Pointers in C

By

Prof. Muhammad Iqbal Bhat

Department of Higher Education

Government Degree College Beerwah

Topics:

1

What are pointers

2

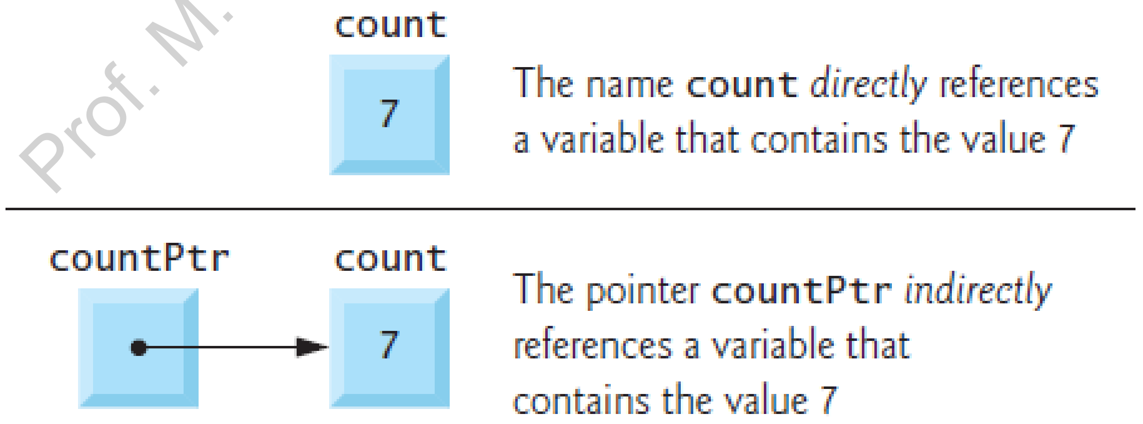
Declaration and initialization of Pointers

3

Accessing Value of Pointers

What are Pointers?

- In C, a pointer is a variable that stores the **memory address** of another variable.
- Pointers allow you to indirectly access and manipulate data in memory.
- A variable name directly references a value, and a pointer indirectly references a value, as in the following diagram:


count
7
The name count *directly* references a variable that contains the value 7
countPtr
count
7
The pointer countPtr *indirectly* references a variable that contains the value 7

Referencing a value through a pointer is called indirection.

Declaration and Initialization of Pointers?

Declare a pointer using the * *operator and a data type* ***(e.g. int*****, char*, float*)**

Initialize a pointer by assigning it the memory address of a variable using the "&" operator
(e.g. int* p = &x;)

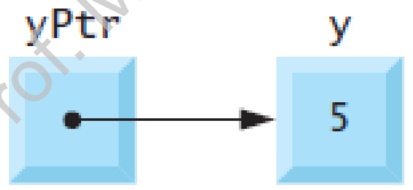
Declaring a pointer variable involves specifying the data type that the pointer will point to. The "*" operator is used to indicate that a variable is a pointer.

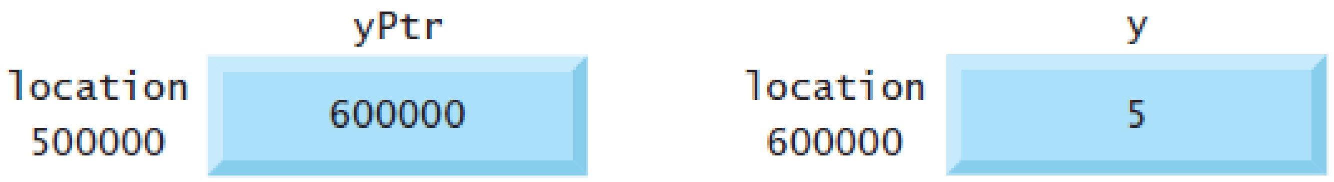
Pointer Operators and their relationship:

The unary **address operator** (**&**) returns the *address* of its operand.

For example, given the following definition of y: **int** y = **5**;

the statement `int *yPtr = &y;` initializes pointer variable yPtr with variable y's *address*—yPtr is then said to "point to" y.





Pointer Operators and their relationship:

You apply the unary indirection operator (*), also called the dereferencing operator, to a pointer operand to get the value of the object to which the pointer points

```
printf("%d", *yPtr);
```

Using * in this manner is called **dereferencing a pointer.**

Dereferencing a pointer that has not been initialized with or assigned the address of another variable in memory is called a dangling pointer and is an error.

