Introduzione alla programmazione in C(++)

Testi Consigliati:

- A. Kelley & I. Pohl "*C didattica e programmazione*"
- B.W. Kernighan & D. M. Ritchie "*Linguaggio C*"

Materiale disponibile sul sito

http://www.fisica.unile.it/~martello/corsi/Dottorato/index.html

Concetti di Base

Il C come altri linguaggi e' un insieme di regole sintattiche e lessicali che mettono insieme parole e punteggiatura allo scopo di creare un programma corretto.

L'oggetto che verifica se queste regole sintattiche sono rispettate e' il compilatore. Il compilatore analizza il programma scritto verifica le regole sintattiche e, se trova errori, produce messaggi inerenti gli errori trovati e termina, altrimenti produce un programma eseguibile.

Andando piu' in dettaglio nel processo di compilazione esso e' costituito da 3 fasi:

Concetti di Base

1. La pre-compilazione. In questa fase viene invocato un pre-compilatore che verifica ed esegue solo alcuni tipi di comandi presenti nel listato processato e produce un nuovo listato.
2. La compilazione vera e propria. In questa fase vengono verificate le regole sintattiche e il listato viene convertito in un modulo scritto in linguaggio macchina.
3. Il link. In questa fase viene invocato un programma in grado di connettere questo modulo con altri moduli forniti dall'utente o facenti parte del sistema per produrre un programma eseguibile.

Concetti di Base: gli elementi

Un listato che deve essere compilato per produrre un programma e' un insieme di caratteri e simboli. Per cui come prima cosa vediamo quali sono i caratteri riconosciuti dal compilatore.

- lettere minuscole `a,b,..,z`
- lettere maiuscole `A,B,…,Z`
- numeri `0,1,…,9`
- spazi bianchi
- altri caratteri `+-*/=(){}[]<>,.?#%&!:;'"|~_^\`

Concetti di Base: Il Compilatore

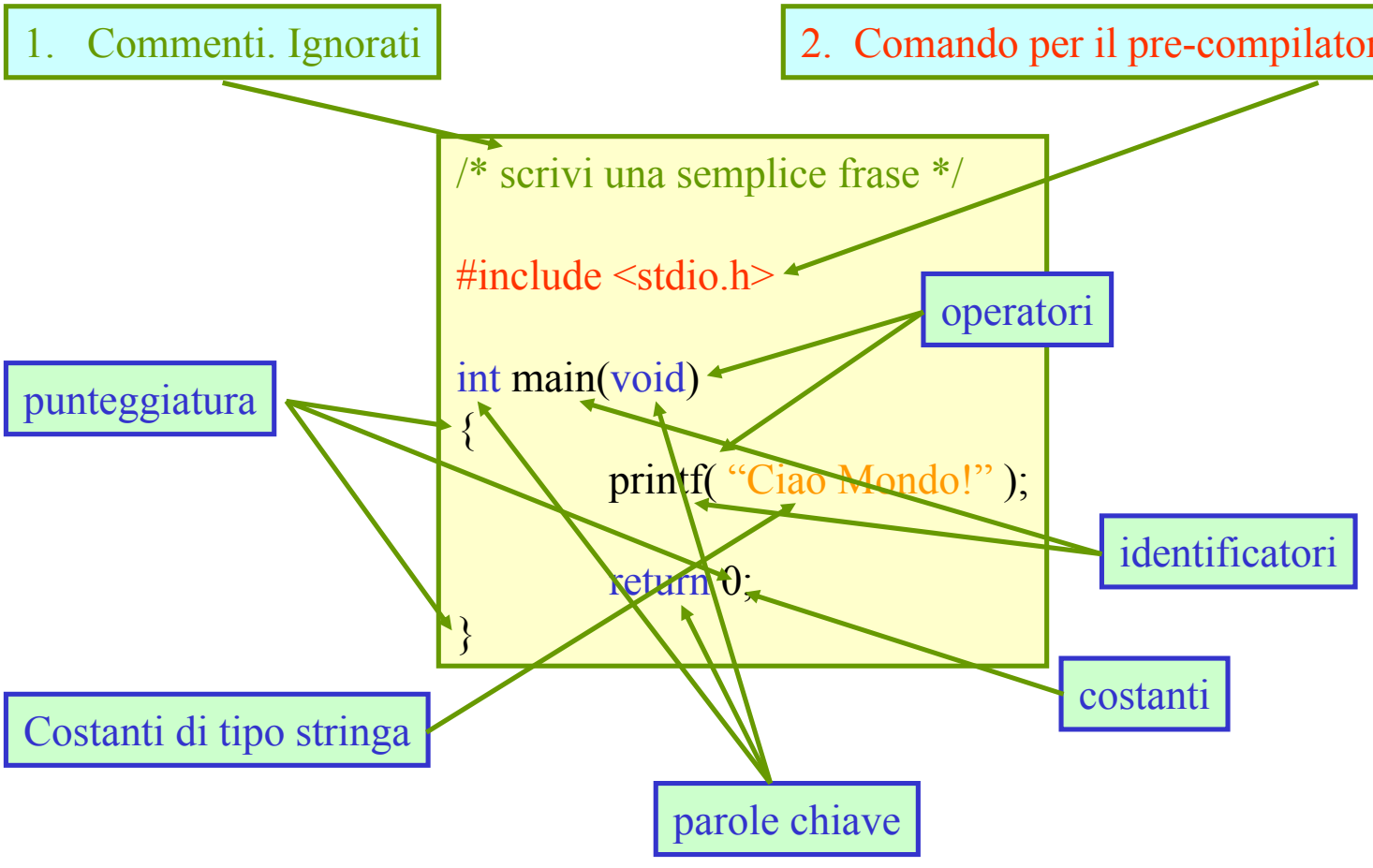
Il compilatore come prima cosa collega i caratteri di cui e' costituito il listato per formare elementi. In C esistono 7 tipi diversi di elementi di cui puo' essere costituito un listato. Essi sono:

- le parole chiave
- gli identificatori
- le costanti
- le costanti di tipo stringa
- gli operatori
- la punteggiatura
- i commenti

Come passo successivo il compilatore verifica se gli elementi che ha identificato sono corretti e soddisfano le regole sintattiche del linguaggio.

E' importante sottolineare subito che nella verifica di queste regole sintattiche il compilatore e' estremamente rigido. A differenza di un umano, il compilatore NON e' in grado di comprendere elementi o successione di elementi che non siano scritti in maniera completamente esatta!

Concetti di Base: gli elementi



Concetti di Base: gli elementi

Primo esempio di programma

Log-in in ambienteLINUX
Apri una finestra terminale
mkdir CorsoC
ls
cd CorsoC
mkdir PrimaLezione
cd PrimaLezione
nedit esempio1.c &
scrivi il codice in esempio1.c

```
/* scrivi una semplice frase */
#include <stdio.h>

int main(void)
{
        printf( "Ciao Mondo!" );

        return 0;
}
```

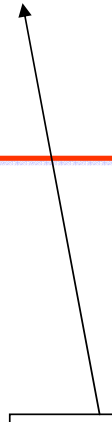
Attenzione printf e' il metodo di stampa su terminale o file usato in linguaggio C. In questo corso abbiamo deciso di utilizzare i metodi Input/Output di C++

```
/*in stile C++*/
#include <iostream>

int main(void)
{
  std::cout << "Ciao Mondo!" <<std::endl;

        return 0;
}
```

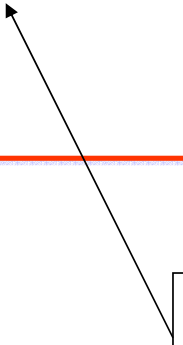
Concetti di Base: gli elementi

Come Compilare ed eseguire il programma

```
g++ esempio1.c –o esempio1
ls –alF
./esempio1
```



file
contente il listato
file da compilare



file contente il
risultato della compilazione
file da eseguire

Concetti di Base: le parole chiave

Le parole chiave sono delle parole che hanno un significato particolare nel codice. Ogniuna di essere serve a definire una particolare azione. Le parole chiave sono in numero limitato e NON possono essere ridefinite dal programmatore.

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

Concetti di Base: gli identificatori

Un identificatore e' costituito da una successione **contigua** di lettere (minuscole e/o maiuscole) numeri e dal carattere _. In C un identificatore puo' iniziare con _ o con una lettera, ma **non** con un numero!

k
la_mia_incognita
_IlMioIndice
parametro2
Id_12345

123_valori
Primo#Secondo
-neg
NO!

Un identificatore serve ad identificare qualcosa all'interno del programma (es. un identificatore puo' essere il nome di una variabile). E' conveniente usare identificatori che abbiano un significato. In modo da ricordare a cosa si riferiscono.

NON e' possibile definire identificatori che siano uguali a una parola chiave!

Concetti di Base: le costanti

In ogni programma si incontrano numerosi tipi di costanti. Una costante e' costituita da un numero o da un carattere.

