EEE: 103

Computer Programming

L4: Pointers in C

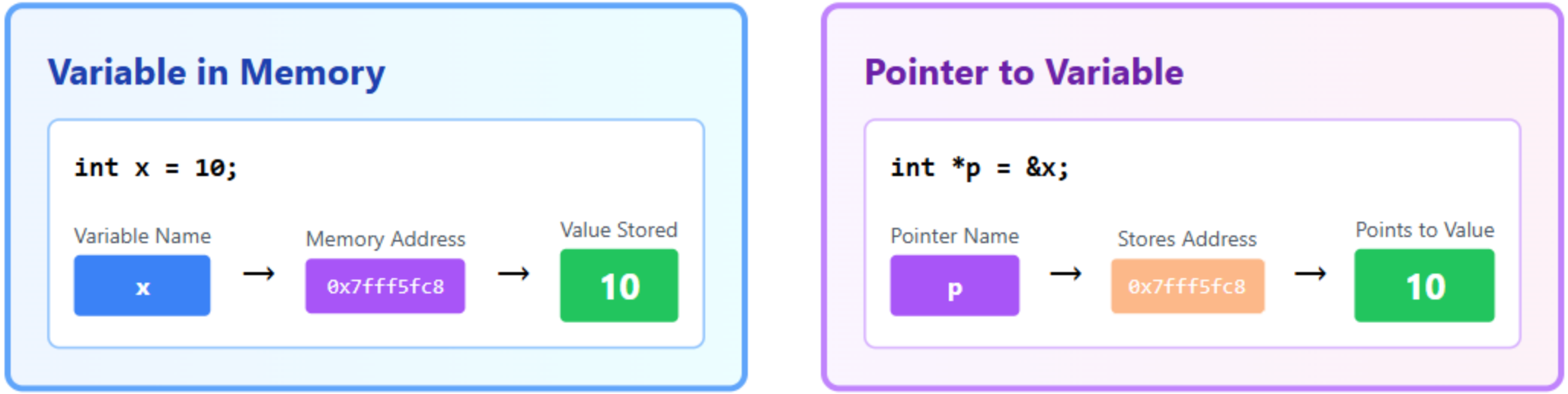
Prepared by: Sk Tahmed Salim Rafid

What is a Pointer?

- A pointer is a variable that stores the memory address of another variable
- Instead of storing actual values, pointers store locations in memory
- Pointers allow direct memory access and manipulation
- Essential for dynamic memory management
- Used extensively in arrays, functions, and data structures

Understanding Memory & Pointers

Visual representation of how pointers work in memory



💡 **Key Insight:** The pointer 'p' doesn't hold the value 10 directly. It holds the address where 10 is stored!

Declaring a Pointer

*To declare a pointer, specify the data type followed by * and the pointer name*

```
int *p;
```

p is a pointer to an integer

```
float *fptr;
```

fptr is a pointer to a float

```
char *cptr;
```

cptr is a pointer to a char

📌 The type of pointer should match the type of variable it points to

📌 The * indicates that the variable is a pointer

Assigning a Pointer

A pointer should store a memory address, not a direct value

✓ Correct Way

```
int x = 10;
int *p;
p = &x;  // Correct: store the address of x
in p
```

& is the address-of operator. It returns the memory address of a variable.

✗ Wrong Way

```
p = 10; // Incorrect: assigning a value
directly to a pointer
```

Never assign a value directly to a pointer!

Accessing Value through Pointer

*To access the value stored at the memory address, use the * operator (dereferencing)*

```
1  int x = 10;
2  int *p = &x;
3
4  printf("%p", p); // Prints address (e.g.,
0x7fff5fbff8ac)
5  printf("%d", *p); // Prints value: 10
```

Step 1 Step 2 Step 3

Key Points:

- Printing the pointer itself shows the memory address
- Printing *p gives the value stored at that address
- The * operator is used for dereferencing

