Tipi di Dato



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)

Caratteri



- 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	ര	80	Р	96	•	112	р
1	SOH	17	DC1	33	!	49	1	65	Α	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	В	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	С	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	Τ	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	Е	85	U	101	е	117	u
6	ACK	22	SYN	38	8	54	6	70	F	86	٧	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	Н	88	Χ	104	h	120	Х
9	HT	25	EM	41)	57	9	73	Ι	89	Υ	105	i	121	у
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	ι	124	1
13	CR	29	GS	45	-	61	=	77	М	93]	109	m	125	}
14	S0	30	RS	46		62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	0	95	_	111	0	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	5 A	81 Q	97 a	113 q
2	STX 18	DC2	34 "	50 2	66	5 B	82 R	98 b	114 r
3	ETX 19	DC3	35 #	51 3	67	7 C	83 S	99 c	115 s
4	EOT 20	DC4	36 \$	52 4	68	B D	84 T	100 d	116 t
5	ENQ 21	NAK	37 %	53 5	69	ΡE	85 U	101 e	117 u
6	ACK 22	SYN	38 &	54 6	76) F	86 V	102 f	118 v
7	BEL 23	ETB	39 '	55 7	71	L G	87 W	103 g	119 w
8	BS 24	CAN	40 (56 8	72	2 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	5 K	91 [107 k	123 {
12	FF 28	FS	44 ,	60 <	76	i L	92 \	108 l	124
13	CR 29	GS	45 -	61 =	77	7 M	93]	109 m	125 }
14	SO 30	RS	46 .	62 >	78	3 N	94 ^	110 n	126 ~
15	SI 31	US	47 /	63 ?	79	0	95 _	111 o	127 DEL

ordine alfabetico: 0<A<a

Caratteri – ASCII Table



0 NUL	16 DLE 32	48 0	64 a 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 E0T	20 DC4 36 \$	52 4	68 D 84 T 100 d 116 t
5 ENQ	21 NAK 37 %	53 5	69 F 85 II 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 S0	30 RS 46 .	62 >	78 N 94 ^ 110 n 126 ~
15 SI	31_US 47 /	63 ?	79 0 95 _ 111 o 127 DEL

Caratteri



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

promozioni

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)

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

Conversioni tra tipi



- Quando assegnamo 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



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)

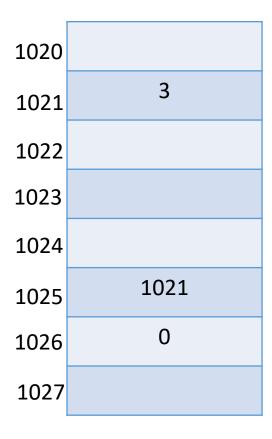
1020	
1021	3
1022	
1023	
1024	
1025	1021
1026	0
1027	

Dichiarazione



- Dichiarazione: tipo *nome;
- Es. int *ptr = NULL; // I-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



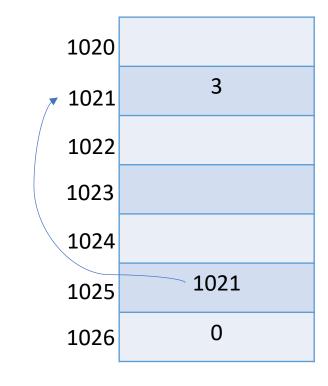
Operatori unari su variabili: & *



- &x (dove x è necessariamente una variabile) restituisce l'I-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	I ₁ =1021	int
У	l ₂	float



Esempi



```
int x = 3;
int *xp; // I-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 \rightarrowx=4
*xp = *xp+1; printf("%d",x); // stampa 5
int *xp2 = &x; //ok, ad xp2 si assegna un I-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	

Esempi



```
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, I-valore=1026
if (!xp2) {
     // puntatore non inizializzato
```

Variabili – Manuale d'Uso



- 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;

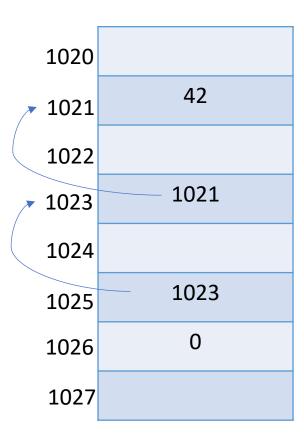
int **p2; // p2 è un puntatore ad una variabile

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

x = 42; p1 = &x;

p2 = &p1;

printf("%d", **p2); //*(*p2) \rightarrow *p1 \rightarrow 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);
```

Puntatori



• 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
```

Input da Tastiera



- 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
```

