LINGUAGGI DI PROGRAMMAZIONE

Il "potere espressivo" di un linguaggio è caratterizzato da:

- **quali tipi di dati** consente di rappresentare (direttamente o tramite definizione dell'utente)
- **quali istruzioni di controllo** mette a disposizione (quali operazioni e in quale ordine di esecuzione)

PROGRAMMA = DATI + CONTROLLO

IL LINGUAGGIO C

UN PO' DI STORIA

- definito nel 1972 (AT&T Bell Labs) per sostituire l'assembler
- prima definizione precisa: Kernigham & Ritchie (1978)
- prima definizione ufficiale: ANSI (1983)

IL LINGUAGGIO C

CARATTERISTICHE

- linguaggio ***sequenziale, imperativo, strutturato*** a blocchi
- usabile anche come linguaggio di sistema
  - adatto a software di base, sistemi operativi, compilatori, ecc.
- portabile, efficiente, sintetico
  - ma a volte poco leggibile...

IL LINGUAGGIO C

Basato su pochi *concetti elementari*

- dati (tipi primitivi, tipi di dato)
- espressioni
- dichiarazioni / definizioni
- funzioni
- istruzioni / blocchi

ESEMPIO: Un semplice programma

Codifica in linguaggio C dell'algoritmo che converte gradi Celsius in Fahrenheit

```
main(){
  float c, f; /* Celsius e Fahrenheit */
  printf("Inserisci la temperatura da convertire");
  scanf("%f", &c);
  f = 32 + c * 9/5;
  printf("Temperatura Fahrenheit %f", f);
  }
```


STRUTTURA DI UN PROGRAMMA C

In prima battuta, la struttura di un programma C è definita nel modo seguente:

```
<programma> ::=
  {<unità-di-traduzione>}
  <main>
  {<unità-di-traduzione>}
```

Intuitivamente un programma in C è definito da tre parti:

- *una o più unità di traduzione*
- *il programma vero e proprio (main)*
- *una o più unità di traduzione*

STRUTTURA DI UN PROGRAMMA C

La parte <main> è l'unica *obbligatoria,* definita come segue:

```
<main> ::=
  main()
  {[<dichiarazioni-e-definizioni>]
   [<sequenza-istruzioni>]
  }
```

Intuitivamente il main è definito dalla parola chiave **`main()`** *e racchiuso tra parentesi graffe al cui interno troviamo*

- *dichiarazioni e definizioni*
- *una sequenza di istruzioni*

} *opzionali* `[ ]`

STRUTTURA DI UN PROGRAMMA C

- `<dichiarazioni-e-definizioni>`

  introducono i nomi di costanti, variabili, tipi definiti dall'utente

- `<sequenza-istruzioni>`

  sequenza di frasi del linguaggio ognuna delle quali è un'istruzione

`main()` è *una particolare unità di traduzione* (una *funzione*)

STRUTTURA DI UN PROGRAMMA C

- **set di caratteri** ammessi in un programma dipende dall'implementazione; solitamente ASCII + estensioni

- **identificatori**

  sequenze di caratteri tali che

  ```
  <Identificatore> ::=
    <Lettera> { <Lettera> | <Cifra> }
  ```

  *Intuitivamente un identificatore è una sequenza (di lunghezza maggiore o uguale a 1) di lettere e cifre che **inizia obbligatoriamente con una lettera***

COMMENTI

Commenti

sequenze di caratteri racchiuse fra i delimitatori `/*` e `*/`

- ```
  <Commento> ::= /* <frase> */
  <frase>  ::= { <parola> }
  <parola> ::= { <carattere> }
  ```

- i commenti ***non*** possono essere innestati
  ```

VARIABILI

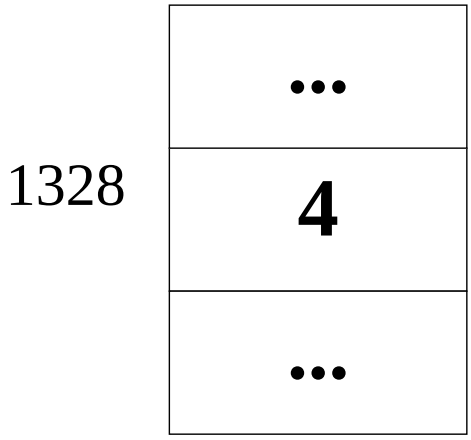
- **Una *variabile* è un'astrazione della *cella di memoria***
- **Formalmente, è un simbolo *associato a un indirizzo fisico (L-value)...***

simbolo	*indirizzo*
x	1328

Perciò, **L-value** di x è 1328 **(fisso e immutabile!)**

VARIABILI

… che *denota un valore (R-value)*



… e **R-value** di x è ***attualmente*** 4 (può cambiare)

DEFINIZIONE DI VARIABILE

Una variabile utilizzata in un programma ***deve essere definita***

La **definizione** è composta da

- ***nome*** della variabile ***(identificatore)***
- ***tipo*** dei valori ***(R-value)*** che possono essere denotati alla variabile

DEFINIZIONE DI VARIABILE: ESEMPI

Definizione di una variabile:

```
<tipo> <identificatore>;
```

```
int x;    /* x deve denotare un valore intero */

float y;  /* y deve denotare un valore reale */

char ch;   /* ch deve denotare un carattere */
```

INIZIALIZZAZIONE DI UNA VARIABILE

- **Contestualmente alla *definizione* è possibile *specificare un valore iniziale* per una variabile**
- **Inizializzazione di una variabile:**

```
<tipo> <identificatore> = <espr> ;
```

Esempio

```
int x = 32;

double speed = 124.6;
```


VARIABILI & ESPRESSIONI

Una variabile

- **può comparire in una espressione**
- **può assumere un valore dato dalla valutazione di un'espressione**

```
double speed = 124.6;
double time = 71.6;
double km = speed * time;
```

CARATTERISTICHE DELLE VARIABILI

campo d'azione (scope): è la parte di programma in cui la variabile è nota e può essere manipolata

- in C, Pascal: determinabile ***staticamente***
- in LISP: determinabile ***dinamicamente***

tipo: specifica la ***classe di valori*** che la variabile può assumere (e quindi gli **operatori** applicabili)

CARATTERISTICHE DELLE VARIABILI

tempo di vita: è l'intervallo di tempo in cui rimane valida l'associazione simbolo/indirizzo fisico (L-value)

- in FORTRAN: allocazione *statica*
- in C, Pascal: allocazione *dinamica*

valore: è rappresentato (secondo la codifica adottata) nell'area di memoria associata alla variabile

ESEMPIO: Un semplice programma

Problema:

"Data una temperatura espressa in gradi Celsius, calcolare il corrispondente valore espresso in gradi Fahrenheit"

Approccio:

- si parte dal problema e dalle proprietà note *sul* dominio dei dati

ESEMPIO: Un semplice programma

Specifica della soluzione:

$$c * 9/5 = f - 32$$

oppure

$$c = (f - 32) * 5/9$$

$$f = 32 + c * 9/5$$

ESEMPIO: Un semplice programma

Algoritmo corrispondente:

- Dato **c**
- calcolare **f** sfruttando la relazione

  f = 32 + c * 9/5

solo a questo punto

si *codifica* l'algoritmo nel linguaggio scelto

ESEMPIO: Un semplice programma

```
main(){
    float c=18;/* Celsius */
    float f = 32 + c * 9/5;
}
```

NOTA: per ora abbiamo a disposizione solo il modo per inizializzare le variabili. Mancano, ad esempio, la possibilità di modificare una variabile, costrutti per l'input/output…

VARIABILI NEI LINGUAGGI IMPERATIVI

Una *variabile* in un linguaggio imperativo

- ***non è solo un sinonimo per un dato*** come in matematica
- **è un'astrazione della cella di memoria**
- **associata a due diverse informazioni:**
  - **il contenuto (R-value)**
  - **l'indirizzo a cui si trova (L-value)**

ESPRESSIONI CON EFFETTI COLLATERALI

- Le espressioni che contengono variabili, *oltre a denotare un valore,* possono a volte comportare *effetti collaterali* sulle variabili coinvolte

- Un *effetto collaterale* è una modifica del valore della variabile (R-value) causato da *particolari operatori:*
  - operatore di *assegnamento*
  - operatori di *incremento e decremento*

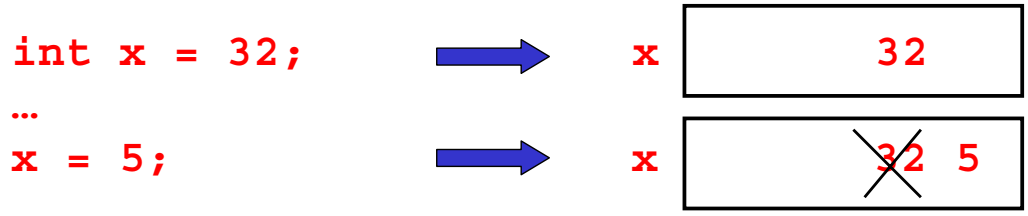
ASSEGNAMENTO

Ad una variabile può essere assegnato un valore nel corso del programma e non solo all'atto della inizializzazione

- **Assegnamento di una variabile: SINTASSI**

  `<identificatore> = <espr> ;`

- **L'assegnamento è l'astrazione della modifica distruttiva del contenuto della cella di memoria denotata dalla variabile**



ASSEGNAMENTO

- L'assegnamento è un ***particolare tipo di espressione***
  - come tale ***denota comunque un valore***

  con un effetto collaterale: quello di ***cambiare il valore*** della variabile
- Esempi di *espressioni di assegnamento*:

  `j = 0` `k = j + 1`

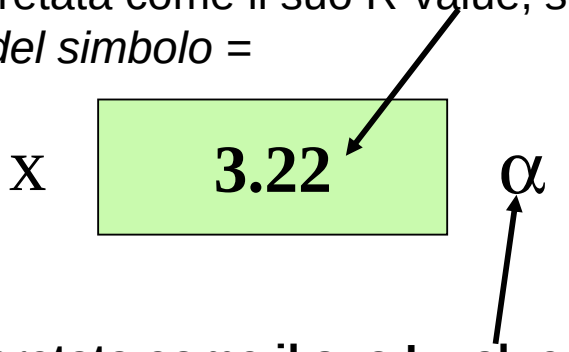
- Se k valeva 2, l'espressione `k = j + 1`
  - denota il valore 1 (risultato della valutazione dell'espressione)
  - *e cambia il valore di k,* che d'ora in poi vale 1 (non più 2)

L'assegnamento è distruttivo

ASSEGNAMENTO & VARIABILI

Una variabile in una espressione di assegnamento:

- è interpretata come il suo R-value, *se compare <u>a destra</u> del simbolo =*



- **è interpretata come il suo L-value, *se compare <u>a sinistra</u> del simbolo =***

ASSEGNAMENTO & VARIABILI

Se x valeva 2, l'espressione

x = x + 1

- denota il valore 3
- *e cambia in 3 il valore di x*
  - il simbolo x **a destra** dell'operatore = denota ***il valore attuale (R-value) di x***, cioè 2
  - il simbolo x **a sinistra** dell'operatore = denota ***la cella di memoria associata a x (L-value),*** a cui viene assegnato il valore dell'espressione di destra (3)
  - l'***espressione*** nel suo complesso denota il ***valore della variabile*** dopo la modifica, cioè 3

OPERATORI DI ASSEGNAMENTO COMPATTI

Il C introduce una *forma particolare di assegnamento* che ***ingloba anche un'operazione***:

```
<identificatore> OP=  <espressione>
```

è ***"quasi equivalente"*** a

```
<identificatore> = <identificatore> OP
                   < espressione>
```

dove `OP` indica un operatore (ad esempio: `+`, `-`, `*`, `/`, `%`, …..).

OPERATORI DI ASSEGNAMENTO COMPATTI

Esempi

`k += j` *equivale a* `k = k + j`

`k *= a + b` *equivale a* `k = k * (a+b)`

Perché "quasi" equivalente ?

- L'identificatore (a sinistra di =) può essere in realtà un'espressione ***`l-espr`***
- le due forme allora sono *equivalenti solo* se la valutazione di ***`l-espr`*** non comporta *effetti collaterali* (***nell'operatore compatto una sola valutazione***; ne vedremo un esempio molto più avanti…)