Fondamenti di Informatica

Ingegneria Clinica

Lezione 03/12/2010

Prof. Raffaele Nicolussi

FUB - Fondazione Ugo Bordoni

Via del Policlinico, 147 - 00161 Roma

Docente	**Raffaele Nicolussi**	***rnicolussi@fub.it*** ***0654803323***
Lezioni **Aula 54 (ex aula 4)** **Via del Castro Laurenziano, 7**	**Lunedì, Giovedì, Venerdì**	**12:00 – 13:30**
Esercitazioni **Aula 15** **Via Tiburtina, 205**	**Lunedì**	**14:00 – 17:30**
Ricevimento:	**Per appuntamento**	**in FUB, per email, per telefono**
Sito web:	**http://w3.uniroma1.it/IngClinFondinf**	

Esempio potenza

Leggere un reale x ed un intero positivo n e calcolare la potenza x^n

```
     X: FLOAT ;
     N: INT ;
   Ris: FLOAT ;
   Uno : INT 1;
 Unofl: FLOAT 1.0;
  READ STINP X;
  READ STINP N;
  LOAD R0 Uno;
   SUB R0 R0;
  LOAD R1 Uno;
   LOAD R2 X;
   LOAD R3 N;
 LOAD R4 Unofl;
```

Modo alternativo per assegnare **R0 = 0**

R0 = 0 intero

R1 = 1 intero

R2 = X reale

R3 = N intero

R4 = 1 reale

R0 = 0 intero

R1 = 1 intero

R2 = X reale

R3 = N intero

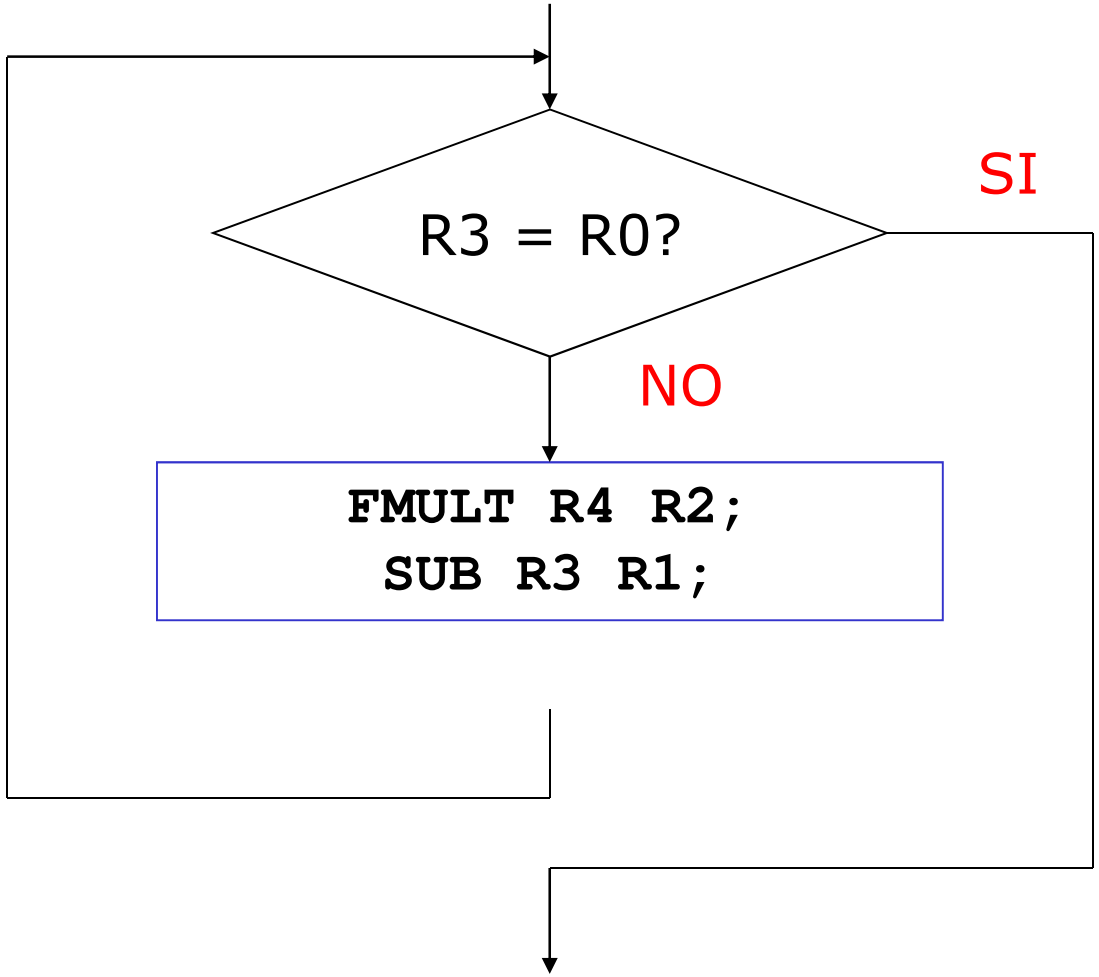
R4 = 1 reale


```
Ciclo: COMP R3 R0;
       BREQ Esci;
       FMULT R4 R2;
       SUB R3 R1;
       BRANCH Ciclo;
Esci:  STORE R4 Ris;
       WRITE STOUT Ris;
       STOP;
```

