

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Algoritmi con predicati

Stefano Ghidoni



Agenda

- Ricerca con predicato: `find_if()`
- Predicati
- Function object

`find_if()`



find_if()

- `find()` confronta valori
- `find_if()` verifica che un *criterio* sia soddisfatto

```
template<typename In, typename Pred>
    // In: iteratore di input
    // Pred è un predicato
In find_if(In first, In last, Pred pred)
{
    while(first != last && !pred(*first)) ++first;
    return first;
}
```



Predicati

- Il criterio è espresso da un predicato
- Un predicato è una funzione che ritorna true o false
 - Esempio: verifica che un valore sia dispari

```
bool odd(int x) { return x%2; }
```



Predicati

- Il predicato è una funzione passata come argomento a `find_if()`
 - La funzione è passata usando solo il suo nome
 - Aggiungendo () la funzione sarebbe chiamata

```
bool odd(int x) { return x%2; }

void f(vector<int>& v)
{
    auto p = find_if(v.begin(), v.end(), odd);
    if (p != v.end()) { /* numero dispari */ }
}
```

Quindi sto passando due iteratori e una funzione, che poi `find_if()` potrà chiamare



Puntatore a funzione

- Il nome di una funzione senza () è il *puntatore a quella funzione*
 - Deriva dal C
 - È possibile ricavare il puntatore di qualsiasi funzione

Il puntatore non è solo il puntatore a una funzione, ma la caratterizzazione del puntatore comprende anche il tipo restituito e quelli di input



Passaggio di valori a un criterio

- Prendiamo in considerazione il criterio "maggiore di un dato valore"
 - Il valore è un parametro che deve essere passato alla funzione
 - `odd()` non ne aveva bisogno
- Come possiamo passare un valore a un predicato?
 - Non possiamo usare argomenti – non esistono () nel puntatore a funzione



Passaggio tramite variabile globale

```
double v_val;  
bool larger_than_v(double x) { return x>v_val; }  
  
void f(list<double>& v, int x)  
{  
    v_val = 31;          // scrittura sulla variabile globale!  
    auto p = find_if(v.begin(), v.end(), larger_than_v);  
    if (p != v.end()) { /* valore trovato */ }  
  
    v_val = x;  
    auto q = find_if(v.begin(), v.end(), larger_than_v);  
    if (q != v.end()) { /* valore trovato */ }  
  
    // ...  
}
```



Passaggio tramite variabile globale

- Il codice visto prima ha **gravi problemi**
 - Uso di una variabile globale
 - È difficile tenere "sotto controllo" il valore: è modificabile ovunque
 - Nessun encapsulamento

Function object



Function object

- Un function object è un oggetto che può essere usato come una funzione
- Ci permette di creare un predicato

```
class Larger_than {  
    int v;  
public:  
    // salvataggio argomento  
    Larger_than(int vv) : v{vv} {}  
    // confronto  
    bool operator() (int x) const { return x>v; }  
};
```



Function object

- Utilizzabile come una funzione grazie all'overloading di **operator()**
 - Chiamato **function call operator** o **application operator**

```
class Larger_than {  
    int v;  
public:  
    // salvataggio argomento  
    Larger_than(int vv) : v{vv} {}  
    // confronto  
    bool operator() (int x) const { return x>v; }  
};
```



Function object

- Utilizzabile come una funzione grazie all'overloading di **operator()**
 - Chiamato **function call operator** o **application operator**
 - Osservazione: `find_if` effettua una chiamata di questo tipo:

```
while(first != last && !pred(*first)) ++first;
```

- È tradotta in una chiamata a `operator()`

Quindi è passabile come predicato



Criterio con function object

- La funzione f() vista prima ora diventa:

```
void f(list<double>& v, int x)
{
    auto p = find_if(v.begin(), v.end(), Larger_than(31));
    if (p != v.end()) { /* valore trovato */ }

    auto q = find_if(v.begin(), v.end(), Larger_than(x));
    if (q != v.end()) { /* valore trovato */ }

    // ...
}
```

Sono effettivamente costruttori
della classe Larger_than

Creazione di oggetti e
salvataggio soglia nel
dato membro



find_if() e function object

- Il funzionamento dei function object è il seguente:
 - Function object creato nell'argomento
 - Oggetto passato a find_if() per copia
 - find_if(), al suo interno, chiama operator()
- Tecnica che permette di fornire dati a una funzione

```
auto p = find_if(v.begin(), v.end(), Larger_than(31));
```



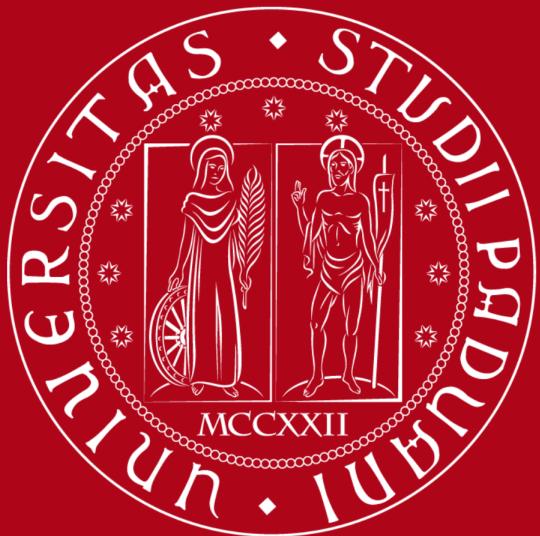
Function object ed efficienza

- Usare i function object è una tecnica efficiente
- Per una migliore efficienza:
 - Piccoli function object passati **per copia**
 - Se più grandi di qualche byte, meglio per riferimento
- **operator() inline**
- Questa combinazione è più efficiente di una chiamata a funzione!



Recap

- Combinazione di un algoritmo con un predicato: `find_if()`
- Esprimere un predicato
 - Con un puntatore a funzione
 - Con un function object



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Algoritmi con predicati

Stefano Ghidoni



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE