CSE 142
Programming I

Arrays

The Road Ahead

- Read Chapter 8
- Today: Motivation and Introduction—No Code
- Wednesday: Code

Motivation

- So far we've only used small numbers of variables
  - ➤3 piles of stones
  - ➤6 types of food
- We had a name for each value
- What if we had 1000 piles?

Motivation: Sorting

- What if we had some numbers

```
12, 1, 7, 0, -5, 15, 2, 2, 9
```

- And we wanted to sort them

```
-5, 0, 1, 2, 2, 7, 9, 12, 15
```

- How would we store this now?
- What would the program look like?

Motivation: Grades

- What if we had 7 grades:
  - ➤`grade1, grade2, ..., grade7`
- How could we average them?

```
ave = (grade1 + grade2 + ... )/7
```

- What if we had, say, 200 grades?

Data Structures

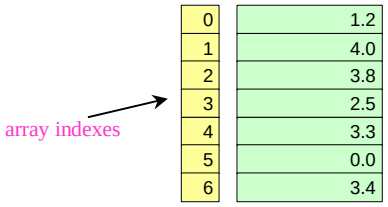
- Data structures are ways of organizing data
- Variables are the simplest kind of data structure:
  - ➤They store 1 value
  - ➤Pros? Cons?

Arrays

- Arrays let one group together many simpler variables
  - ➤One name
  - ➤Each piece of the array is accessed by a number: the index
- In an array, all of the values must have the same type

An Array

- Grades for 7 students



How Do We Average These?

- Use a loop to add them up:

  sum = 0
  for each index
  add the indexed value to sum

- When you're done, divide like usual

Grades in C

```
double grades[7];
int i;
double sum = 0, average;
…
for (i=0; i<7; i++){
  sum = sum + grades[i];
}
average = sum/7.0;
```


Array Declaration

```
int anArray[100];
```

- "Make an array of 100 integers named anArray."
- Can also use other types.

```
double floatArray[12];
char bunchOChars[1000];
```


Array Declarations

- When you declare an array, the size must be an integer constant!
- Consider this set of declarations:

```
int size = 5;
double foo[3.0];
char bar[size];
```

Accessing Array Elements

- If we have an array int foo[100], then we can access each element using the subscript:

```
foo[3] = 12;

printf("theValue: %d\n", foo[20]);

scanf("%d", &foo[66]);
```


When Can You Use Array Elts.?

- You can use an array element wherever you could use a normal variable of the same type
- If we have an array of doubles, for example, each element is of type double

Array Indexes

- Array indexes always begin at 0 in C
- There is no way to change this!
- What are valid indexes in this array?

char someCharacters[20];

Array Bounds

- These array indexes are not checked! This is called array bounds checking.
- What happens if you access element 10 in this array?

double foo[10];

Print Above Average Grades

```
double grades[500], total, average, temp;
int students=0, i;

printf("Enter grades, -1 to quit\n");
do{
  scanf("%lf", &temp);
  if (temp >= 0){
     grades[students] = temp;
     students++;
  }
} while (temp >= 0);
```


Print Above Average Grades

```
total = 0;
for (i=0; i<students; i++){
  total = total + grades[i];
}
average = total / students;

for (i=0; i<students; i++){
  if (grades[i] > average) {
     printf("%f\n", grades[i]);
  }
}
```

Things You Can't Do

- You can't use = to assign arrays
- You can't use == to compare two arrays
- You can't scanf or printf an entire array

- You can do these things to an array element

Parallel Arrays

- We can use parallel arrays to keep track of several pieces of information about the same thing

```
int pellets_x[NUM_PELLET];
int pellets_y[NUM_PELLET];
int pellets_type[NUM_PELLET];
```


Reflecting Points

```
void reflect(int *x, int *y){

  *x = *x * -1;
  *y = *y * -1;

  return;
}
```


Reflecting Points

```
int x_pos[20];
int y_pos[20];
int i;

for (i=0; i<20 ;i++)
  reflect(&x_pos[i], &y_pos[i]);
```


Arrays Unmasked

- What happens if we don't subscript an array?

- We can think of the array as a pointer to the beginning of a block of storage
  - ➤But you cannot assign to this pointer!

Passing Whole Arrays

- We can make a function that takes an entire array as a parameter:

```
int sum(int foo[], int size){
   int i, total = 0;
   for (i=0; i<size; i++)
        total = total + foo[i];
   return total;
}
```

Passing Arrays to Functions

- When we pass an array, we really only copy the pointer
- So, when we change values in the array they **DO** change in the calling function

Passing Arrays to Functions

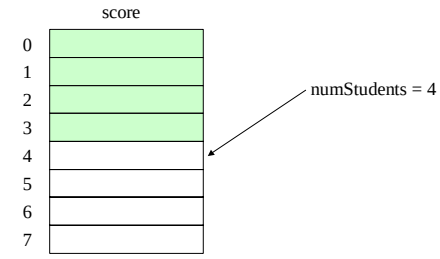
```
void sub1(int foo[],
  int size){

  int i;
  for(i=0;i<size;i++)
    foo[i] = foo[i]-1;

  return;
}
```

```
int main(void){
  int nums[10], i;

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

  return 0;
}
```

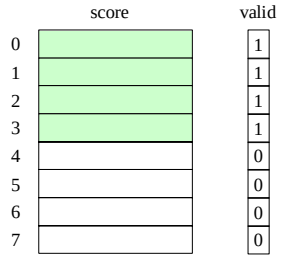

Keeping Track of Array Elts.

- We may not use all of the array—how do we keep track of which elements are being used?
  - ➤Keep them all at the beginning of the array, and store a count of how many there are
  - ➤Use a special flag to indicate “in use”

Keeping Valid Entries Together



In-Use Flag



Array Initialization

- When we create an array, none of the elements are initialized!
- We can do initialization in two ways
  - ➤Loop over the array and initialize it
  - ➤Initialize when we declare

Loop to Initialize

- Say we want each element initialized to its index value

```
int foo[10], i;
for (i=0; i<10; i++){
    foo[i] = i;
}
```


Initialization in Declaration

```
int foo[4] = {0, 1, 2, 3};

char vowels[5] = {'a', 'e', 'i',
  'o', 'u'};

double consts[] = {2.718, 3.14,
  1.414};
```


These are Illegal

- This isn't a declaration:

```
x = {1.0, 2.0, 3.0};
```

- This doesn't indicate the size:

```
double bar[];
```


Functions and Arrays

- Remember:
  - ➤Array elements behave just like normal variables, including as arguments
  - ➤Whole arrays behave like output parameters

General Vector Sum

```
void vectorSum(int a[], int b[], int
  vsum[], int length){

  int i;
  for (i=0; i<length; i++){
     vsum[i] = a[i] + b[i];
  }
  return;
}
```


Using the Vector Sum

```
int main(void){
  int v1 = {1, 3, 7};
  int v2 = {2, 4, 8};
  int vsum[3];

  vectorSum(v1, v2, vsum, 3);

  return 0;
}
```