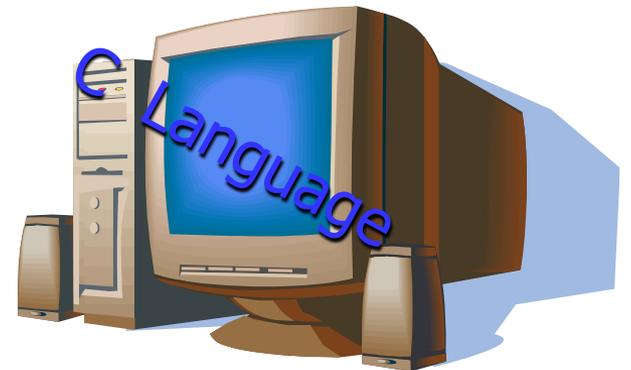


# Introduzione al linguaggio C

- I fondamenti del linguaggio C
  - ↳ Lo sviluppo dei programmi
  - ↳ Compilazione dei file sorgente, link e caricamento
  - ↳ La libreria di run-time
  - ↳ Le funzioni: anatomia di una funzione C
  - ↳ L'impaginazione dei file sorgente
  - ↳ Le funzioni *main()*, *printf()*, *scanf()*
  - ↳ Il preprocessore



# I fondamenti del linguaggio C

# Cos'è un linguaggio

- **Definizione 1** – *Un linguaggio è un insieme di parole e di metodi di combinazione delle parole usati e compresi da una comunità di persone*
- È una definizione poco precisa perché...
  - ✦ ...non evita le ambiguità dei linguaggi naturali
  - ✦ ...non si presta a descrivere processi computazionali automatici
  - ✦ ...non aiuta a stabilire proprietà
- **Definizione 2** – Il linguaggio è un sistema matematico che consente di rispondere a domande come:
  - ✦ quali sono le frasi lecite?
  - ✦ si può stabilire se una frase appartiene al linguaggio?
  - ✦ come si stabilisce il significato di una frase?
  - ✦ quali sono gli elementi linguistici primitivi?

# Linguaggi di programmazione di alto livello – 1

- Consentono al programmatore di trattare oggetti complessi senza doversi preoccupare dei dettagli della particolare macchina sulla quale il programma viene eseguito
- Richiedono un **compilatore** o un **interprete** che sia in grado di tradurre le istruzioni del linguaggio di alto livello in istruzioni macchina di basso livello, eseguibili dal calcolatore
- Un compilatore è un programma traduttore simile ad un assembler, ma più complesso, infatti...
  - ✦ ...esiste una corrispondenza biunivoca fra istruzioni in assembler ed istruzioni macchina
  - ✦ ...ogni singola istruzione di un linguaggio di alto livello corrisponde a molte istruzioni in linguaggio macchina: quanto più il linguaggio si discosta dal linguaggio macchina, tanto più il lavoro di traduzione del compilatore è difficile

# Linguaggi di programmazione di alto livello – 2

- I linguaggi che non dipendono dall'architettura della macchina offrono due vantaggi fondamentali:
    - ◆ i programmatori non devono cimentarsi con i dettagli architetturali di ogni calcolatore
    - ◆ i programmi risultano più semplici da leggere e da modificare
- ⇒ portabilità, leggibilità, manutenibilità

# Linguaggi di programmazione di alto livello – 3

- **Portabilità:** i programmi scritti per un calcolatore possono essere utilizzati su qualsiasi altro calcolatore, previa ricompilazione
- **Leggibilità:** la relativa similitudine con i linguaggi naturali rende i programmi più semplici, non solo da scrivere, ma anche da leggere
- **Manutenibilità:** facilità nell'effettuare modifiche di tipo correttivo, perfettivo, evolutivo e adattivo
- La possibilità di codificare algoritmi in maniera astratta si traduce in una migliore comprensibilità del codice e quindi in una più facile **analisi di correttezza**

# Compilatori ed interpreti – 1

- Affinché un programma scritto in un qualsiasi linguaggio di programmazione sia comprensibile (e quindi eseguibile) da parte di un calcolatore, occorre tradurlo dal linguaggio originario al linguaggio della macchina
- Ogni traduttore è in grado di comprendere e tradurre un solo linguaggio
- Il traduttore converte il testo di un programma scritto in un particolare linguaggio di programmazione (**sorgente**) nella corrispondente rappresentazione in linguaggio macchina (programma **eseguibile**)

PROGRAMMA	TRADUZIONE
main()	
{ int A;	00100101
...	
A=A+1;	11001..
if....	1011100..

# Compilatori ed interpreti – 2

- **Compilatore**: opera la traduzione di un programma sorgente (scritto in linguaggio di alto livello) in un programma oggetto direttamente eseguibile dal calcolatore
  - PRIMA si traduce tutto il programma
  - POI si esegue la versione tradotta
- **Interprete**: traduce ed esegue il programma sorgente, istruzione per istruzione
  - Traduzione ed esecuzione sono intercalate

# Compilatori ed interpreti – 3

- **Esempio di compilatore**

- Dobbiamo sottoporre un curriculum, in inglese, ad una azienda, ma non conosciamo l'inglese
- Abbiamo bisogno di un traduttore che traduca quanto scritto da noi dall'italiano all'inglese
  - ◆ contattiamo il traduttore
  - ◆ il traduttore riceve il testo da tradurre
  - ◆ il traduttore fornisce il testo tradotto
  - ◆ possiamo sottoporre il nostro curriculum all'azienda

# Compilatori ed interpreti – 4

- **Esempio di interprete**

- Dobbiamo incontrare un manager cinese per motivi di lavoro ma non conosciamo il cinese
- Abbiamo bisogno di un interprete che traduca il nostro dialogo
  - ◆ contattiamo l'interprete
  - ◆ parliamo in italiano, in presenza dell'interprete
  - ◆ **contemporaneamente** l'interprete comunica al manager cinese quanto detto da noi e viceversa
- Il compito dell'interprete si svolge contestualmente all'incontro col manager cinese