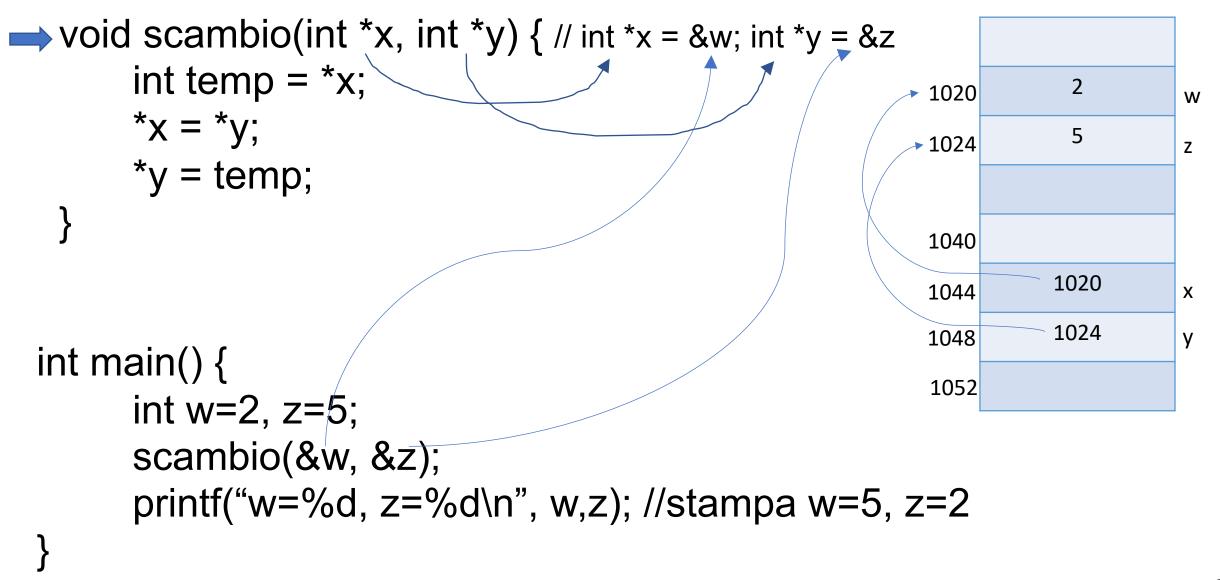


```
void scambio(int *x, int *y) {// int *x = &w; int *y = &z
         int temp = *x;
                                                           1020
         *x = *y;
                                                           1024
         *y = temp;
                                                           1040
→ int main() {
                                                           1044
                                                           1048
         int w=2, z=5;
                                                           1052
         scambio(&w, &z);
          printf("w=\%d, z=\%d\n", w,z); //stampa w=5, z=2
```



```
void scambio(int *x, int *y) {// int *x = &w; int *y = &z
      int temp = *x;
                                                        1020
                                                                         W
      *x = *y;
                                                        1024
                                                                         Ζ
      *y = temp;
                                                        1040
int main() {
                                                        1044
                                                        1048
     int w=2, z=5;
                                                        1052
      scambio(&w, &z);
      printf("w=\%d, z=\%d\n", w,z); //stampa w=5, z=2
```







```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
      int temp = *x;
                                                            1020
      *x = *y;
                                                           ▶ 1024
                                                                              Ζ
      *y = temp;
                                                            1040
                                                                     1020
                                                            1044
                                                                             Χ
                                                                     1024
                                                            1048
                                                                              У
int main() {
                                                                             temp
                                                            1052
      int w=2, z=5;
      scambio(&w, &z);
      printf("w=\%d, z=\%d\n", w,z); //stampa w=5, z=2
```



```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
      int temp = *x;
                                                            1020
      *x = *y;
                                                            1024
      *y = temp;
                                                            1040
                                                                    1020
                                                            1044
                                                                             Χ
                                                                    1024
                                                            1048
                                                                             У
int main() {
                                                                             temp
                                                            1052
      int w=2, z=5;
      scambio(&w, &z);
      printf("w=\%d, z=\%d\n", w,z); //stampa w=5, z=2
```



```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
      int temp = *x;
                                                            1020
                                                                             W
      *x = *y;
                                                            1024
      *_{y} = temp;
                                                            1040
                                                                     1020
                                                            1044
                                                                             Χ
                                                                     1024
                                                            1048
                                                                             У
int main() {
                                                                             temp
                                                            1052
      int w=2, z=5;
      scambio(&w, &z);
      printf("w=%d, z=%d\n", w,z); //stampa w=5, z=2
```



```
void scambio(int *x, int *y) { // int *x = &w; int *y = &z
      int temp = *x;
                                                           1020
                                                                           W
      *x = *y;
                                                           1024
                                                                           Ζ
      *y = temp;
                                                          1040
                                                          1044
                                                          1048
int main() {
                                                           1052
      int w=2, z=5;
      scambio(&w, &z);
      printf("w=\%d, z=\%d\n", w,z); //stampa w=5, z=2
```

Array



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
[10]	6453
[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]);
}</pre>
```

 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)

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

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. Ma si può riassegnare una seconda stringa al puntatore (e.g. comando ps="Ke").

