Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno **luigi.catuogno@uniparthenope.it**
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

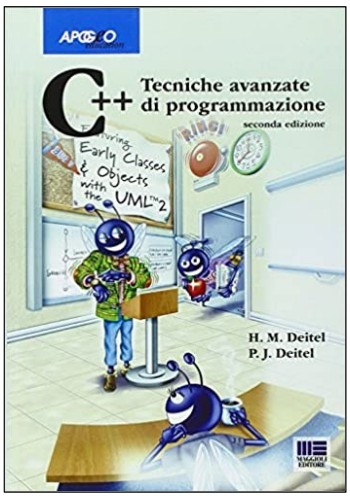
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

3

Libri di testo

Tecniche avanzate e strutture dati elementari

[TAP] H. M. Deitel, P. J. Deitel
C++ Tecniche avanzate di programmazione
II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Dal C al C++

6

Tipo di dati **bool**

7

Tipo di dati **bool**

```
1  // Programma che illustra il tipo 'bool' del C++
2
3  #include <iostream>
4  using namespace std;
5  int main()
6  {
7        bool a1=true, b1=false, x;
8        x = a1 && b1;
9        cout << "a1 AND b1=" << x << endl;
10 }
```

```
a1 AND b1=0
```

8

Tipo di dati **bool**

```
7   bool a1=true, b1=false;
```

Le variabili di tipo **bool** possono essere inizializzate utilizzando in maniera equivalente gli interi **0** e **1** e le costanti **false** e **true**;

```
8   x = a1 && b1;
```

Può essere loro assegnato il risultato di una espressione logico-relazionale.

9

Tipo di dati **bool**

```
9   cout << "a1 AND b1=" << x << endl;
```

Generalmente una variabile **bool** è memorizzata come una variabile intera(*) (sebbene non possa avere valori diversi da 0 e 1) e come tale è visualizzata.

(*) un byte con un solo 1 bit significativo.

10

Tipo di dati **bool**

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6         bool a1, b1, x;
7         cout << "inserire a1 e b1: ";
8         cin >> a1 >> b1;
9         x = a1 && b1;
10        cout << "a1 AND b1= " << boolalpha << x << endl;
11 }
```

Il manipolatore **boolalpha** determina la traduzione tra le costanti numeriche e quelle testuali dei valori booleani

```
inserire a1 e b1: 1 0
a1 AND b1= false
```

11

Tipo di dati **bool**

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6         bool a1, b1, x;
7         cout << "inserire a1 e b1: ";
8         cin >> boolalpha >> a1 >> b1;
9         x = a1 && b1;
10        cout << "a1 AND b1= " << boolalpha << x << endl;
11 }
```

Il manipolatore **boolalpha** determina la traduzione tra le costanti numeriche e quelle testuali dei valori booleani.

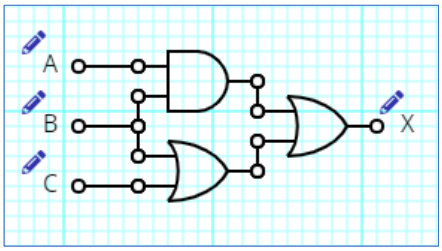
```
inserire a1 e b1: true false
a1 AND b1= false
```

12

Esercizio: *una funzione booleana*

Si scriva un programma C++ che chieda all'utente fornire in input i valori da applicare ai tre ingressi A,B e C del circuito al lato (codificate con delle variabili booleane), quindi calcoli e visualizzi il valore dell'uscita X.

1) Input e output devono essere effettuate utilizzando le costanti testuali true e false
2) Il circuito deve essere implementato nella funzione **`circuito()`**, mentre la funzione **`main()`** si dovrà occupare solo di chiedere l'input, invocare la funzione e visualizzarne il risultato.



13

Esercizio: *una funzione booleana*

```
1  #include<iostream>
2  using namespace std;
3
4  bool circuito (bool a, bool b, bool c)
5  {
6         return (a&&b)||(b||c);
7  }
8
```

14

Esercizio: *una funzione booleana*

```
 9  int main()
10  {
11          bool A, B, C, X;
12
13          cout << "Inserisci A,B e C: ";
14          cin >> boolalpha >> A >> B >> C;
15          X=circuito(A,B,C);
16          cout << "X=" << boolalpha << X << endl;
17  }
```

15

Esercizio: *il Crivello di Eratostene*

Il Crivello di Eratostene

Antico metodo per la ricerca dei numeri primi attribuito al matematico greco Eratostene da Cirene, vissuto tra il terzo e il secondo secolo a.C.

16

Esercizio: *il Crivello di Eratostene*

- Fissato n come limite superiore:
- Si utilizza un elenco (setaccio) dei numeri da 2 a n
  - Si scorre l'elenco partendo da 2 e si prendono in esame uno alla volta, tutti i numeri che non siano stati già «cancellati»
  - si «cancellano» tutti i multipli del numero in esame (escluso il numero stesso);
  - al termine del procedimento, i numeri primi sono quelli che non risultano cancellati;

Suggerimento: per il crivello, si utilizzi un array di N elementi di tipo **`bool`**.

