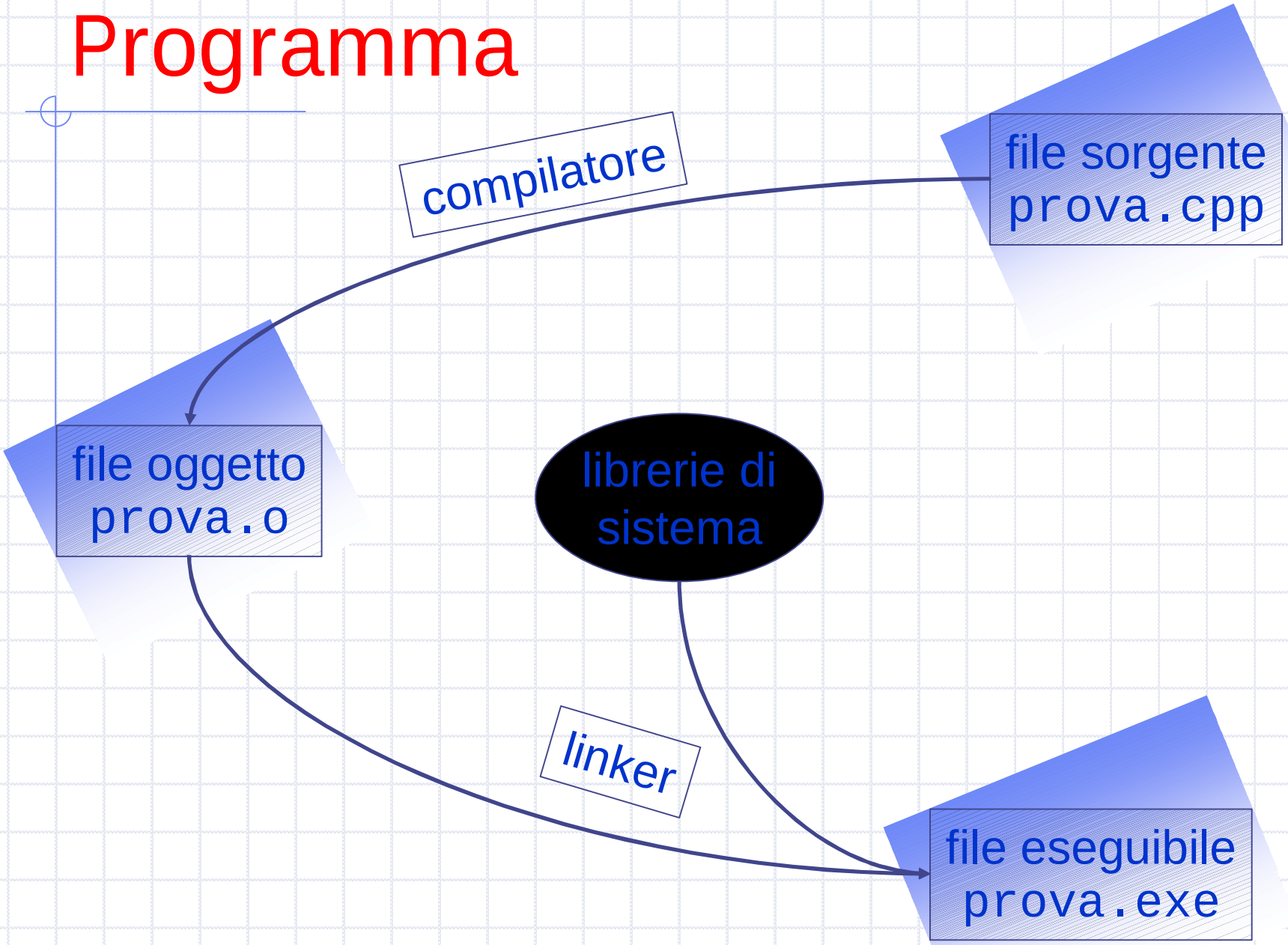
C++ - Lezione I

- Ciao!
- Compilazione di un programma
- Tipi e variabili
- Operatori
- Cicli e costrutto if
- cout e cin
- esercizi

Testi

- Stroustrup, Bjarne. The C++ Programming Language, 3rd Edition, Addison-Wesley, 1997. ISBN 0-201-88954-4.
- Lippman, Stanley B.; Lajoie, José C++ Primer, 3rd edition, Addison-Wesley, 1998. ISBN 0-201-82470-1.
- Bruce Eckel; Thinking in C++, 2nd edition, 2Prentice Hall, 2000 ISBN 0-13-979809-9, scaricabile in rete
  - http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Compilazione di un Programma



Ciao!

