Ripasso 2

Dall’algoritmo al programma

La programmazione

Quello che faremo in questo corso sarà esplorare i fondamenti della programmazione, e, in particolare, di un linguaggio di programmazione: C++.

Il nome C++ deriva dal suo “antenato” C, poiché non è altro che una sua evoluzione.

La programmazione: esempio in linguaggio C

Vediamo un piccolo programma formato solo da poche righe di codice scritto in linguaggio C:

Sembra piuttosto criptico: ci sono diversi simboli strani: hashtag, parentesi angolari, parentesi graffe, parentesi tonde, punto e virgola, barra rovesciata n e qualche altra curiosità sintattica. Cosa significa?

```c
#include <stdio.h>

int main()
{
    printf("Hello, World\n");
}
```

La programmazione: esempio in linguaggio C

Tuttavia potete indovinare cosa fa questo programma, anche se non conoscete la sintassi.

La parola **printf** ricorda “print”, stampa in inglese poi leggete la frase: "**Hello, World**", anche se non siete sicuri di cosa sia **\n**.

Probabilmente immaginate che questo programma, una volta eseguito, stamperà Hello, World

In effetti, è quello che fa.

```c
#include <stdio.h>

int main()
{
    printf("Hello, World\n");
}
```

La programmazione

In questo momento siete nella stessa condizione di chi vuole imparare una nuova lingua, poiché non conoscete i simboli e la sintassi del linguaggio C, ma nella programmazione c'è un aspetto molto più entusiasmante: nonostante esistano dozzine, se non centinaia, di linguaggi di programmazione, essi sono molto più accessibili delle lingue umane scritte e parlate.

Infatti esistono talmente tante somiglianze tra i linguaggi di programmazione che basta conoscerne pochi per impararne da soli molti altri.

La programmazione

Alla fine, la programmazione è scrivere software che serve a controllare l'hardware per risolvere i problemi.

Fare una telefonata su un dispositivo mobile, inviare un'e-mail o cercare qualcosa su Internet sono esempi di problemi risolti controllando l'hardware con il software.

Eppure… non posso fare a meno di ricordare che i computer sono in grado di comprendere solo zero e uno, giusto?

Capiscono solo il cosiddetto sistema binario…

La programmazione

Eppure, il programma visto non sembra proprio fatto di zeri e uni.

Allora com'è possibile che io, essere umano, possa programmare un computer usando il codice in C e ottenere che il computer lo comprenda, anche se questo programma non è fatto di zeri e uni?

In altre parole: come si fa a passare dall'uno all'altro?

```c
#include <stdio.h>

int main()
{
    printf("Hello, World\n");
}
```

La compilazione

Quando si scrive un programma per computer, spesso ci sono più passaggi: a volte questi passaggi sono automatizzati, ma altre volte occorre digitarli, cioè devo farli accadere manualmente.

In generale, il codice scritto in C, detto codice sorgente, può essere convertito in codice macchina, (fatto da zeri e uni) da un particolare tipo di programma chiamato compilatore.

Un compilatore è un programma che *traduce* il codice sorgente in codice macchina, in modo che la macchina possa comprendere ciò che l'essere umano ha scritto.

La programmazione: esempio in linguaggio C++

Consideriamo un altro esempio: questo programma è scritto in un linguaggio molto simile chiamato C++. Anche in questo caso potreste non riconoscere alcuni simboli, ma probabilmente riconoscerete "**Hello, World**", questa volta senza **\n**.

Anche se abbiamo parole diverse: iostream, namespace std, endl, è probabile che anche questo programma Hello, World

```
#include <iostream>

using namespace std;

int main()
{
    cout<<"Hello, World"<<endl;

}
```

La programmazione

Questi due semplici esempi sono una testimonianza del fatto che molti linguaggi di programmazione fanno le stesse cose in modo diverso.

In effetti, che voi conosciate C o C++ o qualsiasi altro linguaggio di programmazione, potrete effettivamente risolvere gli stessi problemi usando qualunque linguaggio di programmazione, proprio come gli esseri umani, possono esprimersi verbalmente in un numero qualsiasi di lingue parlate.

Programmare in C++

Struttura di un programma C++

Nella prossima slide è rappresentato un esempio che illustra le parti che compongono un programma: è fondamentale comprendere la struttura complessiva e non soffermarsi sui contenuti delle singole parti.

