RIPASSO C
Parte 2

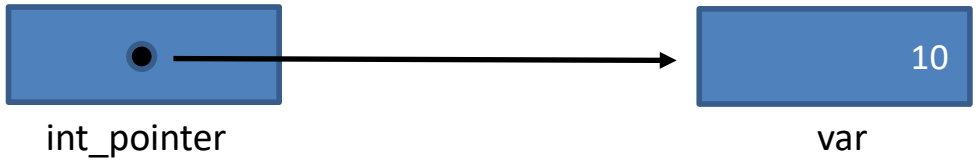
C language

Pointers, arrays and other…

Puntatori

- **int** var = 10;

- **int** *int_pointer;
  - Il simbolo ***** indica che la varibile è di tipo **puntatore** (in questo caso a intero).
- int_pointer = &var;
  - Il simbolo **&** imposta un riferimento indiretto tra il puntatore e var. Si chiama operatore di *indirizzamento* e restituisce l'indirizzo di memoria di una variabile.
  - Non assegna a int_pointer il valore di var!!!



Puntatori

```
1   #include <stdio.h>
2
3   int main()
4   {
5       int a=10;
6       int *p;
7       p=&a;
8
9       printf("%d %d %d %d %d",a, &a, p, *p, &p);
10
11      return 0;
12  }
13
```

Operatore di *indirezione* *
Restituisce il contenuto dell'oggetto puntato!

```
10 1550072728 1550072728 10 1550072720
```

Pointers

Name														
Value														
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113

```
int a=20;
char b='Z';
```

a, b: contents of the variables
&a, &b: addresses of a, b

Name	**a**								**b**					
Value	20								Z					
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113

```
int a=20;
char b='Z';
int *p=&a;
```

p : content of the variable

&p: address of p

Name	a				p				b					
Value	20				100				Z					
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113

```
int a=20;
char b='Z';
int *p=&a;
```

p : content of the variable

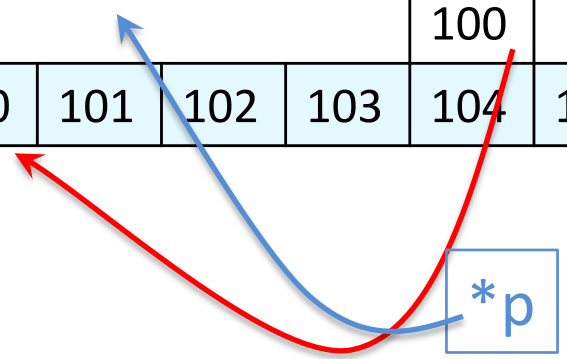
&p: address of p

*p: Indirection

("object pointed to by p")

contents of the memory location pointed to by p

Name	a				p				b					
Value	20				100				Z					
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113

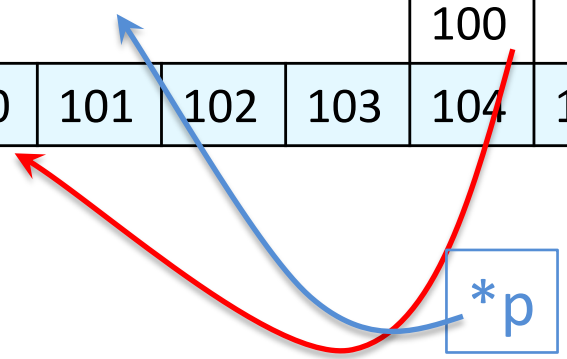


(*p==a) is True!

```
int a=20;
int *p=&a;
a=a+2;
(*p)++;
printf("*p: %d\n",*p);
printf("a: %d\n",a);
```

Output: ?