```
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?

Aritmetica dei Puntatori



- 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

$$&x[0] == p == 1022$$

$$&x[1] == p+1 == 1022 + 1*sizeof(int) == 1026$$

$$&x[2] == p+2 == 1022 + 2*sizeof(int) == 1030$$

$$&x[3] == p+3 == 1022 + 3*sizeof(int) == 1034$$

-45
6
0
72
1543
-89
0
62
-3
1
6453
78

Esempio: Stampa Array



```
int a[] = \{1,2,3,4\};
int i = 0, *p = a;
   while(i<4) {</pre>
      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;
   }
}</pre>
```

```
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;
    }
}</pre>
```

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

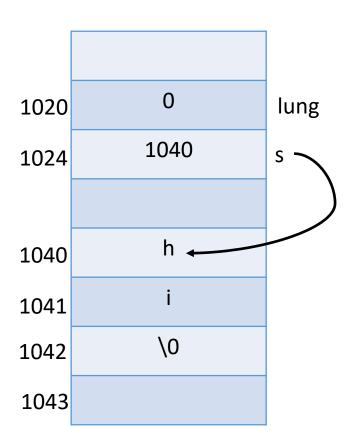
successivo(a,3);
//ar=a
ar[0]+=1 /*
modifica la
variabile
all'indirizzo
1008*/

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



Calcolare la lunghezza di una stringa:

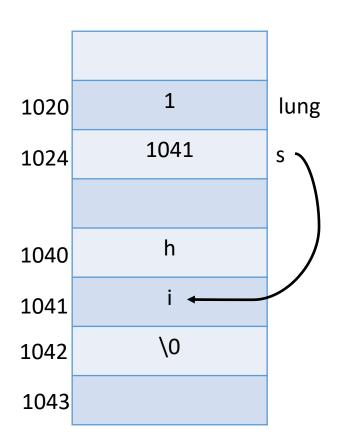
Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.





Calcolare la lunghezza di una stringa:

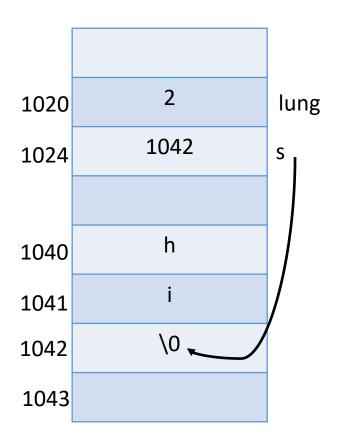
Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.





Calcolare la lunghezza di una stringa:

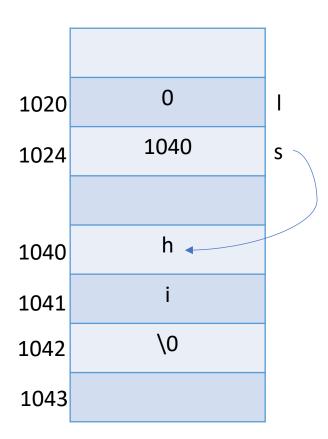
Idea: spostare il puntatore s fino alla fine della stringa, un carattere alla volta, contando il numero di spostamenti effettuati.





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

int *p non viene inizializzata. Quando eseguo l'assegnamento **q=3, il primo *q mi porta alla variabile p (perché qq viene inizializzato con l'indirizzo di p), il secondo *(*q) mi porterebbe alla variabile indicata dall'R-valore di p, che però, non essendo inizializzato, è indefinito. Quindi è un pessimo esempio di programmazione. Posso accadere 2 cose:

- 1) l'R-valore di p indica una cella di memoria accessibile al programma, per cui viene stampato 3
- 2) l'R-valore di p punta ad una cella di memoria non accessibile, il programma va in crash



```
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

Dichiarazioni di Array Multipli



- int p[4][2]?
- p è un array di 4 elementi, di cui ciascuno è un array di 2 interi
- il tipo di p è int[4][2]



- Gli array possono avere più di una dimensione
 - Es. per rappresentare tabelle, matrici, oggetti multidimensionali

tipo nome[indice1][indice2]...

• Es. int x[3][4];

	Colonna 0	Colonna 1	Colonna 2	Colonna 3
Riga 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Riga 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Riga 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Indice di riga

Nome dell'array

- int ar[2][3] = $\{\{1, 2, 3\}, \{4, 5, 6\}\}$;
- int ar[2][3] = {1,2,3,4,5,6}; //identiche
- // il [3] indica come "raggruppare per righe"



- Inizializzazione:
- int ar[2][3] = { {1, 2, 3}, {4, 5, 6} }; // = {1,2,3,4,5,6} es. a[1][0] {0 1 1 }

1020			
1021	1	ar[0][0]	
1022	2	ar[0][1]	
1023	3	ar[0][2]	
1024	4	ar[1][0]	
1025	5	ar[1][1]	
1026	6	ar[1][2]	
1027			

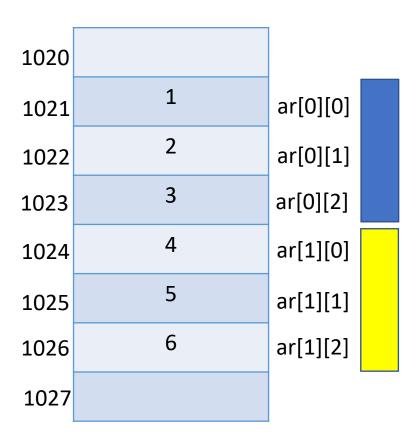


Inizializzazione:

1020			
1021	1	ar[0][0]	
1022	2	ar[0][1]	
1023	3	ar[0][2]	
1024	4	ar[1][0]	
1025	5	ar[1][1]	
1026	6	ar[1][2]	
1027			

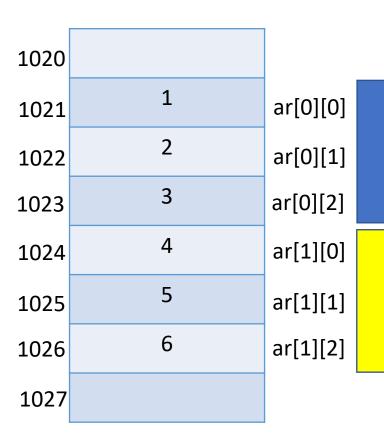


Inizializzazione:





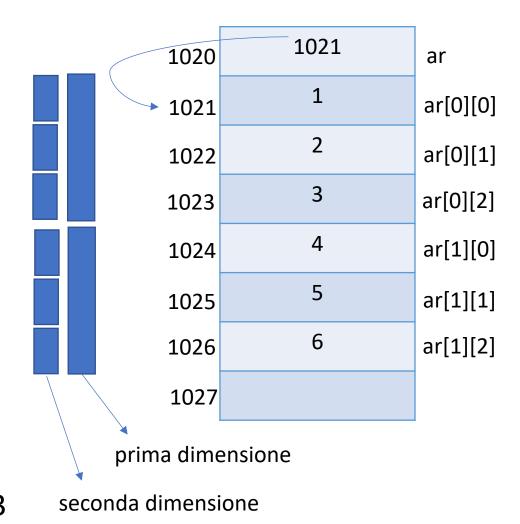
- Inizializzazione:
- int ar[2][3] = { {1, 2, 3}, {4, 5, 6} }; es. a[1][0] {0 1 2
- Anche gli array multidimensionali sono salvati come una sequenza contigua di variabili, riga per riga. Le seguenti inizializzazioni sono equivalenti:
- int $ar[2][3] = \{1, 2, 3, 4, 5, 6\};$
- int ar[][3] = {1, 2, 3, 4, 5, 6}; // solo la prima dimensione può essere lasciata in bianco



Array Multidimensionali in Memoria



- int ar[][3] = {1, 2, 3, 4, 5, 6}; // solo la prima dimensione può essere lasciata in bianco
- Perché le altre devono esserci? Se accedo a ar[1][0] devo sapere che devo saltare 3 elementi (ar è solo un puntatore al primo elemento dell'array)!
 - int ar[][3]; ar[1][2]==6 è l'elemento in posizione
 1*3 + 2 in memoria (a partire da ar)
 - int ar[][3]; int *b=(int*)ar; ar[1][2]==b[1*3 + 2]
- Gli stessi ragionamenti si applicano ad array di più dimensioni: int X[6][3][4];
 - $X[1][2][3] \rightarrow$ elemento in posizione 1*3*4 + 2*4 + 3



Matrici: Aritmetica dei Puntatori



- int c[12]; definisce 12 variabili in memoria
- c[i] viene implementato dal C come *(c+i*sizeof(int))
- notate che c ha tipo int[12], ma per eseguire c+i, c viene trasformato in int* (puntatore al tipo di ciascun elemento di c)
- In generale c[i] →*(c+i*sizeof(TIPO DI CIASCUN ELEMENTO DI c))
- se passiamo c come parametro ad una funzione, il C lo trasforma in un puntatore ad intero
 - int f(int a[]); int f(int *a); sono equivalenti; nel main si invoca con f(c);
 - in generale c viene trasformato ad un puntatore al tipo di ogni c[i]
- int c[3][4]; definisce una matrice di 3 righe, ciascuna con 4 elementi
- sappiamo che c[2] equivale a *(c+2*4); per eseguire l'operazione il C deve sapere che la matrice c ha 4 colonne, questa informazione deve essere mantenuta nel tipo di c

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

Matrici: Aritmetica dei Puntatori



- In generale $c[i] \rightarrow *(c+i*sizeof(TIPO DI c[i]))$
- int c[3][4]; definisce una matrice di 3 righe, ciascuna con 4 elementi
- il tipo di c è int[3][4], il tipo di c[i] è int[4], un puntatore a c avrebbe tipo int(*)[4], un puntatore a c[i] avrebbe tipo int *
- come possiamo trasformare c[2][1] utilizzando il fatto che c[i] = *(c+i*sizeof(TIPO ELEMENTO DI c))?
- c[2] \rightarrow *(c+2*sizeof(int[4])) = *(c+2*4*4) \rightarrow il tipo di c, per eseguire c+32 diventa int(*)[4] (puntatore a int[4]) \rightarrow *(c+32) dereferenzia la variabile 32 byte dopo l'indirizzo di c ottenendo una variabile di tipo int[4], c[2]

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

Matrici: Aritmetica dei Puntatori



- c[2] → *(c+2*sizeof(int[4])) = *(c+2*4*4) → il tipo di c, per eseguire c+32 diventa int(*)[4] (puntatore a int[4]) → *(c+32) dereferenzia la variabile 32 byte dopo l'indirizzo di c ottenendo una variabile di tipo int[4], c[2]
 - (cioè un array unidimensionale di tipo int[4])
 - int (*b)[4] = c+32; // *b equivale a c[2]
- $c[2][1] \rightarrow (c[2])[1] = *(c+2*sizeof(int[4]))[1] = (*(c+32))[1] = (*b)[1] = *(b+1*sizeof(int)) = *(c+32+4) = 1$

*b è di tipo int[4] quindi per sommare *b+1 uso il puntatore a *b, ovvero b *b è di tipo int[4] quindi (*b)[i] è di tipo int

) · 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

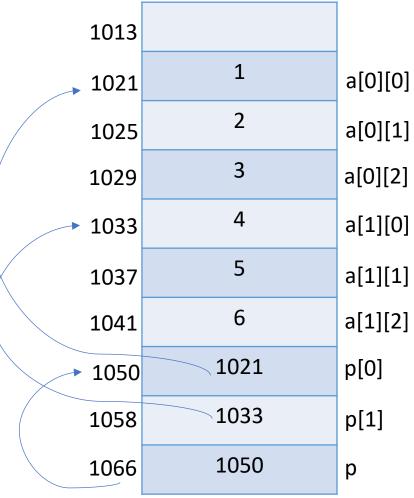
Matrici come Array di Puntatori



