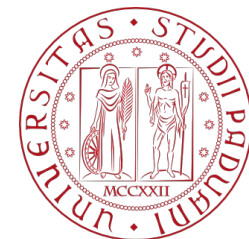


Tipi di Dato



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Tipi di Variabili



- Per gli interi abbiamo già visto come dichiarare diversi tipi di interi

Nome tipo in	Descrizione	Byte	Valore Min	Valore Max	formato in printf
int	intero	4	INT_MIN	INT_MAX	printf("%d", x)
long	intero che usa il doppio dei byte	8	LONG_MIN	LONG_MAX	printf("%ld", x)
short	intero che usa la metà dei byte	2	SHRT_MIN	SHRT_MAX	printf("%hd", x)
unsigned int	un intero positivo	4	0	UINT_MAX	printf("%u", x)
unsigned long	un long positivo	8	0	ULONG_MAX	printf("%lu", x)

- Per i reali abbiamo 2 opzioni
 - float o double (il secondo utilizza il doppio della memoria del primo)

- Lo standard di codifica più diffuso è il codice ASCII, per American Standard Code for Information Interchange
- La versione estesa definisce una tabella di corrispondenza fra ciascun simbolo (carattere minuscolo, maiuscolo, cifre) e un codice a 8 bit (256 caratteri)
- UNICODE (UTF-8 e UTF-16): standard proposto a 8 e 16 bit (65.536 caratteri)
- dichiarazione di una variabile carattere in C: `char x;`
- `char x = 'c';` //dichiarazione ed inizializzazione
- `printf("%c",x)`

Caratteri – ASCII Table



0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Caratteri – ASCII Table



0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

ordine alfabetico: 0<A<a

Caratteri – ASCII Table



0	NUL	16	DLE	32		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

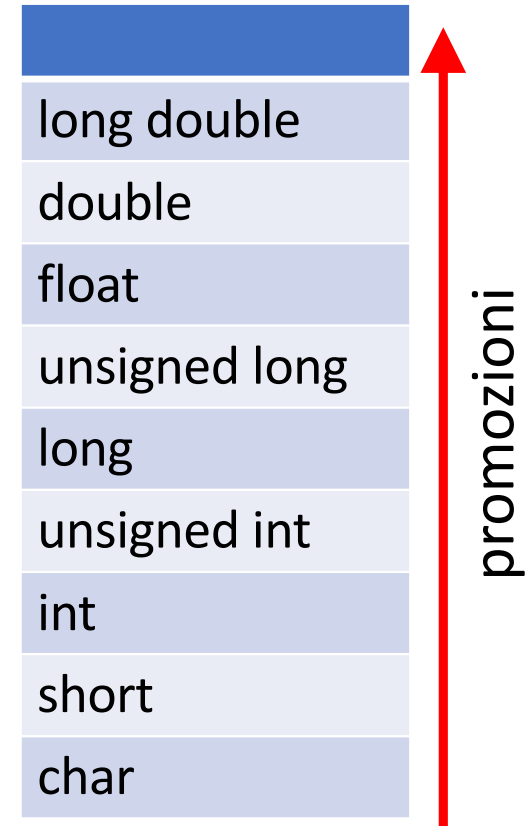
$a - A == b - B == 32$

```
1  #include <stdio.h>
2
3  int main() {
4
5      char ch='b';
6      printf("%c: ", ch);
7      if (ch>=97 && ch<=122) {
8          printf("lettera minuscola. ");
9          printf("Ecco la versione maiuscola: %c\n", ch-32);
10     } else
11         printf("non è una lettera minuscola\n");
12
13 }
```

Conversioni tra tipi



- Gli operatori aritmetici ed i comandi sono definiti tra termini dello stesso tipo
- Durante la valutazione di un'espressione, il C effettua automaticamente alcune conversioni tra tipi, le promozioni:
 - il tipo con minor capacità espressiva viene promosso al tipo con maggiore capacità espressiva
 - ovvero presi due elementi nella tabella a fianco, la conversione avviene da quello più in basso a quello più in alto
 - quando si cambia il tipo di una variabile in un'espressione, viene fatta una copia del valore temporanea per effettuare il calcolo (il cambio non ha effetto permanente sulla variabile)



Conversioni tra tipi



- Quando assegniamo il risultato di un'espressione ad una variabile, il C cerca di convertire il tipo dell'espressione a quello della variabile
- L'assegnamento di un'espressione di tipo float ad una variabile intera provoca il troncamento della parte decimale

```
int x;  
float y=4.6;  
x=y+3;  
printf("%d", x); // stampa 7
```

long double
double
float
unsigned long
long
unsigned int
int
short
char

Conversioni tra tipi



- Attenzione: alcune conversioni, per esempio tra int e unsigned int, possono produrre effetti inaspettati.

```
int g = -6
unsigned int ug = 3, r=g+ug;
printf("%u\n", r);
printf("%d\n", r);
```

- Stampa:
4294967293
-3

- E' possibile forzare la conversione ad un tipo con gli operatori di casting **(tipo)**. Es.

```
int x = 3 + ((int) 3.9);
printf("%d", x); // stampa 6!
```

- Attenzione che forzando la conversione, per esempio da float a int, si può perdere informazione!

