FONDAMENTI DI INFORMATICA I

Finalità del Corso:

2.Fornire le nozioni principali per la programmazione in linguaggio C.

Testo adottato:

H.M.Deitel, P.J.Deitel, "C – Corso completo di programmazione", Apogeo Ed., Milano prof MANFREDI

•Introdurre l'ambiente di programmazione Matlab®.

Tutorial on-line – The MathWorks

Entrambi i linguaggi sono i più utilizzati sia in ambito universitario che nell'industria, per ricerca, sviluppo ed analisi.

Requisiti:

Il Corso richiede una buona conoscenza dei concetti relativi alla struttura dell'elaboratore ed alla codifica dell'informazione.

ORGANIZZAZIONE DEL CORSO

Lezioni: lunedì: 10.15-13.15(opzionale) , martedì: 8.30-10.15, giovedì: 8.30-11.15.

***Argomenti*:**

4. **INTRODUZIONE ALLA PROGRAMMAZIONE IN C**
5. **SVILUPPO DI PROGRAMMI STRUTTURATI - IL CONTROLLO DEL PROGRAMMA**
6. **LE FUNZIONI**
7. **I PUNTATORI**
8. **I VETTORI**
9. **LE STRUTTURE**
10. **INTRODUZIONE ALL'AMBIENTE MATLAB**

Esercitazioni: giovedì, Aula Informatica (I° gruppo: 14-15.30, II gruppo: 15.30-17).

***Argomenti*:**

- **Risoluzione di esercizi relativi agli argomenti trattati a lezione.**
- **Sviluppo ed approfondimento di alcuni argomenti (compilazione, formattazione I/O, le direttive del preprocessore, caratteri e stringhe), in quanto di più immediata comprensione tramite esempi applicativi.**
- **Introduzione all' ambiente di programmazione Matlab.**

MODALITA' DI ESAME

L'esame consiste in una **prova pratica** di programmazione in C e in Matlab, da svolgere in Aula Informatica.

Date esami sessione estiva (da confermare):

Giovedì 23 Giugno 2005 ore 9.00

Martedì 12 Luglio 2005 ore 9.00

INTRODUZIONE ALLA PROGRAMMAZIONE IN C

Sommario

3. Introduzione
4. Un semplice programma in C: Stampa di una linea di testo
5. Un altro semplice programma: Somma di due numeri interi
6. La memoria
7. L'aritmetica del C
8. Prendere delle decisioni: Operatori di uguaglianza e relazionali

L' ELABORATORE ELETTRONICO

- Elaboratore
  - Dispositivo in grado di svolgere calcoli ed effettuare scelte logiche
  - L' elaboratore elabora i *dati* sotto il controllo di insiemi di istruzioni dette "*programmi*"
- Hardware
  - Le varie componenti meccaniche ed elettroniche che costituiscono l'elaboratore:
  - Tastiera, monitor, mouse, dischi, memorie elettroniche, unità di elaborazione, etc.
- Software
  - L'insieme dei programmi che "girano" sull'elaboratore.

ORGANIZZAZIONE DELL'ELABORATORE

Si distinguono le seguenti unità logiche principali:

1. **Unità di ingresso**
   - Riceve informazioni dai dispositivi di ingresso (tastiera, mouse, etc.)
- **Unità di uscita**
   - Invia informazioni in uscita (al monitor, la stampante, etc)
3. **Unità di memoria centrale (RAM)**
   - Memoria ad accesso veloce, di capacità ridotta, non permanente
4. **Unità aritmetica e logica (ALU)**
   - Svolge i calcoli ed esegue le decisioni logiche
- **Unità centrale di elaborazione (CPU)**
   - Coordina le varie attività dell'eleboratore
- **Memorie secondarie**
   - Memorie ad alta capacità, permanenti, a velocità di accesso ridotta

LINGUAGGI DI BASSO E ALTO LIVELLO

Un calcolatore deve ricevere le istruzioni espresse in un opportuno linguaggio di programmazione e il computer comprende solo istruzioni scritte in linguaggio macchina (es: 0100110111101). Un programmatore di linguaggio macchina deve inserire ogni dato o comando in forma binaria.

```
+100001011001
+110001001100
+010011100111
```

Un linguaggio di basso livello (ad esempio l'assembler) fornisce esclusivamente l'accesso all'insieme delle istruzioni del calcolatore, le istruzioni macchina.

Programmare in linguaggio macchina è molto faticoso ed è molto difficile trovare errori di compilazione; quindi si pensò di sviluppare un linguaggio che si avvicinasse alla terminologia umana. Fu così inventato il linguaggio Assembly (es. assembler: DELETE = linguaggio macchina: 001001001).

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```


Successivamente, dopo il linguaggio assembly, si sviluppò un linguaggio sempre più vicino alla terminologia umana: il **linguaggio ad alto livello**. I linguaggi di alto livello impiegano parole della lingua inglese corrente (ad esempio DELETE, OPEN, MOVE) per esprimere singoli comandi che sostituiscono sequenze di centinaia di istruzioni in linguaggio macchina.

```
grossPay = basePay + overTimePay
```

La maggior parte dei linguaggi di *alto livello*, come ad esempio il Fortran, fornisce tutto ciò di cui un programmatore può aver bisogno direttamente nel linguaggio.

IL LINGUAGGIO C

Il C fu progettato ed implementato da Dennis Ritchie ed è in genere considerato un linguaggio ad **alto livello**. Fu inizialmente creato nei laboratori della AT&T Bell Laboratories (anni ’70) per lo sviluppo dei sistemi operativi.

Il **sistema operativo** è un programma di controllo che svolge operazioni fondamentali, risiede in una memoria interna permanente e interpreta i comandi di utente che richiedono varie specie di servizi, come la visualizzazione, la stampa o la copiatura di un file, il raggruppamento logico dei file in una directory o l'esecuzione di un programma. In altre parole *il sistema operativo si occupa di gestire tutte le periferiche del calcolatore, tutti i processi, i dati di input/output.*

Benché implementato su un sistema operativo **UNIX**, questo linguaggio non fu scritto per un particolare sistema operativo ma può essere utilizzato sotto sistemi operativi diversi come UNIX, DOS, OS, POWER PC, ecc.

