Programmazione 2 e Lab. di programmazione 2

Corso di Laurea in Informatica - Anno Accademico 2022-23

Docenti
Prof. Angelo Ciaramella
[angelo.ciaramella@uniparthenope.it]
Prof. Luigi Catuogno
[luigi.catuogno@uniparthenope.it]
Tutor
Dott. Antonio Vanzanella
[antonio.vanzanella@studenti.uniparthenope.it]

1

Descrizione del Corso

Libro di testo

[FdP]

H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

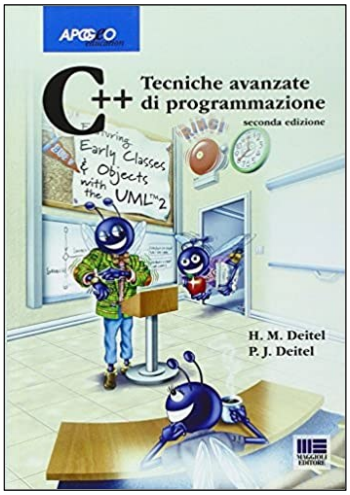
2

Descrizione del Corso

Libro di testo H. M. Deitel, P. J. Deitel

[TAP] C++ Tecniche avanzate di programmazione

II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



3

Orari e modalità di ricevimento studenti

Docenti:

Prof. Angelo Ciaramella	Martedì dalle 14:00 alle 16:00 - telematico (codice Teams r3p3w0z)
Prof. Luigi Catuogno	Giovedì dalle 11:00 alle 13:00 - telematico (codice Teams)

Tutor:

Dott. Antonio Vanzanella	Martedì dalle 11:00 alle 13:00 - telematico (codice Teams 92dbag0)

5

Il Linguaggio C++

(per programmatori C)

Parte prima

6

Puntatori e riferimenti

7

Funzioni: passaggio dei paramteri

8

Funzioni: passaggio di parametri

Consideriamo la seguente definizione della funzione, **swap** che, presi due parametri, ne scambia il valore.

```
void swap (int x, int y)
{
      int t;
      t=x; x=y; y=t;
}
```

In questo codice, **x** e **y** sono i *parametri formali.*

9

Passaggio dei parametri in C++

In C++ il passaggio dei parametri ad una funzione può avvenire:

C

Per valore: il valore dei *parametri reali* è copiato nei parametri formali

Per indirizzo: la funzione chiamante trasferisce a quella invocata *i puntatori* ai parametri reali (*i.e. l'indirizzo di memoria in cui sono memorizzati*).

C++

Per riferimento: I parametri formali della funzione chiamata sono dei *riferimenti* (dei «sinonimi») dai parametri reali.

10

Passaggio dei parametri in C++

In C++ il passaggio dei parametri ad una funzione può avvenire:

Per valore: il valore dei *parametri reali* è copiato nei parametri formali;

Preferibile quando:

Ci si aspetta che la funzione chiamata non debba modificare il valore dei suoi parametri, ma solo restituire un risultato o implementare una procedura.

La taglia dei parametri è di dimensioni contenute, per cui il tempo che il sistema impiega per effettuare le copie non incide significativamente sul tempo di esecuzione.

11

Funzioni: passaggio di parametri

```
1  #include<iostream>
2  using namespace std;
3
4  void swap (int x, int y)
5  {
6          int t;
7          t=x; x=y; y=t;
8  }
9
10 int main()
11 {
12         int pr1=12, pr2=10;
13         cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << endl;
14         swap(pr1,pr2);
15         cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << endl;
16 }
```

Passaggio di parametri *per valore*

Nel codice *chiamante* la **swap** è *invocata* con le variabili **pr1** e **pr2** che sono i *parametri reali*. Nell'effettuare la chiamata alla **swap**, la **main** *copia il valore* dei parametri reali nei suoi parametri formali e le *cede il controllo*.

Esercizio: *passaggio di parametri*

```
4  void swap (int x, int y)
5  {
6          int t;
7          t=x; x=y; y=t;
           (A)
8  }
9
10 int main()
11 {
12         int pr1=12, pr2=10;
13         cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << endl;
           (B)
14         swap(pr1,pr2);
           (C)
15         cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << endl;
16 }
```

Quanto valgono **x** e **y** nel punto **A**?

Quanto valgono **pr1** e **pr2** nei punti **B** e **C** ?

Esercizio: *passaggio di parametri*

```
 4  void swap (int x, int y)
 5  {
 6          int t;
 7          t=x; x=y; y=t;
            (A)   x=10, y=12
 8  }
 9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << endl;
            (B)   pr1=12, pr2=10
14          swap(pr1,pr2);
            (C)   pr1=12, pr2=10
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << endl;
16  }
```

Quanto valgono **x** e **y** nel punto **A**?

Quanto valgono **pr1** e **pr2** nei punti **B** e **C** ?

14

Esempio: *passaggio di parametri per valore*

```
 4  void swap (int x, int y)
 5  {
 6          int t;
 7          t=x; x=y; y=t;
 8  }
 9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << e
14          swap(pr1,pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << e
16  }
```

Identificatore	valore
x (swap)	n/a
y (swap)	n/a
t (swap)	0
…	
pr1 (main)	12
pr2 (main)	10
…	

15

Esempio: *passaggio di parametri per valore*

```
4   void swap (int x, int y)
5   {
6           int t;
7           t=x; x=y; y=t;
8   }
9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14          swap(pr1,pr2);
15          ut << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

Identificatore	valore
x (swap)	12
y (swap)	10
t (swap)	0
…	
pr1 (main)	12
pr2 (main)	10
…	

Il contenuto dei registri di memoria delle variabili **pr1** e **pr2** di **main** viene copiato nei registri delle variabili **x** e **y** di **swap**

16

Esempio: *passaggio di parametri per valore*

Il la **swap** fa il suo lavoro e scambia il valore delle sue *variabili locali* **x** e **y**. Nessun valore viene restituito al chiamante, né si verifica alcun accesso alle sua variabili **pr1** e **pr2**

```
4   void swap (int x, int y)
5   {
6           int t;
7           t=x; x=y; y=t;
8   }
9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14          swap(pr1,pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

dentificatore	valore
x (swap)	10
y (swap)	12
t (swap)	12
…	
pr1 (main)	12
pr2 (main)	10
…	

17

Esempio: *passaggio di parametri per valore*

```
 4  void swap (int x, int y)
 5  {
 6          int t;
 7          t=x; x=y; y=t;
 8  }
 9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << e
14          swap(pr1,pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" <
16  }
```

Identificatore	valore
x(swap)	**10**
y(swap)	**12**
t(swap)	**12**
…	
pr1(main)	**12**
pr2(main)	**10**
…	

Il controllo torna alla **main**. Il valore delle variabili **pr1** e **pr2** è rimasto lo stesso.

18

Passaggio dei parametri in C++

In C++ il passaggio dei parametri ad una funzione può avvenire:

Per indirizzo: la funzione chiamante trasferisce a quella invocata *i puntatori* ai parametri reali (*i.e. l'indirizzo di memoria in cui sono memorizzati*).

Preferibile quando:

effetti collaterali La chiamata a funzione ha lo scopo di modificare il valore dei suoi parametri

La taglia dei parametri reali è tale da rendere inaccettabile il tempo necessario ad eseguirne la copia ad ogni invocazione.

Manipolazione di strutture dati *dinamiche*

19

Esempio: *passaggio di parametri per indirizzo*

Modifichiamo la funzione, **swap** affinché effettui lo scambio delle due variabili del *contesto chiamante*.

```
void swap (int *x, int *y)
{
      int t=0;
      t=*x; *x=*y; *y=t;
}
```

In questa versione, la swap riceve dal chiamante i puntatori alle variabili che chiede di scambiare.

