



# ELEMENTI DI INFORMATICA

DOCENTE: FRANCESCO MARRA

INGEGNERIA CHIMICA

INGEGNERIA ELETTRICA

SCIENZE ED INGEGNERIA DEI MATERIALI

INGEGNERIA GESTIONALE DELLA LOGISTICA E DELLA PRODUZIONE

INGEGNERIA NAVALE


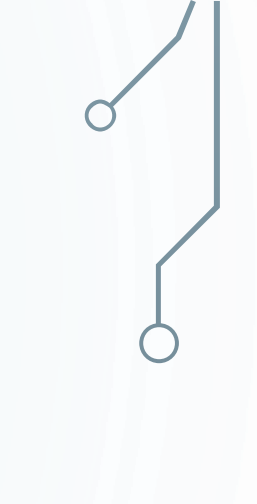
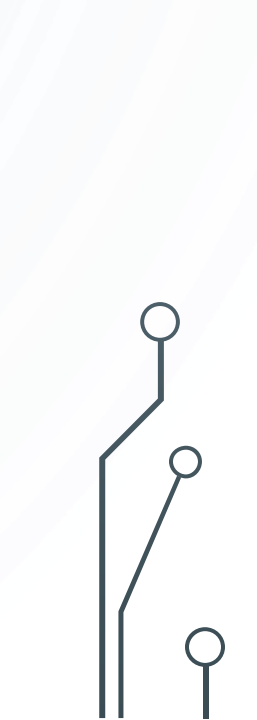
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II  
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE



# STRUTTURA DEI PROGRAMMI IN C



# AGENDA

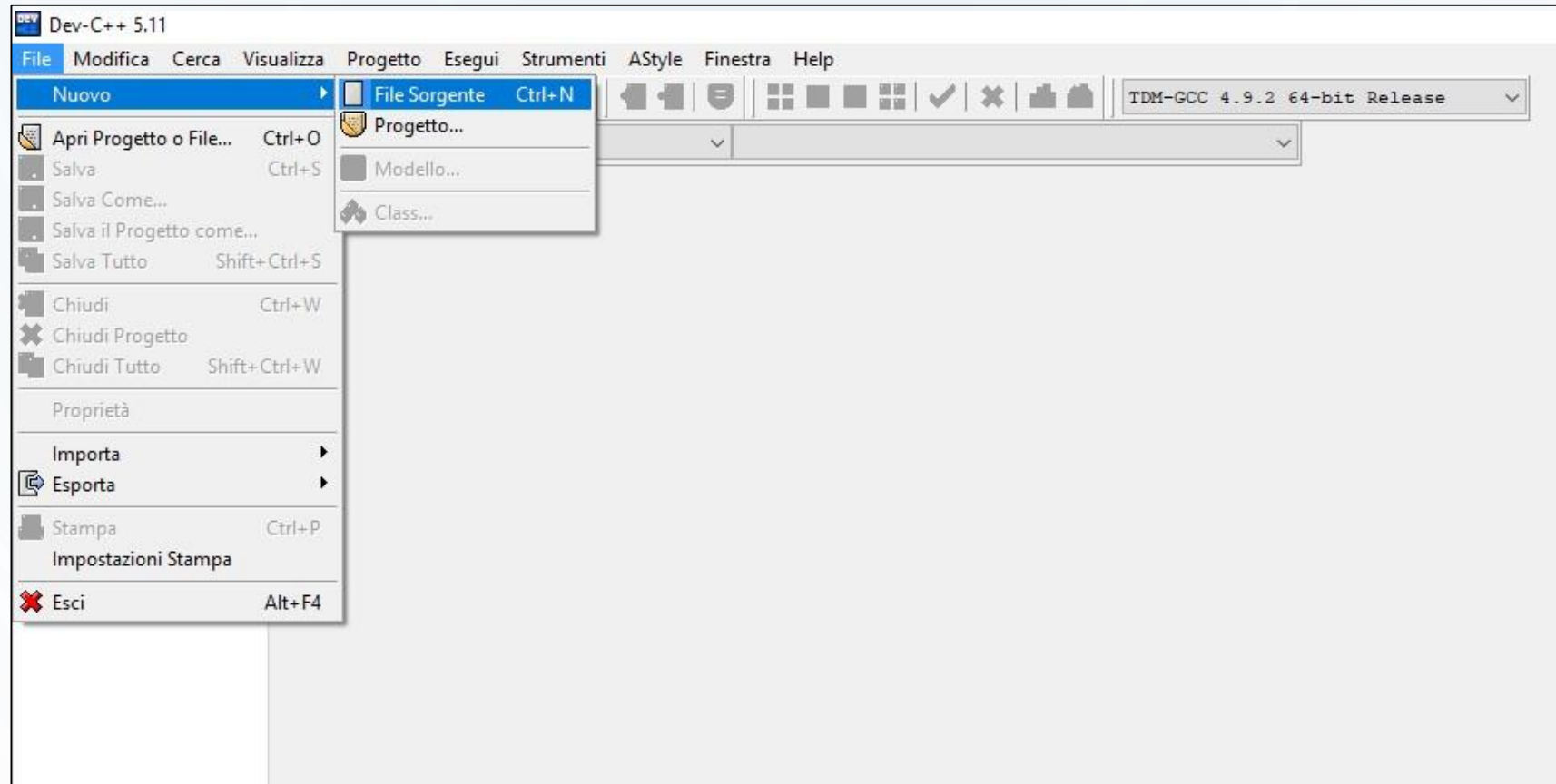
- Struttura dei programmi
  - Frasi in un linguaggio di programmazione
    - Commenti
    - Dichiarazioni
    - Istruzioni
    - Strutture di controllo
- 
- 
- 

