IL LINGUAGGIO *C++*

Note sull'uso di DevC++

- Sistema operativo Microsoft Windows 95, 98, Millenium Edition, NT 4, 2000 o XP
- RAM: 8 Mb (consigliati almeno 32 Mb)
- CPU: Intel Pentium 100 MHz o compatibile (consigliati almeno 233 MHz)
- Spazio su disco: 30 Mb liberi (consigliati almeno 45 Mb)

Strumento: Dev-C++un Ambiente Integrato di Sviluppo per C (e C++)

- Web: http://www.bloodshed.net/dev/devcpp.html.
- Free Software (GNU General Public License).
- L'installatore è scaricabile da:
  http://prdownloads.sourceforge.net/devcpp/devcpp4990setup.exe

Configurazione di Dev-C++

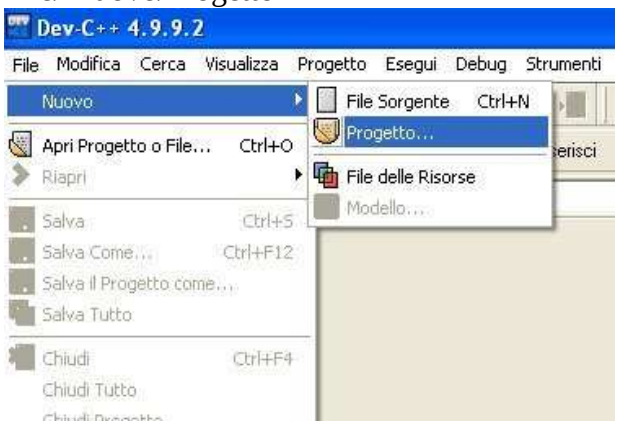
Lanciare il programma (Start → Programmi → Dev-C++ → Dev-C++).
Se si ha un messaggio relativo alla mancanza della libreria wininet.dll, scaricare wininet.exe ed eseguirlo
La prima volta che si lancia il compilatore appare una finestra di dialogo:

- click su "Ok"
- Selezionare "Options → Compiler options"
- Click su "Code generation / Optimization"
- Selezionare "Best optimization"
- Click su "Directories"
- Selezionare "Add the following commands when calling compiler", aggiungere "-Wall"
- Click su "Ok"

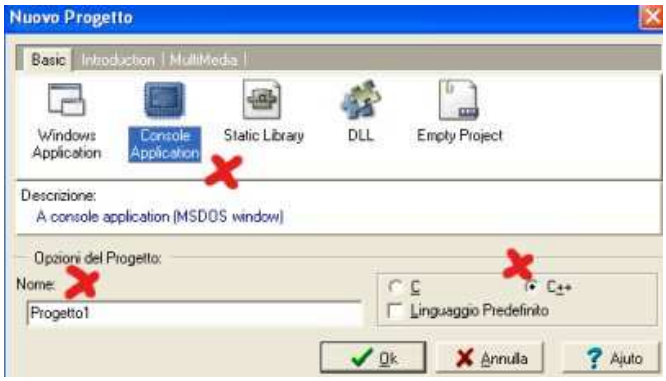
È possibile creare due diversi tipi di applicazioni:
– Applicazioni DOS (solo testo)
– Applicazioni Windows (grafica)
In questo corso ci limiteremo alla creazione di applicazioni DOS

