*Programmazione I*

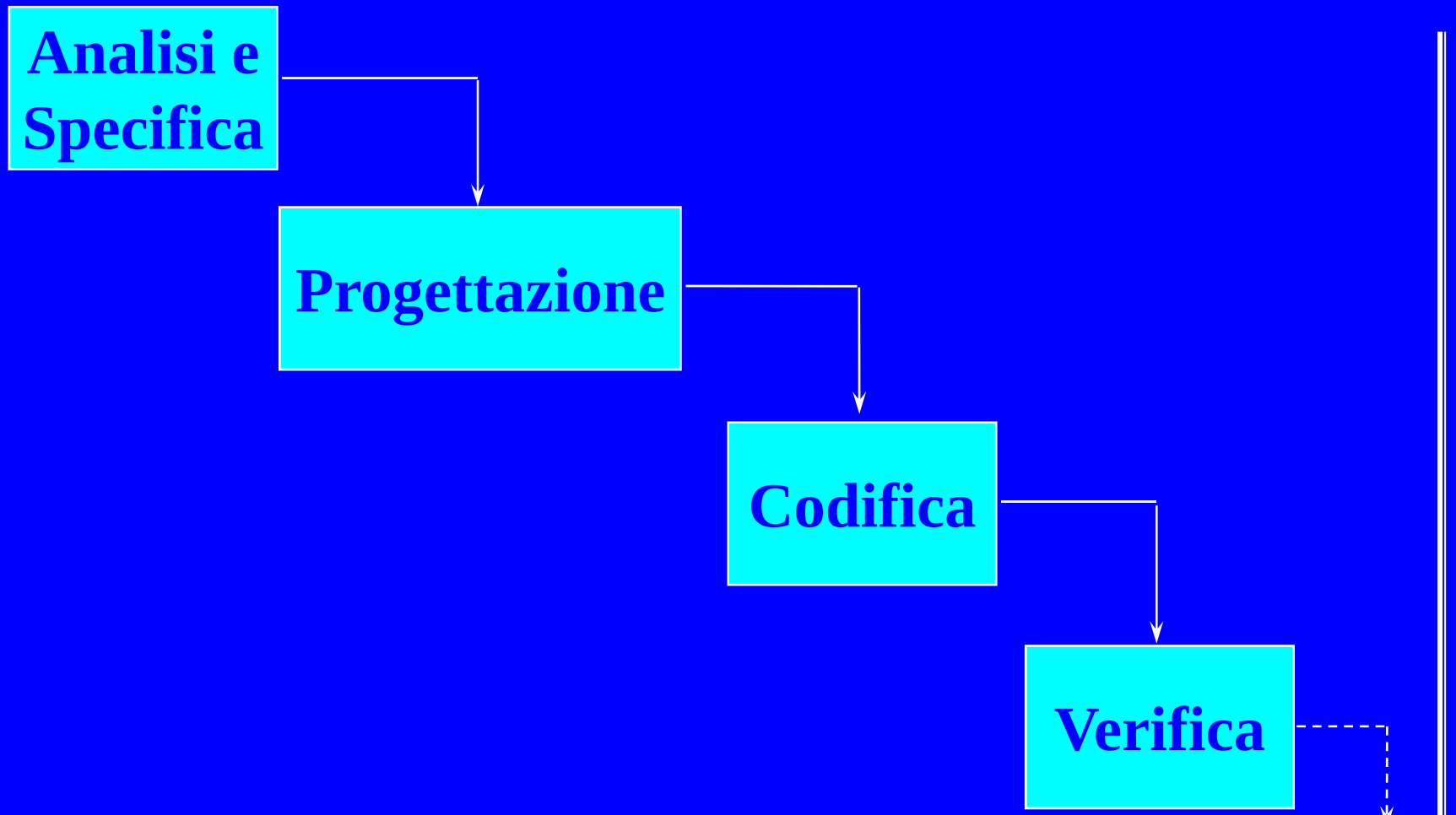
*a.a 2009-2010*

*docente: Carmine Gravino*

# Sviluppo dei Programmi

**Presentazione realizzata dal Prof. Andrea De Lucia**

# *Sviluppo dei programmi*



# *Analisi e Specifica*

- Definizione di *cosa fa* il programma
  - ☞ **Individuazione dei *dati* di *ingresso* e di *uscita*, della *precondizione* e della regola di trasformazione dei dati di ingresso in dati di uscita (*postcondizione*)**
- Buona norma utilizzare un ***dizionario dei dati*** da arricchire durante le varie fasi del ciclo di vita
  - ☞ **Una tabella il cui schema è:**
    - ◆ ***Attributo, Tipo, Descrizione***
    - ◆ **La descrizione serve a specificare meglio l'attributo e a descrivere il contesto in cui il dato viene usato**

