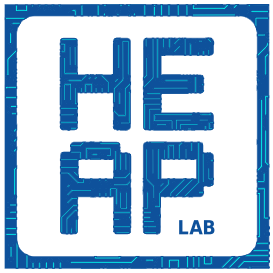
Esercizi su funzioni e puntatori in C

Stefano Cherubin*



03/11/2017

[**Informatica A**] Esercitazione #9

corso per Ing. Gestionale a.a. 2017/18

*<nome>.<cognome>@polimi.it

Indice

1 Analisi di codice **3**
1.1 Soluzione 5
1.1.1 Sandra 5
1.1.2 Raimondo 5
1.1.3 Paolo 5
1.1.4 Luca 5
1.1.5 Bud 5
1.1.6 Terence 5

2 Case-insensitive strcmp **6**
2.1 Approccio alla soluzione 6
2.2 Soluzione C 7
2.3 Versione senza parti superflue 8
2.4 Versione senza puntatori 9

3 Calendario perpetuo **10**
3.1 Approccio alla soluzione 10
3.2 Definizione dei sottoproblemi 10
3.2.1 Anni bisestili 10
3.3 Soluzione C 11

1 Analisi di codice

Si discuta quali errori, se ne esistono, possono verificarsi nel codice delle stesse funzioni. Si dica anche quante e quali stringhe comparirebbero a video nel caso venissero chiamate le seguenti funzioni.

Listato 1: Sandra

```
1 int sandra(int *sonno) {
2   *sonno = 0
3   while (++(*sonno)) {
4     printf("Che barba, che noia!");
5     printf("Che noia, che barba!");
6   }
7   printf("Buonanotte!");
8   return 0;
9 }
```

Listato 2: Raimondo

```
1 int raimondo() {
2   int *sonno;
3   sonno = 0;
4   while (sonno) {
5     printf("Posso leggere il giornale?");
6     *sonno++;
7   }
8   return 0;
9 }
```

Listato 3: Paolo

```
1 int paolo(int *alfa) {
2   *alfa = 0;
3   printf("Un uomo chiamato ");
4   if (*alfa++) {
5     printf("contratto.");
6   }
7   return *alfa;
8 }
```

Listato 4: Luca

```
1  int luca() {
2    int *paga;
3    paga = 0;
4    while (++paga) {
5      printf("Un caffè e poi al lavoro?");
6    }
7    return *paga;
8  }
```

Listato 5: Bud

```
1  int bud() {
2    int *cazzotti, *botte;
3    *cazzotti = 0;
4    botte = cazzotti;
5    for (++*botte; *cazzotti; (*cazzotti)++) {
6      printf("*tonf* E stai giu'!\n");
7    }
8    return *botte;
9  }
```

Listato 6: Terence

```
1  int terence() {
2    int hill[2] = {0, 0};
3    *hill = 0;
4    int *donMatteo = hill;
5    ++donMatteo;
6    *hill = 666;
7    while (++*donMatteo) {
8      printf("\n...E continuavano a chiamarlo");
9      if (--(*hill)) {
10       printf(" Trinita'");
11     } else {
12       (*hill)++;
13     }
14   }
15   return *hill;
16 }
```

1.1 Soluzione

1.1.1 Sandra

La funzione entra in un ciclo infinito e a video vengono stampate ripetutamente le stringhe `"Che barba, che noia!""Che noia, che barba!".`

1.1.2 Raimondo

La funzione non entra mai nel ciclo e restituisce subito il valore `0`.

1.1.3 Paolo

La funzione stampa a video la stringa `"Un uomo chiamato "` e restituisce il valore `1.`

1.1.4 Luca

La funzione entra in un ciclo infinito e a video viene stampata ripetutamente la stringa `"Un caffè e poi torno al lavoro?"` senza mai restituire alcun valore.

1.1.5 Bud

La funzione entra in un ciclo infinito e a video viene stampato ripetutamente `"*tonf* E stai giu'!"`.

1.1.6 Terence

La funzione entra in un ciclo infinito e a video viene stampata per 665 volte la stringa `"...E continuavano a chiamarlo Trinita'"`. A seguito verrà stampata ripetutamente la stringa `"...E continuavano a chiamarlo"`.

2 Case-insensitive strcmp

Si scriva una funzione in grado di confrontare due stringhe, ritornando un risultato compatibile con la funzione `strcmp`, salvo il fatto che si devono ignorare le differenze tra caratteri maiuscoli e minuscoli.

Si utilizzi una scansione delle stringhe basata su puntatori.

2.1 Approccio alla soluzione

Il testo dell'esercizio in questo caso è vincolante su un approccio da utilizzare. In particolare viene richiesto che la scansione delle stringhe sia basata su puntatori. In sede d'esame sarà accettata come parzialmente corretta qualunque soluzione che risolva il problema dato ma il 100% del punteggio sarà assegnato solo a coloro che sapranno soddisfare anche questo vincolo.