Basti sapere che si tratta di un programma che riceve in input due numeri interi, ne calcola la somma e la manda in output.

Direttive iniziali

Rappresentano il modo per indicare al compilatore le librerie che dovranno essere usate nel programma.

Funzione main

È la funzione principale di ogni programma C++ e, come tale, deve sempre essere presente, in quanto l'esecuzione del programma inizia proprio con essa. Le parentesi tonde poste dopo il nome *main* rappresentano il simbolo di riconoscimento di una funzione (pertanto, sono sempre presenti accanto a tutti i nomi di funzione).

```
#include <iostream>
using namespace std;

int main()
{
  int a,b,somma;

  cout << "Inserire il primo numero: ";
  cin >> a;
  cout << "Inserire il secondo numero: ";
  cin >> b;
  somma = a + b;
  cout << "La somma = " << somma;
  return 0;
}
```

Sezione dichiarativa

È la sezione dove si dichiarano le variabili che saranno usate dalle istruzioni presenti nella funzione *main*.
Nel linguaggio C++, a differenza di altri linguaggi di programmazione, non è obbligatorio dichiarare le variabili in una sezione specifica del programma, così come stiamo indicando, ma è comunque buona norma dichiararle all'inizio in modo da ottenere un codice ordinato che faciliterà la ricerca e la correzione degli errori.

Le parentesi graffe racchiudono l'intero blocco di codice.

Sezione esecutiva (o corpo del programma)

È la zona in cui vengono scritte le istruzioni che devono essere eseguite.
Nota che ogni istruzione termina con un punto e virgola (;).
Per garantire una buona leggibilità del codice, è opportuno:

- scrivere una sola istruzione per riga;
- introdurre nelle istruzioni opportuni spazi (che vengono ignorati dal compilatore).

Ricorda comunque che le istruzioni lunghe possono essere scritte su più righe successive, dato che è il punto e virgola a segnalare al compilatore la fine dell'istruzione.

Struttura di un programma in C++

Direttiva iniziale: **#include <iostream>**

Le frasi che iniziano con il simbolo di cancelletto (#) sono direttive per il preprocessore del compilatore.

Esse non sono istruzioni eseguibili ma soltanto indicazioni per il compilatore.

Nel nostro caso la direttiva **#include <iostream>** dice al preprocessore del compilatore di includere il file della libreria standard **iostream**.

Questo particolare file contiene le dichiarazioni delle operazioni basilari di input/output definite nella libreria standard del C++ e viene incluso perché tali operazioni serviranno in seguito nel programma.

Struttura di un programma C++

La dichiarazione
using namespace std;
in **C++** viene utilizzata per specificare che si desidera utilizzare le funzioni e le classi presenti all'interno dello spazio di nomi standard (**namespace std**).

Il namespace std contiene molte delle funzioni e delle classi standard fornite dal compilatore C++, tra cui input/output, stringhe, vettori, algoritmi e molto altro.

Senza la dichiarazione **using namespace std**; sarebbe necessario qualificare ogni funzione o classe con il prefisso **std::** ogni volta che si desidera utilizzarle.

Struttura di un programma C++

Ad esempio, per utilizzare la funzione **cout** per stampare un valore a video, senza la dichiarazione

using namespace std; sarebbe necessario scrivere:

```
#include <iostream>
int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Invece con la dichiarazione **using namespace std;** si può scrivere…

Struttura di un programma C++

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!" << endl;
return 0;
}
```

In questo modo, si evita di dover qualificare ogni funzione o classe con il prefisso **std::** ogni volta che si desidera utilizzarle.

La dichiarazione **using namespace std;** è molto utile e comoda quando si scrive codice che utilizza molte funzioni e classi standard, poiché semplifica la scrittura del codice e lo rende più leggibile.

Struttura di un programma C++

Funzione main: **int main()**

Questa riga è l'inizio della dichiarazione della funzione **main**.

La funzione **main** è la funzione principale di ogni programma C++ e deve essere sempre presente, in quanto l'esecuzione di un qualsiasi programma C++ inizia proprio con essa.

Il punto del programma in cui compare tale funzione è irrilevante poiché essa è sempre la prima ad essere eseguita.

Le parentesi tonde **()** poste dopo il nome **main** rappresentano il simbolo di riconoscimento di una funzione, pertanto in C++ tutte le funzioni sono seguite da una coppia di parentesi **()** che, eventualmente, possono contenere degli argomenti.

