Informatica per Statistica
Raccolta progressiva degli esercizi

Igor Melatti ed Ivano Salvo

18/11/2012

Esercizi da fare in laboratorio

Per ognuno dei programmi, provare sempre a compilare ed eseguire, dando di volta in volta input diversi.

(Lab.1). Modificare il programma di Figura 1 in modo che scriva su schermo il nome dell'autore del programma, vada poi a capo due volte e scriva la sua data di nascita. Verificare che la modifica funzioni, ricompilando e rieseguendo il programma stesso. Realizzare tale modifica sia usando due `printf`, sia usandone una sola. Cancellare una parentesi a caso, e provare a ricompilare ed eseguire (dalla lezione 3)

```
1 #include <stdio.h>
2
3 int main()
4 {
5  printf("Ciao mondo!!!!\n");
6  return 0;
7 }
```

Figure 1: Primo semplice programma in C

(Lab.2). Modificare il programma di Figura 2 in modo tale che faccia la sottrazione se viene immesso `'-'` da tastiera, mentre con un qualsiasi altro carattere viene fatta la somma (dalla lezione 5)

(Lab.3). Modificare il programma di Figura 2 in modo tale che faccia la divisione (intera, ovvero il quoziente) se viene immesso `'/'` da tastiera, mentre con un qualsiasi altro carattere viene fatto il prodotto (dalla lezione 5)

(Lab.4). Modificare il programma di Figura 2 in modo tale che la variabile `operazione` sia di tipo intero, e che venga fatta la somma se `operazione`

vale 1, la sottrazione se `operazione` vale 2, il quoziente se `operazione` vale 3, e il prodotto con qualsiasi altro valore (dalla lezione 5)

- suggerimento: <blocco_istruzioni> può contenere qualsiasi categoria di istruzioni; quindi anche un altro `if`...

(Lab.5). Modificare il programma fatto per l'esercizio (Lab.4) in modo tale che la variabile `operazione` sia nuovamente di tipo carattere, e che venga fatta la somma se `operazione` vale `'1'`, la sottrazione se `operazione` vale `'2'`, il quoziente se `operazione` vale `'3'`, e il prodotto con qualsiasi altro valore (dalla lezione 5)

(Lab.6). Modificare il programma fatto per l'esercizio (Lab.4) in modo tale che la variabile `operazione` sia nuovamente di tipo carattere, e che venga fatta la somma se `operazione` vale `'+'`, la sottrazione se `operazione` vale `'-'`, il quoziente se `operazione` vale `'/'`, e il prodotto con qualsiasi altro valore (dalla lezione 5)

(Lab.7). Risolvere gli esercizi (Lab.2)–(Lab.6), ma modificando il programma in Figura 3 (dalla lezione 5)

```c
#include <stdio.h>

int main(int argc, char **argv)
{
  int primo_operando, secondo_operando;
  char operazione;

  printf("Immettere '+' per addizione, qualsiasi altro carattere per");
  printf(" la sottrazione: ");
  scanf("%c", &operazione);
  printf("Immettere il primo operando: ");
  scanf("%d", &primo_operando);
  printf("Immettere il secondo operando: ");
  scanf("%d", &secondo_operando);
  if (operazione == '+') {
    printf("%d + %d = %d\n", primo_operando, secondo_operando,
           primo_operando + secondo_operando);
  }
  else
    printf("%d - %d = %d\n", primo_operando, secondo_operando,
           primo_operando - secondo_operando);
  return 0;
}
```

Figure 2: Un semplice programma C

```c
#include <stdio.h>

int esegui_op(int primo, char op, int secondo) {
  if (op == '+') return primo + secondo;
  else return primo - secondo;
}

char piu_o_meno(char op) {
  if (op == '+') return '+';
  else return '-';
}

int main(int argc, char **argv) {
  int primo_operando, secondo_operando, risultato;
  char operazione, vera_operazione;

  printf("Immettere '+' per addizione, qualsiasi altro carattere per");
  printf(" la sottrazione: ");
  scanf("%c", &operazione);
  printf("Immettere il primo operando: ");
  scanf("%d", &primo_operando);
  printf("Immettere il secondo operando: ");
  scanf("%d", &secondo_operando);
  vera_operazione = piu_o_meno(operazione);
  risultato = esegui_op(primo_operando, operazione, secondo_operando);
  printf("%d %c %d = %d\n", primo_operando, vera_operazione,
         secondo_operando,  risultato);
  return 0;
}
```

