Array e Puntatori

lunedì 19 dicembre 2022 21:13

ARRAY E PUNTATORI

file: sizeof.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
        int nums[] = { 10, 12, 13, 14, 20 };
        char name[] = {'A', 'l', 'e', 's', 's', 'a', 'n', 'd', 'r', 'o', '\0'};
        char *name_ptr = name;

        printf("The size of an int: %ld\n", sizeof(int));
        printf("The size of nums (int[]): %ld\n", sizeof(nums));
        printf("The number of ints in nums: %ld\n\n", sizeof(nums) / sizeof(int));

        printf("The size of a char: %ld\n", sizeof(char));
        printf("The size of name (char[]): %ld\n", sizeof(name));
        printf("The number of chars: %ld\n\n", sizeof(name) / sizeof(char));

        printf("The size of name_ptr: %ld\n", sizeof(name_ptr));
        printf("Bugged number of chars: %ld\n", sizeof(name_ptr) / sizeof(char));

        return 0;
}
```

la dimensione la mette lui!

Puntatore al char

4

5 · 4 = 20

20 / 4 = 5

1

11 · 1

11 / 1 = 11

8 → 8 perché è indirizzo

8 / 1 = 8

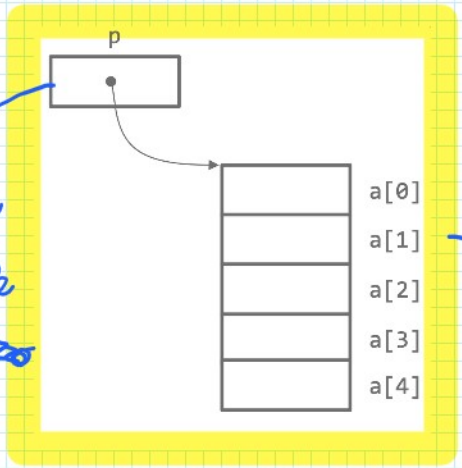
Name è un VETTORE di caratteri!

↳ e ha l'indirizzo del primo elemento.

* name_ptr ha lo stesso indirizzo!

MA NON SOLO LA STESSA COSA!

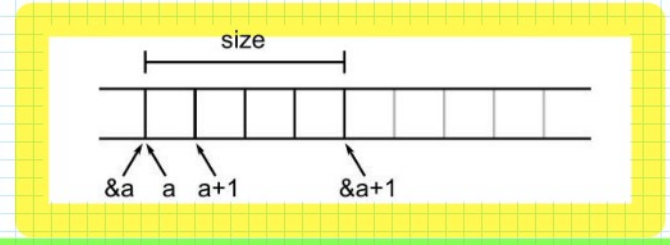
- Vi è una forte relazione tra vettori e puntatori
- Il nome della variabile usata per dichiarare un vettore può facilmente *decadere* ad un puntatore al primo elemento
- La seguente assegnazione è perfettamente legale:
  ```c
  int a[4] = {3, 2, 1, 0};
  int *p = a;
  ```
- Nell'esempio, a[1] e p[1] generano entrambi l'indirizzo dove il valore 2 è memorizzato



È IN MEMORIA

è una variabile dentro cui ho SCRITTO L'INDIRIZZO

→ NON OCCUPA SPAZIO IN MEMORIA!



&a+1 = vado avanti di TUTTO il VETTORE, perché mi porto appresso TUTTA la dimensione di a!

&a = TUTTA la dimensione del VETTORE.

LA RIGA di CIASCUNA matrice VIENE letta TUTTA così, perché è contigua in memoria!

ESEMPIO

```c
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>


int *buggy_return(void);
void using_stack(void);



int main()
{
    int *dummy = buggy_return();
    printf("%d\n", *dummy);
    using_stack();
    printf("%d\n", *dummy);
    return 0;
}
int *buggy_return(void)
{
    int a_variable = 10;
    return &a_variable;
}
void using_stack(void)
{
    char a_string[] = "Hello World!";
    printf("%s\n", a_string);
}
```

non è più legale valido. solo un undefined behaviour

OUTPUT

```
10
Hello World!
894064505
```

Restituisce l'indirizzo di una variabile locale!

la alloca sullo STACK, e RESTITUISCO UN INDIRIZZO che si TROVA sullo STACK, quindi quel RECORD viene invalidato logicamente!

va sullo STACK, sopra la precedente, la SOVRASCRIVE.

Voce 16 : 11:00