

# **Visibilità e tempo di vita delle variabili**

**(più sintesi di alcuni concetti della  
prima parte del corso)**

# Struttura dei programmi C/C++

Un programma C/C++ deve essere contenuto in uno o più **file** (in questo corso si assume tutto in un file):

- Direttive per il *pre-processor*: **#** tipicamente per l'inclusione di file di intestazione (**#include**)
  - Definizione della **funzione speciale** **main()**
- 
- Definizione di eventuali funzioni **funz1()**, **funz2()**, ...
  - Eventuali prototipi di funzioni
  - Eventuali definizioni/dichiarazioni di costanti/variabili al di fuori delle funzioni, come stiamo per vedere ...

# 1. Direttive al *pre-processor*

## SINTASSI: `#comando`

Noi abbiamo visto la sola direttiva:

`#include <nomefile>` (utilizzata per includere altri file)

Esempio: `#include <iostream>`

Esempio: `#include <cmath>`

Questa direttiva serve per utilizzare funzioni e variabili definite all'interno di altri file.

Ad esempio, grazie all'inclusione del file `iostream`, siamo riusciti ad utilizzare gli oggetti *cout*, *cin* e *cerr*

## 2. Funzione `main()`

- `main()` è una funzione speciale con tre caratteristiche:
  - deve essere sempre presente
  - è la prima funzione che viene eseguita ovunque si trovi all'interno del file sorgente (stessa cosa vale nel caso di programma su più file)
  - quando termina l'esecuzione del `main()`, termina il programma
- In C la funzione `main()` contiene due sezioni (la prima può eventualmente essere vuota)
  - **Parte dichiarativa**
  - **Parte esecutiva**

# Esempio in linguaggio C

## Esempio

```
#include <stdio.h>
```

← Direttive al  
pre-processore

```
main()
```

```
{ <dichiarazioni>
```

← Parte dichiarativa

```
<istruzioni>
```

← Parte esecutiva

```
}
```

Non vi sono simboli speciali per separare la parte dichiarativa dalla parte esecutiva

# Esempio in linguaggio C++

## Esempio

```
#include <iostream>
```

Direttive al  
pre-processor



```
main()
```

```
{ <dichiarazione o istruzione>
```

```
  <dichiarazione o istruzione>
```

```
  ...
```

```
}
```

# Concetto di blocco

<blocco in C++> ::=

```
{  
  <istruzione o dich.>  
  <istruzione o dich.>  
  ...  
}
```

<blocco in C> ::=

```
{  
  [ <dichiarazioni >; ]  
  <istruzioni>;  
}
```

## NOTE

- dopo un blocco non occorre il punto e virgola (il ; *termina* le istruzioni semplici, non è un *separatore* di istruzioni)
- i blocchi possono essere innestati

# 3. Concetto di funzione

- L'astrazione di *funzione* è presente in tutti i linguaggi di programmazione di alto livello.
- È l'astrazione del concetto di *operatore* su un tipo di dato, ovvero è un *componente software* che cattura l'astrazione matematica di funzione:

$$f : A \times B \times \dots \times Q \rightarrow S$$

con molti possibili ingressi ed **una sola uscita** (corrispondente al risultato). Esempio:

```
int funzione1 ([ parametri; ] )  
{  
    ...  
}
```

↑  
**possono anche mancare**



# Esempio struttura dei programmi C

```
#include <stdio.h>

dichiarazioni;

funzione1()
    { <dichiarazioni>
      <istruzioni>  }

dichiarazioni;

funzione2()
    { <dichiarazioni>
      <istruzioni>  }

main()
    { <dichiarazioni>
      <istruzioni>  }
```

# Esempio struttura dei programmi C++

```
#include <iostream>

dichiarazioni;

funzione1()
{ <dichiarazioni mescolate ad istruzioni>
}

dichiarazioni;

funzione2()
{ <dichiarazioni mescolate ad istruzioni>
}

main()
{ <dichiarazioni mescolate ad istruzioni>
}
```

# ***Regole di visibilità degli identificatori (scope) e tempo di vita delle variabili***

# Caratteristiche di una variabile (*già note*)

**Nome:** identificatore

**Indirizzo (lvalue):** locazione di memoria a partire dalla quale è memorizzato l'oggetto riferito dalla variabile

**Valore (rvalue):** contenuto dell'area di memoria associata alla variabile

**Tipo:** insieme di valori che la variabile può assumere e di operazioni ad essi applicabili

# Caratteristiche di una variabile *completamento*

**Nome:** statico

**Indirizzo (lvalue):** statico

**Valore (rvalue):** dinamico

**Tipo:** statico

- Il C/C++ è un linguaggio con tipizzazione statica e forte
- Il tipo di una variabile non cambia durante l'esecuzione del programma