Di seguito viene proposta una soluzione che fa uso di puntatori e, di seguito, una versione senza puntatori. Può risultare più istintivo proporre una soluzione che segua un approccio invece di un altro. Si vuole qui dimostrare l'equivalenza tra le due rappresentazioni fornendo il codice per entrambe.

2.2 Soluzione C

Si noti come in questa soluzione i vettori di caratteri rappresentanti le stringhe vengono passati alla funzione `confrontaStr` come puntatori. Si ricorda che le stringhe sono vettori di caratteri. Si ricorda inoltre che i vettori sono degli array monodimensionali. Per come sono definiti gli array in C99, essi contengono l'indirizzo di memoria della prima delle celle contigue che l'array contiene. In questo caso i puntatori parametro rappresentano il primo carattere delle rispettive stringhe.

Per accedere al valore puntato da un puntatore, è sufficiente e necessario dereferenziare il puntatore. Ciò si fa tramite l'operatore di dereferenziazione (`*`, applicato a sinistra del puntatore).

L'incremento di un puntatore equivale all'incremento di una unità di tipo puntato. Accade cioè che l'incremento unitario(operatore `++`) di un puntatore a `char` causerà la variazione del puntatore di un ammontare pari alla dimensione di un `char`, portando così il puntatore a puntare al carattere successivo.

Listato 7: case-insensitive strcmp

```
1  char capitalize(char c);
2  int confrontaStr(char *s0, char *s1);
3
4  int confrontaStr(char *s0, char *s1) {
5    char *p0, *p1;
6    p0 = s0;
7    p1 = s1;
8    while (*p0 != '\0' && *p1 != '\0' && capitalize(*p0)
         == capitalize(*p1)) {
9      p0++;
10     p1++;
11   }
12   return (int)capitalize(*p0) - (int)capitalize(*p1);
13 }
14
15 char capitalize(char c) {
16   if (c >= 'a' && c <= 'z')
17     return (char)((int)c - ((int)'a' - (int)'A'));
18   return c;
19 }
```

2.3 Versione senza parti superflue

La definizione dei prototipi[1] di funzione è opzionale quando tutte le chiamate a quella funzione avvengono in righe di codice successive alla sua implementazione.

Si ricorda inoltre che esiste un *cast* implicito tra i tipi **int** e **char** (che fa riferimento alla codifica ASCII del carattere).

Listato 8: case-insensitive strcmp

```
1
2  char capitalize(char c) {
3    if (c >= 'a' && c <= 'z')
4      return (c - ('a' - 'A'));
5    return c;
6  }
7
8  int confrontaStr(char *s0, char *s1) {
9    char *p0 = s0, *p1 = s1;
10   while (*p0 != '\0' && *p1 != '\0' && capitalize(*p0)
         == capitalize(*p1)) {
11     p0++;
12     p1++;
13   }
14   return capitalize(*p0) - capitalize(*p1);
15 }
```

[1] altrimenti noti come *firma* della funzione o *dichiarazione* della funzione

2.4 Versione senza puntatori

Questa versione è corretta secondo la sintassi C99 e semanticamente equivalente alle versioni precedentemente proposte. Questa versione non sarebbe considerata corretta al 100% in sede d'esame in quanto non fa uso, come esplicitamente richiesto dal testo dell'esercizio di una scansione basata sui puntatori. Si propone anche questa versione a scopo didattico di confronto.

Listato 9: case-insensitive strcmp

```
1
2  char capitalize(char c) {
3    if (c >= 'a' && c <= 'z')
4      return (c - ('a' - 'A'));
5    return c;
6  }
7
8  int confrontaStr(char s0[], char s1[]) {
9    int p0, p1;
10   for (p0 = 0, p1 = 0; s0[p0] != '\0' && s1[p1] != '\0
         ' && capitalize(s0[p0]) == capitalize(s1[p1]); p0
         ++, p1++) ;
11   return capitalize(s0[p0]) - capitalize(s1[p1]);
12 }
```

3 Calendario perpetuo

> Scrivere un programma che realizzi un calendario perpetuo, ovvero calcoli il giorno della settimana corrispondente a una certa data, sapendo che il 1 Gennaio 1900 è stato un lunedì.

3.1 Approccio alla soluzione

Calcolare il numero di giorni trascorsi dal 01/Gen/1900 alla data richiesta, e valutare con l'operatore mod il resto della divisione per 7 di tale numero. Se il resto è 0, il giorno è lunedì, se 1 è martedì e così via.

3.2 Definizione dei sottoproblemi

Cercando di decomporre il problema in sottoproblemi più semplici da risolvere, si possono individuare delle parti di problema che possono essere risolte con pazienza e non troppa difficoltà e che consentono l'avvicinamento alla soluzione del problema principale.