Common Mistakes to Avoid

⚠ Assigning a value to a pointer instead of an address

❌ **Wrong:**

```
int *p = 10;
```

✓ **Right:**

```
int x = 10; int *p = &x;
```

⚠ Dereferencing an uninitialized pointer

❌ **Wrong:**

```
int *p;
printf("%d", *p); // Segmentation fault!
```

✓ **Right:**

```
int x = 10;
int *p = &x;
printf("%d", *p);
```

⚠ Confusing pointer and *pointer while printing

Taking Input using Pointers

You can take input directly into the variable pointed to by a pointer

```c
#include <stdio.h>

int main() {
 int num;
 int *p = &num;

 printf("Enter a number: ");
 scanf("%d", p); // equivalent to scanf("%d", &num)

 printf("You entered: %d\n", *p);
 return 0;
}
```

Step 1 Step 2 Step 3 Step 4

💡 Important: Notice: We use p (not &p) because p already contains the address!

Example 1: Subtract Two Numbers Using Pointers

Subtract the smaller number from the larger one

```c
#include <stdio.h>

int main() {
 int a, b;
 int *p1 = &a, *p2 = &b;

 printf("Enter two numbers: ");
 scanf("%d %d", p1, p2);

 int diff;
 if (*p1 > *p2)
 diff = *p1 - *p2;
 else
 diff = *p2 - *p1;

 printf("Difference (larger - smaller) = %d\n", diff);
 return 0;
}
```

Step 1 Step 2 Step 3 Step 4 Step 5

Output:

```
Enter two numbers: 25 15
Difference (larger - smaller) = 10
```

Arrays and Pointers

Understanding the relationship between arrays and pointers

```
int data[5] = {10, 20, 30, 40, 50};
```

data[0]	data[1]	data[2]	data[3]	data[4]
10	20	30	40	50
0x1000	0x1001	0x1002	0x1003	0x1004

`int *ptr = data;` → ptr points to data[0]

`*(ptr + 0) = 10`

Access first element

`*(ptr + 2) = 30`

Access third element

💡 Remember: Array name 'data' is equivalent to &data[0] - it's a pointer to the first element!

Example 2: Printing Array Using Pointer

Traverse and print array elements using pointer arithmetic

```c
#include <stdio.h>

int main() {
 // 1. Declare and initialize an integer array
 int data[5] = {10, 20, 30, 40, 50};

 // 2. Declare a pointer to the base address
 int *ptr = data; // data is equivalent to &data[0]

 printf("Printing array elements using pointer:\n");

 // 3. Loop through array using pointer arithmetic
 for (int i = 0; i < 5; ++i) {
 printf("Element %d: %d\n", i, *(ptr + i));
 }

 return 0;
}
```

Step 1 | Step 2 | Step 3 | Step 4 | Step 5 | Step 6

Output:

```
Element 0: 10
Element 1: 20
Element 2: 30
Element 3: 40
Element 4: 50
```

Key Points:

- data gives the base address of the array
- *(ptr + i) dereferences the pointer at element i
- Pointer arithmetic: ptr + i moves i positions forward

Pointer Arithmetic

`ptr + n` Move n elements forward

Example:

```
int arr[5] = {10,20,30,40,50};
int *p = arr;
*(p + 2) gives 30
```

`ptr - n` Move n elements backward

Example:

```
int *p = &arr[4];
*(p - 2) gives 30
```

`ptr++` Move to next element

Example:

```
int *p = arr;
p++; // now points to arr[1]
```

`ptr--` Move to previous element

Example:

```
int *p = &arr[2];
p--; // now points to arr[1]
```

Summary: Key Takeaways

- ✓ A pointer stores the address of a variable
- ✓ Use & to get the address of a variable
- ✓ Use * to dereference a pointer and access/change the value
- ✓ Pointer arithmetic allows efficient array traversal
- ✓ Always initialize pointers before dereferencing to avoid errors
- ✓ Array names are pointers to their first element
- ✓ Pointers enable dynamic memory management and data structures

Practice Problems

1. Write a program to swap two numbers using pointers
2. Create a program that finds the largest element in an array using pointers
3. Write a program to reverse an array using pointer arithmetic
4. Implement a function that takes a pointer and modifies the value it points to
5. Create a program that uses pointers to calculate the sum of array elements

Thank You

Master Pointers, Master C! 🚀