MCQs on Arrays				
Question	**Code Snippet**	**Answers**	**Correct Answer**	**Explanation**

1	int numbers[5] = {10, 20, 30};
	printf("%d\n", numbers[1]);
		a) Prints 10
		b) Prints 20
		c) Prints garbage value
		d) Compile time error
			b)	The array numbers is initialized with values. numbers[1] accesses the second element (index 1) based on zero-based indexing.

2	char name[20];
	scanf("%s", name);
	printf("Hello, %s!\n", name);
		a) Prints "Hello, world!\n" b) Prompts user for a name and greets them
		c) Infinite loop
		d) Compile time error
			b)	scanf reads a string from the user, storing it in the name array until a whitespace character is encountered.

3	int arr[10];
	for (int i = 0; i < 10; i++)
	{
	arr[i] = i * i;
	}
	printf("arr[5] = %d\n", arr[5]);
		a) Prints arr[5] = 0
		b) Prints arr[5] = 5
		c) Prints arr[5] = 25
		d) Compile time error
			c)	The loop assigns squares of i (0 to 9) to each element. arr[5] holds the square of 5 (25).

4	int values[3] = {5, 10, 15}; printf("Sum of first two elements: %d\n", values[0] + values[1]);	a) Prints Sum of first two elements: 10 b) Prints Sum of first two elements: 15 c) Prints Sum of first two elements: 25 d) Compile time error	c)	The expression values[0] + values[1] adds the first two elements (5 + 10).

5	float temperatures[7];
	 for (int i = 0; i < 7; i++)
	{
	scanf("%f", &temperatures[i]);
	}
		a) Prints all temperatures
		b) Assigns random values to temperatures
		c)Prompts user to enter 7 temperatures
		d) Compile time error
			c)	The loop uses scanf to read 7 floating-point values from the user, storing them in the temperatures array. The address-of operator (&) is used to pass the memory address of each element to scanf.

6	char message[] = "Hello";
	printf("%s\n", message);
		a) Prints garbage value
		b) Prints Hello
		c) Infinite loop
		d) Compile time error
			b)	The message array is a string literal directly initialized with "Hello". printf prints the entire string using the %s format specifier for strings.

7	int arrSize = 8;
	int numbers[arrSize];
	for (int i = 0; i < arrSize; i++)
	{
	numbers[i] = 0;
	}
		a) Fills the array with random numbers
		b) Fills the array with 1s
		c) Fills the array with 0s
		d) Compile time error
			c)	The loop iterates through the array, assigning 0 to each element. This is a common initialization technique for numerical arrays.

MCQs on Pointers and Strings				
#	Code Snippet	Answers	Correct Answer	Explanation

1	char name[] = "Alice";
	char *ptr = name;
	printf("%c\n", *ptr);
		a) Prints garbage value
		b) Prints A
		c) Compile time error
		d) Unexpected output
			b)	ptr is a character pointer initialized with the address of the first character in name ("Alice"). *ptr dereferences the pointer to access the value at that address (which is 'A').

2	int num = 10;
	int *p = &num;
	printf("Value of num: %d\n", *p);
		a) Prints Value of num: 10
		b) Prints address of num
		c) Compile time error
		d) Unexpected output
			a)	p is an integer pointer that stores the memory address of num. *p dereferences p to access the value stored at that address (which is 10).

3	char str1[] = "Hello";
	char str2[20];
	strcpy(str2, str1);
	printf("%s\n", str2);
		a) Prints Hello (garbage after it)
		b) Prints Hello
		c) Compile time error
		d) Unexpected output
			b	strcpy copies the string from str1 to str2. Since str2 has enough space, the entire "Hello" is copied.

4	int arr[5] = {1, 2, 3, 4, 5};
	int *ptr = arr;
	printf("Second element: %d\n", *(ptr + 1));
		a) Prints Second element: 1
		b) Prints Second element: 2
		c) Prints Second element: 3
		d) Compile time error
			b)	ptr points to the first element of arr. ptr + 1 adds the size of an integer (usually 4 bytes) to ptr, effectively pointing to the second element. Dereferencing it with * prints the value (2).

