Funzioni

- Funzioni libere
  - Dichiarazione / definizione
- Inline
- Parametri
- Overloading
- Template
- Codice di esempio
  - https://github.com/egalli64/corso-cpp folder b4

Funzioni libere

- Funzione libera: non appartiene a nessuna classe
  - Nella programmazione Object Oriented una funzione che appartiene a una classe è detta metodo
- Ogni programma C++ inizia obbligatoriamente dalla funzione main()
  - int main() // se non ci interessa trattare i possibili argomenti passati al programma
  - int main(int argc, char** argv) // nome del programma e argomenti passati
- Una funzione può essere
  - dichiarata: viene indicato solo il prototipo, ma non il codice associato
  - definita: dichiarazione e codice associato, tra parentesi graffe
  - invocata: si passa il controllo a una funzione definita altrove
- Possiamo invocare una funzione
  - solo all'interno del body di una funzione
  - solo se il compilatore ne conosce il prototipo

Inline

- "inline" è un suggerimento per il compilatore
  - È libero di accettarlo o meno
- Si indica che si vorrebbe evitare una vera chiamata a funzione
  - Per non pagarne il costo
- Il codice della funzione inline viene copiato nel chiamante
  - Simile alle macro del linguaggio C, ma più sicure
- L’aspetto negativo è che l’eseguibile diventa più pesante
  - Se il compilatore accetta il suggerimento

Parametri

- By value – se non indicato diversamente, un argomento è passato in un parametro per valore
  - Il contenuto della locazione di memoria relativa all'argomento è copiato in quella del parametro
  - Cambiamenti del parametro all'interno del chiamato non si riflettono nell'argomento del chiamante
- By pointer – si può passare l'indirizzo di un argomento ad una funzione
  - Il contenuto della locazione di memoria dell'argomento è accessibile al chiamato
  - Rischioso, un puntatore può essere nullptr o non inizializzato correttamente
  - La sintassi è un po' ostica
- By reference – si può passare il riferimento di un argomento ad una funzione
  - Il contenuto della locazione di memoria dell'argomento è accessibile al chiamato
  - Più sicuro e un po' più comprensibile
- Default - si può dare un valore di default al parametro, nel caso l'utente non passi l'argomento
- By const reference / pointer – permettono di passare un oggetto di dimensioni considerevoli
  - Evitando di farne una copia e garantendone l'immutabilità

Overloading

- La "signature" di una funzione
  - Combina il nome della funzione e la lista dei suoi parametri
  - Il nome dei parametri non è rilevante
    - Conta solo il loro numero e tipo
- Due funzioni sono diverse se hanno diversa signature
  - Più funzioni nello stesso "scope" possono avere lo stesso nome
- Utile per una famiglia di funzioni che operano su tipi diversi
  - Se sostanzialmente fanno la stessa cosa

Template

- A volte si riscrive lo stesso codice cambiando solo il tipo delle variabili usate
  - L'uso di funzioni generiche ci permette di lasciar fare al compilatore
- Noi scriviamo il codice generico, il compilatore genera il codice specifico
  - L'uso di template permette di creare implicitamente overload di funzioni
- Prima della funzione, indichiamo il tipo generico via template <typename T>
- All'interno della funzione usiamo T come tipo della nostra variabile
  - T è qui una meta variabile, una variabile che definisce il tipo delle variabili usate
- Chi invoca il metodo fornisce il tipo effettivo
  - Il compilatore genera il codice per il tipo fornito, o dà errore
- Si può fornire una specializzazione esplicita con la notazione template <>
- Si possono avere più meta variabili: template <typename T, typename U>