Problema: Calcolo del valore assoluto di un numero intero

Vediamo alcuni esempi di specifica di algoritmi.

Specifica:

Stato iniziale: $\{$ numero $\leadsto$ A $\}$

Stato finale: $\{$ risultato $\leadsto$ $|A|$ $\}$

dove A indica un (generico) valore intero, diverso da zero.

Algoritmo:

```
if (numero > 0)
        risultato =  numero;
    else risultato =  - numero;
```

Problema: scambiare il valore di due variabili

Specifica:

Stato iniziale: $\{ \mathsf{x} \rightsquigarrow \mathsf{A}, \mathsf{y} \rightsquigarrow \mathsf{B} \}$
Stato finale: $\{ \mathsf{x} \rightsquigarrow \mathsf{B}, \mathsf{y} \rightsquigarrow \mathsf{A} \}$

Algoritmo:

```
temp = x;
x = y;
y = temp;
```

Problema: scambiare il valore di due variabili (cont.)

- Si noti l'utilizzo di un nome simbolico aggiuntivo (temp): è essenziale per poter effettuare correttamente lo scambio tra i valori associati ai due interi (a causa del carattere distruttivo dell'assegnamento e della sequenzialità dell'esecuzione).
- È facile verificare che una sequenza del tipo

```
x = y;
y = x;
```

non consente, in generale, di scambiare i valori associati a x e y.

Esempio:

Stato iniziale		Stato Intermedio
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$x = y;$	$\{ x \rightsquigarrow 20, y \rightsquigarrow 20\}$
Stato intermedio		**Stato Finale**
$\{ x \rightsquigarrow 20, y \rightsquigarrow 20\}$	$y = x$	$\{ x \rightsquigarrow 20, y \rightsquigarrow 20\}$

Problema: ordinare due interi positivi

Specifica:

Stato iniziale: $\{$ num1 $\leadsto$ A, num2 $\leadsto$ B $\}$
Stato finale: $\{$ num1 $\leadsto$ *max*(A,B), num2 $\leadsto$ *min(A,B)* $\}$

Algoritmo:

```
if (num1 < num2)
    {
        temp = num1;
        num1 = num2;
        num2 = temp;
    }
```

ATTENZIONE: non basta stabilire il massimo e il minimo; occorre anche che, alla fine, il massimo sia associato a num1 e il minimo a num2.

Problema: Elevamento a potenza

Specifica:

Stato iniziale: $\{$ base $\rightsquigarrow$ A, esponente $\rightsquigarrow$ B $\}$ con A$>$0 e B $\geq$0
Stato finale: $\{$ risultato $\rightsquigarrow$ $A^B\}$

- L'algoritmo si basa sulla seguente, ben nota, definizione di elevamento a potenza.

$$
\begin{aligned}
A^0 &= 1 \\
A^B &= \underbrace{A \times A \times \ldots \times A}_{B \text{ volte}} \qquad (\text{se } B > 0)
\end{aligned}
$$

Algoritmo:

```
risultato =  1;
while (esponente > 0)
{
    risultato =  risultato * base;
    esponente =  esponente - 1;
}
```

- Nell'algoritmo, si ipotizza che i valori iniziali di base ed esponente soddisfino i requisiti della specifica.

Input/Output

- Nei problemi visti fino ad ora non ci siamo preoccupati di acquisire i dati iniziali né di produrre in uscita i risultati finali.

  Esempio: Ordinare due valori interi acquisiti in input e stampare i valori ordinati.

