Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

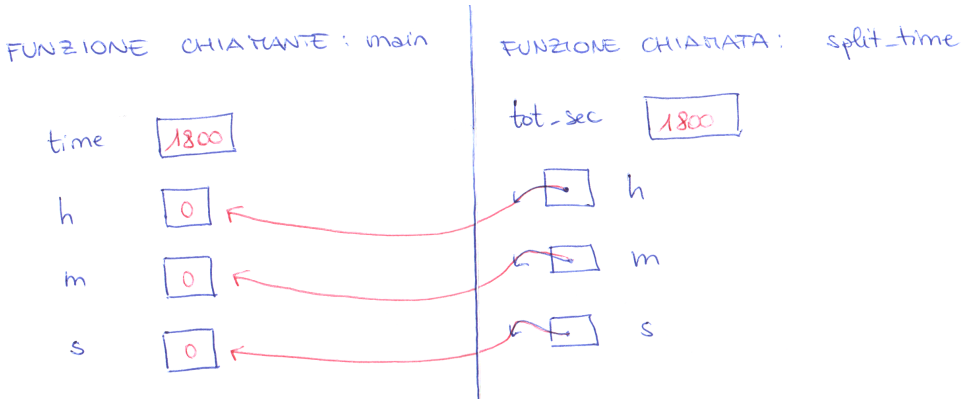
Puntatori - esercizi introduttivi da svolgere a casa*

1 Trasformazione orario in secondi

Considerate il codice contenuto nel file `split-time.c`. Come potete notare, il codice è incompleto. Dato un orario fornito in numero di secondi dalla mezzanotte, la funzione `split_time` deve calcolare l'orario equivalente in ore, minuti, secondi, e memorizzarlo nelle tre variabili puntate da `h`, `m` e `s` rispettivamente.

```
1  #include <stdio.h>
2  #define N 5
3
4  void split_time( long int tot_sec, int *h, int *m, int *s ) {
5    ... = tot_sec / 3600;
6    tot_sec %= 3600;
7    ... = tot_sec / 60;
8    ... = tot_sec % 60;
9  }
10
11 int main( void ) {
12   int time = 1800, h=0, m=0, s=0;
13
14   split_time( time, ... , ... , ... );
15   printf( "h = %d, m = %d, s = %d\n", h, m, s );
16
17   return 0;
18 }
```

Lo stato della memoria subito dopo la chiamata nella riga 14 deve corrispondere con quanto rappresentato nella figura qui sotto. Le parti in rosso si riferiscono all'inizializzazione dei parametri nel momento in cui la funzione viene invocata.



*Ultima modifica 25 ottobre 2019

Analizzate il codice e rispondete per iscritto alle seguenti domande.

1. Di che tipo sono i parametri `h`, `m` e `s` della funzione dichiarata nella linea 4? A cosa serve dichiararli così?
2. Nella figura ci sono due porzioni di memoria identificate dallo stesso nome `h`. In che cosa si distinguono? Individuate le righe di codice in cui vengono dichiarate.
3. Completate opportunamente la invocazione di riga 14.
4. Completate opportunamente la definizione della funzione `split_time` (righe 5-8) facendo in modo che gli assegnamenti modifichino il valore delle variabili locali della funzione `main`.
5. Verificate il funzionamento del programma completo, testandolo con valori di `time` diversi.

2 Scambio di valori

1. Scrivete una funzione con prototipo `void scambia( int *p, int *q )` che scambi i valori delle due variabili puntate da `p` e `q`.
2. Testate la funzione scrivendo un programma che legge due interi, li scambia invocando opportunamente la funzione `scambia` e infine li stampa nel nuovo ordine.
3. Rappresentate graficamente lo stato della memoria subito dopo la chiamata della funzione.

3 Attraversamento

Considerate il codice contenuto nel file `attraversa.c`.

```
1  #include <stdio.h>
2
3  #define LENGTH 100
4
5  int main( void) {
6
7      int a[LENGTH];
8      int *p;
9
10     for( p = a; p < a + LENGTH; p++ ) {
11         scanf( "%d", p );
12         if ( *p == 0 )
13             break;
14     }
15
16     while ( --p >= a )
17         printf( "%d ", *p );
18
19     printf( "\n" );
20     return 0;
21 }
```

Analizzate il codice sorgente e rispondete, possibilmente per iscritto, alle seguenti domande. Se avete dubbi, potete testarlo, eseguendolo su casi di input significativi e modificandolo.

- Che tipo di variabile è `p`?
- Perché nella riga 11 il nome `p` non è preceduto dal simbolo `%` come al solito?
- Senza eseguire automaticamente il programma, tracciatene l'esecuzione quando riceve il seguente input:
  `1 2 3 4 5 0`
- A cosa serve il ciclo `for` nelle righe 10-14?
- A cosa serve il ciclo `while` nelle righe 16-74?
- A cosa serve la variabile `p`?

4 Palindrome con argomenti da linea di comando

1. Scrivete una funzione che stabilisca se il suo argomento è una parola palindroma oppure no, usando due puntatori per scorrere la parola partendo dall'inizio e dalla fine.
   Nota bene. Come va dichiarato l'argomento della funzione? Come vanno dichiarati i due puntatori? Osservate a che cosa puntano...
2. Scrivete un programma che riceve una parola come argomento da linea di comando, quindi invoca la funzione appena scritta per stabilire se si tratta di una parola palindroma oppure no.
3. Modificate il proramma affinché riceva da linea di comando una sequenza di parole e stabilisca, per ciascuna di esse, se si tratta di una parola palindroma oppure no.

5 Puntatore al minimo

1. Scrivete una funzione con prototipo `int *smallest( int a[], int n )` che, dato un array `a` di lunghezza `n`, restituisca un puntatore all'elemento più piccolo dell'array.
2. Testate la funzione inserendola in un programma che legga una sequenza di numeri, la memorizzi un un array `a` e stampi il valore più piccolo del vettore usando l'istruzione `printf( "%d", *smallest( a, n ) )`.

6 Da minuscolo a maiuscolo

1. Scrivete una funzione con prototipo `char *maiuscolo( char *stringa )` che trasformi da minuscolo in maiuscolo tutte le lettere del suo argomento `stringa` e ne restituisca un puntatore al primo carattere. Potete assumere che `stringa` sia dato da una stringa terminata da `'\0'` contenente caratteri ASCII (non solo lettere). Potete usare la funzione `toupper` della libreria `ctype.h`.
2. Testate la funzione inserendola in un programma che legga da standard input una frase terminata da a-capo, la memorizzi in una stringa `s`, invochi la funzione, quindi ristampi la frase tutta in maiuscolo.
3. Come avete dichiarato `s`? Spiegate perché.
4. Spiegate se c'è differenza, e se sì quale, tra le seguenti dichiarazioni per la funzione `maiuscolo`:
   `char *maiuscolo( char *stringa )`
   `char *maiuscolo( char stringa[] )`