Figure 3: Una variante del programma C di Figura 2

(Lab.8). I programmi C in Figura 4 e 5 sono ad input fisso: per cambiare l'input, occorre modificare il programma stesso, ricompilarlo e rieseguirlo. In molti casi, questo non è desiderabile: si vuole un programma che possa accettare input *dopo* essere stato lanciato in esecuzione, ad esempio leggendoli da tastiera. Modificare il programma in Figura 5 in modo tale che (dalla lezione 8):

(a) `LENGTH_OF_A` sia molto alto (ad esempio 1000)

(b) come prima cosa, venga letta da tastiera (chiamando opportunamente `scanf`) la dimensione dell'array (deve essere minore di `LENGTH_OF_A`, altrimenti deve scrivere un messaggio d'errore e terminare il programma)

(c) dopodiché, tutti gli elementi di `A` vanno letti da tastiera (chiamando opportunamente `scanf`), usando un ciclo `for`

(d) per scrivere gli elementi di `A` su schermo, usare ancora un ciclo `for`, chiamando opportunamente `printf`

(Lab.9). Risolvere nuovamente l'esercizio (Lab.8) ma modificando il programma in Figura 4.

(Lab.10). Modificare i programmi ottenuti risolvendo gli esercizi (Lab.8) e (Lab.9), facendo sì che, anziché leggere una sola volta un vettore da tastiera (con relativa dimensione) e stamparlo prima e dopo l'ordinamento, continui a leggere da tastiera vettori da ordinare (ogni volta leggendo prima la relativa dimensione del vettore) finché la dimensione del vettore risulti 0 o maggiore di `LENGTH_OF_A`.

(Lab.11). Modificare il programma ottenuto risolvendo l'esercizio (Lab.10), definendo e chiamando opportunamente delle funzioni in modo tale che che il `main` non abbia più di una decina di righe.

(Lab.12). Con riferimento alla Figura 6, com'è possibile far fare una iterazione in meno al ciclo `for` (dalla lezione 10)?

(Lab.13). Aggiungere alla Figura 6 una funzione `main` che legga da tastiera un numero positivo e poi ne stampi il fattoriale, chiamando opportunamente la funzione `fattoriale` (dalla lezione 10)

(Lab.14). Con riferimento alla Figura 7, aggiungere delle inizializzazioni (a piacere) per le variabili non inizializzate, un main, e scrivere altre 2 funzioni `f1_bis` ed `f2_bis` che siano equivalenti a `f1` ed `f2`, ma che contengano una sola istruzione ciascuna (dalla lezione 14)

(Lab.15). Implementare in C gli algoritmi delle Figure 8–11, aggiungendo di volta in volta un `main` e controllando che il tutto funzioni, usando sia lo schema ad input fisso di Figura 4, sia lo schema ad input da tastiera ottenuto risolvendo l'esercizio (Lab.8)

```c
#include <stdio.h>

void insertion_sort(double *A, unsigned length_of_A)
{
  unsigned j;
  int i;
  double key;

  for (j = 1; j < length_of_A; j++) {
    key = A[j];
    i = j - 1;
    while (i >= 0 && A[i] > key) {
      A[i + 1] = A[i];
      i = i - 1;
    }
    A[i + 1] = key;
  }
}

#define LENGTH_OF_A 5

int main()
{
  double A[LENGTH_OF_A] = {0.1, 1.2, 3e-1, 3.4, -1e10};

  printf("%le\t%le\t%le\t%le\t%le\n", A[0], A[1], A[2], A[3], A[4]);
  insertion_sort(A, LENGTH_OF_A);
  printf("%le\t%le\t%le\t%le\t%le\n", A[0], A[1], A[2], A[3], A[4]);
  return 0;
}
```

Figure 4: Implementazione in C dell'*Insertion Sort*

```c
#include <stdio.h>

#define LENGTH_OF_A 5

int main()
{
  double A[LENGTH_OF_A] = {0.1, 1.2, 3e-1, 3.4, -1e10};
  unsigned j;
  int i;
  double key;
  unsigned length_of_A;

  printf("%le\t%le\t%le\t%le\t%le\n", A[0], A[1], A[2], A[3], A[4]);
  length_of_A = LENGTH_OF_A;
  for (j = 1; j < length_of_A; j++) {
    key = A[j];
    i = j - 1;
    while (i >= 0 && A[i] > key) {
      A[i + 1] = A[i];
      i = i - 1;
    }
    A[i + 1] = key;
  }
  printf("%le\t%le\t%le\t%le\t%le\n", A[0], A[1], A[2], A[3], A[4]);
  return 0;
}
```

