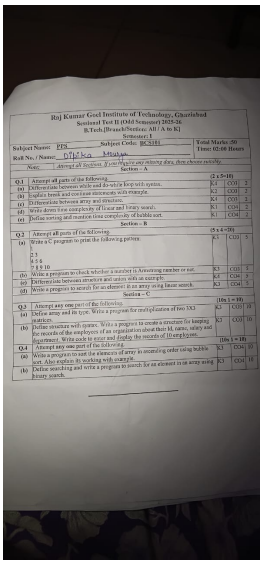
Sessiom A 13

Monday, 17 November 2025 9:13 AM



Raj Kumar Goel Institute of Technology, Ghaziabad
Sessional Test II (Odd Semester) 2025-26
B.Tech (Branch/Section: All / A to K)
Semester: I

Subject Name: PPS Subject Code: BCS201 Total Marks: 50
Roll No. / Name: DIVYA Time: 02:00 Hours

Note: Attempt all Sections. If you require any missing data, then choose suitably.

Section – A

Q.1	Attempt all parts of the following.			(2 x 5=10)
(a)	Differentiate between while and do-while loop with syntax.	K1	CO1	2
(b)	Explain break and continue statements with example.	K2	CO1	2
(c)	Differentiate between array and structures.	K1	CO1	2
(d)	Write down time complexity of linear and binary search.	K1	CO1	2
(e)	Define sorting and mention time complexity of bubble sort.	K1	CO1	2

Section – B

Q.2	Attempt all parts of the following.			(5 x 4=20)
(a)	Write a C program to print the following pattern. 1 2 3 4 5 6 7 8 9 10	K3	CO1	5
(b)	Write a program in C to check whether a number is Armstrong number or not.	K3	CO1	5
(c)	Differentiate between structure and unions with an example.	K2	CO1	5
(d)	Write a program to search for an element in an array using linear search.	K3	CO1	5

Section – C

Q.3	Attempt any one part of the following.			(10x 1 = 10)
(a)	Define array and its type. Write a program for multiplication of two 3X3 matrices.	K3	CO3	10
(b)	Define structure with syntax. Write a program to create a structure for keeping the records of the employees of an organization about their id, name, salary and department. Write code to enter and display the records of 10 employees.	K3	CO3	10

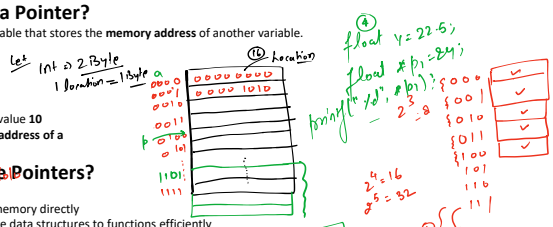
Q.4	Attempt any one part of the following.			(10x 1 = 10)
(a)	Write a program to sort the elements of array in ascending order using bubble sort. Also explain its working with example.	K3	CO4	10
(b)	Define searching and write a program to search for an element in an array using binary search.	K3	CO4	10

Pointers in C – Simple Notes

1. What is a Pointer?

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

Example:



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

- a → stores value **10**
- p → stores **address of a**

2. Why Use Pointers?

Pointers help in:

- Accessing memory directly
- Passing large data structures to functions efficiently
- Dynamic memory allocation
- Creating data structures (linked list, trees, graphs)

3. Pointer Declaration

```
int *p;
float *q;
char *c;
```

4. Address-of Operator (&)

Gives the address of a variable.

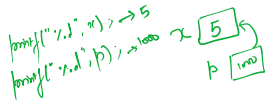
```
int x = 5;
printf("%p", &x);
```

5. Dereferencing Operator (*)

Gives the value stored in the address the pointer is pointing to.