5	char *message = "Welcome";
	message[0] = 'G';
	printf("%s\n", message);
		a) Prints Garbage value
		b) Prints Gelcome
		c) Compile time error
		d) Unexpected output
			b)	String literals are typically read-only. However, in some implementations, modifying the first character might work, resulting in "Gelcome". Note: This behavior is not guaranteed and can vary depending on the compiler.

6	char name[15];
	printf("Enter your name: ");
	scanf("%s", name);
	printf("Hello, %s!\n", name);
		a) Prompts user for a single character
		b) Prompts user for a name and greets them
		c) Prints garbage value
		d) Compile time error
			b)	scanf with %s reads a string from the user until a whitespace character. The entire string is stored in the name array.

7	char *ptr; ptr = "Hi there!";
	printf("%s\n", ptr);
		a) Prints Hi there!
		b) Compile time error
		c) Unexpected output
		d) Segmentation fault
			a)	Assigning a string literal to a pointer directly initializes it with the address of the string in memory (which is constant). Dereferencing ptr with printf prints the entire string. Note: This might not be allowed in all compilers. Check for specific compiler behavior.

8	int numbers[] = {10, 20, 30};
	for (int i = 0; i < 3; i++)
	{
	printf("%d ", *(numbers + i));
	}
		a) Prints garbage values
		b) Prints 10 20 30
		c) Prints addresses of elements
		d) Compile time error
			b)	

MCQs on Structures & Unions				
#	**Code Snippet**	**Answers**	**Correct Answer**	**Explanation**

1	struct Point
	{ int x;
	 int y;
	};
	struct Point pt = {5, 3};
	printf("Point coordinates: (%d, %d)\n", pt.x, pt.y);
		a) Prints Point coordinates: (0, 0)
		b) Prints Point coordinates: (5, 3)
		c) Compile time error
		d) Unexpected output
			b)	This defines a Point structure with x and y coordinates. The pt variable is initialized with values (5, 3). Member access is done using the dot (.) operator.

2	union Data
	{
	int num;
	float value;
	};
	union Data data;
	data.num = 10;
	printf("Value in float: %f\n", data.value);
		a) Prints Value in float: 0.000000
		b) Prints Value in float: 10.000000
		c) Compile time error
		d) Unexpected output
			c)	Unions share the same memory location. Assigning to data.num overwrites the value previously stored in data.value. The output on accessing data.value is undefined.

3	struct Student
	{
	char name[20];
	int age;
	 };
	struct Student std1;
	strcpy(std1.name, "Alice");
	std1.age = 22;
	printf("Student name: %s\n", std1.name);
		a) Prints Student name: (garbage value)
		b) Prints Student name: Alice
		c) Compile time error
		d) Unexpected output
			b)	A Student structure is defined. std1 is a variable of this type. strcpy is used to copy "Alice" to the name member of std1.

4	typedef struct
	{
	float length;
	float width;
	}
	Rectangle;
	Rectangle rect = {5.0, 3.0};
	printf("Rectangle area: %.2f\n", rect.length * rect.width);
		a) Prints Rectangle area: 0.00
		b) Prints Rectangle area: 15.00
		c) Compile time error
		d) Unexpected output
			b)	typedef creates an alias Rectangle for the structure. rect is initialized with length (5.0) and width (3.0). The area is calculated using member access.

5	struct Book { char title[50]; char author[30]; };
	struct Book book1;
	scanf("%s %s", book1.title, book1.author);
	printf("Book: %s by %s\n", book1.title, book1.author);
		a) Prints Book: (garbage value) by (garbage value)
		b) Prompts user for book details and prints them
		c) Compile time error
		d) Unexpected output
			b)	scanf reads two strings (title and author) and stores them in the respective members of book1. Member access is used for printing.

6	union Color
	{ int red; char green; };
	union Color col; col.red = 255;
	printf("Green value: %d\n", col.green);
		a) Prints Green value: 0
		b) Prints Green value: 255
		c) Compile time error
		d) Unexpected output
			b)	Assigning to col.red modifies the same memory location used by col.green. The output depends on the character encoding used (might not be exactly 255).