$R4 = X^{N-R3}$

$R4 = X^{N}$

BREQ = salto se sono uguali (Equal)
FMULT = prodotto float

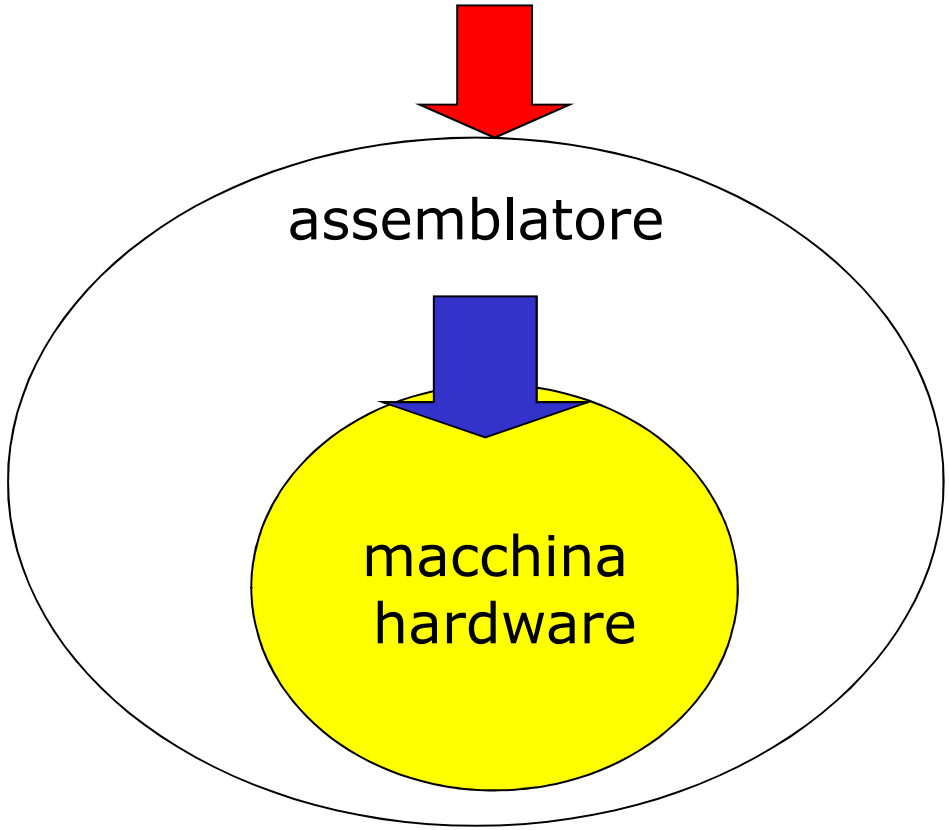


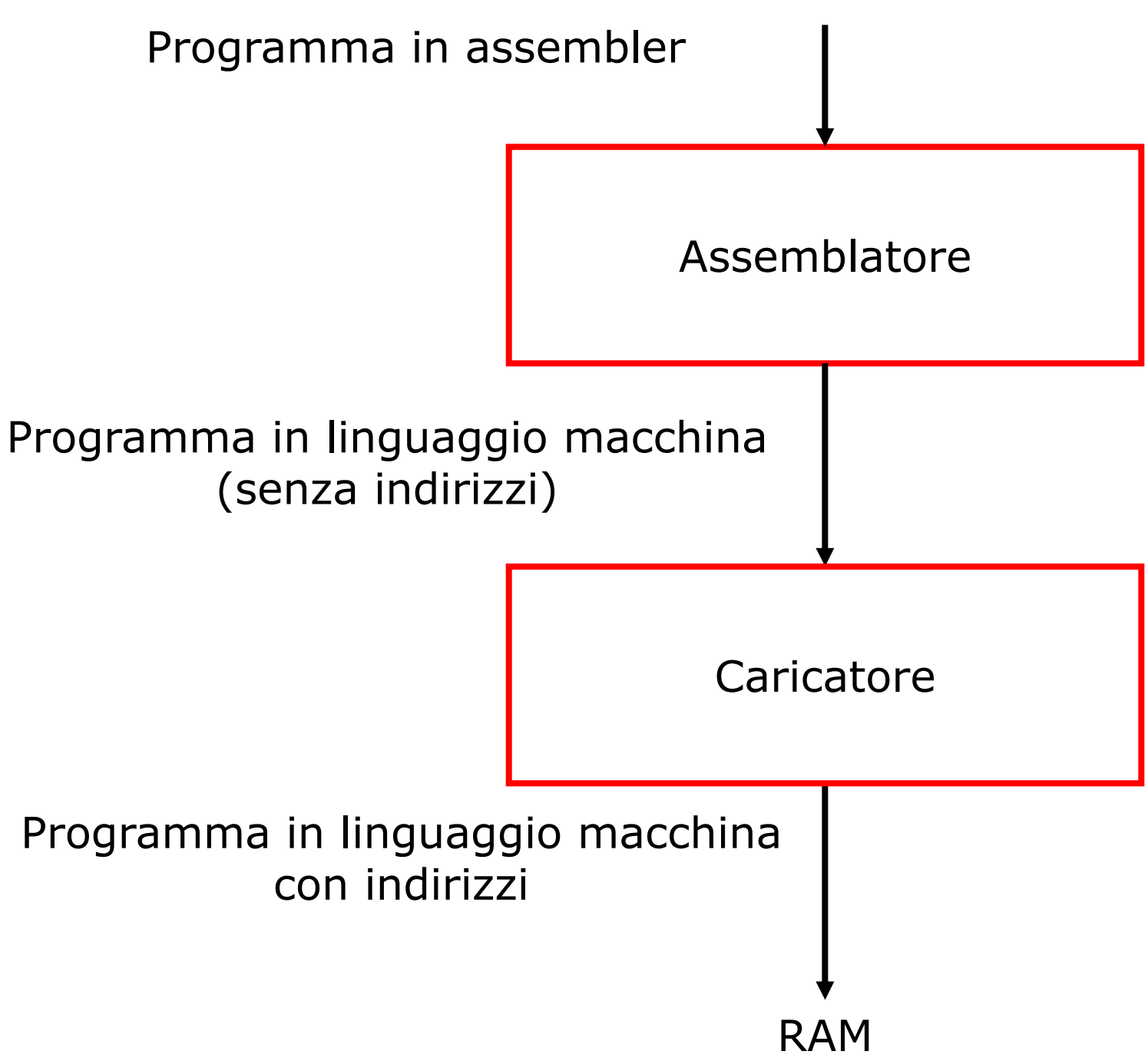
ciclo o iterazione

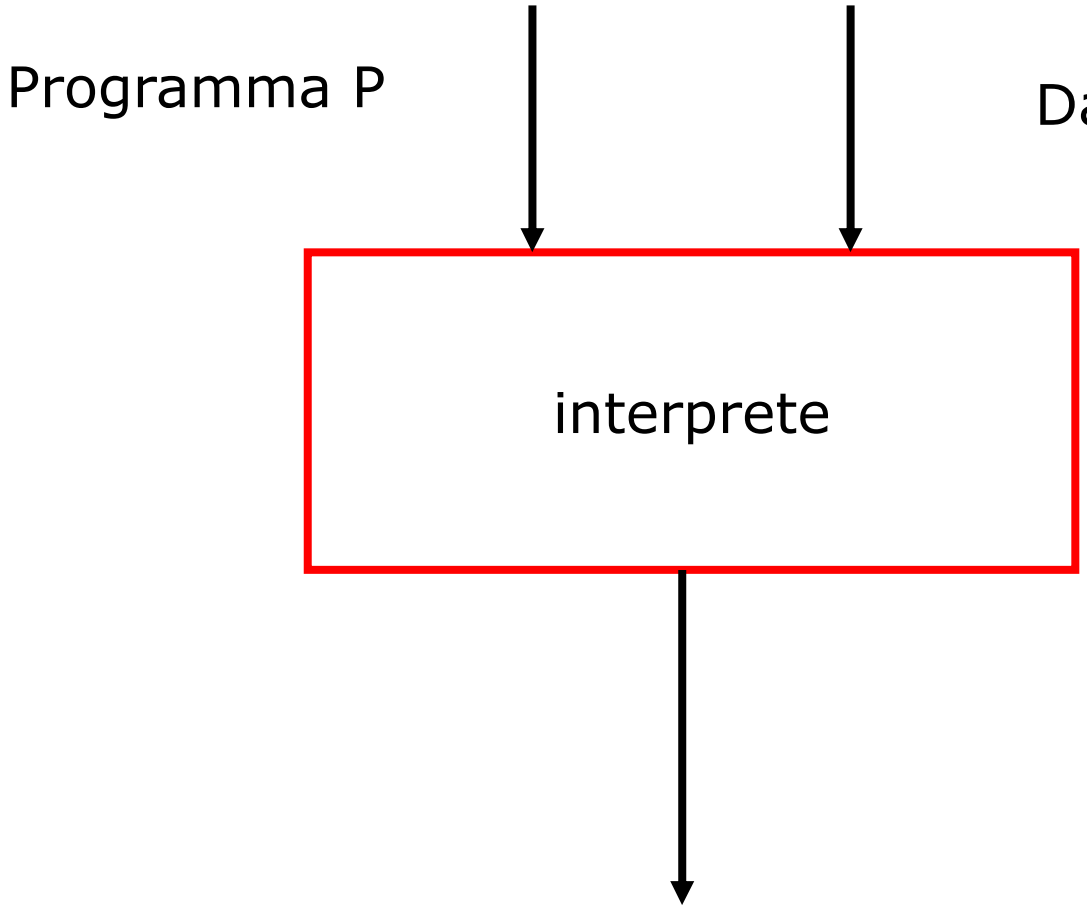
- Nel programma precedente, per calcolare x^n, il ciclo viene ripetuto n volte.
- Il tempo calcolo richiesto aumenterà proporzionalmente con l'aumentare di n.
- Diciamo che il programma ha **complessità tempo O(n).**
- Nel seguito del corso verrà trattata una sezione riguardante il calcolo della complessità

La CPU non "capisce" l'assembler !!

il programma assembler deve essere tradotto in un programma macchina




Programma in assembler
Assemblatore
Programma in linguaggio macchina
(senza indirizzi)
Caricatore
Programma in linguaggio macchina
con indirizzi
RAM



Gli stessi risultati che si otterrebbero eseguendo il programma P con i dati D

OSSERVAZIONE

Il programma *assemblatore* legge un programma in assembler e il programma *interprete* legge sia un programma che i dati per tale programma.

Esistono quindi programmi che hanno dei programmi come dati.