```
• int a[][3] = \{ \{1,2,3\}, \{4,5,6\} \};
```

• int $*p[] = \{a[0], a[1]\};$

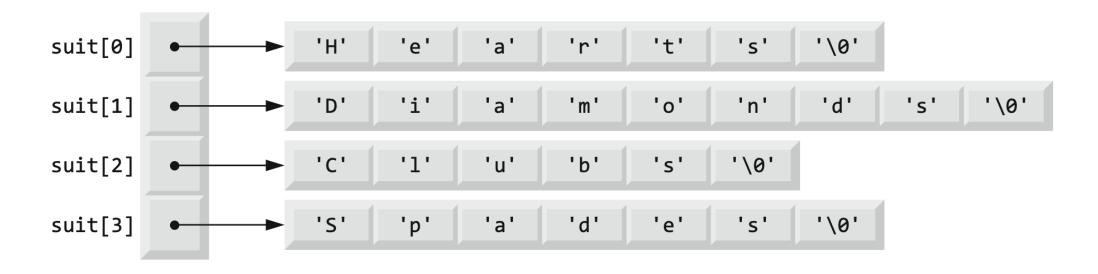
Tramite p ho rappresentato una matrice come un array di puntatori a vettori. Notate che ora potrei avere righe di dimensione diversa (ma dovrei ricordarmi la dimensione di ogni riga)



Array Multidimensionali e Puntatori



- Gli array di puntatori sono più generali, permettono di avere righe di dimensione diversa:
- char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};



Passaggio di Parametri: Array Multidimensionali



• int a[2][3] = { {1, 2, 3}, {4, 5, 6} };

- void stampa1(int a[][3], int righe);
- void stampa2(int** a, int righe, int colonne);
- void stampa3(int* a, int righe, int colonne)