Come Usare Dev-C++

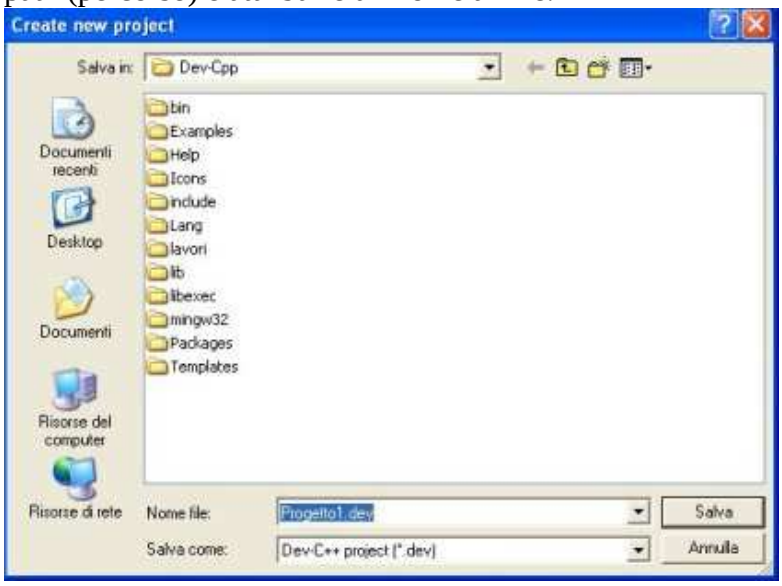
Dopo l'installazione, eseguire:
File/ Nuovo/Progetto



Nella finestra che si apre:
selezionare con un clic l'icona "Console application";
assicurarsi che sia selezionata l'opzione "C++";
scegliere il nome nel campo "nome";
clic su "Ok".

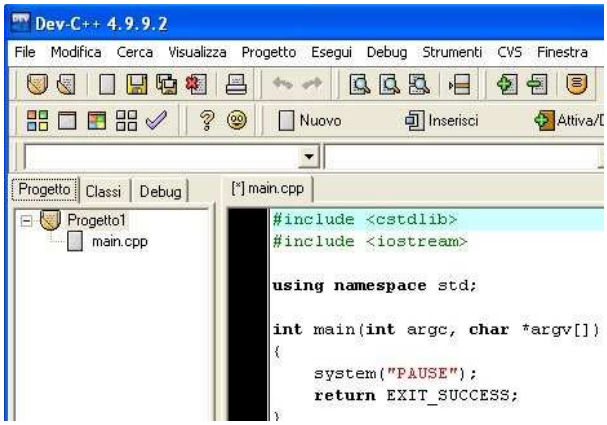


Successivamente, nella finestra che si apre selezionare il path (percorso) e attribuire un nome al file.



A questo punto abbiamo un nuovo progetto

Cliccando sul segno + nell'elenco dei progetti appare il file con estensione .cpp. Nella finestra centrale appare il codice.



Scrivere un file sorgente

La creazione di una applicazione implica la scrittura di un programma costituito da uno o più file sorgenti. Modificare il file sorgente nella parte destra dello schermo. Le istruzioni inserite automaticamente in fase di creazione di un progetto non sono generalmente adatte per un programma C++.

Lo scheletro di un programma C (applicazione MS-DOS):

```
#include <stdlib.h>
main()
{
system("PAUSE");
return 0;
}
```

Per generare le parentesi graffe { e } nelle tastiere italiane premere i tasti con le seguenti combinazioni :

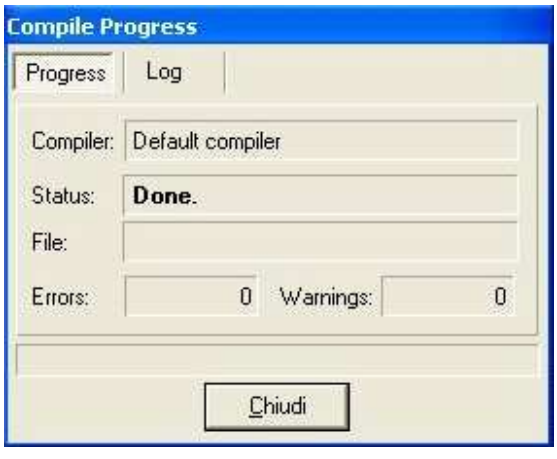
- <Alt> 1 2 3 genera {
- <Alt> 1 2 5 genera }
- <Alt> 1 2 6 genera ~
- ecc. (si invita a visionare la tabella del codice ASCII)