# AMBIENTE DI SVILUPPO

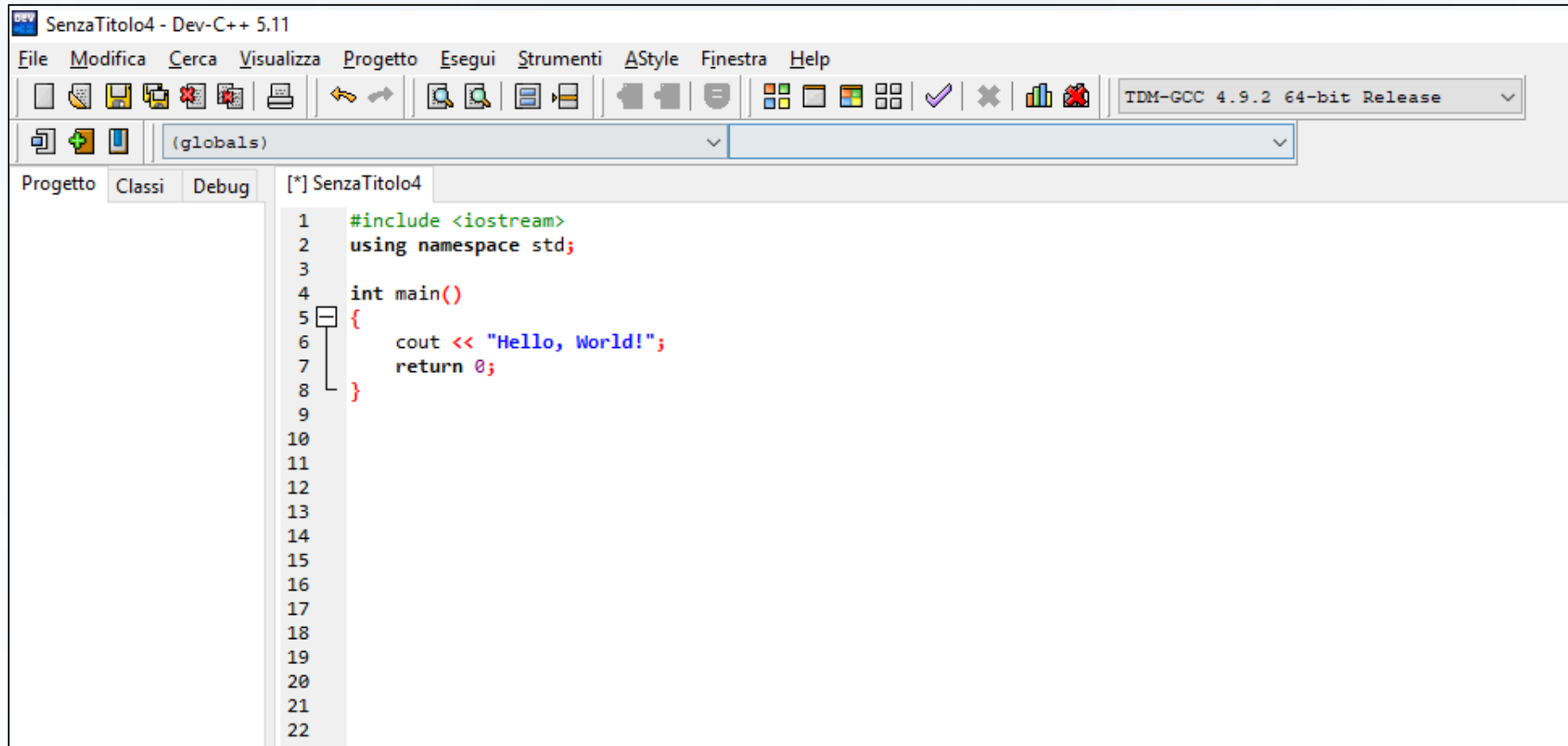
- Ambiente Windows
  - Orwell Dev C++
  - Scaricabile al sito: <http://sourceforge.net/projects/orwelldevcpp/>
- Ambiente Mac
  - Orwell Dev C++
    - Su S.O. Windows utilizzando Parallels Desktop o Boot Camp
  - Xcode
    - Nativo per Mac, scaricabile al sito:
    - <https://developer.apple.com/xcode/>



