



Corso di Laurea in Informatica (*Track B*) - A.A. 2018/2019

Laboratorio di Informatica

Stile di Programmazione

docente: Veronica Rossano

veronica.rossano@uniba.it

Stile di Programmazione

```
char*d,A[9876];char*d,A[9876];char*d,A[9876];char*d,A[9876];char
e;b;*ad,a,c; te;b;*ad,a,c; te;*ad,a,c; w,te;*ad,a, w,te;*ad, and, w,te;*ad,
r,T; wri; ;*h; r,T; wri; ;*h; r; wri; ;*h;_, r; wri;*h;_, r; wri;*har;_, r; wri
;on; ;l ;i(V) ;on; ;l ;i(V) ;o ;l ;mai(V) ;o ;mai(n,V) ;main (n,V)
{-!har ; {-!har ; {har =A; {h =A;ad =A;read
(0,&e,o||n -- +(0,&e,o||n -- +(0,&o||n ,o-- +(0,&on ,o-4,- +(0,n ,o-=94,- +(0,n
,l=b=8,! ( te-*A,l=b=8,! ( te-*A,l=b,! ( time-*A,l=b, time)|-*A,l= time(0)|-*A,l=
~l),srand (1),~l),srand (1),~l),and ,!(1),~l),a ,!(A,1),~l) ,!(d=A,1),~l)
,b))&&+((A + te,b))&&+((A + te,b))+((A -A+ te,b))+A -A+ (&te,b+A -A+(* (&te,b+A
)=+ +95>e?(& c)=+ +95>e?(& c) +95>e?(& _*c) +95>(*& _*c) +95>(*&r= _*c) +95>
5,r+e-r +_ :2-195,r+e-r +_ :2-195+e-r +_ :2-1<-95+e-r +_-1<-95+e-r ++?-1<-95+e-r
|(d==d),!n ?*d||(d==d),!n ?*d||(d==d),!n ?*d||(d==d),!n ?*d||(d==d),!n ?*d||(d=
*( (char)**)+V+ *( (char)+V+ *( (c),har)+V+ (c),har)+ (V+ (c),r)+ (V+ ( c),
+0,*d-7 ) -r+8)+0,*d-7 -r+8)+0,*d-c:7 -r+80,*d-c:7 -r+7:80,*d-7 -r+7:80,*d++-7
+7+! r: and%9- +7+! rand%9-85 +7+! rand%95 +7+!! rand%95 +7+ rand()%95 +7+ r
-(r+o):( +w,_+ A-(r+o)+w,_+( A-(r+o)+w,_+ A-(r+o)+wri,_+ A-(r+o)
+(o)+b)),!write+(o)+b,!wri,(te+(o)+b,!write+(o_=)+b,!write+(o)+b,!((write+(o)+b
-b+*h)(1,A+b,!!-b+*h),A+b,(! !-b+*h),A+b,!!-b+*h),A-++b,!!-b+*h)
, a >T^1,( o-95, a >T,( o-=+95, a >T,( o-95, a )) >T,( o-95, a >T,(w? o-95, a >T
++ &&r:b<<2+a ++ &&b<<2+a+w ++ &&b<<2+w ++ ) &&b<<2+w ++ &&b<<((2+w ++
&& !main(n*n,V) , !main(n,V) , !main(+n,V) ,main(+n,V) ),main,(n,
1)),w= +T-->o +1)),w= +T>o +1)),w=o+ +T>o +1,w=o+ +T>o;{ +l,w=o+T>o;{ +l,w &=o+
!a;}return _+= !a;}return _+= !a;}return _+= !a;}return _+= !a;}
```

Stile di Programmazione



Stile di Programmazione



The International Obfuscated C Code Contest

[[The judges](#) | [IOCCC home page](#) | [How to enter](#) | [FAQ](#) | [Mirrors](#) |
[IOCCC news](#) | [People who have won](#) | [Winning entries](#)]

The [source code](#) for the winners of the 24th IOCCC has been released.

Please see the following news items.

Goals of the Contest

Obfuscate: tr.v. -cated, -cating, -cates.

1. a. To render obscure.
 b. To darken.
2. To confuse: his emotions obfuscated his judgment.
[Lat. obfuscare, to darken : ob(intensive) + Lat. fuscare, to darken < fucus, dark.] -obfuscation n. obfuscatory adj]

The IOCCC:

- To write the most Obscure/Obfuscated C program within the rules.
 - To show the importance of programming style, in an ironic way.
 - To stress C compilers with unusual code.
 - To illustrate some of the subtleties of the C language.
 - To provide a safe forum for poor C code. :-)
-

<http://www.ioccc.org/>

Stile di Programmazione

Alcuni tra i partecipanti dell'edizione 2018 del contest internazionale di offuscamento del codice.

<http://www.ioccc.org/years.html#2018>

Veronica Rossano - Stile di Programmazione Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

A noi interessa il processo opposto!

Scrivere del codice quanto **più chiaro, leggibile ed auto-esplorativo possibile.**

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Martin Fowler

Martin Fowler is a British software developer, author and international public speaker on software development, specialising in object-oriented analysis and design, UML, patterns, and agile software development methodologies, including extreme programming.

[https://en.wikipedia.org/wiki/Martin_Fowler_\(software_engineer\)](https://en.wikipedia.org/wiki/Martin_Fowler_(software_engineer))

Stile di Programmazione - Motivazioni

Quando un programma è comprensibile?

Stile di Programmazione - Motivazioni

Quando un programma è comprensibile?

- per la macchina
 - è comprensibile se può essere **compilato**

Stile di Programmazione - Motivazioni

Quando un programma è comprensibile?

- per la macchina
 - è comprensibile se può essere **compilato**
- per il programmatore
 - è comprensibile **se è chiaro e semplice**, cioè:
 - Ha una logica immediata
 - Usa espressioni vicine al linguaggio adottato dall'uomo
 - Usa forme convenzionali
 - Usa nomi **significativi**
 - Usa una **formattazione pulita**
 - Ha **commenti significativi**, che aiutano la comprensione

Stile di Programmazione - Nomi

Uno stile di programmazione appropriato deve necessariamente partire da una scelta appropriata **dei nomi**

Stile di Programmazione - Nomi

Uno stile di programmazione appropriato deve necessariamente partire da una scelta appropriata **dei nomi**

- **Come si sceglie un nome?**

- Il nome di una funzione o di una variabile porta informazione sul suo scopo

Stile di Programmazione - Nomi

Uno stile di programmazione appropriato deve necessariamente partire da una scelta appropriata **dei nomi**

- **Come si sceglie un nome?**

- Il nome di una funzione o di una variabile porta informazione sul suo scopo

- **Caratteristiche di un buon nome**

- Informativo (chiarisce lo scopo)
- Conciso
- Mnemonico
- Pronunciabile

Stile di Programmazione - Nomi

	SI	NO
Informativo	Media_BMI_over_40	Media2
Conciso	Media_BMI_over_40	Media_degli_individui_over
Mnemonico	Media_BMI_over_40	AvgBMIO40
Pronunciabile	Media_Peso_Over_40	AVGW

Stile di Programmazione - Nomi

- Più ampio è l'ambito (scope) di un nome, **maggioré è l'informazione che deve essere convogliata dal nome**

Stile di Programmazione - Nomi

- Più ampio è l'ambito (scope) di un nome, **maggiorè è l'informazione che deve essere convogliata dal nome**
- Più una variabile è «importante» all'interno di un programma, più attenzione bisogna riporre nel definire il nome!

