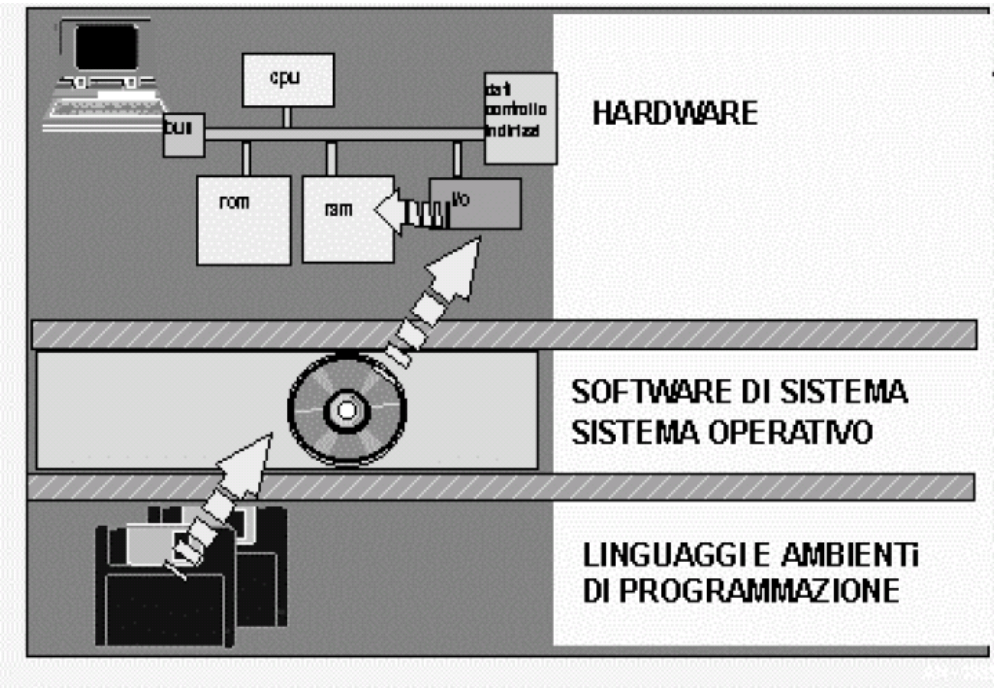
Livelli di astrazione

- Si può risolvere un problema senza prima fissare un insieme di "azioni", di "mosse elementari" possibili per l'elaboratore?
  - No, bisogna prima scendere al livello della macchina, per conoscerne le caratteristiche
  - Ma quale macchina?
- È corretto impostare la soluzione a partire da tali "mosse elementari"?
  - SI, per risolvere il problema con efficienza
  - NO, se la macchina di partenza ha mosse di livello troppo basso
- Bisogna risalire ad un più adeguato livello di ASTRAZIONE (processo di aggregazione di informazioni e dati e di sintesi di modelli concettuali che ne enucleano le proprietà rilevanti escludendo i dettagli inessenziali)

Il ruolo dell'astrazione



Per la codifica di un algoritmo...

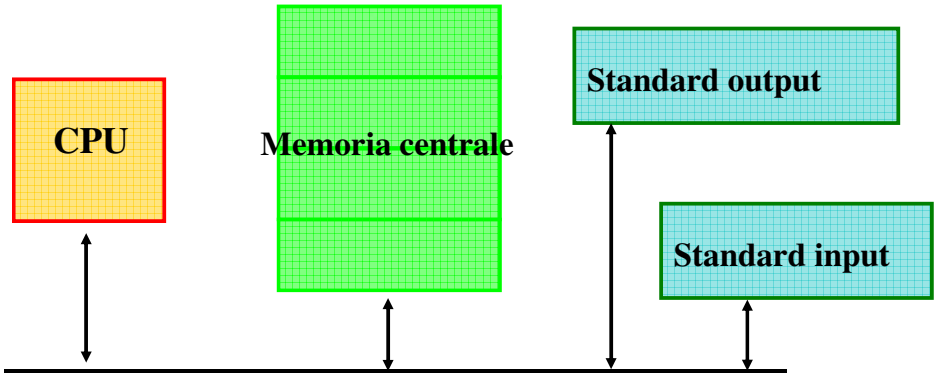
- Per la codifica di un algoritmo bisogna conoscere
  - la sintassi di un linguaggio di programmazione
  - semantica di un linguaggio di programmazione
  - un ambiente di programmazione
- Un linguaggio di programmazione è una notazione formale per descrivere algoritmi
- Un programma è la "codifica" o "implementazione" di un algoritmo in un linguaggio di programmazione
  - Ricordiamo che non tutti i programmi codificano algoritmi …
- Un programma si compone di più istruzioni scritte nel linguaggio di programmazione
  - Quali parole chiave vi sono in un linguaggio?
  - Quali sono i meccanismi di combinazione delle parole chiave?
  - E soprattutto, a quale macchina si riferisce?