Per determinare il numero di giorni trascorsi dal 01/Gen/1900, occorre un criterio per stabilire

- quanti sono i giorni in un mese
- se un anno è bisestile

3.2.1 Anni bisestili

L'anno bisestile è un accorgimento utilizzato per mantenere in sincronia l'anno civile con il ciclo delle stagioni. Infatti, la durata dell'anno civile e la durata media dell'anno tropico (tempo tra due solstizi o due equinozi dello stesso tipo) non è dato da un numero intero di giorni. La durata di un anno civile era di 365 giorni ma la durata di un anno tropico era stata misurata in 365 giorni e 6 ore.

Per ovviare a questo problema il Calendario Giuliano (46 a.C.) introdusse un anno bisestile 1 volta ogni 4 anni (si *recuperava* un giorno perso una volta ogni quattro anni).

In realtà l'anno tropico ha durata di circa 365 giorni, 6 ore, 11 minuti, e 14 secondi. Usando il calendario Giuliano l'anno tropico era in ritardo rispetto all'anno civile (di 1 giorno ogni 128 anni); con la conseguenza che, con il passare dei secoli, la data ufficiale d'inizio delle stagioni veniva anticipata sempre più rispetto alle posizioni astronomiche.

Secondo il Calendario Gregoriano (in vigore dal 1582 in poi), un anno è bisestile se il suo numero è divisibile per 4, con l'eccezione degli anni secolari (divisibili per 100) che sono bisestili solo se divisibili anche per 400 (in questo modo l'inizio dell'anno tropico è in ritardo di circa 26 secondi per anno civile (circa 1 giorno ogni 3323 anni).

3.3 Soluzione C

Listato 10: Calendario perpetuo

```
1  #include <stdio.h>
2
3  #define MAX_LEN 11
4
5  typedef struct {
6    int giorno;
7    int mese;
8    int anno;
9    int bis;
10   char Giorno[MAX_LEN];
11 } Data;
12
13 int bisestile(int anno);
14 int GiorniMese(int mese, int anno);
15 Data Calendario(int gg, int mese, int anno);
16
17 int main() {
18   int mm, aa, gg;
19   Data d;
20   do {
21     do {
22       printf("\nInserire giorno: ");
23       scanf("%d",&gg);
24     } while (gg < 1 || gg > 31);
25     do {
26       printf("\nInserire mese: ");
27       scanf("%d",&mm);
28     } while (mm < 1 || mm > 12);
29     do {
30       printf("\nInserire anno: ");
31       scanf("%d",&aa);
32     } while (aa < 1900);
33   } while (GiorniMese(mm, aa) < gg);
34   d = Calendario(gg,mm,aa);
35   printf("\nLa data inserita corrisponde a: ");
36   printf("\n %s %d / %d / %d", d.Giorno, d.giorno, d.
       mese, d.anno);
37   if (bisestile(d.anno))
38     printf("\nL'anno è bisestile.");
39   return 0;
40 }
41
```

```
42 int bisestile(int anno) {
43   if (((anno % 4 == 0) && (anno % 100 != 0)) || (anno
       % 400 == 0))
44     return 1;
45   else
46     return 0;
47 }
48
49 int GiorniMese(int mese, int anno) {
50   if (mese == 2)
51     return (28 + bisestile(anno));
52   else if ((mese == 11) || (mese == 4) || (mese == 6)
       || (mese == 9))
53     return 30;
54   else
55     return 31;
56 }
57
58 Data Calendario(int gg, int mese, int anno) {
59   typedef char nome[MAX_LEN];
60   nome week[7] = { "Lunedì", "Martedì", "Mercoledì", "
       Giovedì", "Venerdì", "Sabato", "Domenica" };
61   Data risultato;
62   int count, i;
63   risultato.giorno = gg;
64   risultato.mese = mese;
65   risultato.anno = anno;
66   risultato.bis = bisestile(anno);
67   count = 0;
68   for (i = 1900; i < anno; i++) {
69     count = count + 365 + bisestile(i);
70   }
71   for (i = 1; i < mese; i++) {
72     count = count + GiorniMese(i, anno);
73   }
74   count = count + gg - 1;
75   count = count % 7;
76   i = 0;
77   while ((week[count])[i] != '\0') {
78     risultato.Giorno[i] = (week[count])[i];
79     i = i+1;
80   }
81   risultato.Giorno[i] = (week[count])[i];
82   return risultato;
83 }
```

Licenza e crediti

Crediti

Quest'opera contiene elementi tratti da materiale di Gerardo Pelosi redatto per il corso di Fondamenti di Informatica per Ingegneria dell'Automazione a.a. 2014/15.

Licenza beerware[2]

Quest'opera è stata redatta da Stefano Cherubin. Mantenendo questa nota, puoi fare quello che vuoi con quest'opera. Se ci dovessimo incontrare e tu ritenessi che quest'opera lo valga, in cambio puoi offrirmi una birra.

[2] http://people.freebsd.org/~phk/