# Compilatori ed interpreti – 5

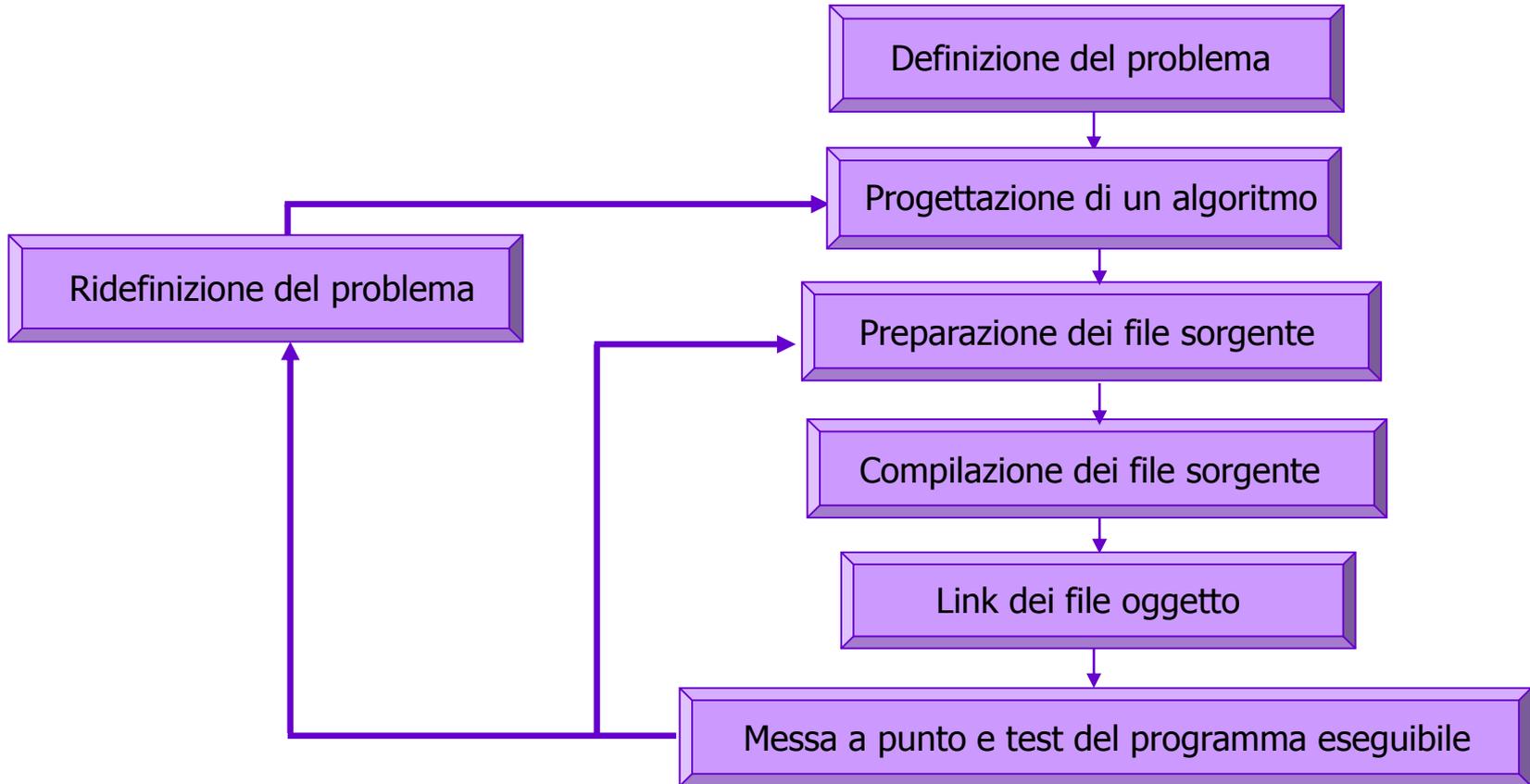
- Riassumendo...

- I compilatori traducono un intero programma dal linguaggio  $\mathcal{L}$  al linguaggio macchina della macchina prescelta:
  - ◆ traduzione e esecuzione procedono separatamente
  - ◆ al termine della compilazione è disponibile la versione tradotta del programma
  - ◆ la versione tradotta è però specifica per quella macchina
  - ◆ per eseguire il programma basta avere disponibile la versione tradotta (non è necessario ricompilare)
- Gli interpreti invece traducono e immediatamente eseguono il programma *istruzione per istruzione*, infatti:
  - ◆ traduzione ed esecuzione procedono insieme
  - ◆ al termine non vi è alcuna versione tradotta del programma originale
  - ◆ se si vuole rieseguire il programma occorre anche ritradurlo

# Compilatori ed interpreti – 6

- L'esecuzione di un programma compilato è più veloce dell'esecuzione di un programma interpretato
- I linguaggi interpretati sono tipicamente più flessibili e semplici da utilizzare (nei linguaggi compilati esistono maggiori limitazioni alla semantica dei costrutti)
- Per distribuire un programma interpretato si deve necessariamente distribuire il codice sorgente, rendendo possibili operazioni di plagio
- Nei programmi interpretati, è facilitato il rilevamento di errori di run-time