  Algoritmo:

```
scanf("%d",&num1);
scanf("%d",&num2);
if num1 < num2
{
      temp =  num1;
      num1 =  num2;
      num2 =  temp;
}
printf("%d",num1);
printf("%d",num2);
```

Moltiplicazione egizia

Primo algoritmo documentato (Papiro di Ahmes, ca 1650 a.C)

Stato iniziale: $\{a \rightsquigarrow A,\ b \rightsquigarrow B\}$ con $A,B > 0$

Stato finale: $\{p \rightsquigarrow A*B\}$

Algoritmo:

```
scanf("%d",&a); scanf("%d",&b); c = 0;
while (a > 0)
{
    if (a è dispari)
        {c = c + b;}
    a = a/2;
    b = b*2; }
printf("%d",c);
```

nota che la divisione è intera

a è dispari può essere verificato con la condizione a%2 == 1

Moltiplicazione egizia (cont.)

Algoritmo:

```
scanf("%d",&a); scanf("%d",&b); c = 0
while (a > 0)
{
    if (a è dispari)
        {c = c + b;}
    a = a/2;
    b = b*2; }
printf("%d",c);
```

a pari $\Rightarrow$ a*b = (a/2)*2b

$$a * b = (a/2) * 2b = (a-1+1)/2 * 2b$$

a dispari $\Rightarrow$

$$(a-1+1)/2 * 2b = ((a-1)/2 + 1/2) * 2b$$

$$((a-1)/2 + 1/2) * 2b = ((a-1)/2) * 2b + b$$

Moltiplicazione egizia (cont.)

Algoritmo:

```
scanf("%d",&a); scanf("%d",&b); c = 0
while (a > 0){
    if (a è dispari) {c = c + b;}
    a = a/2;
    b = b*2;}
printf("%d",c);
```

- $\{a = 16, b = 26, c = 0\}$, $\{a = 8, b = 52, c = 0\}$, $\{a = 4, b = 104, c = 0\}$, $\{a = 2, b = 208, c = 0\}$, $\{a = 1, b = 416, c = 0\}$ $\{a = 0, b = 832, c = 416\}$
- $\{a = 15, b = 26, c = 0\}$, $\{a = 7, b = 52, c = 26\}$, $\{a = 3, b = 104, c = 78\}$, $\{a = 1, b = 208, c = 182\}$ $\{a = 0, b = 416, c = 390\}$

Problema: Calcolare quoziente e resto della divisione tra due numeri naturali non nulli.

Specifica:

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B\}$ con $A,B > 0$

Stato finale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B,\ q \rightsquigarrow Q,\ r \rightsquigarrow R\}$

con $A = Q \cdot B + R$ e $0 \leq R < B$

Supponiamo che l'esecutore non sappia eseguire direttamente la divisione, ma solo la somma e la sottrazione.

Algoritmo:

```
q = 0;
r = x;
while (r ≥ y)
{
    q = q + 1;
    r = r - y;
}
```

Calcolare il MCD con l'algoritmo di Euclide

- Dati due naturali non nulli si calcola il MCD utilizzando le seguenti proprietà:

$$
\begin{aligned}
MCD(x,x) &= x \\
MCD(x,y) &= MCD(x-y,y) \qquad \text{se } x > y \\
MCD(x,y) &= MCD(x,y-x) \qquad \text{se } y > x
\end{aligned}
$$

Specifica:

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B\}$ con $A,B > 0$

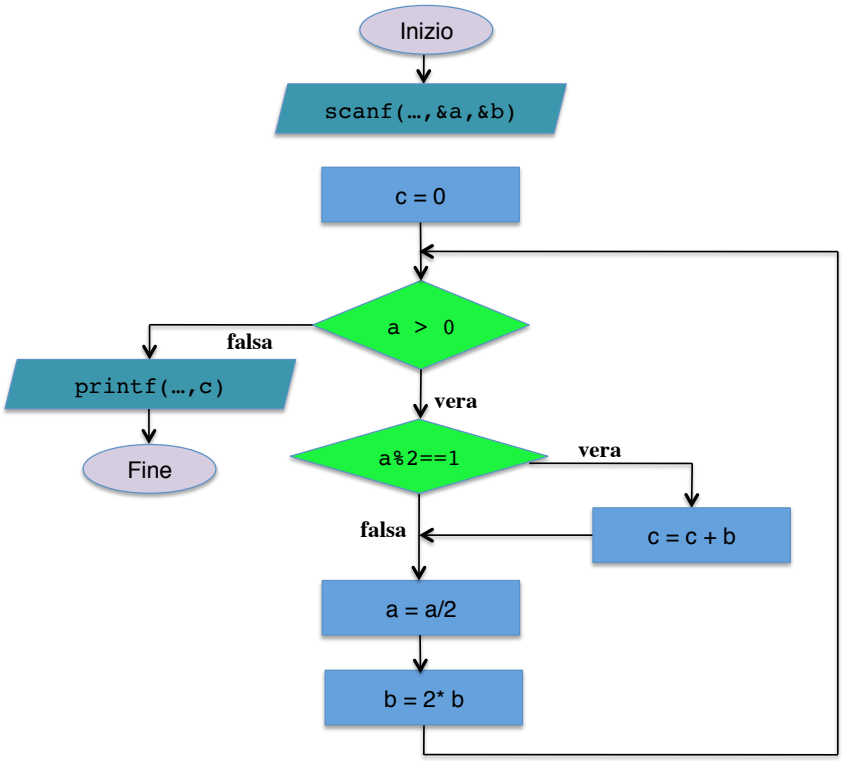
Stato finale: $\{x \rightsquigarrow MCD(A,B)\}$

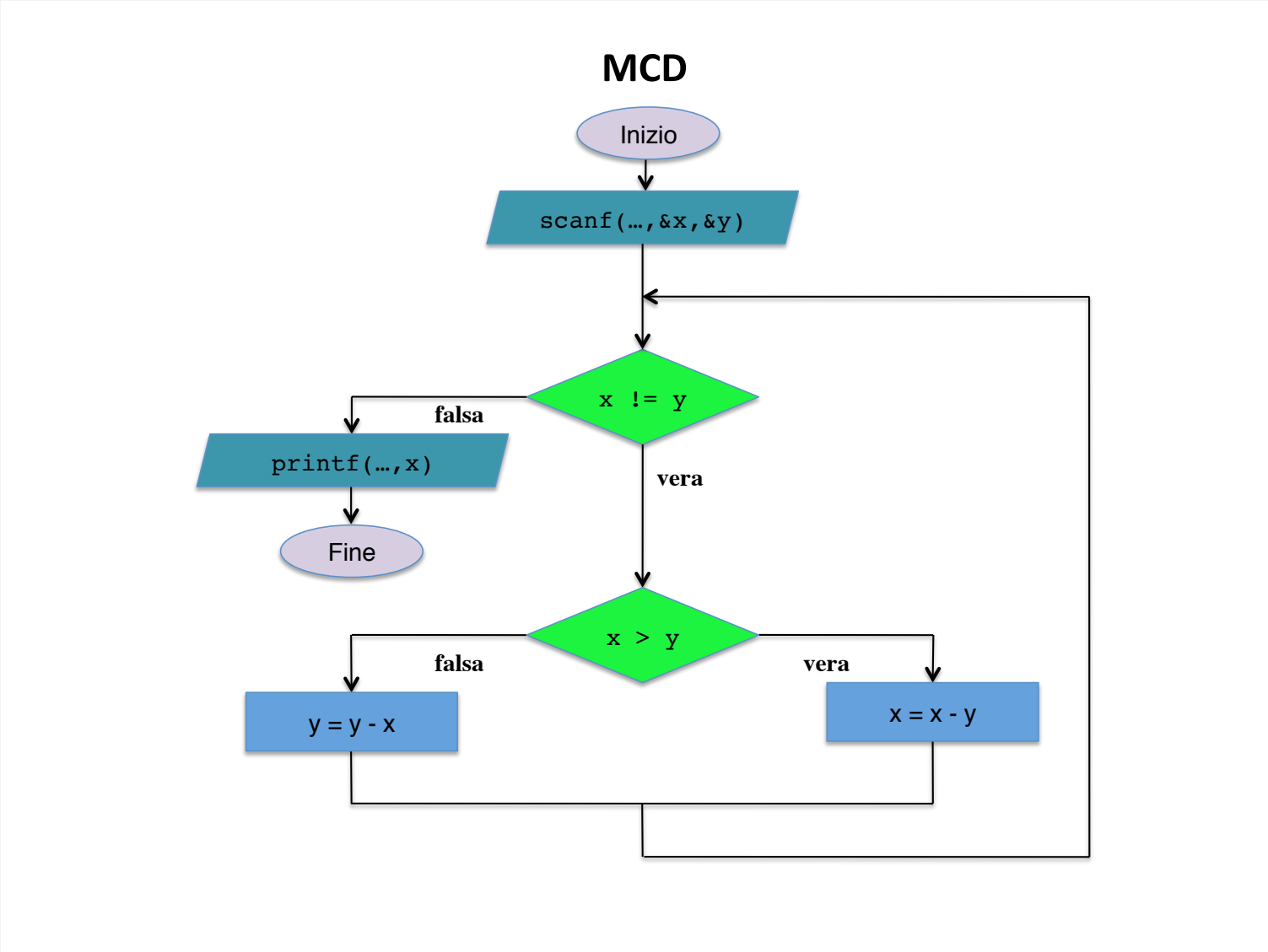
Calcolare il MCD con l'algoritmo di Euclide (cont.)

