Corso di Linguaggi di Programmazione

Lezione 19

Alberto Ceselli
alberto.ceselli@unimi.it

Dipartimento di Informatica
Università degli Studi di Milano

14 Maggio 2013

Generic Programming

Idea

- Programma = Algoritmi + Strutture dati (Wirth)
- idea: rendere l'algoritmo indipendente dai dati a cui è applicato
- Idea: combinare tipi parametrici e overloading

Codice generico in C

- Il sogno di avere codice generico non è nuovo ...

```
#define max(a,b) ( (a) > (b)  ? (a) : (b) )
```

- codice generico tramite definizione di 'macro'

Scansione di una lista

```
struct Element // This is an extremely simplified definition,
{              // but enough for this example.
   int value;
   struct Element * next;
};

int high = list->value; // list points to the first element

struct Element * current = list->next;
                        // refers (points) to second element

while (current != NULL) // test if within the group of elements
{
   if (current->value > high)
   {
      high = current->value;
   }
   current = current->next; // Advance to next element
}
}
```

Scansione di un array

```
int high = *array;
int * one_past_end = array + size;
int * current = array + 1;   // starts at second element

while (current != one_past_end) // test if within the group of elements
{
   if (*current > high)
   {
      high = *current;
   }
   current++;     // Advance to the next element
}
```

Containers e Iterators

- **Container:** una classe che contiene una qualsiasi collezione di oggetti (es. vector, list, hash table ...)
- **Iterator:** una funzione che scandisce questa collezione.
- La **Standard Template Library** di C++ è basata, appunto su templates, containers ed iterators.

Scansione di una lista **o** di un array

```
// list<int>::iterator current = values.begin();
vector<int>::iterator current = values.begin();
int high = *current++;

while (current != values.end())
{
   if (*current > high)
   {
      high = *current;
   }
   current++;
}
```

C#

- Containers: oggetti che implementano `IEnumerable`
- Iterators: costrutto `foreach .. in`
- Esempio:

```
String[] strList = {``Alberto'', ``Chiara'', ``Roberto'', ``Valentina''};
...
foreach (String name in strList)
  Console.WriteLine(``Nome: {0}'', name)
```

Java

- Containers: chiamati *Collection* in Java

```
List<Integer> mylist = new ArrayList<Integer>();
```

- Iterators simili a C++

```
void cancelAll(Collection<TimerTask> c) {
       for (Iterator<TimerTask> i = c.iterator(); i.hasNext(); )
            i.next().cancel();
}
```

- Iterators simili a C#

```
void cancelAll(Collection<TimerTask> c) {
       for (TimerTask t : c)
     t.cancel();
}
```

Python

- Containers: oggetti che implementano

```
__len__(self)
__getitem__(self, key)
__setitem__(self, key, value)
__delitem__(self, key)
__contains__(self, item)
```

- Iterators: restituiti implementando un metodo `__iter__()` in un container.
- Utilizzo:

```
>>> t = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> count = 0
>>> for num in t:
...     count += num
... else:
...     print count
...
45
```

Iterators in Python

Esempio: iterator per classe stack

Type upper bounds

- tipi parametrici e meccanismo di tipi e sottotipi possono essere combinati:

```
public class Tree<V extends Comparable> ...
```

- ... anche in modo più esotico

```
public class Tree<V extends Serializable & Comparable<V>> ...
```

- Java permette anche l'utilizzo di wildcards '?':

```
public static double sum(Tree<? extends Number> t) {
    ...
}
```

Reflection

Cosa si intende per *reflection*?
La capacità di un programma di

(1) ispezionare lo stato / comportamento

(2) modificare lo stato / comportamento

...

(a) di se stesso (structural reflection)

(b) dell'interprete che lo sta eseguendo (behavioural reflection)

Reflection

Analizzando le 4 combinazioni:

- 1a Leggere la lista delle proprietà di un oggetto
- 2a modificare il valore di una proprietà, invocare metodi basandosi in modo "riflessivo"
- 1b Ispezionare lo stack frame corrente per capire qual'è il metodo attualmente in esecuzione
- 2b Modificare lo stack o il modo in cui alcune operazioni vengono eseguite

Esempi di utilizzo: **testing** e serializzazione.

Reification

Letteralmente "rendere concreto qualcosa di astratto". Esempio:

- un interprete trova una dichiarazione di tipo, o la definizione di un metodo, e la rappresenta in memoria con strutture opportune ...
- ho ottenuto il *nome* di un metodo (reflection), ora lo *chiamo* (reification).

Esempi: Java

Esempio di reflection in Java:

```
Method method =
foo.getClass().getMethod("doSomething", null);
method.invoke(foo, null);
```

Esempi: python

```
# without reflection
obj = Foo()
obj.hello()

 # with reflection
 class_name = "Foo"
 method = "hello"
 obj = globals()[class_name]()
 getattr(obj, method)()

  # with eval
  eval("Foo().hello()")
```

Riassumendo

- Approccio procedurale: "Decidi quali procedure vuoi, ed utilizza i migliori algoritmi tu possa trovare".
- Approccio modulare: "Decidi quali moduli vuoi, e partiziona il programma in modo che i dati siano nascosti nei moduli".
- Approccio ad oggetti: "Decidi quali classi di oggetti vuoi, fornisci un set completo di operazioni per ogni classe, e rendi esplicite le parti comuni utilizzando l'ereditarietà"
- Approccio generic programming: "Decidi quali algoritmi vuoi, e parametrizzali per il maggior numero possibile di tipi e strutture dati".

(B. Stroustrup)

Esercizi:

Esercizi:

- Realizzare un ADT "Buffer Circolare FIFO" (interfaccia, implementazione e codice client) (Lez 8 e 9)
- Rendere parametrico l'ADT rispetto al tipo di dati memorizzato nel Buffer (Lez 9)
- Estendere il "Buffer Circolare" in un "Buffer Circolare *Collegabile*", ovvero un buffer che può essere riempito con elementi di un altro buffer.