Informatica Generale

Linguaggi di Programmazione

Linguaggi di Programmazione

- Linguaggi di Programmazione
  - Sintassi e semantica
- Compilatori, interpreti e il linker
- Introduzione al C
- La funzione main del 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?

Lessico, sintassi e semantica

- **Lessico**: l'insieme di regole formali per la scrittura di parole in un linguaggio
- **Sintassi**: l'insieme di regole formali per la scrittura di frasi in un linguaggio, che stabiliscono cioè la grammatica del linguaggio stesso
- **Semantica**: l'insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio
- **Nota**: una frase può essere sintatticamente corretta e tuttavia non avere significato!

Esempio: la sintassi di un numero naturale

<cifra-non-nulla> := 1|2|3|4|5|6|7|8|9

<cifra> := 0 | <cifra-non-nulla>

<naturale> := 0 | <cifra-non-nulla>{<cifra>}

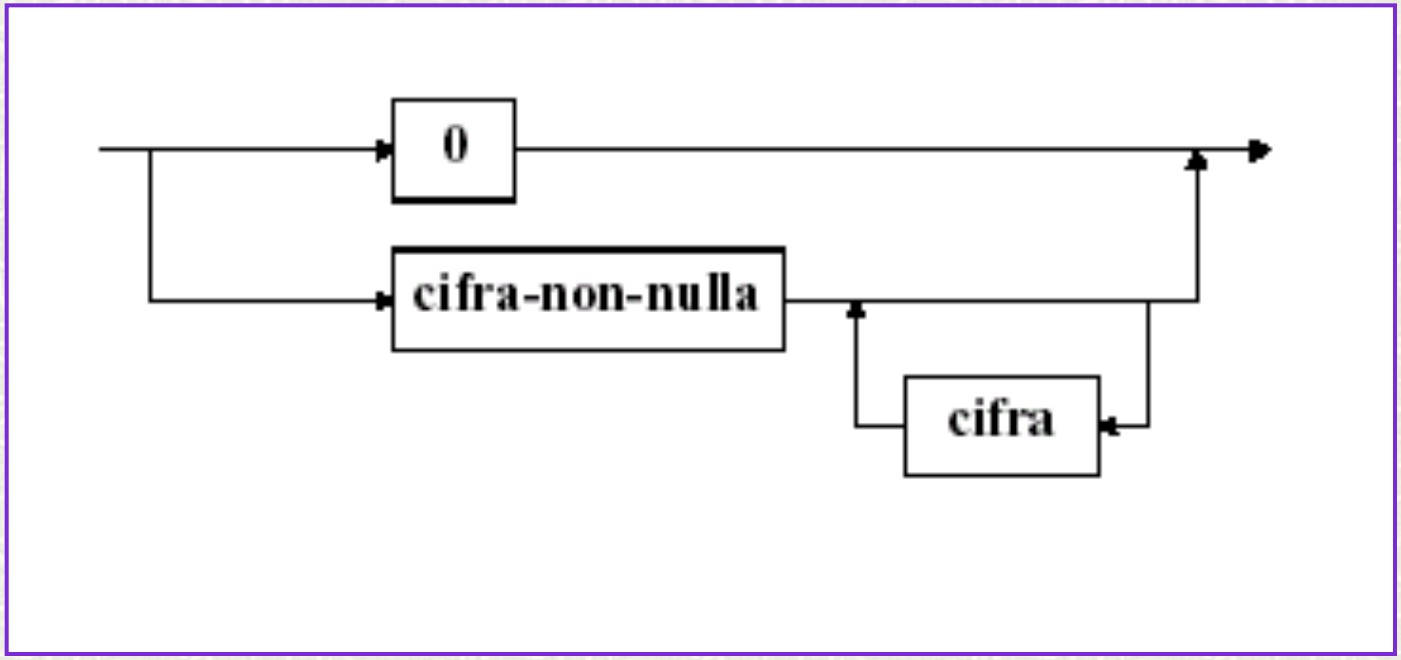
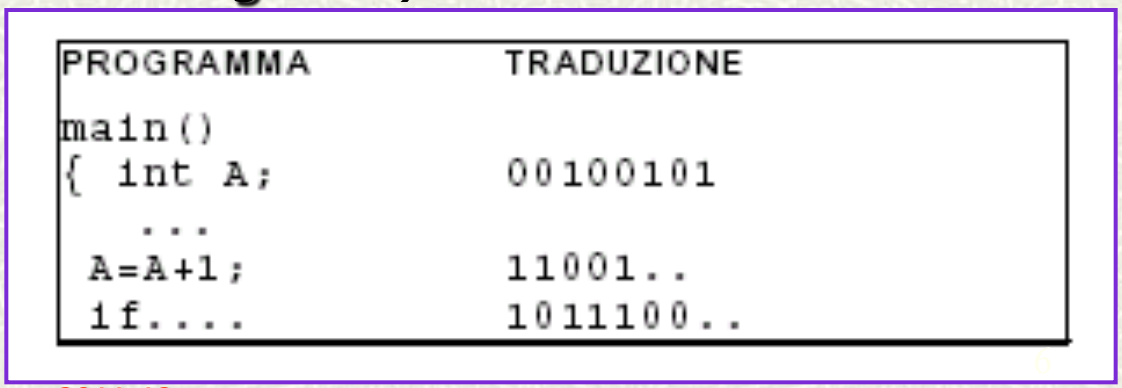


Diagramma sintattico

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**)