# *Un esempio: conversione di misura ...*

- **Dati di ingresso: *cm***
- **Precondizione: *cm* >= 0**
- **Dati di uscita: <*feet*, *inch*>**
- **Postcondizione: *feet* è il massimo numero di piedi contenuti in *cm* e *inch* è il valore in pollici corrispondenti alla differenza tra *cm* e il valore in centimetri di *feet*. Il fattore di conversione tra pollici e centimetri è 2.54 e un piede equivale a 12 pollici**

| *Attributo* | *Tipo* | *Descrizione* |
|---|---|---|
| Cm | reale | Misura in centimetri in ingresso |
| feet | intero | Misura in piedi in uscita |
| inch | reale | Misura in pollici in uscita |

# *Progettazione*

- Definizione di *come* il programma effettua la trasformazione specificata;
- Progettazione dell'algoritmo per raffinamenti successivi (***stepwise refinement***) ...
- Decomposizione funzionale ... (*più avanti nel corso*)

- *Esempio*: Algoritmo per conversione di misura

**input *cm***
**calcola *feet* e *inch* a partire da *cm***
**output *feet* e *inch***

# *Stepwise refinement*

- Dettagliare il passo di calcolo;

**calcola *inch totali* (totalInches = cm / 2.54)**

**calcola *feet* a partire da *inch totali***
**(feet = parte_intera (totalinches / 12))**

**calcola *inch rimanenti* da inch totali e feet**
**(inch = totalInches - feet * 12)**

*… introduzione variabile locale di tipo reale* ***totalInches***

# *Modifica del dizionario dei dati*

- Il dizionario dei dati viene modificato ogni volta che viene introdotta una variabile nella progettazione dell'algoritmo;

| *Attributo* | *Tipo* | *Descrizione* |
|---|---|---|
| **Cm** | **reale** | **Misura in centimetri in ingresso** |
| **feet** | **intero** | **Misura in piedi in uscita** |
| **inch** | **reale** | **Misura in pollici in uscita** |
| **totalInches** | **reale** | **Misura in pollici dei centimetri in ingresso** |

# *Codifica e Verifica*

- Codifica dell'algoritmo nel linguaggio scelto
- Verifica del programma (individuazione degli errori)
  - ☞ **Scelta dei casi di prova**
  - ☞ **Esecuzione del programma**
  - ☞ **Verifica dei risultati rispetto ai *risultati attesi***
- *Utilizzo del software di base ...*