# ORWELL DEV C++



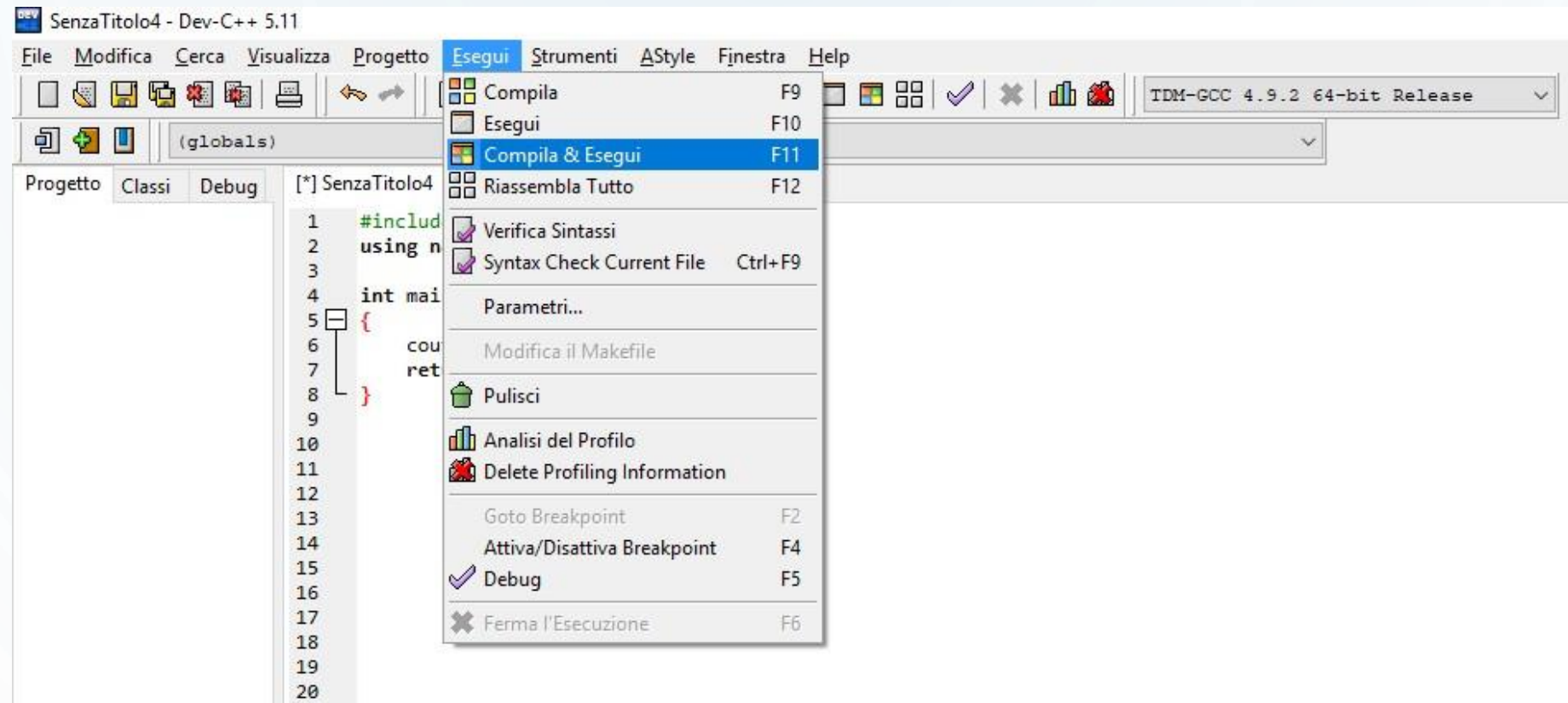
# ORWELL DEV C++



The screenshot displays the Dev-C++ 5.11 IDE interface. The title bar reads 'SenzaTitolo4 - Dev-C++ 5.11'. The menu bar includes 'File', 'Modifica', 'Cerca', 'Visualizza', 'Progetto', 'Esegui', 'Strumenti', 'AStyle', 'Finestra', and 'Help'. The toolbar contains icons for file operations, editing, and execution. The compiler is set to 'TDM-GCC 4.9.2 64-bit Release'. The 'Progetto' tab is active, showing a project named 'SenzaTitolo4'. The code editor displays the following C++ code:

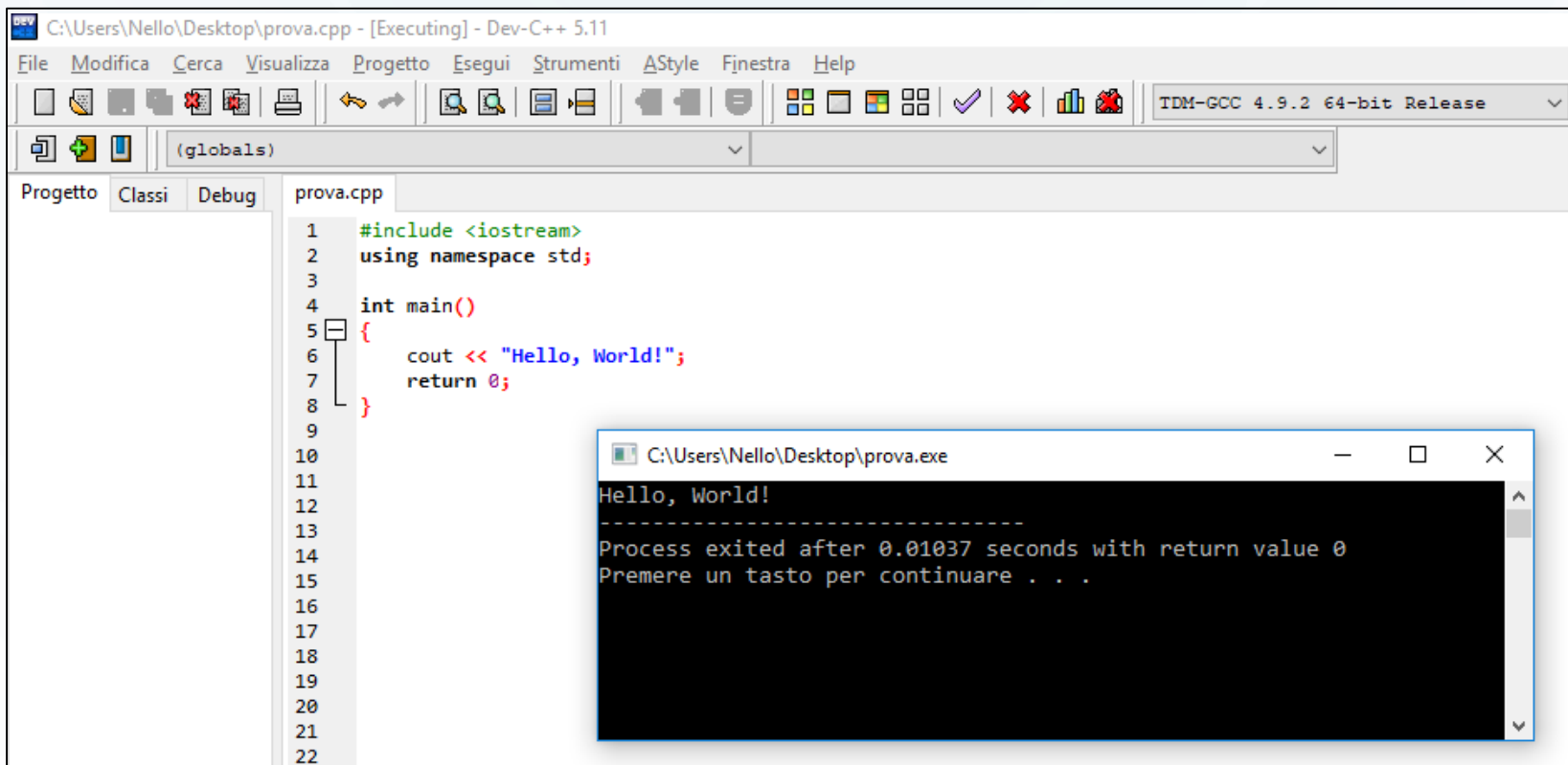
```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      cout << "Hello, World!";
7      return 0;
8  }
```

# ORWELL DEV C++



# ORWELL DEV C++

- ...dopo aver inserito il nome file!





# STRUTTURA DEI PROGRAMMI

- Intestazione
  - Commenti
  - Librerie
  - Namespace
  - Funzioni
- `main()`
  - Deve sempre essere presente
  - Due parti:
    - Definizione di elementi
      - Variabili, tipi, costanti
    - Istruzioni da eseguire
      - La prima istruzione della sequenza `main` è la prima ad essere eseguita

```
// Primo programma  
#include<iostream>  
using namespace std;
```

```
int main()  
{  
    int n;  
    cout<<"Inserisci n: ";  
    cin>>n;  
    cout<<"Hai inserito il numero "<<n;  
}
```

# LE FRASI DI UN LINGUAGGIO DI PROGRAMMAZIONE

Tutti i linguaggi di alto livello prevedono quattro tipologie di frasi diverse:

- Le **frasi di commento**
  - utili a rendere più comprensibili i programmi ad un lettore umano
- Le **dichiarazioni**
  - guidano il processo di traduzione
- Le **istruzioni** di calcolo, assegnazione o I/O
  - indicano al processore le operazioni da svolgere
- Le **strutture di controllo**
  - definiscono l'ordine di esecuzione delle istruzioni.

# FRASI DI COMMENTO

- Servono «solo» a migliorare la comprensione del programma da parte di un lettore

- Linguaggio naturale
- Non sono tradotte dal compilatore

- Tipi di commenti

- Documentazione (Cosa si sta facendo)
- Motivazione (Come si sta facendo)
- Asserzione (Proprietà o valori delle variabili)

- Commento di linea (solo C++)

// ...

- Commento su più righe

/\* ...