Compilatori ed interpreti – 2

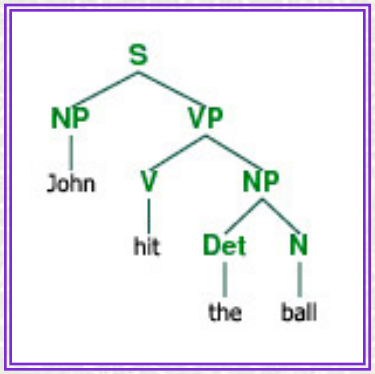
- Compilatore: opera la traduzione di un programma sorgente (scritto in linguaggio di alto livello) in un programma oggetto direttamente eseguibile dal calcolatore
  - PRIMA si traduce tutto il programma
  - POI si esegue la versione tradotta
- Interprete: traduce ed esegue il programma sorgente, istruzione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

Compilatori ed interpreti – 3

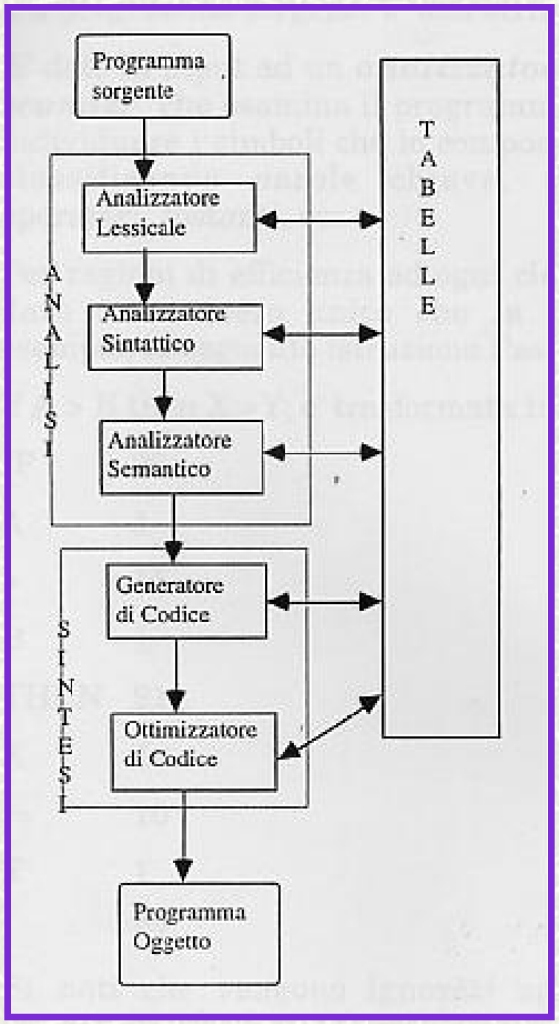
- Esempio di compilatore
  - Dobbiamo sottoporre un curriculum, in inglese, ad una azienda, ma non conosciamo l'inglese
  - Abbiamo bisogno di un traduttore che traduca quanto scritto da noi dall'italiano all'inglese
    - contattiamo il traduttore
    - il traduttore riceve il testo da tradurre
   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

Compilatori ed interpreti – 4

- Compilatore
  - Analisi lessicale ⇨ token
  - Analisi sintattica ⇨ albero sintattico



  - Analisi semantica ⇨ tabella dei simboli



Compilatori ed interpreti – 5

- 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 – 6

- Riassumendo…
  - I compilatori traducono un intero programma dal linguaggio 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 – 7

- 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

Storia del linguaggio C

Storia del linguaggio C - 1

- Il linguaggio C venne definito alla fine degli anni '60 da Dennis M. Ritchie, degli AT&T Bell Labs, come ***linguaggio di programmazione di sistema***
- Il linguaggio C doveva essere…
  - …un linguaggio di livello sufficientemente alto per garantire ai programmi leggibilità e manutenibilità
  - …un linguaggio sufficientemente semplice da stabilire una corrispondenza immediata con la macchina sottostante
  - …indipendente dall'hardware e quindi portabile
- Il linguaggio C si dimostrò così flessibile, ed il codice macchina prodotto così efficiente che, nel 1973, Ritchie e Ken Thompson riscrissero **UNIX** in C

Storia del linguaggio C - 2

- Oggi molti sistemi operativi sono sviluppati in C o C++
- I vantaggi fondamentali della scrittura di sistemi operativi in linguaggio di alto livello sono la velocità di sviluppo e la manutenibilità
- Come effetto collaterale si ottiene un sistema operativo che può essere trasferito su architetture diverse, tramite ricompilazione su macchina target: *porting*
- Nel 1977, Ritchie e Brian Kernighan pubblicarono "The C Programming Language", che formalizza lo ***standard K&R***
- Inizialmente il linguaggio C veniva usato soprattutto sui sistemi UNIX (PCC - Portable C Compiler) ma, con la diffusione dei PC, compilatori C furono prodotti per nuove architetture e nuovi sistemi operativi