# Lo sviluppo dei programmi – 1

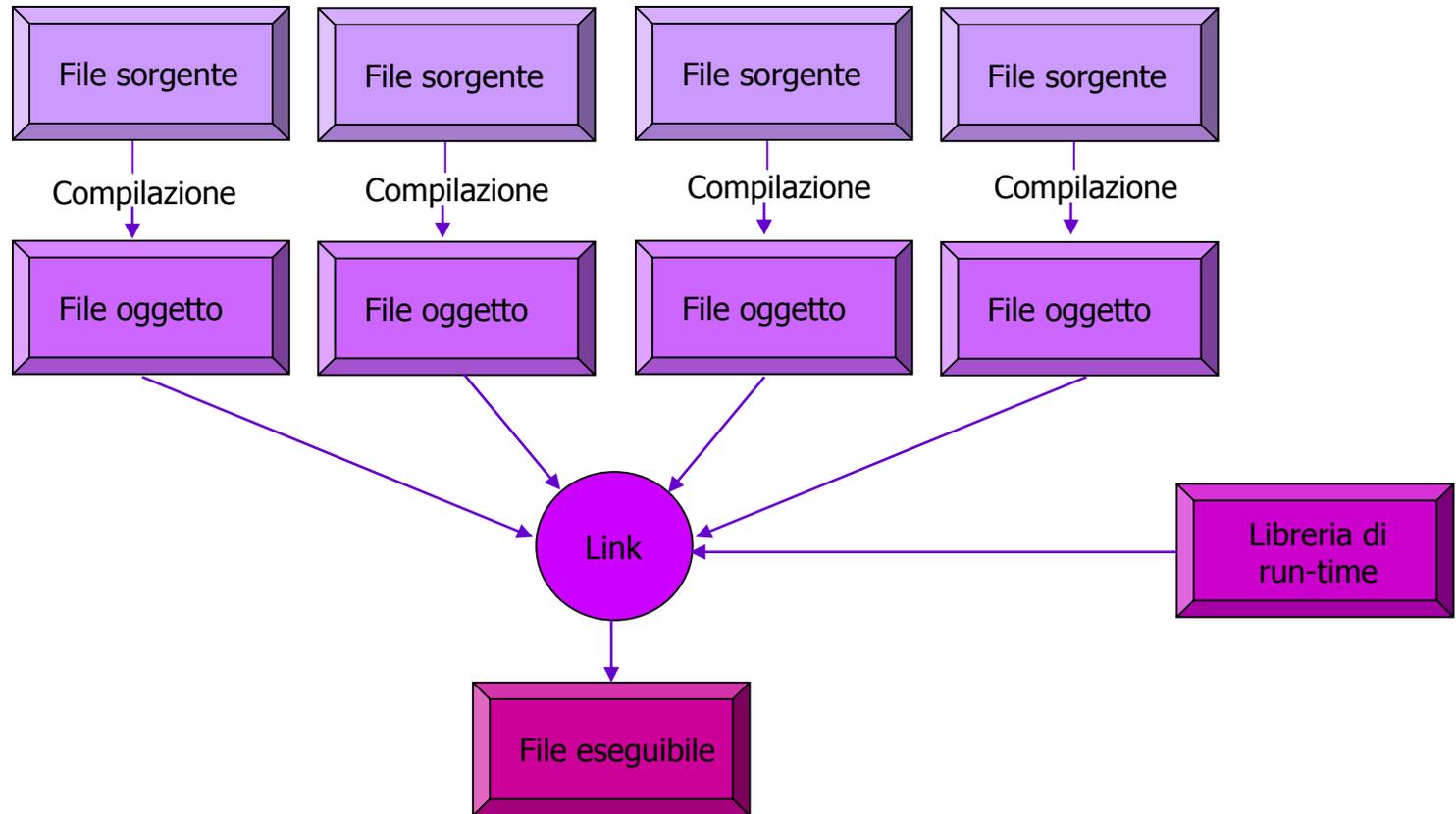


Fasi dello sviluppo di un programma

# Lo sviluppo dei programmi – 2

- Lo sviluppo dei programmi si compone di tre fasi fondamentali:
  - **Analisi** – Descrizione del problema e definizione di un algoritmo di risoluzione
  - **Programmazione** – Sviluppo del software:
    - ◆ Preparazione di ciascun file sorgente, mediante un editor
    - ◆ Compilazione di ogni file sorgente, per produrre i file oggetto
    - ◆ Link dei file oggetto, per produrre un programma eseguibile
  - **Caricamento in memoria ed esecuzione** del programma sulla particolare istanza del problema da risolvere

# Lo sviluppo dei programmi – 3



- I codici sorgente ed oggetto possono essere suddivisi in più file, il codice eseguibile di un programma risiede in un unico file

# La compilazione dei file sorgente

- Al termine della fase di progettazione, deve essere individuato un insieme di routine, chiamate **funzioni**, ognuna delle quali risolve una piccola parte del problema di programmazione
- La fase successiva è la stesura del codice per ogni funzione, mediante la creazione e la scrittura di file di testo in linguaggio C, che prendono il nome di **file sorgente**
- Il **compilatore** ha il compito di tradurre il codice sorgente in codice macchina ed è esso stesso un programma (o un gruppo di programmi) che deve essere eseguito
- I dati in ingresso al compilatore sono costituiti dal codice sorgente, mentre in uscita viene prodotto il **codice oggetto**, che rappresenta una fase intermedia tra il codice sorgente ed il **codice eseguibile**

# Il link ed il caricamento – 1

- I file oggetto creati dal compilatore vengono trasformati in un unico file eseguibile mediante il programma di **link**
- Infatti, il **linker**, nel caso in cui la costruzione del programma oggetto richieda l'unione di più moduli (compilati separatamente), provvede a collegarli formando un unico **programma eseguibile**
- Il **linker** provvede anche alla risoluzione dei riferimenti a funzioni e variabili definite altrove (ad es., in librerie standard o definite dall'utente)

# Il link ed il caricamento – 2

- Nonostante l'operazione di link sia gestita automaticamente in alcuni sistemi operativi (per es., UNIX), il linker è un programma distinto dal compilatore: in alcuni ambienti il programma di link deve essere lanciato separatamente
- Infine, durante la fase di caricamento (o *loading*), il programma eseguibile viene caricato nella memoria principale; la maggior parte dei sistemi operativi carica automaticamente un programma quando viene digitato il nome (o "cliccata" l'icona) di un file eseguibile

# Ambiente di sviluppo

- È l'insieme dei programmi che, complessivamente, consentono la scrittura, la verifica e l'esecuzione di nuovi programmi (fasi di sviluppo)
- Oltre a **editor** (per la scrittura dei file sorgente), **compilatore**, **linker** e **loader**, può includere un programma di rilevamento e correzione degli errori
- **Debugger**: consente di eseguire passo passo un programma, controllandone la correttezza, al fine di scoprire ed eliminare errori non rilevati in fase di compilazione (lessicali o sintattici)
- Esempio: **Code::blocks**

# C oppure C++ ?

- Noi useremo prettamente istruzioni in C, ricorrendo eventualmente ad istruzioni C++ riguardante le fasi input/output come cout e cin (*streaming*)
- Infatti, grazie allo streaming si risolvono tutte le problematiche dovute all'uso di specificatori all'interno delle funzioni di I/O printf() e scanf() in quanto sarà il compilatore stesso a farsi carico dell'assegnazione del giusto tipo ad una data variabile



# La libreria di run-time –1

- Il set di istruzioni del C è molto limitato: le primitive più comunemente utilizzate (es. I/O, matematiche) sono contenute nelle librerie standard del C sotto forma di funzioni
- Ovvero, in C, molte operazioni vengono delegate alla libreria di run-time, che contiene programmi di supporto
- Le funzioni sono divise in gruppi, quali I/O (comunicazione con le periferiche), gestione della memoria, operazioni matematiche e manipolazione di stringhe
- Per ogni gruppo di funzioni esiste un file sorgente, chiamato *file header*, contenente le informazioni necessarie per utilizzare le funzioni

# La libreria di run-time –2

- I nomi dei file header terminano, per convenzione, con l'estensione ".h" (ad es., *stdio.h* è il file header dello standard I/O)
- Per includere un file header in un programma, occorre inserire nel codice sorgente l'istruzione

```
#include <nomefile.h>
```

- **Esempio:** Per utilizzare *printf()*, che permette di visualizzare dati su terminale, è necessario inserire nel sorgente la linea di codice

```
#include <stdio.h>
```

La direttiva **#include** è rivolta al **preprocessore**

# Riassumendo...

## • Traduzione in linguaggio macchina

- Analisi (lessicale, grammaticale, contestuale)
- Trasformazione del programma sorgente in programma oggetto (forma più vicina al linguaggio macchina):
  - ◆ Creazione della tabella dei simboli
  - ◆ Ottimizzazioni (rimozione ripetizioni, eliminazione cicli, gestione registri, etc.)