20

Esempio: *passaggio di parametri per indirizzo*

```
 1  #include<iostream>
 2  using namespace std;
 3
 4  void swap (int *x, int *y)
 5  {
 6          int t=0;
 7          t=*x; *x=*y; *y=t;
 8  }
 9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14          swap(&pr1,&pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

indirizzo	contenuto
0x00f100	swap
0x00f2e0	x=n/a
0x00f2e4	y=n/a
0x00f2e8	t=0
…	
0x01f000	main
0x01f100	pr1=12
0x01f104	pr2=10
…	

21

Esempio: *passaggio di parametri per indirizzo*

```
1  #include<iostream>
2  using namespace std;
3
4  void swap (int *x, int *y)
5  {
6          int t=0;
7          t=*x; *x=*y; *y=t;
8  }
9
10 int main()
11 {
12         int pr1=12, pr2=10;
13         cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << endl;
14         swap(&pr1,&pr2);
15         cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << endl;
16 }
```

I parametri formali della funzione **swap** sono del tipo *puntatori interi.* Ci si aspetta che ricevano l'indirizzo in memoria di due variabili intere.

Nel codice chiamante la **swap** è invocata con *i puntatori alle variabili* **pr1** e **pr2.** L'operatore unario prefisso **&** applicato a una variabile ne restituisce l'indirizzo in memoria.

22

Esempio: *passaggio di parametri per indirizzo*

```
1  #include<iostream>
2  using namespace std;
3
4  void swap (int *x, int *y)
5  {
6          int t=0;
7          t=*x; *x=*y; *y=t;
8  }
9
10 int main()
11 {
12         int pr1=12, pr2=10;
13         cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14         swap(&pr1,&pr2);
15         cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16 }
```

indirizzo	contenuto
0x00f100	swap
0x00f2e0	x=0x01f100
0x00f2e4	y=0x01f104
0x00f2e8	t=0
…	
0x01f000	main
0x01f100	pr1=12
0x01f104	pr2=10
…	

23

Esempio: *passaggio di parametri per indirizzo*

L'operatore di *deferenziazione* ***** applicato a un puntatore, consente di «accedere» alla variabile puntata.

Se posto alla destra di una espressione di assegnamento *(r-valore)*, ***x** restituisce il contenuto della variabile puntata da **x** (in questo caso **pr1**)

```
3
4   void swap (int *x, int *y)
5   {
6           int t=0;
7           t=*x; *x=*y; *y=t;
8   }
9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14          swap(&pr1,&pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

indirizzo	contenuto
0x00f100	swap
0x00f2e0	x=0x01f100
0x00f2e4	y=0x01f104
0x00f2e8	t=12
…	
0x01f000	main
0x01f100	pr1=12
0x01f104	pr2=10
…	

24

Esempio: *passaggio di parametri per indirizzo*

L'operatore di *deferenziazione* ***** applicato a un puntatore, consente di «accedere» alla variabile puntata.

Se posto alla sinistra di una espressione di assegnamento *(l-valore)*, ***y** *deposita* il risultato dell'espressione nella variabile puntata da **y** (in questo caso **pr2**)

```
3
4   void swap (int *x, int *y)
5   {
6           int t=0;
7           t=*x; *x=*y; *y=t;
8   }
9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14          swap(&pr1,&pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

indirizzo	contenuto
0x00f100	swap
0x00f2e0	x=0x01f100
0x00f2e4	y=0x01f104
0x00f2e8	t=10
…	
0x01f000	main
0x01f100	pr1=10
0x01f104	pr2=12
…	

25

Passaggio dei parametri in C++

In C++ il passaggio dei parametri ad una funzione può avvenire:

Per riferimento: I parametri formali della funzione chiamata sono dei *riferimenti* (dei «sinonimi») dai parametri reali.

error-proof A differenza dei puntatori, i riferimenti forniscono un accesso più sicuro ai parametri (sebbene più limitato).

Preferibile quando:

effetti collaterali La chiamata a funzione ha lo scopo di modificare il valore dei suoi parametri

La taglia dei parametri reali è tale da rendere inaccettabile il tempo necessario ad eseguirne la copia ad ogni invocazione.

26

Esempio: *passaggio di parametri per riferimento*

Modifichiamo la funzione, **`swap`** affinché effettui lo scambio delle due variabili del *contesto chiamante*.

```
void swap (int &x, int &y)
{
      int t=0;
      t=x; x=y; y=t;
}
```

In questa versione, la swap riceve dal chiamante i *riferimenti* alle variabili che chiede di scambiare.

27

Esempio: *passaggio di parametri per riferimento*

Modifichiamo la funzione, **swap** affinché effettui lo scambio delle due variabili del *contesto chiamante*.

