Laboratorio di programmazione

Lezione IX

Tatiana Zolo

zolo@cs.unipr.it

TIPI STRINGA

Il C++ fornisce due rappresentazioni: stringa di caratteri stile C e la classe stringa introdotta nel C++ standard.

TIPI STRINGA

Il C++ fornisce due rappresentazioni: stringa di caratteri stile C e la classe stringa introdotta nel C++ standard.

Stringa di caratteri stile C: la stringa è memorizzata in un array di caratteri e generalmente manipolata attraverso un puntatore `char*`.

```
char str1[32];
// ‘str2’ punta al primo elemento
// dell’array di caratteri.
char* str2;
```

Vale tutto quanto abbiamo visto (es. `strlen()`, `strcmp()`).

Si utilizza l’aritmetica dei puntatori... provoca facilmente dei problemi!

TIPI STRINGA

Tipo string: cosa ci si aspetterebbe da un oggetto “stringa”? Quali operazioni dovrebbe supportare una classe “stringa”?

- Inizializzazione (con sequenza di caratteri oppure con un altro oggetto stringa).
- Copia (con le stringhe C si usa `strcpy()`).
- Accesso in lettura e scrittura ai singoli caratteri (con le stringhe C si usa l’operatore `[]` o la deferenziazione dei puntatori).
- Confronto fra due stringhe per l’uguaglianza (con le stringhe C si usa `strcmp()`).
- Accodamento di due stringhe, sia concatenando una stringa all’altra sia combinando le due stringhe per formarne una terza (con le stringhe C si usa `strcat()`, eventualmente combinato con `strcpy()`).
- Possibilità di sapere quanti caratteri compongono la stringa (con le stringhe C si usa `strlen()`).
- Possibilità di sapere se una stringa è vuota.

TIPI STRINGA

La classe string supporta tutte queste operazioni e molte altre (occorre includere il file header associato `#include <string>`).

Costruiamo oggetti stringa:

```
string str1; // stringa vuota.
string str2(''ciao!'');
string str3(str2);
```

Vi sono diverse funzioni (proprie) da utilizzarsi su oggetti string: `size()`, `empty()`, `c_str()`, ...

Sono inoltre ridefiniti molti operatori: `+`, `+=`, ...

es. `cl_string.cpp`.

Gestione delle eccezioni

Eccezioni: anomalie di esecuzione rilevabili dal programma, per esempio

- divisione per 0;
- accesso ad un'array all'esterno dei suoi limiti;
- esaurimento della memoria disponibile;
- ...

Quando in un programma si incontra un'eccezione, la parte del programma che la rileva può comunicare che essa si è verificata *sollevando*, o *lanciando*, un'eccezione.

GESTIONE DELLE ECCEZIONI

```
try {
  // codice che si vuole controllare
  throw exception;
}
catch(type_1 exception) {
  // trattamento di eccezioni di tipo ‘type_1’.
}
catch(type_2 exception) {
  // trattamento di eccezioni di tipo ‘type_2’.
}
```

- Blocco try: deve racchiudere tutte le istruzioni che possono lanciare eccezioni. Introduce un campo d’azione locale.
- Espressione throw: generatore di eccezioni. L’espressione che segue questa parola chiave ha come tipo quello dell’espressione lanciata.
- Clausola catch: procedura di gestione di un’eccezione.

GESTIONE DELLE ECCEZIONI

Esempio: costruttore della classe ‘Razionale’

```
Razionale::Razionale(int num, int den = 1) {
  n = num;
  if (den == 0)
    throw "il denominatore non puo’ essere nullo!";
  d = den;
}
```

Ci sono due possibilità:

1. l’utilizzo del costruttore si trova nel blocco *try*;
2. l’intero corpo della funzione è contenuto nel blocco *try* (*blocco try di funzione*).

```
1. int main() {
   try {
     Razionale r(2,0);
     ...
   }
   catch(const char* str) {
     cerr << "Eccezione: " << str << endl;
   }

     return 0;
   }
```

2. *blocco try di funzione*:

```
int main() try {
  Razionale r(2,0);
  ...
  return 0;
}
catch(char* str) {
  cerr << "Eccezione: " << str << endl;
}
```

Metodo migliore: separazione più netta fra il codice che implementa l'elaborazione normale del programma e quello che supporta la gestione delle eccezioni.

Flusso di controllo del programma:

- Non si verifica alcuna eccezione: il codice nel blocco *try* viene eseguito e le procedure di gestione associate ad esso (clausole *catch*) sono ignorate.
- Viene lanciata una eccezione: relativamente all'esempio di prima se il costruttore lancia un'eccezione il costruttore stesso ed il blocco *try* terminano e viene eseguita la procedura di gestione per le eccezioni di tipo `char*`.
- Prosecuzione del programma dopo l'eccezione: se le clausole *catch* non contengono un'istruzione return l'esecuzione del programma, una volta completato il lavoro nella clausola *catch*, riprende dall'istruzione che segue l'ultima clausola *catch*.
- Non esiste nessuna clausola *catch* in grado di gestire l'eccezione: l'esecuzione del programma riprende dalla funzione `terminate()` definita nella libreria standard del C++, che di default chiama `abort()`.

catch può essere sovraccaricato: le clausole sono esaminate una alla volta nell'ordine in cui appaiono.