# Visibilità e tempo di vita

**Campo di visibilità (*scope*) di un identificatore:**  
parte (del testo) di un programma in cui  
l'identificatore può essere usato

in C, C++, Pascal → determinato staticamente

in LISP → determinato dinamicamente

**Tempo di vita di un oggetto: intervallo di  
tempo in cui l'oggetto è mantenuto in memoria**

in FORTRAN → tempo di vita statico

in C, C++, Pascal → tempo di vita dinamico

**NOTA: *Scope e tempo di vita possono non coincidere***

# Regole di visibilità *statiche* e *dinamiche*

- Regole di visibilità: regole con cui si stabilisce il campo di visibilità degli identificatori
- **Due alternative**
  - Regole di visibilità **statiche**: il campo di visibilità è stabilito in base *al testo del programma*, cioè in base a *dove sono dichiarati* gli identificatori
  - Regole di visibilità **dinamiche**: il campo di visibilità è stabilito dinamicamente in base *alla sequenza di* *chiamata* delle funzioni

# Dichiarazioni, visibilità e tempo di vita

Vi sono tre casi principali di dichiarazione di identificatori:

1. Identificatori dichiarati all'interno di un blocco (di una funzione o di un qualsiasi altro blocco) o nella lista dei parametri formali di una funzione
2. Identificatori dichiarati nel blocco del *main()*
3. Identificatori dichiarati all'esterno di funzioni



# Caso 1: Variabili interne ad un blocco

- Il campo di visibilità degli identificatori dichiarati all'interno di un blocco è il blocco stesso
- Sono **visibili**: dal punto in cui sono dichiarati fino alla fine del blocco (con una eccezione!)
- Se si tratta di definizioni di variabili/costanti, queste ultime hanno **tempo di vita** che va dal momento della definizione fino alla fine dell'esecuzione delle istruzioni del blocco (senza eccezioni)

## ECCEZIONE

- Se nel blocco compare un blocco innestato in cui è dichiarato un identificatore con lo stesso nome, la variabile del blocco esterno rimane in vita, ma non è visibile nel blocco innestato

# Esempio 1

```
#include <iostream>
```

```
funzione1 ()
```

```
{ int bz=5;  
  bz++;  
  cout<<bz;  
}
```

La variabile `bz` ha *tempo di vita* e *scope* (visibilità) limitato a tutto e solo il blocco della funzione `funzione1()`

```
funzione2 ()
```

```
{ int abc=0;  
  abc=abc*10;  
  cout<<abc;  
}
```

# Esempio 2

```
#include <iostream>
```

```
funzione1()
```

```
{ int bz=5;
```

```
  float ak=3.2;
```

```
  int ps=9;
```

```
  ...
```

```
  {
```

```
    int ps=bz+5;
```

```
    cout<<ps ; ?
```

```
  }
```

```
  cout<<ps ; ?
```

```
  ...
```

```
}
```

Le variabili `ak`, `bz`, `ps` hanno tempo di vita relativo a tutto blocco della funzione1()

Le variabili `ps`, `ps` hanno scope relativo a tutto il blocco della funzione1().

La variabile `ps` definita nel blocco esterno ha scope relativo a tutto il blocco della funzione1(), **fatta eccezione per il blocco interno.**

# Unicità e ordine di definizione degli identificatori

- Definizioni di costanti e variabili associano ad un **identificatore** una costante o una variabile
  - L'associazione dell'identificatore all'oggetto deve essere **unica** per tutto il suo *campo di azione*

## Esempi

```
{ int intervallo = 5;  
  int Y, Z, intervallo;  
}
```

Sono scorrette perché il simbolo **intervallo** è definito due volte nello stesso campo di azione (*blocco*)

# Caso 2: Variabili del *main()*

- In C/C++, le variabili dichiarate nella funzione `main()`
  - Se si tratta di definizioni di variabili/costanti, vivono quanto la funzione `main()`, ovvero hanno tempo di vita pari alla durata del programma
  - **ma, in generale non sono visibili in qualunque parte del programma**, perché `main()` è una funzione come tutte le altre
    - visibilità degli identificatori *limitata al blocco* in cui sono dichiarati