Lo Switch

- La selezione in C
- Esempi

Uso degli operatori logici

- Formulazione di condizioni di esecuzione nelle istruzioni con esecuzione condizionata al valore di un'espressione
- Esempio: condizioni perché un triangolo di lati a, b, c risulti equilatero, isoscele, scaleno (formulate come espressioni in C)

equilatero: `( a == b ) && ( a == c )`

scaleno: `(a != b) && (a != c) && (b != c)`

isoscele:
```
(( a == b ) && ( a != c )) ||
(( a == c ) && ( a != b )) ||
(( b == c ) && ( a != b )) ||
```

Strutture di controllo

- La formulazione di molti algoritmi richiede che il linguaggio di programmazione in uso offra la possibilità di modificare il flusso di esecuzione delle istruzioni in base al valore di un'espressione (modifiche condizionate)
- Strutture di selezione: ramificazione condizionata del flusso
- Strutture di iterazione: ripetizione condizionata di un blocco di istruzioni

Strutture di selezione in C

- **Selezione singola if**: viene eseguita o meno una singola azione
- **Selezione doppia if else**: viene eseguita una tra due azioni logicamente distinte (caso particolare: una delle due azioni è nulla

  $\longrightarrow$ selezione singola)
- **Selezione multipla switch**: viene eseguita una tra un certo numero di azioni logicamente distinte
- Un'azione è formulata con un blocco di istruzioni
- La selezione è sempre in base al valore di un'espressione, distinguendo solo due casi: **zero** e **diverso da zero**

C: Selezione switch (1)


```
switch ( espressione )
{
 case costante1   : sequenza-istruzioni1          ; break ;
 case costante2   : sequenza-istruzioni2          ; break ;
 ...
 case costanteN   : sequenza-istruzioniN          ; break ;
 default : sequenza-istruzioni ; break ;
}
```

Se il valore di espressione uguaglia una delle costanti specificate, allora si esegue la sequenza-istruzioni corrispondente, altrimenti si esegue la sequenza-istruzioni di default

C: Selezione switch (2)


```
switch ( espressione )
{
 case costante1   :
 case costante2 : sequenza-istruzioni1-2; break ;
 ...
 case costanteN   : sequenza-istruzioniN          ; break ;
 default : sequenza-istruzioni ; break ;
}
```

Variante al caso precedente: è possibile associare una stessa sequenza istruzioni a più case (esempio: Se il valore di espressione uguaglia costante1 o costante2, si esegue sequenza-istruzioni1-2)

Esempio: soluzione di un problema

Scrivere un programma che legge da standard input tre numeri interi che rappresentano una data (giorno, mese e anno) e stampa tre interi che rappresentano la data del giorno successivo

- Tenere conto del fatto che i mesi possono essere di 28 (o 29 negli anni bisestili), 30 e 31 giorni
- Anno bisestile: divisibile per 400 oppure divisibile per 4 e non divisibile per 100. Ad esempio, il 1996 è un anno bisestile, dato che è divisibile per 4 e non è divisibile per 100. L'anno 2000 è un anno bisestile, dato che è divisibile per 400

Algoritmo: Data giorno successivo (1)

1. Leggi giorno, mese, anno
2. Determina numero giorni mese
   Se mese è 1,3,5,7,8,10,12: giornimese = 31;
   Se mese è 2: se anno è bisestile,
   giornimese = 29, altrimenti giornimese = 28;
   Se mese è 4, 6, 9, 11: giornimese = 30;
   Diversamente: mese errato

Algoritmo: Data giorno successivo (2)

3. Determina data giorno successivo

   Se mese errato oppure giorno errato:

   stampa messaggio d'errore

   Altrimenti, se giorno diverso da giornimese:

   data giorno successivo = (giorno+1, mese, anno)

   Altrimenti, se mese diverso da 12:

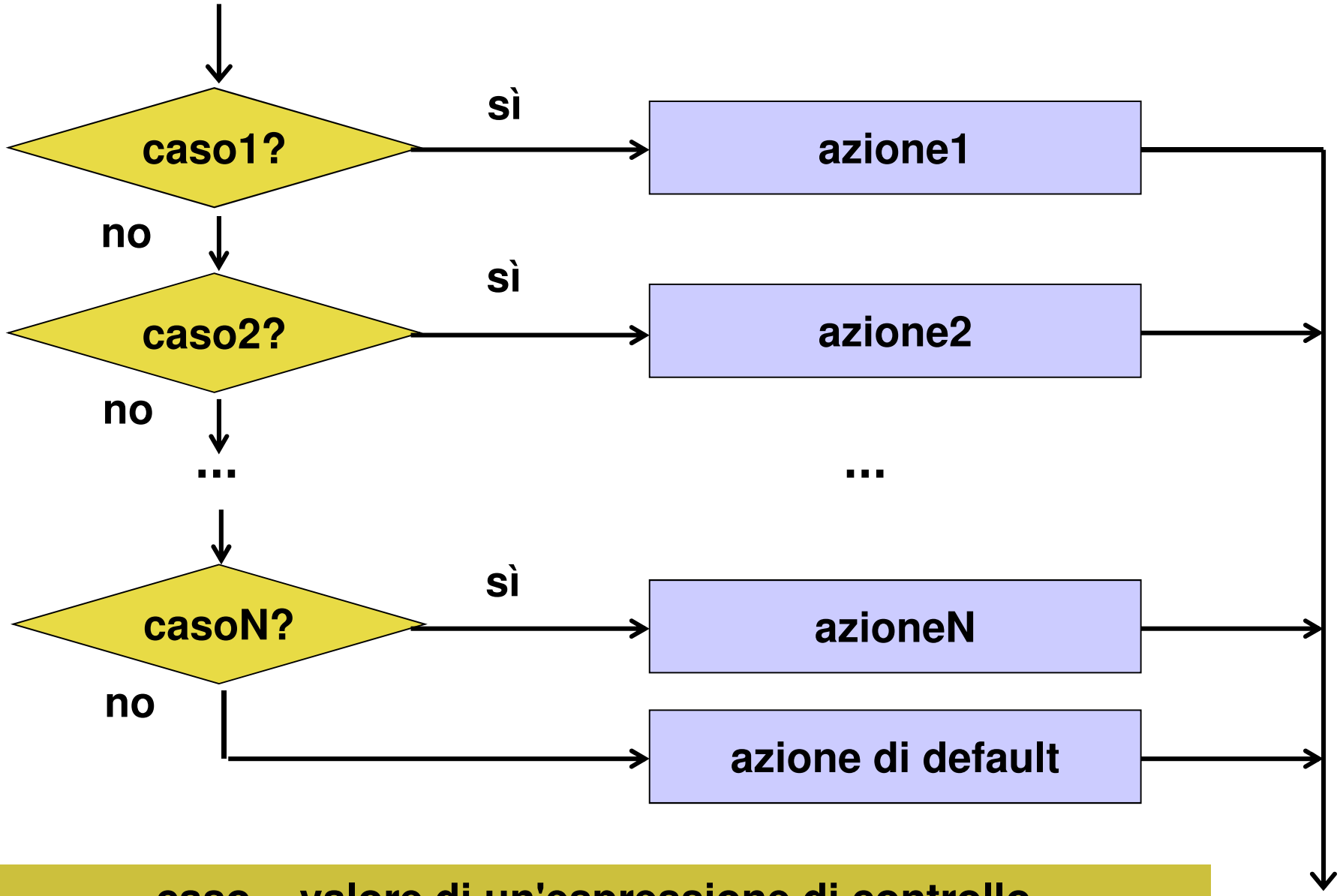
   data giorno successivo = (1, mese+1, anno)

