Q.n1

The output of the program will be 20.

Explanation:

The integer variable 'r' has a value of 20.'p' points to the address of 'r' and is a pointer to an integer.'p' is an integer pointer that is passed to the fun() function.

'q' is a local integer variable with the value 10 inside fun(). The address of "q" is set for the pointer "p."Since a copy of the pointer is supplied to the function, the change to 'p' inside of fun() has no effect on the pointer's original value in the main function.

The output 20 is produced by the main function, which prints the value of \*p, which still points to "r".

So, the pointer box for this program would look like this:

main:

+------+------+ fun:

| r | 20 | +------+

| p---|----->|---+ | q |

+------+------+ | | 10 |

| +------+

|

+--->p

2.

The output of the program will be 10.

The integer variable 'r' has a value of 20.'p' points to the address of 'r' and is a pointer to an integer. A pointer to pointer to an integer (\*\*pptr) is the argument for the function fun().

'q' is a static local integer variable with the value 10 inside the fun() function. The address of 'q' is set as the value of \*pptr.As a pointer to the pointer is supplied to the function, this modification has an impact on the original pointer in the main function.

The main function outputs 10 as a result of printing the value of \*p, which now points to "q."

main:

+------+------+ fun:

| r | 20 | +------+

| p---|----->|------>| q |

+------+------+ | 10 |

+------+

^

|

+----|----+

| pptr |

+--------+

3.

The output of the program will be 25

'a' is an integer array containing {1, 2, 3, 4, 5}.

'ptr' is a pointer to an integer, initialized to the address after the end of the array 'a'.

The main function prints the value of \*(a+1), which is the second element of the array (2).

It also prints the value of \*(ptr-1), which is the last element of the array (5).

+------+------+------+------+------+

a: | 1 | 2 | 3 | 4 | 5 |

+------+------+------+------+------+

^

|

&a

+------+------+------+------+------+

ptr:| &a[5] |

+------+------+------+------+------+

^

|

ptr

4.

The running result is - TEST sQuiz Z CQ

+---+ +---+---+---+---+

c: | | ---> | G | e | k | s | "GeksQuiz"

+---+ +---+---+---+---+

| M | C | Q | | "MCQ"

+---+---+---+---+

| T | E | S | T | "TEST"

+---+---+---+---+

| Q | U | I | Z | "QUIZ"

+------+------+------+------+

cp: | c+3 | c+2 | c+1 | c |

+------+------+------+------+

+------+ +------+------+------+------+

cpp: +->| cp[0]|\ | c+3 | c+2 | c+1 | c |

| +------+ +-------+ +------+------+------+------+

| | cp[1]|\ | | G | M | T | Q |

| +------+ +--+ | +------+------+------+------+

| | cp[2]| | | | e | C | E | U |

| +------+ | | +------+------+------+------+

| | cp[3]| +----+ | k | Q | S | I |

| +------+ | +------+------+------+------+

+------------------+

The first four string literals in the program, c, each have a pointer to the first character. Four pointers to pointers that point to the elements of c in reverse order are contained in the cp array. The first element of cp is where the cpp pointer is initialized to point.

The \* and [] operators are used to dereference pointers and access characters in strings, while cpp is incremented and decremented as the program runs to point to various components of cp.

The graphic displays the variables' memory locations as well as the values to which they refer. The boxes denote arrays and string literals, whereas the arrows denote pointers.

5.

The output is- GeeksQuiz

+------+ +-----+

str: +->| | | |

| | | |

| | | |

| | | |

| | | |

| | | |

| | | |

| | | |

| | | |

| | | |

+------+ +-----+

+------+

str\_ref: | |

+------+

At the beginning of the program, str is a pointer to the "GeeksQuiz" string that is saved in the first element of a block of 100 characters that was dynamically allocated. With its starting pointer pointing to the same memory location as str, str\_ref is a pointer to a pointer to a character.

Since the address of str is supplied by reference when fun is called with &str, str\_ref now points to the same location in memory as str. It now points to the memory location following str when str\_ref is incremented in fun, but this has no impact on the initial value of str in main.

After fun has finished running, the function puts is called with the argument str, printing the string "GeeksQuiz" to the console.

6.

The output is : gh

An array of character pointers called "argv" contains references to the string literals "ab," "cd," "ef," "gh," "ij," and "kl."The character pointer 'p' is passed as the first argument to the function f(). Its initial value is 'argv'.The function reads the preceding element, which is the third element of the array, "ef," before increasing the pointer "p" by the size of an int.The primary function outputs the value "ef" as the result.

