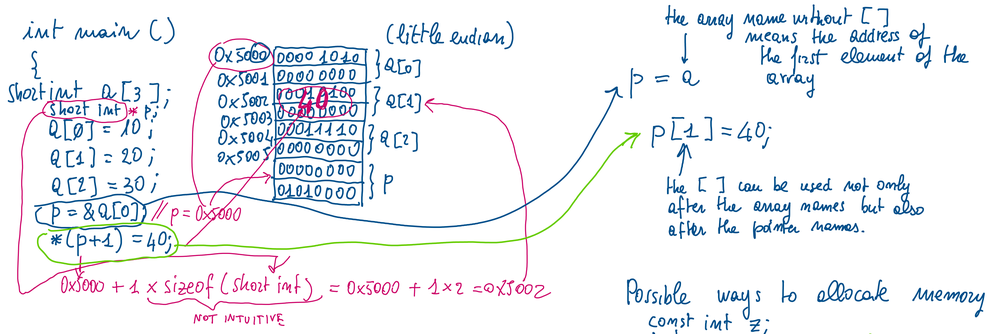
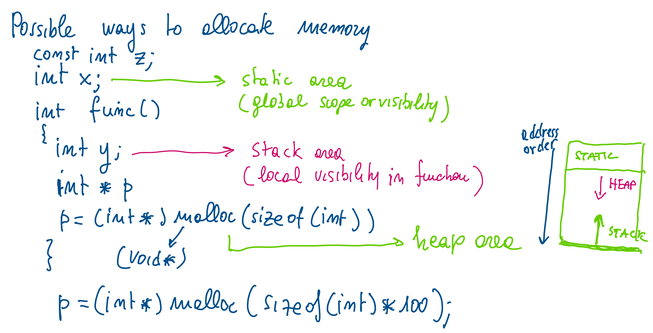
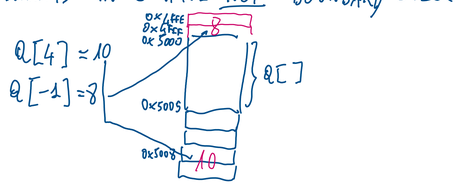
2.3 ARRAY VS POINTERS

mercoledì 28 febbraio 2024 08:52







The [] operator in C is used for array subscripting. It provides a more intuitive way to access elements of an array, but under the hood, it's essentially syntactic sugar for pointer arithmetic. Here's how it relates to pointers and how it compares with using pointer offsets directly.

The [] Operator

- Meaning: Accesses an element of an array.
- Example:

```
Int arr[3] = {10, 20, 30};
Int value = arr[1]; // Accesses the second element of arr, value is now 20
```

Array Subscripting with Pointers

The expression arr[i] is equivalent to *(arr + i). This shows the close relationship between arrays and pointers in C. Here's how you can achieve the same with pointers:

- Pointer Equivalent:

```
int*ptr = arr;
```

- Int value = *(ptr + 1); // Also accesses the second element of arr, value is now 20

This demonstrates that the [] operator is essentially performing pointer arithmetic to access the desired array element.

Comparison Between Arrays and Pointer Offset

When you use an array name in an expression, it is (in most cases) converted to ("decays into") a pointer to its first element. This means that the array name acts very much like a pointer constant, pointing to the first element of the array.

- Direct Array Access vs. Pointer Offset:
  - Using arr[i] directly accesses the ith element of the array arr.
  - Using *(arr + i) or *(ptr + i) (where ptr is a pointer to the first element of arr) accesses the ith element by moving i positions from the start, leveraging pointer arithmetic.

Advantages and Disadvantages

- Readability: The [] operator makes code easier to read and understand, especially for those familiar with arrays but not the intricacies of pointers.
- Flexibility: Using pointer arithmetic (*(ptr + i)) might offer more flexibility in some scenarios, especially when dealing with dynamic memory allocation or manipulating parts of an array based on conditions that change at runtime.
- Performance: There is no performance difference between using [] and explicit pointer arithmetic, as the compiler translates [] into the equivalent pointer arithmetic operation.

Example

Let us illustrate how a char *p pointer can be used to access and manipulate three variables char x, y, z through increments. Imagine having a simple memory table where these variables are allocated consecutively. Note: The specific memory address of the variables depends on the system and program execution. The addresses in the table below are purely hypothetical for educational purposes.

Memory Address	Variable	Value
0x100	x	'A'
0x101	y	'B'
0x102	z	'C'

Here is described how a char *p pointer can be initialized to point to x and then used to access and modify y and z through increments.

Example Code

```
#include<stdio.h>
Int main(){
 charx = 'A', y = 'B', z = 'C';
 char*p = &x; // p points to the address of x
 printf("Before: x = %c, y = %c, z = %c\n", x, y, z); // p now points to x, so *p is 'A'
*p = 'X'; // Modifies the value of x through p
p++; // Increments p to point to y
*p = 'Y'; // Modifies the value of y through p
p++; // Increments p to point to z
*p = 'Z'; // Modifies the value of z through p
printf("After: x = %c, y = %c, z = %c\n", x, y, z); return0; }
```

Expected Output

```
mathematicaCopy code
Before:x=A,y=B,z=C
After:x=X,y=Y,z=Z
```

Explanation

- The pointer p is initialized to point to the address of x.
- By modifying the value pointed by p (*p), you directly modify the value of x in memory.
- Incrementing p (p++), the pointer moves to the next memory address, which in our simplified example corresponds to the location of y, and then z, allowing also these values to be modified.