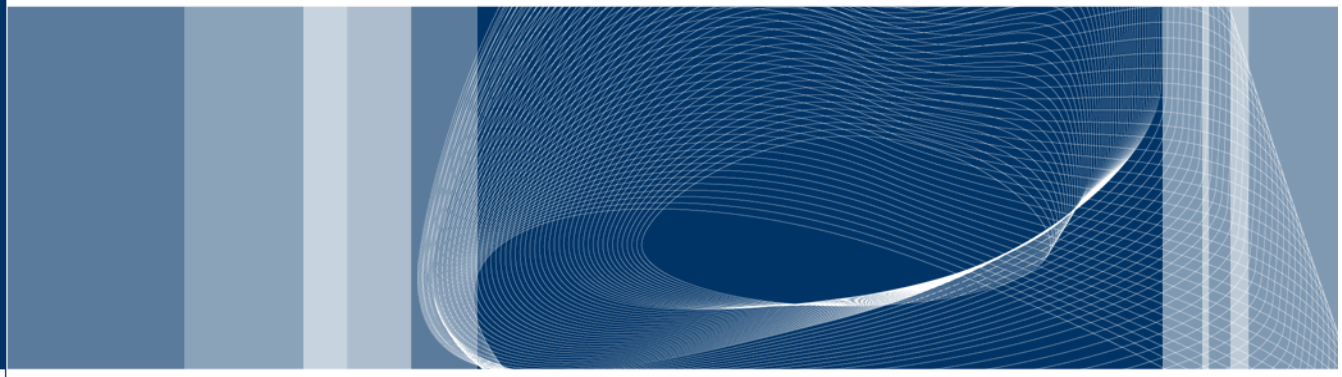
POLITECNICO DI MILANO



Algoritmi e Strutture Dati

Laboratorio 20/10/2008

Prof. Pier Luca Lanzi

Primo Esercizio

- Scrivere un programma per misurare il tempo necessario per ordinare un vettore di interi contenente 10-10^7 elementi utilizzando l'insertion sort e il merge sort

- Il programma deve generare un file con tre colonne: la prima contiene il numero di dati, la seconda il tempo necessario per l'insertion sort e la terza contiene il tempo necessario per il merge sort

Secondo Esercizio

- Scrivere il template di una classe che implementi una coda derivata dalla seguente classe virtuale astratta

```
template <class Elem> class CodaBase {
public:
   virtual bool enque(Elem) = 0;
   virtual bool deque() = 0;
   virtual Elem head() = 0;
   virtual bool isempty() = 0;
};
```

Secondo Esercizio

- Usare la classe coda per eseguire il seguente codice

```
Coda<int> c;

for(i=0;i<10;i++) c.enqueue(rand());

for(i=0;i<10;i++) {
  cout << c.head();
  c.dequeue();
}
```

Terzo Esercizio

- Creare un template di una classe CodaConPriorita derivata dal template della classe Coda in cui l'inserimento di un elemento in coda dipende anche da un valore di priorita'.

Classi C++


```
class NomeClasse {
private:
  // attributi e metodi visibili solo all'interno
  // della classe

public:
  // attributi e metodi visibili da tutti

...
};
```

Template

- Parametrizzano classi e funzioni rispetto a uno o piu' parametri
- Funzioni

```
void sort(int a[], int n)
```

diventa

```
template <typename T>
void sort(T a[], int n)
```

- Il meccanismo dei template in pratica dichiara delle macro, che poi vengono istanziate quando specifico il tipo

```
double b[10]; sort(b, 10);
string c[20]; sort(c,20);
```

Classi C++ & Template

```
template <typename T>
class NomeClasse {
private:
  // attributi e metodi visibili solo all'interno
  // della classe
  T ...
public:
  // attributi e metodi visibili da tutti

...
};
```

Classi C++ & Template (parametri)


```
template <typename T, size_t SIZE>
  class Stack {
  public:
    void push (const T& new_item);
    T top () const;
    void pop ();
    bool isEmpty () const;
    bool isFull () const;
  private:
    size_t top_;
    T stack_[SIZE];
  };
```

Ereditarietà

- Nuove classi sono create/derivate a partire da classi preesistenti
- La classe derivata eredita gli attributi e i metodi della classe definita precedentemente
- La classe derivata contiene parte della superclasse
- È possibile avere ereditarietà multipla o singola

Ereditarietà

- Tre tipi di ereditarietà in C++
  - public: i mebri ereditati rimangono pubblici
  - private: i membri ereditati diventano private
  - protected: ...

- protected: altro tipo di attributi e metodi accessibili solo dalle classi derivate

Classi C++


```
class NomeClasse {
private:
  // attributi e metodi visibili solo all'interno
  // della classe

public:
  // attributi e metodi visibili da tutti

protected:
  // attributi e metodi visibili dalle sottoclassi
};
```