## # Collegamento

- Il codice oggetto così formato...
  - ◆ ...può ancora contenere simboli irrisolti e riferimenti esterni a programmi di servizio (librerie di run-time)
  - ◆ ...contiene indirizzi relativi
- Il linker collega i diversi moduli oggetto

# Riassumendo...

## #Caricamento in memoria

- Il loader serve per caricare in memoria un programma rilocabile
- Nel caricamento vengono fissati tutti gli indirizzi relativi
  - ◆ variabili, salti, etc.
- Vengono caricati anche i programmi di supporto, se necessari

# ESEMPIO in C

```
#include<stdio.h>
```

Include la libreria standard di I/O

```
main()
```

Definisce una funzione *main()* che non riceve alcun valore come argomento

```
{
```

```
    printf("Salve, mondo\n");
```

*main()* richiama la funzione di libreria *printf()* per stampare la sequenza di caratteri specificata; \n indica il *newline*

```
}
```

# ESEMPIO in C++

```
#include<iostream>

using namespace std;

main()
{
    cout << "Salve, mondo\n";
}
```

Include la libreria standard di I/O

Indica che si intende usare lo spazio dei nomi standard

Definisce una funzione *main()* che non riceve alcun valore come argomento

*main()* richiama l'operatore di streaming `cout` della libreria `iostream` per l'output a video; `\n` indica il *newline*

# Compilazione

Per C (printf, scanf, etc):

```
gcc nomeprog.c -o nomeprog
```

Per C++ (cin, cout, etc):

```
g++ nomeprog.c -o nomeprog
```

# IMPORTANTE!

cout, cin **oppure** printf, scanf?

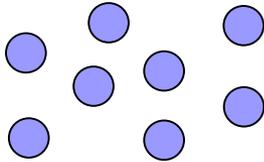
Laddove non specificato altrimenti, si potranno usare a propria convenienza le funzioni di I/O

Quindi, basterà solo fare attenzione nella specifica delle inclusioni delle librerie di C (es: `#include <stdio.h>`) e C++ (es: `#include <iostream>`) per l'utilizzazione delle opportune funzioni di I/O (printf e scanf per C; cout e cin per C++)

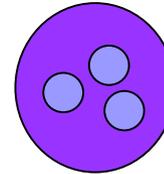
# Le funzioni – 1

- Una funzione C è costituita da un insieme di istruzioni del linguaggio C
- I programmi sono sviluppati definendo livelli gerarchici di funzioni:
  - le funzioni di basso livello svolgono le operazioni più semplici
  - le funzioni di alto livello sono definite “per combinazione” di funzioni di livello inferiore
- L'ingegneria del software si basa sul concetto di **gerarchia di componenti**, definendo strutture complesse a partire da componenti semplici

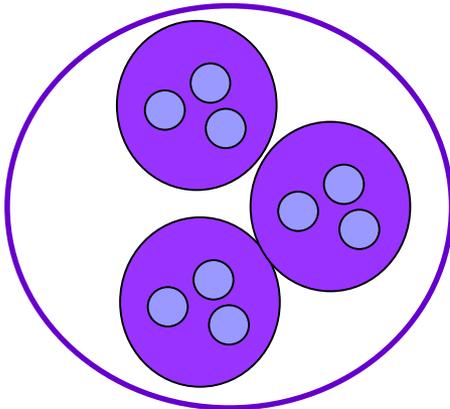
# Le funzioni – 2



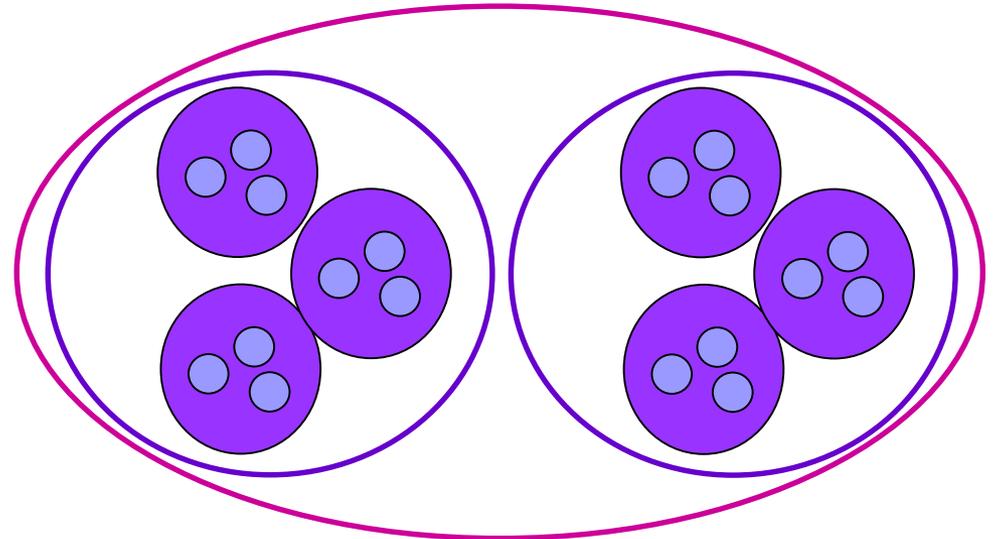
**Istruzioni macchina:** al livello più basso i programmi sono costituiti da istruzioni macchina



**Istruzioni del linguaggio:** i linguaggi di alto livello sono costituiti da istruzioni che eseguono (una o) più istruzioni macchina



**Funzioni:** Le funzioni sono costituite da gruppi di istruzioni del linguaggio



**Programmi:** i programmi sono costituiti da gruppi di funzioni

# Le funzioni – 3

- ✦ Una funzione è una macchina specializzata, che accetta dati in ingresso, li elabora in modo definito, e restituisce i risultati
- ✦ **Esempio** di funzione di basso livello per il calcolo del quadrato di un numero

```
int square(int num)
{
    int answer;
    answer=num*num;
    return answer;
}
```

La funzione *square()* accetta un numero come dato in ingresso e restituisce il quadrato del numero come risultato; può essere "richiamata" quando è necessario calcolare il quadrato di un numero intero qualunque

# Le funzioni – 4

- I nomi delle funzioni sono *alias*, possono essere cioè interpretati come abbreviazioni di sequenze di comandi
- Una funzione deve essere definita una sola volta, ma può essere invocata (richiamata) un numero di volte qualsiasi:
  - Ogni insieme di operazioni che deve essere svolto più di una volta è candidato a divenire una funzione
- Le funzioni aumentano il livello di astrazione del software poiché consentono la costruzione di operazioni complesse basate su componenti più semplici:
  - **Facilità di modifica e maggiore affidabilità del software**
  - **Migliore leggibilità**

# L'approccio top-down – 1

- Le funzioni possono essere utilizzate in modo proficuo per risolvere sottoproblemi di un problema più complesso
- Auspicabilmente, le funzioni relative ai sottoproblemi saranno sufficientemente generali (e ricorrenti) da poter essere utilizzate in altri contesti
- **Esempio:** Contare il numero di parole in un file



# L'approccio top-down – 2

2° passo

*aprire il file;*

*fino a quando ci sono parole nel file*

*leggere i caratteri finché non si incontra un carattere diverso da spazio;*

*leggere i caratteri finché non si incontra uno spazio;*

*incrementare il contatore delle parole;*

*stampare il contatore delle parole;*

*chiudere il file;*

- Il livello di dettaglio raggiunto è sufficiente, infatti esistono le opportune funzioni di libreria:

◆ *fopen()*

apre un file

◆ *fgetc()*

legge un carattere da un file

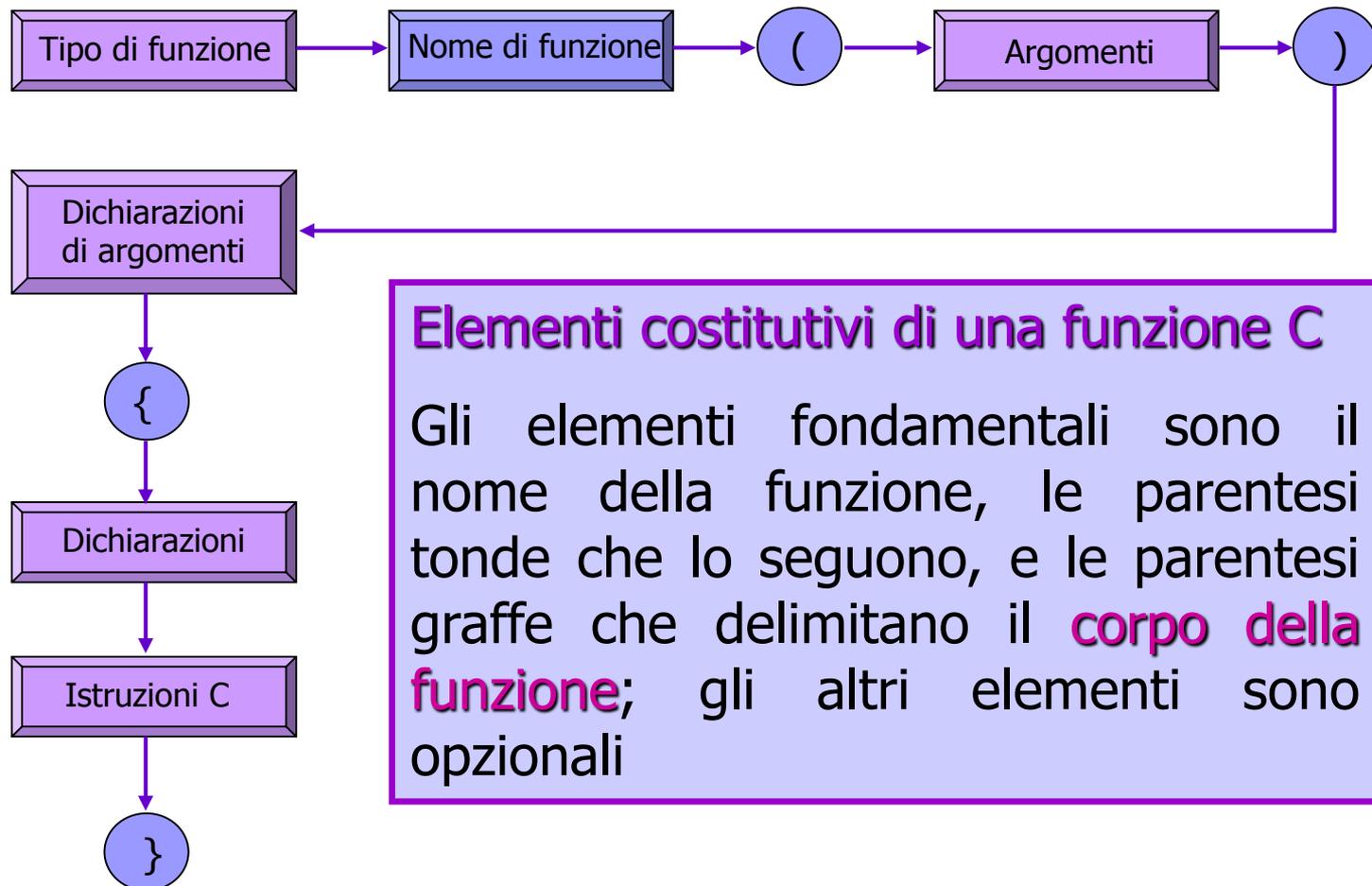
◆ *printf()*

stampa su terminale

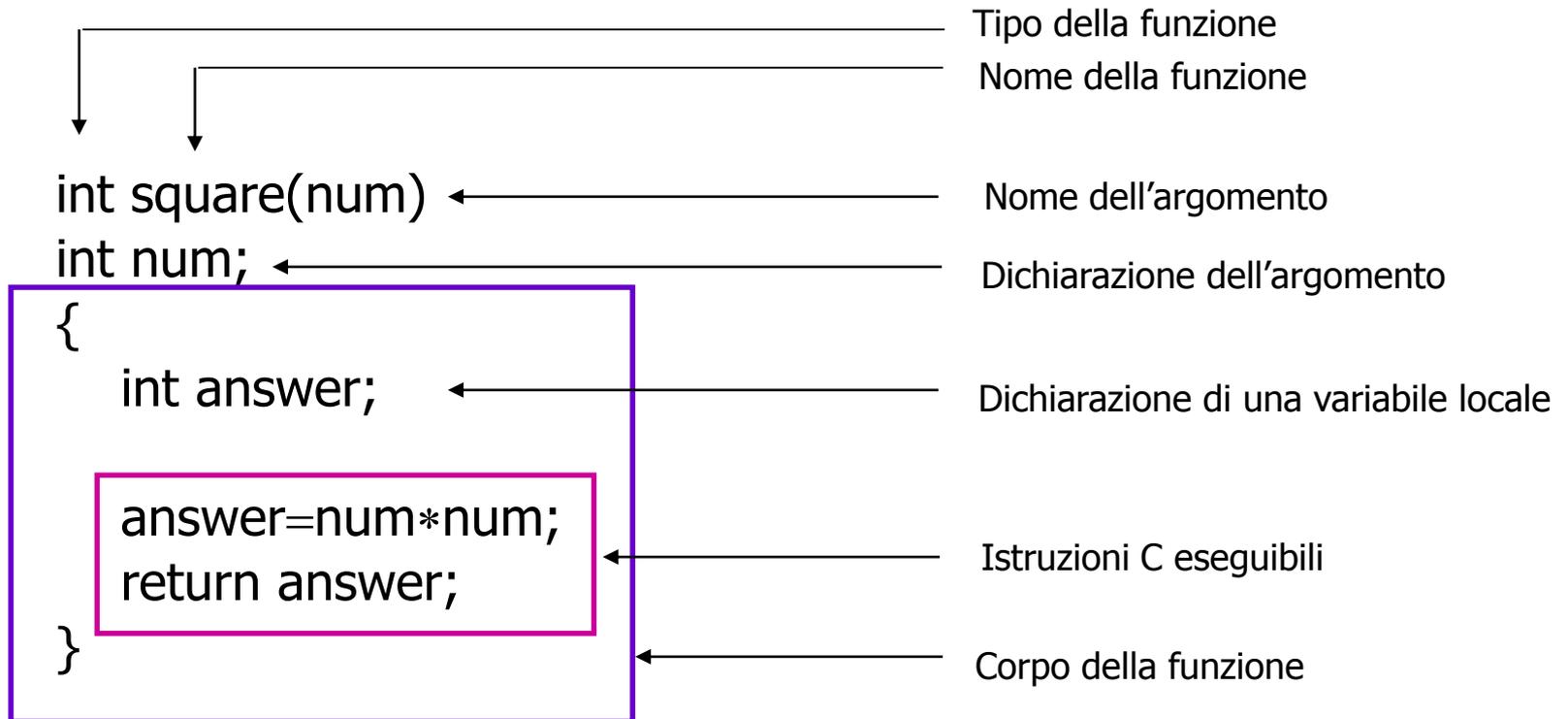
◆ *fclose()*

chiude un file

# Anatomia di una funzione C – 1



# Anatomia di una funzione C – 2



# Anatomia di una funzione C – 3

- I **parametri**, o **argomenti**, di una funzione sono dati passati dalla funzione chiamante alla funzione chiamata
  - ◆ Dal punto di vista del chiamante, gli argomenti sono **parametri attuali**
  - ◆ Dal punto di vista del chiamato, gli argomenti sono **parametri formali**
- Le funzioni possono avere un numero qualsiasi di argomenti
- Le dichiarazioni degli argomenti seguono l'intestazione della funzione
- Le dichiarazioni delle variabili locali alla funzione fanno invece parte del corpo della funzione

# Variabili e costanti – 1

- Con il linguaggio C è possibile associare un nome a quasi tutti gli oggetti: variabili, costanti, funzioni e punti particolari all'interno di un programma
- Le regole per la composizione dei nomi sono le stesse, indipendentemente dall'oggetto, e non esiste un limite imposto alla lunghezza di un nome
- I nomi possono contenere lettere, numeri ed il carattere di sottolineatura “\_” (*underscore*), ma devono iniziare per lettera o per underscore
- I nomi che iniziano con underscore sono generalmente riservati alle variabili di sistema