+---+ +-----+ +-----+ +-----+ +-----+ +-----+

| |---->| "ab" |---->| "cd" |---->| "ef" |---->| "gh" |---->| "ij" |

+---+ +-----+ +-----+ +-----+ +-----+ +-----+

| | | | | | | | | |

+-----+ +-----+ +-----+ +-----+ +-----+

p = &argv[0] (p points to "ab")

p += sizeof(int) (p now points to "gh")

t = (p -= sizeof(int))[-1] (t points to "gh")

7.

The output is: 2 3 5 6

This program defines a pointer to an array of three numbers called ptr that points to the first row of the two-dimensional array a, which has two rows and three columns. The second and third elements of the first row of an are then printed, ptr is then incremented to point to the second row of a, and the second and third elements of the second row of an are then printed.

+-------+ +------+------+------+

| a | ----------> | 1 | 2 | 3 |

+-------+ +------+------+------+

| | | | 4 | 5 | 6 |

| v | +------+------+------+

+-------+ int a[][3] = {1, 2, 3, 4, 5, 6};

+-------+

| ptr | ----------> +-------+

+-------+ | a |

+-------+

| | |

| v |

+-------+

int (\*ptr)[3] = a;

+-------------+

| printf(...) | (\*ptr)[1] is 2 and (\*ptr)[2] is 3

+-------------+ printf("%d %d ", (\*ptr)[1], (\*ptr)[2]);

+-------+

| ptr++ | ptr now points to the second row

+-------+

+-------------+

| printf(...) | (\*ptr)[1] is 5 and (\*ptr)[2] is 6

+-------------+ printf("%d %d\n", (\*ptr)[1], (\*ptr)[2]);

8:

The output is :

0 1 2 2 3

+----+----+----+----+----+

ptr | | | | | |

+----+----+----+----+----+

| | | | |

v v v v v

+----+----+----+----+----+

| 0 | 1 | 2 | 3 | 4 |

+----+----+----+----+----+

'ptr' is a pointer to an integer, and memory is allocated for an array of 5 integers.The program initializes the array with values 0, 1, 2, 3, 4.The main function prints the value of \*ptr++ (0) and increments the pointer.Then, it prints the value of (\*ptr)++ (1) and increments the value at the pointer.It prints the value of \*ptr (1).Then, it prints the value of \*++ptr (2) and increments the pointer.Finally, it prints the value of ++\*ptr (3) and increments the value at the pointer.

#9:

The output : 20 10

This program declares the integer array 'arr' with two elements and initial values of 10 and 20. After that, we call the "fun()" method and give the array's first element an address. To make the pointer 'arr' point to the second element of the array, we use pointer arithmetic in the 'fun()' function to add 1 to the pointer. Then the value 20 of the second element of "arr" is printed. In the'main()' function, we print the value of the first element of 'arr' once more, which is still 10, and then we return 0 to stop the program.

+----+----+

| 10 | 20 |

+----+----+

arr arr

|

v

+----+

| 20 |

+----+

Q.no.2

1.

Result: Value of a is 10

Explanation:

The function fun() takes an integer a and prints its value.

In the main() function, a function pointer fun\_ptr is defined with the same signature as fun(), and it is initialized with the address of fun().

The function pointer fun\_ptr is called with the argument 10. It points to fun() and executes it with the given argument, printing "Value of a is 10".

Here is the visualization of the memory representation of the program:

+-----------------+

| Code of fun() |

+-----------------+

| printf() code |

+-----------------+

| a |

+-----------------+

+-----------------+

| Code of main() |

+-----------------+

| fun\_ptr |

+-----------------+

| 10 |

+-----------------+

2.

Result: Varies based on user input. The output is : Enter Choice: 0 for add, 1 for subtract and 2 for multiply 1

Explanation:

The functions add(), subtract(), and multiply() perform arithmetic operations on two integer arguments and print the results.In the main() function, an array of function pointers fun\_ptr\_arr is defined with the same signature as the arithmetic functions and initialized with the addresses of these functions.The user is prompted to enter a choice (0 for add, 1 for subtract, and 2 for multiply).The choice is stored in variable ch.

If ch is greater than 2, the program exits.The function pointer array fun\_ptr\_arr is used to call the corresponding arithmetic function based on the user's choice, using arguments a and b

+---------------+

| |

| fun\_ptr\_arr |

| |

+---------------+

|

v

+----+ +----+ +----+

| | | | | |

|add |--|sub |--|mul |

| | | | | |

+----+ +----+ +----+

| | |

v v v

+----+ +----+ +----+

| a1 | | a2 | | a3 |

+----+ +----+ +----+

| b1 | | b2 | | b3 |

+----+ +----+ +----+

3.

