Arrays and Pointers in C

CMSC 104 Section 02
April 29, 2024

Administrative Notes

Please see updated class schedule

- Note the changes in date for:
  - Classwork 9
  - Homework 9
  - Quiz 5

Arrays (and pointers) in C

Data structures

- All the data types we've seen so far can hold only one value at a time
  - One char, one int, one float
  - You can do a lot with that, but it's limiting

Why?

- Think of all the numAplus, numA, numAminus,...
- That gets ugly in a hurry

An example

Finding the median grade in a set of grades

- Median: the value where half the set is above that number, and half is below that number
  - There may be more than one median, etc.

To find the median, you need to have the entire set of numbers (grades) in memory so you can sort them out

- 34 students in this section; think about a section with 150 students
- How many different variables are you willing/able to declare?

A simple example:

```
File Edit Options Buffers Tools C Help
#include <stdio.h>

int main()  {

  int grade1, grade2, grade3, grade4, grade5, grade6, grade7, grade8, grade9;

  int median;

  // input the grades
  printf("please enter the grades separated by tabs \n");
  scanf("%d%d%d%d%d%d%d%d%d", &grade1, &grade2, &grade3, &grade4, &grade5, &grade6, &grade7, &grade8, &grade9);

  //calculate median of this would be awful!!
  //you need to sort the 9 grades into order
  //declare nine new variables, first, second, third,...
  //Put the largest value into first, the second largest into second
  // you can't really use a loop



  return 0;
}
```

Solution: multi-value data types

It would be simpler to handle a whole set of related variables together

C gives you a number of ways to do that

- Structs (custom designed types that I didn’t talk about)

We’re going to talk about arrays

Array - definition

An array is a group of related data items that all have the same data type, and share a common name.

- Arrays can be of any data type we choose.
- Arrays are static, in that they remain the same size throughout program execution.
- An array’s data items are stored contiguously in memory.
- Each of the data items is known as an element of the array.
- Each element can be accessed individually.

Example: an integer array

```
int numbers[5];
```

- The name of this example array is “numbers”.
- This declaration sets aside a chunk of memory that is big enough to hold 5 integers.
- It does not initialize those memory locations to 0 or any other value. They contain garbage.
- Initializing an array may be done with an array initializer, as in: int numbers[5] = {5, 2, 6, 9, 3};
- Notice that numbers “points” to the data

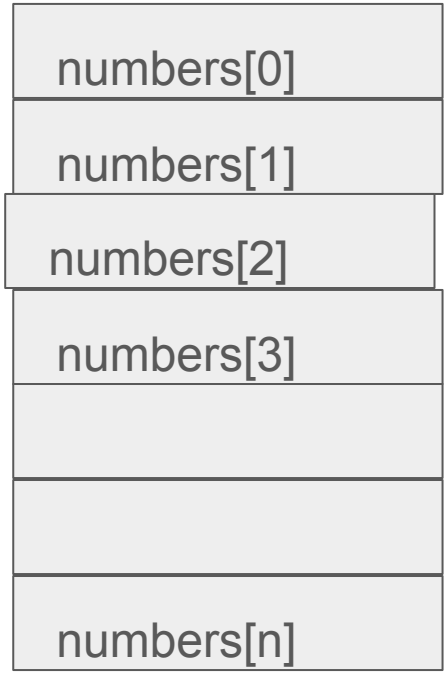
Example: a char array aka a “string”

```
char name[5];
```

- This example array is also known as a *string*.
- A string is an array of char elements, usually ending with a *null terminator*.
- A string can also be initialized like this:
  - `char name[5] = {‘J’, ‘o’, ‘h’, ‘n’,‘\0’};` or:
  - `char name[5] = "John";` Notice that name “points” to the data,
- As with all char variables, the data is really stored as integers.

Arrays in memory - a symbol table illustration

median	int	
numbers	Array of int	



Accessing the elements of an array

- Each element in an array has a subscript (index) associated with it.
- Subscript values are integers and always begin at zero.
- Values of individual elements can be accessed by indexing into the array.
- For example:
  - `printf("The third element is %d.\n", numbers[2]);`
- would give the output:
  - `The third element is 6.`
- A subscript can also be an expression which evaluates to an integer:

```
numbers[(a + b) * 2]
```

- ***Warning: If the index into the array is negative or too large, your program will crash.***

Modifying elements of an array

Individual elements of an array can also be modified using subscripts.

Example:

```
numbers[0] = 123;
```

Initial values may be stored in an array using indexing, rather than using an array initializer.

```
numbers[0] = 5;
numbers[1] = 2;
numbers[2] = 6;
numbers[3] = 9;
numbers[4] = 3;
```

Initializing your array

Some arrays can be quite large, and using an initializer can be impractical.

Large arrays are often intialized using a for loop.

```
for(i = 0; i < 1000; i++) { numbers[i] = 0; }
```

Shortcut: if setting all elements to the same value, this is valid:

```
int numbers[10] = {0};
```

More array declarations

```
int score[39], gradeCount[5];
```

- Declares two arrays of type int.
- Neither array has been initialized.
- “score” contains 39 elements, representing each student.
- “gradeCount” contains 5 elements, one for each possible grade letter.

You can use #define to clean that up

```
#define SIZE 39

#define GRADES 5

int main() {

  int score[SIZE];

  int gradeCount[GRADES];

/* do some stuff */

return 0; }
```

Now let’s look at how we would calculate that median

```
#include <stdio.h>

int main() {
  int grades [9];
  int median;
  int i, j, t;

  for (i = 0; i<9; i++) {
    printf("Enter the next grade: ");
    scanf("%d", &grades[i]);
  }

  // Now sort the array - a simple sort routine
  for (i = 0 ; i < 9 ; i++){
   for (j = 0 ; j < 9-i ; j++){
      if (grades[j] <= grades[j+1]){
         t = grades[j];
         grades[j] = grades[j+1];
         grades[j+1] = t;
      }
   }
}
  // now the median is just the middle value
  for (i = 0; i < 9; i++) {
    printf("%d    %d \n", i, grades[i]);
  }
  median = grades[4];
  printf ("The median grade is: %d \n", median);

  return 0;}
```

Wait, couldn’t we have written and called functions?

Yes:

- The data input loop should be a function
- The “sort the data” loop should be a function

But arrays cause weird things to happen when you pass them to functions as parameters

- And that’s Wednesday’s lecture, not todays!!