Algoritmo:

```
while (x != y)
{
    if (x > y)
        x = x - y;
    else
        y = y - x;
}
```

Moltiplicazione Egizia




MCD
Inizio
scanf(…,&x,&y)
x != y
falsa
printf(…,x)
Fine
vera
x > y
falsa
y = y - x
vera
x = x - y

Tipi di dato

- per tipo di dato si intende un insieme di valori e di operazioni che si possono ad essi applicare.
- Alcuni tipi di dato (numeri interi, caratteri, ecc.), detti tipi primitivi sono forniti direttamente dal linguaggio.
- Si possono tuttavia introdurre nuovi tipi di dato adatti al problema da affrontare.
- Nel caso servano dati aggregati (ad es. una data, un vettore, ecc.), si può infine ricorrere ai tipi di dato strutturati. Si deve poter accedere ai singoli elementi.

Gnocchi in brodo (Pellegrino Artusi, 1891)

... è una minestra da farsene onore; ma se non volete consumare appositamente per lei un petto di pollastra o di cappone, aspettate che vi capiti d'occasione. Cuocete nell'acqua, o meglio a vapore, grammi 200 di patate grosse e farinacee e passatele per istaccio. A queste unite il petto di pollo lesso tritato finissimo colla lunetta, grammi 40 di parmigiano grattato, due rossi d'uovo, sale quanto basta e odore di noce moscata. Mescolate e versate il composto sulla spianatoia sopra a grammi 30 o 40 (che tanti devono bastare) di farina per legarlo, e poterlo tirare a bastoncini grossi quanto il dito mignolo. Tagliate questi a tocchetti e gettateli nel brodo bollente ove una cottura di cinque o sei minuti sarà sufficiente. Questa dose potrà bastare per sette od otto persone. Se il petto di pollo è grosso, due soli rossi non saranno sufficienti...

La ricetta è piacevole da leggere, ma non è facile da seguire.

Gnocchi in brodo (cont.)

DOSE: 7-8 persone
INGREDIENTI: 200 di patate grosse e farinacee, 40 g parmigiano, 2 rossi di uovo, sale e noce moscata q.b., un petto di pollo, 30-40 g farina

- Cuocete nell'acqua o a vapore le patate
- passatele al setaccio
- unite pollo lesso tritato, parmigiano, uova, sale e noce moscata
- versate il composto sulla spianatoia, mescolandolo con la farina
- tirare il composto a bastoncini dello spessore del dito mignolo
- tagliare i bastoncini a tocchetti
- gettare i tocchetti nell'acqua bollente e cuocerli per 5-6 minuti.