(per i numeri occorre premere i corrispondenti tasti sul tastierino numerico)

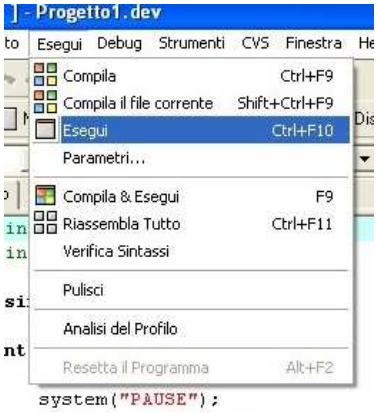
Dopo una modifica del file .cpp, salvare lo stesso file selezionando "File/Salva" o "File/Salva come" e, nel secondo caso, attribuire un nome, quindi "Salva".

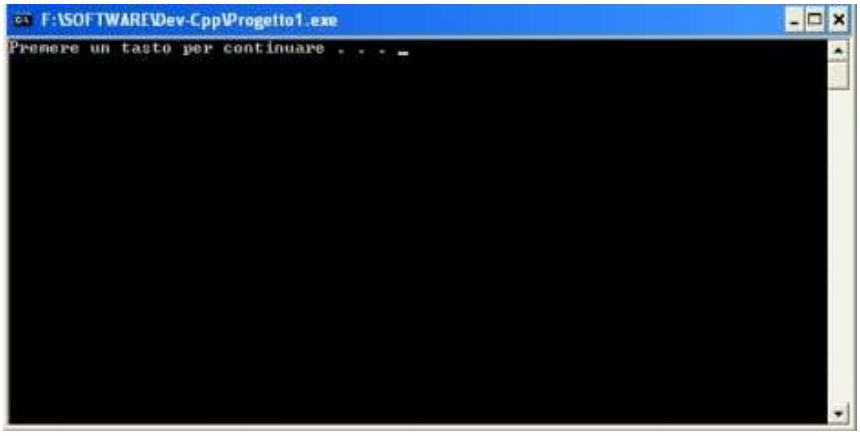
Il file .cpp è inserito da Dev-C++ nel progetto (si noti la parte destra dello schermo).

Abbiamo, ora, un nuovo progetto. Occorre compilarlo: "Esegui/Compila"



Ora eseguiamo il progetto: "Esegui/Esegui", con il risultato seguente (figura a destra).





In seguito, per ritornare sullo stesso progetto: "File/Apri progetto…" e selezionare il file ".dev" desiderato.

Cosa fa il programma di prova? Non molto…

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
system("PAUSE");
return 0;
}
```

Vengono richiamate due librerie standard: "Standard Input Output" e "Standard Library"

"Press any key to continue" permette di visualizzare l'output del programma prima che la finestra si chiuda

Si suggerisce di modificare il programma di prova nel modo seguente:

```
#include <iostream.h>
int main()
{
system("PAUSE");
return 0;
}
```

Il tuo primo programma

