Fondamenti di Informatica

Ingegneria Clinica

Lezione 09/11/2009

Prof. Raffaele Nicolussi

FUB - Fondazione Ugo Bordoni

Via B. Castiglione 59 - 00142 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	**Giovedì**	**14:00 – 16.30**
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

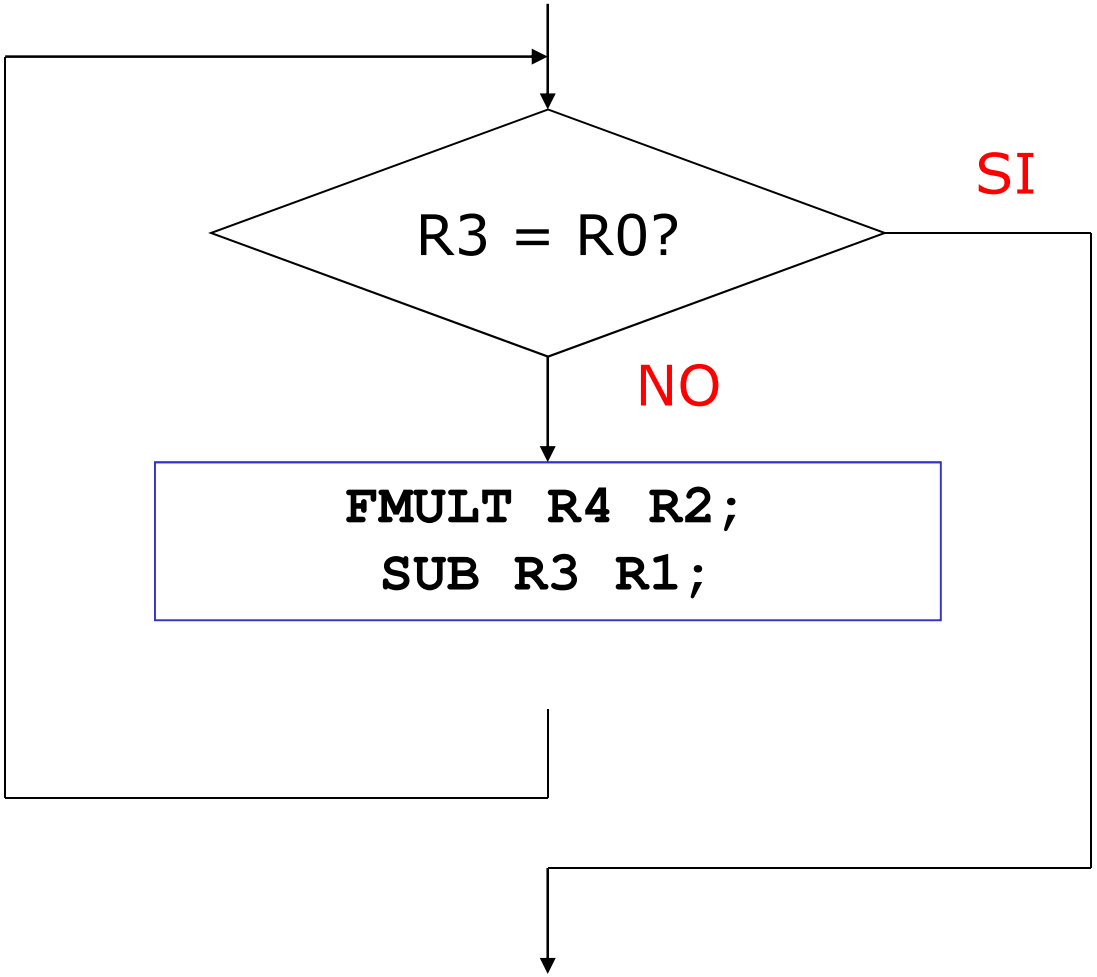
R4 = 1 reale

```
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

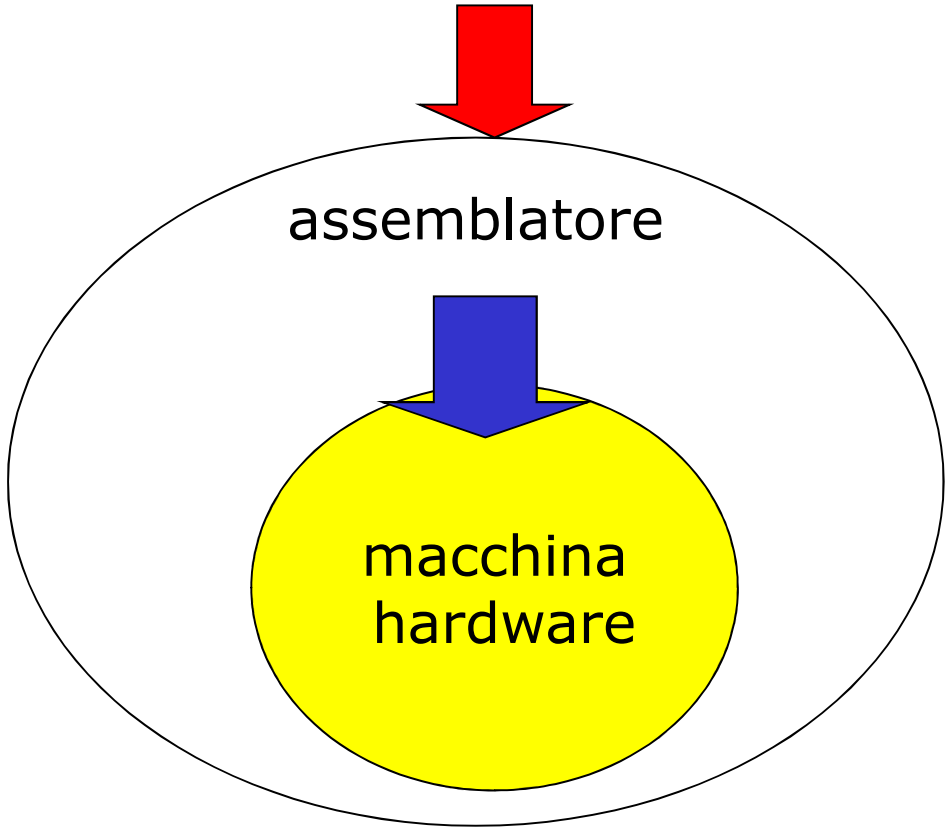


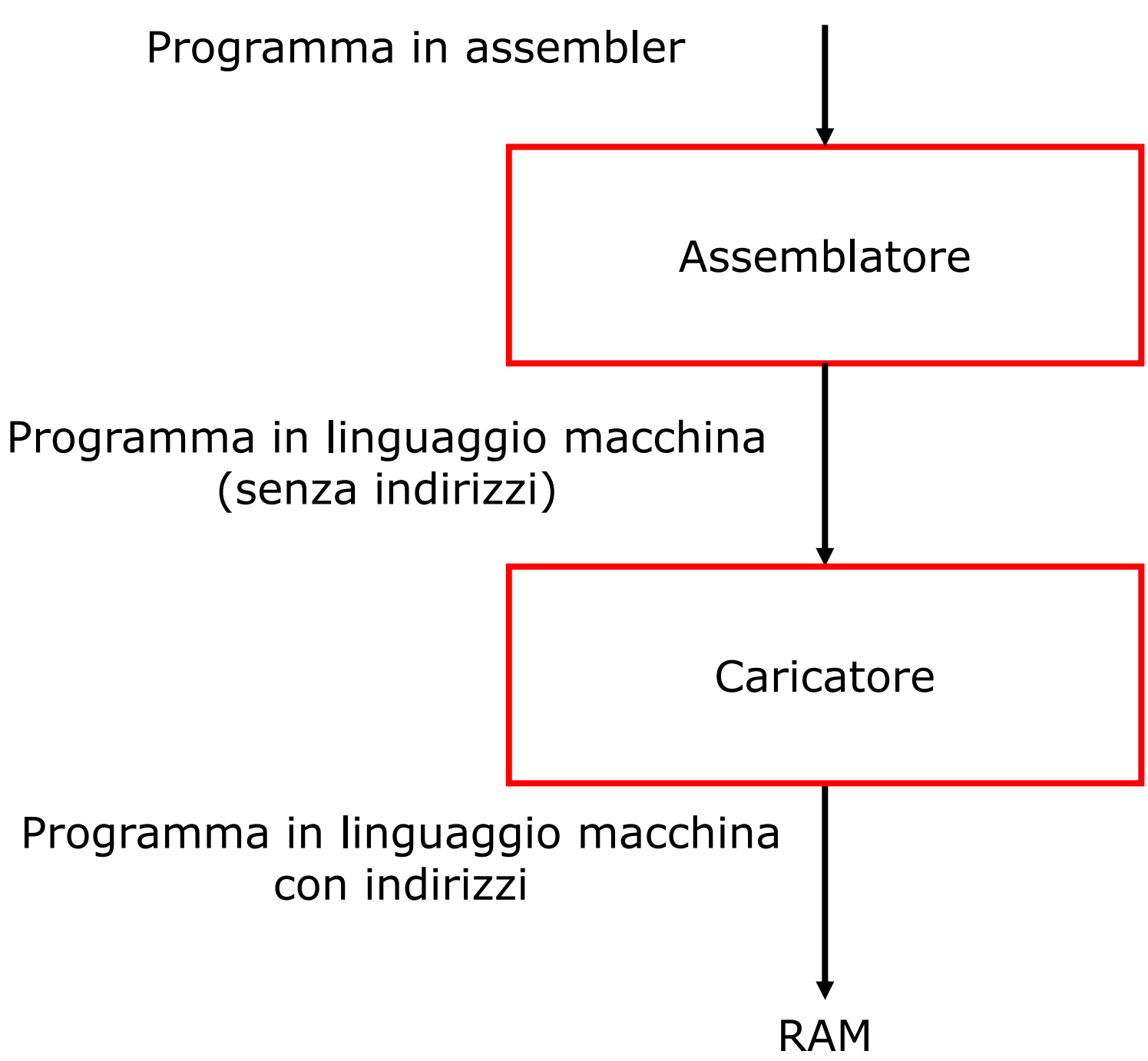
ciclo o iterazione

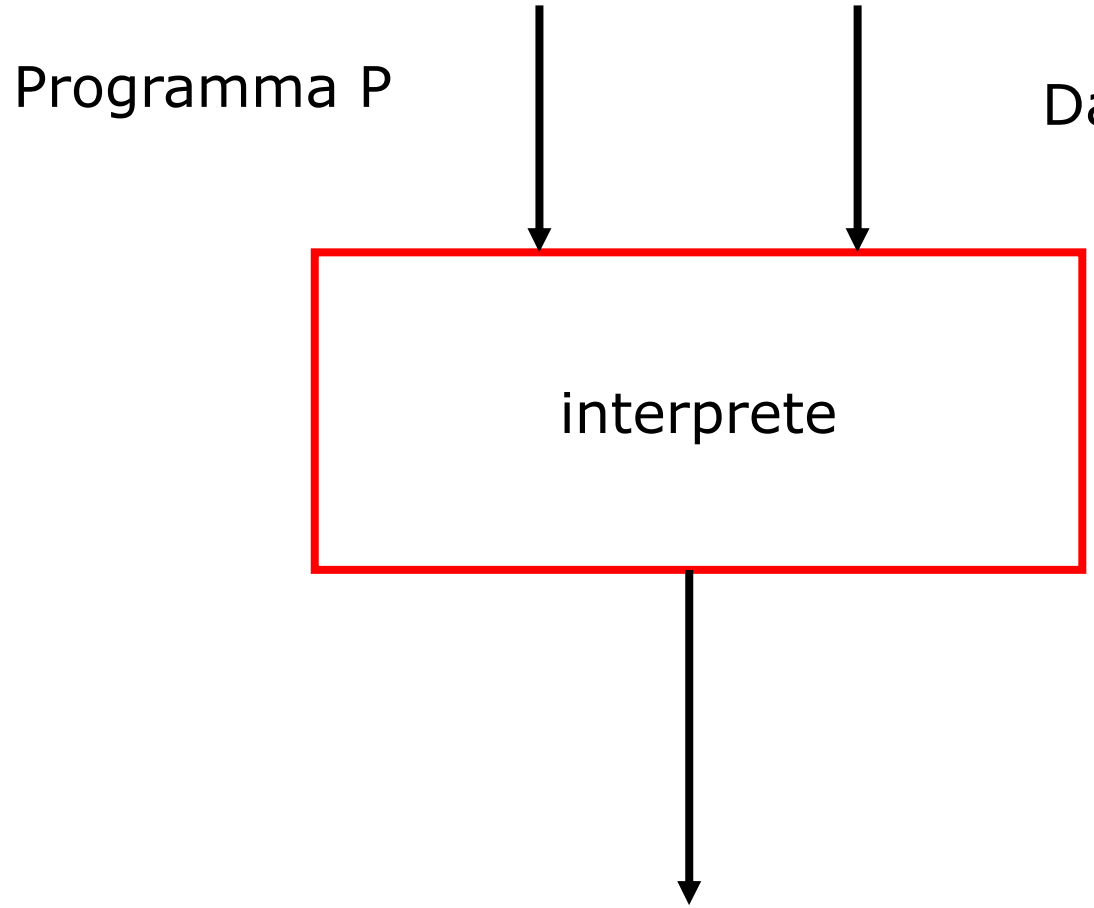