```
int *p[] = {a[0], a[1]};
int *pp = a[0];
stampa1(a,2);
stampa2(p,2,3);
stampa3(pp,2,3);
```

Array Multidimensionali



- In pratica la cosa più complessa è azzeccare i tipi delle variabili, specialmente quando le passiamo come parametri ad una funzione.
- se abbiamo un puntatore ad un array unidimensionale, ad es. un int*, dobbiamo utilizzare *(a+i*colonne+j) per ottenere il valore dell'elemento a(i,j)
 - notate che non dovete usare sizeof, viene aggiunto automaticamente dal C
 - se l'array è tridimensionale, l'elemento a(i,j,k) è *(a+i*righe*colonne + j*colonne + k)
- se abbiamo una struttura a "più livelli",
 - int a[righe][colonne] = { {...}, {...}, ..., {...} } oppure
 - int*p[righe] = {a[0], a[1], ..., a[righe-1]}
 - possiamo semplicemente usare la notazione a[i][j] (o a[i][j][k]), dove il primo indice, i, seleziona il vettore riga ed il secondo, j, l'elemento all'interno del vettore

Esercizio: cosa stampa?



```
#include <stdio.h>
int * f(int x) {
  int a[3] = \{3,4,5\};
  a[x]+=x;
  return a;
int m[3][2]=\{1,2,3,4,5,6\};
int x = 2;
```

```
int main(void) {
  int *p = (int *) m;
    int x = 1;
    int *h = f(x);
    printf("%d\n", h[1]);
  printf("%d\n", *(p+x));
  printf("%d\n", *(*(m+2)+1));
```

Artimetica dei Puntatori: Esercizi



- Date le seguenti matrici, trasformare l'operazione a[i][j] nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
- int *p = &a[0][0]; a[i][j]=*(p +)

- 1. int a[3][4]; a[2][1]
- 2. float b[2][4]; b[2][3]
- 3. int c[3][5][4]; c[2][1][3]

<u>Artimetica dei Puntatori: Esercizi</u>



- Date le seguenti matrici, trasformare l'operazione a[i][j] nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
- int *p = &a[0][0]; a[i][j]=*(p +)
- 1. int a[3][4]; int *p = &a[0][0]; a[2][1] \rightarrow *(p+2*4+1)
- 2. float b[2][4]; int *p = &b[0][0]; b[2][3] \rightarrow *(p+2*4+3) \rightarrow fuori dai limiti dell'array
- 3. int c[3][5][4]; int *p = &c[0][0]; c[2][1][3] \rightarrow *(p+2*5*4+1*4+3)

Artimetica dei Puntatori: Esercizi



- Date le seguenti matrici, trasformare l'operazione a[i][j] nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
- int *p = &a[0][0]; a[i][j]=*(p +)

- 1. int a[4][5]; p = &a[0][0]; a[2][1] $\rightarrow *(p+...)$
- 2. char a[3][2]; p = &a[0][0]; a[1][2] $\rightarrow *(p+...)$
- 3. char a[2][4][3]; p = &a[0][0]; a[1][2][2] $\rightarrow *(p+...)$

Artimetica dei Puntatori: Esercizi



- Date le seguenti matrici, trasformare l'operazione a[i][j] nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
- int *p = &a[0][0]; a[i][j]=*(p +)
- 1. int a[4][5]; p = &a[0][0]; a[2][1] $\rightarrow *(p+2*5+1)$
- 2. char a[3][2]; p = &a[0][0]; a[1][2] \rightarrow *(p+1*2+2) \rightarrow l'array ha 2 colonne, non esiste l'elemento di colonna 2
- 3. char a[2][4][3]; p = &a[0][0]; a[1][2][2] $\rightarrow *(p+1*4*3+2*3+2)$

Numeri Casuali



- Il calcolatore non è in grado di generare una sequenza casuale di numeri
- E' però in grado di generare in modo deterministico una sequenza di numeri pseudocasuale, ovvero deterministica ma che abbia le stesse proprietà di numeri casuali (es. distribuzione uniforme)
- le funzioni per generare numeri casuali sono definite in <stdlib.h>
- rand() restituisce un intero x. 0 <= x <= RAND_MAX (definita in stdlib.h, generalmente almeno 2^8)
- la sequenza di numeri casuali dipende dal valore passato alla funzione srand()
- unsigned int seed; srand(seed);
- se utilizziamo lo stesso valore di seed, otterremo la stessa sequenza di valori casuali

Esempio



• Scegliere casualmente un numero tra 1 e 10 inclusi

```
srand(0);
printf("%d\n", ... );
```

Esempio



• Scegliere casualmente un numero tra 1 e 10 inclusi