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

6. Pointer Example



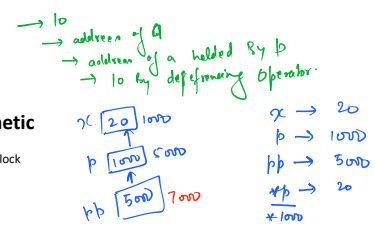
```
#include <stdio.h>
int main() {
   int a = 10;
   int *p = &a;
printf("Value of a = %d\n",a);
   printf("Address of a = %p\n", &a);
   printf("Pointer p stores = %p\n", p);
   printf("Value at address p = %d\n", *p);
return 0;
}
```

7. Pointer to Pointer

A pointer can store the address of another pointer.

```
int x = 20;
int *p = &x;
int **pp = &p;
```

- pp stores address of p
- *pp gives address of x
- **pp gives value of x



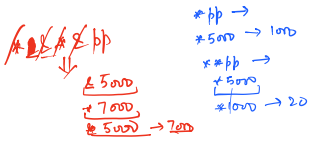
8. Pointer Arithmetic

Valid operations:

- ~~p++~~ → next memory block
- p--
- p + n
- p - n

Example:

```
int arr[5] = {1,2,3,4,5};
int *p = arr;
p++;  // moves to next integer location
```



9. Pointers and Arrays

Array name is itself a pointer.

```
int arr[3] = {10,20,30};
int *p = arr;
printf("%d", *(p+1)); // prints 20
```

10. Pointers and Functions

Passing variable by reference using pointers.

```
void change(int *x) {
   *x = 100;
}
int main() {
   int a = 10;
   change(&a);
   printf("%d", a);  // prints 100
}
```

11. Dangling Pointer

Pointer that points to unavailable/freed memory.

```
int *p = malloc(sizeof(int));
free(p);
printf("%d", *p);  // dangerous
```

12. NULL Pointer

Pointer that points to nothing.

```
int *p = NULL;
Useful for checking:

if (p != NULL) {
   // safe to use p
}
```

13. Void Pointer

A pointer that can store address of any data type.

```
void *ptr;
int a = 10;
ptr = &a;
```

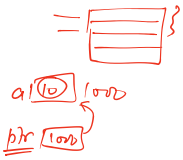
2️ Constant Pointer (pointer is constant, data is not)

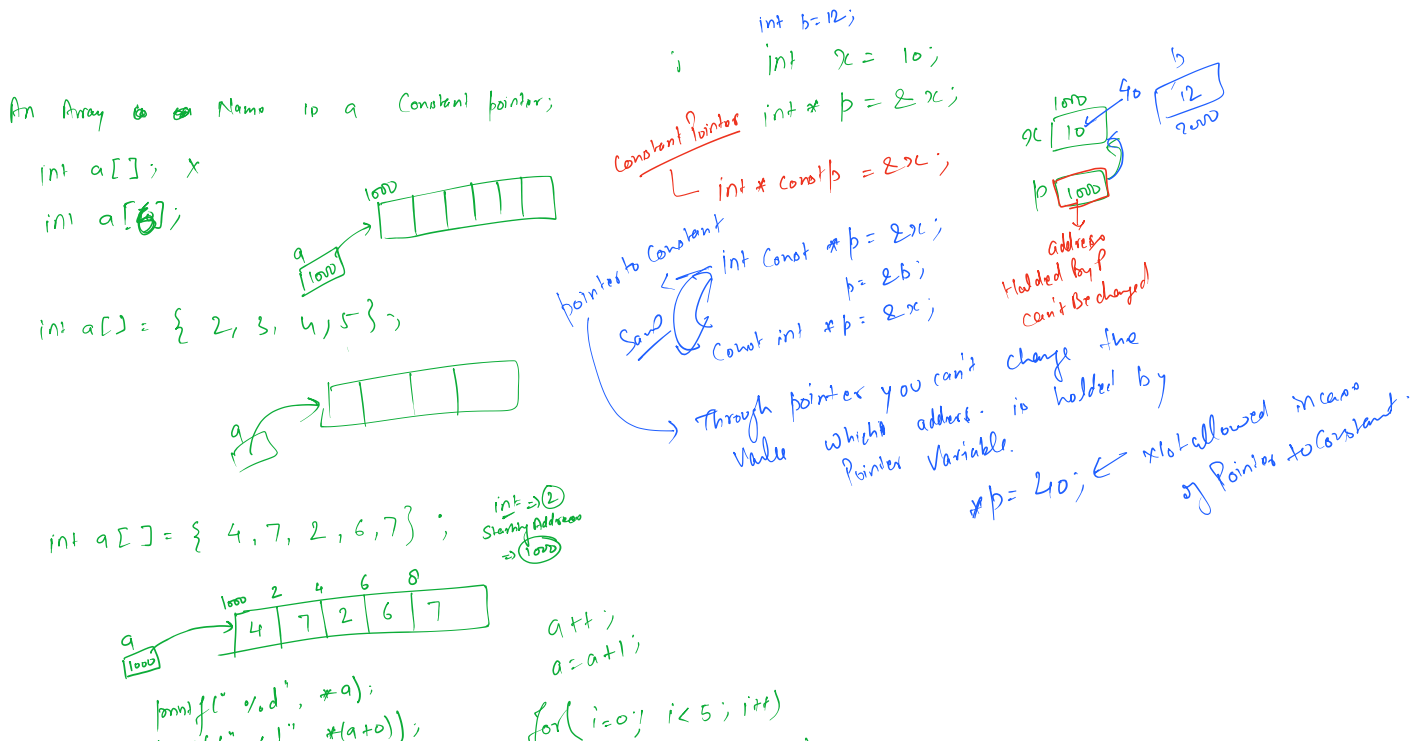
Syntax:

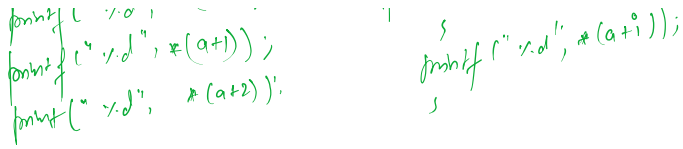
```
int * const p = &a;
```

👉 Meaning:

- The **pointer cannot move** to another variable.
- But the **data can be changed** through the pointer.







✅ Function Pointer in C

A **function pointer** is a pointer that stores the **address of a function**.
Just like pointers to variables store memory addresses of variables, a function pointer stores the address of code (a function).

⭐ Why use Function Pointers?

- To call functions dynamically
- To implement callback functions
- To build menus / event-driven programs
- To pass functions as arguments

✅ Syntax

return_type (*pointer_name)(parameter_list);
Example: pointer to a function returning int and taking two int arguments:

```
int (*ptr)(int, int);
```

📌 Example 1: Basic Function Pointer

```
#include <stdio.h>
int add(int a, int b) {
    return a + b;
}
int main() {
    int (*fptr)(int, int);  // declare function pointer
    fptr = add;             // assign function address
int result = fptr(5, 7); // calling function using pointer
printf("Result = %d", result);
    return 0;
}
```

Output:

```
Result = 12
```

📌 Example 2: Passing Function Pointer to a Function

```
#include <stdio.h>
void operation(int a, int b, int (*fun)(int, int)) {
    printf("Result = %d\n", fun(a, b));
}
int multiply(int x, int y) {
    return x * y;
}
int main() {
    operation(5, 6, multiply);   // pass function as argument
    return 0;
}
```

📌 Example 3: Array of Function Pointers (Menu System)

```
#include <stdio.h>
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int main() {
    int (*menu[3])(int, int) = { add, sub, mul };
int choice = 2;
    printf("Result = %d", menu[choice](10, 4));
    return 0;
}
```

If choice is 2 → multiplication → prints 40.

📌 Example 4: Function Pointer with typedef (Easy & Clean)

```
typedef int (*funptr)(int, int);
int add(int a, int b) { return a + b; }
int main() {
    funptr ptr = add;
    printf("%d", ptr(3, 4));
    return 0;
}
```

⭐ Summary Table

Concept	Meaning
Function Pointer	Stores address of function
Syntax	return_type (*ptr)(parameters)
Calling	ptr(a, b)
Uses	callbacks, menus, dynamic calling