(1-1) C Review: Pointers, Arrays, Strings, & Structs

Instructor - Andrew S. O'Fallon CptS 122 (January 10, 2024) Washington State University



Crash Review on Critical C Topics

- Pointers
- Arrays
- Strings
- Structs



CS Discord

https://discord.com/invite/TkMb3deW

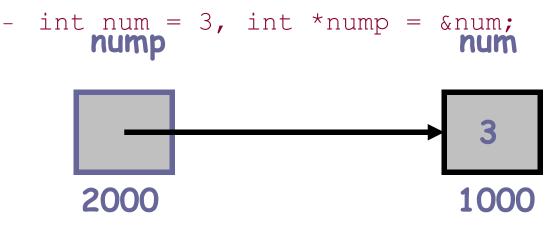


Pointers



Pointer Review (1)

- A pointer variable contains the address of another cell containing a data value
- Note that a pointer is "useless" unless we make sure that it points somewhere:



• The *direct* value of *num* is 3, while the *direct* value of *nump* is the address (1000) of the memory cell which holds the 3



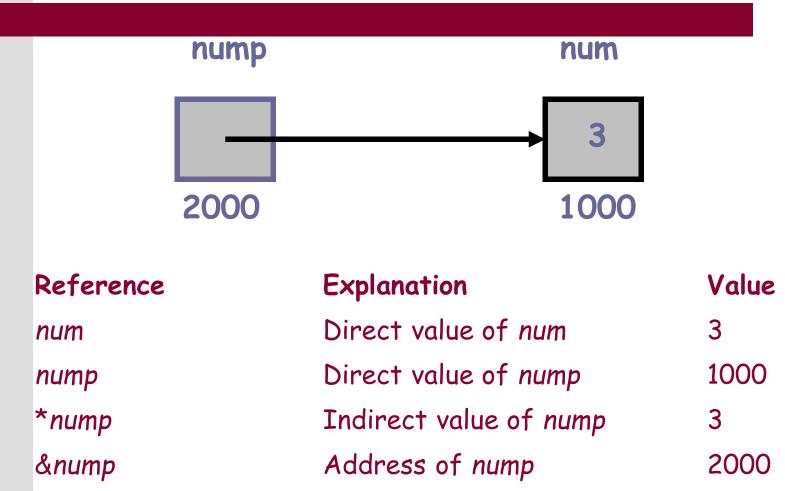
Pointer Review (2)

- The integer 3 is the *indirect* value of *nump*, this value can be accessed by following the pointer stored in *nump*
- If the indirection, dereferencing, or "pointerfollowing" operator is applied to a pointer variable, the indirect value of the pointer variable is accessed
- That is, if we apply **nump*, we are able to access the integer value 3
- The next slide summarizes...



C. Hundhausen, A. O'Fallon

Pointer Review (3)





Pointers as Function Parameters (1)

- Recall that we define an output parameter to a function by passing the address (&) of the variable to the function
- The output parameter is defined as a pointer in the formal parameter list
- Also, recall that output parameters allow us to return more than one value from a function
- The next slide shows a long division function which uses *quotientp* and *remainderp* as pointers



Pointers as Function Parameters (2)

• Function with Pointers as Output Parameters

```
#include <stdio.h>
void long_division (int dividend, int divisor, int *quotientp, int *remainderp);
int main (void)
{
    int quot, rem;
        long_division (40, 3, &quot, &rem);
        printf ("40 divided by 3 yields quotient %d ", quot);
        printf ("and remainder %d\n", rem);
        return 0;
}
void long_division (int dividend, int divisor, int *quotientp, int *remainderp)
{
        *quotientp = dividend / divisor;
        *remainderp = dividend % divisor;
}
```







C. Hundhausen, A. O'Fallon

What is an array?

- A sequence of items that are contiguously allocated in memory
- All items in the array are of the same data type and of the same size
- All items are accessed by the same name, but a different index
- The length or size is fixed



More About Arrays

- An array is a data structure
 - A data structure is a way of storing and organizing data in memory so that it may be accessed and manipulated efficiently



Uses for Arrays?

• Store related information

- Student ID numbers
- Names of players on the Seattle Seahawks roster
- Scores for each combination in Yahtzee
- Many more...

The Many Dimensions of an Array

- A single dimensional array is logically viewed as a linear structure
- A two dimensional array is logically viewed as a table consisting of rows and columns
- What about three, four, etc., dimensions?



Declaring a Single Dimensional Array (1)

• Arrays are declared in much the same way as variables:

int a[6];

declares an array a with 6 cells that hold integers:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
10	12	0	89	1	91

Notice that array indexing begins at 0.



Strings



String Fundamentals

- A string is a sequence of characters terminated by the null character ('\0')
 - "This is a string" is considered a string literal
 - A string may include letters, digits, and special characters
- A string may always be represented by a character array, but a character array is not always a string
- A string is accessed via a pointer to the first character in it

String Basics (1)

 As with other data types, we can even initialize a string when we declare it:

// These are equivalent string declarations!