- Il C è **sensibile alle maiuscole** (*case sensitive*), distingue cioè fra lettere maiuscole e minuscole

# Variabili e costanti – 2

- Un nome non deve coincidere con una **parola riservata**, né con il nome di una funzione di libreria, a meno che non si desideri creare una propria versione della funzione

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

Le parole chiave del linguaggio C

# Variabili e costanti – 3

- **Esempio** – Nomi di variabile corretti:

j

j5

\_system\_name

variable\_name

NoMe\_CoN\_IeTtErE\_MiNuScOIE\_e\_MaIuScOIE

- **Esempio** – Nomi di variabile scorretti:

5j

i nomi non possono iniziare con una cifra

\$name

i nomi non possono contenere il simbolo \$

int

int è una parola riservata

bad%#\*name

i nomi non possono contenere nessun carattere speciale  
eccetto “\_”

# Variabili scalari

- Una variabile scalare:

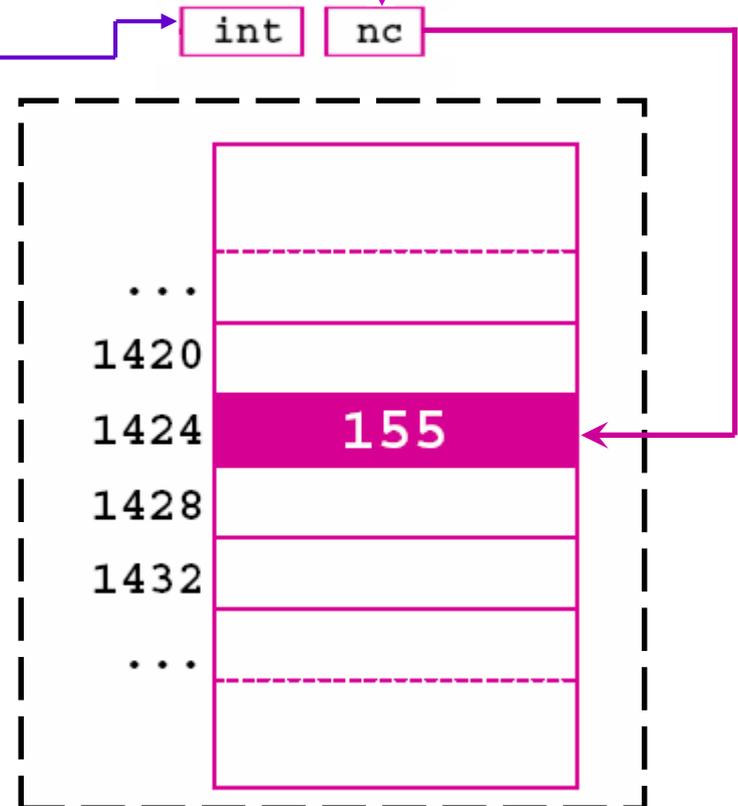
- ↳ Ha un **nome**

- ↳ Ha un **tipo**

- Numero intero
- Numero reale
- Carattere
- ...

- Corrisponde ad un'area di memoria di dimensione adatta

- Contiene un dato semplice



# Variabili complesse

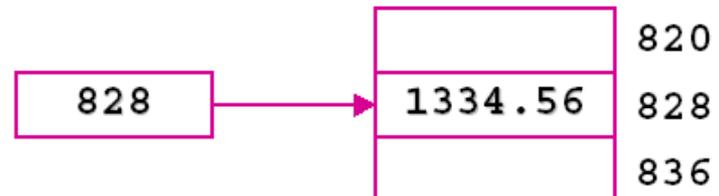
## Variabili composte

- Collezioni di dati omogenei
  - ↳ Vettori
  - ↳ Matrici
- Collezioni di dati eterogenei
  - ↳ Strutture (record)
  - ↳ Unioni
- Riferimenti ai dati
  - ↳ Puntatori

...	16	12	2000	6	...
-----	----	----	------	---	-----

16	12	0	...
12	23	...	...
...	...	...	...