```
#include <iostream.h>
int main()
{
printf("Ciao Mondo!\n");
system("PAUSE");
return 0;
}
```

Ricompliamo e rieseguiamo: abbiamo ottenuto il nostro primo programma "hello world".

IL LINGUAGGIO *C++*

printf (istruzione di output)

`#include <iostream.h>`	Include la libreria **iostream.h** che contiene funzioni necessarie per le operazioni di I/O .
`main()`	Definiamo la funzione **main**, che non riceve nessun valore come argomento. Il programma inizierà l'esecuzione a partire da questa funzione, che deve essere presente in ogni programma.
{	Le istruzioni della funzione sono racchiuse tra **{** e **}**
`printf("Ciao Ciao!\n");`	1. Chiamiamo la funzione di libreria **printf** che si occupa di stampare. 2. **\n** indica il NewLine, ossia serve per andare a capo. 3. Occorre un punto e virgola alla fine di ogni istruzione.
}	Chiudiamo la funzione principale main()

In questo primo programma si notano già delle cose fondamentali:

- La prima istruzione scritta **#include <iostream.h>** è una direttiva ***(questa istruzione non termina con il punto e virgola ;)*** che corrisponde alla richiesta di caricamento del file (libreria) di nome iostream.h, il quale a sua volta contiene le funzioni necessarie alla gestione delle operazioni di ingresso e uscita (input e output). Questi file sono detti header, infatti terminano con il suffisso .h
- La funzione principale **main**(), deve essere sempre presente, in quanto è quella che viene eseguita per prima. Gli argomenti da passare alle funzioni vanno posti tra le parentesi ( ) che si trovano dopo il nome della funzione stessa. La funzione main() non richiedere argomenti, pertanto le parentesi tonde che seguono **main** vanno lasciate vuote, ().
- La funzione **printf**() ha come argomento una stringa di caratteri (il testo da stampare) racchiuso tra " ". Alla fine della stringa di caratteri abbiamo inserito **\n**, per indicare di andare a capo. (Sono disponibili altri ***caratteri speciali*** non editabili, ad esempio **\t** , **\"** , ..)
- I commenti possono essere inseriti tra i caratteri **/*** e ***/** (se interessano una sola riga con //
- Le istruzioni possono essere scritte in qualunque punto dello schermo e contenere dei blank (spazii vuoti) che verranno ignorati purché non interrompano le parole chiave o gli identificatori.

Dichiarazione delle variabili

Gli elementi di base di un programma sono le variabili e le costanti.
In C++ le variabili devono essere dichiarate prima di essere usate. Le variabili fondamentali che si possono definire sono di tipo: **char**, **int**, **float**, **double** e **void**. Ad esempio con **char** si definiscono variabili di tipo carattere, con **int** variabili che possono contenere solo numeri interi, con **float** e **double** variabili che possono contenere numeri reali. Il tipo **void** (vuoto) come si vedrà meglio in seguito si utilizza come argomento di funzioni senza argomenti (es. **main**) o per indicare il tipo di risultato (vuoto ovvero inesistente) restituito da funzioni che non producono risultati.

Esistono altri tipi di variabile ed altre classificazioni che dividono le variabili in globali, locali ecc.

Per dichiarare una variabile basta scrivere il tipo seguito dal nome che decidiamo di associare alla variabile stessa, ad esempio:

```
int c1;
```

In questo caso abbiamo dichiarato una variabile di tipo intero e di nome c1.
Si noti che abbiamo terminato la dichiarazione con un ";".
Il nome da dare alla variabile si chiama **identificatore** può essere scelto a piacere, rispettando alcune limitazioni.

Da notare che il C++ è case-sensitive ovvero distingue tra maiuscole e minuscole, per cui c1 è diverso da C1.

Le parole chiave come **printf**() DEVONO essere scritte in minuscolo, e solitamente anche le variabili sono scritte con caratteri minuscoli, nonostante si possano usare anche caratteri maiuscoli. La regola quindi sarebbe di scrivere le variabili a lettere minuscole, e se si vuole si può usare il carattere "_" come ad esempio:

```
int c1_prova;
```

Detto questo, vediamo come assegnare un valore ad una variabile:

```
c1 = 1000;
```

Abbiamo introdotto l'operatore di assegnamento =, che in questo caso assegna a c1 il valore 1000.
Si noti che:

```
c1 = 500+500;
c1 = (250+250)+500;
```

sono istruzioni equivalenti, infatti si può assegnare anche un'espressione.

printf (istruzione di output) – Stampare le variabili

Per stampare il valore di una variabile sul video con la funzione **printf**() occorre aggiungere **%d** nel punto che ci interessa, e mettere il nome della variabile dopo la chiusura delle " ", in questo modo:

```
printf("Il valore di c1 e' %d.",c1);
```

Al momento della stampa il **%d** sarà sostituito dal valore della variabile **c1**. Da notare la virgola che separa i doppi apici "" dalla variabile.

Esempio

`#include <iostream.h>`	Includiamo la libreria standard.
`int c1;`	Dichiaro **c1**, una variabile di tipo intero
`main()`	Funzione principale, eseguita per prima.
`{`	Inizio della funzione **main**()

`c1 = 1000;`
`printf("Il valore di C1 e'`
`%d.\n",c1);`
	Assegno a **c1** il valore 1000

`c1 = 1500;`
`printf("Il valore di C1 e'`
`%d.\n",c1);`
	Assegno a **c1** il valore 1500

}	Fine della funzione main()

Come risultato si ottiene:

Si noti che abbiamo usato **%d** per indicare che la variabile in questione è di tipo intero. Per stampare il valore di una variabile di tipo float, dovremmo mettere **%f**. Vedremo in seguito altre opzioni di stampa.
*Si noti che il C++ è un linguaggio case-sensitive, pertanto se scrivessimo **Printf** il compilatore darebbe un errore in quanto non troverebbe la funzione nella libreria iostream.h.*
Per dichiarare due variabili dello stesso tipo conviene scrivere:

```
int c1, c2;
```

anziché dichiarare singolarmente.

Per stampare più di una variabile numerica occorre mettere un **%d** o **%f** nel punto del testo che ci interessa, e mettere poi il nome della variabile dopo la stringa. Esempio:

```
printf("C1 = %d , C2 = %d\n", c1, c2);
```

Ossia si mette **%d** o **%f** dove interessa, e poi la lista delle variabili che li sostituiranno nello stesso ordine, separate da virgole.

Tipi di variabili

Le variabili elementari possono appartenere seguenti tipi base:

tipi base	char	int	float	double	void

a cui possono essere applicati i modificatori:

modificatori	*signed*	*unsigned*	*long*	*short*

il cui effetto serve a ridurre o aumentare la memoria a disposizione con conseguente diversa possibilità di rappresentazione. **I modificatori hanno effetto solo sui char e int.**

Tipi	Bit	Range
char	8	da -128 a 127
unsigned char	8	da 0 a 255
signed char	8	da -128 a 127
int	16	-32.768 a 32.767
unsigned int	16	da 0 a 65.535
signed int	16	-32.768 a 32.767
short int	16	-32.767 a 32.767
unsigned short int	16	da 0 a 65.535
signed short int	16	-32.768 a 32.767
long int	32	-2.147.483.648 a 2.147.483.647
unsigned long int	32	da 0 a 4.294.967.295
signed long int	32	-2.147.483.648 a 2.147.483.647
float	32	$1,2*10^{-38}..3,4*10^{38}$ prefisso 7 cifre
double	64	$1.3*10^{-308}.. 0.7*10^{308}$ prefisso 15 cifre
long double	80	$3.4*10^{-4932}$ a $1.1*10^{4932}$ prefisso 19 cifre

Come si vede **i tipi interi (char, int, short, long) possono essere signed e unsigned, in questo caso sono** solo positivi, pertanto contengono il doppio dei valori, in quanto si evita di rappresentare quelli negativi.

I tipi in virgola mobile, ossia float, double e long double, sono comunque signed.

Se si va fuori del range di definizione di una variabile, ad esempio se si aggiunge 100 ad una variabile int che vale già 32700, il risultato è -32736!

Questo a causa dell'aritmetica in complemento a 2, che per semplificare fa "ricominciare" dal numero più basso quando si supera quello più alto... dato che non è segnalato alcun errore e il risultato erroneo può essere usato, ATTENTI A NON FAR ECCEDERE MAI I TIPI, piuttosto definite dei long, se sospettate che un int non basti. Poiché l'occupazione in byte del tipo di dati dipende dal particolare sistema il C++ dispone dell'operatore **sizeof**.

Operatore sizeof

Restituisce in output la dimensione in byte del tipo di dato passato come parametro.
Es: **`sizeof(int)`** produce in output 2.

Costanti

Costanti intere

Una costante intera è un numero decimale (base 10), ottale (base 8) o esadecimale (base 16) che rappresenta un valore intero positivo o negativo.

- Un numero senza prefissi o suffissi è interpretato in base decimale e di tipo int. La prima cifra del numero non deve essere 0.

Un numero con prefisso 0 è interpretato in base ottale e di tipo int.
Es. **`a = 0100;`** (in **a** è memorizzato il numero 64)

Costanti floating-point

Una costante floating-point è un numero decimale (base 10), che rappresenta un valore reale positivo o negativo. Può essere specificato in 2 modi:

- parte_intera.parte_decimale (il punto è obbligatorio)
- notazione esponenziale (il punto non è obbligatorio)

Esempi: **`15.75`** **`-1.5e2`** **`25E-4`** **`10.`**

Costanti carattere

Una costante carattere è rappresentata inserendo fra singoli apici un carattere stampabile oppure una sequenza di escape.
Esempi:

`'A'`	carattere A
`'\n'`	carattere newline
`'\003'`	carattere cuoricino

In memoria un carattere occupa un byte ed è rappresentato da un numero intero di 1 byte (il suo codice ASCII).

Costanti stringa

Una costante stringa è rappresentata inserendo un insieme di caratteri (fra cui anche sequenze di escape) fra doppi apici (virgolette).
Es. **`"Ciao \n"`**
`'A'` è un carattere e occupa 1 byte (con il numero 65)
`"A"` è una stringa e occupa 2 byte (con i numeri 65 e 0)

IDENTIFICATORI

Gli identificatori sono nomi tramite i quali e' possibile fare riferimento a varibili, funzioni, etichette ed altri oggetti definiti dall'utente. Un identificatore è costituito da uno o più caratteri:

- il **primo** deve essere una **lettera o un underscore**
- i caratteri **successivi** possono essere **lettere, numeri o underscore**.

Gli identificatori possono avere qualsiasi lunghezza. Vengono riconosciute diverse le maiuscole dalle minuscole. Non possono essere usate le parole chiave che sono riservate

Parole riservate

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typdef
class	friend	return	union
const	goto	short	unsigned
continue	if	void	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

Si noti che quando si usa una parola riservata seguita da un identificatore va inserito almeno un blank in quanto diversamente non sarebbe possibile distinguere i due elmenti.

Esempio: **`int n`** (esatto) **`intn`** (errato)

Assegnamento di valori alle variabili

Ricordando che %d si usa per stampare una variabile int, mentre %f si usa per stampare una variabile float, facciamo un esempio pratico:

`#include <iostream.h>`	Includiamo la libreria standard.
`int c1, c2;`	Dichiaro 2 variabili di tipo intero
`float a,b;`	Dichiaro 2 variabili di tipo float
`main()`	Funzione principale, eseguita per prima.
`{`	Inizio della funzione **main**()
`c1 = 1000;`	Assegno a **c1** il valore 1000
`c2 = c1 + 500;`	Assegno a **c2** il valore di c1 + 500
`a = 21.06;`	Assegno ad a il valore 21,06
`b = 3.1415;`	Assegno a b il valore 3.1415
`printf("C1 = %d , C2 = %d\n", c1, c2);`	Stampo il valore di 2 variabili di tipo int con un solo printf()
`printf("A = %f , B = %f\n", a, b);`	Stampo il valore di 2 variabili di tipo float con un solo printf()
`printf("C1=%d,C2=%d,A=%f,B=%f\n",c1,c2, a,b);`	Stampo il valore di tutte le 4 variabili con un solo printf()
`}`	Fine della funzione main()

Si che in C++ le assegnazioni producono come risultato (oltre alla assegnamento vero e proprio) il valore che si vuole asegnare, pertanto è possibile effettuare assegnazioni multiple purche' le variabili siano dello stesso tipo. Ad esempio:

`#include <iostream.h>`	
`int somma;`	
`main()`	
`{`	
`int a,b,c=3;`	Equivalente a c=3; b=c; a=c.
`a=b=c;`	
`}`	

OPERATORI ARITMETICI

Gli operatori aritmetici fondamentali sono:
- sottrazione e meno unario
+ addizione
*** moltiplicazione**
/ divisione
% (modulo) resto della divisione intera (non puo' essere usato sui tipi float e double)

L'operatore "%" (modulo) può essere utilizzato solamente con le variabili di tipo integer;
L'operatore "/" e' utilizzato sia per la divisione tra interi che per i reali.

A proposito della divisione si osservi che `z=3/2` assegnerà a z il valore 1, anche se e' stato dichiarato come float in quanto di regola, se entrambi gli argomenti della divisione sono interi, allora verrà effettuata una divisione intera; per avere il risultato corretto sarà necessario scrivere:
`z = 3.0/2` oppure **`z = 3/2.0`** o, ancora meglio, **`z = 3.0/2.0`**

OPERATORI DI INCREMENTO/DECREMENTO

`++` incremento
`--` decremento

Come già accennato, le assegnazioni in C++ vengono effettuate utilizzando "=". Ma oltre agli operatori arimetici standard +,-,*,/ e all'operatore % (modulo) per gli interi, in C++ si hanno anche gli operatori incremento ++ e decremento -- (il cui effetto produce un incremento o decremento di 1 sulla variabile a cui si riferiscono) che possono essere preposti o posposti alla variabile. Se sono preposti (es: `++z`) l'incremento viene effettuato prima che sia valutata l'intera espessione, mentre se sono posposti il prima viene calcolato l'intera espressione e dopo viene effettuato l'incremento. Ad esempio:

```
int x,z = 2;
x=(++z)-1;
```

1)

A questo punto x = 2 e z = 3

```
    int x,z = 2;
2)  x=(z++)-1;
```

A questo punto x = 1 e z = 3

E' importante sottolineare che un'istruzione del tipo **x++** e' piu' veloce della corrispondente **x=x+1**

Priorita' degli operatori aritmetici

Le parentesi tonde si usano per costituire espressioni, come ad esempio:

```
c2 = (a-30+c1*(c++)/-(c));
c2 = (a-30+c1*(c2+b))/(c1*5);
```

ove si intende imporre un diverso ordine di priorità agli operatori.

Le parentesi quadre sono usate per l'indicizzazione degli array e le parentesi graffe la definizione di inizio e fine di una funzione.

Le espressioni vengono valutate in base alla priorità degli operatori. A parità di priorità vengono svolte da sinistra verso destra.
E' importante che le variabili e costanti che compaiono nell'espressione e la variabile cui viene assegnato il risultato siano tutte dello stesso tipo.
In caso negativo il compilatore provvede ad effettuare una conversione di tipo automatica.
Per maggior chiarezza si consiglia di esplicitare la conversione di tipo tramite l'operatore apposito cast.

Conversioni di tipo implicite

Il C++ esercita un forte controllo sui tipi e da messaggio di errore quando si tenta di eseguire operazioni fra operandi di tipo non ammesso.

Es. l'operatore % richiede che entrambi gli operandi siano interi.

I quattro operatori matematici si applicano a qualsiasi variabile di tipo base, ma i tipi dei due operandi devono essere uguali. Tuttavia, nel caso di due tipi diversi, il compilatore esegue una conversione di tipo implicita su uno dei due operandi, adeguando il tipo più semplice a quello più complesso, secondo la seguente gerarchia (in ordine crescente di complessità):

char - short - unsigned short - long - unsigned long - float - double

Es. nell'operazione `3.4/2` il secondo operando è trasformato in 2.0 e il risultato è correttamente 1.7

Conversioni di tipo esplicite – CASTING

Quando si vuole ottenere una conversione di tipo che non verrebbe eseguita implicitamente, bisogna usare l'operatore unario di casting (o conversione esplicita), che consiste nell'indicazione del nuovo tipo fra parentesi davanti al nome della variabile da trasformare.

Es. se la variabile n é di tipo int, l'espressione **`(float)n`** trasforma il contenuto di n da int in float. In C++ si può usare anche il formato funzione: **`float(n)`** é equivalente a **`(float)n`**

scanf (istruzione di input)

Abbiamo visto fin ora come definire e modificare e stampare delle variabili ma non ancora come immettere dei valori da tastiera durante l'esecuzione del programma.

Per leggere i dati dalla tastiera e memorizzarli nelle variabili si può utilizzare la funzione **scanf**(), il suo formato è simile al **printf**():

