Callable

- Funzione
- Puntatore a funzione
- Function object *(Functor)*
  - Classe un cui oggetto è invocabile come se fosse una funzione
  - Ha uno stato, inizializzato dal ctor, e può ricevere parametri
- Espressione lambda – è in realtà un function object
  - Il compilatore genera una classe anonima equivalente
  - Il codice è più snello, con maggiore leggibilità
    - Minimale: **[]{}**
    - Completa: **[***capture list***] (***params***) ->** *type* **{** /*… */ **}**

Lambda

- Il tipo di ritorno di una lambda può essere dedotto dal compilatore
  - In assenza di return si assume void
- Le lambda sono spesso create e passate come argomento ad altre funzioni
  - Soprattutto se il codice è semplice
- Il loro uso è considerato idiomatico nel C++ moderno
  - Funzioni e function object sono ancora usate come approcci alternativi
- Un esempio, std::sort()
  - Accetta un callable come terzo argomento per confrontare gli elementi nel sorting
    - Due parametri, gli elementi da confrontare
    - Ritorna true se il primo precede il secondo

Capture list

- Permette di inizializzare una lambda al momento della sua creazione
  - Usando variabili locali definite nel contesto corrente
  - Agisce come un costruttore canonico per il function object generato dal compilatore
  - Le variabili catturate sono utilizzabili nel body della lambda
- Per default la cattura è effettuata **by value**
  - [variable_name]{ /* … */ }
- Indicando ‘&’, si richiede la cattura **by reference**
  - [&variable_name]{ /* … */ }
  - È responsabilità del programmatore usare il reference correttamente
- *Simile all’effetto di adattamento che si ottiene via bind per le funzioni*

Capture

- È possibile definire capture list più complesse
  - **[=]** tutte by value
    - **[=,** &a, …**]** esplicite (qui ‘a’, …) by reference, le altre by value
  - **[&]** tutte by reference
    - **[&,** a, ...**]** esplicite (qui ‘a’, …) by value, le altre by reference
- Le variabili catturate by value sono immutabili per default
  - Se si vuole modificarle nella lambda, keyword **mutable**
  - Dopo la lista dei parametri prima del body, es: [x]() mutable {/* … */}

Return

- Determinazione implicita del tipo di ritorno
  - Se il codice ritorna un valore e non ci sono ambiguità
  - Letterali interi sono interpretati come int
    - Se non è un valore tale da richiedere un long
  - Letterali reali sono interpretati come double
    - Se non è esplicitamente indicato come float, via F, o long double, via L
- È possibile esplicitare il tipo di ritorno
  - Sintassi “-> type” tra la lista dei parametri e il body della lambda
  - Se necessario viene operato un cast implicito al tipo indicato

Eccezioni

- Generazione di eccezioni in una lambda

```
[] { throw std::runtime_error{ "Hello!" }; }
```

- La specifica noexcept dichiara l’assenza di eccezioni

```
[]() noexcept { /* … */ };
```

- La violazione del contratto causa la fine del programma
  - Via esecuzione di std::terminate()