**NON SONO VARIABILI CON SCOPE GLOBALE!**

# Esempio 3

```
#include <iostream>
```

```
fun ()
```

```
    { int bz=5;  
      bz++;  
    }
```

La variabile `bz` ha *tempo di vita* e *scope* limitato al blocco della `fun()`

```
main ()
```

```
    { int as=10;  
      int xt=5;  
      xt=xt*as;  
      funzione1 ();  
      cout<<xt;  
    }
```

La variabile `as` e la variabile `xt` hanno *tempo di vita* pari alla durata del programma, ma *scope* limitato al blocco del `main()`.  
NON SONO VISIBILI IN `fun()`

# Caso 3: Variabili esterne alle funzioni

- Identificatori dichiarati all'esterno di funzioni sono visibili dal punto in cui sono dichiarati (non prima!) fino alla fine del file (con un'eccezione)
- Se si tratta di definizioni di variabili/costanti, le relative variabili/costanti hanno tempo di vita pari alla durata dell'intero programma (senza eccezioni)

## ECCEZIONE

- Se nel blocco di altre funzioni è dichiarata una variabile con lo stesso nome, la variabile esterna rimane in vita, ma non è visibile all'interno di quel blocco

# Struttura generale di un programma C/C++ all'interno di un unico file

## Esempio

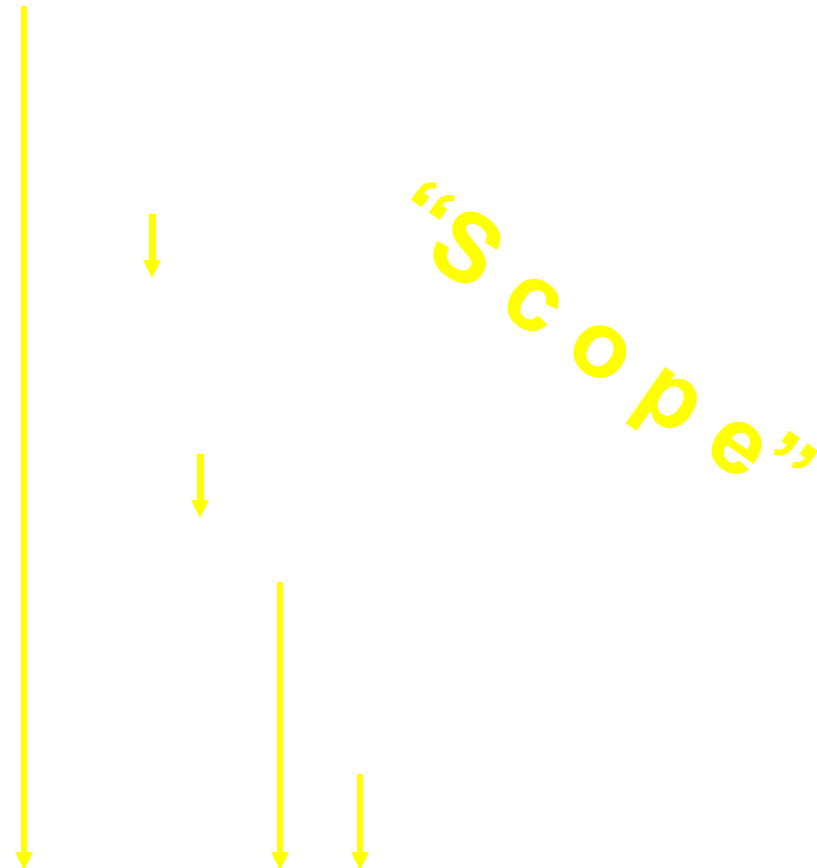
```
#include <iostream>
dichiarazioni;

funzione1()
{ <dichiarazioni>
  <istruzioni> }

funzione2()
{ <dichiarazioni>
  <istruzioni> }

dichiarazioni;

main()
{ <dichiarazioni>
  <istruzioni> }
```





# Esempio 4

```
#include <iostream>
int x=0;
funct1 ()
    { x = x + 3; }
funct2 ()
    {
        int x = 10;          /* è un'altra x rispetto
        funct1 ();          alla x globale! */
        cout<<x; /* x = ?? */ 10
    }
main ()
    {
        x++;
        cout<<x; /* x = ?? */ 1
        funct2 ();
        cout<<x; /* x = ?? */ 4
    }
```