Subito dopo la dichiarazione (l'intestazione) della funzione viene il corpo (il contenuto) della funzione racchiuso tra parentesi graffe **{}**

Commento

// il mio primo programma in C++

Questa è una riga di commento.

Tutte le righe che iniziano con la coppia di slash (//) sono considerate commento e vengono ignorate dal compilatore. Esse possono essere usate dal programmatore per aggiungere note o brevi descrizioni in mezzo alle istruzioni del programma.

In C++ ci sono due modi di inserire commenti:

- **// riga di commento** considera commento tutto ciò che segue la coppia di sbarre (//) fino alla fine della riga.
- **/* blocco di commento */** considera commento tutto ciò che è compreso tra la coppia di caratteri /* e la coppia di caratteri */, eventualmente anche più righe di testo.

Identificatori in C++

Gli identificatori sono i nomi definiti dal programmatore per riferirsi in modo univoco a oggetti di categorie diverse:

- Variabile
- Costante
- Funzione

Identificatori in C++

- Variabile: è un contenitore di valori; ogni variabile può contenere un singolo valore che può cambiare nel tempo. In C++ è necessario dichiarare il tipo di appartenenza di ogni variabile, che definisce:
  - quali valori possono essere contenuti nella variabile
  - quali operazioni possono essere eseguite su questi valori
  - la quantità di memoria (misurata in byte) che la variabile occupa. Infatti non tutti i tipi di valore occupano la stessa quantità di memoria. Ad esempio: occorre una quantità di memoria diversa per registrare un carattere, o un numero intero.

Identificatori in C++

- Costante: è una qualsiasi espressione che ha un valore prefissato. Le costanti si possono suddividere in:
  - Numeri Interi,
  - Numeri in Virgola Mobile,
  - Caratteri
  - Stringhe.
- Funzione: è un sottoprogramma, cioè un insieme di istruzioni identificate da un nome che rappresentano una porzione autonoma di codice, che può essere richiamata all'interno del programma principale o all'interno dei sottoprogrammi. In questo caso si dice che il programma chiamante effettua una chiamata alla funzione.

Identificatori in C++

Un identificatore deve iniziare con una lettera o con carattere underscore _, ma non può iniziare con una cifra numerica.

Può contenere un numero qualsiasi di lettere, cifre o underscore, ma non può contenere spazi.

Non sono validi gli identificatori che coincidono con le parole chiave del linguaggio.

Esempi di identificatori non validi:

un amico (contiene uno spazio)

un'amica (contiene un apostrofo)

7bello (il primo carattere non è una lettera)

for (è una parola-chiave del C++)

@my_mobile (contiene il simbolo @ che non è né lettera, né cifra, né _)

Esempi di identificatori validi:

hello

deep_space9

a123

_7bello

Identificatori in C++

CARATTERISTICHE DELL'IDENTIFICATORE	ESEMPI	
Non può iniziare con un numero	1nano, 2nano	NON AMMESSO
Non può contenere spazi	valore medio, ora legale	NON AMMESSO
Non può contenere caratteri speciali	valore%, anni/mes	NON AMMESSO
Non può contenere lettere accentate	età, velocità, città	NON AMMESSO
Deve indicare quello che contiene	temperatura, media, numero1	
Può non avere senso compiuto	xyz, k123, kkzz	
Può essere composto da due parole	valoreMedio, annoSolare, formaOcchi	
Può essere lungo a piacere	variabiletemporanea, vt, vartempo	
Può essere composto da un solo carattere	x, y, z, i, j	
Può iniziare con l'underscore	_tempo, _var1, _pippo2	

Identificatori in C++

Tutti gli identificatori presenti in un programma devono essere diversi tra loro, indipendentemente dalla categoria cui appartengono.

Inoltre il C++ è case sensitive, cioè fa distinzione tra lettere maiuscole e lettere minuscole, pertanto la variabile **alfa** è diversa dalla variabile **Alfa** che è diversa dalla variabile **ALFA**

Dichiarazione delle variabili in C++

Ogni variabile, prima di poter essere utilizzata, deve essere dichiarata, specificando:

- a quale tipo di dato essa appartenga.
- l'identificatore

La dichiarazione è necessaria per fare in modo che il calcolatore, all'atto dell'esecuzione del programma, possa preparare in anticipo lo spazio in memoria per ospitare la variabile.

Questa operazione viene definita "allocazione".

