Pointers Basic

📘 Understanding Pointers in C

1. What Is a Pointer?

A **pointer** is a variable that stores the **memory address** of another variable.

```
int x = 25;
int *p = &x;
```

Here:

- `x` is an integer variable that stores the value `25`.
- `p` is a pointer variable that stores the **address of** `x`.
- The type of pointer (`int *`) tells the compiler that `p` points to an integer value.

2. Every Pointer Has Its Own Address

Pointers are also stored in memory, so they too have their own addresses.

Variable	Type	Value (Content)	Address
`x`	int	25	1000
`p`	int*	1000 (address of `x`)	2000

So:

- `p` stores the address of `x`.
- `p` itself is stored at another address (for example, 2000).

3. Example Program

```c
#include <stdio.h>

int main() {
    int x = 25;
    int *p = &x;

    printf("Value of x: %d\n", x);
    printf("Address of x: %p\n", (void*)&x);
    printf("Value of pointer p: %p\n", (void*)p);
    printf("Address of pointer p: %p\n", (void*)&p);
    printf("Value pointed by p: %d\n", *p);

    return 0;
}
```

📊 Sample Output

```
Value of x: 25
Address of x: 0x3e8
Value of pointer p: 0x3e8
Address of pointer p: 0x7d0
Value pointed by p: 25
```

---

4. Why Use `(void *)` in `printf()`?

⚠️ Without Casting:

```c
printf("%p\n", &x);
```

Many compilers (like GCC or Clang) will show a **warning**:

```
warning: format '%p' expects argument of type 'void *', but argument 2 has ty
pe 'int *'
```

✅ With Casting:

```
printf("%p\n", (void*)&x);
```

This removes the warning because the `%p` format specifier **expects a** `void *` **argument** (a generic pointer type).

> (void *) tells the compiler to treat the address as a general memory address, not a typed pointer.

---

5. Dereferencing a Pointer

Dereferencing means accessing the value stored at the address that the pointer holds.

```
int x = 25;
int *p = &x;
printf("%d", *p);   // prints 25
```

Explanation:

- `p` contains the address of `x`.
- `p` means "go to the address stored in `p` and fetch the value."
- So `p` gives the value of `x`.

📘 Example with Diagram

```
  x = 25        p = &x
┌──────┐      ┌──────────┐
| 25   | ←——— | 1000     |
└──────┘      └──────────┘
1000 (x)      2000 (p)
```

- `x` is stored at memory address `1000`.
- `p` stores `1000` (address of `x`).

- `p` means → "value at address 1000" → which is `25`.

6. Types of Dynamic Memory Allocation

In C, **dynamic memory allocation** allows allocating memory at runtime using the following functions from `<stdlib.h>`:

Function	Description	Syntax
`malloc()`	Allocates a block of memory	`ptr = (type*)malloc(size);`
`calloc()`	Allocates and initializes memory to zero	`ptr = (type*)calloc(n, size);`
`realloc()`	Changes size of previously allocated memory	`ptr = (type*)realloc(ptr, new_size);`
`free()`	Frees dynamically allocated memory	`free(ptr);`

7. Summary Table

Expression	Meaning
`x`	Value of variable
`&x`	Address of variable
`p`	Value stored in pointer (address of `x`)
`&p`	Address of pointer itself
`*p`	Value pointed by pointer
`(void*)&x`	Type-casting address for safe printing with `%p`

8. 📘 MCQs on Pointers in C

1️⃣

What does a pointer store?

A) Value of a variable

B) Address of a variable

C) Data type of variable

D) None

Answer: B

2️⃣

Which operator is used to access the address of a variable?

A) `*`

B) `&`

C) `->`

D) `%`

Answer: B

3️⃣

Which operator is used to access the value pointed by a pointer?

A) `&`

B) `*`

C) `->`

D) `.`

Answer: B

4️⃣

If `int *p;` and `p` stores address `1000`, what will `p+1` store (assuming 4-byte int)?

A) 1001

B) 1002

C) 1004

D) 1008

Answer: C

5️⃣

What will this code print?

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

Answer: 10

6️⃣

Which of the following is a generic pointer in C?

A) `int *`

B) `char *`

C) `float *`

D) `void *`

Answer: D

7️⃣

What does the following line mean?

```
int **ptr;
```

A) Pointer to integer

B) Pointer to a pointer to integer

C) Pointer to array

D) None

Answer: B

8️⃣

If `p` is a pointer, what does `&p` represent?

A) Value stored in `p`

B) Address of the pointer variable

C) Value pointed by `p`

D) None

Answer: B

9️⃣

Which function is used to free dynamically allocated memory?

A) `remove()`

B) `delete()`

C) `free()`

D) `clear()`

Answer: C

🔟

Which function allocates memory and initializes it to zero?

A) `malloc()`

B) `calloc()`

C) `realloc()`

D) `new()`

Answer: B

1️⃣1️⃣

What is the size of `int *p;`?

