



UNIVERSITÀ
degli STUDI
di CATANIA

DIPARTIMENTO DI
MATEMATICA E INFORMATICA

Template in C++

Alessandro Ortis

alessandro.ortis@unict.it

www.dmi.unict.it/ortis/

Programmazione generica

Si tratta di uno stile di programmazione in cui gli algoritmi e le strutture dati sono scritti in termini di tipi che verranno specificati successivamente quando il codice viene effettivamente usato.

In C++, il template implementa il concetto di *tipo parametrizzato*: il tipo viene specificato come se fosse un parametro.

I templates permettono di implementare strutture (classi) e algoritmi (funzioni) indipendentemente dal tipo di oggetti su cui operano.

Template

I template consentono di generare versione multiple del codice tramite la parametrizzazione dei tipi, e allo stesso tempo mantenere la tipizzazione statica del codice, cioè i tipi vengono determinati in fase di compilazione.

Abbiamo due tipi di template:

- **Funzioni template:** funzioni in grado di operare su tipi generici.
- **Classi template:** classi che hanno membri che usano i parametri del template come tipi.

Sintassi:

```
template <typename T>  
template <typename T, typename U>  
template <typename T, int U>
```

Funzioni template

```
template <typename T>  
T massimo (T x, T y) {  
    return (x > y) ? x : y;  
}
```

Perché il codice della funzione sia compilato, il segnaposto T deve essere sostituito con un tipo. Il compilatore deve istanziare una versione del codice che sia specializzata per questo tipo.

Funzioni template

```
template <typename T>
T massimo (T x, T y) {
    return (x > y) ? x : y;
}
```

Se è possibile inferire il tipo T dal contesto in uso il compilatore provvederà automaticamente alla specializzazione, altrimenti è necessario fornire il tipo al momento dell'invocazione della funzione.

```
int main()
{
    cout << massimo<int>(4, 6) << endl;
    cout << massimo<char>('a', 'c') << endl;
}
```

Funzioni template

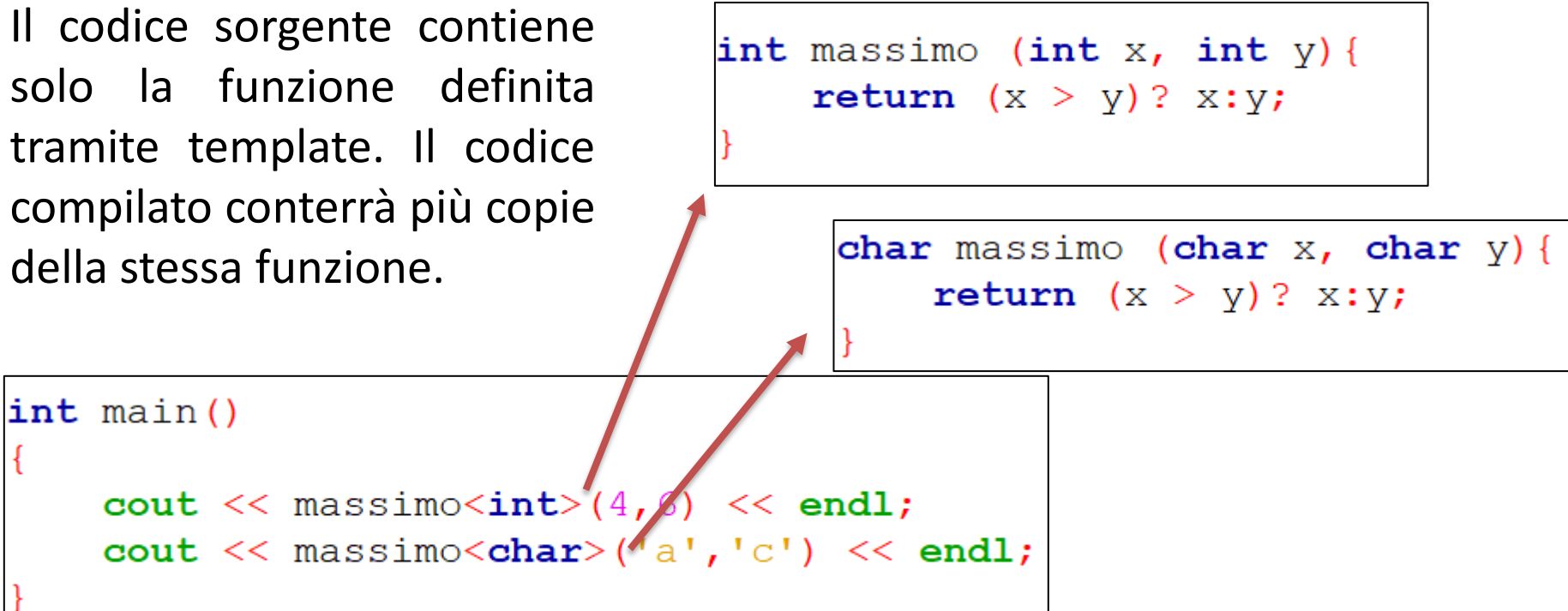
```
template <typename T>
T massimo (T x, T y) {
    return (x > y)? x:y;
}
```