Dichiarazione delle variabili in C++

La sintassi di una dichiarazione di variabile prevede prima il nome del tipo di dato (p.es. **int**, **float** ...) seguito dall'identificatore scelto per denotare tale variabile. Ad esempio:

```
int a;
float raggio;
```

Sono dichiarazioni di variabili corrette.

La prima dichiara una variabile di tipo **int** denotata dall'identificatore **a**.

La seconda dichiara una variabile di tipo **float** denotata dall'identificatore **raggio**.

Una volta dichiarate, le variabili **a** e **raggio** possono essere usate nel programma all'interno del loro campo di validità.

Dichiarazione delle variabili in C++

Se vogliamo dichiarare più di una variabile dello stesso tipo possiamo farlo in una stessa riga indicando una sola volta il tipo e separando gli identificatori con la virgola.

Ad esempio:

```
int a, b, c;
```

dichiara tre variabili (**a**, **b** e **c**) di tipo **int** , ed ha esattamente lo stesso significato di:

```
int a;
int b;
int c;
```

Tipi di variabile in C++

Nella seguente tabella sono riportati i tipi di variabile del linguaggio di programmazione C++ e l'insieme di valori corrispondente ad ogni tipo.

Tipo di variabile	Insieme dei valori
`int`	Numeri interi (è possibile precisare short int o long int per numeri interi piccoli o molto grandi)
`float`	Numeri reali a 32 bit (singola precisione)
`double`	Numeri reali a 64 bit (doppia precisione)
`char`	Caratteri alfanumerici (trattati di fatto come interi)
`bool`	Assume esclusivamente valori true (vero) o false (falso)
`string`	Sequenza alfanumerica di caratteri

Tipi di variabile in C++

Nella seguente tabella è riportata l'occupazione di memoria (in byte) per i vari tipi di variabile del linguaggio di programmazione C++:

Data Type	Memory Size
bool	1 byte
char	1 byte
int	4 bytes
float	4 bytes
double	8 bytes

Valore della variabile dopo la dichiarazione

Subito dopo l'allocazione, che valore assume la variabile?

Subito dopo l'allocazione il valore di una variabile è indeterminato. In particolare, il valore è impredicibile perché dipende dal contenuto preesistente della particolare locazione di memoria che il calcolatore ha deciso di utilizzare per ospitare il dato.

Operazioni di lettura su una variabile allocata e non ancora inizializzata sono, quindi, del tutto prive di senso.

È necessario inizializzare una certa variabile prima di poterla usare assegnandole un valore.

Inizializzazione della variabile

È possibile inizializzare una variabile contemporaneamente all'atto della sua allocazione tramite l'operatore di assegnamento =.

Esempi:

int a = 0;

char c = 'g';

N.B.: l'assegnazione non è una relazione di uguaglianza, ma produce una modifica della memoria.

È importante tenere presente che l'istruzione **a = b;** non significa «*il valore della variabile* **a** *è uguale a quello della variabile* **b**»; significa invece: «*cambiamo l'attuale valore della variabile* **a** *e le assegniamo il valore che in questo momento ha la variabile* **b**».

Esempio

```
int a, b;    // a:?  b:?
a = 10;      // a:10 b:?
b = 4;       // a:10 b:4
a = b;       // a:4 b:4
b = 7;       // a:4 b:7
```

Alla fine otteniamo che il valore contenuto in **a** è **4** e quello contenuto in **b** è 7.

L'ultima modifica di **b** non ha modificato **a**, benché appena prima avessimo scritto **a = b**

Costanti in C++

Le costanti sono valori prefissati inseriti all'interno del programma.

Anche una costante è uno spazio di memoria a cui è assegnato un nome e un tipo di dato, ma ha un valore fissato, che non può cambiare nel corso dell'elaborazione.

Si possono avere:

- Costanti intere
- Costanti reali
- Costanti di tipo carattere
  - Racchiuse tra apici singoli
- Costanti di tipo stringa
  - Racchiuse tra apici doppi

Costanti in C++

In questo corso considereremo le sole costanti tipizzate cioè costanti che accettano un valore ammissibile per il tipo di dato a cui sono associate, dove il tipo è esplicitamente dichiarato dal programmatore.

A differenza delle variabili, che si possono inizializzare al momento della dichiarazione oppure in seguito, le costanti vanno inizializzate quando le si dichiara.