A) Depends on `int`

B) Depends on compiler (typically 4 or 8 bytes)

C) Always 2 bytes

D) None

Answer: B

1️⃣2️⃣

What is printed in the below code?

```
int x = 5;
int *p = &x;
int **q = &p;
printf("%d", **q);
```

Answer: 5

Q1. What will be the output?

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

Options:

A) Address of x

B) 10

C) Error

D) Garbage value

Answer: B

Explanation: `*p` dereferences the pointer and gives the value stored at the address, which is 10.

Q2. What will be the output?

```
int x = 5;
int* p = &x;
*p = 20;
printf("%d", x);
```

Options:

A) 5

B) 20

C) Error

D) Address of x

Answer: B

Explanation: `*p = 20` modifies the value at the address pointed by `p`, which is `x`. So `x` becomes 20.

Q3. What will be the output?

```
int arr[] = {10, 20, 30};
int* p = arr;
printf("%d", *(p+1));
```

Options:

A) 10

B) 20

C) 30

D) Error

Answer: B

Explanation: `p` points to `arr[0]`. `p+1` points to `arr[1]`, and `*(p+1)` gives the value 20.

Q4. What will be the output?

```
int a = 100;
int* p = &a;
int** q = &p;
printf("%d", **q);
```

Options:

A) Address of a

B) Address of p

C) 100

D) Error

Answer: C

Explanation: `**q` dereferences twice: first to get `p`, then to get the value of `a`, which is 100.

Q5. What will be the output?

```
int x = 50;
int* p = &x;
printf("%p\n", (void*)p);
printf("%d", *p);
```

Options:

A) Address of x and 50

B) Address of p and 50

C) 50 and 50

D) Error

Answer: A

Explanation: First `printf` prints the address stored in `p` (address of x), second prints the value at that address (50).

Q6. What will be the output?

```
int x = 10, y = 20;
int* p = &x;
p = &y;
printf("%d", *p);
```

Options:

A) 10

B) 20

C) Address of x

D) Address of y

Answer: B

Explanation: Initially `p` points to `x`, then it's reassigned to point to `y`. So `*p` gives 20.

Q7. What will be the output?

```
int x = 5;
int* p = &x;
int* q = p;
printf("%d", *q);
```

Options:

A) 5

B) Address of x

C) Error

D) Garbage value

Answer: A

Explanation: `q` is assigned the same address as `p`, so `*q` also accesses the value of `x`, which is 5.

Q8. What is the size of a pointer variable?

```
int* p;
printf("%zu", sizeof(p));
```

Options:

A) 2 bytes

B) 4 bytes

C) 8 bytes

D) Depends on system architecture

Answer: D

Explanation: Pointer size depends on the system: 4 bytes on 32-bit systems, 8 bytes on 64-bit systems.

Q9. Which of the following is correct?

Options:

A) `int* p, q;` declares two pointers

B) `int *p, *q;` declares two pointers

C) Both A and B

D) None

Answer: B

Explanation: In option A, only `p` is a pointer, `q` is an integer. In B, both are pointers.

Q10. What will be the output?

```
int arr[3] = {1, 2, 3};
printf("%d", *arr + 1);
```

Options:

A) 1

B) 2

C) 3

D) Address

Answer: B

Explanation: `*arr` gives 1, then `1 + 1 = 2`. Note: This is different from `*(arr + 1)` which would give 2.

---

12. Summary

- **Pointer** stores the memory address of another variable.
- `&` **operator** → gives the address of a variable.
- **operator** → dereferences a pointer (accesses the value at the address).
- **Pointers and arrays** are closely related; array name is a constant pointer.
- **Pointer arithmetic** allows navigation through memory (increment, decrement, addition, subtraction).
- **NULL pointers** are used to indicate that a pointer doesn't point to any valid memory.
- **Dynamic memory allocation** (`malloc`, `calloc`, `realloc`, `free`) requires pointers.
- Pointers are used for:
  - Efficient function parameter passing
  - Dynamic memory management
  - Complex data structures (linked lists, trees, graphs, stacks, queues)
  - Array manipulation

    - Returning multiple values from functions
- **Common mistakes** to avoid:
    - Uninitialized pointers
    - Dangling pointers
    - Memory leaks
    - Wild pointers
- Always initialize pointers and free dynamically allocated memory to write safe and efficient C programs.

✅ Final Summary

- Pointers store **addresses**, not values.
- Every pointer has its **own address** in memory.
- `%p` prints addresses, and it expects `(void *)`.
- **Dereferencing (`p`)** means accessing the value at the address stored by a pointer.
- **Dynamic memory allocation** helps create memory at runtime using `malloc()`, `calloc()`, `realloc()`, and `free()`.

Would you like me to now generate this as a **formatted PDF document** (for class distribution or notes)?