# Nomenclatura

- Gli identificatori dichiarati nei blocchi sono comunemente denominati **locali** (al blocco)
  - Se si tratta di definizioni di variabili, si parla di **variabili locali** (al blocco)
- Gli identificatori dichiarati al di fuori dei blocchi, se il programma sta su un solo file, sono denominati **globali**
  - Se si tratta di definizioni di variabili ed il programma sta su un solo file, si parla di **variabili globali**

# Struttura generale di un programma C/C++ all'interno di un unico file

## Esempio

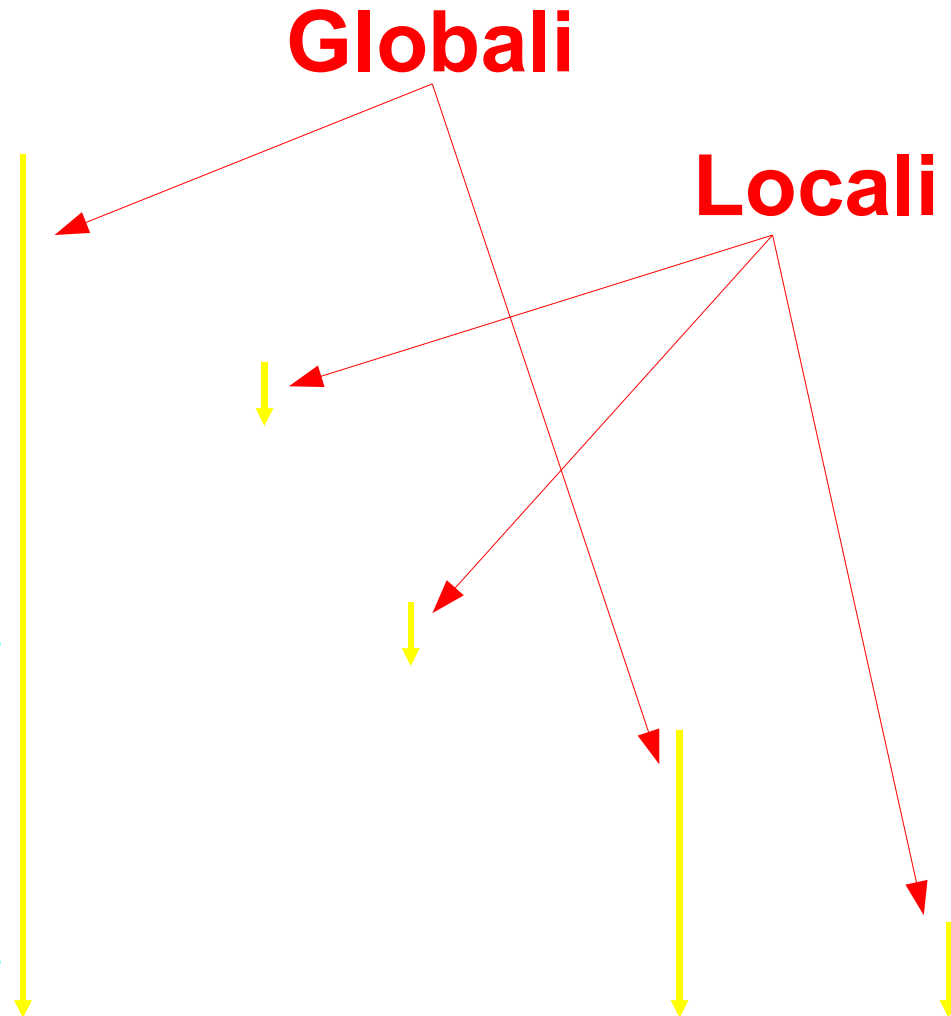
```
#include <iostream>
dichiarazioni;

funzione1()
{ <dichiarazioni>
  <istruzioni> }

funzione2()
{ <dichiarazioni>
  <istruzioni> }

dichiarazioni;

main()
{ <dichiarazioni>
  <istruzioni> }
```



# Valori iniziali oggetti

- Il valore iniziale di un oggetto, variabile o costante con nome, dipende da dove tale oggetto è definito
  - All'interno di un blocco: corpo di una funzione (incluso la funzione *main*) o qualsiasi altro blocco
    - Valore iniziale casuale
  - Al di fuori di tutti i blocchi
    - Valori iniziale 0

# Parole chiave della lezione

- Concetto di
  - Blocco
  - Funzione
- *Visibilità (scope)* degli identificatori
- Tempo di vita delle variabili