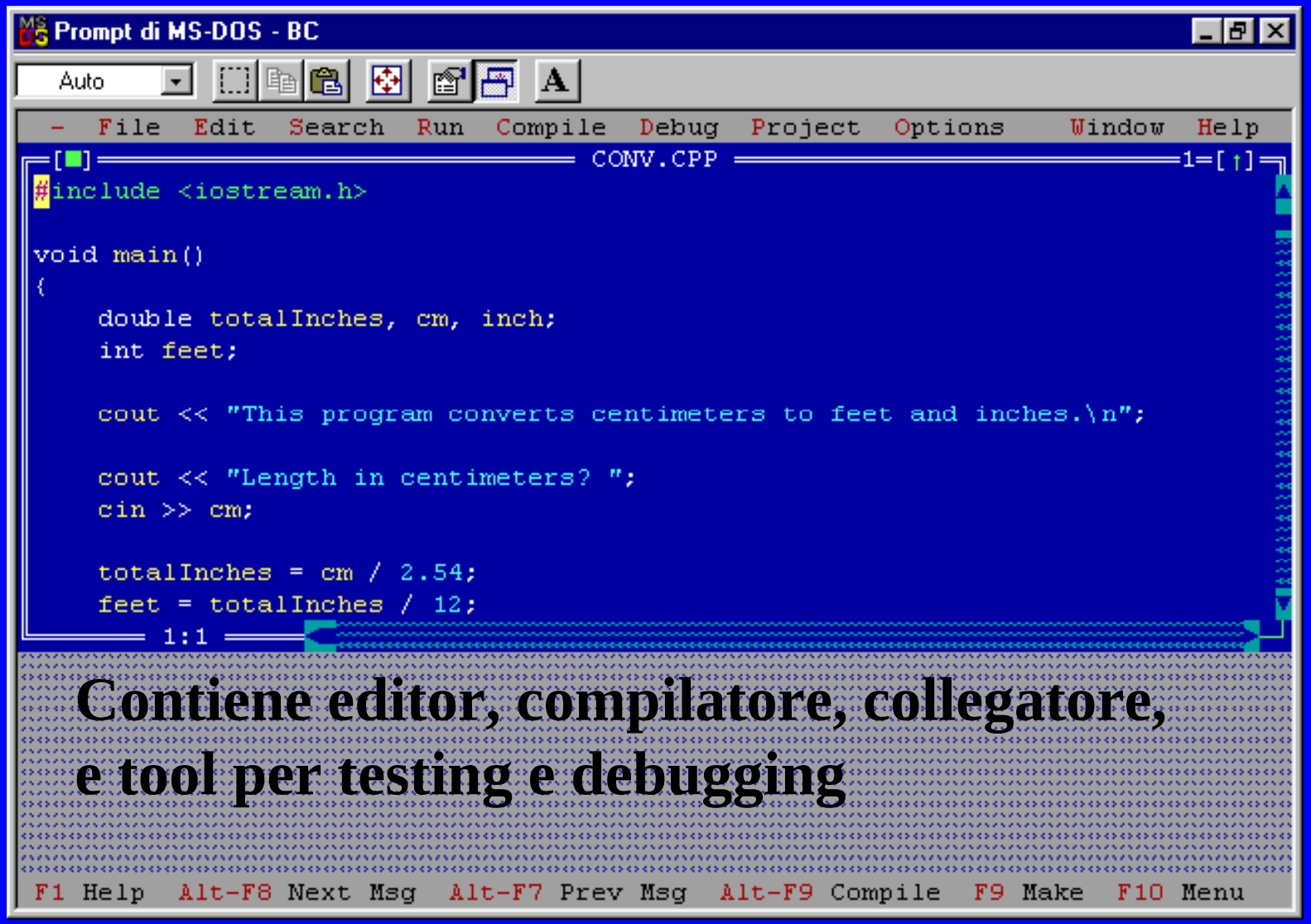
- Il nostro esempio ...

# *Codifica*

```
#include <stdio.h>
main()
{
    float totalInches, cm, inch;
    int feet;

    printf("This program converts centimeters to feet and inches.\n");
    printf("Length in centimeters? ");
    scanf("%f", & cm);
    totalInches = cm / 2.54;
    feet = totalInches / 12;
    inch = totalInches - feet * 12;
    printf("%f cm = %d ft + %f in.\n", cm, feet, inch);
}
```

# *Ambiente di programmazione integrato*




**Contiene editor, compilatore, collegatore, e tool per testing e debugging**

# *Ambiente non integrato (S.O. Linux)*

- Un editor di testo: *vi* oppure *pico*
  - vi *nomefile.c*
  - pico *nomefile.c*
- Un compilatore e un collegatore integrati: *gcc*
  - Compilazione: gcc -c *nomefile.c*
    - **prodotto il programma oggetto *nomefile.o***
  - Collegamento: gcc *nomefile.o* –o *nomefile*
    - **prodotto il programma eseguibile *nomefile***
  - Compilazione e collegamento:
    gcc *nomefile.c* –o *nomefile*
    - **prodotto il file *nomefile***

# *Scelta dei casi di prova e verifica*

- I casi di prova non vanno scelti “a caso”
  - **l’obiettivo è quello di una verifica il più possibile esaustiva … (*NB: verificare la correttezza di un qualsiasi programma è in generale indecidibile* ...)**
  - **… cercare di evitare i casi di prova ridondanti**
  - **i casi di prova e i risultati della verifica vanno documentati ...**

| ***Input*** **(cm)** | ***Output atteso*** **(feet, inch)** | ***Output effettivo*** **(feet, inch)** |
|---|---|---|
| **0** | **< 0 , 0 >** | **< 0 , 0 > --- PASS** |
| **69.85** | **< 2 , 3.5 >** | **< 2 , 3.5 > --- PASS** |

# *Errori nella stesura dei programmi*

- Errori sintattici (individuati in fase di compilazione)
  - ☞ A causa di errata scrittura o di mancanza di conoscenza delle regole del linguaggio (ad es. mancanza del “;” alla fine di una istruzione)
- Errori logici (più difficili da correggere)
  - ☞ A causa di una errata trascrizione durante la codifica dell’algoritmo (ad es. scrittura del numero 4 invece del numero 2)
  - ☞ *Si manifestano con **malfunzionamenti** durante l’esecuzione del programma*