```
void swap (int &x, int &y)
{
      int t=0;
      t=x; x=y; y=t;
}
```

I parametri formali sono dichiarati come *riferimenti a variabili intere.*

In questa versione, la swap riceve dal chiamante i *riferimenti* alle variabili che chiede di scambiare.

28

Esempio: *passaggio di parametri per riferimento*

Modifichiamo la funzione, **swap** affinché effettui lo scambio delle due variabili del *contesto chiamante*.

```
void swap (int &x, int &y)
{
      int t=0;
      t=x; x=y; y=t;
}
```

Le variabili **x** e **y**, costituiscono dei *«nomi aggiuntivi»* delle variabili del chiamante, visibili dal contesto della funzione chiamata. Nel corpo della funzione sono utilizzate come se fossero variabili intere.

In questa versione, la swap riceve dal chiamante i *riferimenti* alle variabili che chiede di scambiare.

29

Esempio: *passaggio di parametri per riferimento*

```
 4  void swap (int &x, int &y)
 5  {
 6          int t=0;
 7          t=x; x=y; y=t;
 8  }
 9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << en
14          swap(pr1,pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

Identificatore	valore
x(swap)	n/a
y(swap)	n/a
t(swap)	0
…	
pr1(main)	**12**
pr2(main)	**10**
…	

30

Esempio: *passaggio di parametri per riferimento*

```
 4  void swap (int &x, int &y)
 5  {
 6          int t=0;
 7          t=x; x=y; y=t;
 8  }
 9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2
14          swap(pr1,pr2);
15          ut << "dopo : pr1=" << pr1 << " pr2=" << pr2
16  }
```

Identificatore	valore
x(swap)	**pr1(main)**
y(swap)	**pr2(main)**
t(swap)	0
…	
pr1(main)	12
pr2(main)	10
…	

La funzione **swap** è invocata con le variabili **pr1** e **pr2** di **main** tuttavia, alle variabili **x** e **y** di **swap** sarà passato il loro riferimento

31

Esempio: *passaggio di parametri per riferimento*

Il la **swap** fa il suo lavoro e scambia il valore delle sue *variabili locali* **x** e **y** utilizzate come variabili intere, sebbene qualsiasi operazione venga fatta su di loro ha effetto sulle variabili **pr1** e **pr2** di main

```
4   void swap (int &x, int &y)
5   {
6           int t=0;
7           t=x; x=y; y=t;
8   }
9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2
14          swap(pr1,pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2
16  }
```

Identificatore	valore
x(swap)	pr1(main)
y(swap)	pr2(main)
t(swap)	12
…	
pr1(main)	10
pr2(main)	12
…	

32

Esempio: *passaggio di parametri per riferimento*

```
4   void swap (int &x, int &y)
5   {
6           int t=0;
7           t=x; x=y; y=t;
8   }
9
10  int main()
11  {
12          int pr1=12, pr2=10;
13          cout << "prima: pr1=" << pr1 << " pr2=" << pr2 << e
14          swap(pr1,pr2);
15          cout << "dopo : pr1=" << pr1 << " pr2=" <<
16  }
```

Identificatore	valore
x(swap)	n/a
y(swap)	n/a
t(swap)	0
…	
pr1(main)	10
pr2(main)	12
…	

Il controllo torna alla **main**. Il valore delle variabili **pr1** e **pr2** è stato invertito

33

I riferimenti *(reference)* in C++

I riferimenti sono stati introdotti per fornire una alternativa *più semplice e sicura* all'uso dei puntatori:

Nel passaggio dei parametri: per evitare l'*overhead* della copia dei valori e/o per consentire alle funzioni di modificare i *parametri reali*

Nella manipolazione di strutture dati *dinamiche:* fornendo una sintassi semplificata rispetto ai puntatori e imponendo una *maggiore disciplina* in situazioni che potrebbero produrre errori a *runtime*

Laddove non siano necessarie alcune caratteristiche dei puntatori (*e.g.* – l'aritmentica dei puntatori, puntatori a funzioni, …)

34

I riferimenti (*reference*) in C++

Dichiarazione:

```
int count=1;
int &countRef = count;
…
countRef++;
cout << "count=" << count << endl;
```

```
count=2
```

La variabile **`countRef`** è un sinonimo (un alias) della variabile **`count`**. Nelle espressioni essa appare con lo stesso significato del tipo base e qualsiasi modifica ha effetto anche sulla variabile originale.