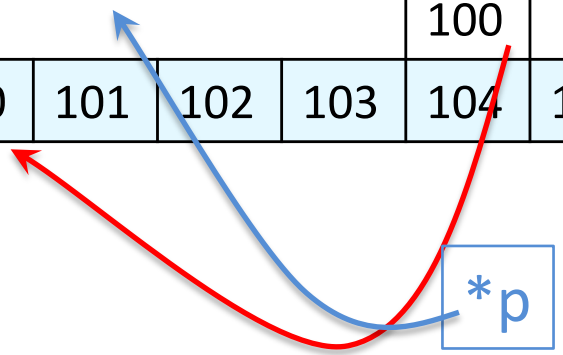
Name	a				p									
Value	20				100									
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113



```
int a=20;
int *p=&a;
a=a+2;
(*p)++;
printf("*p: %d\n",*p);
printf("a: %d\n",a);
```

Output: 23

Name	a				p									
Value	20				100									
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113



Puntatori ed espressioni

```c
//struct_esame
#include <stdio.h>


int main()
{
    int n1, n2;
    int *p1, *p2;

    n1=10;
    p1=&n1;
    n2=*p1 / 2 +10;
    p2=p1;

    printf("n1 = %d, n2 = %d, *p1 = %d, *p2 = %d\n",n1,n2,*p1,*p2);

    return 0;
}
```

OUTPUT ?

Puntatori ed espressioni

```c
//struct_esame
#include <stdio.h>


int main()
{
    int n1, n2;
    int *p1, *p2;

    n1=10;
    p1=&n1;
    n2=*p1 / 2 +10;
    p2=p1;

    printf("n1 = %d, n2 = %d, *p1 = %d, *p2 = %d\n",n1,n2,*p1,*p2);

    return 0;
}
```

OUTPUT 10, 15, 10, 10

Arrays and Pointers

```
int arr[4]={10, 11, 12, 13};
int *p_a=arr;
```

arr, &arr[0], p_a

<u>address</u> of the element in position 0

(arr + k), &arr[k] , p_a+k

<u>address</u> of the element in position k

***(arr + k) , arr[k] ,*(p_a + k)**

<u>element</u> in position k

Arrays and Pointers

DIFFERENCE:

```
int arr[4]={10, 11, 12, 13};
int *p_a=arr;
```

p_a++ ;
p_a=arr ;

are legal (**p_a** is a variable)

arr++ ;
arr=p_a ;

are NOT legal (**arr** is a CONSTANT)

Arrays and Pointers

The array name points to the first element of the array.

```
int a[10];
printf("address:%p\n", a);
printf("address:%p\n", &a[0]);
printf("address:%p\n", &a[1]);
```

```
address:0x7fff5958b7d0
address:0x7fff5958b7d0
address:0x7fff5958b7d4
```

Arrays and Pointers

The array name points to the first element of the array. **IT IS A CONSTANT!**

```
int a[10];
printf("address:%p\n", a);
printf("address:%p\n", &a[0]);
printf("address:%p\n", &a[1]);
```

```
int * b;
int c=0;
b=&c;
```

OK

```
int * b;
int c=0;
a=&c;
```

ERROR

```
int * b;
int c=0;
*a=c;
```

OK

```
int * b;
int c=0;
a[0]=c;
```

OK

Puntatori e array

- Il nome di un array rappresenta un puntatore **costante** al suo primo elemento: punta sempre al primo elemento e non è possibile assegnare l'indirizzo di un'altra cella di memoria

```
int vett[100];
int *vett_ptr;
vett_ptr = vett;
```
equivale a `vett_ptr = &vett[0];`

vett_ptr punta il primo elemento di vett. Non si usa &. Il nome di un array senza indice viene infatti trattato come puntatore al primo elemento di un array