Puntatori



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Definizione



- Gli indirizzi di memoria (L-valori) sono interi positivi.
- Un puntatore è una variabile il cui r-valore è un l-valore
- Es. `int x = 3;` //l-valore = 1021, r-valore = 3
la variabile con l-valore 1025 potrebbe essere un puntatore (vedi figura)
- I puntatori sono uno strumento di basso livello: ci permettono di manipolare altre variabili (altre celle di memoria)

1020	
1021	3
1022	
1023	
1024	
1025	1021
1026	0
1027	

Dichiarazione



- Dichiarazione: tipo *nome;
- Es. `int *ptr = NULL; // l-valore = 1026`
- `ptr` è una variabile di tipo “puntatore ad una variabile di tipo intero”.
- Per ogni Tipo di variabile, esiste il corrispondente tipo “puntatore a Tipo”
 - es. `float *x; char *c; long *l, ...`
 - Perché?
- Esistono anche puntatori a funzioni

1020	
1021	3
1022	
1023	
1024	
1025	1021
1026	0
1027	

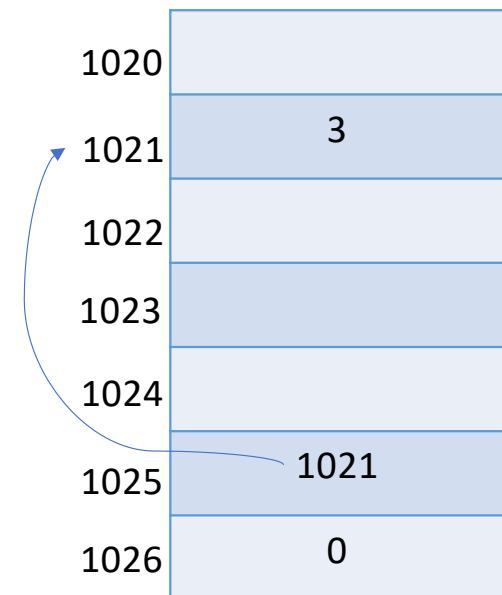
Operatori unari su variabili: & *



- &x (dove x è necessariamente una variabile) restituisce l'l-valore di x
- *p restituisce la variabile indicata dal r-valore del puntatore p (*p è una variabile a tutti gli effetti se p “punta” ad una variabile)
- & e * hanno precedenza minore di () [], ma maggiore degli operatori aritmetici; sono associativi da destra a sinistra

```
int *xptr = &x; // l-valore di xptr = 1025
printf("%p,%d,%p", xptr, *xptr, &x);
// stampa 1021, 3, 1021
```

nome	Indirizzo	tipo
x	$l_1=1021$	int
y	l_2	float



Esempi



```
int x = 3;  
int *xp; // l-valore=1025; r-valore=indefinito  
xp = &x; // r-valore=1021  
printf("%p", xp); // stampa l' r-valore di xp  
*xp = 4; printf("%d",x); // stampa 4; *xp=4 → x=4  
*xp = *xp+1; printf("%d",x); // stampa 5  
int *xp2 = &x; //ok, ad xp2 si assegna un l-valore  
           //equivale a int *xp2; xp2 = &x;  
           // NON a int *xp2; *xp2 = &x;  
int *xp3 = x; // errore di tipo
```

1020	
1021	3
1022	
1023	
1024	
1025	1021
1026	1021
1027	

```
int x = 3; int *xp2;
```

```
*xp2 = x; // xp2 è stata dichiarata ma non inizializzata, per cui l' r-  
valore
```

di xp2, *xp2, punta ad una cella di memoria con un valore indefinito nella quale cerchiamo di scrivere un intero, perciò ci aspettiamo (e speriamo in) un errore di **Segmentation Fault**.

- Come evitare il problema?

```
int *xp2 = NULL; //equivale a int *xp2 = 0, l-valore=1026
```

```
if (!xp2) {
```

```
    // puntatore non inizializzato
```

```
}
```


1. Non usare una variabile non inizializzata.
2. Controllare che il tipo della variabile sia corretto per l'operazione dove si sta per usare.
3. $\&*$ sono associativi da destra $\&*p = \&(*p)$. Hanno priorità minore di $()$, $[]$ ma maggiore di $*$ /
4. Memorizzare alla perfezione il significato degli operatori: per esempio $*$ restituisce la variabile (un oggetto che ha un nome, tipo, l-valore, r-valore) non il suo r-valore.
5. $*$ è usato con due significati apparentemente diversi 1) nella dichiarazione e 2) nel corpo di una funzione.

Puntatori ed Espressioni



- Se un puntatore è stato dichiarato ed inizializzato, può essere utilizzato in un espressione

```
int y, x=3, *p1=&x, *p2=p1;
```

```
y = *p1 * *p2; // (*p1) * (*p2)
```

```
y = x + *p2;
```

```
y = 5* - *p2/ *p1; // se non ci fosse stato lo spazio tra / e *?
```

Puntatori a Puntatori a Puntatori a....



