Programmazione generica mediante template

21 gennaio 2022

Informatica, Corso B - D. Tamascelli

Parte 1: Basi

Programmazione generica

Definizione:

Paradigma di programmazione che, a partire dall'astrazione delle operazioni effettuate da funzioni simili, ma operati su tipi di dato differenti, permette di definire modelli di funzione unificati e parametrici.

La varietà delle funzioni originali è quindi recuperata attraverso diverse istanziazioni dei parametri.

Chiariamo...

```
void scambia(int *a, int *b){
int appo;
appo = *a;
*a = *b;
*b = appo;
}
```

```
void scambia(float *a, float *b)
{
float appo;
appo = *a;
*a = *b;
*b = appo;
}
```

Modello

```
void scambia(T *a, T *b){
T appo;
appo = *a;
*a = *b;
*b = appo;
}
```

- Le due varianti della funzione fanno le stesse operazioni su variabili di di tipo diverso.
- Il modello astrae rispetto alle differenze di tipo, lasciando il tipo parametrico (**T**).
- Le funzioni originali si ottengono per specializzazione del parametro (**T** ←**int**, **T** ←**float**).

Altro esempio...

```
float media(float v[], int dim)
{

float accu = 0;

for(int i=0; i<dim; i++)
accu+=v[i];

return accu/dim;

}
```

```
float media(int v[], int dim){

float accu = 0;

for(int i=0; i<dim; i++)
accu+=v[i];

return  accu/dim;

}
```

```
double media(double v[], int
dim){

double accu = 0;

for(int i=0; i<dim; i++)
accu+=v[i];

return accu/dim;

}
```

Modello

```
T1 media(T2 v[], int dim){

T1 accu = 0;

for(int i=0; i<dim; i++)
accu+=v[i];

return accu/dim;

}
```

Idea

- Consentire di lasciare il tipo di:
    - Parametri di Input
    - Variabili locali
    - Valore di ritorno

  NON specificati al momento della scrittura del codice

- La specifica del tipo dei parametri, variabili, ecc... viene sostituita da "segnaposto con nome", altrimenti detti parametri di tipo.
- Le funzioni originali, raggruppate dal modello, si ottengono assegnando un particolare valore al parametro di tipo.

Template in C++

- Parole chiave
  - template
  - typename (class)
- Costrutti:
  - template <typename T>
  - template <typename T1, typename T2>
  - template <class T>
- Uso
  - template <typename T> precede la dichiarazione/definizione del modello di funzione(classe) e si riferisce solo a quello.
  - nello scope del typename T fa da segnaposto per il parametro tipo.

```
template <typename T>
void scambia(T *, T *);
.....

template <typename T>
void scambia(T *a, T *b){
T appo;
appo = *a;
*a = *b;
*b = appo;
}
```

Altro esempio...

Modello

```
T1 media(T2 v[], int dim){
T2 accu = 0;
for(int i=0; i<dim; i++) accu = accu + v[i];
return  accu/dim;
}
```

```
template <typename giusy, typename tony>
giusy media(tony v[], int dim);

.....

template <typename T1, typename T2>
T1 media(T2 v[], int dim){
T1 accu = 0;
for(int i=0; i<dim; i++) accu = accu + v[i];
return  accu/dim;
}
```

Specializzazione e creazione istanze

- Il C++ è un linguaggio fortemente tipizzato: tutto deve avere un tipo al momento della compilazione.
- Le funzioni template lasciano il tipo parametrico: sono MODELLI e non possono essere compilati.
- La specializzazione di un modello, in cui tutti i parametri di tipo sono stati specificati, può essere compilata.
- Il "compilatore" C++ riesce, nella parte di preprocessing, ad inferire i tipi da assegnare ai parametri di tipo. In pratica, è come se il compilatore stesso facesse copie del modello sostituendo ad ogni occorrenza del nome del parametro di tipo il tipo inferito.
- Terminata questa fase tutte le specializzazioni delle (dichiarazioni/definizioni) delle funzioni sono state istanziate (copia-incolla modello e specializzazione tipo) e possono essere compilate come normali funzioni.
- Verifica comprensione:

```
//Dichiarazione

template <typename T>

void scambia(T *, T *);
```

```
//Definizione

template <typename G>

void scambia(G *a, G *b){

G appo; ....

}
```

Esempio

- Il "compilatore" C++ riesce, nella parte di preprocessing, ad inferire i tipi da assegnare ai parametri di tipo. In pratica, è come se il compilatore stesso facesse copie del modello sostituendo ad ogni occorrenza del nome del parametro di tipo il tipo inferito.

```
template <typename T>
void scambia(T *, T *);

template <typename T>
void scambia(T *a, T *b){
T appo;
appo = *a;
*a = *b;
*b = appo;
}
```

```
int main(){
   int a=1,b=2;
   float c=3.0f, d = 4.0f;

   scambia(&a,&b);
   scambia(&c,&d);
   ...}
```

```
void scambia(int *a, int *b){
int appo;
appo = *a;
*a = *b;
*b = appo;
}
```

```
void scambia(float *a, float *b){
float appo;
appo = *a;
*a = *b;
*b = appo;
}
```

Parte 2:
Concepts e Varia

Alcune osservazioni.

La possibilità di delegare al compilatore la specializzazione di modelli è molto affascinante, ma attenzione che comporta un certo grado di complessità aggiuntiva, e quindi va usata con metodo e cautela.

Alcuni spunti di riflessione/attenzione:

- Compilazione separata: non è possibile separare interfaccia (dichiarazioni) da implementazione (definizioni). vedi **libTemp.hpp**
- Ricordiamo che il tipo di ritorno di una funzione non può essere inferito (due funzioni "normali" non possono differire nella *signature* solo per il tipo di ritorno). I template aggirano questa regola, ma perché usano regole di inferenza diverse. Vedi **esempio2.C**
- Stiamo attenti al significato dell'aggettivo "generico" di "programmazione generica": il modello di una funzione può essere usato per tutti i tipi per cui le operazioni usate nella funzione hanno senso. Vedi **esempio3.C**