Il linguaggio C viene però spesso definito un linguaggio di programmazione di "**medio livello**". Questo non indica che il linguaggio ha delle capacità in qualche modo limitate, ma riflette la sua *capacità di accedere alle funzioni di basso livello del sistema*.

Il C, come linguaggio di *medio livello* probabilmente non mette a disposizione tutti i costrutti che è possibile trovare nei linguaggi di alto livello, ma dispone dei blocchi base per poterli costruire.

Il C è un linguaggio di programmazione di tipo **imperativo**, il programma cioè viene scritto un file in *codice sorgente*, successivamente viene *compilato* e, se non ci sono errori, viene *eseguito*.

Il problema dei linguaggi imperativi è che sono legati alla **piattaforma** in cui vengono compilati. Se si compila un programma sotto Unix, quel programma potra' girare esclusivamente su sistemi operativi Unix.

In linguaggi di programmazione come Java, che sono di tipo interpreti, viene analizzato il codice sorgente, viene compilato riga per riga e sarà il web browser a occuparsi di farlo eseguire (es: Internet Explorer, Netscape, ecc.). Ciò comporta tuttavia che l'esecuzione di un programma interpretato sarà molto più lenta di quella dello stesso programma scritto in linguaggio imperativo ma, d'altro canto, il codice che scrivo sarà portatile cioè potrò eseguirlo in sistemi operativi diversi.

Strumenti per la compilazione

Il percorso di sviluppo di un programma, indipendentemente dal compilatore scelto, è composto dalle seguenti fasi:

3.Sviluppo dell'algoritmo - indipendente dal linguaggio -

4.Scrittura del programma

5.Compilazione

6.Collegamento (Link)

7.Esecuzione

Il primo passo non verrà qui affrontato in quanto prescinde dai problemi legati alla realizzazione del programma eseguibile a partire dal codice sorgente.

Scrittura del codice

In questa fase si scrive il codice del programma utilizzando un editor qualsiasi: quello reso disponibile dall'ambiente di compilazione, se disponibile, oppure un editor di testi qualsiasi, che non introduca alcuna formattazione (as esempio il BloccoNote).

Una volta scritto il programma si salva il file, che dovrà essere appunto un file di testo (un file scritto con Word non va bene, a meno che non venga salvato come file di testo); l'estensione del file abitualmente utilizzata è .c (ad indicare che si tratta di codice sorgente C). Questo file costituisce il file sorgente.

Per questa fase del processo di realizzazione di un programma non è necessario alcun ambiente di programmazione.

Compilazione del codice sorgente

Il file sorgente contiene le istruzioni specificate in C ma non può essere direttamente eseguito in quanto risulta del tutto incomprensibile al calcolatore: il codice in linguaggio C dovrà essere prima tradotto in linguaggio macchina comprensibile al processore del calcolatore.

La compilazione del programma produce un file intermedio, detto file oggetto, con estensione .obj, non ancora eseguibile ma che contiene una prima traduzione del codice sorgente.

Link ed esecuzione del programma

Il motivo per cui viene effettuato questo passo è l'inclusione delle funzioni di libreria di cui può far uso il programma.

Queste funzioni di libreria sono scritte dal produttore del compilatore e svolgono diversi compiti, dalle operazioni di ingresso/uscita, ad operazioni matematiche complesse.

Nel caso del linguaggio C vengono incluse le librerie standard specificate nella parte di direttive al preprocessore (ad esempio la libreria stdio.h).

Al termine di questa fase viene prodotto il file eseguibile del programma, la cui estensione è .exe.

L'esecuzione del programma viene avviata semplicemente scrivendo il nome del programma all'interno di una finestra DOS oppure facendo un doppio click sul file in un ambiente a finestre.

I FILE CREATI

Per riassumere, i file che vengono creati sono i seguenti:
.c - file sorgente prodotto da un editor di testo e preso in ingresso dal compilatore

.obj - file oggetto prodotto dal compilatore dopo aver processato i file sorgente e elaborato poi dal linker

.exe - file eseguibile prodotto dal linker.

L'ambiente Microsoft

La scrittura del programma sorgente può essere fatta mediante uno dei tanti editor.