Stile di Programmazione - Nomi

- Più ampio è l'ambito (scopo) di un nome, **maggioré è l'informazione che deve essere convogliata dal nome**
- **Più una variabile è «importante» all'interno di un programma, più attenzione bisogna riporre nel definire il nome!**
 - Una variabile contatore ha uno scopo molto limitato, quindi anche un nome poco esplicativo (`int i=0, int k=0`) può andare bene

Stile di Programmazione - Nomi

- Più ampio è l'ambito (scope) di un nome, **maggiorè è l'informazione che deve essere convogliata dal nome**
- **Più una variabile è «importante» all'interno di un programma, più attenzione bisogna riporre nel definire il nome!**
 - Una variabile contatore ha uno scope molto limitato, quindi anche un nome poco esplicativo (`int i=0, int k=0`) può andare bene
 - Variabili più complesse (es. `float media, int massimo`, etc.) hanno bisogno di nomi che siano auto-explicativi

Stile di Programmazione - Nomi

- Più ampio è l'ambito (scope) di un nome, **maggiorè è l'informazione che deve essere convogliata dal nome**
- Più una variabile è «importante» all'interno di un programma, più **attenzione bisogna riporre nel definire il nome!**
 - Una variabile contatore ha uno scope molto limitato, quindi anche un nome poco esplicativo (**int i=0, int k=0**) può andare bene
 - Variabili più complesse (es. **float media, int massimo**, etc.) hanno bisogno di nomi che siano auto-explicativi
 - Nomi di funzioni o **typedef** con cui definiamo nuovi tipi di dato richiedono di nomi definiti con **ancora più attenzione!**
 - (es.) **typedef persona, typedef account, int calcolaBMI()**

Stile di Programmazione - Nomi

Come scegliere appropriatamente un nome?

Stile di Programmazione - Nomi

Come scegliere appropriatamente un nome?

- Variabili e funzioni ad ambito limitato: **nomi brevi**
 - **L'ambito** può fornire già informazioni sul loro ruolo
 - **i, j, k** per indici
 - **s, t** per stringhe; **c** per caratteri
 - **n, m** per interi, **x, y** per numeri frazionari
 - **p, q** per puntatori

Stile di Programmazione - Nomi

Come scegliere appropriatamente un nome?

- Variabili e funzioni ad ambito limitato: **nomi brevi**
 - L'ambito può fornire già informazioni sul loro ruolo
 - i, j, k per indici
 - s, t per stringhe; c per caratteri
 - n, m per interi, x, y per numeri reali
 - p, q per puntatori
- Variabili e funzioni ad ampio ambito: **nomi lunghi e descrittivi**
 - utilizzate **in sezioni disparate del programma**
 - utile accompagnarle da un commento
 - es. `int max_bmi_maggiorenni = 0;` // valore massimo del BMI per i maggiorenni

Stile di Programmazione – Nomi - Esempi

```
for (theElementIndex = 0 ;  
     theElementIndex < number0fElements;  
     theElementIndex++)  
    elementArray[theElementIndex] = theElementIndex;
```

No.

Stile di Programmazione – Nomi - Esempi

```
for (theElementIndex = 0 ;  
     theElementIndex < number0fElements;  
     theElementIndex++)  
    elementArray[theElementIndex] = theElementIndex;
```

No.

```
for (i = 0 ; i < n_elems; i++) {  
    elem[i] = i ;  
}
```

Ok!

Stile di Programmazione – Nomi - Esempi

```
for (theElementIndex = 0 ;  
     theElementIndex < number0fElements;  
     theElementIndex++)  
    elementArray[theElementIndex] = theElementIndex;
```

No.

```
for (i = 0 ; i < n_elems; i++) {  
    elem[i] = i ;  
}
```

Ok!

Lo stesso blocco di codice, scritto utilizzando una nomenclatura di variabili diversa, **ha una leggibilità estremamente diversa!** E' sufficiente scegliere una convenzione errata per gli indici **per rendere il programma di difficile comprensione**

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro
 - float max_bmi = 0;
 - float min_bmi = 0;

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro
 - float max_bmi = 0;
 - float min_bmi = 0;

Se due diverse variabili si riferiscono allo stesso «concetto», bisogna utilizzare **sempre lo stesso stile nella nomenclatura.**

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro
 - float max_bmi = 0;
 - float min_bmi = 0;

NO!

```
float max_bmi = 0  
float bmi_minimo = 0;
```

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro
 - float max_bmi = 0;
 - float min_bmi = 0;
- E' errato utilizzare una **nomenclatura diversa** per riferirsi allo **stesso concetto**
 - int num_studenti = 0;
 - int max_eta_stud = 0;

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro
 - float max_bmi = 0;
 - float min_bmi = 0;
- E' errato utilizzare una **nomenclatura diversa** per riferirsi allo **stesso concetto**
 - int num_studenti = 0;
 - int max_eta_stud = 0;

stud? studenti?

Bisogna fare una scelta e mantenerla per tutto il programma!

Stile di Programmazione - Nomi

Un altro elemento importante è la consistenza dei nomi

- **Variabili correlate** devono avere nomi «simili» tra loro
 - float max_bmi = 0;
 - float min_bmi = 0;
- E' errato utilizzare una **nomenclatura diversa** per riferirsi allo **stesso concetto**
 - int num_studenti = 0;
 - int max_eta_stud = 0;
 - **stud** e **studenti** sono due nomi diversi, che si riferiscono allo stesso concetto. **Perché utilizzare due nomi diversi?**
 - Garantiamo la consistenza → **int max_eta_studenti = 0** è un nome corretto.

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

- Le costanti simboliche si scrivono **in maiuscolo**
 - `#define STUDENTI 150`

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

- Le costanti simboliche si scrivono **in maiuscolo**
 - `#define STUDENTI 150`
- Nomi di variabili lunghi devono essere suddivisi **per renderli più leggibili**
 - `int maxbmimaggiorenni` → NO
 - `int max_bmi_maggiorenni` → SI
 - `int maxBMIMaggiorenni` → SI
- La scelta tra underscore e alternanza maiuscole/minuscole **è personale**

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

- I nomi di funzioni devono **avere uno stile imperativo**, indicando i nomi delle azioni che la funzione implementa
 - `printf()` → **stampa**
 - `calcolaBMI()` → **calcola il BMI**
 - `calculateAverage()` → **calcola la media**

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

- I nomi di funzioni devono **avere uno stile imperativo**, indicando i nomi delle azioni che la funzione implementa
 - `printf()` → **stampa**
 - `calcolaBMI()` → **calcola il BMI**
 - `calculateAverage()` → **calcola la media**
 - **funzionePerCalcoloMedia()** → NO!