• Here's what the memory allocated to name looks like after either of the above is executed:

null character (terminates all strings)



String Basics (2)

- When a variable of type char* is initialized with a string literal, it may be placed in memory where the string can't be modified
- If you want to ensure modifiability of a string store it into a character array when initializing it

String Basics (3)

• Arrays of Strings

- Suppose we want to store a list of students in a class
- We can do this by declaring an array of strings, one row for each student name:

```
#define NUM_STUDENTS 5
#define MAX_NAME LENGTH 31
char student names[NUM_STUDENTS][MAX_NAME_LENGTH];
```

- We can initialize an array of strings "in line":

```
char student_names[NUM_STUDENTS][MAX_NAME_LENGTH] =
{"John Doe", "Jane Smith", "Sandra Connor", "Damien White",
"Metilda Cougar"};
```

 In most cases, however, we're probably going to want to read the names in from the keyboard or a file...



String Basics (4)

- Use gets() to read a complete line, including whitespace, from the keyboard until the <enter> key is pressed; the <enter> is not included as part of the string
 - Usage: gets (my_array)
 - If the user enters "Bill Gates" and presses <enter>, the entire string will be read into my_array excluding the <enter> or newline
- Use puts() to display a string followed by a newline
 - **Usage:** puts(my_array)



String Manipulation in C (1)

- Standard operators applied to most numerical (including character) types cannot be applied to strings in C
 - The assignment operator (=) can't be applied except during declaration
 - The + operator doesn't have any true meaning (in some languages it means append)
 - The relational operators (==, <, >) don't perform string comparisons
 - Others?



String Manipulation in C (2)

- The string-handling library <string.h> provides many powerful functions which may be used in place of standard operators
 - strcpy () or strncpy () replaces the assignment operator
 - strcat () or strncat () replaces the + or append operator
 - strcmp () replaces relational operators
 - Several others...i.e. strtok (), strlen ()



Pointers Representing Arrays and Strings (1)

• Consider representing two arrays as follows:

- double list_of_nums[20];
- char your_name[40];
- When we pass either of these arrays to functions, we use the array name without a subscript
- The array name itself represents the address of the initial array element



Pointers Representing Arrays and Strings (2)

- Hence, when we pass the array name, we are actually passing the entire array as a pointer
- So, the formal parameter for the string *name* may be declared in two ways:
 - char name[]
 - char *name
- Note that, in general, it is a good idea to pass the maximum size of the array to the function, e.g.:
 - void func (char *name, int size);



Structs



struct Type (1)

- C supports another kind of user-defined type: the struct
- structs are a way to combine multiple variables into a single "package" (this is called "encapsulation")
- Sometimes referred to as an *aggregate*, where all variables are under one name
- Suppose, for example, that we want to create a database of students in a course. We could define a student struct as follows:



struct Type (2)

```
typedef enum {freshman, sophomore, junior, senior}
           class t; /* class standing */
  typedef enum {anthropology, biology, chemistry,
                  english, compsci, polisci,
  psychology,
           physics, engineering, sociology}
major t; /* representative majors */
typedef struct
       int id number;
       class t class standing; /* see above */
       major t major; /* see above */
       double gpa;
      int credits taken;
  } student t;
```



struct Type (3)

• We can then define some students:

```
student_t student1, student2;
student1.id_num = 123456789;
student1.class_standing = freshman;
student1.major = anthropology;
student1.gpa = 3.5;
student1.credits_taken = 15;
student2.id_num = 321123456;
student2.class_standing = senior;
student2.major = biology;
student2.gpa = 3.2;
student2.credits_taken = 100;
```

Notice how we use the "." (selection) operator to access the "fields" of the struct



More About Structs

- Recall structs are used to represent real world objects
- They contain attributes that describe these objects
 - Such as a car, where the attributes of the struct car could include steering wheel, seats, engine, etc.
 - Such as a student, where the attributes of the struct student could include ID#, name, standing, etc.
- In many cases, we need a list or array of these objects
 - A list of cars representing a car lot
 - A list of students representing an attendance sheet



Arrays of Structs (1)

- Let's first define a struct student
 typedef struct student
 - int ID; char name[100]; int present; // Attended class or not
 - } Student;
- Next we will build up an attendance sheet



Arrays of Structs (2)

```
int main (void)
```

```
{
```

Student attendance_sheet[100]; // 100 students in the class

```
return 0;
```

```
}
```

 Let's look at a logical view of this attendance sheet on the next slide



Arrays of Structs (3)

• Attendance sheet, which consists of multiple struct student types

0	1	2	 99
{ID,	{ID,	{ID,	 {ID,
name,	name,	name,	name,
present}	present}	present}	present}
1000	1108	1216	10692



Arrays of Structs (4)

 To initialize one item in the array, try: attendance_sheet[index].ID = 1111; strcpy (attendance_sheet[index].name, "Bill Gates"); Attendance_sheet[index].present = 1;

// 1 means in attendance, 0 means not in present

Pointers to Structures

• Recall that when we have a pointer to a structure, we can use the indirect component selection operator -> to access components within the structure

```
typedef struct {
    double x;
    double y;
} Point;
int main (void)
{
    Point p1, *struct_ptr;
    p1.x = 12.3;
    p1.y = 2.5;
    struct_ptr = &p1;
    struct_ptr->x; /* Access the x component in Point, i.e. 12.3 */
    .
    .
}
```

Keep Reviewing C Material!



References

- J.R. Hanly & E.B. Koffman, Problem Solving and Program Design in C (8th Ed.), Addison-Wesley, 2016.
- P.J. Deitel & H.M. Deitel, *C How to Program* (7th Ed.), Pearson Education , Inc., 2013.



Collaborators

• Chris Hundhausen