```
srand(0);
printf("%d\n", rand()%10+1);
```

Puntatori a Funzione



- il nome di una funzione è un identificatore, esattamente come gli identificatori che indicano variabili (x in "int x;");
 - Il nome di una funzione in un'espressione diventa un puntatore ad essa (come per gli array)
- Così come le variabili, ogni funzione ha un tipo, indicato dal tipo del valore restituito e dal tipo dei parametri
 - tipo_restituito (tipo parametri)
 - int somma(int x, int y); → la funzione somma ha tipo "int (int, int)"
- possiamo assegnare ad un puntatore una funzione, possiamo passare funzioni come parametri
- come indichiamo un puntatore a funzione?
- tipo_restituito (* nome_variabile)(parametri)
- Es. int (*c)(int a, int b)
- c è un puntatore ad una funzione che riceve due parametri interi e restitusce un valore intero

Puntatori a Funzione



- Consideriamo la funzione che ordina un array di interi selezionando ripetutamente l'elemento minore ed inserendolo ad inizio array
- Alla seconda iterazione seleziona il minimo tra gli elementi dell'array che vanno dal secondo all'ultimo e lo inserisce in seconda posizione
- In letteratura l'algoritmo prende il nome di Selection Sort.

```
void selectionSort(int *X, int size) {
int indice min;
  for(int i=0; i<size-1; i+=1) {
    indice_min = trova_indice_minimo(X+i, size-i);
    scambia(X+i, &X[indice min+i]);
```

Puntatori a Funzione



- Se vogliamo generalizzare selectionSort in modo da poter ordinare l'array in modo crescente o decrescente, possiamo
- passare un puntatore a funzione come parametro, assumendo che tale funzione calcoli il minimo o il massimo di un array:
- void selectionSortGeneral(int *X, int size, trova_indice_minimo);

```
void selectionSortGeneral(int *X, int size,
int (*seleziona elem)(int *A, int size)) {
  int elem;
  for(int i=0; i<size-1; i+=1) {
    elem = (*seleziona elem)(X+i, size-i);
    scambia(X+i, &X[elem+i]);
```

Esempio: Menu



- Un modo compatto per creare un menu per un'applicazione
 - nel quale l'utente indica con un numero, tra 0 e n, la funzionalità desiderata
- è quello di creare un array di puntatori a funzioni e poi richiamare la funzione dell'indice nell'array corrispondente alla scelta dell'utente:

```
void function1(int a);
void function2(int b);
void function3(int c);
void (*f[3])(int) = { function1, function2, function3 };
```

Esempio: Menu



```
void function1(int a);
void function2(int b);
void function3(int c);
void (*f[3])(int) = { function1, function2, function3 };
scanf("%d", &choice);
(*f[choice])(choice);
void function1(int a) {
 printf("You entered %d so function1 was called\n\n", a);
```



• Forniscono un modo compatto per esprimere espressioni come x=x+1 e x=x-1

Esempio di espressione	Spiegazione
++a	Incrementa a di 1, quindi usa il nuovo valore di a nell'espressione in cui si trova a.
a++	Usa il valore corrente di a nell'espressione in cui si trova a, quindi incrementa a di 1.
b	Decrementa b di 1, quindi usa il nuovo valore di b nell'espressione in cui si trova b.
b	Usa il valore corrente di b nell'espressione in cui si trova b, quindi decrementa b di 1.

 La differenza tra ++a, a++ diventa significativa se sono usati all'interno di un'espressione o come parametri di una funzione



```
int c = 5;
printf( "%d\n", c ); // stampa 5
printf( "%d\n", c++ ); // stampa 5 e poi incrementa c
printf( "%d\n\n", c ); // stampa 6
c = 5;
printf( "%d\n", c ); // stampa 5
printf( "%d\n", ++c ); // incrementa c e poi (quindi) stampa 6
printf( "%d\n", c ); // stampa 6
```



Gli operatori di incremento e decremento

- si applicano solamente a nomi di variabili: non si può scrivere ++(x + 1)
- hanno priorità rispetto agli altri operatori aritmetici:
 - *dim-- NON decrementa di uno il valore puntato da dim, ma
 - dim viene fatto puntare alla cella di memoria precedente a quella attuale e DOPO si effettua la dereferenziazione, ovvero ci si reca alla cella di memoria di indirizzo dim-1
 - se dim puntava al primo elemento di un array, con *dim
 si spera di ottenere un segmentation fault
- permettono di scrivere codice compatto:

```
char a[10] = "hello";
int i=0;
while(i<5)
    printf("%c ", a[i++]);</pre>
```



- ma, a volte, meno leggibile
 int i=2; printf("%d\n", 3*i+++3); //stampa 9, i==3 dopo la printf.
- int fatt()
- gli operatori ++ -- non dovrebbero essere utilizzati in un'espressione se la stessa variabile è presente più di una volta nella stessa espressione (anche a sinistra dell'uguale):

int
$$i=2$$
; int $x=3$; $x = i++*i++;$

- è un comportamento non definito dallo standard: i compilatori hanno libertà di decidere quando effettuare le i++, per cui può succedere che dopo la valutazione del primo i la variabile venga incrementata subito, oppure può succedere che l'incremento avvenga dopo la valutazione del secondo i:
 - x=2*3 e i=4 oppure x=2*2, i=4

Argomenti da linea di Comando



• E' possibile passare argomenti ad un programma C direttamente da linea di comando al momento dell'esecuzione (invece di leggerli da tastiera)

gcc –o palindroma palindroma.c

palindroma abba

stampa "la stringa abba è palindroma"

- i parametri sono separati da spazi: "palindromo abba 1221" indica due parametri stringa
- gli argomenti da linea di comando sono i parametri della funzione main: int main (int argc, char *argv[])
- i nomi delle variabili sono arbitrari, ma si usa argc e argv per tradizione

Argomenti da linea di Comando



int main (int argc, char *argv[])

- una volta definiti i parametri della funzione main, ovvero int main (void) -> int main (int argc, char *argv[])
- si avrà sempre almeno un argomento passato da linea di comando

```
argc>=1
```

argv[0] è il nome del programma

- argc: numero di parametri passati dalla linea di comando (incluso il nome del programma)
- argv[i]: l'i-esimo argomento
 - il primo argomento ha indice 1, il secondo ha indice 2

Esercizio



• Scrivere un frammento di codice che stampi la lista dei parametri passati da linea di comando, escluso il nome del programma:

```
int main (int argc, char *argv[]) {
    ...
}
```

Esercizio



 Scrivere un frammento di codice che stampi la lista dei parametri passati da linea di comando, escluso il nome del programma

```
int main (int argc, char *argv[]) {
     for(int i=1; i < argc; i+=1)
          printf("argv[%d]:=%s\n", i, argv[i]);
     }
}</pre>
```

Gestione Input



- Ma se volessimo passare dei numeri come argomenti da linea di comando?
- Non è possibile. Ciò che si può fare è convertire le stringhe in numeri
- <stdlib.h> fornisce le seguenti funzioni

Prototipo della funzione	Descrizione della funzione
<pre>double strtod(const char *nPtr, char **endPtr);</pre>	
	Converte la stringa nPtr in un double.
<pre>long strtol(const char *nPtr, char **endPtr, int base);</pre>	
	Converte la stringa nPtr in un long.
<pre>unsigned long strtoul(const</pre>	<pre>char *nPtr, char **endPtr, int base);</pre>
	Converte la stringa nPtr in un unsigned long.

Conversione Stringa -> Double



- double strtod(const char *nPtr, char **endPtr);
- nPtr è la stringa che contiene il nostro numero: i caratteri di spaziatura all'inizio della stringa vengono ignorati
- la stringa può contenere altri caratteri dopo le cifre del numero, es. "
 12.4ciao", dopo l'esecuzione di strtod(), endPtr punta al primo
 carattere dopo il numero tradotto, nell'esempio 'c'.
- Se nessun carattere viene tradotto endPtr punta al primo carattere di *nPtr

```
char *endPtr;
char *p = " 23.2abc";
printf("%f -%s-\n", strtod(p, &endPtr), endPtr); // 23.20000 -abc-
```

Conversione Stringa -> Long



- long strtol(const char *nPtr, char **endPtr, int base);
- nPtr, endPtr hanno lo stesso che hanno in strtod()
- base è la base di decodifica del numero (binaria, decimale), solitamente base = 0

```
char *endPtr;
char *p = " -232abc";
printf("%f -%s-\n", strtol(p, &endPtr, 0), endPtr); // -232 -abc-
```

Altre Funzioni per Stringhe



• in <stdio.h> sono presenti anche le seguenti funzioni:

- int sprintf(char *s, const char *format, ...);
 - equivalente alla printf, ma invece di stampare a video, memorizza nella stringa s
 int sprintf(char *s, const char *format, ...);
 - può essere utilizzata per simulare l'assegnamento ad una stringa

- int sscanf(char *s, const char *format, ...);
 - equivalente alla scanf, ma l'input è letto dalla stringa s
 - può essere utilizzata come alternativa alle funzioni strtod(), strtol()

Switch e Break



- L'istruzione Switch è un'alternativa ad una serie di if
- si confronta grade con 'A', se sono uguali si esegue il codice dopo i : (in questo caso nessuna istruzione)
- si passa al case seguente: 'a', e così via
- il break sposta l'esecuzione al comando seguente allo switch (agli esami switch e break non devono essere utilizzati)
- default equivale ad un test sempre vero (viene sempre eseguita a meno di aver incontrato un break in precedenza)

```
char grade;
int aCount=0, bCount=0;
switch (grade) {
       case 'A':
       case 'a':
               aCount+=1;
               break;
       case 'b':
               bCount+=1;
               break;
       default:
               printf("%s", "ERRORE");
```