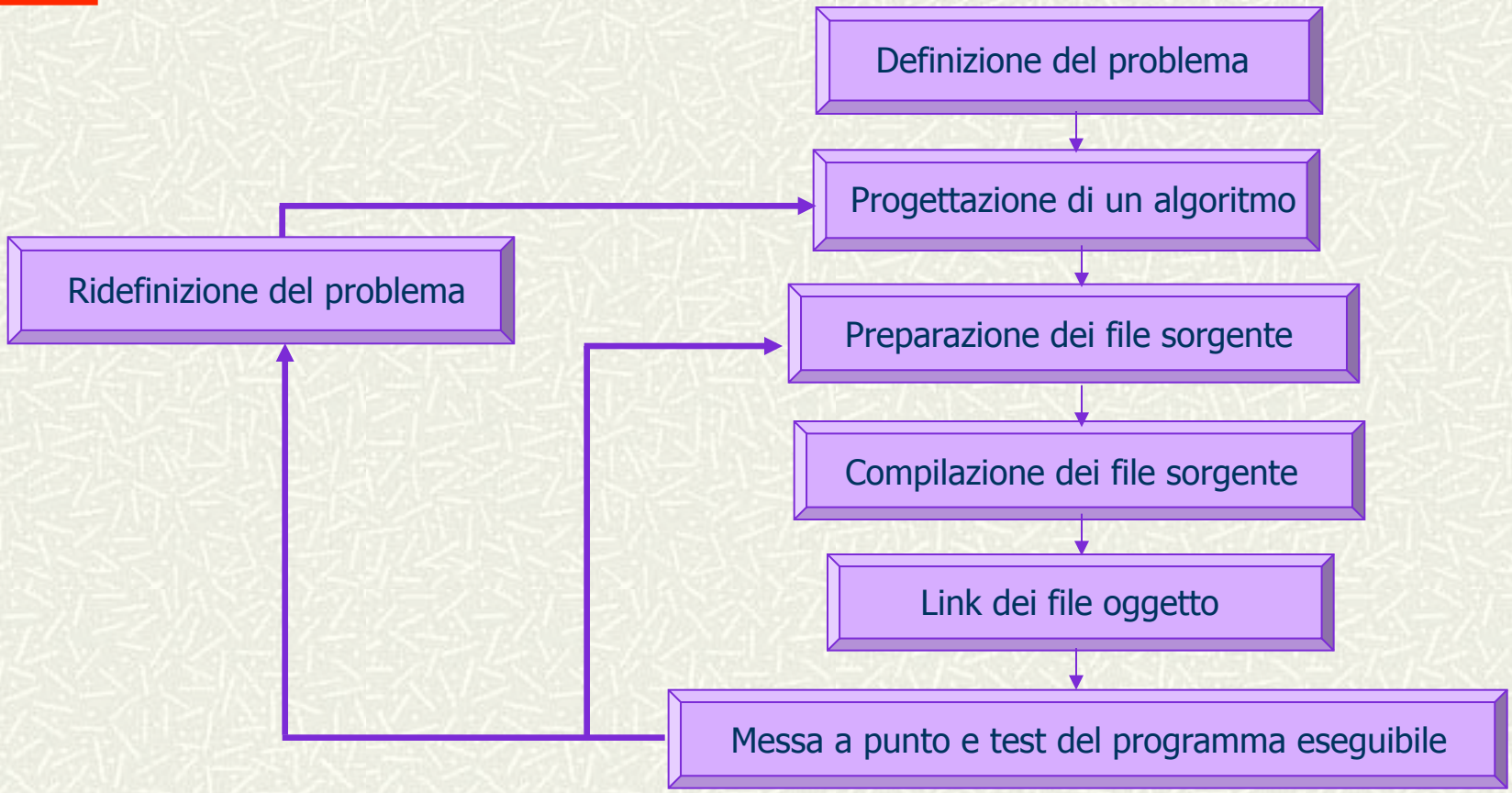
ANSI C

- Nel 1983, l'***American National Standards Institute*** (ANSI), costituì la commissione X3J11, che doveva formulare uno standard per il C, che includesse le nuove caratteristiche che il linguaggio aveva progressivamente maturato, mantenendone la portabilità
- La versione finale dello standard C fu approvata dall'ANSI nel 1989
- Lo Standard ANSI C è descritto nel documento "***American National Standard for Information Systems - Programming Language C***"
- Lo standard è stato rivisto ed aggiornato nel 1999

Caratteristiche del linguaggio C

- Linguaggio di medio/alto livello; basso livello di controllo degli errori nella fase di compilazione
- Variabili tipizzate, con notevoli possibilità di conversione mediante il type casting, che permette di forzare una variabile a cambiare tipo
- Abbina ad un livello medio/alto di astrazione, un buon controllo delle operazioni a basso livello