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

- I nomi di funzioni **booleane** devono **avere la dicitura ‘is’** nel nome della funzione, per far capire immediatamente **che restituiscono una risposta del tipo «si/no»**
 - `isDigit(char)` , `isAlNum(char)` , `isAdult(age)`

Stile di Programmazione - Nomi

Esistono anche delle convenzioni che è OBLIGATORIO seguire

- I nomi di funzioni **booleane** devono **avere la dicitura ‘is’** nel nome della funzione, per far capire immediatamente **che restituiscono una risposta del tipo «si/no»**
 - `isDigit(char)` , `isAlNum(char)` , `isAdult(age)`

- Questo rende il codice estremamente più leggibile e comprensibile, perché le espressioni diventano più vicine a quelle che utilizzerebbe l'uomo

```
if ( isDigit(char) )
    puts('E' una cifra');
```

Stile di Programmazione - Nomi

E' importante che i nomi delle funzioni siano coerenti con ciò che realmente implementano

Stile di Programmazione - Nomi

E' importante che i nomi delle funzioni siano coerenti con ciò che realmente implementano

- Il nome della funzione **fa pensare ad uno scopo**
- Il «corpo» della funzione (cioè il **codice sorgente**) rappresenta **cioè che la funzione realmente fa.**
- Le due cose devono **corrispondere**
 - Facciamo un esempio.

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    //cosa ci aspettiamo dentro questa funzione?  
}
```

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    if (examVote>25) return 1;  
    else return 0;  
}
```

No.

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    if (examVote>25) return 1;  
    else return 0;  
}
```

No.

```
int vote = 20;  
if ( isPassed(vote) )  
    puts(''Passed'');  
else  
    puts(''Not Passed'')
```

Cosa stampa?

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    if (examVote>25) return 1;  
    else return 0;  
}
```

No.

```
int vote = 20;  
if ( isPassed(vote) )  
    puts(''Passed'');  
else  
    puts(''Not Passed'')
```

Cosa stampa?

Not Passed

E' quello che ci aspettiamo?

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    if (examVote>25) return 1;  
    else return 0;  
}
```

No.

```
int isPassed(int examVote) {  
    if (examVote>=18) return 1;  
    else return 0;  
}
```

Ok!

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    if (examVote>25) return 1;  
    else return 0;  
}
```

L'implementazione deve essere coerente con lo scopo che ci è indicato dal nome della funzione, altrimenti i programmi possono avere degli effetti collaterali.

No.

```
int isPassed(int examVote) {  
    if (examVote>=18) return 1;  
    else return 0;  
}
```

Utilizzereste mai una funzione (es. isDigit()) il cui comportamento non è coerente con il suo nome?

Ok!

Coerenza!

Stile di Programmazione – Nomi - Esempi

```
int isPassed(int examVote) {  
    if (examVote>25) return 1;  
    else return 0;  
}
```

L'implementazione deve essere coerente con lo scopo che ci è indicato dal nome della funzione, altrimenti i programmi possono avere degli effetti collaterali.

No.

```
int isPassed(int examVote) {  
    if (examVote>=18) return 1;  
    else return 0;  
}
```

Utilizzereste mai una funzione (es. isDigit()) il cui comportamento non è coerente con il suo nome?

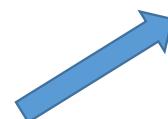
Ok!

Il nome della funzione non è solo un'etichetta: deve fornire delle informazioni!

Stile di Programmazione – Nomi - Esercizio

E' la migliore scelta possibile per i **nomi delle variabili?**

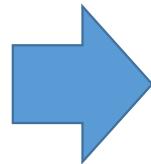
```
#define TRUE 0  
#define FALSE 1  
if ((ch = getchar()) == EOF)  
    not-eof = FALSE;
```

 **getchar()** → è una funzione che legge un carattere da file
ch → variabile che memorizza un carattere
EOF → indicatore di fine file

Stile di Programmazione – Nomi - Esercizio

E' la migliore scelta possibile per i **nomi delle variabili?**

```
#define TRUE 0  
#define FALSE 1  
if ((ch = getchar()) == EOF)  
    not-eof = FALSE;
```



```
#define TRUE 0  
#define FALSE 1  
if ((ch = getchar()) == EOF)  
    eof = TRUE;
```

L'espressione restituisce '**true**' se siamo arrivati alla fine del file. E' molto più **leggibile, coerente, interpretabile e chiaro** chiamare la variabile '**eof**' e impostarla a **TRUE**.

Stile di Programmazione - Espressioni

Considerazioni analoghe possono essere fatte in merito alla scelta delle espressioni

Stile di Programmazione - Espressioni

Considerazioni analoghe possono essere fatte in merito alla scelta delle espressioni

- Le espressioni devono essere scritte in modo che il loro significato sia il più possibile **trasparente**
- Più espressioni sono equivalenti: scegliere quella più chiara
 - Non sempre la più chiara è la più breve
 - **Usare spazi** tra operatori per suggerire raggruppamenti
 - In generale, formattare l'espressione **per aumentarne la leggibilità**

Stile di Programmazione - Espressioni

Considerazioni analoghe possono essere fatte in merito alla scelta delle espressioni

- Le espressioni devono essere scritte in modo che il loro significato sia il più possibile **trasparente**
- Più espressioni sono equivalenti: scegliere quella più chiara
 - Non sempre la più chiara è la più breve
 - **Usare spazi** tra operatori per suggerire raggruppamenti
 - In generale, formattare l'espressione **per aumentarne la leggibilità**

```
if(a>b&&c !=0&&d<1)
```

NO

Stile di Programmazione - Espressioni

Considerazioni analoghe possono essere fatte in merito alla scelta delle espressioni

- Le espressioni devono essere scritte in modo che il loro significato sia il più possibile **trasparente**
- Più espressioni sono equivalenti: scegliere quella più chiara
 - Non sempre la più chiara è la più breve
 - **Usare spazi** tra operatori per suggerire raggruppamenti
 - In generale, formattare l'espressione **per aumentarne la leggibilità**

```
if ( a>b && c!=0 && d<1 )
```

SI

Stile di Programmazione - Espressioni

Il primo concetto fondamentale è quello dell'indentazione

- L'indentazione mostra la struttura di un programma
- Ci sono delle convenzioni precise per indentare correttamente un programma

Stile di Programmazione - Espressioni

Il primo concetto fondamentale è quello dell'indentazione

- L'indentazione mostra la struttura di un programma
- Ci sono delle convenzioni precise per indentare correttamente un programma

```
for(n++;n<100;field[n++]='\0');  
*i='\0'; return ('\n');
```

No Indentazione e uso non corretto delle espressioni.

Stile di Programmazione - Espressioni

Il primo concetto fondamentale è quello dell'indentazione

- L'indentazione mostra la struttura di un programma
- Ci sono delle convenzioni precise per indentare correttamente un programma

```
for(n++;n<100;field[n++]= '\0');  
*i= '\0'; return( '\n');
```

```
for(n=n+1; n<100; n++){  
    field[n] = '\0';  
}  
  
*i = '\0';  
return '\n';
```

No Indentazione e uso non corretto delle espressioni.

Codice correttamente indentato.

Stile di Programmazione - Espressioni

Il primo concetto fondamentale è quello dell'indentazione

- L'indentazione mostra la struttura di un programma
- Ci sono delle convenzioni precise per indentare correttamente un programma

Attenzione all'uso corretto degli spazi!
L'uso di più spazi e l'inserimento di
qualche riga vuota rende il programma
più leggibile!

```
for(n=n+1; n<100; n++){
    field[n] = '\0';
}

*i = '\0';
return '\n';
```

No Indentazione e uso non corretto delle espressioni.

Codice correttamente indentato.