2 23 172 sono esempi di costanti intere

1.23 13.452 sono esempi di costanti reali

'a' 'b' 'F' sono esempi di costanti carattere.
Notare che per distinguerle dagli identificatori quest'ultime devono essere delimitate da due '.

Esistono poi delle costanti speciali come '\n'. Il \ sta ad indicare che la n che lo segue ha un significato particolare e non semplicemente quello di lettera dell'alfabeto (in questo caso essa rappresenta il carattere di *newline*).

Concetti di Base: le costanti stringa

Una qualunque successione di caratteri racchiusa tra doppio apice (es.: “ciao”) viene sempre identificata dal compilatore come un singolo elemento: una stringa.
Le stringhe vengono trattate in maniera diversa dagli altri tipi di costanti.

“a” e’ una stringa
‘a’ e’ una costante di tipo carattere!

“io sono una stringa”

“Questa non puo’
essere una stringa”

Concetti di Base: operatori e punteggiatura

Gli operatori e la punteggiatura vanno trattati insieme in quanto in molti casi lo stesso simbolo riveste il ruolo di operatore o di elemento di punteggiature a seconda del posto in cui si trova e del modo con cui viene usato.

Gli operatori piu' semplici da comprendere sono:

```
+  -  *  /  %
```

Che rappresentano le quattro operazioni matematiche elementari e il modulo tra due interi.

In un programma gli operatori (come la punteggiatura) possono essere utilizzati per separare identificatori (es.: a+b).

Normalmente gli operatori sono costituiti da un solo simbolo ma in alcuni casi questo non e' vero ed esistono operatori costituiti da due simboli.

Concetti di Base: operatori e punteggiatura

```
#include <iostream>
int main(void)
{
        int a,b=7 , c = 3;
        double f=2.0,
                g=1.23;
        a=b%c;
        std::cout<<“a = “ << a <<std::endl;
        return 0;
}
```

esempio2.c

Concetti di Base: Precedenza e Associativita'

Cosi come in matematica, anche in un linguaggio di programmazione esistono delle regole di precedenza e di associativita' tra operatori. Consideriamo l'espressione:

```
1+5*3-4
```

Cosi come in matematica, l'operatore * (moltiplicazione) ha una priorita' piu' alta dell'operatore +. Per cui nella valutazione dell'espressione, verra' prima eseguito il prodotto tra 5 e 3 e dopo verranno eseguite le somme procedendo da sinistra verso destra (dato che gli operatori di sottrazione e somma hanno la stessa priorita' in questa caso vale la loro associativita').
Per cambiare la priorita' di esecuzione degli operatori si puo' ricorrera a le parentesi tonde.

```
(1+5)*(3-4)
```

NOTA 1 In questo caso le parentesi tonde svolgono a loro volta il ruolo di operatori. Operatori atti a alterare la priorita'.

NOTA 2 Solo le parentesi tonde possono svolgere questo ruolo!! (no [] e {} !!!)

Concetti di Base: Precedenza e Associativita'

	Operatori	Associativita'
	() ++ (*postfisso*) -- (*postfisso*)	Sin. - Des.
	+ (*unario*) – (*unario*) ++ (*prefisso*) -- (*prefisso*)	Des. - Sin.
	* / %	Sin. – Des.
	+ -	Sin. – Des.
	= += -= *= /=	Des. – Sin.

-a * b + c
I= ++a + b

Concetti di Base: Operatori di incremento/decremento

Un tipo di operatori che si incontreranno di frequente in C sono gli operatori di incremento e decremento (++ e --). Questi sono uno dei pochi casi in cui un operatore e' identificato da due simboli invece che da uno solo. Essi permettono di incrementare o decrementare di uno la variabile (identificatore) a cui sono applicati.
Di operatori di incremento e decremento ne esistono due tipi:

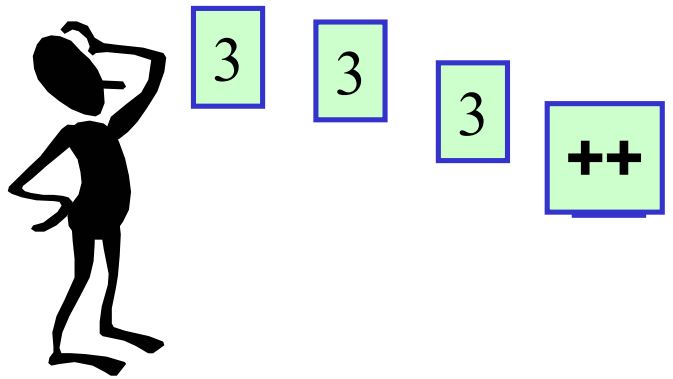
Prefissi:	++a , --numero_di_elementi , etc.
Postfissi:	a++ , numero_di_elementi-- , etc.