Lo sviluppo dei programmi - 1

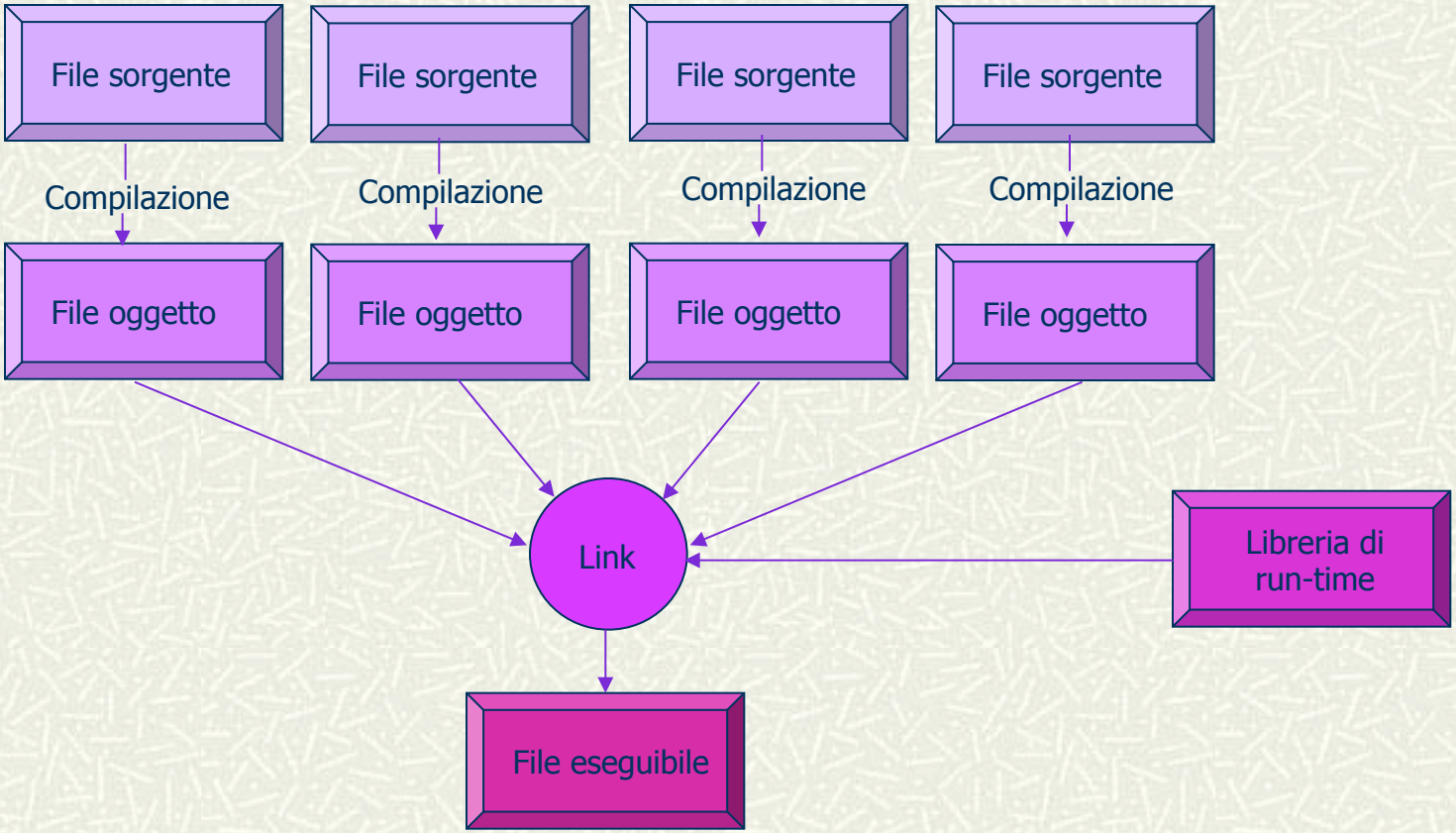


Fasi dello sviluppo di un programma

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 all' **editor** (per la scrittura dei file sorgente), il **compilatore**, il ***linker*** e il ***loader***, è utile 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)

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

Primo Programma in C: la funzione main()

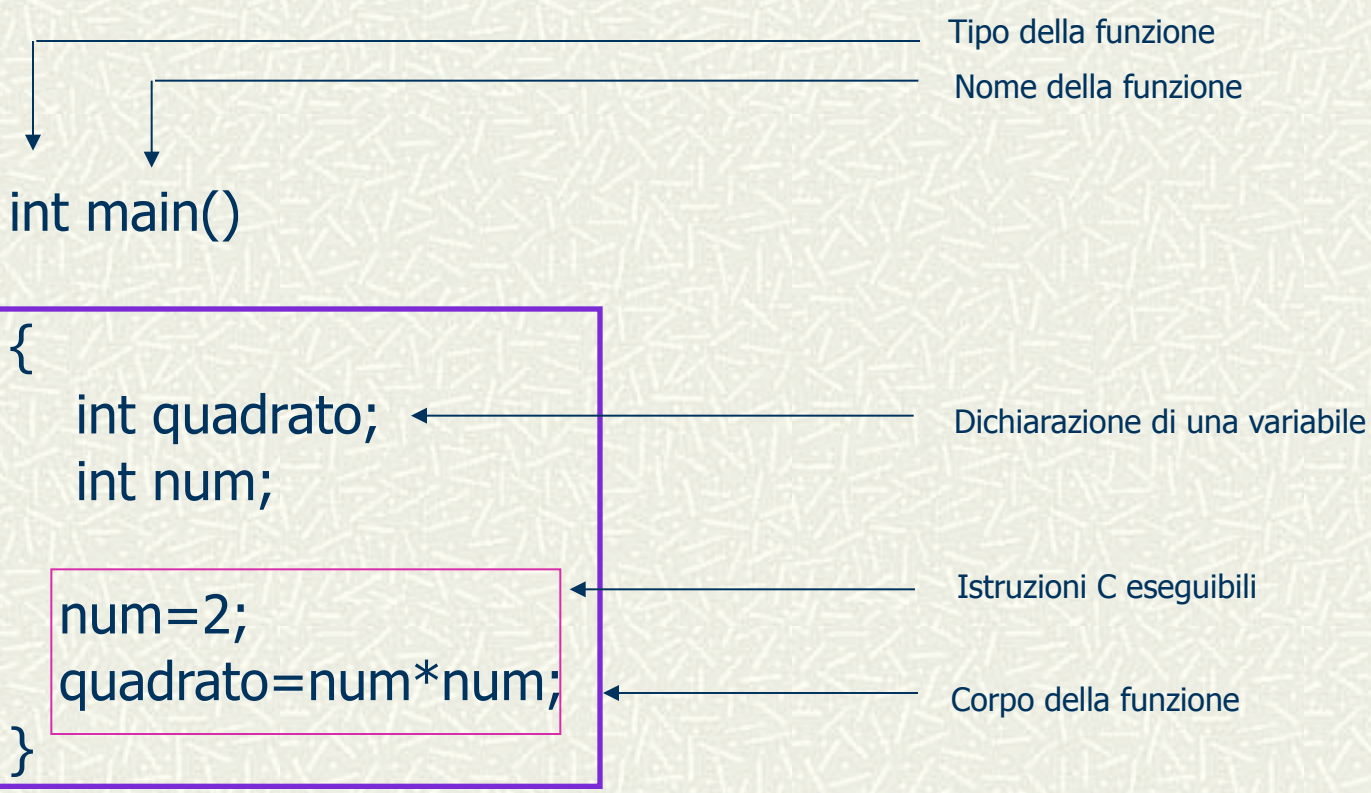
```
#include<stdio.h>

main()

{

    printf("Salve, mondo\n");

}
```