Stile di Programmazione - Espressioni

Le espressioni che inseriamo nei nostri programmi devono essere quanto più «naturali» possibili

Stile di Programmazione - Espressioni

Le espressioni che inseriamo nei nostri programmi devono essere quanto più «naturali» possibili

- Scrivere le espressioni **come se fossero pronunciate ad alta voce**
 - **Limitare l'uso delle forme negative**
 - Evitare espressioni che siano **troppo lunghe**
 - nel caso, spezzare le espressioni in sotto-espressioni oppure fattorizzare in funzioni

Stile di Programmazione - Espressioni

Le espressioni che inseriamo nei nostri programmi devono essere quanto più «naturali» possibili

- Scrivere le espressioni **come se fossero pronunciate ad alta voce**
 - **Limitare l'uso delle forme negative**
 - Evitare espressioni che siano **troppo lunghe**
 - nel caso, spezzare le espressioni in sotto-espressioni oppure fattorizzare in funzioni

```
if (!(a==0) || !(b==0))  
if (!((a==0) && (b==0)))  
if ((a!=0) || (b!=0))...
```

Stile di Programmazione - Espressioni

Le espressioni che inseriamo nei nostri programmi devono essere quanto più «naturali» possibili

- Scrivere le espressioni come se fossero pronunciate ad alta voce
 - **Limitare l'uso delle forme negative**
 - Evitare espressioni che siano **troppo lunghe**
 - nel caso, spezzare le espressioni in sotto-espressioni oppure fattorizzare in funzioni

NO →

```
if (!(a==0) || !(b==0))  
if (!((a==0) && (b==0)))  
if ((a!=0) || (b!=0))...
```

← **SI**

Stile di Programmazione - Espressioni

L'uso attento delle parentesi può aiutare a migliorare la leggibilità e la comprensibilità di un'espressione

Stile di Programmazione - Espressioni

L'uso attento delle parentesi può aiutare a migliorare la leggibilità e la comprensibilità di un'espressione

- Le parentesi **possono chiarire il significato di un'espressione** anche quando non sono necessarie
 - Come l'indentazione, **il raggruppamento riduce la complessità del codice sorgente**
 - Anche se le regole di precedenza degli operatori fanno capire in che ordine l'espressione sarà valutata, **le parentesi possono aiutare a migliorare la leggibilità**

Stile di Programmazione - Espressioni

L'uso attento delle parentesi può aiutare a migliorare la leggibilità e la comprensibilità di un'espressione

- Le parentesi **possono chiarire il significato di un'espressione** anche quando non sono necessarie
 - Come l'indentazione, **il raggruppamento aiuta a gestire la complessità**
 - Anche se le regole di precedenza degli operatori fanno capire in che ordine l'espressione sarà valutata, **le parentesi possono aiutare a migliorare la leggibilità**

```
a !=0&&b+1==0
```

vs.

```
(a !=0) && (b == -1)
```

Stile di Programmazione – Espressioni - Esempio

Scrivere un'espressione che calcoli se l'anno è bisestile

```
leap_year = y % 4 == 0 && y % 100 != 0 || y % 400 == 0;
```

vs.

```
leap_year = ((y % 4 == 0) && (y % 100 != 0)) || (y % 400 == 0);
```

L'uso attento delle parentesi può aiutare a migliorare la leggibilità e la comprensibilità di un'espressione

Stile di Programmazione - Espressioni

In particolari scenari, può essere utile spezzare le espressioni molto complesse

- A volte la compattezza estrema può rende il codice molto difficile da comprendere
- **Spezzare le espressioni aumenta la leggibilità**

Stile di Programmazione - Espressioni

In particolari scenari, può essere utile spezzare le espressioni molto complesse

- A volte la compattezza estrema può rende il codice molto difficile da comprendere
- **Spezzare le espressioni aumenta la leggibilità**

```
x += (xp=(2*k < (n-m) ? c[k+1] : d[k--]));
```

Poco Leggibile.

Stile di Programmazione - Espressioni

In particolari scenari, può essere utile spezzare le espressioni molto complesse

- A volte la compattezza estrema può rende il codice molto difficile da comprendere
- **Spezzare le espressioni aumenta la leggibilità**

```
if (2*k < n-m)
    xp = c[k+1];
else
    xp = d[k--];
x += *xp;
```

Più Leggibile.

La scrittura del codice deve essere orientata all'individuazione **delle forme più comprensibili**, non di quelle più compatte.

Stile di Programmazione - Espressioni

Le scelte in fase di codifica devono essere orientate a massimizzare la chiarezza del codice sorgente

- Lo scopo non è dimostrare la bravura o la creatività 😊
- **Lo scopo è scrivere del codice facilmente comprensibile a chi lavorerà su quel codice dopo di voi!**

Stile di Programmazione - Espressioni

Le scelte in fase di codifica devono essere orientate a massimizzare la chiarezza del codice sorgente

- Lo scopo non è dimostrare la bravura o la creatività 😊
- **Lo scopo è scrivere del codice facilmente comprensibile a chi lavorerà su quel codice dopo di voi!**

```
child =  
    (!LC&&!RC)?0:(!LC?RC:LC)
```

Stile di Programmazione - Espressioni

Le scelte in fase di codifica devono essere orientate a massimizzare la chiarezza del codice sorgente

- Lo scopo non è dimostrare la bravura o la creatività ☺
- **Lo scopo è scrivere del codice facilmente comprensibile a chi lavorerà su quel codice dopo di voi!**

```
child =  
    (!LC&&!RC)?0:(!LC?RC:LC)
```

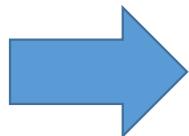


Stile di Programmazione - Espressioni

Le scelte in fase di codifica devono essere orientate a massimizzare la chiarezza del codice sorgente

- Lo scopo non è dimostrare la bravura o la creatività ☺
- **Lo scopo è scrivere del codice facilmente comprensibile a chi lavorerà su quel codice dopo di voi!**

```
child =  
    (!LC&&!RC)?0:(!LC?RC:LC)
```



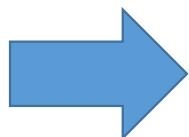
```
if ((LC==0) && (RC==0))  
    child = 0;  
else if (LC==0)  
    child = RC;  
else  
    child = LC;
```

Stile di Programmazione - Espressioni

Le scelte in fase di codifica devono essere orientate a massimizzare la chiarezza del codice sorgente

- Lo scopo non è dimostrare la bravura o la creatività ☺
- **Lo scopo è scrivere del codice facilmente comprensibile a chi lavorerà su quel codice dopo di voi!**

```
child =  
    (!LC&&!RC)?0:(!LC?RC:LC)
```



```
if ((LC==0) && (RC==0))  
    child = 0;  
else if (LC==0)  
    child = RC;  
else  
    child = LC;
```

L'operatore ternario di selezione è da utilizzare meno possibile, perché rende il codice poco leggibile. **Si può adottare solo per espressioni molto semplici.**

Stile di Programmazione - Espressioni

**Una scarsa chiarezza nel codice e scelte errate possono portare ad
‘effetti collaterali’ inaspettati**

- Non ci sono bug... ma i programmi non si comportano come ci aspettiamo!
- **Esempio**

Stile di Programmazione - Espressioni

Una scarsa chiarezza nel codice e scelte errate possono portare ad ‘effetti collaterali’ inaspettati

- Non ci sono bug... ma i programmi non si comportano come ci aspettiamo!
- **Esempio**

```
printf('Indica la posizione del vettore da modificare e il nuovo valore:');  
scanf("%d %d", &yr, &profit[yr]);
```

Stile di Programmazione - Espressioni

Una scarsa chiarezza nel codice e scelte errate possono portare ad ‘effetti collaterali’ inaspettati

- Non ci sono bug... ma i programmi non si comportano come ci aspettiamo!
- **Esempio**

```
printf('Indica la posizione del vettore da modificare e il nuovo valore:');  
scanf("%d %d", &yr, &profit[yr]);
```

Cosa fa? Legge due variabili di tipo intero.