- Si può definire un puntatore ad una variabile puntatore utilizzando più volte il simbolo * nella dichiarazione

```
int x, *p1;
```

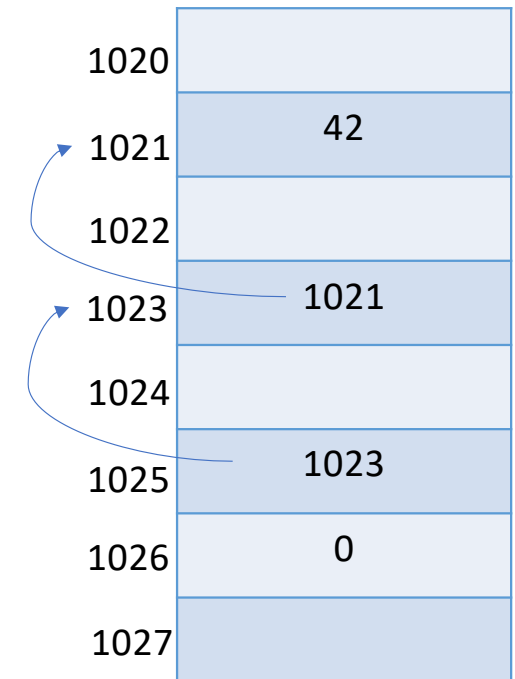
```
int **p2; // p2 è un puntatore ad una variabile
```

```
           //puntatore a variabile intera. L-valore p2=1025
```

```
x = 42; p1 = &x;
```

```
p2 = &p1;
```

```
printf("%d", **p2); //*( *p2) → *p1 → x
```



Dangling Reference



- Una funzione può restituire un puntatore
- Bisogna stare però attenti:
 - le variabili locali vengono deallocate (distrutte) al termine del blocco in cui sono dichiarate
 - per cui una funzione non deve restituire un puntatore ad un oggetto nel record di attivazione (parametro, variabile locale)
- anche somma viene deallocata (il contenuto della cella di memoria potrebbe essere utilizzato per creare altre variabili)
- s ha un valore indefinito (si chiama dangling reference=riferimento penzolante) → punta ad una variabile che non esiste più

```
int * somma(int x, int y) {  
    int somma;  
    somma = x+y;  
    return &somma;  
}
```

```
int main () {  
    int x=2, int y=3, int *s;  
    s = somma(x, y);  
    printf("%d", *s);  
}
```

- il tipo di una variabile puntatore deve corrispondere con il tipo della variabile puntata

```
int a=4;  
float *b;  
b=&a; // NO
```

```
int a=4;  
int *b;  
b=a; // NO
```

- Il C fornisce il comando `scanf` per ricevere un input da tastiera durante l'esecuzione.
- La sintassi ricorda quella di `printf`:
`int x; scanf("%d", &x);`
- nell'esempio legge un intero (%d) da tastiera. Notate che `scanf()` richiede l'indirizzo di x (L-valore di x)
- durante l'esecuzione in locale si dovrà digitare un intero e premere invio, quando si sottopone la soluzione su Moodle il sistema farà tutto automaticamente

Esercizio



```
#include <stdio.h>
int c = 2;
int main(void) {
int a = 3;
int *p;
{
a += 1;
int c = 4;
p = &c;
}
float c = 3.82;
int b = c;
printf("%d", b); printf(" %d", a); printf(" %d\n", *p);
}
```

Funzioni: Passaggio di Parametri



- Passaggio per valore: i valori dei parametri attuali vengono assegnati ai parametri formali al momento dell'invocazione della funzione.
 - La funzione (f) non modifica le variabili parametri attuali (y)

```
void f(int x) {  
    x = x + 1;  
}  
  
int y = 2;  
f(y);  
printf("%d", y); // stampa 2
```


Passaggio per Riferimento



- Il passaggio per riferimento permette di modificare all'interno della funzione i parametri attuali (y)
- In C è implementato solamente il passaggio per valore
- Gli effetti del passaggio per riferimento sono ottenuti passando per valore il puntatore ad una variabile.

```
void f(int *x) {  
    *x = *x + 1;  
}  
  
int y = 2;  
f(&y)  
printf("%d", x); // stampa 3
```

Passaggio per Riferimento: Esempi



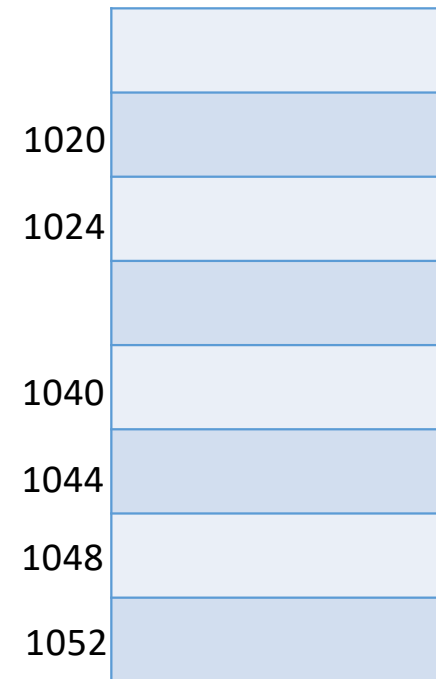
- Scambio di variabili

```
void scambio(int *x, int *y) {  
    int temp = *x;  
    *x = *y;  
    *y = temp;  
}  
  
int main() {  
    int x=2, y=5;  
    scambio(&x, &y);  
    printf("x=%d, y=%d\n", x,y); //stampa x=5, y=2  
}
```