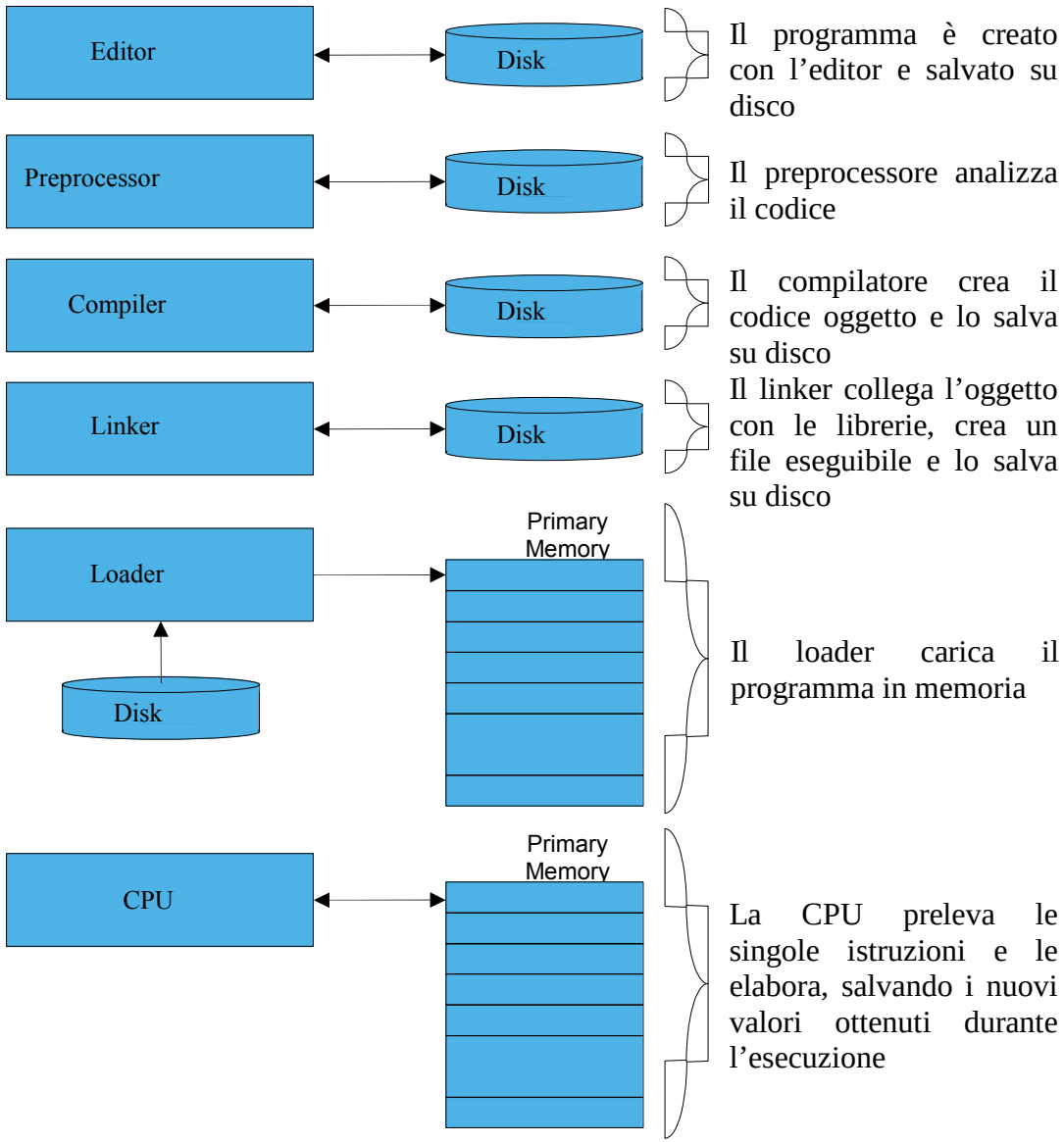
Compilazione e link: corrisponde solitamente alla voce Build e viene svolto in un unico passo.

L'esecuzione viene fatta selezionando la voce Run da menù e che semplicemente corrisponde a chiamare il programma creato.

Ambiente di sviluppo dei programmi in C

Fasi di un programma in C:

1. *Editing*
2. *Preprocessing*
3. *Compilazione*
4. *Link*
5. *Caricamento*
6. *Esecuzione*



SCRITTURA DI UN PROGRAMMA

- Prima di scrivere un programma:
  - Comprendere bene e in dettaglio il problema da risolvere
  - Pianificare con attenzione il modo di risolverlo
- Mentre si scrive un programma:
  - Ricordarsi di utilizzare i programmi disponibili
  - Fare uso di tecniche di programmazione efficienti

ALGORITMI

- Problemi di calcolo numerico
  - Si risolvono eseguendo una serie di operazioni in un ordine specifico

- **Algoritmo**: procedura in termini di:
  - *Azioni* da eseguire
  - *Ordine* in cui tali azioni devono essere eseguite

- Controllo del programma
  - Specifica l'ordine in cui le istruzioni devono essere eseguite

Rappresentazione di algoritmi

- **Pseudocodice**
  - **Linguaggio informale. Può essere di aiuto per descrivere un algoritmo**
- **Diagramma di flusso**
  - **Rappresentazione grafica di un algoritmo**
  - **Si disegna utilizzando dei simboli grafici particolari collegati da frecce dette linee di flusso.**
  - **Rettangolo: indica qualsisi tipo di azione.**
  - **Rombo: indica un confronto**
  - **Ovale: indica l'inizio e la fine del programma, o una parte di codice.**
- **Strutture di controllo ad un ingresso/un'uscita**
  - **Connettono il punto di uscita di una struttura a quello di ingresso della successiva**
  - **Semplificano la scruttura dei programmi**

PSEUDOCODICE E STRUTTURE DI CONTROLLO

- **Pseudocodice**
  - **Linguaggio artificiale ma informale, che può essere di aiuto nello sviluppo e nella stesura di un algoritmo.**
  - **E' simile alla lingua inglese parlata**
  - **Non può essere eseguito dall'elaboratore**
  - **Può aiutare a "buttar giù" un programma prima di scriverlo**
  - **E' facile da tradurre in un programma in C**
  - **Consiste unicamente di istruzioni eseguibili**
- **Strutture di controllo – linguaggi strutturati**
  - **Tutti i programmi possono essere scritti facendo uso di tre sole strutture di controllo:**
    - **Strutture sequenziali: Tutti i programmi sono eseguiti sequenzialmente se non diversamente specificato.**
    - **Strutture di selezione: Il C ha tre tipi di tali strutture: `if`, `if/else`, and `switch` .**
    - **Strutture di ripetizione: Il C ha tre tipi di tali strutture - `while`, `do/while`, and `for`.**

L'insieme dei caratteri del linguaggio C

Il linguaggio C non usa, e non richiede che venga utilizzato, ogni carattere presente sulla tastiera di un moderno calcolatore. Gli unici caratteri a cui si fa solitamente riferimento, sono i seguenti:

- A - Z
- a - z
- 0 - 9
- spazio . , : ; ' $ "
- # % & ! _ {} [] () < > |
- + - / * =

L'aspetto di un programma C

Tutti i programmi C sono costituiti da almeno una funzione: al crescere della complessità dei programmi e della capacità di organizzare un programma in sottoprogrammi, tale numero aumenterà.

La funzione che è sempre presente in tutti i programmi è chiamata **main**. Più in generale ci si riferisce alla funzione main come al "*programma principale*", in quanto è la prima funzione che viene chiamata quando si esegue un programma.

Il linguaggio C è costituito da sole **32 parole chiave riservate** che combinate realizzano la sintassi formale del linguaggio. Si noti che tutte le parole chiave vengono scritte in minuscolo. Una parola chiave non può essere utilizzata per altri scopi, quali ad esempio il nome di una variabile.

Il linguaggio di programmazione C può essere definito come un approccio strutturato alla programmazione

Programmazione strutturata: Sono necessarie solo tre strutture di controllo: sequenziale, di selezione, iterativa.