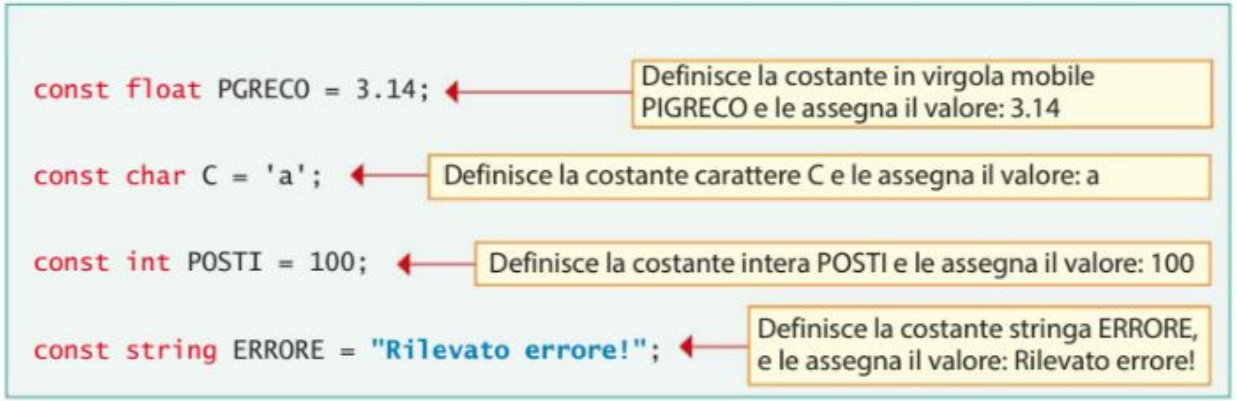
N. B. Ogni tentativo di modifica di una costante verrà segnalato come errore in fase di compilazione

Costanti in C++

In C++ le costanti tipizzate sono definite dal qualificatore **const** e la sintassi è la seguente:

const tipo nome_costante = valore_costante

Ad esempio:



Istruzioni di Input/Output in C++

L'interfaccia del calcolatore è di solito composta da una tastiera ed un video e prende il nome di *console*.

La tastiera viene usata come unità di *input* standard e il video come unità di *output* standard.

Nella libreria standard **iostream** del C++ le operazioni di *input* ed *output* di un programma vengono gestite da due flussi di dati (*stream*):

- **cin** per l'input (console input)
- **cout** per l'output (console output)

Dove **cout** (il flusso di output standard) è normalmente diretto al video e **cin** (il flusso di input standard) è normalmente assegnato alla tastiera.

Istruzioni di Input/Output in C++

Usando questi due flussi un programma può interagire con un utente mostrando messaggi sullo schermo e ricevendo l'input da parte dell'utente dalla tastiera.

<<x

```
cout << x;
```

>>y

```
cin >> y;
```

Istruzioni di Input/Output in C++

Il flusso **cout** viene usato assieme all'operatore **<<** detto operatore di inserimento o operatore di ridirezione dell'output in quanto: se lo interpretiamo come una specie di freccia, esso indica che il valore alla sua destra deve essere inviato allo standard output (rappresentato da **cout**), cioè allo schermo del PC.

Esempi:

```
cout << "Ciao a tutti!";    // stampa sullo schermo il messaggio Ciao a tutti!
cout << 120;                // stampa il numero 120 sullo schermo
cout << x;                  // stampa il valore della variabile x sullo schermo
```

Negli esempi precedenti esso inserisce nel flusso di output standard **cout** la stringa costante **Ciao a tutti!**, la costante numerica **120** e il valore della variabile **x**.

Istruzioni di Input/Output in C++

Osserviamo che nella prima riga la frase da stampare è racchiusa tra doppi apici ("), questo perché essa è una stringa di caratteri.

Quando vogliamo usare una stringa costante di caratteri dobbiamo racchiuderla tra doppi apici (") per distinguerla da un identificatore di variabile.

Ad esempio, le due frasi seguenti hanno un significato molto diverso:

```
cout << "Salve";      // stampa Salve sullo schermo
cout << Salve;        // stampa il valore della variabile Salve sullo schermo
```

Istruzioni di Input/Output in C++

Si può utilizzare un unico **cout** per visualizzare diversi elementi, concatenando gli operatori di *inserimento* (<<). Ad esempio:

```
cout << "La variabile x vale: " << x <<" Invece la variabile y vale: "<<y;
```

Supponendo che le variabili x e y valgano rispettivamente 10 e 15 verrà visualizzato sullo schermo:

La variabile x vale: 10 Invece la variabile y vale: 25

Istruzioni di Input/Output in C++

Attenzione: **cout** non va a capo dopo aver stampato, a meno che non glielo si dica esplicitamente. Pertanto, con le due seguenti frasi:

```
cout << "Questa e' una frase.";
cout << "Questa e' un'altra frase.";
```

otteniamo:

Questa e' una frase.Questa e' un'altra frase.

anche se sono state scritte con due distinte chiamate a **cout** .