   Altrimenti:

   data giorno successivo = (1, 1, anno+1)

4. Stampa data giorno successivo

Selezione multipla



caso = valore di un'espressione di controllo

Codifica in C (1)


```
int main( ) {
int giorno, mese, anno, giornimese ;
printf( "\nInserire giorno, mese, anno interi\n" ) ;
scanf( "%d%d%d" , &giorno , &mese , &anno ) ;
    switch (mese) {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
        giornimese = 31; break;
    case 4: case 6: case 9: case 11:
        giornimese = 30; break;
    case 2:
        if (( anno%400==0 ) || (( anno%4==0 ) && ( anno%100!=0 )))
            giornimese = 29 ;
        else
            giornimese = 28 ;
        break;
    default: giornimese = 0; break;
}
```

Codifica in C (2)


```
if ( giornimese == 0 || giorno < 1 || giorno > giornimese )
     printf( "\nData immessa non valida!\n" ) ;
else {
     if  ( giorno < giornimese )
          giorno = giorno +1 ;
     else {
          giorno = 1 ;
          if ( mese == 12 ) {
               mese = 1 ;
               anno = anno +1 ;
          }
          else
               mese = mese +1 ;
     }
printf( "giorno successivo:\n%d,%d,%d\n", giorno, mese, anno );
}
return 0 ;
  }
```

Qualificatori di tipo

- tipi di dato primitivi
- qualificatori di tipo
- short int
- int
- long int
- unsigned short int
- short
- unsigned int
- unsigned long int
- byte
- float
- double
- specifiche di conversione per gli interi
- operandi interi
- tipo di una costante
- operatori aritmetici per interi
- operatori di assegnamento
- conversione di tipo
- Espressioni che coinvolgono tipi di dato primitivi numerici diversi
- Assegnazioni fra tipi di dato primitivi numerici diversi
- Esercitazione 4 – 22 Maggio 2007

Tipi di dato primitivi

- Per definire un tipo di dato si deve specificare:
  - **Dominio:** insieme dei possibili valori rappresentabili nella memoria del calcolatore mediante il tipo di dato primitivo (sempre un insieme finito di valori)
  - **Operazioni:** operatori del linguaggio che permettono di effettuare operazioni elementari su valori del tipo primitivo (Es. `+`, `-`, `/`, `*`, ecc.)
  - **Letterali:** simboli del linguaggio che definiscono i valori del tipo primitivo (Es. `10`, `3.14`, `'A'`, `true`, ecc.)

C: qualificatori di tipo

- Tipi base: char (caratteri), int (interi), float e double (reali)
- E' possibile modificare alcune caratteristiche dei tipi base mediante i *qualificatori* short, long, signed e unsigned
- short e long possono modificare il numero di bit utilizzato per la rappresentazione
- signed e unsigned possono modificare l'insieme dei valori rappresentati
- Esempio:

  long int (interi, intervallo rappresentato include quello rappresentato con int)

C: Tipo short int

- Intervallo *minimo*: [-32767, 32767]
- Intervallo *vero*: dipende dall'implementazione; è specificato da SHRT_MIN e SHRT_MAX in "limits.h"
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella in complemento a 2
- Nelle dichiarazioni, short int o short
- Specifica di conversione: %hd

C: Tipo int

- Intervallo *minimo*: [-32767, 32767]
- Intervallo *vero*: dipende dall'implementazione; è specificato da INT_MIN e INT_MAX in "limits.h"; deve includere l'intervallo usato per il tipo short int
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella in complemento a 2
- Nelle dichiarazioni int
- Specifica di conversione: %d