Passaggio di Parametri in C



```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
    int temp = *x;
    *x = *y;
    *y = temp;
}
→ int main() {
    int w=2, z=5;
    scambio(&w, &z);
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```

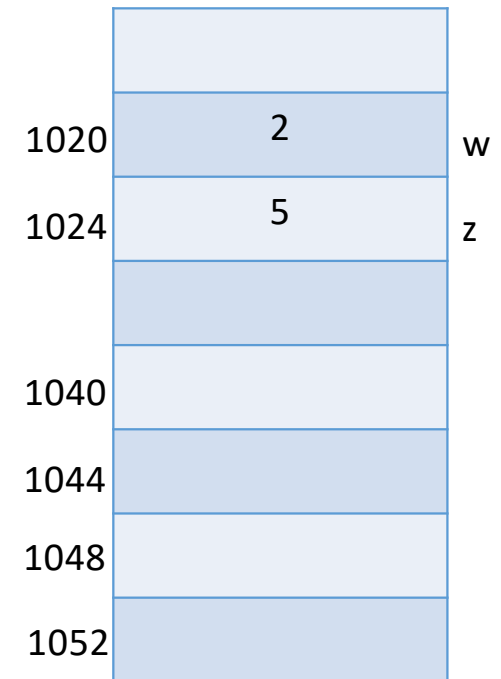


Passaggio di Parametri in C



```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    ➔ int w=2, z=5;
    scambio(&w, &z);
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```

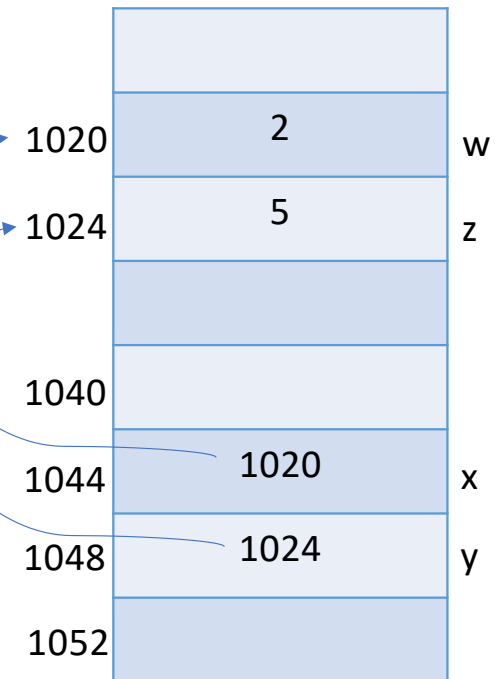


Passaggio di Parametri in C



```
→ void scambio(int *x, int *y) { // int *x = &w; int *y = &z
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int w=2, z=5;
    scambio(&w, &z);
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



Passaggio di Parametri in C



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
```

```
    int temp = *x;
```

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

```
    *y = temp;
```

```
}
```

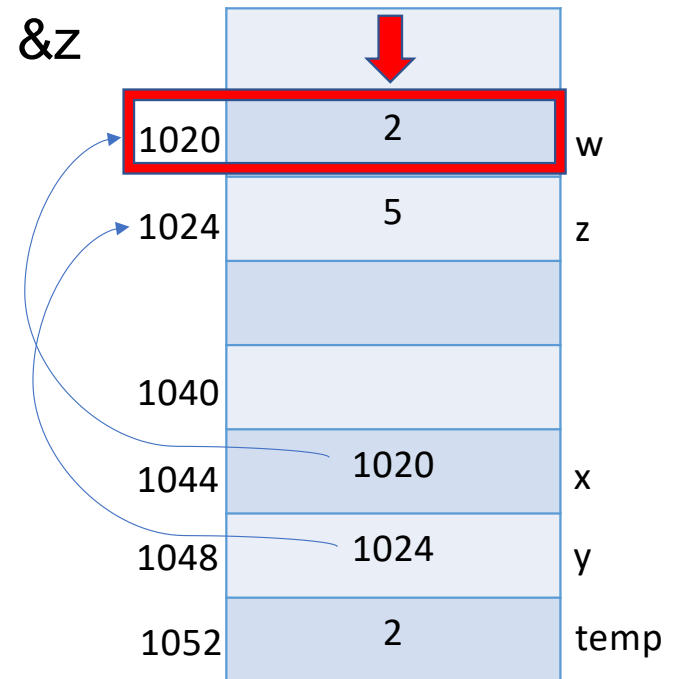
```
int main() {
```

```
    int w=2, z=5;
```

```
    scambio(&w, &z);
```

```
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
```

```
}
```



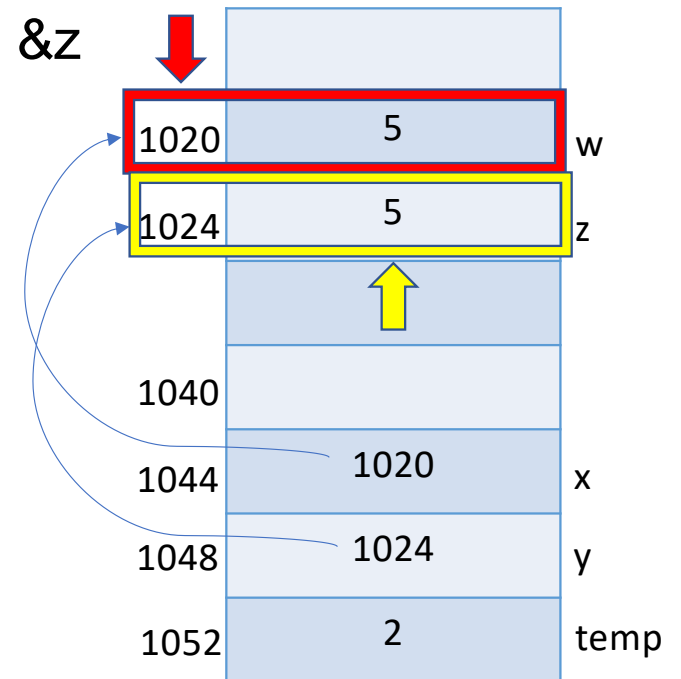
Passaggio di Parametri in C



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main() {
    int w=2, z=5;
    scambio(&w, &z);
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



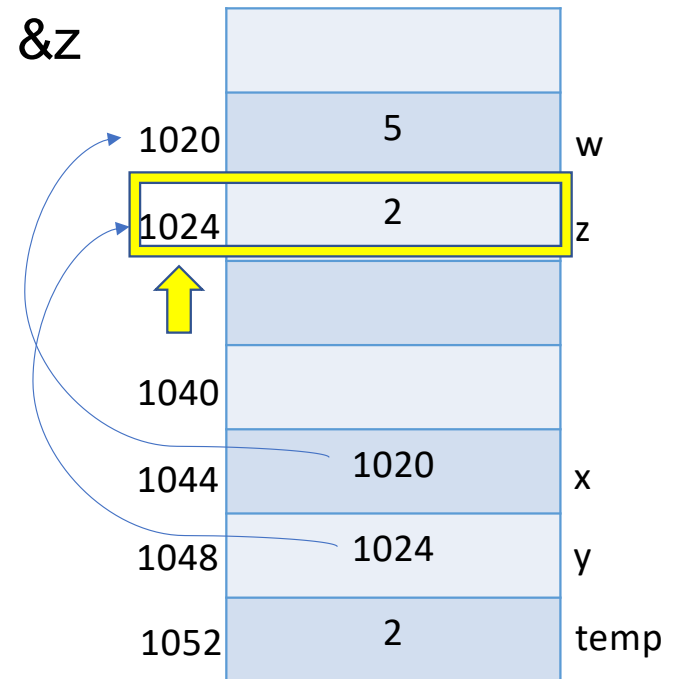
Passaggio di Parametri in C



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main() {
    int w=2, z=5;
    scambio(&w, &z);
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```



Passaggio di Parametri in C

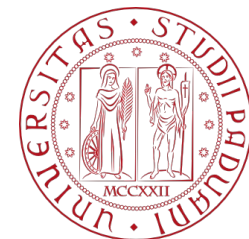


```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main() {
    int w=2, z=5;
    scambio(&w, &z);
    printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
}
```

1020	5	w
1024	2	z
1040		
1044		
1048		
1052		

Array



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Array



- Un array è un gruppo di variabili dello stesso tipo che hanno locazioni di memoria contigue.
- Dichiarazione: tipo nome[dimensione];
- Es. `int c[12];` // dichiara un array (una sequenza) di 12 variabili intere
- `c[i]` si comporta come una variabile di tipo intero
- `c[i]` accede all'i-esimo elemento dell'array (si usa la parola indice per riferirsi al numero tra `[]`):
 - Il primo elemento ha indice 0
 - l'ultimo ha indice dimensione-1 (11 nel nostro esempio)
- Es. `printf("secondo elemento di c=%d", c[1])`
- `c[2] = 1;` // il valore del terzo elemento dell'array q destra passa da 0 a 1

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Array: Esempi



- ```
int n[5];
for (int i = 0; i < 5; i=i+1) { // inizializza a zero gli elementi dell'array
 n[i] = 0;
}
```

```
for (int i = 0; i < 5; i=i+1) { // stampa gli elementi dell'array
 printf("%d = %d\n", i, n[i]);
}
```
- ```
int n[5] = {32, 27, 64, 18, 95}; // dichiara ed inizializza l'array  
int n[] = {32, 27, 64, 18, 95}; // dichiarazione equivalente
```

Confini di un Array



- Il C non ha meccanismi di controllo dei confini di un array
- Un programma può fare riferimento a un elemento che non esiste e non ricevere un errore!



- ```
int n[5] = {32, 27, 64, 18, 95};
for (int i = 0; i < 6; i=i+1) {
 printf("%d = %d\n", i, n[i]);
}
```
- Nell'ultima iterazione si stampa il contenuto della cella di memoria seguente all'array; se siamo fortunati questo genera un errore, altrimenti viene stampato un valore indefinito (859 nell'esempio)

|      |     |
|------|-----|
|      |     |
| n[0] | 32  |
| n[1] | 27  |
| n[2] | 64  |
| n[3] | 18  |
| n[4] | 95  |
|      | 859 |
|      |     |

# Array a Dimensione Fissata



- In C89 la dimensione di un array deve essere una costante intera positiva (no variabile `const int`) nota a tempo di compilazione (non una variabile, letta da tastiera o meno)
- Di norma la costante si indica tramite `#define`
  - `#define DIM_ARRAY 3`
  - `int x[DIM_ARRAY] = {1,2,3} //OK`
- Dal C99 è possibile dichiarare array la cui dimensione non è fissata a tempo di compilazione (si chiamano VLA, variable length arrays)
- Tali array sono implementati internamente in modo diverso dal C. Una delle differenze è che non possono essere inizializzati
  - `int n = 3; int x[n] = {1,2,3}; //NO`
  - `x[0]=1; x[1]=2; x[2]=3; //OK`

# Stringhe (Sequence di Caratteri)



- Una sequenza di caratteri che termina con il carattere `'\0'` (tra apici singoli, non doppi) viene chiamata stringa. Es. `char s[] = {'c', 'i', 'a', 'o', '\0'}` è una stringa, ma anche `char s2[] = {'c', 'i', 'a', 'o', '\0', 'a', 'a'}` indica la stessa stringa.
- `char string1[] = "ciao mondo";` // il carattere `\0` viene aggiunto  
// automaticamente ("" indicano una stringa)

```
printf("la stringa inizia per %c\n", string1[0]); // stampa c
```

```
printf("%s\n", string1); // stampa ciao mondo, %s sta per stringa, assume che
l'array di caratteri termini con \0
```

```
printf("%s\n", s2); // stampa ciao
```

# Array di Caratteri e Puntatori a Stringhe



Una stringa può essere rappresentata da

- un array di caratteri, se si aggiunge '\0' alla fine.
  - La dichiarazione riserva un certo numero di celle di memoria. Quindi Si riesce a modificare un singolo carattere (s[0]='K')
- un puntatore a char, a cui può essere assegnata una stringa costante, della quale non si possono modificare i caratteri. Si può riassegnare una seconda stringa al puntatore.

```
char s[8] = "Hearts";
char *ps = "Hearts";
printf("%s - %s\n\n", s, ps);
s[0] = 'K';
ps = "Ke";
printf("%s - %s\n", s, ps);
--Output--
Hearts - Hearts

Kearts - Ke
```



# Passaggio di Parametri in C: Array



- I due prototipi seguenti sono equivalenti ed intercambiabili:  

```
void f(int *array);
```

```
void f(int array[]);
```
- `int x[5];`
- `x == &x[0]` // puntatore al primo elemento dell'array
- `int *p = x;` // corretto, equivale a `&x[0]`
- `f(x); f(p);` // sono corrette per entrambi i prototipi
- Gli elementi di un array vengono sempre modificati all'interno di una funzione! Perché?

- Si può sommare o sottrarre un intero ad un puntatore
  - `int x, *p = &x;`
  - `p+2` equivale a `p+2*sizeof(int)`
- Si può sottrarre due puntatori (ha senso se si riferiscono allo stesso array)
  - `p1-p2`
- Si possono confrontare due puntatori:
  - `p1>p2`, `p1==p2`, `p1 != p2`
- Non è ammesso utilizzare puntatori in moltiplicazioni o divisioni
  - `p1/3`, `p1/p2`, `p1*p2`
- Non è ammesso sommare due puntatori
  - `p1+p2`

# Array e Puntatori



- `int x[] = {1,2,3,4,5}`
- quando usato in un'espressione, `x` è un puntatore **costante** al primo elemento dell'array:
  - `x == &x[0]`
- `int *p = x; // corretto, equivale a &x[0]`  
//assumendo che L-valore di `x` sia 1022  
`&x[0] == p == 1022`  
`&x[1] == p+1 == 1022 + 1*sizeof(int) == 1026`  
`&x[2] == p+2 == 1022 + 2*sizeof(int) == 1030`  
`&x[3] == p+3 == 1022 + 3*sizeof(int) == 1034`

|         |      |
|---------|------|
| c[ 0 ]  | -45  |
| c[ 1 ]  | 6    |
| c[ 2 ]  | 0    |
| c[ 3 ]  | 72   |
| c[ 4 ]  | 1543 |
| c[ 5 ]  | -89  |
| c[ 6 ]  | 0    |
| c[ 7 ]  | 62   |
| c[ 8 ]  | -3   |
| c[ 9 ]  | 1    |
| c[ 10 ] | 6453 |
| c[ 11 ] | 78   |

# Esempio: Stampa Array



```
int a[] = {1,2,3,4};
int i = 0, *p = a;
while(i<4) {
 printf("%d\n", *p);
 i+=1;
 p = p+1;
}
```

Stampa:

1  
2  
3  
4

# Esempio: Stampa Array (versioni 2 - 3)



```
int a[] = {1,2,3,4};
int i = 0, *p = a;
while(i<4) {
 printf("%d\n", *(p+i));
 i+=1;
}
```

// Oppure

```
while(i<4) {
 printf("%d\n", *(a+i));
 i+=1;
} //ma a+=i sarebbe sbagliato!
```

Entrambi Stampano:

1

2

3

4

# Array Come Parametri di Funzioni



- Quando passiamo un array ad una funzione, in realtà passiamo per valore il puntatore al primo elemento dell'array
- È quindi possibile modificare gli elementi di un array passato come parametro ad una funzione

```
void successivo(int ar[], int N) {
 for(int i=0; i<N; i+=1) {
 ar[i] += 1;
 }
}
```

```
int main (void) {
 int a[]={1,2,3};
 successivo(a,3);
 //a={2,3,4}
}
```

# Array Come Parametri di Funzioni



```
void successivo(int ar[], int N) {
 //ar=a; N=3;
 for(int i=0; i<N; i+=1) {
 ar[i] += 1;
 }
}
```

```
int main (void) {
 int a[]={1,2,3};
 successivo(a,3);
 //a={2,3,4}
}
```

- successivo(a,3);  
//ar=a  
ar[0]+=1 /\*  
modifica la  
variabile  
all'indirizzo  
1008\*/

|      |      |      |
|------|------|------|
|      |      |      |
| 1004 | 1    | a[0] |
| 1008 | 2    | a[1] |
|      | 3    | a[2] |
|      | 1004 | a    |
|      | 1004 | ar   |
|      | 3    | N    |
|      |      |      |

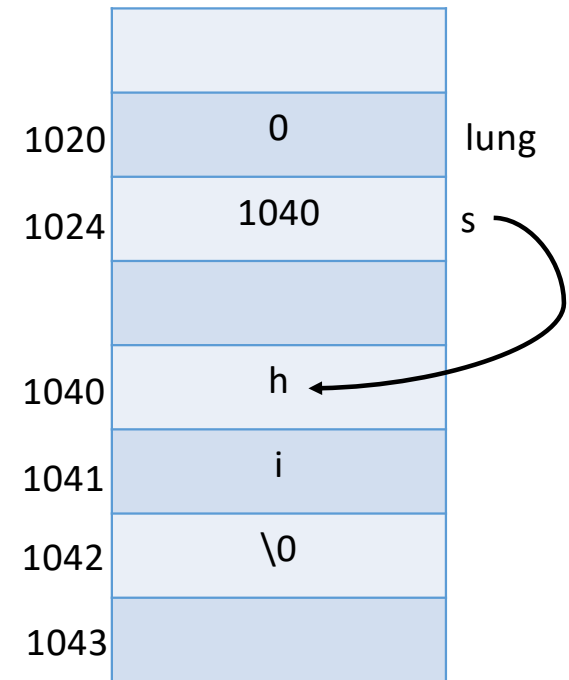
# Esempio: Lunghezza Stringa



- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```

Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.





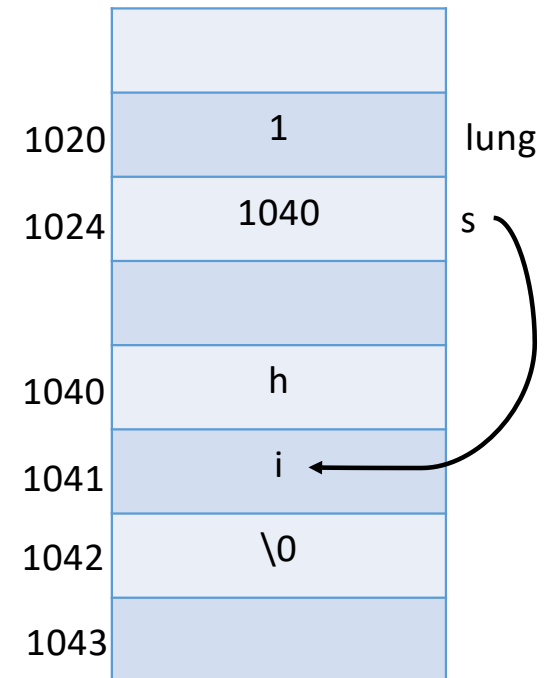
# Esempio: Lunghezza Stringa



- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```

Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.



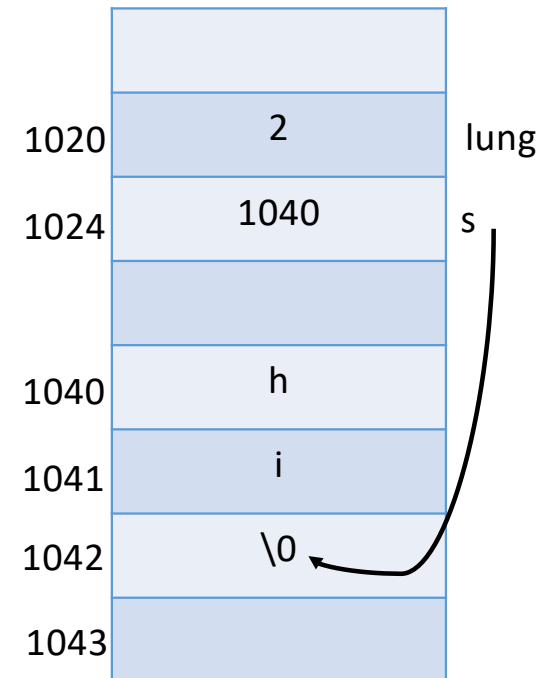
# Esempio: Lunghezza Stringa



- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```

Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.