Ci aspetteremmo però che il valore letto dalla variabile **yr** venga utilizzato per modificare la posizione del vettore

Stile di Programmazione - Espressioni

Una scarsa chiarezza nel codice e scelte errate possono portare ad ‘effetti collaterali’ inaspettati

- Non ci sono bug... ma i programmi non si comportano come ci aspettiamo!
- **Esempio**

```
printf('Indica la posizione del vettore da modificare e il nuovo valore:');  
scanf("%d %d", &yr, &profit[yr]);
```

Questo non accade, perché il linguaggio C valuta tutta l’intera istruzione `scanf` nello stesso momento. Dunque il valore `yr` nell’indice del vettore `profit` non viene modificato

Stile di Programmazione - Espressioni

Una scarsa chiarezza nel codice e scelte errate possono portare ad ‘effetti collaterali’ inaspettati

- Non ci sono bug... ma i programmi non si comportano come ci aspettiamo!
- **Esempio**

```
printf('Indica la posizione del vettore da modificare e il nuovo valore:');  
scanf("%d %d", &yr, &profit[yr]);
```

Codice troppo compatto e una scarsa conoscenza dei meccanismi interni del Linguaggio C può portare ad effetti collaterali inaspettati!

Stile di Programmazione - Espressioni

**Una scarsa chiarezza nel codice e scelte errate possono portare ad
‘effetti collaterali’ inaspettati**

- Non ci sono bug... ma i programmi non si comportano come ci aspettiamo!
- **Esempio**

```
scanf("%d", &yr);
scanf("%d", &profit[yr]);
```

**Separando l’istruzione in due `scanf()` separate otteniamo una versione
meno compatta, ma più intuitiva e soprattutto funzionante!**

Stile di Programmazione - Esercizio

Migliorare la leggibilità dei seguenti frammenti di codice

1) `if (!(c=='y' || c=='Y')) return;`

2) `length = length<BUFSIZE?length:BUFSIZE;`

Stile di Programmazione - Esercizio

Migliorare la leggibilità dei seguenti frammenti di codice

1) `if (!(c=='y' || c=='Y')) c=FALSE;`

→ `if (c != 'y' && c != 'Y')
 c=FALSE;`

2) `length = length<BUFSIZE?length:BUFSIZE;`

→ `if (length > BUFSIZE)
 length = BUFSIZE;`

Stile di Programmazione - Esercizio

Migliorare la leggibilità dei seguenti snippet

1) `if (!(c=='y' || c=='Y')) c=FALSE;`

→ `if (c != 'y' && c != 'Y')
 c=FALSE;`

2) `length = length<BUFSIZE?length:BUFSIZE;`

→ `if (length > BUFSIZE)
 length = BUFSIZE;`

Suggerimento.

Quando si utilizzano operatori logici (`&&`, `||`) è utile aggiungere degli spazi aggiuntivi per migliorare la leggibilità.

Spesso è utile anche con gli operatori relazionali (`>`, `<`, `>=`, etc.)

Stile di Programmazione - Consistenza

Porzioni diverse del programma devono essere organizzate in modo prevedibile e immediatamente comprensibile

- **Consistenza nel codice** = usare la stessa **forma** per snippet di codice che hanno un significato simile
 - Indentazione
 - uso delle parentesi
 - struttura dei cicli
 - struttura delle decisioni

Stile di Programmazione - Consistenza

Porzioni diverse del programma devono essere organizzate in modo prevedibile e immediatamente comprensibile

- **Consistenza nel codice** = usare la stessa **forma** per snippet di codice che hanno un significato simile
 - Indentazione
 - uso delle parentesi
 - struttura dei cicli
 - struttura delle decisioni
 - **Esistono delle Convenzioni di stile**
 - es. Linux Kernel Coding Style
- <https://www.kernel.org/doc/Documentation/process/coding-style.rst>

Stile di Programmazione - Consistenza

**Le parentesi graffe devono essere usate sempre nello stesso modo
Stesso rigo dell'istruzione? Rigo successivo?
Fare una scelta e portarla avanti in tutto il programma!**

Stile di Programmazione - Consistenza

**Le parentesi graffe devono essere usate sempre nello stesso modo
Stesso rigo dell'istruzione? Rigo successivo?
Fare una scelta e portarla avanti in tutto il programma!**

Importante: quando modificate del codice altrui, sforzatevi a mantenere lo stile consistente! Utilizzate gli stessi accorgimenti del programmatore precedente, anche se sono diversi dai vostri!

Stile di Programmazione - Consistenza

**Le parentesi graffe devono essere usate sempre nello stesso modo
Stesso rigo dell'istruzione? Rigo successivo?
Fare una scelta e portarla avanti in tutto il programma!**

Importante: quando modificate del codice altrui, sforzatevi a mantenere lo stile consistente! Utilizzate gli stessi accorgimenti del programmatore precedente, anche se sono diversi dai vostri!

A volte le graffe non sono necessarie, **ma bisogna assicurarsi che la rimozione non crei bug (problema del dangling else)**
Esempio!

Stile di Programmazione - Consistenza

```
1 if (month==FEB) {  
2     if (year%4 == 0)  
3         if (day > 29)  
4             legal = FALSE;  
5     else  
6         if (day > 28)  
7             legal = FALSE;  
8 }
```

Stile di Programmazione - Consistenza

```
1 if (month==FEB) {  
2     if (year%4 == 0)  
3         if (day > 29)  
4             legal = FALSE;  
5     else  
6         if (day > 28)  
7             legal = FALSE;  
8 }
```

Come si comporta il codice con questi input ?

Day = 29
Month = FEB
Year = 2000

Stile di Programmazione - Consistenza

```
1 if (month==FEB) {  
2     if (year%4 == 0)  
3         if (day > 29)  
4             legal = FALSE;  
5     else  
6         if (day > 28)  
7             legal = FALSE;  
8 }
```

Restituisce **FALSE**, anche se la data è corretta!

Day = 29
Month = FEB
Year = 2000

Stile di Programmazione - Consistenza

```
1 if (month==FEB) {  
2     if (year%4 == 0)  
3         if (day > 29)  
4             legal = FALSE;  
5     else  
6         if (day > 28)  
7             legal = FALSE;  
8 }
```

Perché? Cosa abbiamo sbagliato?

Stile di Programmazione - Consistenza

```
1 if (month==FEB) {  
2     if (year%4 == 0)  
3         if (day > 29)  
4             legal = FALSE;  
5     else  
6         if (day > 28)  
7             legal = FALSE;  
8 }
```

Senza le parentesi graffe, l'espressione **if (day > 28)**

Viene valutata insieme all'else del rigo precedente, generando un comportamento inaspettato!

Stile di Programmazione - Consistenza

```
1 if (month==FEB) {  
2     if (year%4 == 0)  
3         if (day > 29)  
4             legal = FALSE;  
5     else  
6         if (day > 28)  
7             legal = FALSE;  
8 }
```

```
if (month==FEB) {  
    if (year%4 == 0) {  
        if (day > 29)  
            legal = FALSE;  
    } else {  
        if (day > 28)  
            legal = FALSE;  
    } }
```

**Inserire parentesi
graffe in più può
aiutare ad evitare bug
inattesi!**

Stile di Programmazione - Consistenza

In un linguaggio, le espressioni idiomatiche sono ‘delle espressioni tipiche dell’idioma’ (cioè della lingua!)