Il tipo di dato int

Tipo	int	
Dimensione	32 bit (4 byte)	
Dominio	l'insieme dei numeri interi nell'intervallo $[-2^{31},+2^{31}-1]$ (oltre 4 miliardi di valori)	
Operazioni	+	somma
	-	sottrazione
	*	prodotto
	/	divisione intera
	%	resto della divisione intera
Letterali	sequenze di cifre che denotano valori del dominio (es. 275930)	

```
int a,b,c;        // Dichiarazione di variabile di tipo int
a = 1;            // Uso di letterali
b = 2;
c = a + b;        // Espressione aritmetica
```

C: Tipo long int

- Intervallo *minimo*: [-2147483647, 2147483647]
- Intervallo *vero*: dipende dall'implementazione; è specificato da LONG_MIN e LONG_MAX in "limits.h"; deve includere l'intervallo usato per il tipo int
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 32 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella in complemento a 2
- Nelle dichiarazioni: long int o long
- Specifica di conversione: %ld

Il tipo di dato long

Tipo	long	
Dimensione	64 bit (8 byte)	
Dominio	l'insieme dei numeri interi nell'intervallo $[-2^{63},+2^{63}-1]$	
Operazioni	+	somma
	-	sottrazione
	*	prodotto
	/	divisione intera
	%	resto della divisione intera
Letterali	sequenze di cifre terminate con una l o L che denotano valori del dominio (es.9000000000L)	

```
long a,b;                 // Dichiarazione di variabile di tipo long
a = 9000000000L;          // Uso di letterali
b = a+1;                  // Espressione aritmetica
```

C: Tipo unsigned short int

- Intervallo *minimo*: [0, 65535]
- Intervallo *vero*: dipende dall'implementazione, è specificato da USHRT_MAX in "limits.h" (valore minimo è sempre 0)
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella posizionale
- Nelle dichiarazioni: unsigned short int o unsigned short
- Specifica di conversione: %hu, %ho (ottale), %hx o %hX (esadecimale, lettere minuscole o maiuscole)

Il tipo di dato short

Tipo	short	
Dimensione	16 bit (2 byte)	
Dominio	l'insieme dei numeri interi nell'intervallo $[-2^{15},+2^{15}-1] = [-32768, 32767]$	
Operazioni	+	somma
	-	sottrazione
	*	prodotto
	/	divisione intera
	%	resto della divisione intera
Letterali	sequenze di cifre che denotano valori del dominio (es. 22700)	

```
short a,b;        // Dichiarazione di variabile di tipo short
a = 22700;     // Uso di letterali
b = a+1;                    // Espressione aritmetica
```

C: Tipo unsigned int

- Intervallo *minimo*: [0, 65535]
- Intervallo *vero*: dipende dall'implementazione, è specificato da UINT_MAX in "limits.h" (valore minimo è sempre 0)
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 16 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella posizionale
- Nelle dichiarazioni: unsigned int o unsigned
- Specifica di conversione: %u, %o (ottale), %x o %X (esadecimale, lettere minuscole o maiuscole)

C: Tipo unsigned long int

- Intervallo *minimo*: [0, 4294967295]
- Intervallo *vero*: dipende dall'implementazione, è specificato da ULONG_MAX in "limits.h" (valore minimo è sempre 0)
- Occupazione: dipende dall'implementazione; per l'intervallo minimo sono sufficienti 32 bit
- Rappresentazione: dipende dall'implementazione; tipicamente, è quella posizionale
- Nelle dichiarazioni: unsigned long int o unsigned long
- Specifica di conversione: %lu, %lo (ottale), %lx o %lX (esadecimale, lettere minuscole o maiuscole)

Il tipo di dato byte

Tipo	byte	
Dimensione	8 bit (1 byte)	
Dominio	l'insieme dei numeri interi nell'intervallo $[-2^7,+2^7-1]$ = [-128, 127]	
Operazioni	+	somma
	-	sottrazione
	*	prodotto
	/	divisione intera
	%	resto della divisione intera
Letterali	sequenze di cifre che denotano valori del dominio (es. 47)	

```
int a,b,c;      // Dichiarazione di variabile di tipo byte
a = 1;          // Uso di letterali
b = a+1;                    // Espressione aritmetica
```

Il tipo di dato float

Tipo	float		
Dimensione	32 bit (4 byte)		
Dominio	insieme di 2^{32} numeri reali + e -	|min|	$1.4012985 * 10^{-38}$
		|max|	$3.4028235 * 10^{+38}$
		Precisione	~7 cifre decimali
Operazioni	+	somma	
	-	sottrazione	
	*	prodotto	
	/	divisione	
Letterali	sequenze di cifre con punto decimale terminate con una f (o F) che denotano valori del dominio (es. 3.14f)		
	rappresentazione in notazione scientifica (es. 314E-2f)		

Il tipo di dato float

Esempio:

```
float a;    // Dichiarazione di variabile di tipo float
a = 3.14f;  // Uso di letterali
a = a*2f;   // Espressione aritmetica
```

Il tipo di dato double

Tipo	double		
Dimensione	64 bit (8 byte)		
Dominio	insieme di 2^{64} numeri reali + e -	|min|	$1.7976931348623157 * 10^{-308}$
		|max|	$2.250738585072014 * 10^{+308}$
		Precisione	~ 15 cifre decimali
Operazioni	+	somma	
	-	sottrazione	
	*	prodotto	
	/	divisione	
Letterali	sequenze di cifre con punto decimale opzionalmente terminate con una d o (D), che denotano valori del dominio (Es. 3.14 oppure 3.14d)		
	Rap. in not. scientifica (Es. 314E-2 oppure 314E-2d)		

Specifiche di conversione per tipi interi

short int	int	long int
%hd	%d	%ld

unsigned short int	unsigned int	unsigned long int
%hu	%u	%lu
%ho	%o	%lo
%hx	%x	%lx
%hX	%X	%lX