HelloWorld.cpp

```
#include <iostream>

using namespace std;

int main() {
// commento

/* commento che puo' essere scritto su
piu' linee */

cout<< "ciao!"<<endl;

return 0;
}
```

direttiva preprocessore

commenti

tipo e valore di ritorno

- Note
  - Il C++ e' case sensitive
  - istruzioni separate da ;
  - cout e' l'OGGETTO che rappresenta lo stdout
  - endl end of line
  - << operatore di inserimento

compiliamo e runniamo HelloWorld.cpp

g++ HelloWorld.cpp -o runHelloWorld
./runHelloWorld

Compilazione

- g++ e' il compilatore della GNU
- compilazione del file main
  - g++ main.cpp –o runMain
- creazione dell'oggetto libreria
  - g++ -c mylib.cc –o mylib.o
- Link delle librerie al main
  - g++ main.cpp mylib.o –o runme
  - g++ main.cpp *.o –o runme

Esempio di Compilazione

definiamo una funzione

saluti.h

```
#include <iostream>

void ciao();
```

```
g++ -c saluti.cc
g++ -c test.cpp
g++ -o runme saluti.o test.o
./runme

oppure :
g++ -o runme saluti.cc test.cpp
```

- un altro po' di sintassi
  - void non ritorna nessun tipo
  - \n fine di linea

implementiamo la funzione

saluti.cc

```
#include "saluti.h"

using namespace std;


void ciao ( ) {
        cout << " ciao ! \ n " ;
}
```

usiamo la funzione

test.cpp

```
#include "saluti.h"

int main() {

        ciao();

        return 0;
}
```

Make e Makefile

- Per automatizzare la compilazione dei sorgenti lo strumento standard make
- In ogni directory che contiene sorgenti e' necessario creare un file di testo, chiamato *Makefile*, che contiene le istruzioni per la compilazione
- Una volta creato il Makefile sufficiente lanciare il comando make (o make bersaglio)
- make controlla se qualcuno fra i *prerequisiti* stato modificato piu' recentemente del *bersaglio*. In caso affermativo esegue i *comandi*.

Sintassi delle regole

```
bersaglio: prerequisiti
        comando1
        comando2
        ...
```

Spiegazione

bersaglio	Il file da creare
prerequisiti	I file che servono per crearlo
comando	I comandi (si noti il *tab* obbligatorio) per creare il *bersaglio*

Makefile

```
CXXFLAGS = -Wall -W -pedantic

test.o : test.cc
        g++ $(CXXFLAGS) -c test.cpp

saluti.o : test.cc
        g++ $(CXXFLAGS) -c saluti.cc
```

Make e Makefile II

un esempio piu' complesso :

```
CXXFLAGS = -Wall -W -pedantic

SOURCES          := $(wildcard *.cc)
OBJECTS          := $(SOURCES:.cc=.o)
EXESRC           := $(wildcard *.cpp)
EXECUTABLES      := $(EXESRC:.cpp=.exe)

all: $(EXECUTABLES)

%.exe : %.cpp $(OBJECTS)
        g++ $(CXXFLAGS)  $< $(OBJECTS)  -o  $@

%.o : %.cc
       g++ $(CXXFLAGS) -c $< -o $@

clean :
       rm -f *.o
       rm -f *.exe
```

g++ flags

definizione dei sorgenti, file oggetti ed eseguibili

definizione dei comandi
make all
e
make clean

$< primo prerequisito
$@ bersaglio

cancelliamo e ricompiliamo:
make clean
make all
./test.exe

Torniamo al C++ : concetto di tipo

- Il "*tipo*" rappresenta un insieme di elementi all'interno del quale può essere scelto un valore.
  - Per es. il *tipo intero* raggruppa in un insieme tutti i numeri interi.
- Un *tipo semplice* contiene un'informazione unica, non divisibile, cioè, in più informazioni.
- Un *tipo strutturato* contiene un'informazione divisibile in più sotto-informazioni, ciascuna di queste, a loro volta, di tipo semplice o strutturato.
  - Per es.:
    - il tipo *numero intero* è un tipo semplice;
    - il tipo *numero complesso* è un tipo strutturato;
    - il tipo *studente universitario* è un tipo strutturato.
- Ogni linguaggio introduce una serie di *tipi predefiniti*
- il C++ ha la possibilità di costruire nuovi tipi per le specifiche esigenze.
  - Questa possibilità è una delle basi della programmazione orientata agli oggetti.