16/12/2000
1334.56
"New York"



# Ancora sulle variabili...

- In C, ogni variabile è caratterizzata da:
  - ↳ **Tipo**
  - ↳ **Classe di memorizzazione**
- Assegnare un tipo ad una variabile significa assegnarle il dominio dal quale assume i valori
- La classe di memorizzazione determina la durata della vita (ciclo di vita) e l'ambito di visibilità (scope) delle variabili

# Il concetto di tipo

- I dati sono memorizzati come sequenze di bit
- Il tipo di una variabile...
  - ...determina come interpretare tale sequenza di bit
  - ...ne definisce le dimensioni
- Dal punto di vista del programmatore
  - Aiuta a strutturare il programma
  - Evita errori causati dal possibile uso inappropriato di dati eterogenei
  - Permette la definizione di strutture dati complesse

# Le espressioni

- Gli elementi costitutivi di un'espressione sono: variabili, costanti e chiamate di funzione
- Gli elementi di un'espressione sono essi stessi espressioni elementari, che possono essere combinate mediante operatori (ad es., +, -, \*, /) a formare espressioni più complesse
- Ad esempio, sono espressioni:

5	costante
j	variabile
5+j	costante più variabile
5+j*6	costante più variabile moltiplicata per una costante
f()	chiamata di funzione
f()/4	chiamata di funzione il cui risultato è diviso per una costante

# Le istruzioni di assegnamento

- La sintassi dell'istruzione di assegnamento è



- ✦ L'espressione che appare alla destra del simbolo di uguale, detta **rvalue**, è un valore
  - ✦ La parte sinistra dell'istruzione di assegnamento, detta **lvalue**, rappresenta un luogo dove memorizzare un valore
- La distinzione tra **lvalue** ed **rvalue** fa sì che l'istruzione...

**num\*num=answer;**

...non abbia senso in C, poiché l'espressione num\*num non rappresenta una locazione di memoria, ma un valore

# L'impaginazione dei file sorgente

## – 1

- Il carattere speciale *newline* sposta il cursore all'inizio della riga successiva
- Il carattere newline si ottiene con la pressione del tasto **Return** (o **Enter**, o **Invio**) sulla tastiera
- In C, i newline nel codice sorgente sono trattati come spazi (eccettuato quando compaiono in una costante di caratteri o in una stringa)
- La funzione *square()* poteva essere scritta come...

```
int square(num) int num; { int answer;  
    answer=num*num; return answer; }
```

- Sebbene questo formato sia equivalente per il calcolatore, è un esempio di codice scarsamente leggibile e quindi di stile di programmazione scadente

# L'impaginazione dei file sorgente

## – 2

- Il compilatore non considera gli spazi ed i newline che vengono inseriti tra i componenti del programma
- È buona norma utilizzare l'**indentazione** nella scrittura dei programmi, cioè utilizzare gli spazi ad inizio riga, per scrivere con ugual margine sinistro tutte le istruzioni che costituiscono un blocco logico (ad es., tutte le istruzioni che vengono eseguite all'interno di un ciclo **for**)
- L'indentazione è trasparente per il compilatore, ma migliora la leggibilità del codice

# I commenti – 1

- Un commento è un testo che viene incluso in un file sorgente per spiegare il significato del codice
- I commenti sono ignorati dal compilatore
- I commenti sono un elemento fondamentale nello sviluppo del software: il linguaggio C consente di inserire commenti racchiudendo il testo tra i simboli `/*` e `*/` (oppure `//` per commentare un'intera riga)

```
/* Questa funzione restituisce
 * il quadrato del suo argomento
 */
int square(num)
int num;
{
    int answer;

    answer=num*num; /* elevamento al quadrato */
    return answer;
}
```

# I commenti – 2

- Non sono ammessi commenti innestati
- Un commento può occupare più linee
- Affinché i commenti non interrompano il flusso di un programma...
  - ...occorre dedicare ai commenti intere linee di codice
  - ...o collocarli sulla destra del codice, quando condensabili in un'unica riga
- Cosa deve essere commentato? Tutto ciò che non è ovvio:
  - Espressioni complesse, strutture dati e scopo delle funzioni
  - Eventuali modifiche apportate al programma, per poterne tenere traccia
- In particolare, ogni funzione dovrebbe avere un commento di intestazione, che descrive "cosa fa" la funzione ed il significato dei suoi parametri

# I commenti – 3

- Tuttavia, commenti con scarso contenuto informativo possono rendere un programma difficile da leggere
- Un esempio di stile di documentazione scadente...

```
j=j+1;    /* incrementa j */
```

- Inoltre, commenti molto lunghi non compensano codice illeggibile o stilisticamente imperfetto

# La funzione `main()` – 1

- Dopo avere scritto e compilato la funzione `square()`, non si è ancora in grado di eseguirla
- Ogni programma scritto in C deve contenere una funzione speciale, chiamata `main()`, che indica il punto da cui inizia l'esecuzione del programma
- Le regole di scrittura della funzione `main()` coincidono con quelle delle altre funzioni; tuttavia, non viene (normalmente) specificato il tipo della funzione e non vengono (di solito) dichiarati argomenti
- La funzione `main()` può richiamare (e generalmente richiama) altre funzioni

# La funzione main ( ) – 2

✦ **Esempio:** funzione *main()* che invoca la funzione *square()*

```
#include <stdlib.h>
#include <stdio.h>

main()
{
    extern int square();
    int solution;
    int input_val;

    printf("Introdurre un valore intero:");
    scanf("%d",&input_val);
    solution=square(input_val);
    printf("Il quadrato di %d è %d\n",input_val,solution);
    exit(0);
}
```

## Note:

- La funzione *square()* viene dichiarata **extern** perché il codice relativo può essere esterno, cioè scritto in un file sorgente diverso dal file corrente
- La funzione *exit()* appartiene alla libreria di run-time (si noti l'include di *stdlib.h*) e provoca la terminazione dell'esecuzione di un programma, ritornando il controllo al sistema operativo

# La funzione `printf( )` – 1

- La funzione *printf()* può avere un numero variabile di argomenti
- Il primo argomento è un parametro speciale, detto **stringa di formato**, che specifica il numero di argomenti che contengono i dati da stampare e le modalità di formattazione dei dati
- La stringa di formato è racchiusa fra doppi apici e può contenere testo e **specificatori di formato** — sequenze speciali di caratteri che iniziano con il simbolo di percentuale (%) ed indicano le modalità di scrittura di un singolo dato

- **Esempio:** nell'istruzione

```
printf("Il valore di num è %d",num);
```

- ↘ "Il valore di num è %d" è la stringa di formato
- ↘ %d è lo specificatore di formato per gli interi decimali
- ↘ num è la variabile intera decimale da stampare

# La funzione printf( ) – 2

- Esistono altri specificatori per altri tipi di dati:

%c          dato di tipo carattere

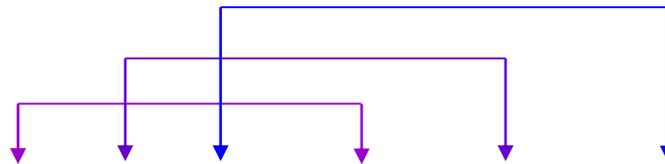
%f          dato di tipo floating–point

%s          array di caratteri terminato da *null* (stringa)

%o          intero ottale

%x          intero esadecimale

■ La stringa di formato può contenere un numero qualunque di specificatori di formato, ma il loro numero deve coincidere con il numero dei dati da stampare, passati come argomenti



```
printf("Stampa tre valori: %d %d %d",num1,num2,num3);
```

