C++: miscellaneous

Docente: Ing. Edoardo Fusella
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione
Via Claudio 21, 4° piano – laboratorio SECLAB
Università degli Studi di Napoli Federico II
e-mail: edoardo.fusella@unina.it

Indentazione

Indentazione

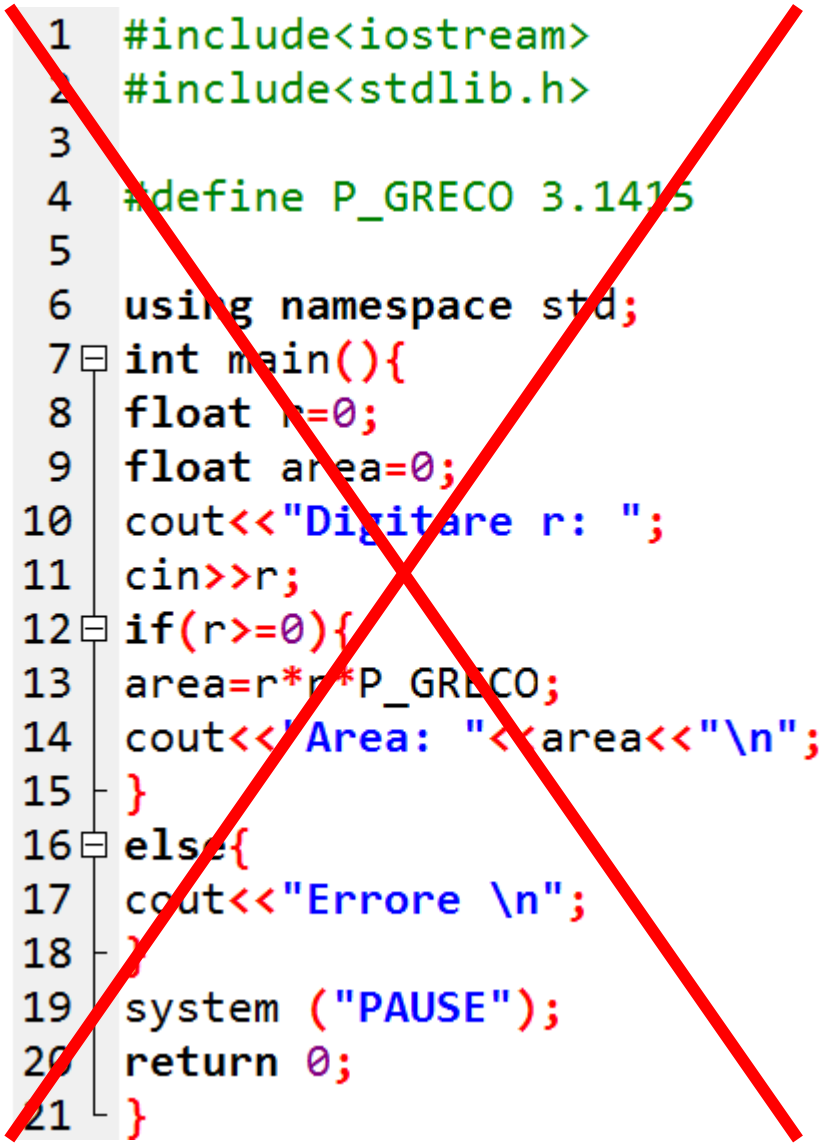
- “Indentare” significa:
  - Disporre il testo del programma in modo da evidenziare la struttura delle istruzioni composte
  - Disporre il testo del programma in modo da facilitare il compito di chi legge il codice scritto da noi
  - L’indentazione non serve al compilatore ma al programmatore

Come si indenta?

- L'**indentazione** è l'inserimento di una certa quantità di spazio vuoto all'inizio di una riga di testo.
- Consiste nell'anteporre a ogni istruzione una quantità di spazio bianco (**tabulatori**) proporzionale al numero di strutture di controllo o blocchi a cui tale istruzione appartiene.
  - Il **tabulatore**, o tasto **TAB** (Tab ⇆), è un tasto presente nella maggior parte delle tastiere per computer la cui funzione è quella di aggiungere un certo numero di spazi vuoti prima del cursore per indentare.

