

A decorative vertical bar on the left side of the slide, consisting of several vertical lines of varying shades of blue and grey. To the right of these lines are several blue circles of different sizes, some overlapping the lines. One of the circles contains the number '1'.

# **Laboratorio di Programmazione 1**

**Docente: dr. Damiano Macedonio**  
**Lezione 19 - 14/04/2014**

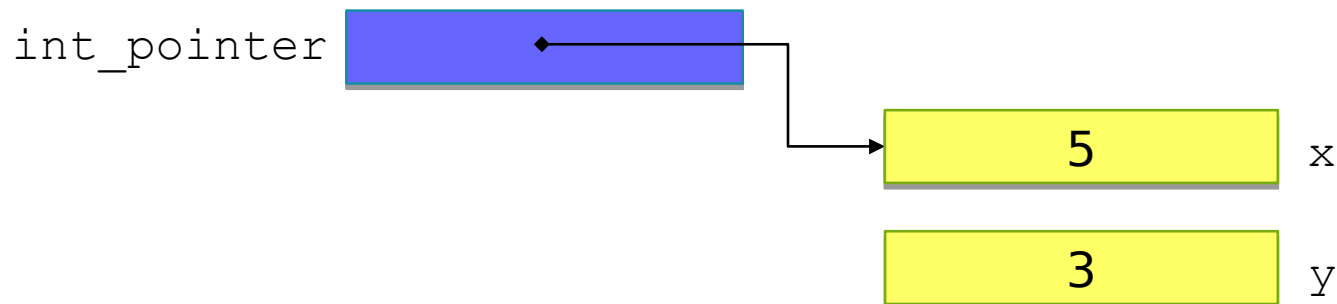
# Puntatori

- I *puntatori* sono tipi di dati che rappresentano la posizione (*indirizzi di memoria*) di elementi (oggetti) di un programma come variabili e strutture.
- Un puntatore viene *dichiarato* utilizzando il simbolo *\**:
  - `int *int_pointer; // puntatore ad int`
  - La variabile `int_pointer` è un puntatore ad `int`, cioè può essere usata per memorizzare la posizione in memoria di una variabile di tipo `int`.
- La *creazione* di un puntatore ad un oggetto avviene attraverso l'*operatore di indirizzamento* `&`.
  - `int x; // variabile intera`
  - `int *int_pointer; // puntatore ad int`
  - `int_pointer = &x;`
  - L'operatore di indirizzamento assegna ad `int_pointer` un puntatore alla variabile `x`, non il valore di `x`.

# Puntatori

- Per accedere *indirettamente*, attraverso un puntatore, al valore dell'oggetto puntato si usa l'*operatore di indirazione \**.

- `int x = 3; // variabile intera`
- `int *int_pointer = &x; // puntatore ad int`
- `int y = *int_pointer;`
- L'operatore di indirazione assegna ad `y` il valore identificato indirettamente da `int_pointer`.
  
- `*int_pointer = 5;`
- L'operatore di indirazione assegna alla variabile puntata da `int_pointer` (cioè `x`) il valore 5.



# Puntatori nelle Espressioni

- I puntatori possono essere utilizzati all'interno di espressioni aritmetiche.
- L'operatore di indirizzamento `&` e l'operatore di indirezione `*` hanno precedenza più alta rispetto a tutti gli operatori binari del C.

```
int i1, i2;  
int *p1, *p2;  
i1 = 5;  
p1 = &i1;  
i2 = *p1/2 + 10; // equivale a i1/2 + 10  
p2 = &i2;
```

# Puntatori a Strutture

- Un puntatore può puntare anche ad una struttura.

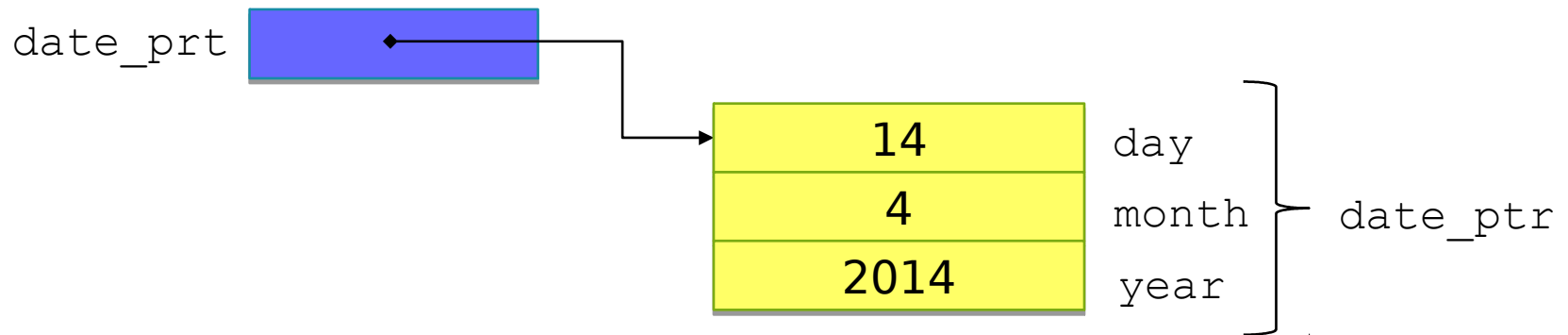
- ```
struct date {  
    int day;  
    int month;  
    int year;  
}
```

```
struct date *date_ptr; // puntatore a struttura  
date_ptr = &struct_date;
```

- Per accedere *indirettamente* ad un membro della struttura si usa l'operatore di indirezione \* opportunamente combinato con le parentesi.

- ```
int day = (*date_ptr).day;
```
- ```
(*date_ptr).month = 5;
```
- Le parentesi sono necessarie perché l'operatore . di accesso ai membri della struttura ha **precedenza più alta** rispetto agli operatori di indirezione.