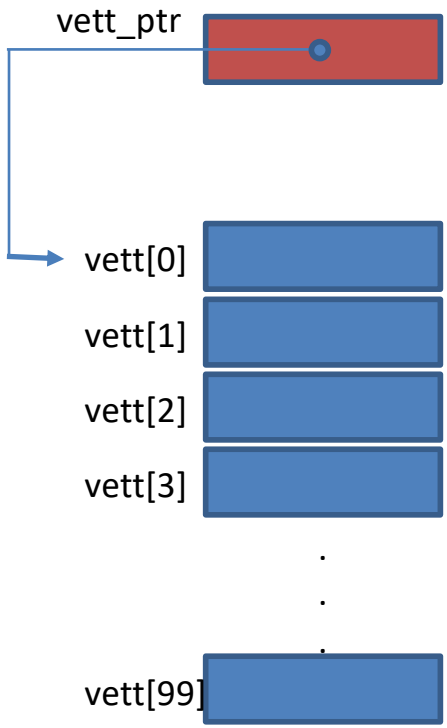
Puntatori e array

```
int vett[100];
int *vett_ptr;

vett_ptr = vett;
```

o in alternativa:

```
vett_ptr = &vett[0];
```



Aritmetica dei puntatori

- Incremento: permette di passare ad un indirizzo successivo
  - p++, p+1, p+i
- Decremento: permette di passare ad un indirizzo precedente
  - p--, p-1, p-i

- **Lo spostamento dipende dal tipo base!**

Pointers and functions

Example: use a function to print elements of an array

```
5   void printArray(int *p, int n){
6     for (int i=0; i<n; i++){
7       printf("%d\n",*(p+i));
8     }
9   }
10
11  int main() {
12    // INIT
13    int arr[4]={10, 11, 12, 13};
14    printArray(arr,4);
15    getchar();
16    return 0;
17  }
```

What's the problem with this code?

```
int *p;
*p=100;
```

Name					**p**									
Value														
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113

What's the problem with this code?

```
int *p;
*p=100;
```

Name					**p**									
Value														
Address	100	101	102	103	104	105	106	107	108	109	110	111	112	113

We don't know what is the location pointed out by p

What's the problem with this code?

```
typedef struct {
   float x;
   float y;
   float z;} Coord;
Coord *set_coord(){
   Coord xyz;
   xyz.x=1.1;
   xyz.y=2.2;
   xyz.z=3.3;
   return &xyz;
}
```

struct

alias

What's the problem with this code?

```
typedef struct {float x,y,z;} Coord;
Coord *set_coord(){
  Coord xyz;
  xyz.x=1.1;
  xyz.y=2.2;
  xyz.z=3.3;
  return &xyz;
}
int main(){
  Coord point_1;
  point_1=*set_coord();
  printf("%0.2f - %0.2f - %0.2f ",
  point_1.x,point_1.y,point_1.z);
  getchar();
}
```

What's the problem with this code?

```
typedef struct {float x,y,z;} Coord;
Coord *set_coord(){
  Coord xyz;
  xyz.x=1.1;
  xyz.y=2.2;
  xyz.z=3.3;
  return &xyz;
}
int main(){
  Coord point_1;
  point_1=*set_coord();
  printf("%0.2f - %0.2f - %0.2f ",
  point_1.x,point_1.y,point_1.z);
  getchar();
}
```

warning: address of stack memory associated with local variable 'xyz' returned

What's the problem with this code?

The function **set_coords** return a pointer to a struct, but the struct 'live' in the local space of the function!

A function should **never return a pointer to a local variable**, even though the compiler will probably not complain.

Why pointers?

Why Pointers?

- Exchange data between modules (efficiently)
- Dynamic allocation

Why Pointers?

