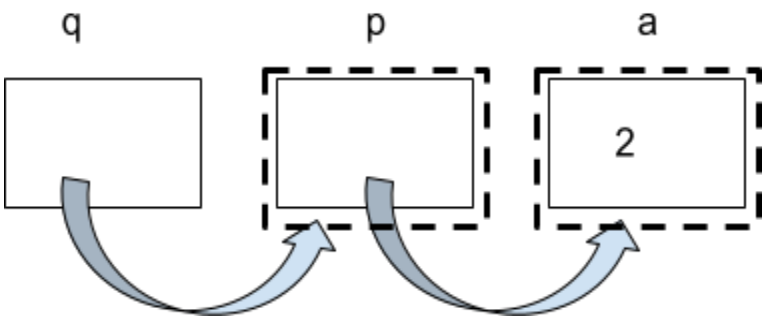
1. Here is a program, and answer some questions

```c
int main(void)
{
    int a = 2, *p = &a, **q = &p;

    printf("The value of a is 0x%-8x, Its address is %p\n", a, &a);
    printf("The value of p is 0x%-8x, Its address is %p\n", p, &p);
    printf("The value of q is 0x%-8x, Its address is %p\n", q, &q);

    return 0;
}
```

   a. Draw a picture to show the relationship between these 3 variables



   b. What does the following expression mean?

```c
*a, **a;
*p, **p;
*q, **q;
```

Explained these pointer expressions:

1. For variable a (integer) :

   *a - Invalid operation. Cannot dereference a non-pointer variable

   **a - Invalid operation. Cannot double dereference a non-pointer variable

2. For pointer p (pointer to int) :

   *p - Value at address stored in p (value of a = 2)

   **p - Invalid operation. Cannot double dereference a single pointer

3. For pointer q (pointer to pointer) :

   *q - Value at address stored in q (address of a)

   **q - Value at address obtained after double dereferencing (value of a = 2)

c. Based on the main program. What can you learn from the following program?

```
int **p1 = &a;
int *p2 = &p;

printf("%d %d\n", *p1, **(int **)p2);
```

What we have learned from the program :

i. Pointer type safety matters
   1. The assignments **int **p1 = &a**; and **int *p2 = &p;** are incorrect. They violate pointer type compatibility rules.
   2. The program might work due to implicit type conversions, but it is not guaranteed and depends on the compiler.

ii. Pointer casting can sometimes "fix" type mismatches but is dangerous.
   1. The ****(int **)p2** works because of an explicit cast, but it is risky and should be avoided unless absolutely necessary.

iii. Strict compilers will give warnings/errors for such incorrect assignments.

2. Read the program below, and answer some questions

```
int main(int argc, char *argv[], char *envp[])
{
    return 0;
}
```

a. What kind of data is stored in argv and envp? Where are they stored in memory?
argv (argument vector) stores command-line arguments passed to the program. **envp** (environment pointer) stores environment variables available to the program. The last entry in both is NULL.
Both argv and envp are stored in the stack segment of the process memory.

b. How many ways to get the content of argv?

```c
#include <stdio.h>

int main(int argc, char *argv[], char *envp[]) {

    // Method 1: Array notation
    printf("Method 1:\n");
    for(int i = 0; i < argc; i++) {
        printf("argv[%d]: %s\n", i, argv[i]);
    }

    // Method 2: Pointer arithmetic
    printf("\nMethod 2:\n");
    char **arg = argv;
    while(*arg) {
        printf("%s\n", *arg++);
    }


    return 0;
}
```

c. What is the difference between **char *argv[]** and **char **argv** in here?

Both **char *argv[]** and **char **argv** are equivalent in function signatures because in C, an array name decays into a pointer.

However, they have different conceptual representations:

i. char *argv[] : Array of pointers to characters (each pointing to a string).

ii. char **argv : Pointer to a pointer (points to the first element of an array of character pointers).

3. In which scenario is a pointer to pointer used?

   Core scenarios where double pointers are used

```c
// 1. Dynamic 2D Arrays
    int **matrix = malloc(rows * sizeof(int *));
    for(int i = 0; i < rows; i++) {
        matrix[i] = malloc(cols * sizeof(int));
    }


//2. Modifying Pointer in Functions
    void modify_ptr(int **ptr) {
        *ptr = malloc(sizeof(int));
        **ptr = 10;
    }


//3. String Arrays (Command Line Args)
    int main(int argc, char **argv) {
        // char *names[] = {"John", "Jane", "Bob"};
    }


//4. Linked List Operations
    void insert(Node **head, int data) {
        Node *new = malloc(sizeof(Node));
        new->next = *head;
        *head = new;
    }
```

```c
//5. Memory Management
    void free_array(int **arr, int rows) {
        for(int i = 0; i < rows; i++) {
            free(arr[i]);
        }
        free(arr);
    }



//6. Buffer Management
    void resize_buffer(char **buf, size_t *size) {
        *buf = realloc(*buf, *size * 2);
        *size *= 2;
    }
```

4. What is the relationship between an array of pointer and a pointer to pointer?
    a. **Array of Pointers (type *arr[N])** stores multiple pointers. **Pointer to Pointer (type **ptr)** stores the address of a single pointer.They are related because an array of pointers can be accessed using a pointer to pointer.

    A **pointer to a pointer** can be used to manipulate an **array of pointers** dynamically.

    Example:

```c
#include <stdio.h>

int main() {
    char *arr[] = {"Hello", "World", "C", NULL};  // Array of pointers to strings.
    char **ptr = arr;                             // Pointer to pointer pointing
                                                  // to the first element.

    while (*ptr) {
        printf("%s\n", *ptr);
        ptr++;                                    // Move to the next pointer.
    }

    return 0;
}

```