Stile di Programmazione - Consistenza

In un linguaggio, le espressioni idiomatiche sono ‘delle espressioni tipiche dell’idioma’ (cioè della lingua!)

Rappresentano dei concetti che si esprimono solamente in quel modo
es. piove sul bagnato, lavarsene le mani, non vedere l’ora, etc.

Stile di Programmazione - Consistenza

In un linguaggio, le espressioni idiomatiche sono ‘delle espressioni tipiche dell’idioma’ (cioè della lingua!)

Rappresentano dei concetti che si esprimono solamente in quel modo
es. piove sul bagnato, lavarsene le mani, non vedere l’ora, etc.

**Anche i linguaggi di programmazioni hanno espressioni idiomatiche:
concetti che si esprimono solo in un modo!**

Questo si verifica tipicamente nei cicli

Probabilmente utilizzate le espressioni idiomatiche del C senza neppure saperlo ☺

Stile di Programmazione - Consistenza

Come possiamo esprimere un ciclo?

```
i=0;  
while (i <= n-1)  
    array[i++] = 1.0;
```

Stile di Programmazione - Consistenza

Come possiamo esprimere un ciclo?

```
i=0;  
while (i <= n-1)  
    array[i++] = 1.0;
```

```
for (i=0; i<n; )  
    array[i++] = 1.0;
```

Stile di Programmazione - Consistenza

Come possiamo esprimere un ciclo?

```
i=0;  
while (i <= n-1)  
    array[i++] = 1.0;
```

```
for (i=0; i<n; )  
    array[i++] = 1.0;
```

```
for (i=n; --i >= 0; )  
    array[i] = 1.0;
```

Stile di Programmazione - Consistenza

Come possiamo esprimere un ciclo?

```
i=0;  
while (i <= n-1)  
    array[i++] = 1.0;
```

```
for (i=0; i<n; )  
    array[i++] = 1.0;
```

```
for (i=n; --i >= 0; )  
    array[i] = 1.0;
```

Tutti quanti però **adottiamo l'espressione idiomatica**

```
for (i=0; i<n; i++)  
    array[i] = 1.0;
```

Stile di Programmazione - Consistenza

Come possiamo esprimere un ciclo?

```
i=0;  
while (i <= n-1)  
    array[i++] = 1.0;
```

```
for (i=0; i<n; )  
    array[i++] = 1.0;
```

```
for (i=n; --i >= 0; )  
    array[i] = 1.0;
```

Tutti quanti però **adottiamo l'espressione idiomatica**

```
for (i=0; i<n; i++)  
    array[i] = 1.0;
```

Utilizzare le espressioni idiomatiche rende il codice più comprensibile, anche «visivamente»

Stile di Programmazione - Consistenza

Anche le decisioni multiple hanno la loro espressione idiomatica

```
if (condition_1) {  
    ...  
} else if (condition_2) {  
    ...  
} else if...  
    ...  
} else {  
    // caso di default  
}
```

Stile di Programmazione - Consistenza

Anche le decisioni multiple hanno la loro espressione idiomatica

```
if (condition_1) {  
    ...  
} else if (condition_2) {  
    ...  
} else if...  
    ...  
} else {  
    // caso di default  
}
```



Le varie alternative vengono espresse in sequenza, una sotto l'altra.

Il codice si sviluppa in verticale ed è molto più leggibile.

Senza questo accorgimento,
avremmo del codice «diagonale»

Stile di Programmazione - Consistenza

```
if (argc==3)
    if ((fin = fopen(argv[1], "r")) != NULL)
        if ((fout = fopen(argv[2], "w")) != NULL)
            while ((c = getc(fin)) != EOF) {
                putc(c, fout);
                fclose(fin );
                fclose(fout);
            }
        else
            printf("Can't open output file %s\n", argv[2]) ;
    else
        printf("Can't open input file %s\n", argv[1]);
else
    printf ("Usage: cp inputfile outfile\n");
```

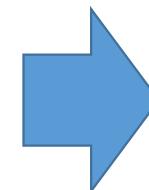


**Senza l'espressione
idiomatica il codice viene
espresso in forma 'diagonale'
, con tutti gli if all'inizio e
tutti gli else alla fine, e
diventa difficile capire quale
comportamento causa
ciascun percorso di decisione**

Stile di Programmazione - Consistenza

```
if (argc != 3) {  
    printf ("Usage: cp inputfile outputfile\n");  
} else if ((fin=fopen(argv[1], "r")) == NULL) {  
    printf("Can't open input file %s\n", argv[1]);  
} else if ((fout = fopen(argv[2], "w")) == NULL) {  
    printf("Can't open output file %s\n", argv[2]);  
    fclose(fin);  
}  
else {  
    while ((c = getc(fin)) != EOF)  
        putc(c, fout);  
    fclose(fin);  
    fclose(fout);  
}
```

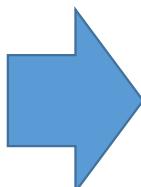
La struttura basata su
decisioni multiple migliora
leggibilità e comprensione!



Stile di Programmazione - Consistenza

Anche la scrittura dell'istruzione «case» prevede delle espressioni idiomatiche

```
switch (c) {  
    case '-':  
        sign = -1;  
        /*fall through */  
    case '+':  
        c = getchar();  
        break;  
    case '.':  
        break;  
    default:  
        if (!isdigit(c))  
            return 0;  
        break;
```



Ogni case deve avere il suo 'break'

Anche il default deve avere il suo 'break'

Quando il 'break' non è presente, bisogna esplicitamente indicarlo nei commenti!

Stile di Programmazione – Numeri Magici

Tutti i numeri presenti nel programma (con esclusione degli 0 e 1) dovrebbe essere sostituiti da costanti simboliche

- **Un numero (senza fornire ulteriori dettagli)** non fornisce informazioni a chi legge il programma
 - **Da dove deriva? Perché è importante?**

Stile di Programmazione – Numeri Magici

Tutti i numeri presenti nel programma (con esclusione degli 0 e 1) dovrebbe essere sostituiti da costanti simboliche

- **Un numero (senza fornire ulteriori dettagli)** non fornisce informazioni a chi legge il programma
 - **Da dove deriva? Perché è importante?**
- **Meglio utilizzare una costante simbolica**
 - **Fornisce informazioni aggiuntive** attraverso il nome
 - La modifica del valore di una costante si propaga in tutto il programma → **più facile la manutenzione**

