Tipi di Dato e Array

Informatica B a.a. 2021/2022

Francesco Trovò

28 Settembre 2021

francesco1.trovo@polimi.it

Warm-up

- Scrivere un programma per conteggiare quanto la vostra aula ha speso in totale per il pranzo Venerdì scorso
- Calcolare la spesa media per il pranzo
- Dire anche:
  - chi ha speso di più di tutti
  - se qualcuno ha speso più di tutti gli altri messi assieme
  - supponendo di fare «alla romana», dire chi deve ricevere e chi deve pagare perchè tutti abbiano pagato la propria quota

Ulteriori Funzionalità non Ancora a noi Disponibili

- Per rispondere all'ultima domanda servirebbe **tener traccia** di quanto viene «versato» da ciascuno (ossia i valori assegnati alla variabile `soldi`
- Riprendendo il paragone variabili-foglietti su cui scrivere, servirebbe, al posto i un foglietto `soldi`, una **sequenza di foglietti**, e ciascun foglietto tiene traccia dei valori inseriti

Tipi di Dato

Tipi di Dato

- I **tipi di dato** dicono che una variabile ha:
  - un insieme di **valori** ammissibili
  - un insieme di **operazioni** applicabili
  - uno **spazio** in memoria riservato (numero di celle/parole)
- In C **tutte le variabili** hanno un tipo, associato stabilmente mediante la dichiarazione

NB: La memoria utilizzata per allocare le variabili di un determinato tipo cambia con la piattaforma (combinazione compilatore/sistema operativo/hardware)

Tipi di Dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile**, ad esempio `int, char, float`
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
- Classificazione sulla base di uno standard:
  - **Built-in**, tipi già presenti nel linguaggio base
  - **User-defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built-in

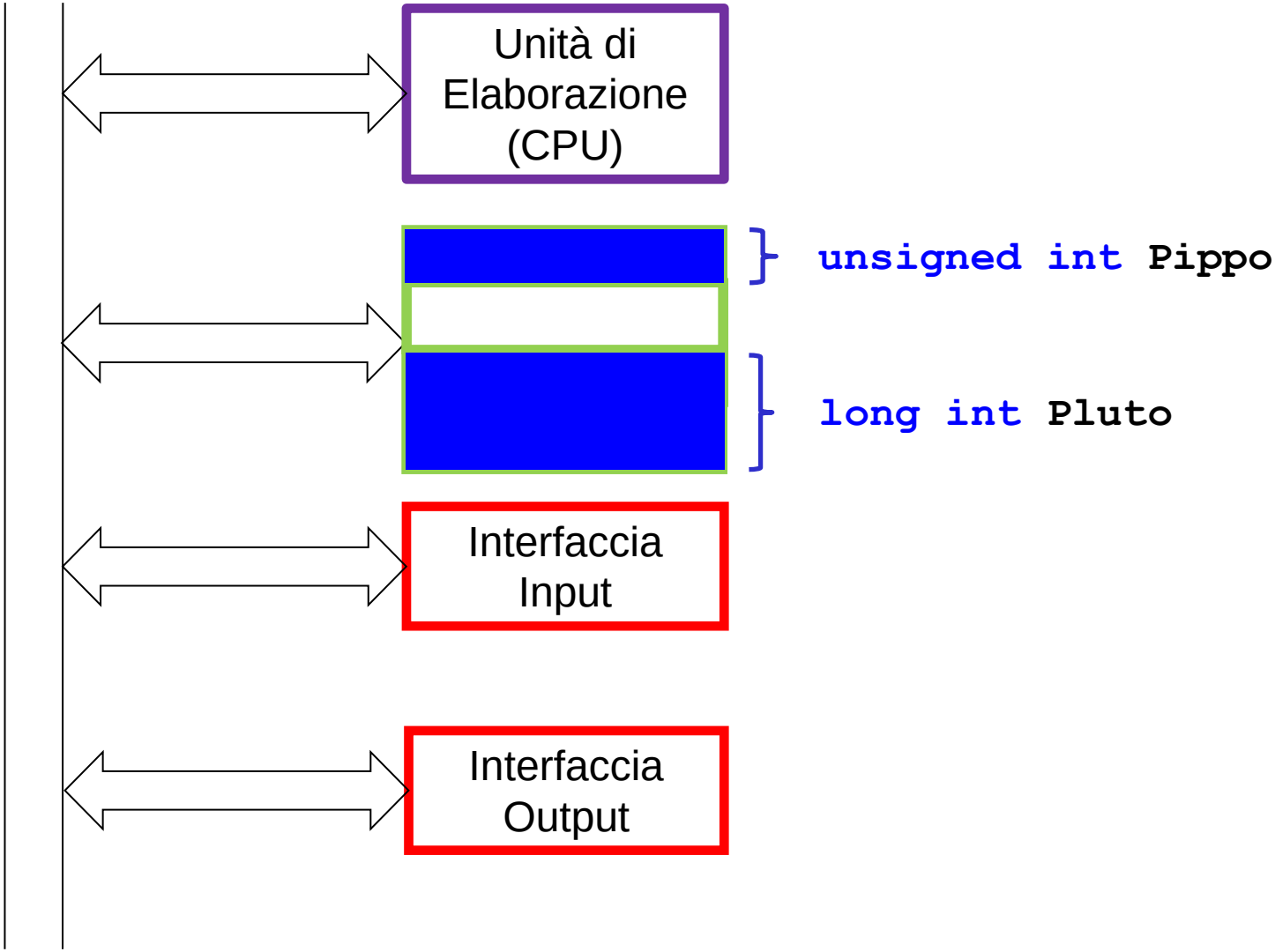
Tipi Semplici

`char, int, float, double`

Tipi Semplici

- Ecco i **quattro tipi semplici** del C e la loro dimensione:
  - `char`: 1 Byte
  - `int`: tipicamente 1 parola di memoria
  - `float`: (spesso 4 Byte)
  - `double`: (spesso il doppio del `float`)
- **Qualificatori** di tipo (per `int` e `char`):
  - `signed` utilizza una codifica con il segno
  - `unsigned` prevede solo valori positivi
- **Quantificatori** di tipo:
  - `short` (per `int`)
  - `long` (per `int` e `double`)

Qualificatori, Quantificatori e Spazio Allocato



Il Tipo `int`

- Rappresentano un sottoinsieme di $\mathbb{N}$
- Fatti garantiti:
  - spazio (**`short int`**) $\leq$ spazio (**`int`**) $\leq$ spazio (**`long int`**)
  - spazio (**`signed int`**) $=$ spazio (**`unsigned int`**)

Es. se la parola è a 32 bit:

- **`signed int`** $\{-2^{31}, \ldots, 0, \ldots, +2^{31}-1\}$, ($2^{32}$ numeri)
- **`unsigned int`** $\{0, \ldots, +2^{32}-1\}$, (2^{32} numeri)

- Come faccio a sapere i limiti per un intero?
  - **`#include <limits.h>`**, e richiamo le costanti **`INT_MIN`**, **`INT_MAX`**
- Quando il valore di una variabile **`int`** eccede **`INT_MAX`** si ha **overflow**

Operazioni Built-in per Dati di Tipo `int`

- = Assegnamento di un valore `int` a una variabile `int`
- + Somma (tra `int` ha come risultato un `int`)
- - Sottrazione (tra `int` ha come risultato un `int`)
- * Moltiplicazione (tra `int` ha come risultato un `int`)
- / Divisione con troncamento della parte non intera (risultato `int`)
- % Resto della divisione intera
- == Relazione di uguaglianza
- != Relazione di diversità
- < Relazione “minore di”
- > Relazione “maggiore di”
- <= Relazione “minore o uguale a”
- >= Relazione “maggiore o uguale a”

Operatoori Aritmetici

Operatoori Relazionali

I Tipi `float` e `double`

- Approssimazione di $\mathbb{R}$ (che è un insieme denso), quindi i **valori** vengono approssimati per «magnitudine», e limiti nella **precisione** della rappresentazione
- Nella rappresentazione in virgola mobile (floating point) il numero n si scrive come due parti separate da "E":
  - m mantissa
  - e esponente (rispetto alla base 10)

  tali che $n = m * 10^e$

Es.1 780 000.000 0023 in virgola mobile diventa:

178 000.000 000 23 E1 oppure
17 800 000 000 023 E-7 oppure
1.780 000 000 0023 E+6

Spazio per `float` e `double`

- Fatto garantito:
  - spazio (`float`) ≤ spazio (`double`) ≤ spazio (`long double`)
- Su architetture standard un `float` occupa 4 byte e un `double` 8 byte con
  - accuratezza: 6 decimali per `float`
    15 decimali per `double`
  - valori tra 10^{-38} e 10^{+38} per `float`
    tra 10^{-308} e 10^{+308} per `double`
- Standard library **`math.h`** fornisce funzioni predefinite (**`sqrt`**, **`pow`**, **`exp`**, **`sin`**, **`cos`**, **`tan`**) applicate a valori double

Operazioni Built-in per Dati di Tipo `float`

- = Assegnamento di un valore `float` a una variabile `float`
- + Somma (tra `float` , risultato `float`)
- - Sottrazione (tra `float` , risultato `float`)
- * Moltiplicazione (tra `float`, risultato `float`)
- / Divisione (tra `float`, risultato `float`)
- == Relazione di uguaglianza
- != Relazione di diversità
- < Relazione “minore di”
- > Relazione “maggiore di”
- <= Relazione “minore o uguale a”
- >= Relazione “maggiore o uguale a”

Operatoori Aritmetici

Operatoori Relazionali

Operazioni tra `float`: la Divisione

- Operazioni applicabili a **`float`** (anche a **`double`** e **`long double`**) sono le stesse degli **`int`**, ma divisione `'/'` dà risultato reale

NB: il simbolo dell'operazione è identico a quello per la divisione intera

Operazioni tra `float`: l'Uguaglianza

- Nella rappresentazione di un numero decimale possono esserci **errori di approssimazione:**
  - non sempre: **`(x / y) * y == x`**
  - per verificare l'uguaglianza tra **`float`** o **`double`**, definire delle tolleranze:
    - **`if (x == y) ...`** è meglio
    - **`if (x <= y + .000001 && x >= y - .000001)`**

Il Tipo `char`

- La codifica ASCII prevede di allocare sempre 1 Byte per rappresentare caratteri
  - alfanumerici
  - di controllo (istruzioni legate alla visualizzazione)
- C'è una corrispondenza tra i **`char`** e 256 numeri interi
- Le operazioni sui **`char`** sono le stesse definite su **`int`**
  - hanno senso gli operatori aritmetici (+ - * / %)
  - hanno senso gli operatori di relazione (== , > , < , etc.)
- **`unsigned char`** coprono l'intervallo $[0, 255]$.
- **`signed char`** coprono l'intervallo $[-128, 127]$.

NB: non esistono tipi semplici più «piccoli» del **`char`**

Il Tipo `char`

- I valori costanti di tipo **`char`** nel codice sorgente si delimitano tra apici singoli `' '`
- Gli apici doppi `" "` vengono utilizzati per delimitare stringhe, ovvero sequenze di caratteri

La Codifica ASCII

Caratt.	Byte	Caratt.	Byte	Caratt.	Byte	Caratt.	Byte
NUL	00000000	SPACE	00100000	@	01000000	`	01100000
SOH	00000001	!	00100001	A	01000001	a	01100001
STH	00000010	"	00100010	B	01000010	b	01100010
EXT	00000011	#	00100011	C	01000011	c	01100011
EQT	00000100	$	00100100	D	01000100	d	01100100
EQN	00000101	%	00100101	E	01000101	e	01100101
ACK	00000110	&	00100110	F	01000110	f	01100110
BEL	00000111	'	00100111	G	01000111	g	01100111
BS	00001000	(	00101000	H	01001000	h	01101000
HT	00001001	)	00101001	I	01001001	i	01101001
LF	00001010	*	00101010	J	01001010	j	01101010
VT	00001011	+	00101011	K	01001011	k	01101011
FF	00001100	,	00101100	L	01001100	l	01101100
CR	00001101	-	00101101	M	01001101	m	01101101
SO	00001110	.	00101110	N	01001110	n	01101110
SI	00001111	/	00101111	O	01001111	o	01101111
DLE	00010000	0	00110000	P	01010000	p	01110000
DC1	00010001	1	00110001	Q	01010001	q	01110001
DC2	00010010	2	00110010	R	01010010	r	01110010
DC3	00010011	3	00110011	S	01010011	s	01110011
DC4	00010100	4	00110100	T	01010100	t	01110100
NAK	00010101	5	00110101	U	01010101	u	01110101
SYN	00010110	6	00110110	V	01010110	v	01110110
ETB	00010111	7	00110111	W	01010111	w	01110111
CAN	00011000	8	00111000	X	01011000	x	01111000
EM	00011001	9	00111001	Y	01011001	y	01111001
SUB	00011010	:	00111010	Z	01011010	z	01111010
ESC	00011011	;	00111011	[	01011011	{	01111011
FS	00011100	<	00111100	\	01011100	|	01111100
GS	00011101	=	00111101	]	01011101	}	01111101
RS	00011110	>	00111110	^	01011110	~	01111110
US	00011111	?	00111111	_	01011111	DEL	01111111

Il tipo `char` esempi

```
char a,b;
b = 'q';        /* Le costanti di tipo carattere si
                   indicano con ' */
a = "q";        /* NO: "q" è una stringa, anche se di
                   un solo carattere */
a = '\n';       /* OK: \n è un carattere a tutti gli
                   effetti anche sono due elementi*/
b = 'ps';       /* NO:'ps' non è un carattere valido*/
a = 75;         /*associa ad a il carattere 'K'
a = 'c' + 1;    /* a diventa  'd' */
a = 'c' - 1;    /* a diventa  'b' */
a = 20;
a *= 4;         // sta per a = a * 4, quindi a=80 ('P')
a -= 10;        // a = 70 che corrisponde al carattere 'F'
a = '1';        // a è il carattere 1, ovvero l'intero 49
```

Riepilogando sui tipi built-in

- I tipi **integral** sono discreti, rappresentano valori **numerabili**
  - sono `char` ed `int` con tutti i qualificatori e quantificatori (`signed/unsigned char`, `short/long int`, `signed/unsigned int`)
- I tipi **floating** approssimano insiemi **densi**
  - sono `float` e `double`, eventualmente con il quantificatore `long`
- I tipi **integral** e **floating** assieme compongono il tipo **arithmetic**

Tipi di Dato Strutturati

gli Array

Tipi di Dato

- Classificazione sulla base della struttura:
  - **Tipi semplici**, informazione logicamente **indivisibile**, ad esempio `int, char, float`
  - **Tipi strutturati**: aggregazione di variabili di tipi semplici
- Classificazione sulla base di uno standard:
  - **Built-in**, tipi già presenti nel linguaggio base
  - **User-defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built-in

I Tipi Strutturati in C

- Permettono di immagazzinare informazione aggregata
  - Vettori e matrici in matematica
  - Testi (sequenza di caratteri)
  - Immagini
  - Rubriche
  - Archivi
  - …

- Le variabili strutturate memorizzano diversi elementi che possono essere tra loro:
  - omogenei
  - eterogenei

Il Costruttore Array

- Gli array sono **sequenze** di **variabili omogenee**
  - **sequenza:** hanno un ordinamento (sono indicizzabili)
  - **omogenee:** tutte le variabili della sequenza sono dello stesso tipo
- Ogni elemento della sequenza è individuato da un indice

Il Costruttore Array

Sintassi dichiarazione di una variabile mediante costruttore array

```
tipo nomeArray[Dimensione];
```

- `tipo` la keyword di un tipo (built in o user-defined)
- `nomeArray` è il nome della variabile
- `Dimensione` è **un numero** che stabilisce il numero di elementi della sequenza

NB: `Dimensione` è un **numero fisso**, noto a **compile-time**:

- non può essere una variabile, perché il suo valore sarebbe definito solo a run-time)
- non è possibile modificare le dimensioni durante l'esecuzione, ovvero allungare o accorciare l'array

Esempi

Es.

- `int vet[8];`
- `char stringa[5];`
- `float resti[8];`

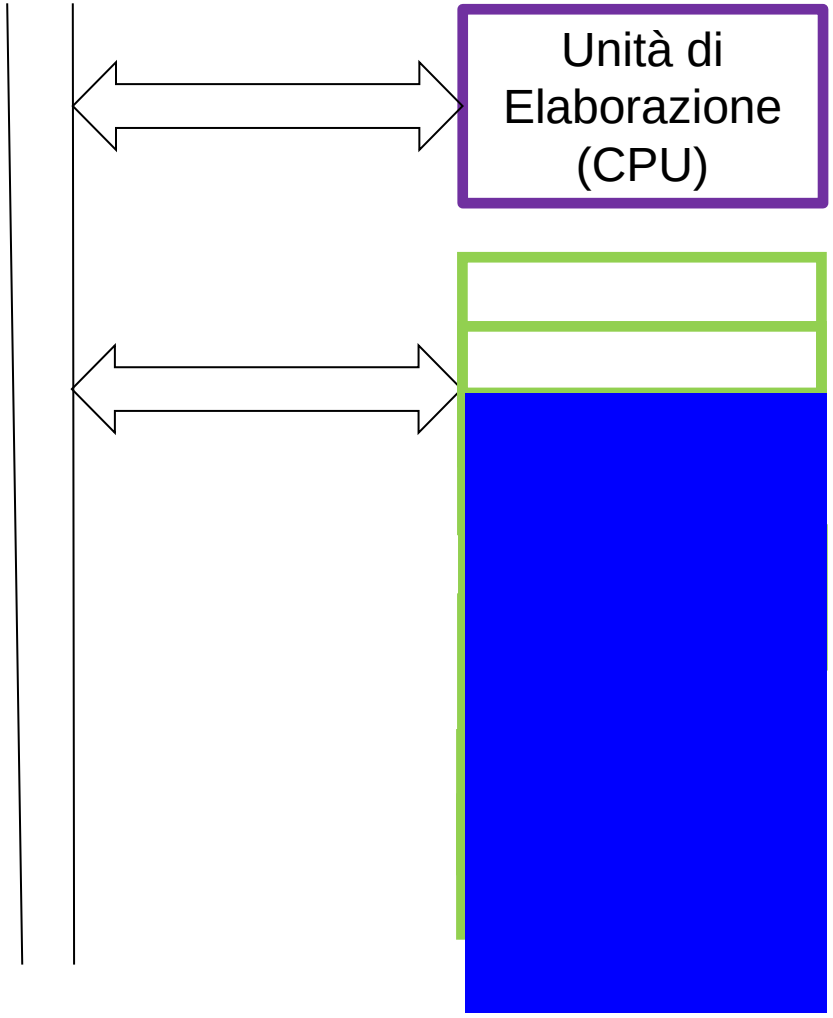
vet
134
34
123
43215
2365
-145
523
45

stringa
‘a’
‘K’
‘\n’
‘3’
‘\t’

resti
2.45
3.24
4.23
1245.2
236.5
-5.0
43.53
0

Lo Spazio Allocato per gli Array

- `int Bilbo[20];`



- Occupo 20 celle consecutive
- Le dimensioni devono essere note **prima della compilazione** per decidere quanto spazio lasciare in memoria per l'array

Accedere agli Elementi dell'Array

- È possibile accedere agli elementi dell'array specificandone **un indice** tra parentesi quadre `[]`

```
int vet[20];
```

`vet[0]` è il primo elemento della sequenza

`vet[19]` è l'ultimo elemento della sequenza

- **Ogni elemento** dell'array è una **variabile** del **tipo** dell'array:

`vet[7]` conterrà un valore intero

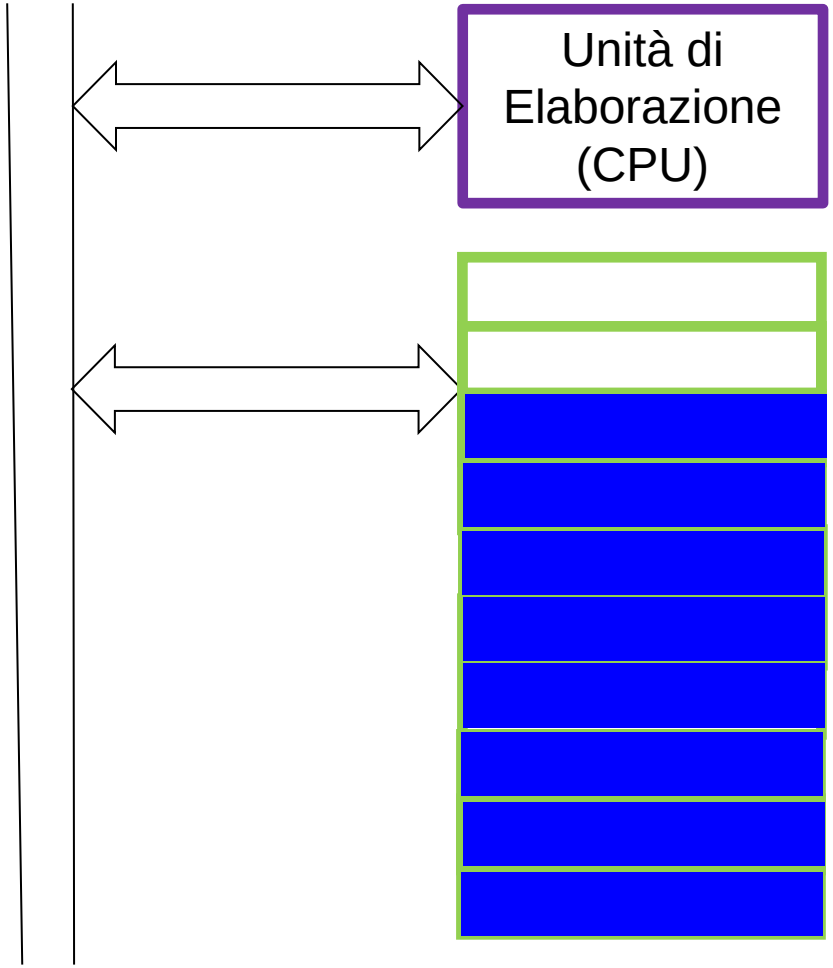
- Una volta **fissato l'indice**, non c'è differenza tra un elemento dell'array ed una **variabile** dello stesso tipo

```
int a; a = vet[0];
vet[0] = a; vet[0] += a;
```

NB: in C gli array sono indicizzati a partire da 0

Lo spazio allocato per gli array

```
int Bilbo[20];
```



- **`Bilbo[2]`** è a tutti gli effetti una variabile intera

```
int Bilbo[0]
int Bilbo[1]
int Bilbo[2]
int Bilbo[3]
int Bilbo[4]
int Bilbo[5]
int Bilbo[6]
int Bilbo[7]
```

Accedere agli Elementi dell'Array

- Il valore **dell'indice** è di tipo **integral** (`char` , `int`)
- È quindi possibile utilizzare una **variabile per definire l'indice** all'interno dell'array
  - `int vet[20]; int i = 0;`
- L'espressione `vet[i]` va interpretata nel seguente modo:
  - leggi il valore di `i`
  - accedi all'elemento di `vet` alla posizione di indice `i`
  - leggi il valore che trovi in quella cella di memoria (`vet[i]`)
- Con lo stesso criterio posso interpretare `vet[i+1];`

Esempi di Operazioni su Array

- Una volta **fissato l'indice** in un array si ha una **variabile** del **tipo dell'array** che può essere usata per
  - assegnamenti

```
vet[2] = 7; vet[4] = 8 % 3;
i = 0; vet[i] = vet[i+1];
```

  - operazioni logiche

```
vet[0] == vet[9]; vet[1] < vet[4];
```

  - operazioni aritmetiche

```
vet[0] == vet[9] / vet[2] + vet[1] / 6;
```

  - operazioni di I/O

```
scanf("%d", &vet[9]);
printf("valore pos %d = %d", i, vet[i]);
```

Esercizio

Scrivere un frammento di codice per dichiarare un array di dimensione 3 e per scrivere in ogni variabile un numero (da 1 a 3) corrispondente alla posizione della cella

```
int vet[3];
vet[0] = 1;
vet[1] = 2;
vet[2] = 3;
```

```
int a,b,c;
a = 1;
b = 2;
c = 3;
```

Vantaggi degli Array

Come faccio a richiamare "il secondo valore inserito"?

- Con le variabili devo salvare da qualche parte che **a** contiene il primo valore, **b** il secondo e **c** il terzo
- Con il vettore mi basta accedere a **vet[1]** perché gli elementi di un vettore seguono un ordinamento

```
int vet[3];
vet[0] = 1;
vet[1] = 2;
vet[2] = 3;
```

```
int a,b,c;
a = 1;
b = 2;
c = 3;
```

Vantaggi degli Array

- La soluzione diventa decisamente impraticabile quando si richiedono molte variabili
- Occorre usare array:
  - perché sono indicizzati
  - perché posso popolarli/elaborarli con un ciclo
- Con i vettori tipicamente il **`for`** risulta molto più comodo del **`while`** perché la variabile del ciclo viene usata per indicizzare gli elementi dell'array

```
int vet[3];
vet[0] = 1;
vet[1] = 2;
vet[2] = 3;
```

```
int a,b,c;
a = 1;
b = 2;
c = 3;
```

Esercizio

Scrivere un programma che dichiara un array di dimensione 300 e scrive in ogni cella un numero da 1 a 300

```
#include <stdio.h>
void main(){
    int vet[300]; int i;
    for (i = 0; i < 300 ; i++)
      vet[i] = i + 1 ;
}
```

Il Valore dell'Array

- Abbiamo visto che gli elementi dell'array contengono valori del tipo dell'array
- Quando scrivo `int vet[300];` so che in `vet[0]` troverò un intero
- Cosa c'è invece in `vet`?
  - L'indirizzo del primo elemento in memoria, ovvero:

```
vet == &vet[0];
```

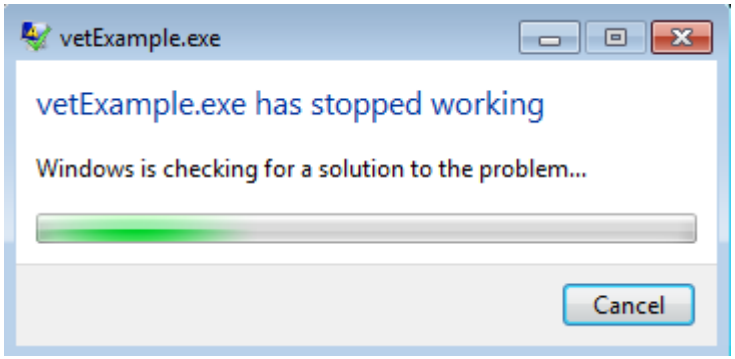
Le Dimensioni degli Array

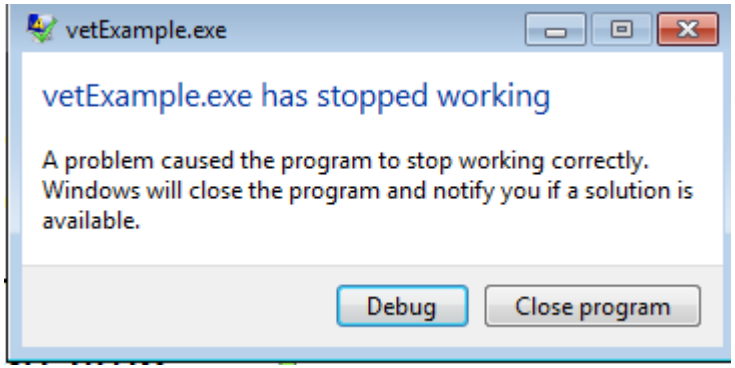
- Non è possibile accedere ad un elemento dell'array ad una posizione superiore alla dimensione
- Se definisco un vettore come:

```
int vet[20];
```

non posso scrivere **`vet[40]`** o **`vet[20]`** visto che le 20 celle vanno da **`vet[0]`** a **`vet[19]`**

- In tal caso si ha **segmentation fault**, che nella migliore delle ipotesi si manifesta **solamente** a run-time





Cosa da non Fare

Errore:

```
int dim; /* il valore a dim è associato solo
durante l'esecuzione */

scanf("%d", &dim);

float resti[dim]; /* quindi il compilatore non
sa quanto spazio riservare in memoria per
resti */
```

Anche se il compilatore lo permette, questo NON si deve fare, perché espone il codice a vulnerabilità

#define

- Spesso si ricorre alla direttiva di precompilazione **define** per dichiarare la lunghezza degli array

```
#define NOME_DEFINE valoreNumerico
```

- Prima della compilazione, ogni istanza di **NOME_DEFINE** (riferibile all'uso di variabile) verrà sostituita da **valoreNumerico**
- Se dichiaro `int vet[NOME_DEFINE];` le dimensioni di **vet** sono note prima di iniziare la compilazione
- L'utilizzo di **define** rende il codice più leggibile e facilmente modificabile quando occorre cambiare la dimensione dell'array

NB: non occorre il **;** dopo **valoreNumerico**

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
        int  v1 [MAX_LEN];
        int i;
        for(i = 0; i < MAX_LEN; i++)
        {
         printf("Inserire elemento posizione %d" , i+1);
         scanf("%d" , &v1[i]);
        }
}
```

Uso **MAX_LEN** come una costante nel codice

L'acquisizione con **scanf** avviene come per una qualsiasi variabile intera

Dettaglio per evitare di stampare «Inserire elemento posizione 0»

Le Dimensioni degli Array: Effettive *vs* Reali

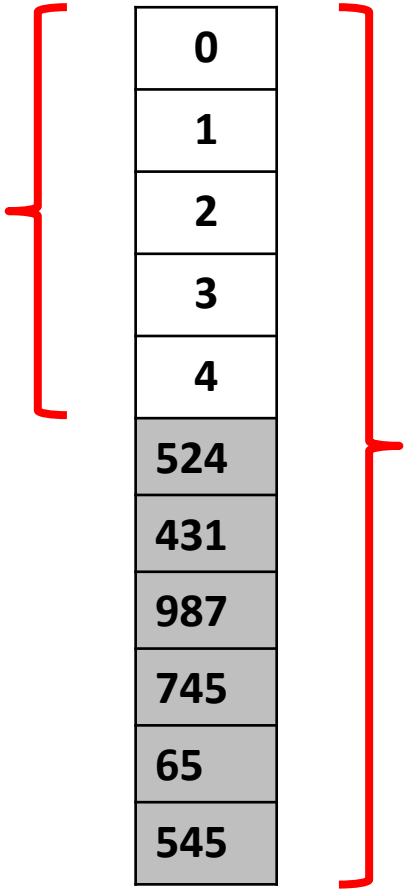
- Le dimensioni **reali** sono quelle con cui viene **dichiarato un array** e non sono modificabili
- Si fissano «grandi a sufficienza»

- Le dimensioni **effettive** delimitano la **parte dell'array che si utilizzerà** durante l'esecuzione
- Possono essere specificate dall'utente in una variabile, previo controllo di compatibilità con quelle reali

Esempio

`int v1[11];` con dimensioni effettive `n = 5;`

Dimensione effettiva dell'array: data dall'utente con **`n`**



Dimensioni reali dell'array: definite da **`MAX_LEN`**

Esempio++

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
        int  v1 [MAX_LEN];
        int i, n; // n contiene le dimensioni effettive
        do
        {
         printf("quanti numeri vuoi inserire?");
         scanf("%d" , &n);
        }while(n < 0 || n > MAX_LEN);

        for(i = 0; i < n; i++)
        {
         printf("Inserire elemento posizione %d" , i+1);
         scanf("%d" , &v1[i]);
        }
}
```

Sono certo che **n** è compatibile con le dimensioni reali di **v1**

Da qui in poi opero solo sulle prime **n** celle

Stampa dei Valori dell'Array

- In generale non esiste un metodo univoco per stampare gli array
- Quindi occorre procedere iterando
- Assumiamo che l'array **v1** abbia dimensioni effettive **n**

```
printf("\nHai inserito: [");
    for(i = 0 ; i < n ; i++)
        printf(" %d ", v1[i]);
     printf("]");
```

- Per gli array di caratteri questo non è necessario (ma è comunque possibile seguire questa strada)

Esercizio

- Scrivere un programma per conteggiare quanto la vostra aula ha speso in totale per il pranzo Venerdì scorso
- Calcolare la spesa media per il pranzo
- Dire anche:
  - chi ha speso di più di tutti
  - se qualcuno ha speso più di tutti gli altri messi assieme
  - **supponendo di fare «alla romana», dire chi deve ricevere e chi deve pagare perchè tutti abbiano pagato la propria quota**

Homeworks

- Scrivere un programma che richiede all'utente di inserire una sequenza di numeri, specificando anticipatamente il numero di elementi che si intende inserire (controllare che sia compatibile con le dimensioni massime, 100)
- Il programma
  - Calcola il massimo della sequenza
  - Stampa tutti gli elementi inseriti in ordine contrario
  - Stampa tutti i numeri dal massimo fino all'ultimo elemento dell'array
  - Conta le occorrenze del massimo

Assegnamento tra Array

- Non c'è un modo per assegnare direttamente **tutti** i valori di un primo array ad un secondo array

```
#include <stdio.h>

void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            v1[i] = i+1;
      v2 = v1;
}
```

```
Error: incompatible
types when assigning
to type 'int[300]'
from type 'int *'
```

Assegnamento tra Array

- Occorre operare su ogni singolo elemento dell'array

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{       int  v1 [MAX_LEN], v2[MAX_LEN];
        int i;
        // popolo v1
        for(i = 0; i < MAX_LEN; i++)
                v1[i] = i;
        // copio i valori in v2
        for(i = 0; (i < MAX_LEN) ; i++)
                v2[i] = v1[i];
        // stampo
        for(i = 0; (i < MAX_LEN) ; i++)
                printf("\nv1[%d] = %d , v2[%d] = %d", i,
                        v1[i], i, v2[i]);
}
```

Confronto tra Array

- Non c'è un modo per confrontare direttamente **tutti** i valori di due array

```
#include <stdio.h>

void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            { v1[i] = i+1;
              v2[i] = v1[i]; }
      if (v1 == v2)
            printf("ok"); }
```

Non da errore di compilazione ma non fa quello che vorremmo…

Confronto tra Array

- Occorre operare su **ogni singolo elemento**

```
#define MAX_LEN 300
#include <stdio.h>
void main()
{       int  v1 [MAX_LEN], v2[MAX_LEN];
        int i, uguali;
        for(i = 0; i < MAX_LEN; i++)
        {
                v1[i] = i+1;
                v2[i] = v1[i];
        }
        uguali = 1;
        for(i = 0; (i < MAX_LEN) && uguali; i++)
                if(v1[i] != v2[i])
                        uguali = 0;
        if(uguali)
                printf("ook!");}
```

Variabile di flag, diventa **0** appena trova una cella per cui **v1** e **v2** differiscono

Scorro tutti gli elementi dei vettori e mi arresto appena trovo due elementi differenti

Sta per (**uguali != 0**)

Variabili di Flag per Verificare Condizioni su Array

- Per controllare che una condizione (uguaglianza in questo caso) sia soddisfatta da tutti gli elementi del vettore

```
uguali = 1;
for(i = 0; i < MAX_LEN; i++)
  if(v1[i] != v2[i])
    uguali = 0;
```

- Al termine del ciclo, se uguali è rimasta 1 sono certo che la condizione da verificare **non è mai stata negata** (i.e., `v1[i] != v2[i]` è sempre stata falsa), quindi che **tutti** gli elementi degli array coincidono

- La variabile di flag (`uguali`) può solo cambiare da `1` in `0`, **mentre non può mai passare da `1` a `0`**

Errore Frequente

- Errore: modificare il valore della variabile di flag nel anche nel verso opposto

```
uguali = 1;
  for(i = 0; i < MAX_LEN; i++)
    if(v1[i] != v2[i])
      uguali = 0;
    else
      uguali = 1;
```

- Alla fine del ciclo se uguali è 1 posso solo concludere che l'ultima coppia di elementi controllati coincide
- Quando la condizione di permanenza nel ciclo è `(i < MAX_LEN) && uguali` il problema non si pone ma `else` risulta comunque inutile

Copiare «Senza Lasciare Buchi»

- In molti casi è richiesto di **scorrere** un array **`v1`** e di **selezionare** alcuni valori secondo una data condizione
- Tipicamente i valori selezionati in **`v1`** vengono **copiati in un secondo array**, **`v2`**, per poter essere utilizzati
- È buona norma copiare i valori **nella prima parte** di **`v2`** , eseguendo quindi una copia «senza lasciare buchi»
- Ciò permette di sapere quali sono i valori significativi in **`v2`** e quali no

Es. copiare i numeri pari di **`v1`** in **`v2`**

- **`v1`**

5	6	7	89	568	68	657	989	96	98

- **`v2`**

?	6	?	?	568	68	?	?	96	98

- **`v2`**

6	568	68	96	98	?	?	?

Copiare «Senza Lasciare Buchi»

Per fare questo è necessario usare **due indici**:

- **`i`** per **scorrere `v1`**: parte da **0** e arriva a **`n1`**, la dimensione effettiva di **`v1`** , con **incrementi regolari**
- **`n2`** parte da **0** e viene **incrementato solo quando un elemento viene copiato**
  - **`n2`** **indica** quindi **il primo elemento libero** in **`v2`** ,
  - al termine, **`n2`** conterrà il **numero di elementi** in **`v2`** , ed è quindi la sua **dimensione effettiva**

Es.

5	6	7	89	568	68	657	989	96	98

```
i = 10;
n1 = 10;
```

6	568	68	96	98	?	?	?

```
n2 = 5;
```

Esercizio

- Chiedere all'utente di inserire un array di interi (di dimensione definita precedentemente) e quindi un numero intero **`t`**
- Il programma quindi:
  - salva gli elementi inseriti in un vettore **`v1`**
  - copia tutti gli elementi di **`v1`** che sono maggiori di **`t`** in un secondo vettore **`v2`**
  - Stampa il contenuto di **`v2`**

Array di Caratteri

Array di Caratteri: le Stringhe

- Nel C le stringhe (sequenze ininterrotte di caratteri) sono realizzate mediante array di caratteri

  Es. `char luogo[100];`

  è un array atto a contenere 100 elementi di tipo `char`

- Dato il frequente utilizzo ci sono standard e comandi particolari per facilitare l'uso delle stringhe:
  - I/O
  - calcolo lunghezza
  - confronto e copia

NB: NON esiste il tipo predefinito "string"

Acquisizione e Stampa di Stringhe

- Come per ogni array è possibile popolare un array di caratteri mediante inserimento carattere per carattere

```
printf("Inserire lunghezza stringa");
scanf("%d" , &n);
for(i = 0; i < n; i++)
    scanf("%c" , &luogo[i]); scanf(" %c");
```

- Alternative più comode:

```
1. scanf("%s", luogo);
2. gets(luogo);
```

NB: `luogo` è l'indirizzo del primo elemento `&luogo[0]`, `scanf` quindi non ha bisogno della `&`

NB: `gets` richiede la libreria string, quindi occorre inserire

```
#include <string.h>
```

Esercizio

- Acquisizione e stampa di una stringa elemento per elemento

Soluzione

```
void main(){
int i;
char s[10];
for (i = 0; i < 10; i++){
  printf("\ninserire carattere %d", i+1);
  scanf("%c", &s[i]); scanf(" %c");
}
for(i = 0; i < 10; i++)
  printf("%c\n", s[i]);

}
```

Il Terminatore di Stringa

- Sia **`scanf("%s",…)`** che **`gets(…)`** delimitano la **parte significativa** (i caratteri inseriti dall'utente) con il **carattere speciale** `'\0'` (tappo, con codifica ASCII = 0)
- Se dopo **`gets(luogo);`** digito `Milano`, in memoria verrà scritto **`"Milano\0"`**
- Differenze:
  - **`scanf("%s", luogo);`** **termina** l'inserimento **al primo spazio o invio**
  - **`gets(luogo);`** **termina** l'inserimento al primo **invio**

Es. digito `Piazza san Babila`

- **`scanf("%s", luogo);`** **`"Piazza\0"`**
- **`gets(luogo);`** **`"Piazza san Babila\0"`**

Stampa di Stringhe

- È possibile stampare i caratteri in una stringa **fino al terminatore** utilizzando `printf("%s", …);`

Es.

```
gets(luogo);
printf("Io abito a %s", luogo);
```

- Quando si popola la stringa carattere per carattere, è necessario lasciare un carattere al terminatore di stringa `'\0'`:

Es.

```
char provincia[3];
provincia[0] = 'M';
provincia[1] = 'I';
provincia[2] = '\0';
printf("Io abito a %s", provincia);
```

Calcolo della Lunghezza

- È possibile calcolare la lunghezza di una stringa andando a contare gli elementi che precedono `'\0'`

```
int len = 0;
char luogo[100];
scanf("%s", luogo);
while(luogo[len] != '\0')
    len++;
printf("%s e' lunga %d", luogo, len);
```

- Oppure è possibile usare la funzione **`strlen`**, contenuta nella libreria **`string`**

```
len = strlen(luogo);
```

Esercizio

- Acquisire due stringhe e valutare le loro lunghezze

```
#include <string.h>
void main(){
  int len1,len2;
  char str1[30], str2[30];
  printf("Inserire prima stringa: ");
  gets(str1);
  printf("Inserire seconda stringa ");
  gets(str2);
  // calcolo le lunghezze
  len1 = strlen(str1);
  // calcolo le lunghezze
  len2 = strlen(str2);
  printf("\n%s e' lunga %d, %s e' lunga %d", str1,
len1, str2, len2);
}
```

Confronto tra Stringhe

- Per verificare se due stringhe coincidono:
  - controllo che la loro lunghezza coincide
  - controllo che coincidono elemento per elemento

```
int flag = 1; int len,i;
char str1[30], str2[30];
gets(str1); gets(str2);
len = strlen(str1);
if(len == strlen(str2))
    for(i = 0; i < len && flag; i++)
          {if(str1[i] != str2[i])
                flag = 0; }
else
    flag = 0;
printf("%s == %s : %d", str1, str2, flag);
```

È indispensabile mettere le parentesi attorno a questo if altrimenti l'else seguente viene associato a questo e non a quello più esterno

Confronto tra Stringhe

- Oppure è possibile usare la funzione **`strcmp`**, contenuta nella libreria **string**. Sintassi

```
strcmp(s1, s2);
```

- Restituisce un intero:
  - == 0 se coincidono
  - < 0 se **`s1`** precede **`s2`** in ordine alfabetico
  - > 0 se **`s1`** segue **`s2`** in ordine alfabetico

```
int cmpr; cmpr = strcmp(str1, str2);
if (cmpr == 0)
   printf("%s e %s coincidono", str1, str2);
```

NB: Le stringhe **`str1`** e **`str2`** devono terminare con **`'\0'`**

Esempio

```
#include<string.h>
void main(){
int coincidono, len1, len2, flag;
char str1[30], str2[30];
// strcmp che restituisce 0 se coincidono
flag = strcmp(str1 , str2);
// metto coincidono a 1 quando flag è 0
coincidono = (flag == 0);
printf("\n%s == %s : %d", str1, str2, coincidono);
if (flag > 0)
  printf("\n%s precede %s(flag = %d)", str2, str1,
flag);
if(flag < 0)
  printf("\n%s precede %s(flag = %d)", str1, str2,
flag);
}
```

Copia tra Stringhe

- È possibile eseguire la copia elemento per elemento da un array ad un altro, come nell'esercizio precedente
- Oppure è possibile usare la funzione **`strcpy`**, contenuta nella libreria **string.h** Sintassi:

```
strcpy(s1, s2);
```

- Copia il contenuto di **`s2`** in **`s1`** incluso il tappo **`'\0'`**

- Per accodare le stringhe si usa la funzione **`strcat`**, contenuta nella libreria **string**. Sintassi:

```
strcat(s1, s2);
```

- Accoda il di **`s2`** in **`s1`** (il **`'\0'`** appare solo alla fine)

Esempio di Copia Tra Stringhe

```
#include <string.h>
void main(){

int coincidono,len1,len2,flag;
char str1[30], str2[30], str3[30];
…

// copia in str3 il contenuto di str2
strcpy(str3,str2);
printf("\nrisultato copia str2 = %s e str3 =%s",
str2, str3);

// accoda
strcat(str3,str1);
printf("\naccodo str1 a str3: %s ", str3);
}
```