... \*/

```
/* primo programma di saluto */  
#include <iostream>  
using namespace std;  
  
char tast;  
  
int main ()  
{  
    cout << "Ciao sono io: il tuo Personal Computer." << endl;  
    cout << "Che desidera imparare tante cose nuove." << endl;  
    cout << "Devi solo progettare cosa devo fare." << endl;  
    cout << "Inventando nuovi programmi." << endl;  
    cout << "Forza datti da fare. Ci divertiremo.";  
  
    cout << endl << endl;  
    cout << "Per cortesia dammi un cenno di assenso." << endl;  
    cout << "Premi s e il tasto enter!";  
    cin >> tast;  
}
```

# LIBRERIE

- Per introdurre funzioni complicate definite in librerie
  - Es. `#include <iostream>`
- Per utilizzare linguaggio più compatto
  - Es. `using namespace std;`

# SEPARATORI

- Servono a distinguere le varie frasi o componenti di una frase
- Es.
  - carattere spazio
    - separatore implicito
  - punto e virgola
  - virgola
  - operatori
  - parentesi tonde, quadrate e graffe

```
/* primo programma di saluto */  
#include <iostream>  
using namespace std;  
  
char tasto;  
  
int main ()  
{  
    cout << "Ciao sono io: il tuo Personal Computer." << endl;  
    cout << "Che desidera imparare tante cose nuove." << endl;  
    cout << "Devi solo progettare cosa devo fare." << endl;  
    cout << "Inventando nuovi programmi." << endl;  
    cout << "Forza datti da fare. Ci divertiremo.";   
  
    cout << endl << endl;  
    cout << "Per cortesia dammi un cenno di assenso." << endl;  
    cout << "Premi s e il tasto enter!";  
    cin >> tasto;  
}
```

# IDENTIFICATORI

- Gli identificatori sono simboli definiti dal programmatore per riferirsi ad oggetti presenti nel codice:
  - Variabili
  - Costanti simboliche
  - Etichette
  - Tipi definiti dal programmatore
  - Funzioni

# IDENTIFICATORI

- Un identificatore deve iniziare con una lettera, o con il carattere di underscore “\_”, e può contenere un numero qualsiasi di lettere, cifre o underscore
  - viene fatta distinzione tra lettere maiuscole e minuscole (linguaggio di programmazione case sensitive)
- Tutti gli identificatori presenti in un programma devono essere diversi tra loro, indipendentemente dalla categoria cui appartengono

# IDENTIFICATORI

- **Nome** che si vuole attribuire ad una variabile
- Valido se composto di lettere, cifre e caratteri underscore ma che non inizia con una cifra

`<identificatore> ::= <lettera>|_ {<lettera>|<cifra>|_}`

- Il linguaggio è sensibile (case sensitive)
  - e.g. mela ≠ Mela
- Identificatori riservati
  - if, main, function, int, ...

```
/* primo programma di saluto */
#include <iostream>
using namespace std;

char tasto;

int main ()
{
    cout << "Ciao sono io: il tuo Personal Computer." << endl;
    cout << "Che desidera imparare tante cose nuove." << endl;
    cout << "Devi solo progettare cosa devo fare." << endl;
    cout << "Inventando nuovi programmi." << endl;
    cout << "Forza datti da fare. Ci divertiremo.";

    cout << endl << endl;
    cout << "Per cortesia dammi un cenno di assenso." << endl;
    cout << "Premi s e il tasto enter!";
    cin >> tasso;
}
```



# IDENTIFICATORI

- Le **variabili** sono contenitori di valori di un qualche tipo
  - il valore contenuto in una variabile può cambiare nel tempo
  - il tipo di valore che una variabile può contenere viene comunque stabilito una volta per tutte e non può cambiare
- Esempi
  - `char tasto`
  - `int n`
  - `bool scelta`

# IDENTIFICATORI

- Le **costanti simboliche** servono ad identificare dei valori che non cambiano nel tempo
  - non possono essere considerate dei contenitori, ma solo dei nomi associati a dei valori che restano costanti nel tempo

- Esempi

- `const char tasto = 'c';`
  - `const int n = 10;`
  - `const bool scelta = true;`
- Costanti esplicite

# IDENTIFICATORI

- Una **etichetta** è un nome il cui compito è quello di identificare una particolare istruzione del codice programma
  - sono utilizzate dalle istruzioni di salto incondizionato (goto)
  - ad esempio, si consideri l'identificatore stop nel seguente codice

```
if ( i == 5 )  
    goto stop;  
  
cout<<"Hello world 1"<<endl;  
  
stop: cout<<"Hello world 2";
```

# IDENTIFICATORI

- Un **tipo** identifica un insieme di valori e di operazioni definite sui valori dell'insieme
  - ogni linguaggio di programmazione, tipicamente, fornisce un certo numero di tipi primitivi (cui è associato un identificatore predefinito) e dei meccanismi per definire nuovi tipi a partire da questi
- Esempi
  - char tasto
  - int n
  - boolean scelta

# IDENTIFICATORI

- Una **funzione** è il termine che il C/C++ utilizza per indicare i sottoprogrammi
  - ad esempio, si consideri la funzione saluta nel codice seguente