Per andare a capo bisogna inserire esplicitamente nel flusso un carattere speciale di *nuova linea* che, in C++, si indica con **\n**

Istruzioni di Input/Output in C++

cout << "Prima frase.\n ";

cout << "Seconda frase.\nTerza frase.";

produce:

Prima frase.

Seconda frase.

Terza frase.

Per andare a capo possiamo anche usare il *manipolatore* **endl** . Ad esempio:

cout << "Prima frase." << endl;

cout << "Seconda frase." << endl;

stampa:

Prima frase.

Seconda frase.

Istruzioni di Input/Output in C++

In C++ l'input standard si effettua applicando al flusso **cin** l'operatore di >> detto operatore di *estrazione* o operatore di ridirezione dell'input.

Tale operatore deve essere seguito dalla variabile in cui deve essere memorizzato il valore da leggere. Ad esempio:

```
int raggio;
cin >> raggio;
```

dichiara la variabile **raggio** di tipo **int** e quindi aspetta un input da **cin** (tastiera) per poter memorizzare un valore intero in tale variabile.

Istruzioni di Input/Output in C++

Attenzione: **cin** elabora l'input ricevuto da tastiera soltanto dopo che è stato premuto il tasto di invio **ENTER**.

Quindi, anche se viene richiesto di leggere un singolo carattere, **cin** non elabora l'input fino a quando, dopo aver premuto il tasto del carattere da leggere, non viene premuto anche il tasto **ENTER**.

Quando si usa l'estrattore (>>) su **cin** bisogna tener presente il tipo della variabile che si usa per memorizzare il valore letto:

- se viene richiesto un intero deve essere introdotto un intero,
- se viene richiesto un carattere deve essere introdotto un carattere
- se viene richiesta una stringa deve essere introdotta una stringa.

Istruzioni di Input/Output in C++

Si può anche usare **cin** per richiedere più di un dato alla volta, concatenando gli operatori di estrazione (>>).

Per esempio:

cin >> a >> b;

è equivalente a:

cin >> a;

cin >> b;

In entrambi i casi l'utente deve fornire due valori dei tipi appropriati, uno per la variabile **a** ed uno per la variabile **b**.

Esercizio

Date due variabili A e B che contengono due numeri interi, scrivere un algoritmo che scambi il contenuto delle due variabili.

Per esempio: se prima dell'esecuzione dell'algoritmo A vale 2 e B vale 6, dopo l'esecuzione dell'algoritmo A vale 6 e B vale 2.

A prima vista potremmo pensare che la soluzione è immediata:

- prendi il contenuto di B e mettilo in A
- prendi il contenuto di A e mettilo in B… O no?

Analisi del problema

Naturalmente questa soluzione è errata, infatti: se prendi il contenuto di B e lo metti in A, avrai perso irrimediabilmente il contenuto della variabile A.

Quindi, per risolvere questo problema dobbiamo servirci di una terza variabile C in cui "salvare temporaneamente" il contenuto di A.

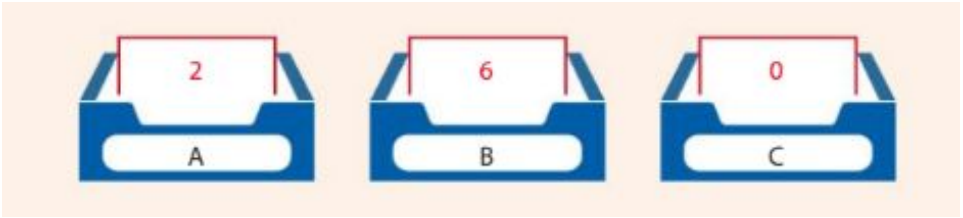
È un po' come se volessimo scambiare il contenuto di due bottiglie che contengono latte e acqua: non possiamo versare il latte contenuto nella prima bottiglia all'interno della seconda bottiglia, ma dobbiamo utilizzare una terza bottiglia vuota.