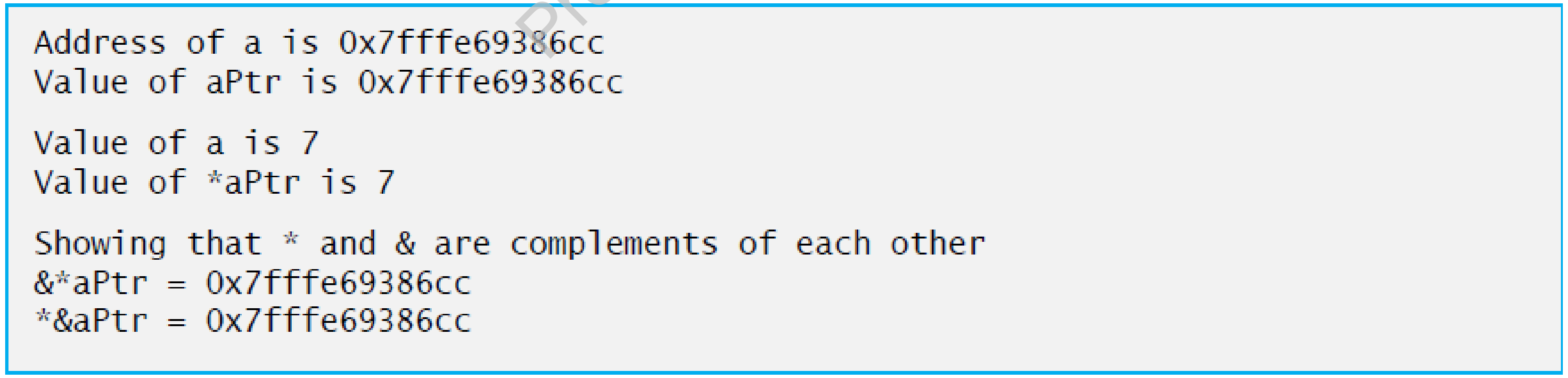
Dangling Pointers Can cause:

***cause a fatal execution-time error,**

- **accidentally modify important data and allow the program to run to completion with incorrect results, or**
- **lead to a security breach.**

Example Demonstrating the & and * Operators

```c
#include <stdio.h>
int main(void) {
	int a = 7;
	int *aPtr = &a; // set aPtr to the address of a
	printf("Address of a is %p\nValue of aPtr is %p\n\n", &a, aPtr);
	printf("Value of a is %d\nValue of *aPtr is %d\n\n", a, *aPtr);
	printf("Showing that * and & are complements of each other\n");
	printf("&*aPtr = %p\n*&aPtr = %p\n", &*aPtr, *&aPtr);
}
```



```
Address of a is 0x7fffe69386cc
Value of aPtr is 0x7fffe69386cc

Value of a is 7
Value of *aPtr is 7

Showing that * and & are complements of each other
&*aPtr = 0x7fffe69386cc
*&aPtr = 0x7fffe69386cc
```

Pointer Increments and Scale Factor

Pointer increments use the ++ and -- operators to move a pointer to the next or previous memory location.

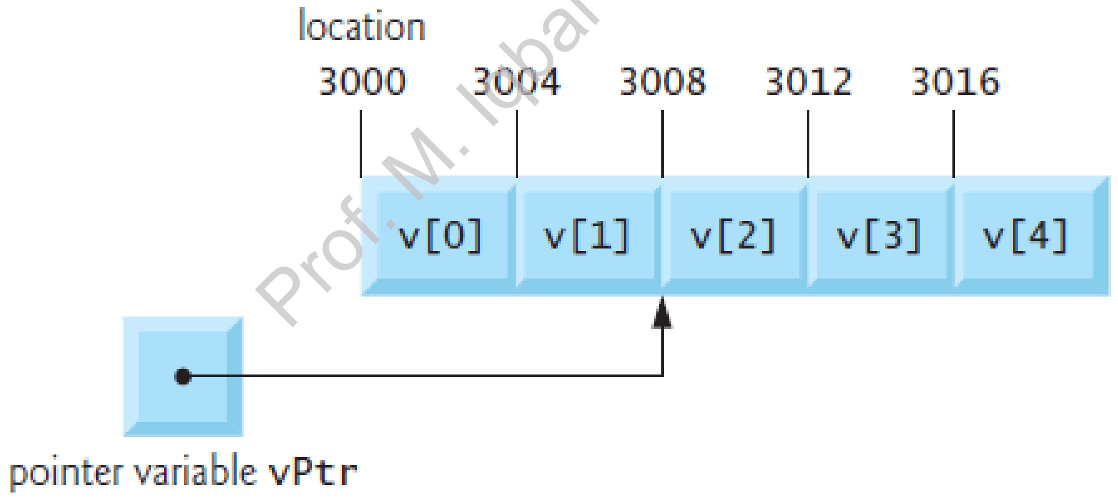
Incrementing or decrementing a pointer moves it to the next or previous memory location of the same data type.