La ricetta non è altrettanto piacevole da leggere, ma è facile da seguire, anche se presenta delle ambiguità.

Introduzione al linguaggio C

- Abbiamo già visto come un programma non sia altro che un algoritmo codificato in un linguaggio di programmazione e che
- I linguaggi di programmazione ad alto livello si prestano a codificare algoritmi, pur rimanendo comprensibili, ponendosi quindi ad un livello intermedio tra il linguaggio naturale e il linguaggio macchina
- I compilatori sono programmi che traducono programmi scritti nel linguaggio di più alto livello in programmi equivalenti nel linguaggio macchina.

- Vedremo il cosidetto ANSI C (standard del 1989, con successive aggiunte)
- Il primo programma C: ciao mondo

```
#include <stdio.h>
int main()
  /* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

- Questo programma stampa sullo schermo una riga di testo:

```
Ciao mondo!
>
```

- Vediamo in dettaglio ogni riga del programma.

```
/* Stampa un messaggio sullo schermo. */
```

- testo racchiuso tra "/*" e "*/" è un commento
- i commenti sono a "uso umano", cioè servono a chi scrive o legge il programma, per renderlo più comprensibile
- il compilatore ignora i commenti
- attenzione a non dimenticare di chiudere i commenti con */, altrimenti tutto il resto del programma viene ignorato

`int main()`

- è una parte presente in tutti i programmi C
- le parentesi "`(`" e "`)`" dopo main indicano che main è una funzione
- i programmi C sono composti da una o più funzioni, tra le quali ci deve essere la funzione `main`, che è quella principale
  $\Longrightarrow$ `main` è una funzione speciale, perché l'esecuzione del programma comincia l'esecuzione all'inizio di `main`
- la parentesi "{" apre il corpo della funzione e "}" lo chiude
  - la coppia di parentesi e la parte racchiusa da esse costituiscono un blocco
  - il corpo della funzione contiene le istruzioni (e dichiarazioni) che costituiscono la funzione

```
printf("Ciao mondo!\n");
```

- è un'istruzione semplice (ordina al computer di eseguire un'azione) in questo caso visualizzare (stampare) sullo schermo la sequenza di caratteri tra apici
- ogni istruzione semplice deve terminare con ";"
- oltre alle istruzioni semplici, esistono anche istruzioni composte (che non devono necessariamente terminare con ";")
- la parte racchiusa in una coppia di doppi apici è una stringa (di caratteri)
- "\n" non viene visualizzato sullo schermo, ma provoca la stampa di un carattere di fine riga
  - "\" è un carattere di escape e, insieme al carattere che lo segue, assume un significato particolare (sequenza di escape)
- in realtà anche `printf` è una funzione, e l'istruzione di sopra è un'attivazione di funzione (le vedremo più avanti)

`#include <stdio.h>`

- è una direttiva di compilazione
- viene interpretata dal compilatore durante la compilazione
- la direttiva "`#include`" dice al compilatore di includere il contenuto di un file nel punto corrente
- `<stdio.h>` è un file che contiene i riferimenti alla libreria standard di input/output (dove è definita la funzione di stampa `printf`)
- il linguaggio C non prevede istruzioni esplicite di input/output. Queste operazioni sono definite tramite funzioni nella libreria standard di input/output.

Note:

- è importante distinguere i caratteri maiuscoli da quelli minuscoli `Main`, `MAIN`, `Printf`, `PRINTF` non andrebbero bene
- si è usata l'indentazione per mettere in evidenza la struttura del programma ▸)

Alcune varianti del programma

```
#include <stdio.h>
int main()
  /* Stampa un messaggio sullo schermo. */
{
  printf("Ciao");
  printf(" mondo!\n");
  return 0;
}
```

- produce lo stesso effetto del programma precedente, ma la seconda invocazione di `printf` incomincia a stampare dal punto in cui aveva smesso la prima. Cosa viene stampato se usiamo