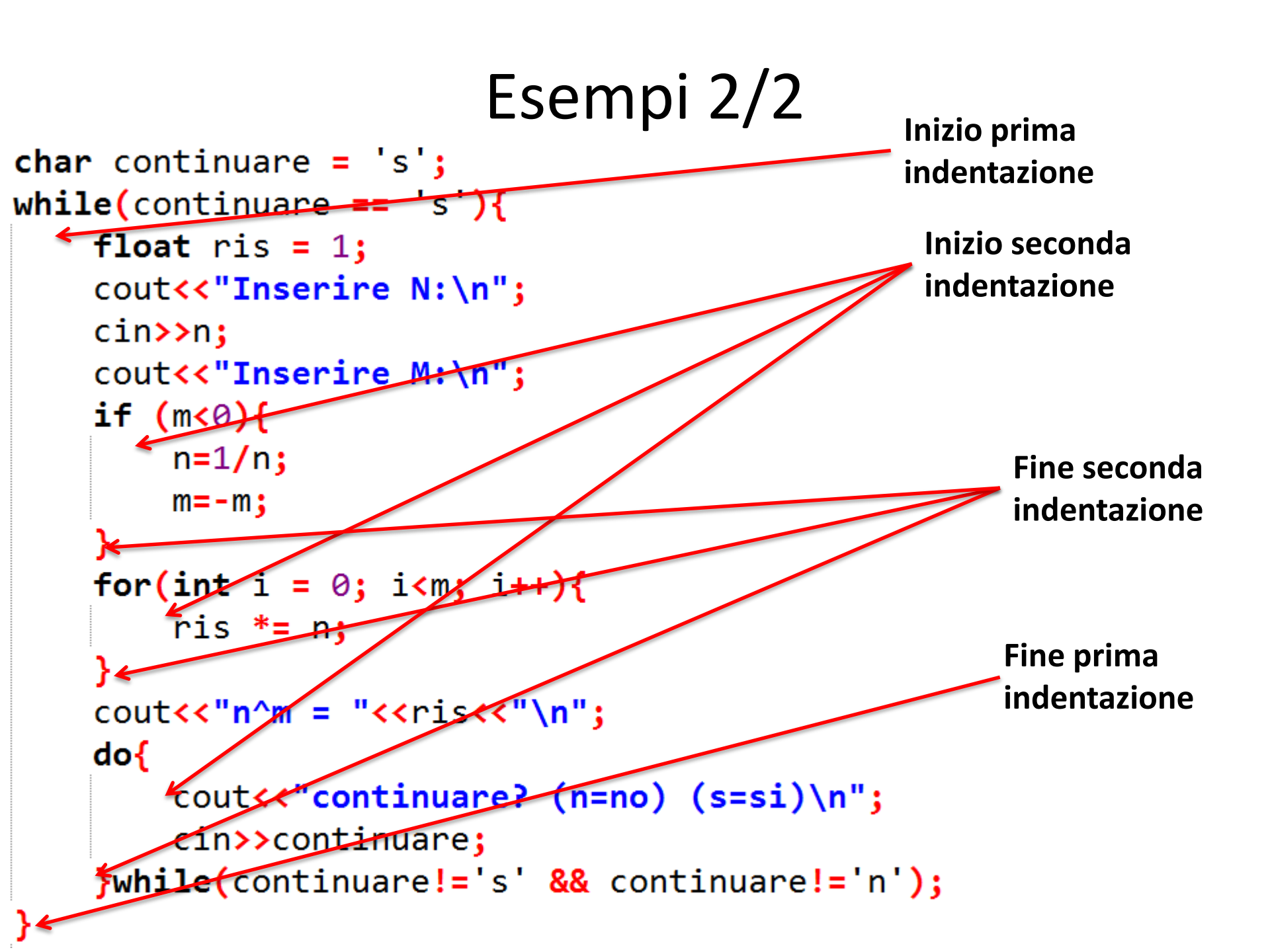
Esempi 1/2

```
1  #include<iostream>
2  #include<stdlib.h>
3
4  #define P_GRECO 3.1415
5
6  using namespace std;
7  int main(){
8      float r=0;
9      float area=0;
10     cout<<"Digitare r: ";
11     cin>>r;
12     if(r>=0){
13         area=r*r*P_GRECO;
14         cout<<"Area: "<<area<<"\n";
15     }
16     else{
17         cout<<"Errore \n";
18     }
19     system ("PAUSE");
20     return 0;
21 }
```



Esempi 2/2

```
char continuare = 's';
while(continuare == 's'){
    float ris = 1;
    cout<<"Inserire N:\n";
    cin>>n;
    cout<<"Inserire M:\n";
    if (m<0){
        n=1/n;
        m=-m;
    }
    for(int i = 0; i<m; i++){
        ris *= n;
    }
    cout<<"n^m = "<<ris<<"\n";
    do{
        cout<<"continuare? (n=no) (s=si)\n";
        cin>>continuare;
    }while(continuare!='s' && continuare!='n');
}
```