C: Operandi interi (1)

- Variabili di uno dei tipi int
- Costanti usate direttamente nelle espressioni
- Esempio: b = 2 * a + 33 * b - c / 19 ;
- Una costante viene specificata con

  segno: + o -, opzionale

  sequenza di cifre: in base 10, 8 (prefisso 0, [0-7]), 16 (prefisso 0x o 0X, [0-9], [a-f] o [A-F])

  suffisso: u o U per unsigned, l o L per long
- Esempi: -165438L, 0xFFFFFFFFl, -0765, 0XaAaA1, +2147483647L

C: Operandi interi (2)

- Costanti introdotte con #define
- Forma: #define nome valore
- Effetto: *ogni* occorrenza successiva di nome sarà rimpiazzata con valore (*qualunque* esso sia!)
- Nome: stesse regole date per il nome di variabili
- #define è una direttiva per il compilatore (elaborata dal preprocessore a *tempo di compilazione*)
- Uso tipico: per modificare valori di costanti *senza interventi pesanti* sul testo del programma (si ricompila il programma dopo aver aggiornato solo il valore che compare nella #define)

C: Tipo di una costante

- Il *tipo* di una costante intera *dipende* da come viene specificata
- Base 10, senza suffisso: primo tipo possibile tra int, long int e unsigned long int
- Base 8 o 16, senza suffisso: primo tipo possibile tra int, unsigned int, long int e unsigned long int
- Con suffisso u o U: primo tipo possibile tra unsigned int e unsigned long int
- Con suffisso l o L: primo tipo possibile tra long int e unsigned long int

C: Operatori aritmetici per interi (1)

- Operatori binari (due operandi): somma (+), sottrazione (-), prodotto (*), quoziente (/), resto (%)
- Operatori unari (un operando): segno (+), inversione segno (-)
- Attenzione: se il risultato di un'operazione eccede i limiti della rappresentazione, il comportamento *dipende* dall'implementazione
- Attenzione: se almeno uno degli operandi è negativo, il comportamento di / e % *dipende* dall'implementazione

C: Operatori di assegnamento

- Operatore = usato nella forma

  variabile = espressione
- Operatore +=, -=, *=, /=, %= (indicato qui con op=) usato nella forma

  variabile op= espressione
- Significato

  variabile = variabile op (espressione)

  Esempi:

  a += 3*b + c → a = a + (3*b + c)

  x *= 3 + b - 2*c → x = x * (3 + b - 2*c)

C: Conversione di tipo (1)

- Conversione *implicita* nel caso di operatori binari utilizzati con due operandi di tipo diverso: uno dei due tipi (inferiore) viene promosso all'altro (superiore) ed il risultato è del tipo superiore
- ESEMPIO di regola (informale)

  oper1 * oper2: se oper1 è double e oper2 è float, oper2 *promosso* a double e risultato è double
- Attenzione: a , b float e c double

  a = b + c ; assegnerà il valore double risultato della somma ad una variabile float con potenziale *perdita di informazione* (in un assegnamento il tipo della variabile destinazione definisce il tipo finale)

C: Conversione di tipo (2)

- Conversione *implicita* nella chiamata di funzione
- Gli argomenti passati ad una funzione *dichiarata con prototipo* vengono convertiti secondo quanto specificato nella dichiarazione con potenziale *perdita di informazione*
- ESEMPIO: se gli argomenti formali sono float e int, una chiamata con argomenti attuali int e float provoca una potenziale *perdita di informazione*

C: Conversione di tipo (3)

- Conversione *implicita* nella return all'interno di una funzione
- In return (espressione), il tipo assegnato al valore di espressione è quello specificato nella dichiarazione della funzione (potenziale *perdita di informazione*)
- ESEMPIO: b float e c int, tipo di ritorno int return b * c ; convertirà il risultato float del prodotto in un int con potenziale *perdita di informazione*

C: Conversione di tipo (4)

- Conversione *esplicita* di tipo può essere forzata con operatore cast
- (tipo) espressione provoca la valutazione di espressione come se il risultato dovesse essere assegnato ad una variabile del tipo forzato
- ESEMPIO di utilizzazione: conversione tipo argomento nella chiamata funzioni di libreria (potrebbero non fare uso di dichiarazione tramite prototipo)
- sqrt((double) n) chiamata di sqrt su n int convertito a double (valore di n immutato!)

Espressioni che coinvolgono tipi di dato primitivi numerici diversi (1)

- Vediamo una tabella che descrive il tipo risultante di una espressione della forma a+b per ciascuna coppia di tipi possibili di a e di b

Espressioni che coinvolgono tipi di dato primitivi numerici diversi (2)

a+b	byte b	short b	int b	long b	float b	double b
byte a	byte	short	int	long	float	double
short a	short	short	int	long	float	double
int b	int	int	int	long	float	double
long b	long	long	long	long	float	double
float b	float	float	float	float	float	double
double b	double	double	double	double	double	double

Assegnazioni fra tipi di dato primitivi numerici diversi (1)

- Un valore di un tipo di dato *non* può essere assegnato ad una variabile di un tipo di dato con minor dimensione, altrimenti si rischia perdita di precisione dell'informazione.
- Un valore reale non può essere assegnato ad una variabile intera.

  int a; long b; a = b;

  Errore: un valore long non può essere assegnato ad una variabile int

  int a; float b; a = b;