# Esempio: Lunghezza Stringa



- Calcolare la lunghezza di una stringa:

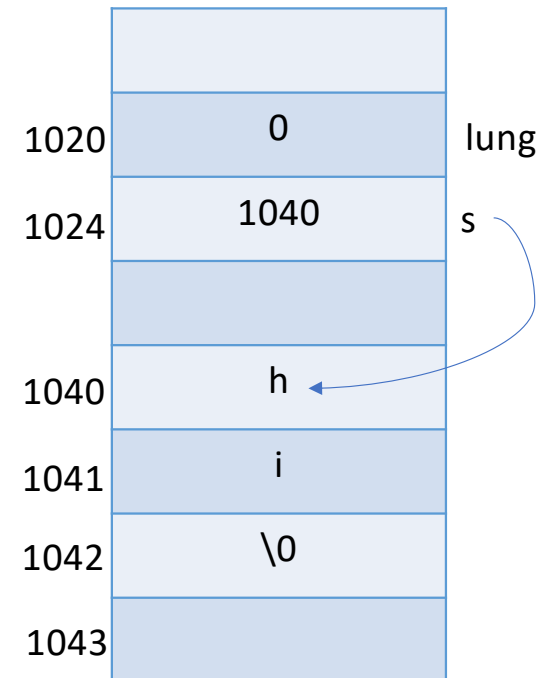
```
char *s="hi";
```

```
int l;
```

```
for(l=0; *s!='\0'; s+=1)
```

```
 l+=1;
```

```
printf("lunghezza stringa: %d\n", l);
```

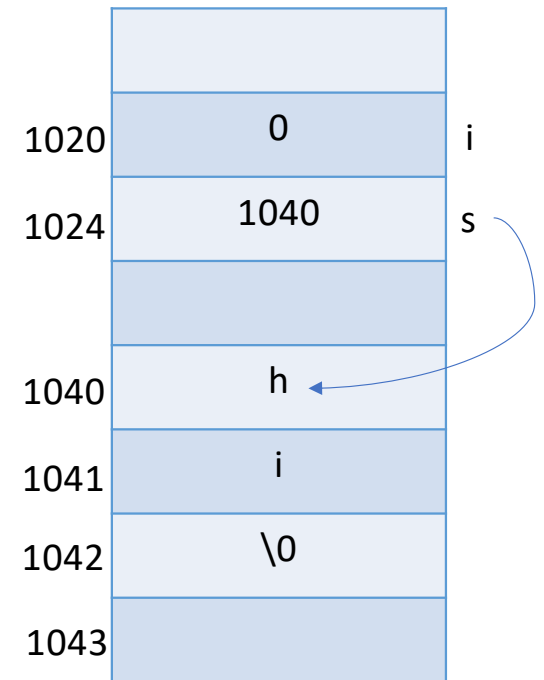


# Esempio: Lunghezza Stringa



- Calcolare la lunghezza di una stringa:

```
char *s="hi";
```



Cosa Stampa?

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

```
int main() {
 int x;
 int y=2;
 int *p, *q = &y;
 int **qq =&p;
 **qq = 3;
 printf("%d\n",**qq);
}
```

Cosa Stampa?

```
#include<stdio.h>

int main() {
 int y=2;
 int **qq =&&y;
 printf("%d\n", **qq);
}
```

- Vero – Falso: una variabile puntatore ad intero occupa la stessa quantità di memoria di una variabile di tipo puntatore a double.

- Vero – Falso: una variabile puntatore ad intero occupa la stessa quantità di memoria di una variabile di tipo puntatore a double.

Vero: la variabile puntatore ha come R-valore un L-valore, cioè l'indirizzo di memoria della cella dove inizia la rappresentazione del dato puntato, che è indipendente dal numero di celle che serve a rappresentare il dato puntato.



- Cosa stampa il codice seguente?

```
void fun(int* a){
 a[1]=a[1]*2;
 a[2]=a[2]*2;
}

int main(void) {
 int x[]={0,1,2,3,4};
 fun(x+2);
 for(int i=0; i<5; i+=1) {
 printf(" %d", x[i]);
 }
 printf("\n");
}
```

- Cosa stampa il codice seguente?

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

```
void quadrato(int x) {
```

```
 x = x*x;
```

```
}
```

```
int main (void) {
```

```
 int x[3] = {1,2,3};
```

```
 for(int i=0; i<3; i+=1) {
```

```
 quadrato(x[i]);
```

```
 printf(" %d", x[i]);
```

```
 }
```

```
}
```

# Dichiarazioni di Array



- Le dichiarazioni in C possono essere piuttosto convolute
- Per “risolvere” una dichiarazione si parte dal nome della variabile e ci si sposta a destra/sinistra a seconda della priorità degli operatori
- `int a[5];` //a è un array di 5 interi (a è un array di 5 elementi, ciascuno di questi è un intero)
- `int *p[4] ?`
- `int (*p)[4] ?`

# Dichiarazioni di Array



- `int *p[4]` ?
- poiché `[]` ha priorità su `*`, `int *p[4]` si legge: `p` è un array di 4 elementi (ciascuno) `int *`, ovvero un array di 4 puntatori ad intero
- `int (*p)[4]` ?
- `int (*p)[4]` si legge: `p` è un puntatore ad un array di 4 interi