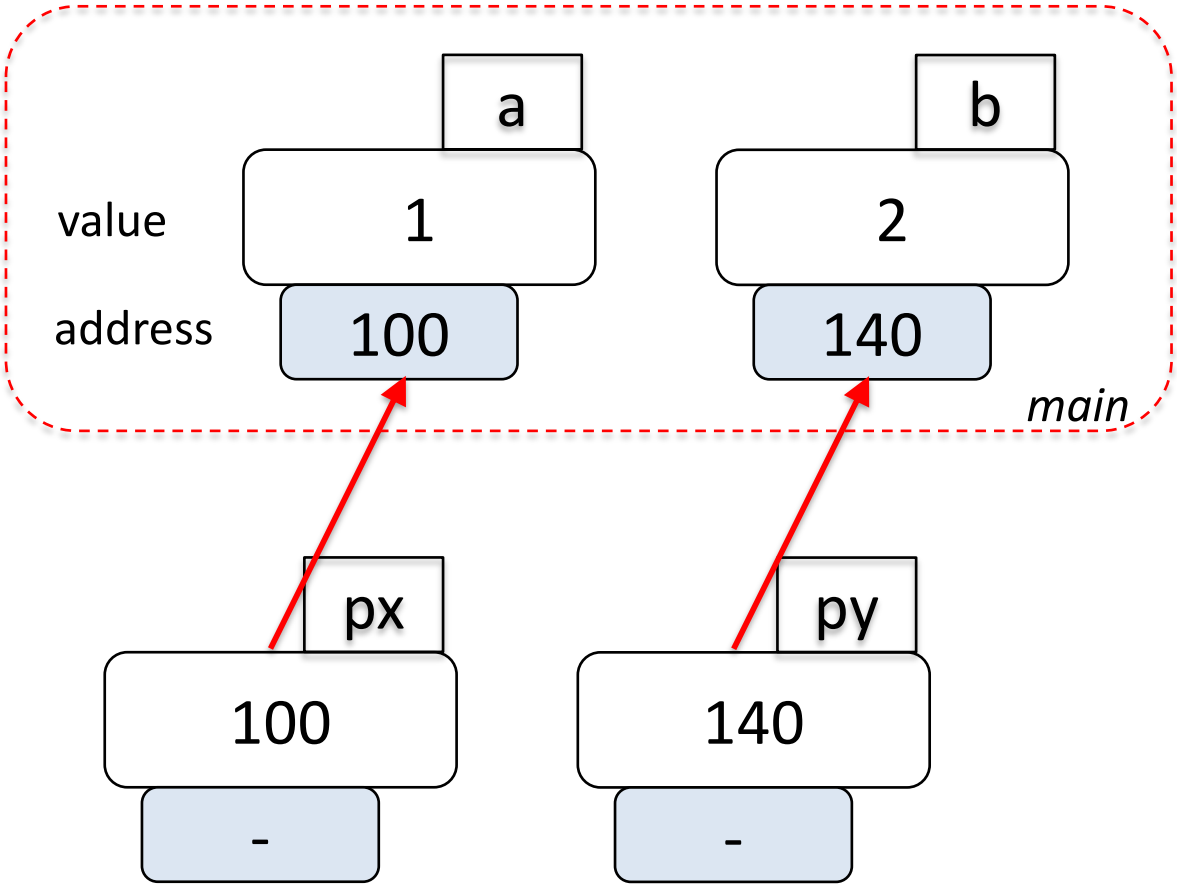
```
void swap (int x, int y){
    int temp;
    temp=x;
    x=y;
    y=temp;
}
swap (a,b);
```

Why Pointers?

```
void swap (int *px, int *py){
    int temp;
    temp=*px;
    *px=*py;
    *py=temp;
}
swap (&a,&b);
```

Why Pointers?

```
void swap (int *px, int *py){
    int temp;
    temp=*px;
    *px=*py;
    *py=temp;
}
swap (&a,&b);
```



Why Pointers?

```
const int N=5;
typedef int vett[N];


vett v1;
v1[10]=10;
printf("%d\n", v1[0]);
```

!!!

Why Pointers?

```
int main() {
  // ...
  a=f1();
  // ...
  return 0;
}
int f1(){
  //...
}
int f2(){
  //...
}
```

complete control over
how much memory to allocate and
<u>when</u> to allocate and release it

Write a program in C to count the number of vowels and consonants in a string using a pointer

fgets(str1, sizeof str1, stdin);

reads a line, stores it into the string pointed to by str.

It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.