```
scanf("stringa di controllo", &variabile);
```

Se vogliamo leggere un numero nella stringa di controllo si deve mettere un **%d** se si tratta di intero e **%f** se si tratta di vuole un numero reale mentre **%c** se vogliamo leggere un carattere.

Attenzione! Oltre allo specificatore di formato **%d, %f** ecc., non si deve però mettere altro. Altri caratteri sarebbero interpretati come "caratteri da scartare" in lettura. Prima di ogni operazione di lettura da input conviene sempre stampare un messaggio:

```
printf("Immetti un numero\n");      (stampiamo il messaggio)
scanf("%d",&c1);                    (leggiamo il numero, e lo salviamo)
```

Si noti che, a differenza della **printf**, il nome della variabile deve essere preceduto dal carattere **&.**

Esempio:
programma che richiede una coppia di numeri, li somma e visualizza il risultato:

`#include <iostream.h>`	
`float a,b,c;`	Definisco 3 variabili di tipo float
`main()`	
`{`	
`printf("\nDammi il primo numero: ");`	
`scanf("%f", &a);`	Lettura e valore attribuito ad **a**
`printf("\nDammi il secondo numero: ");`	
`scanf("%f", &b);`	Lettura e valore attribuito a **b**
`c = a+b;`	Calcolo
`printf("\%f + %f = %f\n",a,b,c);`	Stampo risultato
`}`	

Si noti che è possibile limitare il numero delle cifre da stampare *modificando* **%f** ad esempio:

`%6.4f` stampa un numero con 6 cifre totali di cui 4 dopo il punto.
`%.4f` lunghezza totale non fissata ma 4 cifre dopo il punto.
`%2.0f` lunghezza totale di 2 cifre e 0 cifre dopo la virgola.

```
#include <iostream.h>
int a,b,c;
main()
{
 printf("\nDammi 2 numeri: ");
 scanf("%d%d",&a,&b);
 c = a+b;
 printf("\n%d + %d = %d\n",a,b,c);
}
```

Istruzioni di controllo

Iterazione	**for**	**while**	**do...while**		
Selezione	**if**	**if else**	**if else if**	**?**	**switch**
Salto	**return**	**break**	**continue**	**goto**	**exit**

Istruzione iterativa for

Ora vedremo come eseguire un ciclo, ossia come ripetere l'esecuzione di una istruzione per un numero determinato di volte.
Per prima utilizzaremo l'istruzione for, che ha questa forma:

`for (inizializzazione;condizione;incremento) <istruzione del ciclo>;`

- *inizializzazione* e' una espressione (es. e' una assegnazione);
- *condizione* e' una espressione usata per valutare la variabile o le variabili di controllo del loop; **l'esecuzione del ciclo prosegue fino a quando questa condizione e' vera**, quando diviene falsa viene eseguita l'istruzione successiva al for
- *incremento* e' una espressione che definisce il modo in cui la variabile di controllo cambia il suo valore ad ogni ripetizione del loop
- l'istruzione del corpo del ciclo può essere semplice o composta, inquesto caso consiste di un blocco di istruzioni comprese tra parentesi gaffe.