Tipi predefiniti

```
#include <iostream>

using namespace std;

int main()
{
cout << "Dimensione (in byte) dei tipi di dato:" << endl;
cout << "bool: " << sizeof(bool) << endl;
cout << "char: " << sizeof(char) << endl;
cout << "short: " << sizeof(short) << endl;
cout << "int: " << sizeof(int) << endl;
cout << "long: " << sizeof(long) << endl;
cout << "float: " << sizeof(float) << endl;
cout << "double: " << sizeof(double) << endl;
 return 0;
}
```

possono essere unsigned

la funzione sizeof() restituisce il numero di bytes di RAM occupati

Operatore sizeof

- L'operatore *`sizeof`*, può essere prefisso a:
  - una variabile (esempio: `sizeof x`)
  - una costante (esempio: `sizeof 'a'`)
  - al nome di un tipo (esempio: `sizeof double`)
- Restituisce un intero rappresentante la dimensione in byte
- È applicabile a qualsiasi tipo (non solo ai tipi fondamentali)

Definizione e "Scope" delle Variabili

- La dichiarazione (o allocazione) di una variabile avviene con la seguente sintassi:
  - tipovar nomevar;
  - Per es., per dichiarare una variabile di tipo intero di nome *i*, si scrive:
    - `int i;`
  - `i` è un'*istanza* del tipo intero.
  - all'atto della sua dichiarazione viene allocata dal cal colatore un'area di memoria di 16 bit
- Le dichiarazioni di variabili sono valide antro un certo scope
  - Lo scope tipicamente è definito dal blocco {...} nel quale si dichiara la variabile

```
int main() {

  int i;
  {
  int k = 5;
  }
  i = k; //errore!!!!!
  return 0;
}
```

```
int main() {

  int i = 5;
  if(i>0){
      int k = 0;
  }
  i = k; //errore!!!!!
  return 0;
}
```

"Variabili" Costanti

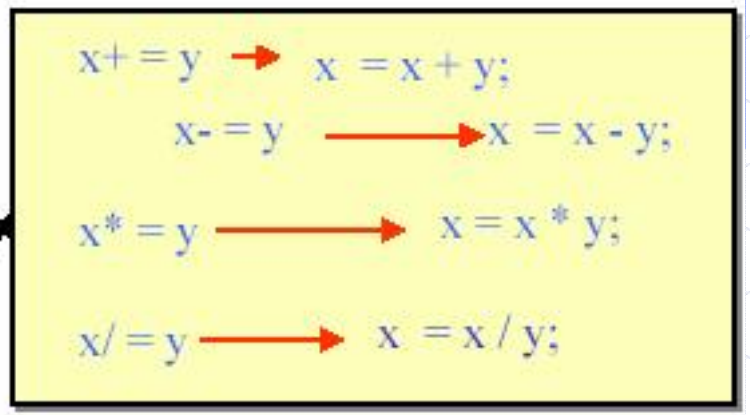
- In molti casi è utile assegnare a degli identificatori dei valori che restino costanti durante tutto il programma e che non possano essere cambiati nemmeno per errore
- Per questo si usa la parola chiave const

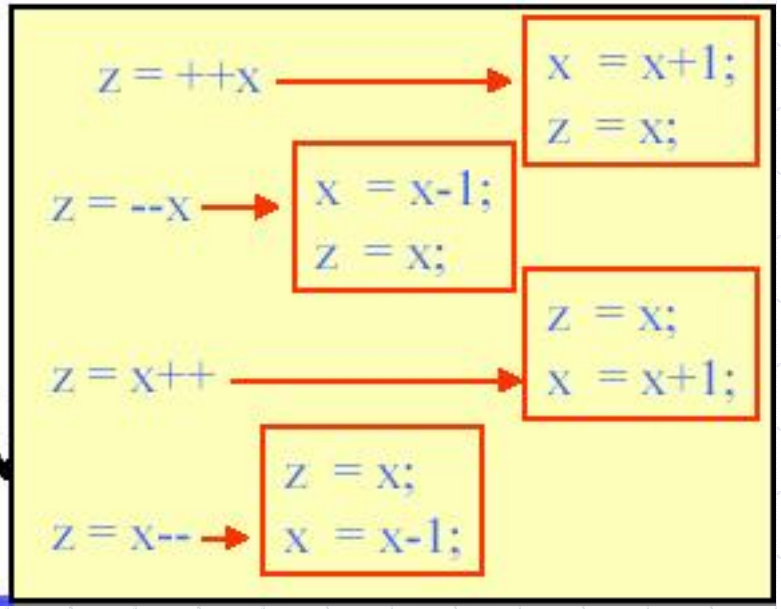
Esempio:

```
const double pi = 3.14159;
```

Operazioni e Operatori

x+y x-y x*y x/y	Le quattro operazione base
x=y x+=y x-=y x*=y x/=y	Operatori di assegnazione. Sono operatori che permettono di assegnare il valore di una variabile ad un'altra o di alterarne il valore.
x++ ++x x-- --x	Incrementi e decrementi. Questi operatori permettono di incrementare o decrementare il contenuto di una variabile.