Scale factors are used to account for the size of the data type being pointed to when incrementing or decrementing a pointer.

The scale factor is determined by the size of the data type being pointed to

Pointer Increments and Scale Factor

- When you add an integer to or subtract one from a pointer, the pointer increments or decrements by that integer times the size of the object to which the pointer refers.
- **`vPtr += 2;`** would produce **3008 (3000 + 2 * 4),** assuming int is stored in 4 bytes



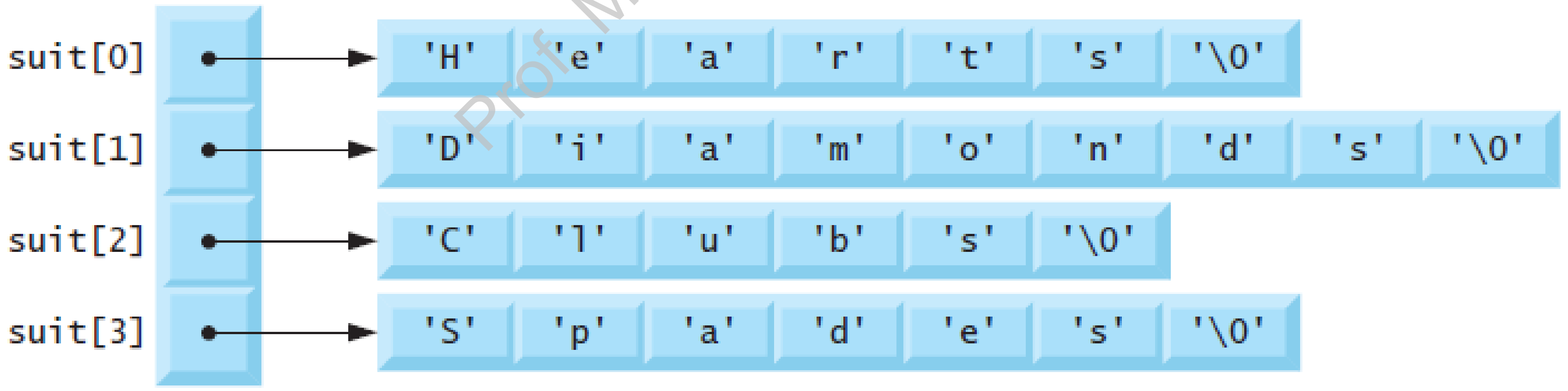
Pointer Increments and Scale Factor

- If vPtr contains the location 3000 and v2Ptr contains the address 3008, the statement
- `x = v2Ptr - vPtr;`
- assigns to x the number of array elements between vPtr and v2Ptr, in this case, 2 (not 8).
- Pointer arithmetic is undefined unless performed on elements of the same array.

Array of Pointers:

- Arrays may contain pointers.
- A common use of an array of pointers is to form an array of strings, referred to simply as a string array. Each element in a C string is essentially a pointer to its first character.
- So, each entry in an array of strings is actually a pointer to a string's first character.

```c
const char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};
```


suit[0]
'H' 'e' 'a' 'r' 't' 's' '\0'
suit[1]
'D' 'i' 'a' 'm' 'o' 'n' 'd' 's' '\0'
suit[2]
'C' 'l' 'u' 'b' 's' '\0'
suit[3]
'S' 'p' 'a' 'd' 'e' 's' '\0'

Calculate the sum of an array using pointers

```
#include <stdio.h>

int sum(int *arr, int size) {
    int total = 0;
    for (int i = 0; i < size; i++) {
        total += *(arr + i);
    }
    return total;
}

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(int);
    int total = sum(arr, size);
    printf("The sum of the array is %d\n", total);
    return 0;
}
```

Allocate memory dynamically using pointers

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr;
    int n = 5;
    ptr = (int*) malloc(n * sizeof(int));
    if (ptr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        *(ptr + i) = i + 1;
    }
    for (int i = 0; i < n; i++) {
        printf("%d ", *(ptr + i));
    }
    printf("\n");
    free(ptr);
    return 0;
}
```

Questions?