Stile di Programmazione – Numeri Magici

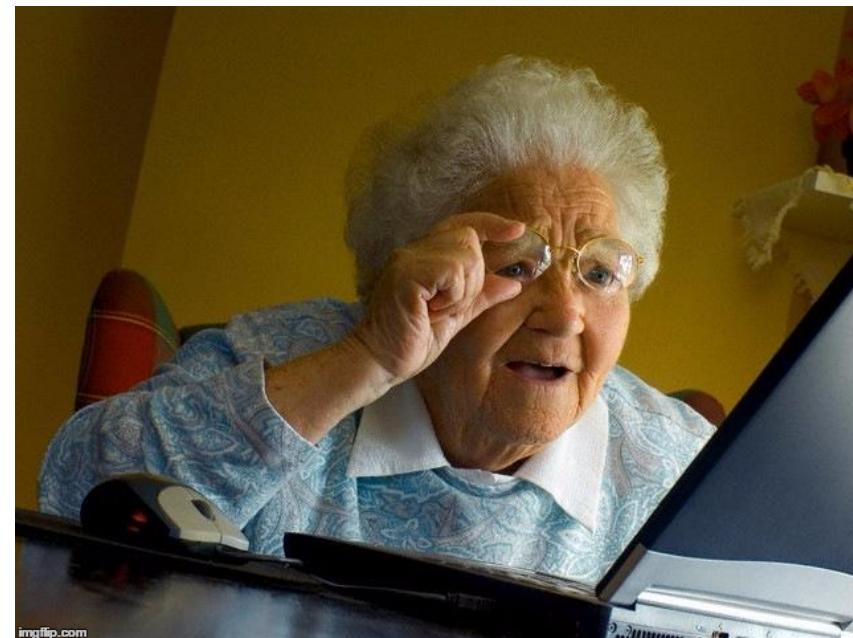
Un uso esagerato di valori numerici rende il codice molto complesso da comprendere

```
18 int main()
19 {
20     int total,fine,speeding;
21     puts("Speeding Ticketsn");
22 /* first ticket */
23     speeding = 85 - 55;
24     fine = speeding * 15;
25     total = total + fine;
26     printf("For going %d in a %d zone: $%dn", 85,55,fine);
27 /* second ticket */
28     speeding = 95 - 55;
29     fine = speeding * 15;
30     total = total + fine;
31     printf("For going %d in a %d zone: $%dn", 95,55,fine);
32 /* third ticket */
33     speeding = 100 - 55;
34     fine = speeding * 15;
35     total = total + fine;
36     printf("For going %d in a %d zone: $%dn", 100,55,fine);
37 /* Display total */
38     printf("nTotal in fines: $%dn",total);
39     return(0);
40 }
```

Stile di Programmazione – Numeri Magici

Un uso esagerato di valori numerici rende il codice molto complesso da comprendere!

```
18 int main()
19 {
20     int total,fine,speeding;
21     puts("Speeding Ticketsn");
22     /* first ticket */
23     speeding = 85 - 55;
24     fine = speeding * 15;
25     total = total + fine;
26     printf("For going %d in a %d zone: $%dn", 85,55,fine);
27     /* second ticket */
28     speeding = 95 - 55;
29     fine = speeding * 15;
30     total = total + fine;
31     printf("For going %d in a %d zone: $%dn", 95,55,fine);
32     /* third ticket */
33     speeding = 100 - 55;
34     fine = speeding * 15;
35     total = total + fine;
36     printf("For going %d in a %d zone: $%dn", 100,55,fine);
37     /* Display total */
38     printf("nTotal in fines: $%dn",total);
39     return(0);
40 }
```



Stile di Programmazione – Numeri Magici

Si procede definendo le costanti simboliche nel programma e sostituendole al codice

```
12 #include <stdio.h>
13 #define SPEEDLIMIT 55
14 #define RATE 15
15 #define FIRST_TICKET 85
16 #define SECOND_TICKET 95
17 #define THIRD_TICKET 100
```

```
18 int main()
19 {
20     int total,fine,speeding;
21     puts("Speeding Ticketsn");
22     /* first ticket */
23     speeding = FIRST_TICKET - SPEEDLIMIT;
24     fine = speeding * RATE;
25     total = total + fine;
26     printf("For going %d in a %d zone: $%dn", FIRST_TICKET,SPEEDLIMIT,fine);
27     /* second ticket */
28     speeding = SECOND_TICKET - SPEEDLIMIT;
29     fine = speeding * RATE;
30     total = total + fine;
31     printf("For going %d in a %d zone: $%dn", SECOND_TICKET,SPEEDLIMIT,fine);
32     /* third ticket */
33     speeding = THIRD_TICKET - SPEEDLIMIT;
34     fine = speeding * RATE;
35     total = total + fine;
36     printf("For going %d in a %d zone: $%dn", THIRD_TICKET,SPEEDLIMIT,fine);
37     /* Display total */
38     printf("nTotal in fines: $%dn",total);
39     return(0);
40 }
```

Stile di Programmazione – Numeri Magici

Si procede definendo le costanti simboliche nel programma e sostituendole al codice

```
18 int main()
19 {
20     int total,fine,speeding;
21     puts("Speeding Ticketsn");
22 /* first ticket */
23     speeding = 85 - 55;
24     fine = speeding * 15;
25     total = total + fine;
26     printf("For going %d in a %d zone: $%dn", 85,55,fine);
27 /* second ticket */
28     speeding = 95 - 55;
29     fine = speeding * 15;
30     total = total + fine;
31     printf("For going %d in a %d zone: $%dn", 95,55,fine);
32 /* third ticket */
33     speeding = 100 - 55;
34     fine = speeding * 15;
35     total = total + fine;
36     printf("For going %d in a %d zone: $%dn", 100,55,fine);
37 /* Display total */
38     printf("nTotal in fines: $%dn",total);
39     return(0);
40 }
```

Prima!

```
18 int main()
19 {
20     int total,fine,speeding;
21     puts("Speeding Ticketsn");
22 /* first ticket */
23     speeding = FIRST_TICKET - SPEEDLIMIT;
24     fine = speeding * RATE;
25     total = total + fine;
26     printf("For going %d in a %d zone: $%dn", FIRST_TICKET,SPEEDLIMIT,fine);
27 /* second ticket */
28     speeding = SECOND_TICKET - SPEEDLIMIT;
29     fine = speeding * RATE;
30     total = total + fine;
31     printf("For going %d in a %d zone: $%dn", SECOND_TICKET,SPEEDLIMIT,fine);
32 /* third ticket */
33     speeding = THIRD_TICKET - SPEEDLIMIT;
34     fine = speeding * RATE;
35     total = total + fine;
36     printf("For going %d in a %d zone: $%dn", THIRD_TICKET,SPEEDLIMIT,fine);
37 /* Display total */
38     printf("nTotal in fines: $%dn",total);
39     return(0);
40 }
```

Dopo!

Stile di Programmazione – Commenti

- I commenti sono intesi per aiutare il lettore di un programma
 - Non aiutano se dicono le stesse cose del codice
 - Non aiutano se sono in contraddizione con il codice
 - Non aiutano se distraggono il lettore

Stile di Programmazione – Commenti

Esempi di commenti inutili

```
/*
 * default
 */
default:
    break;

/* return SUCCESS */
return SUCCESS;

zerocount++; /* Increment zero entry counter */

/* Initialize "total" to "number_received" */
node->total = node->number_received;
```

I commenti sono intesi per aiutare il lettore di un programma
Non aiutano se dicono le stesse cose del codice

Stile di Programmazione – Commenti

I commenti sono inutili **anche quando le variabili sono già sufficientemente esplicative**

Stile di Programmazione – Commenti

I commenti sono inutili **anche quando le variabili sono già sufficientemente esplicative**

```
while ((c = getchar())!=EOF && isspace(c))
        /* skip whitespace */
if (c == EOF)          /* end of file */
    type = endoffile;
else if (c == '(')    /* left paren */
    type = leftparen;
else if (c == ')')    /* right paren */
    type = rightparen;
else if (c == ';')    /* semicolon*/
    type = semicolon;
else if (is_op(c))   /* operator */
    type = operator;
```