Include la libreria standard di I/O

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

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

ESEMPIO 2


Tipo della funzione
Nome della funzione
int main()
{
int quadrato;
int num;
num=2;
quadrato=num*num;
}
Dichiarazione di una variabile
Istruzioni C eseguibili
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:

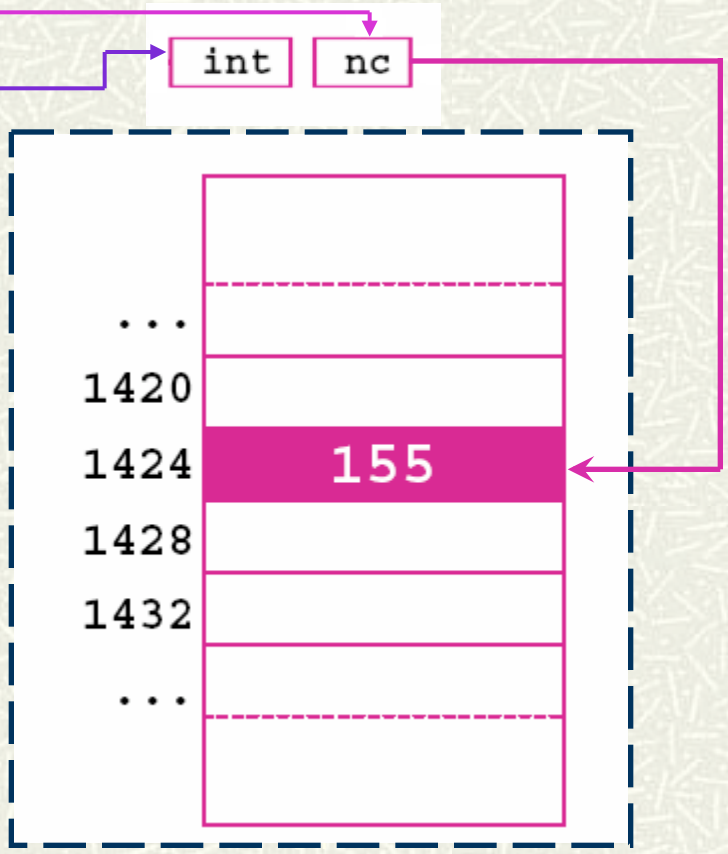
  j
  j5
  _system_name
  variable_name
  NoMe_CoN_lEtTeRe_MiNuScOlE_e_MaIuScOlE

- 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 "_"

Allocazione delle variabili

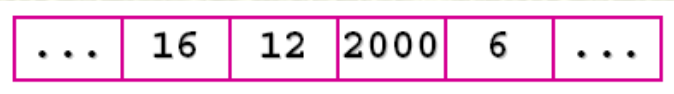
- Una variabile scalare:
  - Ha un nome
  - Ha un tipo
    - Numero intero
    - Numero reale
    - Carattere
    - ...
- Corrisponde ad un'area di memoria di dimensione adatta
- Contiene un dato semplice