Visibilità

Il concetto di blocco e annidamento

- Un blocco è composto da una serie di istruzioni sintatticamente racchiuse tra parentesi graffe.
- All'interno di un blocco di istruzioni è possibile inserire sia istruzioni semplici sia ulteriori blocchi che vengono detti annidati.

```
if (condizione1) {
    /* blocco if */
    if (condizione 2) {
        /* blocco if annidato
          */
    } else {
        /* blocco else annidato
          */
    }
} else {
    /* blocco else */
}
```

Regole di visibilità

- Una variabile è visibile (e quindi utilizzabile) sono all'interno del blocco in cui viene dichiarata e nei blocchi annidati al suo interno.
- Le variabili vengono create ogni volta che si entra nel loro ambito di visibilità e vengono distrutte ogni volta che se ne esce.
- I loro valori non vengono mantenuti in memoria all'esterno del proprio ambito di visibilità.

- Le variabili globali sono variabili dichiarate nella parte dichiarativa globale, cioè prima del main, e sono visibili in tutto il programma.

Buona programmazione

- Usate sempre le parentesi graffe nel caso dei costrutti di selezione e iterazione (if, for, while, do..while).

- Non dichiarare variabili globali se non strettamente necessario.
- Dichiarare una variabile solo all'interno del blocco in cui vi serve in modo tale che venga distrutta al termine del blocco.
- Non dichiarare variabili che non usate.

Tipici errori:

- Cercare di usare una variabile dove non è visibile

Esempio

- n, x: visibili in tutto il file
- a,b,c,y: visibile in tutto il main
- d,z: visibile sono nel blocco verde

```
int n;
double x;
main()
{
    int a,b,c;
    double y;
    {
        int d;
        double z;
    }
}
```

Conversione

Conversioni tra tipi

- Date due variabili è possibile trasferire il contenuto di una variabile nell'altra. L'operazione avviene senza perdita di informazione se le due variabili sono del medesimo tipo.
- Se tuttavia presentano tipi differenti, l'operazione di trasferimento **può** generare perdita di informazione.
- Questo comportamento del linguaggio C/C++ dipende dal fatto che il trasferimento di informazioni tra variabili di tipo differente comporta un processo di conversione (**casting**).
  - Implicita - eseguite automaticamente dal compilatore
  - Esplicita - eseguite esplicitamente dal programmatore

Conversione implicita

```
1. #include<iostream>
2. #include<stdlib.h>

3. using namespace std;
4. int main(){
5.    int i    = 1 + 1.5;  // 2
6.    float f = 1 + 1.5; // 2.5
7.    int j     = i + f;      // 4
8.    int k    = 1 + 0.8; // 1
9. }
```

Conversione esplicita

- Si usa l'operatore **()** con all'interno il tipo di destinazione

```
1. #include<iostream>
2. #include<stdlib.h>

3. using namespace std;
4. int main(){
5.    int i         = 5;                  // 5
6.    float f       = 1.5;                // 1.5
7.    float ris1 = i/f;                   // 5/1.5 = 3.33
8.    float ris2 = i/(int)f;              // 5/1 = 5
9.    float ris3 = (int)(i/f);            // tron (5/1.5) = 3
10.   double d =  -53686781; // -53686781
11.   float g      = (float) d;           // -53686780
12.}
```

Altri errori

Utilizzo di una variabile prima che assuma un valore

SI

```
int somma=0;
for(int i=0; i<n; i++){
    somma += v[i];
}
```

```
int n=10;
int v[n];
```

```
int n=10;
int m=10;
int mat[n][m];
```

NO

```
int somma;
for(int i=0; i<n; i++){
    somma += v[i];
}
```

```
int n;
int v[n];
```

```
int n;
int m;
int mat[n][m];
```

Va prima riempito il vettore

```
int n;
cout<<"inserire N: ";
cin>>n;
int v[n];
int somma=0;
for(int i=0; i<n; i++){
    somma += v[i];
}
```

Dividere la parte di calcolo da quella di stampa dei risultati

- La stampa dei risultati deve avvenire separata dalla parte di calcolo, spesso nella parte finale del programma.
- La parte centrale del programma effettua tutti i calcoli e memorizza i risultati in apposite variabili (semplici o strutturate) che andranno stampate in seguito.