Stile di Programmazione – Commenti

- Quando i commenti **sono realmente utili?**
- **I commenti alle funzioni sono indispensabili**
 - Indicano **cosa fa la funzione**
 - Indicano **il significato dei parametri**
 - **Aggiunge informazioni non desumibili** dal nome della funzione
 - Chiariscono **passaggi fondamentali** nell'implementazione

Stile di Programmazione – Commenti

- Quando i commenti **sono realmente utili**?
- **I commenti alle funzioni sono indispensabili**
 - Indicano **cosa fa la funzione**
 - Indicano **il significato dei parametri**
 - **Aggiunge informazioni non desumibili** dal nome della funzione
 - Chiariscono **passaggi fondamentali** nell'implementazione
- I commenti sono utili a chiarire frammenti di codice confusi o non usuali
- ***Nota: se il commento stesso diventa troppo lungo e intricato, meglio riscrivere il codice!***

Stile di Programmazione – Commenti

- Attenzione a scrivere commenti che contraddicano il codice!
 - I commenti devono aiutare, non confondere!

```
int strcmp(char *s1, char *s2)
/* string comparison routine returns -1 if s1 is */
/* above s2 in an ascending order list, 0 if equal */
/* 1 if s1 below s2 */
{
    while(*s1==*s2){
        ...
}

int strcmp(char *s1, char *s2)
/* strcmp: return <0 if s1<s2, >0 if s1>s2, 0 if equal */
/* ANSI C, section 4.11.4.2 */
{
    while(*s1==*s2){
        if(*s1=='\0') return(0);
        ...
}
```

Stile di Programmazione – Commenti

- Attenzione a scrivere commenti che contraddicano il codice!
 - I commenti devono aiutare, non confondere!

Errato!

```
int strcmp(char *s1, char *s2)
/* string comparison routine returns -1 if s1 is */
/* above s2 in an ascending order list, 0 if equal */
/* 1 if s1 below s2 */
{
    while(*s1==*s2){
        ...
}
```

Corretto!

```
int strcmp(char *s1, char *s2)
/* strcmp: return <0 if s1<s2, >0 if s1>s2, 0 if equal */
/* ANSI C, section 4.11.4.2 */
{
    while(*s1==*s2){
        if(*s1=='\0') return(0);
        ...
}
```

Stile di Programmazione – Commenti

- Quando i commenti **sono realmente utili**?
- I commenti sono utili a **spiegare il ruolo delle variabili**
- I commenti sono utili a **illustrare i passaggi fondamentali del codice**
- I commenti sono utili a spiegare **blocchi di istruzioni complicati**
- I commenti sono utili a **spiegare scelte di progetto**
- I commenti non sono utili **se spiegano la sintassi del linguaggio**
- **I commenti non sono utili se non forniscono informazione aggiuntiva!**

Stile di Programmazione – Commenti

- I commenti sono utili a **spiegare il ruolo delle variabili**
 - float media = 0.0 // conterrà la media del campione
- I commenti sono utili a **illustrare i passaggi fondamentali del codice**
 - // il programma legge l'età e distingue il caso in cui l'individuo sia over o under 40
if(eta > SOGLIA_UNDER)

Stile di Programmazione – Commenti

- I commenti sono utili a spiegare **blocchi di istruzioni complicati**
 - // controllo che il numero di under non sia zero, per evitare valori errati
if(count_under > 0) [...]
- I commenti sono utili a spiegare **scelte di progetto**
 - Unsigned char peso = 0 // unsigned perché il valore non può essere negativo. Char perché il peso è sicuramente minore di 256

Stile di Programmazione – Commenti

- I commenti non sono utili **se spiegano la sintassi del linguaggio**
 - num_under++ // **incremento under**
- I commenti non sono utili se non forniscono **informazione aggiuntiva!**
 - if (eta > SOGLIA_UNDER) // **caso over**

K&R Coding Style

- Nel libro Kernighan&Ritchie sono indicate delle convenzioni di stile suggerite dai creatori del linguaggio.
- https://it.wikipedia.org/wiki/Stile_d%27indentazione#Stile_K.26R

K&R Coding Style

- **Ampia tabulazione**
 - 4 o 8 spazi
 - Scoraggia l'annidamento
- **Si indentano**
 - statement nel corpo di una funzione
 - statement nei blocchi
 - statement nei case
- **NON si indentano**
 - statement in una switch

K&R Coding Style

- **Ampia tabulazione**
 - 4 o 8 spazi
 - Scoraggia l'annidamento
- **Si indentano**
 - statement nel corpo di una funzione
 - statement nei blocchi
 - statement nei case
- **NON si indentano**
 - statement in una switch

```
int compare(int x, int y)
{
    if (x < y) {
        return -1;
    } else if (x > y) {
        return 1;
    } else {
        return 0;
}
```

K&R Coding Style

- **Ampia tabulazione**
 - 4 o 8 spazi
 - Scoraggia l'annidamento
- **Si indentano**
 - statement nel corpo di una funzione
 - statement nei blocchi
 - statement nei case
- **NON si indentano**
 - statement in una switch

```
int foo(int bar)
{
    switch (bar) {
        case 0:
            ++bar;
            break;
        case 1:
            --bar;
        default: {
            bar += bar;
            break;
        }
    }
}
```

K&R Coding Style (2)

- **Uso delle parentesi graffe**
 - A capo nella definizione di una funzione
 - Sulla stessa linea negli altri casi

K&R Coding Style (2)

- **Uso delle parentesi graffe**
 - A capo nella definizione di una funzione
 - Sulla stessa linea negli altri casi

```
int digits[] = { 0, 1, 2, 3, 4,
                 5, 6, 7, 8, 9 };

int compare(int x, int y)
{
    if (x < y) {
        return -1;
    } else if (x > y) {
        return 1;
    } else {
        return 0;
    }
}
```

K&R Coding Style – Flusso di Esecuzione

```
void bar()
{
    do {
        action()
    } while (condition);
}
```

```
void foo2()
{
    if (condition) {
        action();
    }

    if (condition1) {
        action1();
    } else if (condition2) {
        action2();
    } else {
        action3();
    }
}
```

K&R Coding Style – Istruzioni Multi-Linea

```
void f(int arg1, int arg2,
        int arg3, int arg4,
        int arg5, int arg6)
{
    bar(100, 200, 300, 400,
        600, 700, 800, 900);
}
```

Quando le funzioni hanno troppi parametri, bisogna cercare di distribuirli equamente tra le due righe, per avere più leggibilità

```
enum Example {
    CANCELLED,
    RUNNING,
    WAITING,
    FINISHED
};
```

Gli enum si sviluppano in verticale

K&R Coding Style – Nomenclatura e Commenti

`MY_CONSTANT`

`my_variable` o `myvar`

`my_function` o `myfun`

I nomi devono essere preferibilmente in inglese. In ogni caso, non mescolare italiano e inglese,

```
/* i commenti su linee dedicate  
documentano  
* blocchi di codice  
*/
```

```
int x = 0; // brevi commenti in linea  
int y = 0; /* commenti in linea  
piu' lunghi */
```

```
// puts("codice temporaneamente  
eliminato");
```



Esercizio 4.1

- **Modifiche l'implementazione dell'Esercitazione 0 e dell'Esercitazione 1, migliorando lo stile di programmazione e seguendo le convenzioni K&R.**