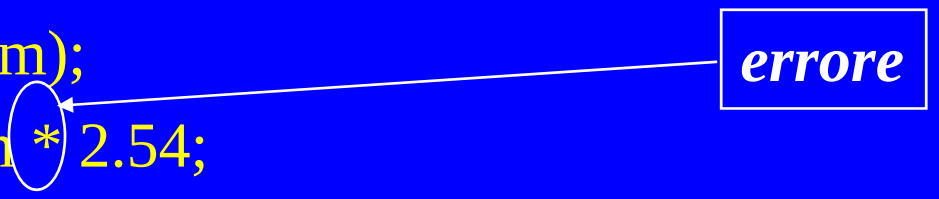
# *Errori logici*

- Possono essere di diverso tipo:
  - ☞Errori nella definizione del problema (fasi di analisi e specifica)
  - ☞Errori di progettazione (es. nella definizione/scelta dell'algoritmo)
  - ☞Errori nella stesura (*codifica*) dei programmi
- Causano malfunzionamenti del programma
- Più alta è la fase in cui avviene l'errore più è difficile correggerlo:
  - ☞propagazione dell'errore alle fasi più basse
  - ☞correzione degli errori: ***debugging***

# *Programma con errore (bug)*

```
#include <stdio.h>
main()
{
    float totalInches, cm, inch;
    int feet;

    printf("This program converts centimeters to feet and inches.\n");
    printf("Length in centimeters? ");
    scanf("%f", & cm);
    totalInches = cm * 2.54;
    feet = totalInches / 12;
    inch = totalInches - feet * 12;
    printf("%f cm = %d ft + %f in.\n", cm, feet, inch);
}
```



# *Verifica e Debugging*

| *Input* (cm) | *Output atteso* (feet, inch) | *Output effettivo* (feet, inch) | |
|---|---|---|---|
| 0 | < 0 , 0 > | < 0 , 0 > | --- OK |
| 69.85 | < 2 , 3.5 > | < 14 , 9.419 > | --- FAIL |

- Ricerca dell'errore nel programma
  - **Ispezione del codice ed esecuzione simulata**
  - **Esecuzione e controllo dello stato del programma nei punti "sospetti" e nei punti critici**
    - ***Stato del programma: l'insieme dei valori assegnati alle diverse variabili prima dell'esecuzione di una data istruzione***
  - **Strumenti automatici negli ambienti di programmazione**
- Rimozione dell'errore e nuova verifica con il caso di prova su cui è stato rilevato il malfunzionamento

# *Controllo precondizione*

- Cosa succede se al programma precedente viene dato in ingresso un valore negativo per ***cm*** ?
  - ☞**valori di uscita indefiniti ...**
  - ☞**preferibile evitare queste situazioni ...**
- L'istruzione di selezione può essere utilizzata per controllare la precondizione, prima dell'elaborazione dei dati di ingresso
- durante la verifica del programma usare anche casi di prova che violano la precondizione ...

# *... rivisitiamo l'algoritmo per la conversione di misura ...*

**input *cm***

**if(cm >= 0) {  /* test precondizione */**

**calcola *feet* e *inch* a partire da *cm***

**/* da raffinare */**

**output *feet* e *inch* }**

**else output "errore: valore di ingresso negativo"**

# *Conversione da centimetri a piedi e pollici*

```
#include <stdio.h>
main() {
    float totalInches, cm, inch;
    int feet;

    printf("Conversion of centimeters to feet and inches.\n“);
    printf("Length in centimeters? “);
    scanf("%f", & cm);   /* Input cm */
    if (cm >= 0) {    /* test precondizione */
        totalInches = cm / 2.54;
        feet = totalInches / 12;
        inch = totalInches - feet * 12;
        printf("%f cm = %d ft + %f in.\n", cm, feet, inch);}
    else printf(“errore, valore di ingresso negativo \n”);
}
```

# *... verifica*

| *Input* (cm) | *Output atteso* (feet, inch) | *Output effettivo* (feet, inch) |
|---|---|---|
| 0 | < 0 , 0 > | < 0 , 0 > --- PASS |
| 69.85 | < 2 , 3.5 > | < 2 , 3.5 > --- PASS |
| -9.7 | *msg errore* | *msg errore* --- PASS |