Linguaggi di alto livello

- Si basano su una "macchina" le cui "mosse" non sono quelle della macchina hardware ⇒ realizzano una "macchina virtuale"
- Supportano concetti ed astrazioni
  - esiste la possibilità di riferirsi agli elementi del programma tramite un nome (identificatore)
- Promuovono metodologie per agevolare lo sviluppo del software da parte del programmatore
- Hanno capacità espressive molto superiori rispetto a quelle del linguaggio macchina
  - esiste la possibilità di esprimere le istruzioni in un modo più vicino al linguaggio naturale
  - Esistono centinaia di linguaggi di programmazione! (anche se pochi sono in uso)
- Per semplicità, faremo riferimento ad una macchina astratta in grado di comprendere ed eseguire i programmi codificati nel linguaggio corretto
  - il linguaggio è il C
  - la macchina astratta è la macchina C

La macchina C

- E' un'astrazione dell'architettura di von Neumann
- E' dotata di
  - un'unità CPU
  - una memoria centrale
  - un bus di comunicazione
  - un'unica unità di ingresso (detta standard input)
  - un'unica unità di uscita (detta standard output)



Il linguaggio C

- Definito nel 1972 (AT&T Bell Labs) per sostituire l'assembler nella programmazione di sistemi operativi: UNIX
  - Prima definizione precisa: Kernigham & Ritchie (1978)
  - Prima definizione ufficiale: ANSI C (1983)

Il linguaggio C

- Definito nel 1972 (AT&T Bell Labs) per sostituire l'assembler nella programmazione di sistemi operativi: UNIX
  - Prima definizione precisa: Kernigham & Ritchie (1978)
  - Prima definizione ufficiale: ANSI C (1983)

Ciao, mondo!

- Vedremo il cosiddetto ANSI C (standard del 1989, con aggiunte del 1994)

Il primo programma C: ciao mondo

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

- Questo programma stampa sullo schermo una riga di testo:

```
Ciao mondo!
#
```

("#" denota la posizione del cursore)

- Vediamo in dettaglio ogni riga del programma.

Ciao, mondo!

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

- Il testo racchiuso tra "`/*`" e "`*/`" `e un commento
- I commenti 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 `*/`
- Alternativa per commenti di una sola riga: //

- In questo caso, lo standard output è lo schermo
  - lo è sempre, salvo indicazioni esplicite

Ciao, mondo!

`main()`

- E' una parte presente in tutti i programmi C
- Le parentesi "`(`" e "`)`" dopo main indicano che main è una funzione
- I programmi C contengono una o più funzioni, tra le quali ci deve essere la funzione `main`
  - `main` è una funzione speciale, perché l'esecuzione del programma incomincia con l'esecuzione di `main`

La parentesi graffa "`{`" 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

Ciao, mondo!

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

- E' 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
  - "`\`" `e 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)

Ciao, mondo!

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

- E' 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 `printf`)
- Il linguaggio C non prevede istruzioni esplicite di input/output. Queste operazioni sono definite tramite funzioni nella libreria standard di input/output.

- *Note:*
- E' 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

Varianti di "Ciao, mondo!"

```
#include <stdio.h>
main()
{
  printf("Ciao");
  printf(" mondo!\n");
}
```

- Produce lo stesso effetto del programma precedente
- 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");
```

Area di un rettangolo

```
#include <stdio.h>
main() {
  int base;
  int altezza;
  int area;
  base = 3;
  altezza = 4;
  area = base * altezza;
  printf("Area: %d\n", area);
} /* main */
```

- Questo programma stampa:

```
Area: 12
#
```

Variabili

- Una variabile è caratterizzata dalle seguenti proprietà:

1. nome: serve a identificarla
   - Esempio: `area`
   - E' un identificatore C: sequenza di lettere, cifre, e "_" che comincia con una lettera o con "_" (non con una cifra)
   - Esempio: `Numero elementi`, `x1`, ma non "`1 elemento`"
   - può avere lunghezza qualsiasi, ma solo i primi 31 caratteri sono significativi (ANSI)
   - lettere minuscole e maiuscole sono considerate distinte
2. valore: valore associato nello stato corrente
   - Esempio: 4 (può cambiare durante l'esecuzione)
3. tipo: specifica l'insieme dei possibili valori
   - Esempio: `int` (numeri interi)
4. indirizzo: della cella di memoria che contiene il valore associato;
   - Ad ogni variabile è associata una cella di memoria (o più celle di memoria consecutive, a seconda del tipo)

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

Variabili

Analogia con una scatola di scarpe etichettata in uno scaffale

- nome ⇒ etichetta
- valore ⇒ scarpa che c'è nella scatola
- tipo ⇒ capienza (che tipo di scarpe ci metto dentro)
- indirizzo ⇒ posizione nello scaffale (la scatola è incollata)

N.B.

- non tutte le variabili sono denotate da un identificatore
- non tutti gli identificatori sono identificatori di variabile (ad es. funzioni, tipi, . . .)

Dichiarazione di variabile