Result: Fun1

Fun2

Explanation:

The functions fun1() and fun2() print "Fun1" and "Fun2", respectively.The function wrapper() takes a function pointer fun with no arguments and no return value, and calls it.

In the main() function, wrapper() is called with fun1 and fun2 as arguments. This results in the execution of both functions and prints "Fun1" . In this visualization, there are three boxes representing functions: main, wrapper, and either fun1 or fun2 depending on which function is being called. The arrows indicate the flow of control: main calls wrapper, which in turn calls either fun1 or fun2.

+------+ +-------+ +-------+

| main | ---> |wrapper| ---> | fun1 |

+------+ +-------+ +-------+

^

|

| +-------+

+-----> | fun2 |

+-------+

4.

Result: 5 10 12 15 80 90

Explanation:

The function compare() takes two const void \* pointers and casts them to integer pointers, then returns the difference of the integer values they point to.

In the main() function, an integer array arr is defined and initialized.

The function qsort() is called with arr, its size n, the size of an integer, and the function pointer compare as arguments. qsort() sorts the array in ascending order using the provided comparison function.

The sorted array is printed, resulting in "5 10 12 15 80 90".

+----+----+----+----+-----+-----+

arr | 10 | 5 | 15 | 12 | 90 | 80 |

+----+----+----+----+-----+-----+

+----+----+----+----+-----+-----+

arr | 5 | 10 | 12 | 15 | 80 | 90 |

+----+----+----+----+-----+-----+

5. The output is: Return index is 2.

+-----+

void \*a ->| | --> +-----+

| arr | | 2 |

+-----+ +-----+

| 5 |

+-----+

| 7 |

+-----+

| 90 |

+-----+

| 70 |

+-----+

This is a visualization of an integer array arr and an integer variable x using a memory diagram. The array arr has five elements {2, 5, 7, 90, 70} and the variable x has a value of 7.

The pointer arr points to the first element of the array which is 2. Each element of the array occupies a memory space of the size of an integer (usually 4 bytes) and is stored in contiguous memory locations.

The pointer x points to another memory location where the value 7 is stored. This value can be compared with the elements of the array using the compare function in the search function.

Q.no.3

1.

The output is : \*ptr1=10

\*ptr2 =10

+-----+

void \*p ->| | --> +---+

| 10 | | 10|

+-----+ +---+

^

|

ptr2

In this program, we declare an integer variable `i` and initialize it to `10`. We then declare two integer pointers `ptr1` and `ptr2`, and set `ptr1` to point to `i`. We call the `returnPointer` function and pass it the address of `i`. The function simply returns the pointer passed to it. We set `ptr2` to the value returned by `returnPointer`.

The program then uses the `printf` function to print the values pointed to by `ptr1` and `ptr2` to the console.

The output shows that both `ptr1` and `ptr2` point to the same value, which is `10`.

Program 2

An integer pointer is returned by the program's returnFunc function, which is named after it. The & operator is used to get the location of the local variable i, which is created with a value of 10. The address is then returned as a pointer.

The value returned by returnFunc is assigned to the pointer ptr, which is declared in the main function. In the returnFunc function's local variable i, this pointer points to the memory location of that variable.

+-------------+

| main |

|-------------|

| ptr |

+-------------+

|

|

v

+-------------+

| returnFunc |

|-------------|

| i = 10 |

+-------------+

^

|

|

+-------------+

| |

| (unallocated memory)

| |

+-------------+

| returnFunc |

|-------------|

| i = 10 |

+-------------+

^

|

|

+-------------+

| |

| (unallocated memory)

| |

+-------------+

As shown in the diagram, the local variable i in returnFunc is deallocated when the function returns, so the memory location that ptr points to is no longer valid. Therefore, the behavior of the program is unpredictable and may result in a segmentation fault or other runtime errors.

Program 3

Running result:

\*ptr = 10

\*ptr = 10

Pointer box:

+------+ +------+

static i->| 10 | | |

+------+ +------+

^

|

ptr

In this program, we declare a static integer variable `i` inside the `returnFunc` function and initialize it to `10`. We then return a pointer to `i`. Since `i` is declared as `static`, it is not destroyed when the function returns, and its address remains valid.

The program then declares an integer pointer `ptr`, and sets it to the value returned by the `returnFunc` function. The program uses the `printf` function to print the value pointed to by `ptr` to the console three times.

The output shows that all three calls to `printf` correctly print the value `10`, which is the value of `i`.