La struttura di un programma C

```
direttive per il preprocessore
dichiarazioni globali
main( )
{
variabili locali alla funzione main ;
istruzioni della funzione main
}
f1( )
{
variabili locali alla funzione f1 ;
istruzioni della funzione f1
}
f2( )
{
variabili locali alla funzione f2 ;
istruzioni della funzione f2
}
 . . .
 etc
```


Si noti l'utilizzo delle parentesi tonde e graffe.

Le **parentesi tonde ( )** vengono utilizzate in unione con i nomi delle funzioni, mentre le **parentesi graffe {}** vengono utilizzate per delimitare le espressioni.

Infine il **punto e virgola ;** indica la terminazione di un'espressione in C.

Il linguaggio C ha un formato piuttosto flessibile: espressioni lunghe possono essere continuate su righe successive senza problemi: il punto e virgola segnala al compilatore che è stata raggiunta la fine dell'espressione.

Inoltre, è possibile introdurre spazi indiscriminatamente, in genere allo scopo di dare un miglior aspetto al programma.

*Un errore molto comune è **dimenticarsi il punto e virgola**: in questo caso il compilatore concatenerà più linee di codice generando un'espressione priva di qualsiasi significato. Per questo motivo il messaggio d'errore che il compilatore restituisce non è la mancanza del punto e virgola, quanto la presenza di un qualcosa di incomprensibile. Attenzione dunque a non scordarsi i punti e virgola e ad interpretare i messaggi del compilatore.*

Direttive per il preprocessore

In generale qualsiasi programma comunica al mondo esterno il risultato dell'elaborazione. Quindi tutti i programmi avranno bisogno di funzioni di base per acquisire dati dall'esterno e per restituire i risultati. ***La libreria standard che contiene le funzioni di base per l'ingresso e l'uscita dei dati si chiama stdio.h e viene dichiarata prima del main.***

Il linguaggio C è in genere piuttosto flessibile per quanto riguarda il formato del codice: si possono scrivere più istruzioni su una stessa riga, si possono inserire spazi dove si desidera, ... Questo è vero ad eccezione delle direttive per il preprocessore.

Tutte le direttive per il preprocessore iniziano con un # e devono essere scritte all'inizio della riga. Un esempio è il seguente:

#include <stdio.h>

L'utilizzo delle parentesi ad angolo (< e >) indicano che il file in questione (stdio.h) deve essere cercato sul disco dove c'è tutto il resto dell'ambiente di compilazione del C.

La direttiva include non è terminata da un punto e virgola e il simbolo # deve essere il primo della riga.

LA LIBRERIA STANDARD DEL C

- **I programmi in C consistono di parti/moduli detti funzioni**
  - Un programmatore può creare le sue funzioni
    - Vantaggio: il programmatore sa esattamente come funzionano
    - Svantaggio: richiede tempo
  - I programmatori usano spesso le funzioni di libreria del C
    - Si usano come blocchi predefiniti
    - Se una funzione esiste già, è in genere meglio utilizzarla piuttosto che riscriverla
    - Le funzioni di libreria sono scritte in modo efficiente e sono ottimizzate e "portabili"
- **Chiarezza di un programma**
  - E' bene evitare di scrivere programmi complessi e di difficile lettura e comprensione, in quanto difficili da modificare.

Un semplice programma in C: scrittura di una riga di testo

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "Welcome to C!\n" );
8
9     return 0;
10 }
```

IL PROGRAMMA SORGENTE

```
Welcome to C!
```

IL RISULTATO

- **Commenti**
  - Il testo comprteso fra **/*** e ***/** è ignorato dall'elaboratore
  - E' utile per commentare e descrivere il programma
- **#include <stdio.h>**
  - Direttiva del preprocessore: dice all'elaboratore di caricare un certo programma
  - **<stdio.h>** Operazioni di input/output standard

COMMENTI AL PROGRAMMA

- `int main()`
  - Una delle funzioni in tutti i programmi in C deve essere `main`
  - Le **parentesi tonde** indicano una funzione
  - `int` significa che main "restituisce" un valore di tipo intero
  - Le **parentesi graffe** indicano un blocco di programma
    - Il "corpo" delle funzioni deve essere racchiuso fra parentesi graffe
- `printf( "Welcome to C!\n" );`
  - Indica l'esecuzione di un'azione
    - Stampa la stringa di caratteri compresa fra gli apici
  - Tutta la linea è detta un' istruzione
    - Tutte le istruzioni devono terminare con il ;
  - `\` - carattere di escape
    - Indica che **`printf`** deve eseguire qualcosa di particolare
    - `\n` è il carattere di "a capo"

COMMENTI AL PROGRAMMA (cont.)

- `return 0;`
  - E' un modo di "uscire" da una funzione
  - **`return 0`**, in questo caso indica che il programma termina normalmente
- Parentesi graffa `}`
  - Indica che si è raggiunta la fine del **`main`**
- Linker
  - Quando viene chiamata una funzione, il linker:
  - la individua nella libreria
  - la inserisce nel programma oggetto
  - Se il nome della funzione non è scritto correttamente, il linker segnala un errore perchè non riesce a trovarla nella libreria

```
/* A first program in C */

#include<stdio.h>
main( )
{
   printf("Welcome to C!\n");
}

/* Printing on one line with two printf statements */

#include<stdio.h>
main( )
{
   printf("Welcome ");
   printf("to C!\n");
}

/* Printing multiple lines with a single printf */

#include<stdio.h>
main( )
{
   printf("Welcome\nto\nC!\n");
}
```


Le variabili

Una variabile è un'area di memoria cui viene dato un nome, in grado di memorizzare un singolo valore (numerico o un carattere). Per poter utilizzare una variabile per salvare un valore è necessario *dichiararla specificandone il **nome** ed il **tipo*** di valore per i quali di desidera utilizzarla.

Dunque, **prima di utilizzare una variabile è necessario dichiararla**. In C la dichiarazione delle variabili viene messa all'inizio della funzione che ne farà uso.

Ci sono cinque tipi "primitivi":

int - numero intero.

float - numero in virgola mobile, ossia un numero con una parte frazionaria.

double - numero in virgola mobile in doppia precisione.

char - un carattere.

void - un tipo per usi speciali

Uno degli aspetti meno chiari del linguaggio è la quantità di memoria che ciascuno di questi tipi richiede. Essa infatti non è predefinita, in quanto dipende dal calcolatore.

Dichiarazione di tipo

La parte di dichiarazione del tipo specifica che tipi di valori, e quali comportamenti, sono supportati dall'entità che si sta dichiarando. Non vi è differenza tra variabili dichiarate attraverso un'unica dichiarazione o mediante dichiarazioni multiple dello stesso tipo. Ad esempio:

```
float x, y;
```

è equivalente a

```
float x;
float y;
```

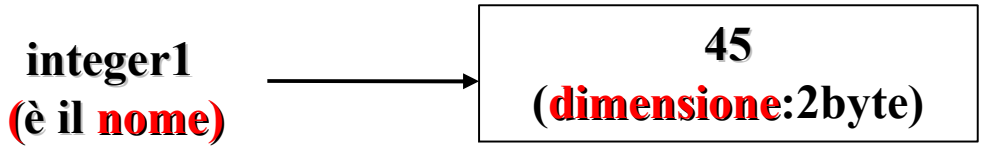
Ogni inizializzazione viene effettuata con l'assegnamento (utilizzando l'operatore =) di un'espressione del tipo appropriato. Ad esempio:

```
x = 3.14;
y = 2.81;
```


VARIABILI E MEMORIA

- **Variabili**
  - I nomi di variabili corrispondono a *locazioni* nella memoria dell'elaboratore
  - Ogni variabile ha un ***nome***, un ***tipo***, una ***dimensione*** e un ***valore***.
  - Tutte le volte che un nuovo valore è posto in una variabile (ad es. con **`scanf`**), esso sostituisce (e distrugge) il precedente valore memorizzato in quella locazione di memoria.
  - La semplice lettura di variabili dalla memoria non le modifica nè distrugge
- **Esempio**:

```
int integer1;       /* integer1 è di tipo intero */
integer1=45;        /* ad integer1 è assegnato il valore 45 */
```



Variabili intere

Una variabile di **tipo int** può memorizzare un valore compreso tra -32768 e +32767 (i valori derivano da 2^15, numero con segno rappresentabile in complemento a 2 su 16 bit). Per dichiarare una variabile di tipo intero si utilizza l'istruzione seguente:

int *variable_name*;

Ad esempio:

int a;

dichiara una variabile chiamata a di tipo intero. Per *assegnare un valore* alla variabile intera si usa l'espressione:

a=10;

Il linguaggio C utilizza il carattere **=** per **l'operatore di assegnamento**. Un'espressione della forma

a=10;

Significa: *prendi il valore numerico 10 e salvalo nella locazione di memoria associata alla variabile intera a*.

In generale, una variabile di tipo int occupa *2 byte* di memoria.

Variabili reali

Il C ha due tipi per dichiarare variabili con una parte frazionaria: *float* e *double*, che differiscono per la quantità di memoria che occupano e per la precisione che consentono di garantire

Una variabile di tipo **float** (da floating point, virgola mobile) consente di rappresentare numeri con una precisione di 7 cifre, per valori che vanno (più o meno) da 1.E-36 a 1.E+36. In generale, una variabile di tipo float occupa *4 byte* di memoria.

Una variabile di tipo **double** (da double precision, doppia precisione) consente di rappresentare un numero con una precisione di 13 cifre, per valori che vanno da 1.E-303 a 1.E+303.
In generale, una variabile di tipo float occupa *8 byte* di memoria.

Variabili carattere

Alla base del C ci sono numeri e caratteri, anche se questi ultimi vengono in realtà visti anch'essi come numeri, e più precisamente come il valore intero del codice ASCII corrispondente al carattere trattato. Per dichiarare una variabile di tipo carattere si utilizza la parola chiave **char**. Un carattere viene memorizzato in *un byte*.

Per esempio, la seguente è una dichiarazione di una variabile di tipo carattere:

```
char c;
```

Per assegnare, o memorizzare, un carattere nella variabile c è necessario indicare il carattere tra apici singoli, come segue:

```
c = 'A';
```

Si noti che *in una variabile di tipo char si può memorizzare un solo carattere*.

Si faccia attenzione al fatto che i singoli caratteri sono indicati tra apici singoli e non tra doppi apici:

```
char alfa, beta;
alfa = 'x';          /* -1- */
beta = "y";          /* -2- */
```

La prima istruzione è corretta, la seconda è semanticamente scorretta.

Cambiamento automatico di tipo

Nelle espressioni aritmetiche è possibile mischiare variabili numeriche di tipo diverso. In quasi tutti i casi *i valori con una precisione inferiore vengono convertiti ad una precisione superiore* ed utilizzati dagli operatori. Si consideri lo stralcio di codice seguente:

```
int a;
float b;
 ... ... = a*b;
```

L'espressione a*b viene valutata dopo aver convertito il valore intero in un float (avrà parte frazionaria nulla) e dopo aver svolto la moltiplicazione. Il risultato della moltiplicazione è un valore di tipo float. Nel caso in cui questo venga assegnato ad una variabile float tutto funziona normalmente. Nel caso in cui venga assegnato ad una variabile di tipo int il valore verrà automaticamente troncato (non arrotondato).

Questa conversione automatica di tipo vale anche per i caratteri. Un carattere viene rappresentato come un carattere ASCII esteso o un altro codice con un valore tra 0 e 255. Un altro modo di vedere la cosa è che una variabile di tipo char è una variabile intera di un solo byte in grado di memorizzare un valore compreso tra 0 e 255, che può anche essere interpretato come un carattere.

Aritmetica del C

- Calcoli aritmetici vengono utilizzati in quasi tutti i programmi
  - Si usa **`*`** per la moltiplicazione e **`/`** per la divisione
  - La divisione fra interi tronca il resto, ad es.

    `7 / 5` dà come risultato **`1`**
  - L'operatore modulo dà come risultato il resto delle divisione. Es.

    `7 % 5` dà come risultato **`2`**
- Precedenze fra operatori
  - Alcune operazioni aritmetiche vengono eseguite prima di altre (ad es. la moltiplicazione precede la somma)
    - Si usano le parentesi quando necessarie
  - Es.: Trovare la media delle tre variabili **`a`**, **`b`** e **`c`**
    - Non corretto: **`a + b + c / 3`**
    - Corretto: **`(a + b + c ) / 3`**

OPERATORI ARITMETICI

- **Operatori aritmetici:**

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	f + 7	f + 7
Subtraction	-	p - c	p - c
Multiplication	*	bm	b * m
Division	/	x / y	x / y
Modulus	%	r mod s	r % s

- **Regole di precedenza degli operatori:**

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to ri ght.

Espressione di assegnamento

Una volta dichiarata, una variabile è possibile utilizzarla (se si omette la dichiarazione il compilatore lo segnala con un errore) per memorizzare un valore ed in seguito manipolarlo. È possibile memorizzare un valore con la seguente sintassi:

***nome_variabile = valore*;**

Si faccia attenzione a non scambiare la variabile a cui assegnare il valore con quella che contiene il valore da assegnare: *ciò che sta* ***a sinistra dell'operatore*** *= è la destinazione dell'assegnamento e può essere solo una variabile. Ciò che sta* ***a destra*** *è la sorgente e può essere qualsiasi espressione che dia un valore.* L'esempio seguente:

```
a = 10;
```

memorizza il valore 10 nella variabile int chiamata a. Sarebbe sbagliato sintatticamente scrivere l'espressione al contrario:

```
10 = a;
```

e come errore sintattico il compilatore lo segnalerebbe. Non così nel caso in cui si desideri assegnare alla variabile a il valore contenuto nella variabile b:

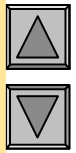
```
a = b;
```

In questo caso girare l'espressione produrrebbe un effetto diverso e non sarebbe sintatticamente scorretto!

```
b = a;
```


```
1  /      /* SOMMA DI DUE NUMERI INTERI */
2
3  #include <stdio.h>
4
5  int main()
6  {
7     int integer1, integer2, sum;         /* declaration */
8
9     printf( "Enter first integer\n" );   /* prompt */
10    scanf( "%d", &integer1 );            /* read an integer */
11    printf( "Enter second integer\n" );  /* prompt */
12    scanf( "%d", &integer2 );            /* read an integer */
13    sum = integer1 + integer2;           /* assignment of sum */
14    printf( "Sum is %d\n", sum );        /* print sum */
15
16    return 0;  /* indicate that program ended successfully */
17 }
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```



Outline

2. Dichiarazione di tipo (int)

5. Input dei dati

8. Calcolo della somma

10. Stampa dei risultati

14. Output del programma

Commenti al programma

- Già visti:
  - Commenti, **`#include <stdio.h>`** e **`main`**
- **`int integer1, integer2, sum;`**
  - Dichiarazione di variabili
    - Variabili: locazioni di memoria dove vengono memorizzati dei valori
  - **`int`** significa che la variabile può contenere valori interi (es.: **`-1`**, **`3`**, **`0`**, **`47`**)
  - **`integer1`**, **`integer2`**, **`sum`** Nomi di variabili (identificatori)
    - **Identificatori**: consistono di lettere, numeri (non possono iniziare con un numero), trattini; sono "case sensitive"
  - Le dichiarazioni devono essere poste prima delle istruzioni eseguibili, altrimenti viene segnalato un errore di sintassi dal compilatore

Commenti (cont.)

- **`scanf( "%d", &integer1 );`**
  - Acquisisce i dati dall'utente
    - **`scanf`** usa l'input standard (in genere la tastiera)
  - Qui **`scanf`** ha due argomenti:
    - **`%d`** – indica che il dato deve essere un intero
    - **`&integer1`** – locazione di memoria dove memorizzare la variabile
    - **`&`** per ora non ne parliamo – ricordarsi di metterlo prima del nome della variabile nelle istruzioni **`scanf`**
      - Ne parleremo più avanti
  - L'utente risponde allo **`scanf`** digitando un numero e poi premendo il tasto enter (return)

Commenti (cont.)

- **=** (operatore di assegnazione)
  - Assegna un valore ad una variabile
  - E' un operatore "binario" (cioè ha due operandi)

    ```
    sum = variable1 + variable2;
    ```

    a **sum** è assegnato il risultato dell'operazione **variable1 + variable2;**
  - La variabile sum che riceve il risultato deve essere sulla sinistra dell'operatore =
- **printf("Sum is %d\n", sum);**
  - Simile a **scanf** - **%d** significa che verrà stampato un valore intero
    - **sum** specifica quale valore verrà stampato
  - I calcoli possono essere effettuati all'interno dell'istruzione **printf**

    ```
    printf( "Sum is %d\n", integer1 + integer2 );
    ```


Scrivere un programma che effettua la somma di due numeri interi acquisiti da tastiera e che stampa il risultato sullo schermo

```
#include<stdio.h>
void main()      /* void significa che main non “restituisce” nulla*/
{
      /* dichiarazione delle variabili. n1 e n2 servono per */
      /* immagazzinare i valori di ingresso, somma per il risultato */
      int n1, n2, somma;
      /* lettura dei valori da sommare */
      printf("Inserisci il primo numero");
      scanf("%d", &n1);
      printf("Inserisci il secondo numero");
      scanf("%d", &n2);
      /* calcolo della somma */
      somma = n1 + n2;
      /* stampa del risultato */
      printf("Il risultato della somma e': %d", somma);
}
```


Scrivere un programma che acquisisce due numeri interi da tastiera e calcola, stampando a video, la somma e la media dei due valori

```
#include<stdio.h>
void main()
{
      int n1, n2, somma;          /* dichiarazione delle variabili di tipo int */
      double media;               /* dichiarazione della variabile di tipo double */
      /* lettura dei valori da sommare */
      printf("Inserisci il primo numero");
      scanf("%d", &n1);
      printf("Inserisci il secondo numero");
      scanf("%d", &n2);
      somma = n1 + n2;                    /* calcolo della somma */
      media = (n1 + n2)/2.0;              /* calcolo della media */
      /* stampa del risultato */
      printf("La somma e': %d\n", somma);
      printf("la media e': %g\n", media);
}
```