Soluzione

Le azioni da compiere sono le seguenti:

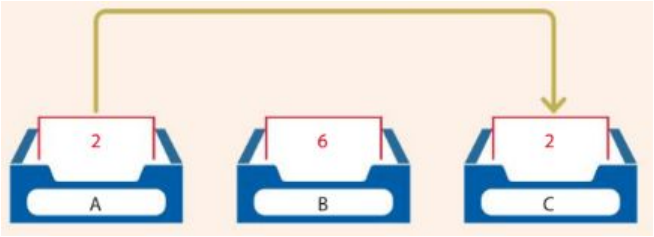
- prendi il contenuto di A e mettilo in C
- prendi il contenuto di B e mettilo in A
- prendi il contenuto di C e mettilo in B

Vediamo graficamente ricorrendo all'analogia con i contenitori.
Per esempio la situazione iniziale è:

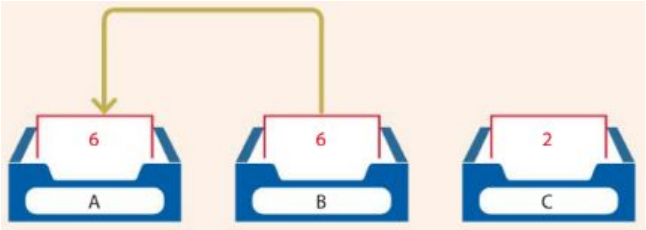


Soluzione

- Prendi il contenuto di A e copialo in C (cioè C conterrà una copia del valore di A)



- prendi il contenuto di B e copialo in A (cioè A conterrà una copia del valore di B)



Soluzione

- Prendi il contenuto di C e copialo in B (cioè B conterrà una copia del valore di C):

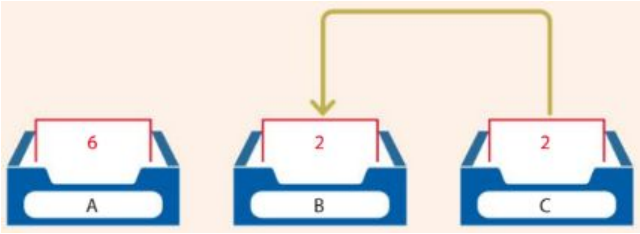
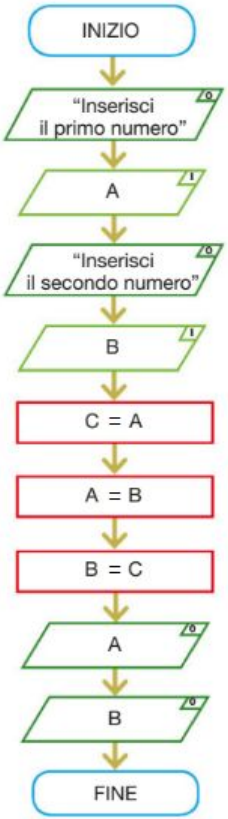


Diagramma di flusso e pseudocodice

- Dati di input: A, B
- Dati di output: A, B
- Dato di lavoro: C




```
PROGRAM main
  OUTLN "Inserisci il primo numero"
  INPUT A
  OUTLN "Inserisci il secondo numero"
  INPUT B
  C = A
  A = B
  B = C
  OUTLN A
  OUTLN B
END
```

Programma C++

```cpp
//Programma che scambia il contenuto di due variabili

#include <iostream>
using namespace std;

int main ()
{
  int A, B, C;                  //Dichiarazione delle variabili
  cout << "inserisci il primo numero intero: " << endl;
  cin >> A;
  cout << "inserisci il secondo numero intero: " << endl;
  cin >> B;

    /* Scambio dei valori contenuti nelle due variabili A e B
     mediante l'utilizzo della variabile ausiliaria C */

  C = A;
  A = B;
  B = C;
  cout << "Ora il primo numero vale: " << A <<" mentre il secondo numero vale: " << B << endl;

  return 0;
}
```

Operatori in C++

Abbiamo già visto l'operatore di assegnamento (=) che serve per assegnare un valore ad una variabile.

