Lezione 08: Funzioni

Prototipi

- **Devono precedere la chiamata**.
- I nomi dei parametri possono essere **omessi**.
- Esempi:

```
void f();       // prototipo senza parametri (C++)
void f(...);    // prototipo con numero indefinito di parametri
void f(void);   // valido sia in C che in C++
```

Il **prototipo** abilita il controllo dei tipi a tempo di compilazione e consente la **conversione automatica** del tipo dove ha senso.

Passaggio dei Parametri

- In C++ il passaggio è **per valore**.
- **Attenzione con gli array**:

```
int sum_arr(int arr[], int n);     // equivalente a:
int sum_arr(int* arr, int n);
```

Funzioni Inline

- La parola chiave `inline` **suggerisce** al compilatore di sostituire il codice della funzione direttamente nel punto di chiamata.
- Utile per **funzioni molto brevi**

Puntatori a Funzione

- Il corpo di una funzione ha un indirizzo in memoria → può essere **referenziato**.
- Operazioni possibili:
  1. **Chiamare** la funzione

2. **Prenderne l'indirizzo**

- Il **dereferenziamento è opzionale**:

```
void (*pf)(int) = funzione;
(*pf)(5);   // equivalente a pf(5);
```

Ottimo per **parametrizzare algoritmi**, ad esempio passare una funzione di confronto a un algoritmo di ordinamento.

Funzioni Lambda

Cosa sono?

- Funzioni **anonime**, ispirate al lambda-calcolo.
- Possono essere usate direttamente come parametri:

```
[](int x){ return x % 3 == 0; }
```

Forniamo di seguito un esempio pratico dell'utilizzo di queste funzioni:

```
std::vector<int> nums(size);
int count3 = std::count_if(nums.begin(), nums.end(),
            [](int x){ return x % 3 == 0; });
```

Accesso a variabili esterne

- `[=]` tutte per **valore**
- `[&]` tutte per **riferimento**
- `[var]` solo `var` per **valore**
- `[&var]` solo `var` per **riferimento**
- `[=,&var]` tutte per valore tranne `var`

Possono essere **annidate**, **salvate in variabili**, e **definite localmente**:

```
auto filtro = [] (double x) -> double { int y = x; return x - y; };
count3 = std::count_if(nums.begin(), nums.end(), filtro);
```

Template di funzione

Cosa sono?

- Permettono di scrivere **funzioni generiche**.
- Il compilatore genera il codice per i tipi usati:
-

```
template <typename AnyType>
void Swap(AnyType &a, AnyType &b) {
    AnyType temp = a;
    a = b;
    b = temp;
}
```

Vanno scritti negli **header**. Per separare interfaccia/implementazione, si può includere il `.cpp` direttamente nel `.h`.