Gli identificatori di formato %

Gli identificatori previsti dall'ANSI C sono:

Tipo	**Espressione**	**A video**
%c	char	singolo carattere
%d (%i)	int	intero con segno
%e (%E)	float or double	formato esponenziale
%f , %lf	float or double	reale con segno
%g (%G)	float or double	utilizza %f o %e in base alle esigenze
%o	int	valore base 8 senza segno
%p	pointer	valore di una variabile puntatore
%s	array of char	stringa (sequenza) di caratteri
%u	int	intero senza segno
%x (%X)	int	valore base 16 senza segno

Caratteri di controllo

Ci sono alcuni *codici di controllo* che non stampano caratteri visibili ma contibuiscono a formattare ciò che viene stampato:

\n nuova linea

\r inizio linea corrente (senza una nuova linea)

\t tabulatore

\' visualizza un apice

\” visualizza le virgolette

\\ visualizza un backslash

Scrivere un programma che converte una misura in pollici nell'equivalente in centimetri. 1 pollice = 2.54 centimetri

```
#include<stdio.h>
void main()
{
        /* dichiarazione delle variabili. */
        double dpoll, dcm;
        /* lettura del valore da convertire */
        printf("Inserisci la dimensione in pollici");
        scanf("%lf", &dpoll);        /*lf indica un valore doppia precisione*/
        /* calcolo dell'equivalente in cm */
        dcm = dpoll * 2.54;
        /* stampa del risultato . %g non visualizza gli zeri superflui */
        printf("%g in pollici = %g in centimetri\n", dpoll, dcm);
}
```


Scrivere un programma che converte la temperatura espressa in gradi Celsius in gradi Fahrenheit. La relazione tra gradi Celsius e Fahrenheit è C = (5/9)*F - 32

```
#include<stdio.h>
void main()
{
        /* dichiarazione delle variabili. */
        double dcel, dfar;
        /* lettura del valore da convertire */
        printf("Inserisci la tempertura in gradi Celsius: ");
        scanf("%lf", &dcel);            /*lf indica un valore doppia precisione*/
        /* calcolo dell'equivalente */
        dfar = 9.0/5.0 * (dcel + 32);
        /* stampa del risultato */
        /* g non visualizza gli zeri superflui */
        printf("%g in gradi Celsius = %g in Fahrenheit\n", dcel, dfar);
}
```


ISTRUZIONI E PAROLE CHIAVE

- Istruzioni eseguibili
  - Eseguono azioni (calcoli, ingresso/uscita di dati)
  - Effettuano decisioni
    - Stampa di "promosso" or “bocciato" sulla base di un voto di esame
- Parole chiave
  - Sono parole speciali, riservate per il C
  - Non possono essere usate come nomi di variabili

Parole chiave

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

Espressioni e operatori logici

Un' *espressione logica* è un'espressione che genera come risultato un valore vero o falso e viene utilizzata dalle istruzioni di controllo del flusso di esecuzione.

La valutazione delle espressioni logiche può avere o un valore diverso da zero (interpretato come vero) oppure un valore pari a zero (interpretato come falso).

Un semplice esempio di espressione logica è una variabile: se il suo contenuto è diverso da zero, allora l'espressione è vera, altrimento l'espressione è falsa.

Le espressioni logiche possono contenere gli *operatori relazionali* usati per confrontare tra loro dei valori.

Operatori di uguaglianza e di relazione

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

LA STRUTTURA IF

La struttura di controllo if si usa per effettuare una scelta fra condizioni alternative.

- Se una certa condizione è "vera", viene eseguito il "corpo" dell'istruzione if e il programma prosegue con l'istruzione successiva.
- Se la condizione è falsa l'istruzione viene ignorata e il programma prosegue con l'istruzione successiva.

0 è falso, altrimenti è vero

- Il controllo torna al programma dopo la struttura **if**
- **Pseudocodice**: *Se il voto dello studente è maggiore o uguale a 60*
  scrivi "Passed"
- L'*indentazione* rende il programma di più facile lettura (il C ignora gli spazi bianchi)

Codice in C:

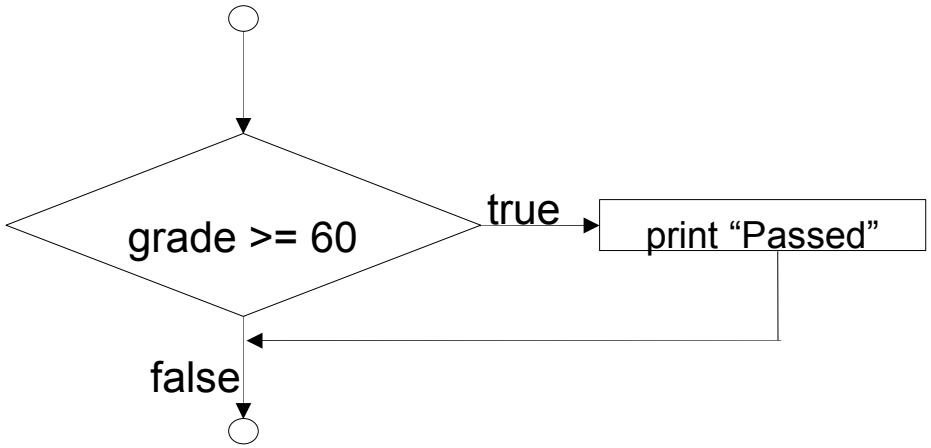
```
if ( grade >= 60 )
    printf( "Passed\n" );
```


LA STRUTTURA DI SELEZIONE IF

Diagramma di flusso: Simbolo del rombo (simbolo di decisione) – indica che va presa una decisione

- Contiene un'espressione che può essere **true** o **false**
- Verifica la condizione e prosegue secondo la direzione appropriata

- La struttura **if** è una struttura ad un ingresso ed un'uscita



NOTA:

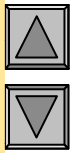
Può essere presa una decisione su qualunque espressione:

zero - **`false`**

Diverso da zero - **`true`**

Esempio:

`3 - 4` è **`true`**



Outline

```
1  /* CONFRONTO FRA DUE NUMERI INTERI */
2
3
4  #include <stdio.h>
5
6  int main()
7  {
8     int num1, num2;
9
10    printf( "Enter two integers, and I will tell you\n" );
11    printf( "the relationships they satisfy: " );
12    scanf( "%d%d", &num1, &num2  );    /* read two integers */
13
14    if ( num1 == num2 )
15       printf( "%d is equal to %d\n", num1, num2 );
16
17    if ( num1 != num2 )
18       printf( "%d is not equal to %d\n", num1, num2 );
19
20    if ( num1 < num2 )
21       printf( "%d is less than %d\n", num1, num2 );
22
23    if ( num1 > num2 )
24       printf( "%d is greater than %d\n", num1, num2 );
25
26    if ( num1 <= num2 )
27       printf( "%d is less than or equal to %d\n",
28               num1, num2 );
```

Dichiarazione delle variabili

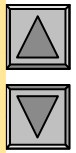
Input

Istruzione if

e

stampa risultato del confronto

```
29
30    if ( num1 >= num2 )
31       printf( "%d is greater than or equal to %d\n",
32               num1, num2 );
33
34    return 0;   /* indicate program ended successfully */
35 }
```



Outline

Uscita dal programma

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

Visualizzazione dei risultati (il programma viene fatto “girare” due volte)

if-else

L'istruzione if-else permette di scegliere quale istruzione eseguire tra quelle che la seguono. La sintassi è:

if (*espressione*)
**   *istruzione1***
else
**   *istruzione2***

Viene valutata l'espressione: se il suo valore è diverso da 0 (spesso uguale a 1 - *vero*), allora si passa ad eseguire *istruzione1*; altrimenti (valore uguale a 0 - *falso*), se vi è una clausola else, si passa all'istruzione *istruzione2*. La clausola else è opzionale.

È possibile costruire una sequenza di test collegando un altro if alla clausola else dell'if precedente.

```
...
if (a==0)
    printf("numero nullo\n");
else if ((a%2)==0)
    printf("numero pari\n");
else
    printf("numero dispari\n");
 ....
```

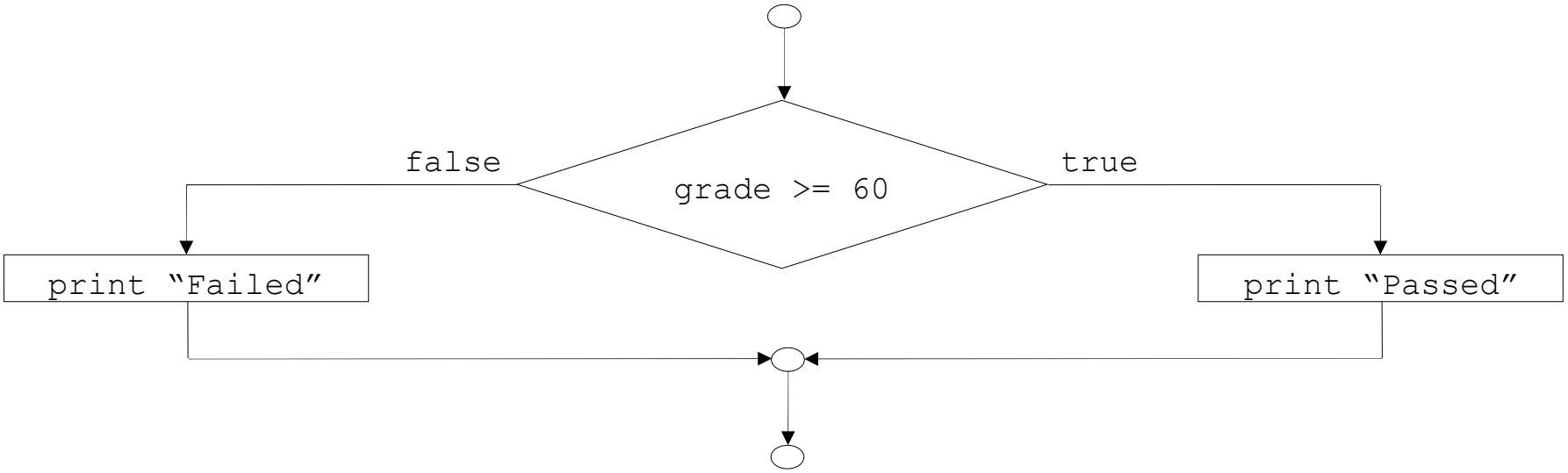

LA STRUTTURA DI SELEZIONE IF/ELSE

if - Esegue un'azione solo se la condizione è vera (**true)**.

if/else – Viene eseguita un'azione diversa a seconda che la condizione sia vera (**true)** o falsa (**false)**

Codice C:

```
if ( grade >= 60 )
        printf( "Passed\n");
else
        printf( "Failed\n");
```



```
/* Utilizzo di if-else */
/* Dato un numero intero, si verifica se è > o < di 100 */
#include <stdio.h>

main ()
{
int i;

printf(“Inserisci un intero: ");
scanf("%d", &i);

if(i<100)
  printf("minore di 100\n");
else
 printf("maggiore o uguale a 100\n");
}
```


```
/* Verifica se il numero inserito è <100 o >=100 */
#include <stdio.h>

/*variabili globali*/
int i;
int mag_100;
int min_100;

main ()
{
mag_100 = 0;   /*inizializzazione delle variabili*/
min_100 = 0;
printf(“Inserisci un intero: ");
scanf("%d", &i);
if(i<100) {
            printf("minore di 100\n");
            min_100 = 1;
            }
else {
            printf("maggiore o uguale a 100\n");
            mag_100 = 1;
            }
}
```


Scrivere un programma che determina se un numero è pari o dispari

```
#include <stdio.h>
void main()
{
        int n;
        printf("Inserisci un numero intero ");
        scanf("%d", &n);
        if ((n % 2) == 0)
            printf("Il numero %d è pari\n", n);
        else
            printf("Il numero %d è dispari\n", n);
}
```