# *Un altro esempio: Il fattoriale*

- Analisi e specifica
  - **Dati di ingresso: *n***
  - **Dati di uscita: *fatt***
  - **Precondizione: *n* >= 0**
  - **Postcondizione: se n = 0 allora fatt = 1**
    **altrimenti *fatt* = *n* * (*n*-1) * … * 2 * 1**
- Progettazione

```
input n
if (n  >= 0) {      /* test  precondizione */
        calcola il fattoriale di n e assegnalo a fatt
        output fatt
      }
else output "errore: valore di ingresso negativo"
```

# *Stepwise refinement*

- Raffinamento del passo:

  ***calcola il fattoriale di n e assegnalo a fatt***

  ***fatt* = 1**

  **for (*i* = 2 ; i <= *n* ; i++)**

  ***fatt = fatt * i***

| *Attributo* | *Tipo* | *Descrizione* |
|---|---|---|
| **n** | **intero** | **Numero in ingresso** |
| **fatt** | **intero** | **Fattoriale del numero in uscita** |
| **i** | **intero** | **Indice del ciclo** |

# *Programma fattoriale*

```
#include <stdio.h>

main()
{
   int n, fatt, i;

   printf("Inserisci un intero positivo: ");
   scanf("%d", &n);

   if (n >= 0) {        /* test precondizione */
         fatt = 1;
         for(i =2; i <= n; i++)
               fatt = fatt * i;
         printf("Fattoriale di %d : %d \n", n, fatt);
      }
    else printf("errore, valore di ingresso negativo \n");
}
```

# *... verifica*

| *Input* (n) | *Output atteso* (fatt) | *Output effettivo* (fatt) | |
|---|---|---|---|
| 0 | 1 | 1 | --- PASS |
| 4 | 24 | 24 | --- PASS |
| -2 | *msg errore* | *msg errore* | --- PASS |

# *Forzare la precondizione ad essere verificata*

- Nell'esempio precedente il programma si arresta con un messaggio di errore nel caso in cui la precondizione non è soddisfatta

  ☞ **input x**
  **if (*precondizione(x)*) *elabora***
  **else *messaggio di errore***

- In alcuni casi potrebbe essere preferibile ripetere l'immissione dell'input (***è una scelta di progetto***)

  ☞ **do input x**
  **while (!*precondizione(x)*)**
  ***elaborazione***

# *… Forzare la precondizione ...*

```
#include <stdio.h>

main()
{
   int n, fatt, i;

   do {
         printf("Inserisci un intero positivo: ");
         scanf("%d", &n);
   }while(n < 0);        /* test precondizione */

   fatt = 1;
   for(i =2; i <= n; i++)
           fatt = fatt * i;
   printf("Fattoriale di %d : % d \n", n, fatt);
}
```

***Ripete l'input finché la precondizione non è verificata***

# *Differenziare la prima immissione ...*

- In alcuni casi è preferibile visualizzare un messaggio di errore nel caso di input che violano la precondizione
- Ovviamente tale messaggio non deve comparire alla prima immissione: necessità di trattare la prima immissione fuori dal ciclo

```
input x;
while (!precondizione(x)) {
        messaggio di errore;
        input x; }
elaborazione
```

```
done = FALSE;
while (! done) {
        input x;
        if(precondizione(x))
                done = TRUE;
        else messaggio di errore;}
elaborazione
```

# *… prima immissione ...*

```
#include <stdio.h>

main()
{
   int n, fatt, i;
   printf(“Inserisci un intero positivo: ”);
   scanf(“%d”, &n);
   while(n < 0) {
         printf(“Errore: numero negativo ! \n”);
         printf(“Inserisci un intero positivo: ”);
         scanf(“%d”, &n); }
   fatt = 1;
   for(i =2; i <= n; i++)
           fatt = fatt * i;
   printf(“Fattoriale di %d : %d \n”, n, fatt);
}
```

# *… prima immissione*

```
#include <stdio.h>

void main()
{
   int n, fatt, i;
   int done = 0;
   while(!done) {
         printf(“Inserisci un intero positivo: ”);
         scanf(“%d”, &n);
         if(n >= 0 )     done = 1;
         else printf(“Errore: numero negativo ! \n”);   }
   fatt = 1;
   for(i =2; i <= n; i++)
           fatt = fatt * i;
   printf(“Fattoriale di %d : %d \n”, n, fatt);
}
```