35

I riferimenti (*reference*) in C++

Dichiarazione:

NO!

```
int x=0;
int &xRef;
…
```

Le variabili *reference* devono obbligatoriamente essere inizializzate. La mancata inizializzazione causa un errore in fase di compilazione

36

I riferimenti (*reference*) in C++

Espressioni:

```
int x=0, y=100;
int &xRef=x;
…
xRef=y;
xRef++;
cout << "x=" << x << endl;
```

Questa è una espressione di assegnamento tra interi e produce produce l'assegnamento a x del valore di y.

Una volta inizializzato, un *reference* è «per sempre». Non è consentito farne l'alias di un'altra variabile;

37

I riferimenti (*reference*) in C++

Espressioni:

```
int x=0, y=100;
int &xRef=x;
…
xRef=y;
xRef++;
cout << "x=" << x << endl;
```

Questa è una espressione di incremento che produce effetti sulla variabile cui **xRef** è riferita (in questo caso **x**). Non esiste una *aritmetica dei riferimenti*.

Una volta inizializzato, un *reference* è «per sempre». Non è consentito farne l'alias di un'altra variabile;

38

Esercizio: *input di un record*

Dato la seguente struttura dati PUNTO e il codice della funzione main() che ne fa uso, scrivere le funzioni inpunto e outpunto, utilizzando i riferimenti.

```
1  #include<iostream>
2  #include<cmath>
3  using namespace std;
4
5  typedef struct {
6         double x;
7         double y;
8  } PUNTO;
9
```

39

Esercizio: *input di un record*

```
19  int main()
20  {
21          PUNTO p1, p2;
22
23          cout << "Inserisci il punto p1: ";
24          inPunto(p1);
25          cout << "Inserisci il punto p2: ";
26          inPunto(p2);
27          cout << "p1:";
28          outPunto(p1);
29          cout << " p2:";
30          outPunto(p2);
31          cout << endl;
32  }
```

40

Esercizio: *input di un record*

```
10  void outPunto(PUNTO &p)
11  {
12          cout << "("<<p.x<<","<<p.y<<")";
13  }
14
15  void inPunto(PUNTO &p)
16  {
17          cin >> p.x >> p.y;
18  }
```

41

Puntatori

42

I puntatori in C/C++

I *puntatori* sono variabili che contengono *indirizzi di memoria*:

Permettono un accesso *flessibile* a strutture dati *dinamiche* (*i.e.* di dimensioni che possono variare nel tempo) di varia natura: sia composta da tipi di dati e operatori *nativi* (int, float[],..) sia tipologie di dati, e relativi operatori, definiti dal programmatore

Implementazione degli array

Implementazione delle stringhe

Passaggio di parametri a funzione *per indirizzo* (con *effetti collaterali*)

43

I puntatori in C/C++

Nel codice sotto, al *puntatore a interi* **maxPtr** è assegnato l'indirizzo della variabile intera **max**;

```
int cnt=0,max;
int *maxPtr;
…
maxPtr=&max;
```

Nel gergo del C/C++, si dice che ora **maxPtr** *punta a* **max**;

45

I puntatori in C/C++

Nel codice sotto, al *puntatore a interi* **maxPtr** è assegnato l'indirizzo della variabile intera **max**;

```
maxPtr=&max;
```

L'operatore prefisso **&**, posto prima di una variabile, ne restituisce l'indirizzo di memoria.

46

I puntatori in C/C++

Sebbene possano essere dichiarati puntatori a qualsiasi tipo di dato, inclusi dati aggregati e classi…

```
…
int *maxPtr;
double *avgPtr;
struct punto *pPtr;
…
```

Tutte le variabili puntatore contengono il medesimo «tipo» di dato: quello utilizzato dall'architettura sottostante per rappresentare gli indirizzi di memoria.

47

I puntatori in C/C++

L'accesso al contenuto della variabile puntata si effettua usando l'operatore di *dereferenziazione* *

```
int max=0, *maxPtr;
maxPtr=&max;
…
cout << *maxPtr << endl;
*maxPtr+=1;
Cout << max << endl;
```

```
0
1
```

Fintantochè **maxPtr** punta a **max**, qualsiasi modifica fatta a ***maxPtr** ha effetto su **max**.

48