```
printf("Ciao");
printf("mondo!\n");
```

```
printf("Ciao\n");
printf("mondo!\n");
```

Un altro programma: area di un rettangolo

```
#include <stdio.h>

int main() {
  int base;
  int altezza;
  int area;

  base = 3;
  altezza = 4;
  area = base * altezza;

  printf("Area: %d\n", area);
  return 0;
}
```

Quando viene eseguito stampa:

```
Area: 12
>
```

Le variabili (di programma)

Lo stato rappresenta il contenuto (modificabile) della memoria.

Una posizione della memoria, destinata a contenere dati, si identifica con una variabile Le variabili dunque rappresentano, nei programmi, le associazioni (modificabili) dello stato
$\Longrightarrow$ cf. $x \rightsquigarrow val$ nello pseudo-linguaggio

Una variabile è caratterizzata dalle seguenti proprietà:

1. nome: serve a identificarla — esempio: `area`
2. valore: valore associato allo stato corrente — Esempio: 4 (può cambiare durante l'esecuzione)
3. tipo: specifica l'insieme dei possibili valori assunti durante l'esecuzione — Esempio: `int` (numeri interi) e implicitamente le operazioni possibili.
4. indirizzo: della cella di memoria a partire dal quale è memorizzato il valore.

Nome, tipo e indirizzo non possono cambiare durante l'esecuzione.

Le variabili (cont.)

- Il nome di una variabile è un identificatore C
  $\Longrightarrow$ sequenza di lettere, cifre, e _ che inizia con una lettera o con _
  Esempio: `Numero_elementi`, `x1`, ma non `1_posto`
  - può avere lunghezza qualsiasi, ma solo i primi 31 caratteri sono significativi
  - lettere minuscole e maiuscole sono considerate distinte
- A ogni variabile è associata una cella di memoria o più celle consecutive, a seconda del suo tipo. Il suo indirizzo è quello della prima cella.
- Analogia con un cassetto etichettato in un mobile
  - nome $\Longrightarrow$ etichetta
  - valore $\Longrightarrow$ oggetto che c'è nel cassettos
  - tipo $\Longrightarrow$ capienza (che tipo di oggetto ci metto dentro)
  - indirizzo $\Longrightarrow$ posizione nel mobile
  - **N.B.** non tutte le variabili sono denotate da un identificatore (strut. din.) e non tutti gli identificatori sono identificatori di variabile (ad es. funzioni, tipi, parole riservate, ...)

Area del rettangolo

- `int base;` — è una dichiarazione di variabile
  - viene creata il cassetto e inserito nel mobile
  - ha tipo `int` $\Longrightarrow$ può contenere interi
  - ha nome `base`
  - ha un indirizzo (posizione nel mobile), che è quello della cella di memoria associata alla variabile
  - ha un valore iniziale, che però non è significativo (è casuale)
    $\Longrightarrow$ il cassetto viene creato pieno, però con un oggetto scelto a caso, ovvero
    $\Longrightarrow$ l'associazione nello stato è del tipo nome $\rightsquigarrow$?

- `int altezza;`
  `int area;`
  $\Longrightarrow$ come per `base`

Variabili numeriche

Variabili intere

- per dichiarare variabili intere si può usare il tipo `int`
- i valori di tipo `int` sono rappresentati in C con almeno 16 bit
- il numero effettivo di bit dipende dal compilatore
  Esempio: 32 bit per il compilatore gcc (usato in ambiente Unix)
- in C esistono altri tipi per variabili intere (`short`, `long`) — li vedremo più avanti

Variabili reali

- per dichiarare variabili reali si può usare il tipo `float`
  Esempio: `float temperatura;`

Area del rettangolo

`base = 3;`
è un'istruzione di assegnamento (come nello pseudo-linguaggio)

- in C l'operatore di assegnamento è denotato dal simbolo "="
- come già sappiamo, l'effetto è di modificare una associazione nello stato
  $\Longrightarrow$ in questo caso il valore 3 viene associato a base, come?
  $\Longrightarrow$ il nuovo valore viene scritto nella spazio associato alla variabile
- a questo punto la variabile `base` ha un valore significativo
  $\Longrightarrow$ da base $\rightsquigarrow$? a base $\rightsquigarrow$ 3

`altezza = 4;` $\Longrightarrow$ come sopra

`area = base * altezza;`
a destra di "=" possono comparire espressioni $\Longrightarrow$ il valore assegnato è quello dell'espressione calcolata nello stato corrente

- una variabile all'interno di una espressione sta per il valore ad essa associato in quel momento (cf. pseudo-linguaggio)

Nota: operatori aritmetici tra interi del C sono: `+`, `-`, `*`, `/`, `%`, ...

Area del rettangolo