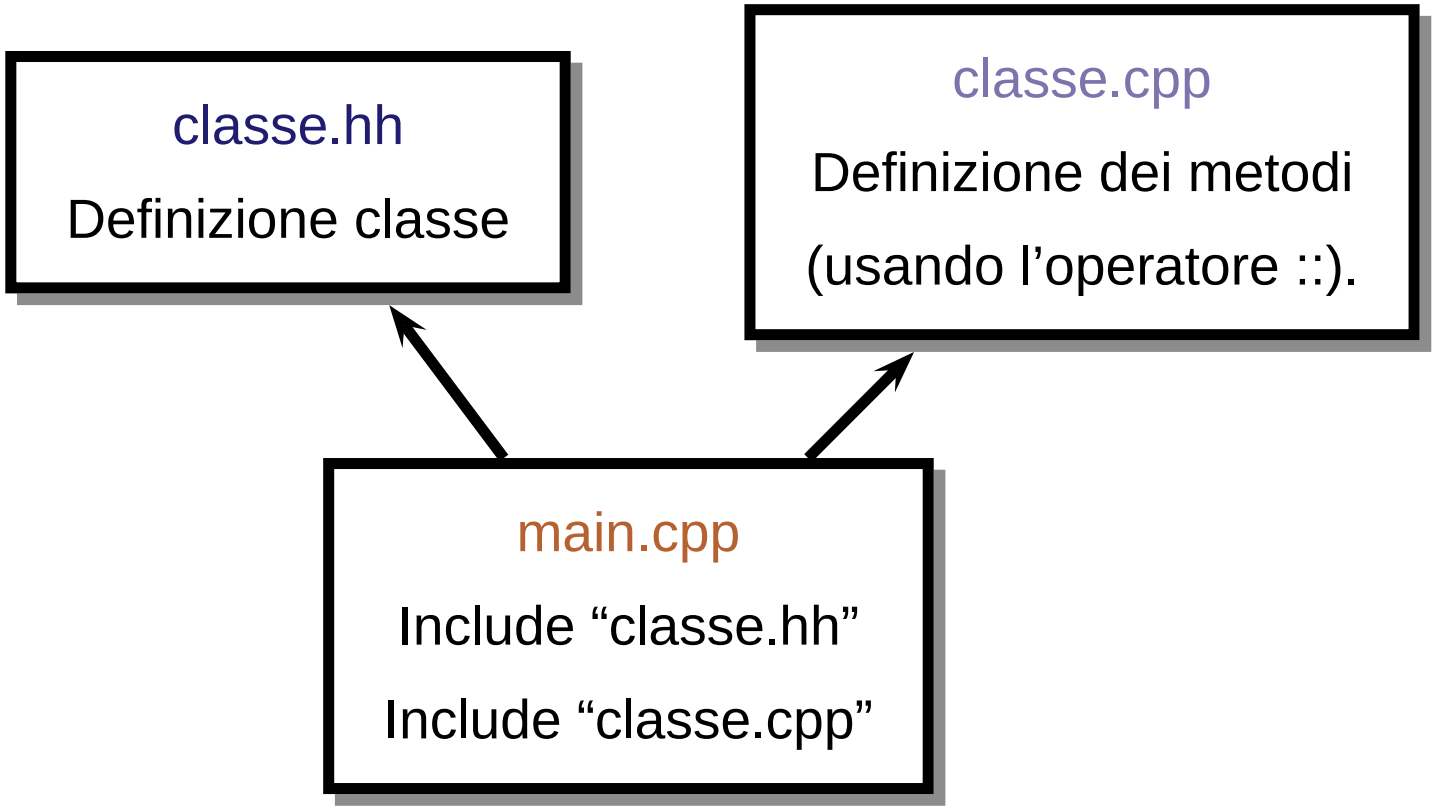
```
try {
  ...
     throw exception;
  ...
}
catch(const char* str) {
  ... }
catch(int x) {
  ... }
catch(...) {  // ‘‘catch-all’’, clausola catch generica:
  ... }       // valida per ogni tipo di eccezione.
```

Classe stdexcept: contiene un insieme di eccezioni standard (es. `range_error`, `out_of_range`, `logic_error`, `...`).

```
#include<stdexcept>
#include<iostream>
#include<typeinfo>
int main() try {
  ...
  throw std::invalid_argument("Sintassi errata!");
}
catch (const std::exception& e) {
  cerr << "std::exception caught: " << e.what()
       << " (type == " << typeid(e).name() << ")" << endl;
  exit(1); }
```

(es. `str_num_exception.cpp`).

STRUTTURA DI UN PROGRAMMA: SENZA MAKEFILE



Esercizi

1. Riprendere l’esercizio sulla classe “Razionali” della lezione scorsa: separarlo in più files ed aggiungere la gestione delle eccezioni.
2. Riprendere l’esercizio sulla classe “Tempo” della lezione scorsa: separarlo in più files ed aggiungere la gestione delle eccezioni.