# La funzione printf ( ) – 3

- I dati da stampare possono essere espressioni  

```
printf("Il quadrato di %d è %d\n",num,num*num);
```
- Il simbolo speciale `\n` è una **sequenza di escape**
- Quando le sequenze di escape sono inviate ad un dispositivo di uscita sono interpretate come segnali che controllano il formato della visualizzazione
- `\n` forza il sistema ad effettuare un ritorno a capo (newline)

# La funzione `scanf()`

- La funzione `scanf()` legge dati introdotti da tastiera
- `scanf()` può ricevere un numero qualunque di parametri preceduti da una stringa di formato
- I parametri di `scanf()` devono essere lvalue, e devono pertanto essere preceduti dall'**operatore indirizzo** `&`
- **Esempio:**

```
scanf("%d",&num);
```

richiede al sistema di leggere un intero da terminale e di memorizzare il valore nella variabile `num`

# Il preprocessore

- Il **preprocessore** C è un programma che viene eseguito prima del compilatore (non è necessario "lanciarlo" esplicitamente)
- Attraverso il preprocessore si esprimono direttive al compilatore
- Il preprocessore ha la sua grammatica e la sua sintassi che sono scorrelate da quelle del C
- Ogni direttiva inizia con il simbolo **#**, che deve essere il primo carattere diverso dallo spazio sulla linea
- Le direttive del preprocessore terminano con un newline (non con ";" )

# Direttive del preprocessore

- Principali compiti richiesti al preprocessore:
  - ↳ Inclusione del codice sorgente scritto su altro file
  - ↳ Definizione delle costanti simboliche
  - ↳ Compilazione condizionale del codice

# La direttiva `#include` – 1

- La direttiva `#include` fa sì che il compilatore legga il testo sorgente da un file diverso da quello che sta compilando
- `#include` lascia inalterato il file da cui vengono prelevati i contenuti
  - Utile quando le stesse informazioni devono essere condivise da più file sorgente: si raccolgono le informazioni comuni in un unico file e lo si include ovunque sia necessario
  - Si riduce la quantità di testo da digitare e si facilita la manutenzione: i cambiamenti al codice condiviso hanno effetto immediato su tutti i programmi che lo includono

# La direttiva `#include` – 2

- La direttiva `#include` può assumere due formati

`#include <nome_file.h>`

`#include "nome_file.h"`

- ...nel primo caso, il preprocessore cerca il file in una directory speciale, definita dall'implementazione del compilatore, dove sono contenuti i file che vengono normalmente inclusi da tutti i programmi utente (sintassi usata per includere file di intestazione, *header file*, della libreria standard)
- ...nel secondo caso, il file viene prima cercato nella directory del file sorgente e, quando non reperito, seguendo il percorso classico

# La direttiva #define

- La direttiva **#define** consente di associare un nome ad una costante

- **Esempio:**

```
#define NIENTE 0
```

associa il nome "NIENTE" alla costante 0

- Per evitare confusione fra nomi di costanti e nomi di variabili, è pratica comune usare solo lettere maiuscole per le costanti e solo minuscole per le variabili
- L'associazione di nomi alle costanti permette...
  - ↳ ...di utilizzare un nome descrittivo per oggetti altrimenti non autoreferenziali
  - ↳ ...di semplificare la modifica del software: cambiare il valore ad una costante equivale a cambiarne la sola definizione e non tutte le occorrenze

# La compilazione condizionale

- Consente al programmatore di controllare la compilazione del codice del programma
- **Esempio:**

```
#define DEBUG 1
#ifdef DEBUG
    printf("Variabile x = %d\n",x);
#endif
```

l'istruzione *printf()* viene compilata (ed eseguita) solo nel caso in cui la costante DEBUG sia definita

# Esempio 1

```
#include<stdio.h>
/* Questo è il nostro primo programma C */
/* I commenti possono occupare...
...più linee! */

int main()
{
    printf("Salve, ");
    printf("mondo");
    printf("\n");
    return 0;
}
```

Commento: testo esplicativo aggiunto solo per chiarezza; non ha alcuna funzione ed è ignorato dal compilatore

Commento su più linee

Ogni funzione ritorna un valore: la parola chiave **int** indica che si tratta di un valore intero

La funzione *printf()* non inserisce automaticamente caratteri di ritorno a capo

Specifica il valore che deve essere ritornato; 0 attesta che il programma è terminato correttamente

# Esempio 2: somma di due interi

```
#include<stdio.h>
int main()
{
    int a, b, c;
    printf("Inserire primo intero: \n");
    scanf("%d", &a);
    printf("Inserire secondo intero: \n");
    scanf("%d", &b);
    c = a + b;
    printf("Risultato: %d\n", c);
    return 0;
}
```

Dichiara che nella funzione *main()* verranno utilizzate tre variabili intere con nomi **a**, **b** e **c**; inizialmente il valore non è definito

Stampa un messaggio

Legge da tastiera (standard input)

Specifica che i caratteri letti da tastiera devono essere interpretati come le cifre di un numero intero

Memorizza nella variabile **a** ciò che è immesso da tastiera

Esegue la somma

Stampa il risultato

Indica che la variabile **c** deve essere interpretata come numero intero

# Esempio 3: differenza di due interi

```
#include<stdio.h>
int main()
{
    int a, b, c; //Dichiarazione delle variabili
    //Inserimento dati
    printf("Inserire primo intero: \n");
    scanf("%d", &a);
    printf("Inserire secondo intero: \n");
    scanf("%d", &b);
    if (a>b){ //Determinazione del numero maggiore
        c=a-b;
        //Stampa risultato
        printf("Risultato di %d - %d: %d\n", a, b, c);
    }
    else{
        c=b-a;
        //Stampa risultato
        printf("Risultato di %d - %d: %d\n", b, a, c);
    }
    return 0;
}
```

Costrutto di **selezione condizionale**

# Esempio 4: somma di una sequenza di numeri

```
#include<stdio.h>
int main()
{
    int somma, numero; //Dichiarazione variabili
    somma=0;           //Inizializzazione variabile
    printf("Inserire un numero (l'inserimento termina con 0): \n");
    scanf("%d", &numero);
    while(numero != 0)
    {
        somma = somma+numero;
        printf("Inserire un numero (l'inserimento termina con 0): \n");
        scanf("%d", &numero);
    }
    printf("Somma della sequenza inserita: %d\n", somma);
    return 0;
}
```

**Ciclo while** con controllo in testa ed iterazione per vero

# Considerazioni finali

- Gli esempi hanno permesso di introdurre alcuni dei concetti di base del linguaggio C:
  - Ogni programma C contiene la funzione *main()*
  - I blocchi di codice sono delimitati da parentesi graffe
  - Le istruzioni sono terminate dal punto e virgola
  - Le variabili
    - Devono essere dichiarate
    - Hanno un tipo in base ai dati che dovranno contenere
- Inoltre, il linguaggio C permette di...
  - Leggere e scrivere dati
  - Svolgere operazioni aritmetiche
  - Verificare condizioni
  - Eseguire cicli