Operazione Assegnazione

- L'operatore di assegnazione (=) puo' essere utilizzato:
  - Contestualmente alla dichiarazione
  - Differito rispetto a questa
  - E' possibile assegnare un valore attraverso il costruttore.

```
int c=0;

float d;
d=3.5;

double t(4.56);
```

Operatori Relazionali

- && (AND)
- ¦¦ (OR)
- ! (NOT, negazione)
- == (uguaglianza)
- != (disuguaglianza)
- < <= (minore di, minore o uguale di)
- > >= (maggiore di, maggiore o uguale di)

Controllo del Flusso : Cicli

```
while (condizione) {
      …
}
```

```
int c = 0;
while (c<10) {
      cout << c++ <<
endl;
}
```

```
do {
      …
} while(condizione);
```

```
int c = 0;
do {
      cout << c++ << endl;
} while(c<10);
```

```
for (istr_1; condizione; istr_n) {
      cout << c << endl;
}
```

```
for (int c=0; c<10; c++) {
      cout << c << endl;
}
```

Controllo del Flusso : if...else

```
if(condizione){ istruzioni }
```

```
if(condizione1){
 ...
} else if (condizione2) {
 ...
} else {
 ...
}
```

esempi:

```
if(a>0){
 b=a;
} else if (a<0) {
 a++;
} else {
 a--;
}
```

```
if(a>0){
 b=a;
} else {
 b=-a;
}
```

piu' compatto →

```
b = a>0 ? a : -a;
```

Controllo del Flusso : break e continue

- Mentre si e' all'interno di un loop e' possibile saltare una parte dell'istruzione con l'istruzione continue
- oppure interrompere bruscamente l'intera esecuzione del loop con l'istruzione break

```
/* calcolo la radice quadrata di numeri inseriti da tastiere */
#include <iostream>
#include <math.h>
using namespace std;
int main() {
double radicando,radice;
while (1) {
std::cin >> radicando;
if (radicando< 0.0) break; \\ oppure : if (radicando< 0.0) continue;
radice=sqrt(radicando);
cout<<"La radice di "<< radicando << " è " << radice<< endl;
}
return 0;
}
```

Funzioni Matematiche

- In C++ non esistono funzioni predefinite

```
#include <iostream>
#include <math>
int main(){
  double r, theta, phi;
  cin >>r>>theta>>phi;
  double x =  r * sin(theta) * sin(phi);
  double y =  r * sin(theta) * cos(phi);
  double z =  r *cos(theta);

  cout <<x <<“,”<<y<<“,”
       <<z<<endl;
  return 0;
}
```

cmath.h definisce sin, cos, ...

Alcune Funzioni Matematiche

Nella libreria <stdlib.h>

abs(n)	valore assoluto
rand()	numero pseudocasuale tra 0 e la costante RAND_MAX
srand(n)	inizializza la funzione rand

Nella libreria <math.h>

fabs(x)	valore assoluto di tipo float
sqrt(x)	radice quadrata di x
pow(x,y)	eleva x alla potenza di y (x^y)
exp(x)	eleva e alla potenza di y (e^x)
log(x)	logaritmo naturale di x
log10(x)	logaritmo in base 10 di x
sin(x) e asin(x)	seno e arcoseno trigonometrico
cos(x) e asin(x)	coseno e arcoseno trigonom.
tan(x) e atan(x)	tangente e arcotangente trig.

Gli Oggetti cout e cin

- Per utilizzare gli oggetti cin e cout è necessario includere nel programma la libreria che li definisce
  - #include <iostream.h>
- Sono istanze di una classe definita nella libreria di sistema iostream.h
- cout
  - Per l'output di un dato verso lo standard output (video)
  - La sintassi da utilizzare è la seguente:
    - cout << espressione;
  - Il simbolo << è l'operatore di invio verso la periferica di uscita
- cin
  - Per l'input di un dato dallo standard input (tastiera)
  - La sintassi da utilizzare è la seguente:
    - cin >> variabile;
  - Il simbolo >> è l'operatore di invio alla variabile ciò che proviene da cin.
  - Quando il programma deve eseguire un'operazione `cin`, esso si blocca in attesa di un input dall'utente.