Il codice sorgente contiene solo la funzione definita tramite template. Il codice compilato conterrà più copie della stessa funzione.

```
int massimo (int x, int y) {
    return (x > y)? x:y;
}
```

```
char massimo (char x, char y) {
    return (x > y)? x:y;
}
```

```
int main()
{
    cout << massimo<int>(4, 8) << endl;
    cout << massimo<char>('a', 'c') << endl;
}
```



Classi template

```
template<typename T, typename U>
class MyPair{
public:
    MyPair(T first, U second){
        a = first;
        b = second;
    }
private:
    T a;
    U b;
};

int main(){
    MyPair<int, char> p(10, 'X');
    MyPair<int, int> p2(10, 5);
}
```

Il compilatore genera una classe diversa per ogni tipo per cui è richiesta la specializzazione.

Esempio: classe Contenitore

```
template <typename T>
class Contenitore{
    T dati[50];
    int elementi;
public:
    Contenitore(): elementi(0) {}
    void inserire(T el);
    T estrarre();
    int numero(); //Numero attuale di elementi
    bool vuoto();
};
```

```
Contenitore <int> interi;
Contenitore <float> reali;
```


Template

Possiamo inoltre:

- Forzare la specializzazione per un tipo ridefinendo il corpo della funzione per quel tipo
- Specificare dei valori di default per i parametri template

```
template <typename T>
void func(T x) {
    ...
}
// solo per T=int
template <>
void func <int>(int x) {
    ...
}
```

```
template <typename T=int, int N=5>
class MyClass{
    T arr[N];
}
```

Esercizio

Implementare un algoritmo di ordinamento tramite un metodo

```
void ordina(T a[], int n)
```

che prende in input un array di elementi di tipo T di lunghezza n e li ordina.

Definire inoltre una semplice classe Rettangolo.

Nel corpo del metodo main() istanziare:

- un array di interi
- un array di char
- un array di rettangoli

applicare l'algoritmo di ordinamento a tutti e tre gli array facendo in modo che gli oggetti di tipo Rettangolo vengano confrontati in base al valore della loro area.

Per verificare il funzionamento del codice, stampare il contenuto degli array prima e dopo l'ordinamento.

Esercizio

Implementare un metodo

```
int ricerca(T dato, T v[], int dim)
```

che prende in input un oggetto di tipo T, un array di elementi tipo T di lunghezza dim e cerca il primo elemento all'interno dell'array, restituendo l'indice se presente. Se l'elemento non è presente il metodo restituisce il valore -1. Definire inoltre una semplice classe Punto.

Nel corpo del metodo main() istanziare:

- un array di interi
- un array di char
- un array di Punto

Permettere all'utente di inserire dei valori per ciascuno dei tre tipi elencati sopra ed applicare l'algoritmo di ricerca degli elementi inseriti dall'utente.

Per verificare il funzionamento del codice, stampare il contenuto degli array.