Oltre ad esso il linguaggio di programmazione C++ prevede cinque operatori aritmetici riportati nella seguente tabella:

Operatore	Operazione	LINGUAGGIO C++
+	Addizione	Addiziona due operandi o concatena due stringhe
–	Sottrazione	Sottrae il secondo operando dal primo
*	Moltiplicazione	Moltiplica i due operandi
/	Divisione	Applicato a variabili di tipo intero produce un risultato troncato della parte decimale
%	Resto (modulo)	Fornisce il resto della divisione tra due operandi interi
=	Assegnamento	Assegna alla variabile posta alla sinistra dell'uguale il valore posto a destra

Precedenza tra operatori

La **precedenza tra operatori** indica l'ordine con cui gli operatori vengono valutati dal **compilatore**. Un operatore con precedenza maggiore verrà valutato prima di un operatore con precedenza minore, anche se quest'ultimo compare prima dell'operatore con precedenza maggiore.

Ecco un esempio:

int risultato = 4 + 5 * 7 + 3;

Il risultato in questo caso dipende proprio dalla precedenza tra operatori.

L'operatore di **moltiplicazione** (*) ha precedenza rispetto all'operatore **addizione** (+).

Quindi la moltiplicazione 5*7 avverrà prima di tutte le altre addizioni e il valore di **risultato** è: 4+35+3=42

Precedenza tra operatori

Gli operatori di moltiplicazione (*), divisione (/) e modulo (%) hanno la precedenza rispetto agli operatori di addizione (+) e sottrazione (-).

Perché le operazioni siano effettuate con ordine diverso, sarà sufficiente introdurre delle **parentesi tonde**.

Ad esempio se vogliamo moltiplicare la somma 4+5 con la somma di 7+3, basterà scrivere:

```
int risultato = (4 + 5) * (7 + 3);
```

La variabile **risultato**, in questo caso, varrà 90.

Operatori di assegnamento composti

Gli operatori di assegnamento composti sono una caratteristica del C++ che contribuisce alla sua fama di essere un linguaggio sintetico.

Essi permettono di modificare il valore di una variabile con una sola operazione:

Pippo += 5; // equivale a **Pippo = Pippo + 5; Pippo -= 10;** // equivale a **Pippo = Pippo - 10; Pippo *= 3;** // equivale a **Pippo = Pippo * 3;** si tratta cioè di operatori derivanti dalla concatenazione dell'operatore di assegnamento con un altro operatore aritmetico.

Esempi

- **a = a + b** oppure: **a + = b**
- **a = a – b** oppure: **a – = b**
- **a = a * b** oppure: **a * = b**
- **a = a / b** oppure: **a / = b**
- **a = a % b** oppure: **a % = b**

Operatori aritmetici unari

Gli operatori aritmetici unari si applicano ad un solo operando e ne modificano il valore.

Incremento

```
Pippo++;
Pippo = Pippo + 1
Pippo += 1
```

fanno la stessa cosa: aumentano di 1 il valore di `Pippo`

Decremento

```
Pippo--;
Pippo  =  Pippo   -  1
Pippo  -=  1
```

fanno la stessa cosa: diminuiscono di 1 il valore di `Pippo`

Operatori aritmetici unari

Gli operatori incremento e decremento si possono usare sia come *prefissi* che come *postfissi*, cioè possono essere scritti prima dell'operando (**++a**) o dopo di esso (**a++**).

Benché in espressioni semplici come **a++** o **++a** gli operatori prefissi e postfissi abbiano lo stesso significato, in altri casi in cui viene utilizzato il risultato dell'operazione nella valutazione di un'altra espressione essi assumono un significato diverso.

Operatori aritmetici unari

- Se l'operatore di incremento viene usato come *prefisso* (**++a**) il valore dell'operando viene incrementato prima della valutazione dell'espressione e quindi l'espressione viene valutata usando il valore incrementato;
- se l'operatore di incremento viene usato come *postfisso* (**a++**) il valore dell'operando viene incrementato dopo la valutazione dell'espressione e quindi l'espressione viene valutata usando il valore non incrementato.

Vediamo un esempio della differenza tra i due modi di usare l'operatore…

Operatori aritmetici unari

Nel primo esempio, la variabile **B** viene incrementata prima che il suo valore venga copiato in **A**

Invece nel secondo esempio viene prima copiato in **A** il valore della variabile **B** e poi viene incrementato il valore della variabile **B**.

```
B = 3;
A = ++B;

equivale a:

B = 3;
B++;
A = B; // A è 4, B è 4
```

```
B = 3;
A = B++;

equivale a:

B = 3;
A = B;
B++; // A è 3, B è 4
```