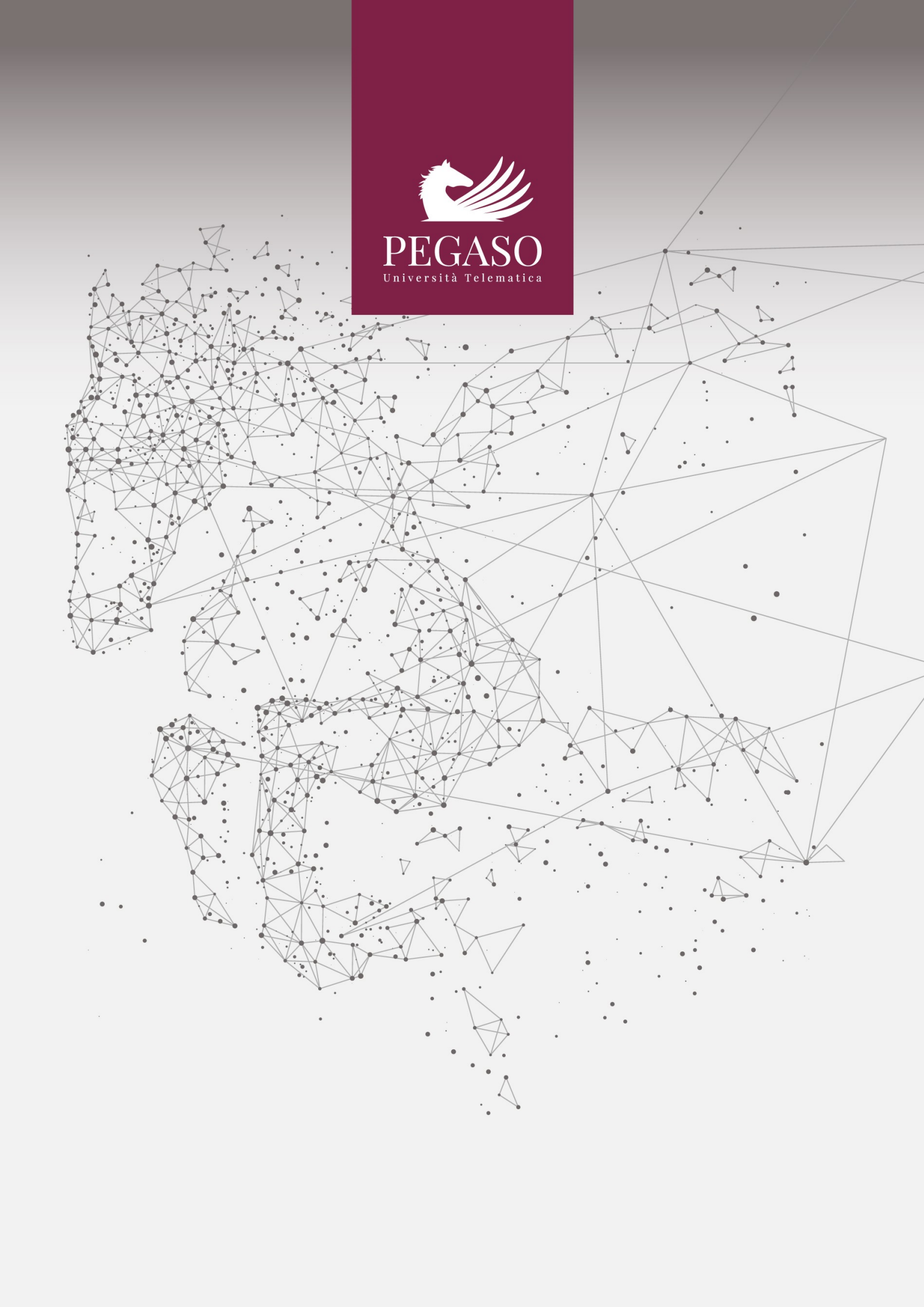
PEGASO
Università Telematica

Indice

1. Relazioni tra puntatori e array

Array e puntatori sono intimamente correlati in C e spesso possono essere usati in maniera intercambiabile.

Al nome di un array si può pensare come a un puntatore costante.

I puntatori possono essere usati per fare qualsiasi operazione che implichi l'indicizzazione di un array.

Supponete le seguenti definizioni:

```
int b[5];
int *bPtr;
```

Poiché il nome dell'array `b` (senza indice) è un puntatore al primo elemento dell'array, possiamo rendere `bPtr` uguale all'indirizzo del primo elemento nell'array b con l'istruzione

```
bPtr = b;
```

Questa istruzione è equivalente all'assegnazione a `bPtr` dell'indirizzo del primo elemento dell'array, cioè:

```
bPtr = &b[0];
```

Notazione puntatore/offset

All'elemento `b[3]` dell'array si può fare alternativamente riferimento con l'espressione con puntatori

```
*(bPtr + 3)
```

Il 3 nell'espressione è l'offset (letteralmente "scarto") rispetto al puntatore.

