Corso “Programmazione 1”
Capitolo 05: Le Funzioni

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Giovanni De Toni - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: https://bit.ly/2VgfYwJ

Ultimo aggiornamento: 13 ottobre 2021

Terms of Use and Copyright

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Funzioni ricorsive

- In C++ una funzione può invocare se stessa (funzione ricorsiva)
- ... o due o più funzioni possono chiamarsi a vicenda (funzioni mutualmente ricorsive)
- Formulare alcuni problemi in maniera ricorsiva risulta naturale:
  - il fattoriale: $0! \stackrel{\text{def}}{=} 1; n! \stackrel{\text{def}}{=} n \cdot (n-1)!$
  - pari/dispari: $even(n) \iff odd(n-1)$; $odd(n) \iff even(n-1)$;
  - espressioni: $somma \stackrel{\text{def}}{=} numero; somma \stackrel{\text{def}}{=} (somma + somma)$
- Due componenti:
  - una o più condizioni di terminazione
  - una o più chiamate ricorsive
- Intrinseco rischio di produrre sequenze infinite
  - Analoghe considerazioni rispetto ai cicli
- Alcune “insidie” computazionali
  - Es: funzione di Fibonacci: $f_0 \stackrel{\text{def}}{=} 1; f_1 \stackrel{\text{def}}{=} 1; f_n \stackrel{\text{def}}{=} f_{n-1} + f_{n-2}$

Ricorsione fortemente collegata al principio di induzione matematico.

Esempi

- fattoriale:
  { `FUNZIONI_RICORSIVE/fact_nocomment.cc` }
- ..., con chiamate tracciate:
  { `FUNZIONI_RICORSIVE/fact.cc` }
- ..., errore (loop infinito) :
  { `FUNZIONI_RICORSIVE/fact_infloop.cc` }
- ..., stack tracciato :
  { `FUNZIONI_RICORSIVE/fact_stack.cc` }
- funzioni mutualmente ricorsive:
  { `FUNZIONI_RICORSIVE/pariDispari.cc` }
- Fibonacci:
  { `FUNZIONI_RICORSIVE/fibonacci_nocomment.cc` }
- ..., con chiamate tracciate:
  { `FUNZIONI_RICORSIVE/fibonacci.cc` }
- versione iterativa:
  { `FUNZIONI_RICORSIVE/fibonacci_iterativa.cc` }

Nota sulla ricorsione

La realizzazione ricorsiva di una funzione può richiedere due funzioni:

- una funzione ausiliaria ricorsiva, con un parametro di ricorsione aggiuntivo (simile a contatore in loop)
- una funzione principale (wrapper) che chiama la funzione ricorsiva con un valore base del parametro di ricorsione
- situazione molto frequente nell'uso di array (prossimo capitolo)

- Esempio di funz. ricorsiva che necessita wrapper:
  { `FUNZIONI_RICORSIVE/stampanumeri.cc` }
- ... variante 1:
  { `FUNZIONI_RICORSIVE/stampanumeri1.cc` }
- ... variante 2:
  { `FUNZIONI_RICORSIVE/stampanumeri2.cc` }
- analoga variante del fattoriale, con wrapper:
  { `FUNZIONI_RICORSIVE/fact_rec1.cc` }

Ricorsione vs. Iterazione

- Ricorsione spesso più naturale, semplice ed elegante
- Efficienza della ricorsione critica:
  - Attenzione a chiamate identiche in rami diversi! (es. Fibonacci)
    $\Longrightarrow$ rischio esplosione combinatoria
  - Dimensione dello stack dipende dalla profondità di ricorsione
    $\Longrightarrow$ notevole overhead e spreco di memoria
    $\Longrightarrow$ quando possibile, tipicamente iterazione più efficiente
    $\Longrightarrow$ passando oggetti "grossi", è indispensabile usare passaggio per riferimento o puntatore
- Molte funzioni ricorsive possono essere riscritte in forma iterativa:
  - tail recursion: una chiamata ricorsiva, operata come ultimo passo
    - Es: somma, pari/dispari, ...
  - in generale, quando non comporta una "biforcazione"
    - Es: fattoriale, Fibonacci, ...
  - `g++ -O2` effettua una conversione da tail-recursive in iterative

Da ricorsione in coda a iterazione (caso void)

```
void F(int x,...) {
 if (CasoBase(x,...))
  {IstrBase(...);}
 else {
  Istr(...);
  x=agg(x,...);
  F(x,...);
}}
```

$\Longleftrightarrow$

```
void F(int x,...) {
 while (!CasoBase(x,...)){
  Istr(...);
  x=agg(x,...);
 }
 {IstrBase(...);}
}
```

- Esempio funzione void tail-recursive :
  { FUNZIONI_RICORSIVE/stampanumeri3.cc }
- ... corrispondente versione iterativa :
  { FUNZIONI_RICORSIVE/stampanumeri3_while.cc }

Da ricorsione in coda a iterazione (caso generale)

```
type F(int x,...) {
 if (CasoBase(x,...))
  res = IstrBase(...);
 else {
  Istr(...);
  x=agg(x,...);
  res = F(x,...);
 }
 return res;
}
```

$\Longleftrightarrow$

```
type F(int x,...) {
 while (!CasoBase(x,...)){
  Istr(...);
  x=agg(x,...);
 }
 res = IstrBase(...);
 return res;
}
```

- Esempio funzione tail-recursive:
  { FUNZIONI_RICORSIVE/sum.cc }
- ... corrispondente versione iterativa:
  { FUNZIONI_RICORSIVE/sum_while.cc }
- Compilazione di funzioni tail-recursive in iterative:
  { FUNZIONI_RICORSIVE/tailrecursive-comp.cc }

Ricorsione vs. Iterazione II

- ...
- Talvolta non è agevole riscrivere la ricorsione in forma iterativa
  - funzioni non-tail recursive, chiamate multiple
  - Es: manipolazione di espressioni

  Esempio di gestione di espressioni:
  { `FUNZIONI_RICORSIVE/espressione.cc` }
- In generale, convertire una funzione ricorsiva in iterativa richiede l'uso di uno stack

Esercizi proposti

Vedere file `ESERCIZI_PROPOSTI.txt`