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

void saluta() {
    cout<<"Hello world"<<endl;
}

int main() {
    saluta();
}
```

# IDENTIFICATORI

- Il C/C++, così come ogni linguaggio, si riserva delle parole chiave (keyword) il cui significato è prestabilito
  - non possono essere usate dal programmatore come identificatori

- Alcune parole riservate, vengono ereditate dal C:

auto	const	double	float	int	short	struct	unsigne
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatil
char	do	extern	if	return	static	union	while

- Le parole riservate esclusive del C++ invece sono le seguenti:

asm	dynamic_cast	namespace	reinterpret_cast	try
bool	explicit	new	static_cast	typeid
catch	false	operator	template	typename
class	friend	private	this	using
const_cast	inline	public	throw	virtual
delete	mutable	protected	true	wchar_t

# DICHIARAZIONE DI VARIABILE

- Un identificatore può essere usato solo dopo essere stato «definito»
  - Per le variabili, va definito il «tipo» con la dichiarazione

nome\_tipo identificatore\_di\_variabile

```
int a;
```

- Dichiarazioni di più variabili dello stesso tipo

```
int a, b, numeroIntero;
```

- Assegnazione valore iniziale

- con = oppure con ()

```
int a = 10;           // a = 10
int b(10);            // b = 10
int c(a+1);           // c = 11
```

# ASSEGNAZIONE DI VALORE

- istruzione che consente di assegnare il risultato di un'espressione ad una variabile

```
variabile = espressione;
```

- **Passi**
  - viene calcolato il risultato dell'espressione
  - il risultato viene assegnato alla variabile



# DICHIARAZIONE DI TIPI

- La dichiarazione di tipo di una variabile
  - Fissa i *valori* che essa può assumere
  - Fissa le *operazioni* che si possono fare con essa
- **Tipi atomici**
  - Il compilatore fissa caratteristiche e regole d'uso delle variabili ad esse associati
    - `int`, `float`, `double`, `bool`, `char`
- **Tipi non atomici**
  - Necessitano di costruttori di tipo con cui se ne definiscono le caratteristiche
    - Es. dimensione degli array, campi delle strutture, ecc.
    - `float v[10]`

# DICHIARAZIONE DI TIPI

- Dichiarazione `typedef`
  - Rende più *leggibile e modificabile* un programma perché consente di
    - Assegnare un nome ai tipi definiti dal programmatore
    - Assegnare un alias (nome alternativo) ai tipi predefiniti

```
// senza typedef
float a, b, x, y;

// con typedef
float a, b;
typedef float COORDINATA;
COORDINATA x, y;
```

# COSTANTI

- Valori prefissati non alterabili
  - Numerici o alfanumerici
- Costanti intere
  - Con segno (signed)
    - +300, -12, +12, 12, 1000
  - Senza segno (unsigned)
    - 12u, 1000u, 55u
  - Per le costanti unsigned, si possono adottare anche le notazioni
    - Ottale (anteponendo la cifra 0)
    - Esadecimale (anteponendo la sequenza 0x)

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

#define A 11u
#define B 011u
#define C 0x11u

int main ()
{
    cout << "A= " << A << endl; // restituisce 11
    cout << "B= " << B << endl; // restituisce 9
    cout << "C= " << C << endl; // restituisce 17
}
```

*ottale*

*esadecimale*

# COSTANTI

- Costanti reali

- In virgola fissa

- 9.55                      -1.34                      +0.0001

- In virgola mobile

- 0.2e-23                      15.001E-99                      0.1E10

- Costanti carattere

- Un carattere ASCII racchiuso tra una coppia di apici ' '

- 'A'      'a'      '1'      '\*'

- Costanti stringa di caratteri

- Sequenza di caratteri ASCII racchiusa tra una coppia di doppie virgolette ""

- "stringa"                      "due parole"                      "A"                      ""

- Caratteri speciali possono essere inseriti indicandoli con il carattere backslash

- Sequenze di escape

- \n \r      \'      \"      \?      \\      ...

# DICHIARAZIONE DI COSTANTI

- Migliora la leggibilità e la parametricità del programma
  - Manutenzione più semplice
- **Modo 1:** Direttiva `#define`

```
#define nome_costante valore_costante
```

- Al nome della costante non viene associata memoria
- Non deve essere terminata dal ;

- **Modo 2:** Prefisso `const`

```
const nome_tipo nome_costante = valore;
```

- Al nome della costante viene associata memoria
- Deve essere terminata dal ;

# TIPI ATOMICI FONDAMENTALI

- **Intero**
- **Reale**
- **Booleano o logico**
- **Carattere**
- Con essi si specifica:
  - l'insieme di valori che la variabile può assumere
    - dipendente dal tipo di occupazione in memoria prevista dal compilatore usato
  - le operazioni permesse su di essi
- Il tempo di calcolo di una espressione dipende fortemente dalla quantità di memoria coinvolta
  - usare le variabili che rappresentano la realtà da trattare con minore memoria usata rende il calcolo più efficiente

# TIPI ATOMICI FONDAMENTALI

- `sizeof(nome_tipo)`
  - restituisce la occupazione effettiva di memoria di un tipo
    - es. `sizeof(char) = 1` con un qualsiasi compilatore