Quando `bPtr` punta al primo elemento di un array, l'offset indica a quale elemento dell'array si fa riferimento e il suo valore coincide con l'indice dell'array.

Questa notazione è detta notazione puntatore/offset.

Le parentesi sono necessarie perché la precedenza dell'operatore `*` è maggiore della precedenza dell'operatore +.

Senza le parentesi, l'espressione precedente aggiungerebbe 3 al valore dell'espressione

`*bPtr` (cioè il 3 verrebbe aggiunto a `b[0]`, supponendo che `bPtr` punti all'inizio dell'array).

Così come si può fare riferimento all'elemento dell'array con un'espressione con puntatori, l'indirizzo

```
&b[3]
```

può essere scritto con la seguente espressione con puntatori:

```
bPtr + 3
```

È possibile trattare lo stesso array come un puntatore e usarlo nell'aritmetica dei puntatori. Ad esempio, l'espressione

```
*(b + 3)
```

fa riferimento anch'essa all'elemento `b[3]` dell'array.

In generale, è possibile scrivere con un puntatore e un offset tutte le espressioni con array indicizzati.

In questo caso, la notazione puntatore/offset è stata usata con il nome dell'array come puntatore.

L'istruzione precedente non modifica in alcun modo il nome dell'array; b punta ancora al primo elemento nell'array.

Notazione puntatore/indice

I puntatori possono essere indicizzati come gli array.

Se `bPtr` ha il valore di `b`, l'espressione

```
bPtr[1]
```

si riferisce all'elemento `b[1]` dell'array.

Questa è chiamata notazione puntatore/indice.

Non è possibile modificare il nome di un array con aritmetica dei puntatori

Ricordate che il nome di un array è essenzialmente un puntatore costante; esso punta sempre all'inizio dell'array.

Pertanto, l'espressione

```
b += 3
```

è scorretta poiché tenta di modificare il valore del nome dell'array con l'aritmetica dei puntatori.

☹ *Errore comune di programmazione*

Tentare di modificare il valore del nome di un array con l'aritmetica dei puntatori genera un errore di compilazione.

2. Dimostrare indicizzazione e offset con un puntatore

Il programma seguente usa i quattro metodi che abbiamo esaminato per il riferimento agli elementi di un array (indicizzazione di un array, puntatore/offset con il nome dell'array come puntatore, indicizzazione di un puntatore e puntatore/offset con un puntatore) per stampare i quattro elementi dell'array intero b.

```
1  // Uso delle notazioni con indice
 2  //  e con puntatori per gli array.
 3  #include <stdio.h>
 4  #define ARRAY_SIZE 4
 5
 6  int main(void)
 7  {
 8     int b[] = {10, 20, 30, 40}; // crea e inizializza l'array b
 9     int *bPtr = b; // crea bPtr e fallo puntare all'array b
10
11     // stampa l'array b usando la notazione degli array con indice
12     puts("Array b printed with:\nArray index notation");
13
14     // effettua un ciclo lungo l'array b
15     for (size_t i = 0; i < ARRAY_SIZE; ++i) {
16        printf("b[%u] = %d\n", i, b[i]);
17     }
18
19     // stampa l'array b con il nome e la notazione puntatore/offset
20     puts("\nPointer/offset notation where\n"
21           "the pointer is the array name");
22
23     // effettua un ciclo lungo l'array b
24     for (size_t offset = 0; offset < ARRAY_SIZE; ++offset) {
25        printf("*(b + %u) = %d\n", offset, *(b + offset));
26     }
27
28     // stampa l'array b con bPtr e la notazione degli array con indice
```

```
29     puts("\nPointer subscript notation");
30
31     // effettua un ciclo lungo l'array b
32     for (size_t i = 0; i < ARRAY_SIZE; ++i) {
33        printf("bPtr[%u] = %d\n", i, bPtr[i]);
34     }
35
36     // stampa l'array b usando bPtr e la notazione puntatore/offset
37     puts("\nPointer/offset notation");
38
39     // effettua un ciclo lungo l'array b
40     for (size_t offset = 0; offset < ARRAY_SIZE; ++offset) {
41        printf("*(bPtr + %u) = %d\n", offset, *(bPtr + offset));
42     }
43  }
```

```
Array b printed with:

Array subscript notation

b[0] = 10

b[1] = 20

b[2] = 30

b[3] = 40


Pointer/offset notation where

the pointer is the array name

*(b + 0) = 10

*(b + 1) = 20

*(b + 2) = 30

*(b + 3) = 40

Pointer subscript notation

bPtr[0] = 10

bPtr[1] = 20
```

```
bPtr[2] = 30

bPtr[3] = 40

Pointer/offset notation

*(bPtr + 0) = 10

*(bPtr + 1) = 20

*(bPtr + 2) = 30

*(bPtr + 3) = 40
```

3. Copiare stringhe con array e puntatori

Per illustrare ulteriormente l'intercambiabilità fra array e puntatori, esaminiamo le due funzioni che copiano stringhe, `copy1` e `copy2`, nel programma seguente.

