to handle arrays

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

9

What does this code do?

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

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

10

What does this code do?

```
int a[5]={1,3,6,4,2}
int b[5], size_b=0;

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

11

Out-of-range indexing

- An index value not allowed by array declaration is "out of range"

```
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

12

Out-of-range indexing

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

	???
	???
scores[0]	1
scores[1]	2
scores[2]	0
scores[3]	0
	???
idNum	~~34253~~ 12
	???

13

Array elements in functions

- Array element accepted as normal function argument

If

```
int my_function(int n);
int a[10], b;
```

Then can execute:

```
b=my_function(a[2]);
b=my_function(a[5]);
```

14

Arrays in functions

We can pass full array to a function

- Function declaration

```
void printList(int list[], int size);
```

- Call

```
int list[5], size=5;
printList(list, size);
```

15

Roughly "pass by reference"

- By default, elements of input array can be changed by function

```
void getList(char a[],int size);
// Precondition: Receives blank list
//   of chars and size of list
// Postcondition: list of chars is
//   filled by user
```

16

Roughly "pass by reference"

- Function will see variable type and memory location of first element
- **Useful** to include formal parameter reporting array size

```
void getList(char a[],int size);
// Precondition: Receives blank list
//   of chars and size of list
// Postcondition: list of chars is
//   filled by user
```

17

"Variable" array size

Can simulate a user-selected array size

- Define `max_size` of array
- Define user-selected `array_size`

```
const int max_size=500;
int array_size, scores[max_size];
cout << "What is our array size? ";
cin >> array_size;
```

18

Programming with arrays

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

19

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

20

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);
```

21

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;
}
```

22

Selection sort

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

23

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++

24

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**

25

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';
```

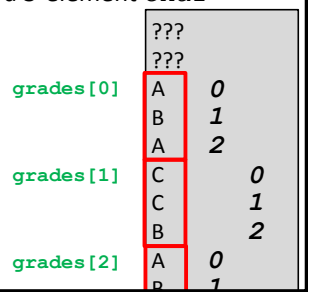
grades

???	???	???
???	???	???
A	???	???
B	???	???
???	???	???

26

"Array of arrays"

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



Passing multi-dimensional arrays

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

Size of inner array must be specified

28