17

Esercizio: *il Crivello di Eratostene*

$N = 30;$

Usiamo due cicli annidati i cui indici *i* e *j* indicano rispettivamente il numero in esame e il moltiplicatore. Per ogni *i* da 2 a N (che non sia stato già marcato), e per ogni *j* da *i* a N, si «marcano» le posizioni i*j

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

scelto i=**2**…
j=**2, 3, 4, 5**…

	2	3	✗	5	✗	7	✗	9	✗
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

18

Esercizio: *il Crivello di Eratostene*

$N = 30;$

Usiamo due cicli annidati i cui indici *i* e *j* indicano rispettivamente il numero in esame e il moltiplicatore. Per ogni *i* da 2 a N (che non sia stato già marcato), e per ogni *j* da *i* a N, si «marcano» le posizioni i*j

scelto i=**2**…
J= … **6, 7, 8, 9, 10**…

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	22	23	24	25	26	27	28	29	30

j= … **11, 12, 13, 14, 15**

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~

19

Esercizio: *il Crivello di Eratostene*

$N = 30;$

Dopo aver marcato i multipli di due (ma non due) passiamo a marcare i multipli di tre (non importa se alcuni sono già marcati). Saltiamo i=4 perché questo è già marcato, e si passa a enumerare i multipli di 5…

scelto i=**3**…
j=**3, 4, 5, 6, 7, 8, 9, 10**…

	2	3	~~4~~	5	~~6~~	7	~~8~~	✓ 9	~~10~~
11	~~12~~	13	~~14~~	✓ 15	~~16~~	17	~~18~~	19	~~20~~
✓ 21	~~22~~	23	~~24~~	25	~~26~~	✓ 27	~~28~~	29	~~30~~

scelto i=**5**…
j=**5, 6**

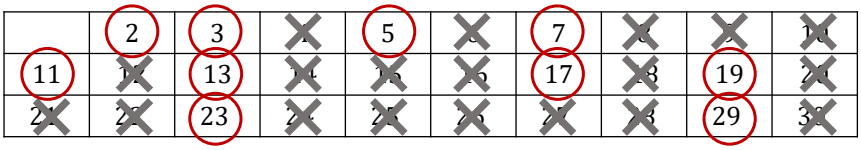
	2	3	~~4~~	5	~~6~~	7	~~8~~	✓ 9	~~10~~
11	~~12~~	13	~~14~~	✓ 15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	~~25~~	~~26~~	27	~~28~~	29	~~30~~

20

Esercizio: *il Crivello di Eratostene*

$N = 30;$

L'operazione di marcatura termina quando *i*i* diventa maggiore di N (nel nostro caso quando *i=5*).
A questo punto, si scorre l'array e si visualizzano tutti i numeri non marcati. Questi saranno i numeri primi compresi tra 2 e N.



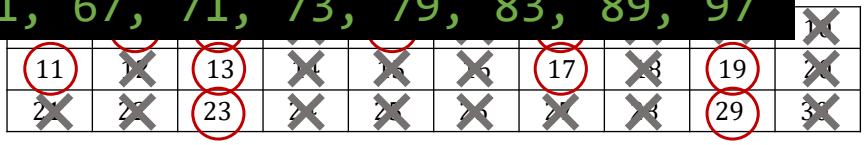
21

Esercizio: *il Crivello di Eratostene*

$N = 30;$

L'operazione di marcatura termina quando *i*i* diventa maggiore di N (nel nostro caso quando *i=5*).
A qu
sarar

```
I numeri primi da 1 a 100 sono:
2, 3, 5, 7, 11, 13, 17, 19, 23,
29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97
```



22

Puntatori e riferimenti

25

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*)

26

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**;

27

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 **&** (detto di *referenziazione*) posto prima di una variabile, ne restituisce l'indirizzo di memoria.

28

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.

29

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**.

30

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'aritmetica dei puntatori, puntatori a funzioni, …)

31

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.

32

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

33

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;

34

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;

35

Funzioni: passaggio dei paramteri

36

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.*

37

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.

38

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.

39

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*.

40

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** ?

41

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** ?

42

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          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
…	

43

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          cout << "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**

44

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          cout << "dopo : pr1=" << pr1 << " pr2=" << pr2 << en
16  }
```

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**

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

45

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          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.

46

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*

47

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.

48

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 << e
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
…	

49

Esempio: *passaggio di parametri per indirizzo*

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.

```
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 }
```

50

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
…	

51

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 *
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
…	

52

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 << e
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
…	

53

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.

54

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.

55

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.

56

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.

57

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
…	

58

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

59

Esempio: *passagg…nento*

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
…	

60

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

61