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

int main()
{
    cout << "Size of int    = " << sizeof(int) << endl;
    cout << "Size of float = " << sizeof(float) << endl;
    cout << "Size of bool  = " << sizeof(bool) << endl;
    cout << "Size of char  = " << sizeof(char) << endl << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
Size of int    = 4
Size of float  = 4
Size of bool   = 1
Size of char   = 1
```

```
Premere un tasto per continuare . . .
```

# TIPI INTERI

Nome	Occ. di memoria	Intervallo valori rappresentati
<b>unsigned short</b> int	1 byte	0, 255 (tot. $2^8$ )
signed <b>short</b> int	1 byte	-128, 127 (tot. $2^8$ )
<b>unsigned int</b>	2 byte	0, 65535 (tot. $2^{16}$ )
signed <b>int</b>	2 byte	-32768, 32767 (tot. $2^{16}$ )
<b>unsigned long</b> int	4 byte	0, 4294967295 (tot. $2^{32}$ )
signed <b>long</b> int	4 byte	-2147483648, 2147483647 (tot. $2^{32}$ )

- Per la tabella si è fatto riferimento ad un compilatore operante su un sistema con una memoria organizzata in word da 16 bit (2 byte)
- In generale si assume che il tipo `int` abbia l'occupazione della word di memoria e che la rappresentazione `short`, `int` e `long` abbia una ampiezza ( $r$ ) che soddisfi la seguente condizione:

$$r_{\text{short}} \leq r_{\text{int}} \leq r_{\text{long}}$$

- N.B.
  - Se si indicano `short` o `long` si può omettere `int`
  - Se non si specificano `unsigned` o `signed` si assume che il tipo sia `signed`

```
Size of unsigned short int = 2
Size of signed short int = 2

Size of unsigned int = 4
Size of signed int = 4

Size of unsigned long int = 4
Size of signed long int = 4
```

Caso memoria in word da 64 bit...  
... su SO Windows 7

L'API di Windows definisce il tipo `long` come un intero di 4 byte. Questo vale indifferentemente sui sistemi a 16, 32 e 64 bit per ragioni di retrocompatibilità



# TIPI REALI

- Anche per il tipo reale la rappresentazione in memoria dipende dallo specifico compilatore
- In generale si assume però che
  - la precisione di `float` sia minore o uguale a quella di `double`
  - la precisione di `double` sia minore o uguale a quella di `long double`
- Es. dichiarazione di variabili reali
  - `float alpha;`
  - `double beta;`

# TIPO BOOLEANO

- Le variabili di tipo booleano assumono uno dei due valori di verità
  - TRUE
  - FALSE
- Occupano un solo byte di memoria e si dichiarano tramite la parola chiave **bool**
  - Es. `bool cond;`

# TIPO CARATTERE

- Con char si dichiarano variabili contenenti uno dei caratteri della tabellina ASCII

```
char carattere = 'A';
```

- Le variabili di tipo carattere occupano un solo byte
- Il linguaggio consente anche di operare direttamente sul codice del carattere considerando la variabile come un intero

valori tra 0 e 255 se dichiarata `unsigned char`

valori tra -128 e 127 se dichiarata `signed char`

# TIPO CARATTERE

- se alla variabile **v** di tipo char viene assegnato un valore intero **x**, alla variabile **v** viene assegnato il carattere avente posizione **x** nella tabella ASCII
  - `char a = 100; → a = 'd';`
- se alla variabile **v** di tipo char viene assegnata la somma aritmetica tra un valore intero **x** e un carattere **c** avente posizione **y** nella tabella ASCII, alla variabile **v** viene assegnato il carattere avente posizione **(x+y)** nella tabella ASCII
  - `char b = 'f' + 1; → b = 'g';`
  - `char b = 'f' - 32; → b = 'F';`

# TABELLA ASCII

0		32		64	@	96	`	128	Ç	160	á	192	Ł	224	Ó
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┐	225	ß
2	☹	34	"	66	B	98	b	130	é	162	ó	194	└	226	Ë
3	♥	35	#	67	C	99	c	131	â	163	ú	195	┌	227	Ò
4	♦	36	\$	68	D	100	d	132	ä	164	ñ	196	─	228	ö
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	+	229	Õ
6	♠	38	&	70	F	102	f	134	á	166	ª	198	ã	230	μ
7	·	39	'	71	G	103	g	135	ç	167	º	199	Ä	231	þ
8	■	40	(	72	H	104	h	136	ê	168	¿	200	ℒ	232	ƒ
9	○	41	)	73	I	105	i	137	ë	169	®	201	℞	233	Ú
10	◼	42	*	74	J	106	j	138	è	170	¬	202	ℓ	234	Û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	℣	235	Ü
12	♀	44	,	76	L	108	l	140	î	172	¼	204	℥	236	Ý
13	♪	45	-	77	M	109	m	141	í	173	⅓	205	═	237	Ÿ
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	≠	238	ˉ
15	☼	47	/	79	O	111	o	143	Å	175	»	207	◻	239	˘
16	▶	48	0	80	P	112	p	144	É	176	◼	208	◊	240	-
17	◀	49	1	81	Q	113	q	145	æ	177	◼	209	⊖	241	±
18	↑	50	2	82	R	114	r	146	Æ	178	◼	210	⊕	242	—
19	!!	51	3	83	S	115	s	147	ø	179		211	⊗	243	¾
20	¶	52	4	84	T	116	t	148	ö	180	└	212	⊘	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	┌	245	§
22	─	54	6	86	V	118	v	150	ú	182	Â	214	┐	246	÷
23	↑	55	7	87	W	119	w	151	û	183	Ã	215	└	247	,
24	↑	56	8	88	X	120	x	152	ÿ	184	©	216	┌	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	≡	217	└	249	˙
26	→	58	:	90	Z	122	z	154	Ü	186		218	└	250	·
27	←	59	;	91	[	123	{	155	ø	187	¶	219	■	251	¹
28	└	60	<	92	\	124		156	£	188	¶	220	■	252	²
29	↔	61	=	93	]	125	}	157	∅	189	¢	221	!	253	³
30	▲	62	>	94	^	126	~	158	×	190	¥	222	ì	254	■
31	▼	63	?	95	_	127	△	159	f	191	└	223	■	255	

# ENUMERAZIONE

- Operatore `enum`

- consente di dichiarare un nuovo tipo elencando i valori costanti che lo compongono