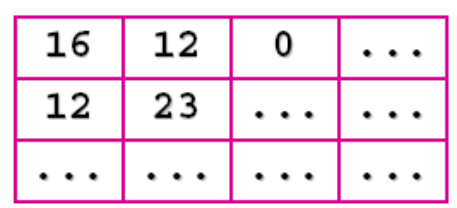


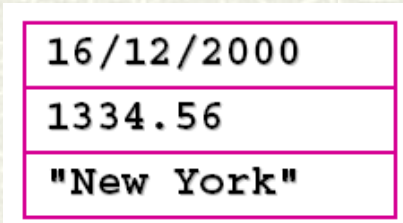
Variabili complesse

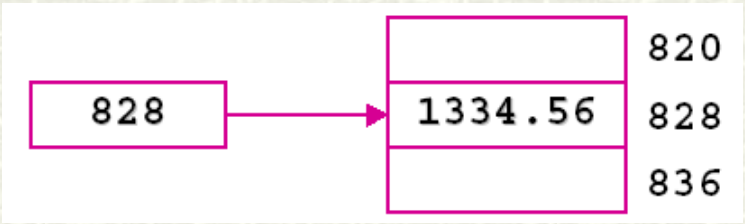
Variabili composte

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









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

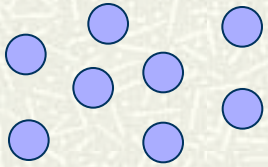
L'impaginazione dei file sorgente - 2

- 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

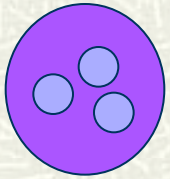
Le funzioni - 1

- 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

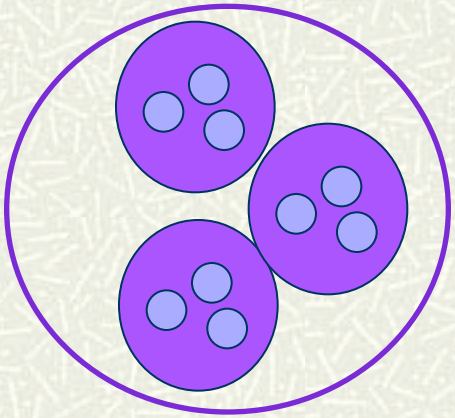
Le funzioni - 2



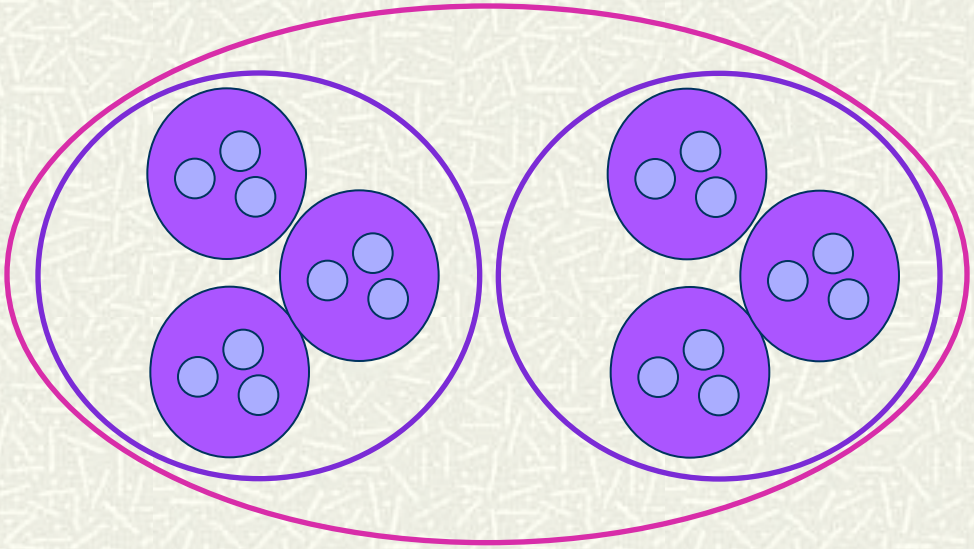
Istruzioni macchina: al livello più basso i programmi sono costituiti da istruzioni macchina



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

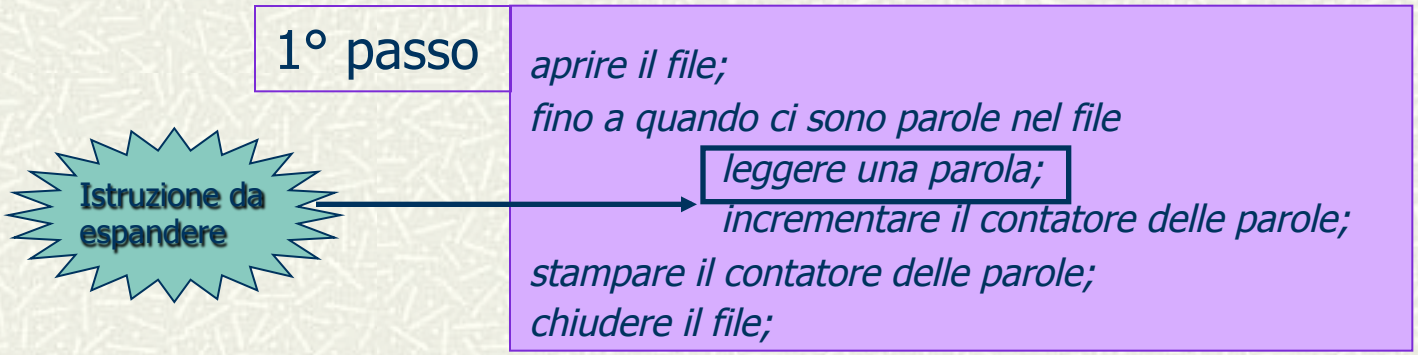
La funzione *square* ( ) accetta un numero come dato in ingresso e restituisce il quadrato del numero come risultato; può essere "richiamata" quando è necessario calcolare il quadrato di un numero qualunque