# Puntatori a Strutture



- I puntatori a strutture sono usati molto spesso, pertanto è stato definito un operatore speciale chiamato *operatore dei puntatori a struttura* `->`:

- `(*x) . y` può essere riscritto come `x->y`

- `int day = date_ptr->day;`

- `date_ptr->month = 5;`

# Puntatori e Funzioni

- I puntatori possono essere utilizzati come argomenti o valori di ritorno di una funzione.
  - `void print_date(struct date *today);`
  - `struct el *find_element(struct li *list);`
- Quando un puntatore è passato come argomento di una funzione è necessario ricordare che:
  - Il valore del puntatore (quindi l'indirizzo) viene copiato nel parametro formale quando la funzione viene chiamata.
  - Qualsiasi modifica al *valore* del puntatore non ha effetto sul puntatore che è stato passato alla funzione.
  - Al contrario, i dati ai quali il puntatore fa riferimento possono essere modificati!

# Puntatori e Funzioni: Argomenti

```
void test1(int *int_pointer) {  
    *int_pointer = 5; // modifica il valore puntato  
}
```

```
void test2(int *int_pointer) {  
    int i = 6;  
    int_pointer = &i; // modifica il puntatore  
}
```

```
int maini(void) {  
    int n = 8;  
    int *p = &n; // *p == 8  
    test1(p); // *p == n == 5  
    test2(p); // *p == n == 5  
}
```



# Esercizio 1

- Scrivere un programma C che dichiara e inizializza (con valori a piacere) un `int`, un `double` e un `char`. Quindi dichiara e inizializza un puntatore a ciascuna delle tre variabili. Il programma infine stampa l'indirizzo, il valore memorizzato e lo spazio occupato (in byte) di ciascuna delle sei variabili.

- Per stampare un indirizzo con la funzione `scanf()` usate la formattazione `"%p"`.

```
int x = 5;
printf("%p", &x);
// l'indirizzo di x viene stampato come valore esagesimale
```

- Per determinare lo spazio di memoria allocato per ogni variabile usate l'operatore `sizeof()` che ha tipo `long unsigned int` e individua il numero di byte occupati dall'argomento.

```
sizeof(x);
```

## Esercizio 2

- Aggiungete il codice che manca e scoprite gli indirizzi delle variabile `x` in `foo1` e `y` in `foo2`. Che cosa notate? Riuscite a spiegare quello che succede?

```
#include <stdio.h>
void foo1(int xval) {
    int x;
    x = xval;
    /* stampate l'indirizzo e il valore di x */
}

void foo2(int dummy) {
    int y;
    /* stampate l'indirizzo e il valore di y */
}

int main(void) {
    foo1(7);
    foo2(11);
    return 0;
}
```

## Esercizio 3

- Scrivere una funzione

```
void sum(double *s, double x, double y)
```

- Che assegna la somma tra il secondo e il terzo argomento (passati per valore) al primo argomento (passato per riferimento).
- La funzione deve essere richiamata dalla funzione `main`, la quale si occupa anche di richiedere due `double` all'utente e stampare il risultato.

## Esercizio 4

- Scrivere una funzione `sort` che ordina 3 interi in ordine crescente.
- La funzione non deve utilizzare un array ma 3 puntatori.
- La funzione deve essere richiamata dalla funzione `main`, la quale si occupa anche di richiedere i tre interi all'utente e stampare il risultato.

## Esercizio 5

- Scrivere una funzione

```
void split_time(long int tot_sec,  
                struct time *t)
```

che, dato un orario fornito in numero di secondi dalla mezzanotte, calcoli l'orario equivalente in ore, minuti, secondi, e lo memorizzi nella struttura `time (ore, minuti, secondi)` puntata da `t`.

- La funzione `split_time()` deve essere richiamata dalla funzione `main()`, la quale richiede all'utente il numero di secondi, calcola l'orario aggiornato e ritorna il risultato.

# Esercizio 6

- Scrivere una funzione

```
void updateDate(struct date *d)
```

che riceve in ingresso un puntatore ad una struttura `date` e modifica la data aggiungendo 10 giorni.

- La data modificata deve essere una data valida!
- La funzione `updateDate()` deve essere richiamata dalla funzione `main()`, la quale richiede all'utente una data, calcola la data aggiornata e ritorna il risultato.
  - Eseguire un check sulla data eseguita: finché la data eseguita non è valida viene richiesta una data valida all'utente.