```
printf("Area: %d\n", area);
```

- è un'istruzione di stampa
- il primo argomento è la stringa di formato che può contenere specificatori di formato
- lo specificatore di formato `%d` indica che deve essere stampato un intero in notazione decimale (`d` per decimal)
- ad ogni specificatore di formato nella stringa deve corrispondere un valore che deve seguire la stringa di formato tra gli argomenti di `printf`

  Esempio: `printf(" %d %d··· %d", i1, i2, ..., in );`
- nel caso di `printf("Ciao mondo!\n");` la stringa di formato non conteneva specificatori e quindi non vi erano altri argomenti.

Struttura dei programmi C

- Nel semplice programma che abbiamo appena analizzato possiamo già vedere la struttura generale di un programma C.

```
/* DIRETTIVE DI COMPILAZIONE */
#include <stdio.h>
 main() {

 /* PARTE DICHIARATIVA */
  int base;
  int altezza;
  int area;

 /* PARTE ESECUTIVA */
  base = 3;
  altezza = 4;
  area = base * altezza;
  printf("Area: %d\n", area);
}
```

Un programma C deve contenere nell'ordine:

- Una parte contenente direttive per il compilatore. Nel nostro programma la direttiva

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

- l'identificatore predefinito `main` seguito dalle parentesi `()`.
- due parti racchiuse tra parentesi graffe
  - la parte dichiarativa. Nell'esempio:

```
int base;
int altezza;
int area;
```

  - la parte esecutiva. Nell'esempio:

```
base = 3;
altezza = 4;
area = base * altezza;
printf("Area: %d\n", area);
```

La parte dichiarativa

- Posta prima della codifica dell'algoritmo, obbliga a dichiarare i nomi simbolici che saranno presenti nello stato e di cui farà uso nella parte esecutiva. Prima si dichiara e poi si usa. Contiene i seguenti elementi:
  - la sezione delle dichiarazioni di costanti
  - la sezione delle dichiarazioni di variabili;
- Le dichiarazioni: rendono più pesante la fase di costruzione dei programmi, ma consentono di individuare e segnalare errori in fase di compilazione.

  Esempio:

```
int x;
int alfa;
alfa = 0;
x=alfa;
alba=alfa+1;
```

- Il compilatore vede `alba` come variabile non dichiarata.

Dichiarazioni di variabili

- Abbiamo già visto esempi di dichiarazioni di variabili.