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



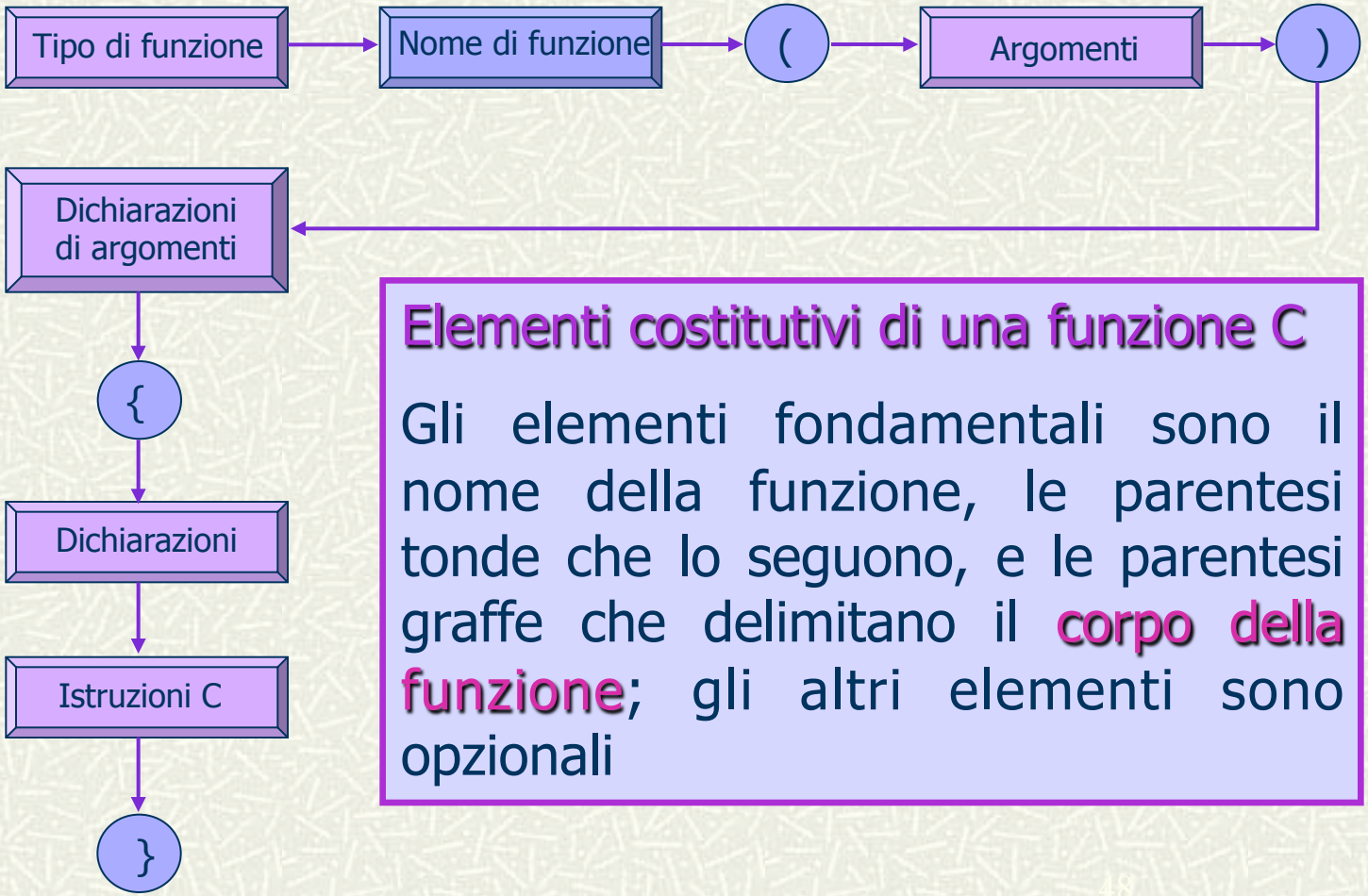
L'approccio top-down - 2

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

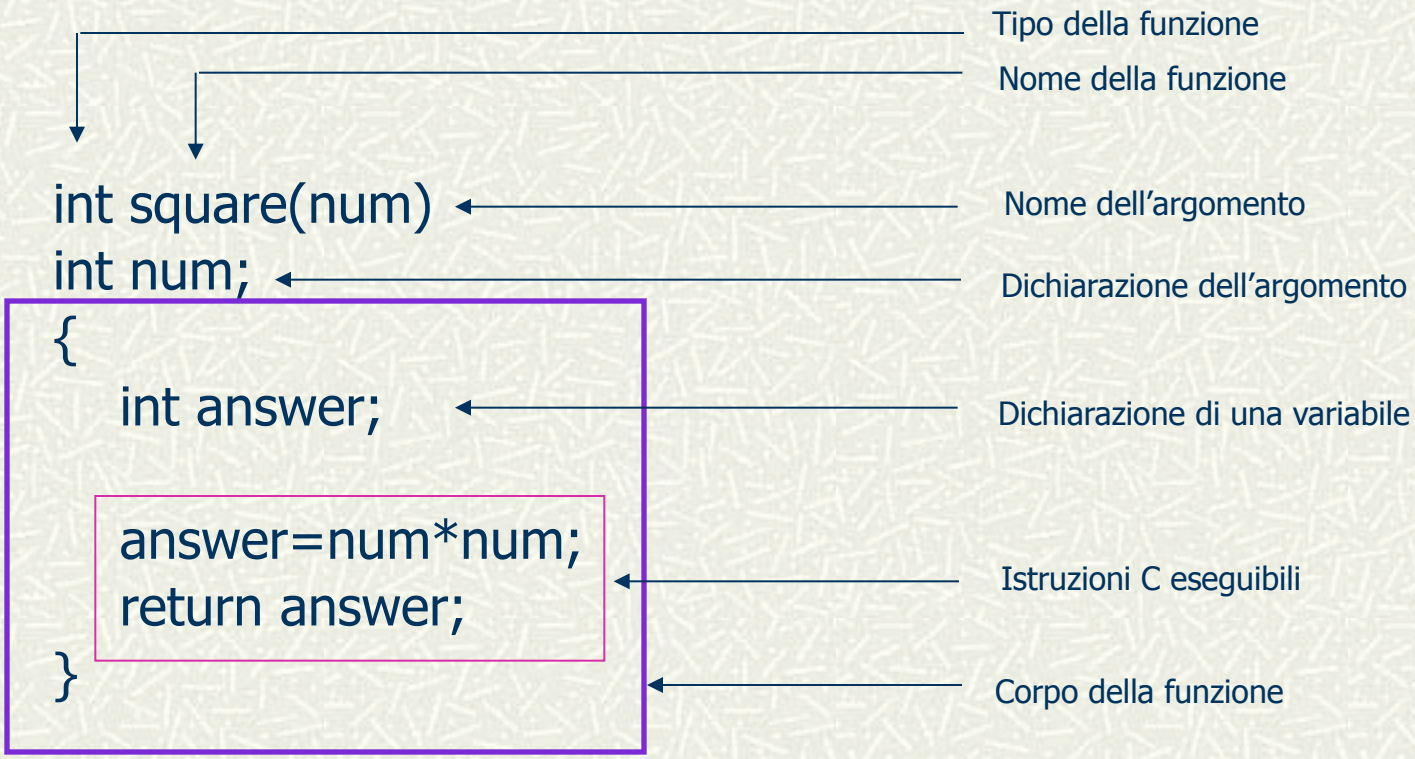
Anatomia di una funzione C - 1



Elementi costitutivi di una funzione C

Gli elementi fondamentali sono il nome della funzione, le parentesi tonde che lo seguono, e le parentesi graffe che delimitano il corpo della funzione; gli altri elementi sono opzionali

Anatomia di una funzione C - 2



La funzione *main()* - 1

- La funzione ***square()***, deve essere invocata in qualche punto del programma principale.
- 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);

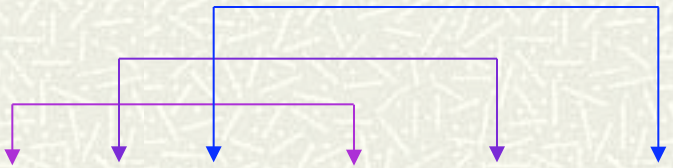
  - "Il valore di num è %d" è la stringa di formato
  - %d è lo specificatore di formato per gli interi decimali
  - num è la variabile intera decimale da stampare

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

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

Caratteri, interi e float

- Abbiamo visto il tipo **int**
  - Indica variabili di tipo intero
- Altri tipi importanti sono:
  - **float** → numeri in virgola mobile, e.g., **0.35**
  - **char** → caratteri, e.g., **'A', 'Z', '-', '£'**
- Hanno limiti superiori quini si usano
  - long int
  - Double float

→ La prossima lezione

→ int a,b; float c; char A,Z;

Caratteri e interi - 2

- **Le costanti di tipo carattere sono racchiuse tra apici singoli**
- Esempio: Leggere un carattere da terminale e visualizzarne il codice numerico

```
/* Stampa del codice numerico di un carattere */
#include<stdio.h>

main()
{
    char ch;

    printf("Digitare un carattere: ");
    scanf("%c", &ch);
    printf("Il codice numerico corrispondente è %d\n", ch);
    exit(0);
}
```

Tabella ASCII

Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr
32	20	**SPACE**	48	30	**0**	64	40	@	80	50	**P**	96	60	`	112	70	**p**
33	21	**!**	49	31	**1**	65	41	**A**	81	51	**Q**	97	61	**a**	113	71	**q**