```
int altezza;
```

E' una dichiarazione di variabile

- viene creata la scatola e incollata allo scaffale
- ha tipo `int` ⇒ può contenere interi
- ha nome `altezza`
- ha un indirizzo (posizione nello scaffale), che è quello della cella di memoria associata alla variabile
- ha un valore iniziale, che però non è significativo (è casuale) ⇒ la scatola viene creata piena, con una scarpa scelta a caso

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

- come per `altezza`

(Alcuni) tipi di variabili

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`
- per inizializzare un reale si può usare la notazione con il punto decimale

Assegnamento

```
base = 3;
altezza = 4;
```

E' un'istruzione di assegnamento (`base←3`)

- "=" è l'operatore di assegnamento
- il suo effetto è quello di assegnare il valore a destra di "=" (in questo caso 3) alla variabile a sinistra (in questo caso `base`)
  - il valore viene scritto nella cella di memoria associata alla variabile
  - a questo punto la variabile `base` ha un valore significativo

Esempio:

```
area = base * altezza;
```

E' un'istruzione di assegnamento, in cui a destra di "=" abbiamo un'espressione

- vengono presi i valori di `base` (3) e `altezza` (4) e viene calcolato il loro prodotto (12)
- tale valore viene assegnato alla variabile `area`
- Nota: il C mette disposizione gli operatori aritmetici tra interi `+`, `-`, `*`, `/`, `%`, . . .

Stampa

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

E' 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`

```
printf("%d%d...%d", i1, i2, ..., in);
```

Struttura dei programmi

- 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)
}
```

Struttura dei programmi

Un programma C deve contenere nell'ordine:

- Una parte contenente direttive per il compilatore. Nel nostro programma c'è
- la direttiva `#include <stdio.h>`
- l'identificatore predefinito `main` seguito dalle parentesi `()`
- due parti racchiuse tra parentesi graffe
  - la parte dichiarativa. Nel nostro caso

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

  - la parte esecutiva. Nel nostro caso

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

La parte dichiarativa

- E' posta prima della codifica dell'algoritmo e obbliga il programmatore a dichiarare i nomi simbolici di cui farà uso nella parte esecutiva. Contiene i seguenti elementi:
  - la sezione delle dichiarazioni di costanti
  - la sezione delle dichiarazioni di variabili
- La dichiarazione dà la possibilità di individuare e segnalare errori in fase di compilazione
- Esempio: Consideriamo il seguente frammento di codice
  - dove abbiamo erroneamente scambiato una b con una f

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

Il compilatore individua `alba` come *variabile non dichiarata.*

Dichiarazioni di variabili

- Abbiamo già visto esempi di dichiarazioni di variabili

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

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

- Una dichiarazione di costante crea un'associazione non modificabile ⇒ associa in modo permanente un valore ad un identificatore
- 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 sulla stessa riga
- Esempio:

```
const float PIGRECO =3.14, E=2.718;
const int N=100, M=200;
```

Parametrizzazione

- Con le dichiarazioni precedenti, 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 tutte le occorrenze di `3.14` in `3.1415` ...

Input da tastiera

```
#include <stdio.h>
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);
}
```

- In assenza di differenti specificazioni, la tastiera è lo standard input

Input da tastiera

`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*
  - la sequenza di caratteri immessa viene convertita in un intero (formato `%d`)
  - l'intero ottenuto viene assegnato alla variabile `base` (viene cioè scritto nella cella di memoria il cui indirizzo è stato passato a `scanf`)
- N.B. il precedente valore della variabile `base` va perduto

Input da tastiera

- Cosa appare sullo schermo:

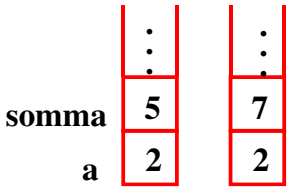
```
Immetti base del rettangolo e premi INVIO
# => 5^
Immetti altezza del rettangolo e premi INVIO
# => 4^
Area: 20
#
```

- (quello che segue "=>" `e l'input dell'utente
- "^" denota la pressione del tasto *Invio*)

Osservazioni sull'assegnamento

- Nell'istruzione `x = e` viene
  - prima valutato il valore dell'espressione `e` a destra di "=" (usando i valori correnti delle variabili);
  - poi tale valore viene assegnato alla variabile `x` a sinistra di "="

```
somma = 5;
a = 2;
somma = somma + a;
```




```
int a, b;
a = 2;
b = 3;
a = b;
a = a + b;
b = a + b;
```

a	b
?	?
2	?
2	3
3	3
6	3
6	9

Osservazioni sull'assegnamento

- A sinistra di "=" ci deve essere un identificatore di variabile
  $\Rightarrow$ denota la corrispondente associazione modificabile nello stato

- Esempio: Quali istruzioni sono corrette e quali no?

- `a = a;` SI corretta (anche se poco significativa . . .)
- `a = 2 * a;` SI corretta (il valore associato ad `a` viene raddoppiato)
- `5 = a;` NO, `5` non denota una associazione nello stato ma un valore costante
- `a + b = c;` NO, `a+b` è un'espressione, non una variabile!