```
float x;
int base;
int altezza;
```

- Ad ogni variabile viene attribuito, al momento della dichiarazione, un tipo

  $\Longrightarrow$ specifica l'insieme dei valori che la variabile può assumere

- La dichiarazione può anche attribuire un valore iniziale alla variabile (inizializzazione)

```
int x = 0;
```

- Variabili dello stesso tipo possono essere dichiarate contemporaneamente

```
int base, altezza, area;
```

(ma inizializzate singolarmente)

Esempio: `int x, y, z=0;` solo `z` è inizializzata a `0`.

Dichiarazioni di costanti (variabili *read-only*)

- Una dichiarazione di costante[2] crea un'associazione non modificabile $\Longrightarrow$ associa in modo permanente un valore ad un identificatore. Di solito si estende il campo dei valori rappresentabili.

  Esempio:

  ```
  const float PiGreco=3.14;
  const int N=100;
  ```

- L'associazione tra il nome `PiGreco` ed il valore `3.14` non può essere modificata durante l'esecuzione.
- Come per le dichiarazioni di variabili, più costanti dello stesso tipo possono essere dichiarate insieme

  Esempio:

  ```
  const float PiGreco=3.14, e=2.718;
  const int N=100, M=200;
  ```

- **N.B.** cosa succede se si modifica una costante non è specificato dallo standard ANSI C, dipende dal compilatore.

[2] cf. uso diverso di `#define`, vedi pagina successiva.

Digressione sulle costanti: la direttiva `#define`

- La dichiarazione di costante, ad esempio

  ```
  const int Nmax = 10;
  ```

  causa l'allocazione di memoria (si tratta di una dichiarazione di variabile *read only*)
- C'è un altro modo per ottenere un identificatore costante, che utilizza la direttiva **#define**.

  ```
  #define Nmax 10
  ```
- **#define** è una direttiva di compilazione
- dice al compilatore di sostituire ogni occorrenza di `Nmax` con `10` prima di compilare il programma
- a differenza di **const** non alloca memoria

Uso di costanti

- Con la dichiarazione `const float PiGreco=3.14;`
  l'istruzione
  `AreaCerchio=PiGreco*RaggioCerchio*RaggioCerchio;`
  è equivalente a
  `AreaCerchio=3.14*RaggioCerchio*RaggioCerchio`
- Maggiore leggibilità dei programmi, dovuta all'uso di nomi simbolici
- Maggiore adattabilità dei programmi che usano costanti

  Esempio:
  Per aumentare la precisione, basta cambiare la dichiarazione in
  `const float PiGreco = 3.1415;`

  Senza l'uso della costante si dovrebbero rimpiazzare nel codice tutte le occorrenze di `3.14` in `3.1415` ...
  NOTA: anche con la `#define` abbiamo gli stessi vantaggi

Area di un rettangolo di dimensioni lette da tastiera

```
#include <stdio.h>

int main()
{
  int base, altezza, area;

  printf("Immetti base del rettangolo e premi INVIO\n");
  scanf("%d", &base);
  printf("Immetti altezza del rettangolo e premi INVIO\n");
  scanf("%d", &altezza);

  area = base * altezza;

  printf("Area: %d\n", area);
  return 0;
}
```

Nuova istruzione: `scanf("%d", &base);`

- `scanf` è la funzione duale di `printf`
- legge da input (tastiera) un valore intero e lo assegna alla variabile `base`
- `"%d"` è la stringa di controllo del formato (in questo caso viene letto un intero in formato decimale)
- "`&`" è l'operatore di indirizzo
  - `&base` indica (l'indirizzo del)la locazione di memoria associata a `base`
  - `scanf` memorizza in tale locazione il valore letto
- quando viene eseguita `scanf` il programma si mette in attesa che l'utente immetta un valore. Quando l'utente digita Invio
  1. la sequenza di caratteri immessa viene convertita in un intero (formato `%d`) e
  2. l'intero ottenuto viene assegnato alla variabile `base` (viene cioè scritto nella/e cella/e di memoria a partire dall'indirizzo passato a `scanf`)

N.B. il precedente valore della variabile `base` va perduto

Esempio di esecuzione

- Vediamo cosa avviene durante l'esecuzione (indichiamo in rosso ciò che l'utente digita e in particolare con $\hookleftarrow$ il tasto Invio).

```
Immetti base del rettangolo e premi INVIO
5 ↩
Immetti altezza del rettangolo e premi INVIO
4 ↩
Area: 20
```