34	22	”	50	32	**2**	66	42	**B**	82	52	**R**	98	62	**b**	114	72	**r**
35	23	**#**	51	33	**3**	67	43	**C**	83	53	**S**	99	63	**c**	115	73	**s**
36	24	**$**	52	34	**4**	68	44	**D**	84	54	**T**	100	64	**d**	116	74	**t**
37	25	**%**	53	35	**5**	69	45	**E**	85	55	**U**	101	65	**e**	117	75	**u**
38	26	**&**	54	36	**6**	70	46	**F**	86	56	**V**	102	66	**f**	118	76	**v**
39	27	’	55	37	**7**	71	47	**G**	87	57	**W**	103	67	**g**	119	77	**w**
40	28	**(**	56	38	**8**	72	48	**H**	88	58	**X**	104	68	**h**	120	78	**x**
41	29	**)**	57	39	**9**	73	49	**I**	89	59	**Y**	105	69	**i**	121	79	**y**
42	2A	*	58	3A	**:**	74	4A	**J**	90	5A	**Z**	106	6A	**j**	122	7A	**z**
43	2B	**+**	59	3B	**;**	75	4B	**K**	91	5B	**[**	107	6B	**k**	123	7B	**{**
44	2C	**,**	60	3C	**<**	76	4C	**L**	92	5C	\	108	6C	**l**	124	7C	**|**
45	2D	**-**	61	3D	**=**	77	4D	**M**	93	5D	**]**	109	6D	**m**	125	7D	**}**
46	2E	**.**	62	3E	**>**	78	4E	**N**	94	5E	**^**	110	6E	**n**	126	7E	**~**
47	2F	**/**	63	3F	**?**	79	4F	**O**	95	5F	_	111	6F	**o**	127	7F	**DEL**

Nota: il valore numerico di una cifra può essere calcolato come differenza del suo codice ASCII rispetto al codice ASCII della cifra 0 (es. ‘5’-‘0’ = 53-48 = 5)

Caratteri e interi - 3

- Dato che in C i caratteri sono trattati come interi, su di essi è possibile effettuare operazioni aritmetiche

```
int j;
j = 'A'+'B';
```

  j conterrà il valore 131, somma dei codici ASCII 65 e 66

- Esempio: Scrivere una funzione che converte un carattere da maiuscolo a minuscolo

Funziona per la codifica ASCII

```
char to_lower(ch)
char ch;
{
    return ch+32;
}
```

- In C, esistono le routine di libreria ***toupper*** e ***tolower*** in grado di convertire anche nel caso di codifiche diverse dall'ASCII

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;  /* non si controlla l'overflow */
   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

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

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