Gli Oggetti cout e cin - Esempi

definizione di una variabile di tipo intero

inserisce lo standard input nella variabile

stampa la variabile

```
#include <iostream>

using namespace std;

int main()
{
  int i;
  cout << "Inserire il un numero:" << endl;
  cin >> i ;
  cout << "Il un numero inserito e': " << i << endl;
  return 0;
}
```

é anche possibile concatenare gli operatori >> e <<

cout << espressione1 << espressione2;

cin >> var1 >> var2;

Esercizi

1. Calcolare il fattoriale di un intero assegnato dallo standard input

2. Trovare le radici dell'equazione $ax^2+bx+c=0$
   - a, b e c sono assegnati dallo standard input
   - aiuto : la fuzione radice di x è
     - sqrt(x)

Backup

Concetti di Base: Il Pre-Compilatore

Il pre-compilatore esegue delle direttive che gli vengono impartite dal programmatore e modifica il listato prima di passarlo al compilatore vero e proprio. Una direttiva destinata al pre-compilatore e' sempre preceduta dal simbolo #

Con frequenza incontreremo due tipi di direttive:

```
#include "nome di un file"
o
#include <nome di un file>
e
#define   PI   3.14159
```

La prima direttiva dice al pre-compilatore di ***includere*** un file. La differenza tra le due sintassi sta nel fatto che nel primo caso il file viene cercato prima nella directory corrente e successivamente in quelle di sistema. Nel secondo caso, invece, avviene il viceversa.

La seconda direttiva definisce una ***macro*** (PI).

In tutti i punti del codice sottostante in cui comparira' la ***macro*** questa verra' sostituita dal suo valore.

Alcune opzioni di g++

Queste opzioni segnalano in fase di compilazione delle probabile sviste del programmatore che non danno errori in compilazione.
È consigliabile usarle sempre.

Warnings	
-Wall	Abilita molti avvisi utili (variabili usate senza inizializzazione, classi polimorfe senza distruttore virtuale, ecc.)
-pedantic	Avvisa se si devia dallo standard ISO
-W (-Wextra)	Abilita altri avvisi (confronto di unsigned con 0, funzioni che possono o meno restituire un valore, ecc.)

Tipi Fondamentali e Derivati

- Nel C++ i tipi sono distinti in
  - tipi fondamentali o predefiniti, che servono a rappresentare informazioni semplici, come i numeri interi o i caratteri (`int`, `char`, ...)
  - tipi derivati, che permettono di costruire strutture dati complesse, a partire dai tipi fondamentali (puntatori, array, ...)

Il tipo `int`

- il tipo `int` è costituito dai numeri interi compresi tra due estremi, a seconda dell'implementazione
- con N bit impiegati per la rappresentazione e assumendo una codifica in complemento a 2, si ha:

N	Min (-2^{N-1})	Max ($2^{N-1}-1$)
16	-32768	32767
32	-2147483648	2147483647

I tipi `short` e `long`

- Il tipo `short` tipicamente non ha più di 16 bit
  Esempio:

```
short x=1232;
short int x=-132;
```

- Il tipo `long` tipicamente ha almeno 32 bit
  Esempio:

```
long y=-34213332;
long int x=-6767899;
```

Il tipo unsigned

- Il tipo unsigned rappresenta numeri interi non negativi di varie dimensioni

- Esempi:

```
unsigned int x=1232;
unsigned short int x=567;
unsigned long int x=878678687;
```

Tipi reali: `float` e `double`

- I tipi reali hanno come insieme di valori un sottoinsieme dei numeri reali, ovvero quelli rappresentabili all'interno del computer in un formato prefissato
- Ne esistono tre tipi a seconda della precisione:
  - `float`
  - `double`
  - `long double`
- La precisione di ciascun tipo è maggiore o uguale a quella del tipo precedente

Tipo carattere

- Il tipo `char` ha come insieme di valori i caratteri di stampa (es. `'a'`, `'Y'`, `'6'`, `'+'`)
- Il tipo `char` è un sottoinsieme del tipo `int` che generalmente un carattere occupa 1 byte
- Il valore numerico associato ad un carattere è detto codice e dipende dalla codifica utilizzata dal computer (es. ASCII, EBCDIC, BCD, ...)
- La codifica più usata è quella ASCII ma il tipo `char` è indipendente dalla particolare codifica adottata

```
/* calcoliamo un fattoriale  */
#include <iostream>
int main(void)
{
        int k=0, fact=1;
        while (++k <= 10) {
                fact*=k;
        }
        std::cout<<“fact= ”<<fact<<std::endl;
        return 0;
}
```