- Nel programma precedente, per calcolare x^n, il ciclo viene ripetuto n volte.
- Il tempo calcolo richiesto aumenterà proporzionalmente con l'aumentare di n.
- Diciamo che il programma ha **complessità tempo O(n).**
- Nel seguito del corso verrà trattata una sezione riguardante il calcolo della complessità

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:

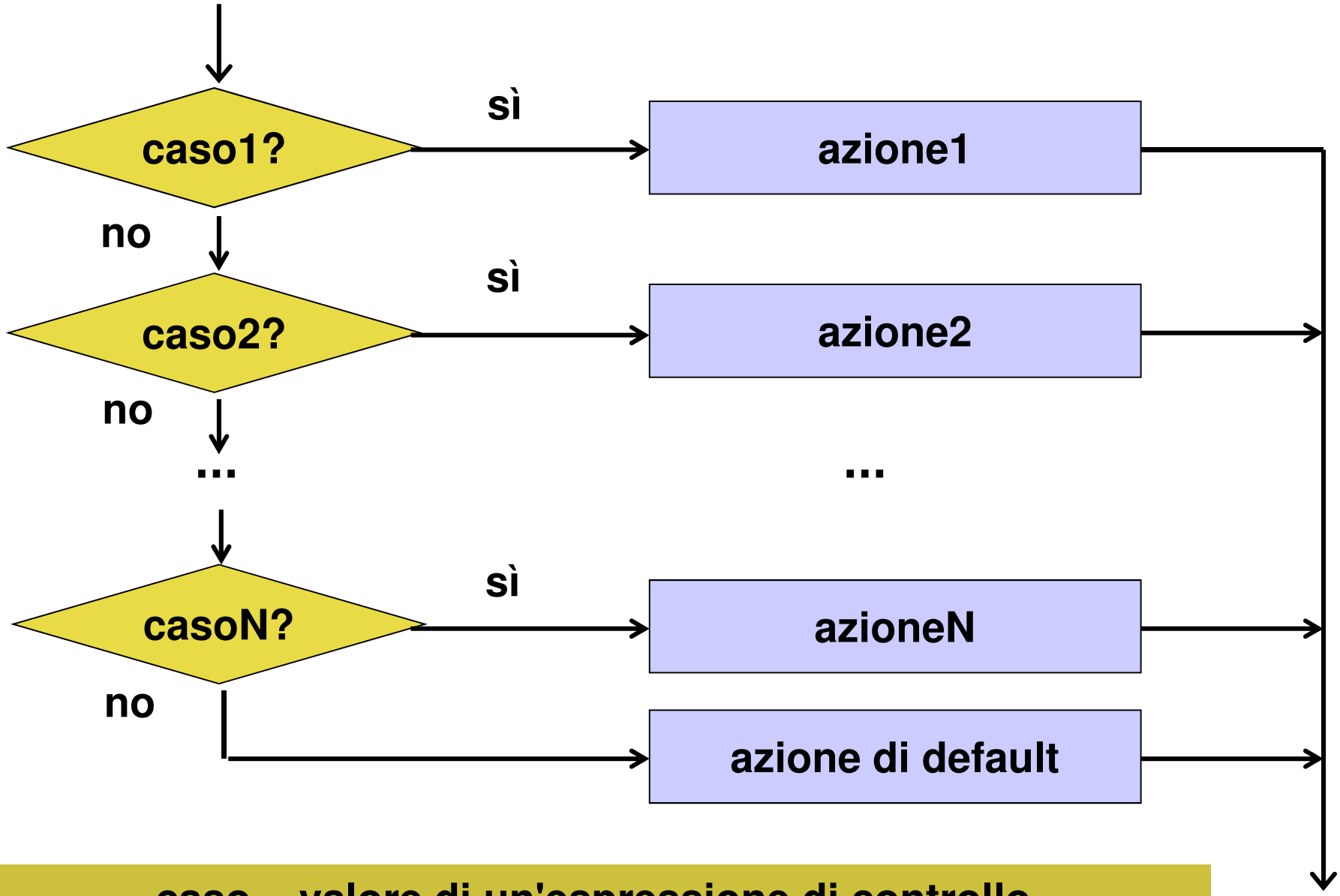
   data giorno successivo = (1, mese+1, anno)

   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 ;
  }
```

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 ;
```