Il modo di operare nei due casi e' diverso. Nel primo caso (prefisso) il valore della variabile viene incrementato/decrementato e quindi restituito per poter proseguire il calcolo.
Nel secondo caso (postfissi) il valore della variabile viene salvato in un oggetto temporaneo (registro), viene quindi incrementato/decrementato e dall'operatore viene restituito il valore salvato **prima** dell'incremento/decremento.

Concetti di Base: Operatori di incremento/decremento

Prefisso

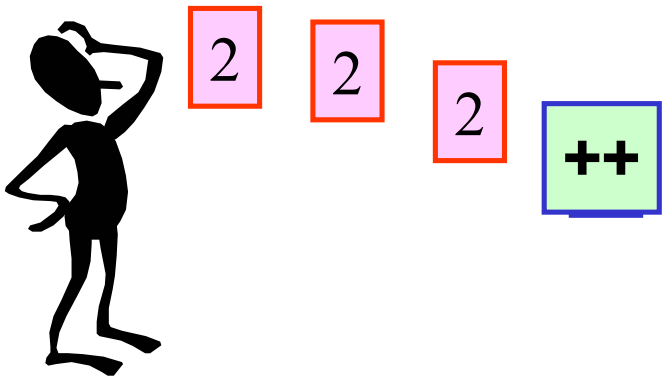
```
int a=2;
b=++a;
```



Concetti di Base: Operatori di incremento/decremento

Postfisso

```
int a=2;
b=a++;
```



Concetti di Base: Operatori di assegnazione

Gli operatori di assegnazione sono usati per assegnare a una variabile il valore di un altra o il risultato di un'espressione.

a=b	assegno ad a il valore di b
a*=b	assegno ad a il prodotto tra a e b
a/=b	assegno ad a il quoziente di a diviso b
a+=b	assegno ad a la somma di a piu' b
a-=b	assegno ad a la differenza di a meno b

```
a=b;
a=b+c;
a=b=c=2;
a*=b;
```

Concetti di Base: Operatori logici

Gli operatori logici operano tra due identificatori e verificano se la condizione in loro implicita e' verificata.
Un operatore logico ritorna quindi vero (**true**) se la condizione e' verificata, falso (**false**) altrimenti.

==	uguale	(a == b)
>	maggiore	(a>b)
<	minore	(a<b)
>=	maggiore o uguale	(a>=b)
<=	minore o uguale	(a<=b)
&&	AND logico	(a>b && a<c)
||	OR logico	(a<b || a>c)
!	NOT logico	(!(a>b))

Concetti di Base: Un Esempio

```
/* 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;
}
```

esempio3.c

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.

Concetti di Base: Il *Linker*

Finita la fase di compilazione si passa alla fase di ***link***. In questa fase il modulo di codice macchina prodotto dal compilatore a partire dal listato viene messo insieme ad altri moduli facenti parte di librerie (di sistema e non) per produrre un programma eseguibile.

Spesso si ricorre all'utilizzo di moduli facenti parte delle librerie standard del C per svolgere numerosi compiti.
printf() e' un esempio di un identificatore che identifica un modulo presente in una libreria standard. Questo modulo (funzione) gestisce l'output sullo schermo.
Dato che la fase di link segue quella di compilazione, il compilatore ha necessita' di sapere a priori le ***caratteristiche*** dei moduli che verranno collegati dal linker. Deve, infatti, verificare che la sintassi di utilizzo di questi moduli sia corretta. Ha necessita, quindi, di conoscere a priori la definizione di questi moduli.

I file che vengono inclusi dal pre-compilatore normalmente contengono proprio queste definizioni, per questo vengono inclusi all'inizio del listato e hanno l'estensione .h (***header***).

printf() e' definito all'interno del file stdio.h.

Concetti di Base: Un Esempio

```
#include <iostream>
#include <stdlib.h>
int main(void) {
        int k , n ;
        double r;
        std::cout<<“Verranno stampati un certo numero ”;
       std::cout<<“di valori interi, estratti casualmente”<< std::endl;
       std::cout<<“Quanti ne vuoi vedere?” <<std::endl;
        std::cin>>n ;
        for (k=0 ; k<n ; ++k){
                r = rand();
                std::cout<<“Numero_” <<n<< “  =  “ <<r <<std::endl;
        }
        return 0;
}
```

esempio4.c