Figure 5: Versione della Figura 4 con una sola definizione di funzione

```c
unsigned long fattoriale(unsigned n)
{
  unsigned long res = 1;
  unsigned i;

  for (i = 1; i <= n; i++)
    res *= i; /* esattamente come scrivere res = res*i */
  return res;
}
```

Figure 6: Fattoriale iterativo

```
1  int f2(int arg1)
2  {
3    int c = arg1;
4
5    return c;
6  }
7
8  int f1(int arg1)
9  {
10   int a, b, c;
11
12   a = c + b*arg1;
13   return a;
14 }
```

Figure 7: Scoping di variabili (correzione 1)

```
1  Search(A, k)
2    for i ← 1 to length(A) do
3      if (A[i] = k) then
4        return i
5    return 0
```

Figure 8: Ricerca di un valore

```
1  Ordered-Search(A, k)
2    for i ← 1 to length(A) do
3      if (A[i] = k) then
4        return i
5      if (A[i] > k) then
6        return 0
7    return 0
```

Figure 9: Ricerca di un valore su un array ordinato

```
1  Binary-Search(A, p, r, k)
2    if (p > r) then
3      return 0
4    q ← ⌊(p+r)/2⌋
5    if (A[q] = k) then
6      return q
7    if (A[q] > k) then
8      return Binary-Search(A, p, q − 1, k)
9    else
10     return Binary-Search(A, q + 1, r, k)
```

Figure 10: Ricerca binaria

(Lab.16). Riscrivere il programma in Figura 4 sostituendo tutti i `for` con dei `while` (dalla lezione 14)

(Lab.17). Riscrivere il programma in Figura 4 sostituendo tutti i `while` con dei `for` (dalla lezione 14)

(Lab.18). Riscrivere il programma in Figura 4 sostituendo tutti i `while` e i `for` con dei `do...while`; farlo sia nella versione inefficiente che in quella efficiente (ovvero che usa la variabile aggiuntiva, dalla lezione 14)

(Lab.19). Scrivere un semplice programma con un `while` in cui la traduzione in `do..while` senza variabile aggiuntiva non sia corretta (usare una variabile globale...)

(Lab.20). Riscrivere il programma ottenuto dall'esercizio (Lab.18) sostituendo tutti i `do...while` con dei `for` (dalla lezione 14)

(Lab.21). Riscrivere il programma ottenuto dall'esercizio (Lab.18) sostituendo tutti i `do...while` con dei `while` (dalla lezione 14)

(Lab.22). Scrivere una funzione `numero_a_parole_con_if` che faccia la stessa cosa della funzione `numero_a_parole` di Figura 12, ma usando una catena di `if`. Aggiungere un `main` che chiami entrambe le funzioni su numeri letti da tastiera. È necessario mettere anche gli `else` di tali `if`? Se non è necessario, rende almeno la funzione più efficiente? (dalla lezione 14)

(Lab.23). Aggiungere le funzioni mancanti e il controllo di errore su `Push` e `Pop` alle funzioni in Figura 13 (dalla lezione 16)

(Lab.24). Aggiungere una funzione che scriva su schermo tutti gli elementi validi di uno stack in Figura 13 (dalla lezione 16)

(Lab.25). Realizzare una pila di `double` anziché di `int` in Figura 13 (dalla lezione 16)

```
1  Merge-Sort(A, p, r)
2    if (p < r) then
3      q ← ⌊(p+r)/2⌋
4      Merge-Sort(A, p, q)
5      Merge-Sort(A, q+1, r)
6      Merge(A, p, q, r)
7
8  Merge(A, p, q, r)
9    i ← 1
10   j ← p
11   m ← q+1
12   while j ≤ q and m ≤ r do
13     if (A[j] < A[m]) then
14       B[i] ← A[j]
15       j ← j+1
16       i ← i+1
17     else
18       B[i] ← A[m]
19       m ← m+1
20       i ← i+1
21   while j ≤ q do
22     B[i] ← A[j]
23     j ← j+1
24     i ← i+1
25   while m ≤ r do
26     B[i] ← A[m]
27     m ← m+1
28     i ← i+1
29   for i ← p to r do
30     A[i] ← B[i−p+1]
```

Figure 11: Merge Sort

```c
void numero_a_parole(unsigned n)
{
  switch (n) {
    case 1:
      printf("%u e' uno", n);
      break;
    case 2:
      printf("%u e' due", n);
      break;
    case 3:
      printf("%u e' tre", n);
      break;
    default:
      printf("%u e' piu' grande di tre", n);
      break; /* inutile */
  }
}
```

Figure 12: Esempio di `switch`

(Lab.26). Scrivere un programma che legge da tastiera dapprima un numero k (che dev'essere minore della lunghezza dell'array usato per la pila), poi legge k interi, e infine, sfruttando una pila, scrive i k interi nell'ordine inverso a quello di immissione (dalla lezione 16)

(Lab.27). Riscrivere le funzioni della pila in modo tale che lo stack `S` sia non una variabile globale, ma un parametro delle funzioni. Lasciare invece `top` come variabile globale. Provare un programma di esempio: funziona? E se anche `top` viene passata come parametro delle funzioni, il tutto funziona ancora?

(Lab.28). Scrivere l'implementazione in C delle operazioni sulla coda date in Figura 14 (con l'overflow e l'underflow, dalla lezione 16)

(Lab.29). Con riferimento alla Figura 15, scrivere scrivere la definizione della funzione `stampa_matrice`, che stampa sul monitor i valori della matrice (una linea di schermo per ogni riga della matrice, dalla lezione 17)

(Lab.30). Scrivere pseudocodice e codice di una funzione che cerca un elemento all'interno di una matrice, e ritorna la riga sulla quale lo trova, o -1 altrimenti

(Lab.31). Scrivere pseudocodice e codice di una funzione che cerca un elemento all'interno di una matrice, e ritorna la colonna sulla quale lo trova, o -1 altrimenti

```
1  #define LENGTH_S 50
2
3  /* variabili globali */
4  int S[LENGTH_S];
5  int top;
6
7  void InitStack()
8  {
9    top = -1;
10 }
11
12 void Push(int x)
13 {
14   top++;
15   S[top] = x;
16 }
17
18 int Pop()
19 {
20   top--;
21   return S[top + 1];
22 }
```

Figure 13: Operazioni su una pila

```
1  InitQueue(Q)
2    head[Q] ← 1
3    tail[Q] ← 1
4
5  Enqueue(Q, x)
6    Q[tail[Q]] ← x
7    if tail[Q] = length[Q]
8    then tail[Q] ← 1
9    else tail[Q] ← tail[Q] + 1
10
11 Dequeue(Q)
12   x ← Q[head[Q]]
13   if head[Q] = length[Q]
14   then head[Q] ← 1
15   else head[Q] ← head[Q] + 1
16   return x
17
18 QueueEmpty(Q)
19   return head[Q] = tail[Q]
20
21 QueueFull(Q)
22   return head[Q] = tail[Q] + 1
```

Figure 14: Operazioni su una coda

```c
#define NUM_COLS 3
#define NUM_ROWS 2

void SommaMatrici(int A[NUM_ROWS][NUM_COLS], int B[NUM_ROWS][NUM_COLS],
                  int C[NUM_ROWS][NUM_COLS])
{
  unsigned i, j;

  for (i = 0; i < NUM_ROWS; i++) {
    for (j = 0; j < NUM_COLS; j++)
      C[i][j] = A[i][j] + B[i][j];
  }
}

int main()
{
  int A[NUM_ROWS][NUM_COLS] = {{2, 3, 4}, {12, 13, 14}};
  int B[NUM_ROWS][NUM_COLS] = {{22, 23, 24}, {32, 33, 34}};
  int C[NUM_ROWS][NUM_COLS];
  SommaMatrici(A, B, C);
  stampa_matrice(C);
}
```

Figure 15: Operazioni su una matrice

(Lab.32). Scrivere pseudocodice e codice di una funzione che cerca un elemento all'interno di una matrice, e ritorna la riga e la colonna sulla quale lo trova, o -1, -1 altrimenti

- suggerimento: usare due variabili globali per ritornare il risultato

(Lab.33). Scrivere pseudocodice e codice di una funzione che calcola il prodotto di matrici

(Lab.34). Modificare `fattoriale.c` (vedere lezione 17) in modo che avvisi che si è superato il limite nelle varie funzioni

- ad esempio, se si chiama `ul_fattoriale(13)` su una macchina a 32 bits, deve ritornare un errore (è accettabile che ritorni 0)
- suggerimento: controllare che il numero di bits richiesti sia sufficiente usando `sizeof`

(Lab.35). Considerare, tra tutti i programmi scritti finora, quelli che usano un array, e riscriverli dichiarando tali array come puntatori (vedere lezione 19)

- in particolare, occhio all'esercizio (Lab.8): ora non è più necessario dare errore se si immette più di `LENGTH_OF_A`
- rifare anche le implementazioni di pile e code

(Lab.36). Riscrivere l'esercizio (Lab.32) usando i puntatori per ritornare i due interi di risultato

(Lab.37). Considerare l'algoritmo in Figura 16 e le sue possibili implementazioni alle Figure 17–20:

(a) si può fare di meglio che avere `unsigned` come tipo base del vettore `risultato`?

(b) aggiungere anche la conversione inversa (da binario a decimale)

(c) fare la conversione anche a base ottale ed esadecimale (e viceversa)

(d) dichiarare `risultato` come una pila, ed usare le proprietà della pila per scrivere correttamente la conversione

(e) fare la conversione binaria dichiarando `risultato` come variabile intera, e senza usare nessun vettore

```
1 DaDecimaleABinario(n ∈ ℕ)
2   H = n.
3   Finché  H ≠ 0 ripeti i seguenti passi:
4     K = H mod 2, H = H div 2.
5     Dai in output il bit K,
6       posizionandolo a sinistra del bit precedente.
```

Figure 16: Conversione da Decimale a Binario

```
1 #include <stdio.h>
2
3 void da_decimale_a_binario(unsigned long n)
4 {
5   unsigned long H = n;
6
7   while (H != 0) {
8     unsigned K = H%2;
9
10    H = H/2;
11    printf("%u", K);
12  }
13  /* la printf scrive sempre verso destra, quindi... */
14  printf("\nIl risultato va letto al contrario\n");
15  /* questo perche' l'algoritmo diceva "a sinistra" */
16 }
```

Figure 17: Implementazione ingenua dell'algoritmo di Figura 16

```c
#include <stdio.h>
/* al massimo, su una macchina a 64 bits, serviranno 64 bits per
   rappresentare un unsigned long */
#define LENGTH 64

void da_decimale_a_binario_array(unsigned long n)
{
  unsigned long H = n;
  unsigned i = 0;
  int j;
  unsigned risultato[LENGTH];

  while (H != 0) {
    unsigned K = H%2;

    H = H/2;
    risultato[i] = K;
    i = i + 1;
  }
  /* ora basta stampare il vettore risultato al rovescio */
  /* l'ultimo valore per j sara' -1: per questo e' dichiarato int e non
     unsigned */
  for (j = i - 1; j >= 0; j--)
    printf("%u", risultato[j]);
  printf("\n");
}
```

Figure 18: Implementazione con array fisso dell'algoritmo di Figura 16

```c
#include <stdio.h>
#include <stdlib.h> /* per calloc e free */

void da_decimale_a_binario_array_din(unsigned long n)
{
  unsigned long H = n;
  unsigned i = 0;
  int j;
  unsigned *risultato;

  /* primo ciclo per sapere quanti bit ha il risultato */
  while (H != 0) {
    H = H/2;
    i = i + 1;
  }
  /* il risultato ha i bit, si puo' allocare */
  risultato = (unsigned *)calloc(i, sizeof(unsigned));
  /* i ed H vanno reinizializzati, il primo ciclo li ha cambiati */
  i = 0;
  H = n;
  while (H != 0) {
    unsigned K = H%2;

    H = H/2;
    risultato[i] = K;
    i = i + 1;
  }
  for (j = i - 1; j >= 0; j--)
    printf("%u", risultato[j]);
  free(risultato);
  printf("\n");
}
```

Figure 19: Implementazione con array dinamico dell'algoritmo di Figura 16

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void da_decimale_a_binario_array_din_1loop(unsigned long n)
{
  unsigned long H = n;
  unsigned i = 0;
  int j;
  unsigned *risultato;

  /* la formula ⌊log2(n)⌋ + 1 da' direttamente il numero di bit per n */
  risultato = (unsigned *)calloc(floor(log(n)/log(2)) + 1,
                                 sizeof(unsigned));
  while (H != 0) {
    unsigned K = H%2;

    H = H/2;
    risultato[i] = K;
    i = i + 1;
  }
  for (j = i - 1; j >= 0; j--)
    printf("%u", risultato[j]);
  free(risultato);
  printf("\n");
}
```

Figure 20: Implementazione con array dinamico, con lunghezza calcolata con una formula, dell'algoritmo di Figura 16