  Errore: un valore float non può essere assegnato ad una variabile int

Assegnazioni fra tipi di dato primitivi numerici diversi (2)

a=b	byte b	short b	int b	long b	float b	double b
byte a	OK	ERRORE	ERRORE	ERRORE	ERRORE	ERRORE
short a	OK	OK	ERRORE	ERRORE	ERRORE	ERRORE
int b	OK	OK	OK	ERRORE	ERRORE	ERRORE
long b	OK	OK	OK	OK	ERRORE	ERRORE
float b	OK	OK	OK	OK	OK	ERRORE
double b	OK	OK	OK	OK	OK	OK

Le funzioni

- Concetto di modulo di un programma
- Funzioni in C
- Introduzione di una funzione
- Esempio di funzione in C
- Prototipo di una funzione
- Definizione di funzione
- Chiamata di funzione
- Prototipo, definizione, chiamata
- Programma con funzioni
- Passaggio degli argomenti
- Variabili locali
- Esempi
- Libreria matematica

Concetto di modulo di un programma (1)

- Lo sviluppo di programmi complessi avviene tipicamente per composizione di moduli, ognuno dei quali esegue uno compito semplice
- Con l'uso della modularizzazione (scomposizione di un programma in moduli) si ottengono normalmente programmi facilmente
  - testabili/mantenibili (sono possibili operazioni di test/modifica limitate a singoli moduli)
  - leggibili/documentabili (il programma si presenta come una composizione di compiti semplici)

  (… e si semplifica la ripartizione di un progetto tra più progettisti)

Concetto di modulo di un programma (2)

- La modularizzazione è particolarmente apprezzata quando almeno un modulo è usato più volte durante l'esecuzione del programma (tipicamente, in questo caso si ha anche una compattazione del codice sorgente)
- E' utile tentare di anticipare la modularizzazione alla fase di sviluppo dell'algoritmo per la soluzione di un problema

Funzioni in C (1)

- Un programma C si compone di funzioni
- E' disponibile una collezione *predefinita* di funzioni che possono essere usate *direttamente* in ogni programma (libreria standard)
- Per introdurre una nuova funzione occorre specificare:
  - una sequenza di operazioni
  - gli argomenti su cui operare
  - il risultato da restituire
- Un programma C viene modularizzato con l'uso di funzioni

Funzioni in C (2)

- Le istruzioni specificate in una funzione sono eseguite quando la funzione viene attivata
- La funzione main è attivata quando il programma viene messo in esecuzione
- Le altre funzioni sono attivate durante l'esecuzione del programma, tramite chiamata (call)
- Una funzione può chiamare una o più funzioni
- Una funzione può chiamare se stessa (chiamata ricorsiva)

C: Introduzione di una nuova funzione

- Nel programma, una funzione può essere chiamata solo dopo averla dichiarata
- La dichiarazione di una funzione avviene per mezzo di un prototipo di funzione nel quale vengono specificati nome, argomenti e risultato della funzione
- La sequenza di istruzioni eseguite da una funzione è invece specificata nella sua definizione
- La chiamata di una funzione avviene specificando il nome di questa e gli argomenti da usare durante l'esecuzione

Esempio di funzione C (1)

- Cubo di un intero: argomento int, risultato int
- Dichiarazione: nome, argomenti, risultato

  int cubo (int) ;
- Chiamata: specificare nome, argomento

  a = cubo (b) ; (a e b interi, produrrà a = b^3)
- Definizione: specificare nome, argomento su cui operare, risultato

```
int cubo ( int n )
{
  return n*n*n ;
}
```

Esempio di funzione C (2)


```
…
int cubo( int ) ;          /* prototipo funzione cubo */
int main( )
{
   int num ;
   …                       /* attivazione funzione cubo */
   printf( "cubo(%d) = %d", num , cubo( num ) ) ;
   …
   return 0;
}                          /* definizione funzione cubo */
int cubo( int n )
{
   return n*n*n ;
}
```

C: Prototipo di funzione

- **Formato**

  tipo-restituito nome (lista-argomenti) ;

- **tipo-restituito** è il tipo del valore restituito dalla funzione come risultato: se non viene restituito alcun tipo usare void
- **nome** è utilizzato nella chiamata e nella definizione: stesse regole date per il nome di una variabile
- lista-argomenti specifica un tipo ed un nome opzionale per ogni argomento della funzione; una virgola separa i vari argomenti; può essere vuota

C: Definizione di funzione

- **Formato (deve essere *coerente* con il prototipo)**

  tipo-restituito nome (lista-argomenti)

  { corpo-funzione }

- **tipo-restituito** e **nome**: vedi prototipo
- **lista-argomenti** per ogni argomento della funzione specifica un tipo ed un nome; una virgola separa i vari argomenti; può essere vuota; *numero/tipo di argomenti e loro ordine come nel prototipo*
- **corpo-funzione** specifica le azioni da eseguire: contiene dichiarazioni di variabili, istruzioni, etc. Per restituire un valore si usa (*almeno una volta*)

  return espressione ;

C: Chiamata di funzione

- **Formato (deve essere *coerente* con il prototipo)**

  nome (lista-espressioni)

- nome: vedi prototipo
- lista-espressioni contiene un'espressione per ogni argomento; espressioni separate da virgola; *numero/tipo di espressioni e loro ordine come nel prototipo*
- **Chiamata di funzione inserita in un'istruzione.**
- Se la funzione restituisce un valore, la chiamata compare normalmente in un'espressione il cui valore è assegnato ad una variabile
- Esempi

```
funct1( alpha ) ;

y = 2*cubo(x) + 7 ;
```