```
enum colori {bianco, rosso, verde} A;  
enum colori B;
```

A e B sono  
variabili del tipo  
`enum colori`

- Il compilatore associa ad ogni costante il valore intero corrispondente alla posizione occupata
    - bianco=0, rosso=1, verde=2
  - Possono anche essere esplicitati i valori interi

```
enum colori {bianco=4, rosso=5, verde=6};
```

# ENUMERAZIONE

- Dichiarare un tipo `enum`

- Per dichiarare un nuovo tipo `enum`

```
typedef enum {bianco, rosso, verde} colori;  
colori A, B;
```

- Un tipo enumerato è totalmente ordinato (Integral Type)

- Su un dato di tipo enumerato sono applicabili tutti gli operatori relazionali:

- `bianco < rosso`  $\rightarrow$  vero
      - `rosso >= verde`  $\rightarrow$  falso

- Non c'è controllo sugli estremi dell'intervallo di interi utilizzato!

- `A = 15; //viene accettato ed eseguito!`

- L'utilizzo di tipi ottenuti per enumerazione rende più leggibile il codice

# INPUT E OUTPUT TRAMITE CONSOLE

- Per inserire un valore da assegnare a una variabile:
  - `cin >> var;`
- Per produrre in output:
  - `cout << var << '\\n';`
  - `cout << var << endl;`
- Per riuscire a vedere in tempo la console, alla fine del programma:
  - `system("pause");`



# CONVERSIONE DI TIPO

- In una istruzione di assegnazione l'espressione deve essere di tipo «compatibile» con quello della variabile
  - Sono tipi compatibili:
    - I tipi numerici se il valore che si assegna può essere contenuto nella variabile che lo riceve
- **Coercizione (type casting)**
  - Si possono imporre conversioni di tipo con due modalità

```
variabile = (tipo)espressione;  
variabile = tipo(espressione);
```

- Esempio:

```
int    i;  
float x = 5.11;  
  
i = (int)x;  
cout << "i=" << i << endl;
```



i = 5

# OPERATORI

- Aritmetici

+ - \* / %

- Logici

&& (AND) || (OR) ! (NOT)

- Bitwise

- Operano sui singoli bit della rappresentazione

& (AND) | (OR) ^ (XOR) ~ (NOT) << >> (SHIFT)

- Relazionali

- Restituiscono TRUE se il confronto tra due valori ha successo, FALSE se fallisce

== != > < >= <=

# OPERATORI

- **Incremento e decremento unario**

- **Prefisso:** incremento/decremento prima dell'uso della variabile

`++i`   `--i`

- **Postfisso:** prima si usa la variabile, poi la si incrementa/decrementa

`i++`   `i--`

- **Composti**

`var = var op espressione;`

`var op = espressione;`

**Esempio:** `b=b*4` **equivalente a** `b*=4`

# OPERATORI

- Condizionale ?

- Restituisce il primo risultato se la condizione (di tipo logico) è TRUE, il secondo se la condizione è FALSE

`c = a < b ? b : a;`

- Virgola

- Consente di elencare più espressioni in un secondo membro di un'istruzione di assegnazione
  - solo il valore dell'espressione più a destra viene assegnato alla variabile

`a = (b=10, c=b+10, c+10);`

- Precedenza operatori

- In una espressione con diversi operatori vengono eseguite per prime le operazioni con priorità più elevata (si veda tabella sul libro di testo)
- In presenza di operatori con egual priorità si stabilisce di procedere da sinistra verso destra
- L'introduzione di parentesi consente di cambiare l'ordine di esecuzione delle istruzioni

**DOMANDE, DUBBI, PERPLESSITÀ**