Entrambe le funzioni copiano una stringa in un array di caratteri.

Da un confronto dei prototipi di funzione per `copy1` e `copy2`, le funzioni appaiono identiche.

Esse eseguono lo stesso compito, ma sono implementate in maniera diversa.

```
1  //Copia di una stringa usando
2   // le notazioni con array e con puntatori.
3   #include <stdio.h>
4   #define SIZE 10
5
6   void copy1(char * const s1, const char * const s2); // prototipo
7   void copy2(char *s1, const char *s2); // prototipo
8
9   int main(void)
10  {
11     char string1[SIZE]; // crea l'array string1
12     char *string2 = "Hello"; // crea un puntatore a una stringa
13
14     copy1(string1, string2);
15     printf("string1 = %s\n", string1);
16
17     char string3[SIZE]; // crea l'array string3
18     char string4[] = "Good Bye"; // crea un puntatore a una stringa
19
20     copy2(string3, string4);
21     printf("string3 = %s\n", string3);
22  }
23
24  // copia s2 in s1 usando la notazione con array
25  void copy1(char * const s1, const char * const s2)
26  {
27     // effettua un ciclo lungo le stringhe
```

```
28      for (size_t i = 0; (s1[i] = s2[i]) != '\0'; ++i) {
29         ; // non fare niente nel corpo
30      }
31   }
32
33   // copia s2 in s1 usando la notazione con puntatori
34   void copy2(char *s1, const char *s2)
35   {
36      // effettua un ciclo lungo le stringhe
37      for (; (*s1 = *s2) != '\0'; ++s1, ++s2) {
38         ; // non fare niente nel corpo
39      }
40   }
```

```
string1 = Hello

string3 = Good Bye
```

Copiare con la notazione degli array con indice

La funzione `copy1` usa la notazione degli array con indice per copiare la stringa in `s2` nell'array di caratteri `s1`.

La funzione definisce la variabile contatore i come indice dell'array.

L'intestazione dell'istruzione `for` (riga 28) esegue l'intera operazione di copiatura (il suo corpo è l'istruzione vuota).

L'intestazione specifica che i è inizializzata a zero e viene incrementata di uno a ogni iterazione del ciclo.

L'espressione `s1[i] = s2[i]` copia un carattere da `s2` a `s1`.

Quando in `s2` si incontra il carattere nullo, questo viene assegnato a `s1` e il valore dell'intera assegnazione diventa il valore assegnato all'operando sinistro (`s1`).

Il ciclo termina proprio quando il carattere nullo viene assegnato a `s1` da `s2` (falso).

Copiare con puntatori e aritmetica dei puntatori

La funzione `copy2` usa i puntatori e l'aritmetica dei puntatori per copiare la stringa in `s2` nell'array di caratteri `s1`.

Ancora, l'intestazione dell'istruzione `for` (riga 37) esegue l'intera operazione di copiatura.

L'intestazione non include alcuna inizializzazione della variabile.

Come nella funzione `copy1`, l'espressione `(*s1 = *s2)` esegue l'operazione di copiatura.

Il puntatore `s2` è dereferenziato e il carattere risultante è assegnato al puntatore dereferenziato `*s1`.

Dopo l'assegnazione nella condizione, i puntatori sono incrementati per puntare, rispettivamente, all'elemento successivo dell'array `s1` e al carattere successivo della stringa `s2`.

Quando in `s2` si incontra il carattere nullo, esso è assegnato al puntatore dereferenziato `s1` e il ciclo termina.

Note riguardanti le funzioni `copy1 e copy2`

Il primo argomento sia per `copy1` che per `copy2` deve essere un array grande abbastanza da contenere la stringa fornita come secondo argomento.

Altrimenti, può verificarsi un errore in seguito al tentativo di scrivere in una locazione di memoria che non fa parte dell'array.

Inoltre, in entrambe le funzioni, il secondo argomento è copiato nel primo argomento – i caratteri sono letti da esso uno alla volta, ma non vengono mai modificati.

Pertanto, il secondo parametro è dichiarato come puntatore a una stringa costante, in modo che venga applicato il principio del privilegio minimo: nessuna delle due funzioni richiede la modifica della stringa nel secondo argomento.

Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione", Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.