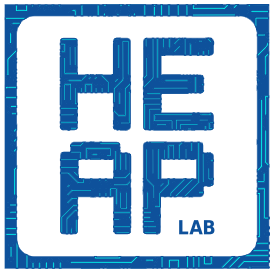
Programmazione in C

Stefano Cherubin*



04/10/2019

[**Informatica A**] Esercitazione #**5**

corso per Ing. Gestionale a.a. 2019/20

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

Indice

1 Ripasso: printf

Istruzione per **stampare** su *standard output* una stringa secondo una dato **f**ormato.

print format string $\Longrightarrow$ printf

Questa istruzione è una funzione[1] che riceve un parametro obbligatorio (la format string) e accetta un elenco di variabili da stampare, secondo il formato imposto dalla format string.

Esempi:

```
printf("Ciao");
```

Stampa «Ciao».

```
printf("Ciao\n");
```

Stampa «Ciao» e va a capo.

```
printf("%d", num);
```

Stampa la variabile *num* come un numero intero.

```
printf("%g", num);
```

Stampa la variabile num come un numero in notazione scientifica.

```
printf("%d %c", 'A', 'A');
```

Stampa 65 A.

```
int day = 11;
printf("Ciao oggi e' il %d del mese di ottobre.\n", day);
```

Stampa «Ciao oggi e' il 11 del mese di ottobre.» e va a capo.

[1] più avanti nel corso spiegheremo meglio cos'è e come si scrive una fuzione.
Per il momento possiamo definire *funzione* l'incapsulamento di un aben definita procedura.
Questa procedura è invocabile tramite una istruzione di *chiamata a funzione*, la quale riceve zero o più parametri in input e restituisce al più un valore in output.

1.1 Format string

La format string accetta diversi *placeholder* per variabili. Si consiglia di fare riferimento alla documentazione del manuale per una lista esaustiva[2]. Di seguito un riassunto dei più usati.

%c char single character

%d int signed integer

%u int unsigned decimal

%o int unsigned octal value

%x int unsigned hex value

%e (%E) float or double exponential format

%f float or double signed decimal (fixed point format)

%g (%G) float or double use %f or %e, depending on the value

%s string as sequence of characters

[2] `https://linux.die.net/man/3/printf`

2 Esempio di esercizio: Numeri non decrescenti

> Buongiorno ragazzi, benvenuti all'esame di Informatica A. Scrivete un programma in linguaggio C che risolva il seguente problema. Letti tre numeri interi a, b, c dallo standard input, stampare a terminale la sequenza dei tre numeri in ordine non decrescente.

Esempio: $a = 10$, $b = 7$, $c = 9$ deve dare in uscita 7 9 10.

2.1 Approccio al problema

Quando si parla di programmazione, la formulazione di un problema contiene quasi sempre

- vincoli
- descrizione della logica
- consigli su come trovare più facilmente una soluzione al problema
- elementi inutili o fuorvianti[3]
- casi di test o esempi di output atteso

Sta al programmatore riconoscere questi elementi e gestirli al meglio delle sue possibilità.

2.1.1 Analisi del testo dell'esercizio

Buongiorno ragazzi elemento totalmente fuorviante. Oggi dovrete concentrarvi e produrre cose sensate, non ci sarà spazio per il divertimento.

benvenuti all'esame elemento inutile. Si tratta solo di convenevoli.

di informatica A consiglio. Se una volta ricevuto in mano il foglio dell'esame vi siete scordati di cosa si sta parlando, probabilmente dovrete ricordarvi delle lezioni di Informatica A.

Scrivete un programma vincolo. L'esercizio è un test di produzione e verrete valutati sulla capacità di scrivere un programma.

in linguaggio C vincolo. In questo esercizio dovrete rispettare lo standard previsto dal linguaggio C. La capacità di attenersi a questo stanrdard sarà oggetto di valutazione.

che risolva il seguente problema elemento inutile. Dovrebbe essere sottinteso.

Letti tre numeri descrizione della logica. Dovete fare questo.

interi vincolo. Dovrete lavorare con numeri interi. Lasciate perdere numeri con virgola fissa, mobile e altre cose buffe.

a, b, c consiglio. Nello specifico caso, questo non è un brutto modo di chiamare i dati su cui lavorate.

dallo standard input vincolo. I dati dovranno essere acquisiti dallo standard input.

stampare [...] la sequenza dei tre numeri descrizione della logica.

[3] in sede di esame questi elementi tenderanno ad essere minimizzati. Al di fuori, costituiscono una parte decisamente non trascurabile.

a terminale vincolo. L'output deve essere emesso sul terminale, quindi standard output.

in ordine non decrescente vincolo. La parte di logica che non è descritta nel testo dovrà essere studiata per soddisfare questo vincolo. L'impostazione di tale logica è lasciata alla libera interpretazione del programmatore.

Esempio [...] Si tratta di un esempio. Al termine della produzione del programma, si consiglia di usare questi dati per eseguire una simulazione di esecuzione del programma e verifcare che l'output fornito in simulazione coincida con l'output atteso. Se il vostro programma funziona con l'esempio fornito, si consiglia di inventarsi altri esempi che rispettino le regole e provare anche con essi. Il programma che avete scritto non deve funzionare solo con gli esempi forniti.

2.2 Iniziare la stesura

Partendo dai vincoli e dalla logica, si possono identificare alcuni elementi ricorrenti che possono essere tradotti immediatamente in codice.

2.2.1 Scrivere un programma in linguaggio C

```
int main() {
   return 0;
}
```

2.2.2 tre numeri interi

```
int main() {
   int a, b, c;
   return 0;
}
```

2.2.3 Letti tre numeri interi dallo standard input

```
#include <stdio.h>  /* necessaria se voglio utilizzare
                       standard input o standard output */
int main() {
   int a, b, c;
   printf("\n Inserisci il numero a: ");
   scanf("%d",&a);
   printf("\n Inserisci il numero b: ");
   scanf("%d",&b);
   printf("\n Inserisci il numero c: ");
   scanf("%d",&c);
   return 0;
}
```

2.2.4 Stampare a terminale la sequenza dei numeri

```
#include <stdio.h>
int main() {
   int a, b, c;
   printf("\n Inserisci il numero a: ");
   scanf("%d",&a);
   printf("\n Inserisci il numero b: ");
   scanf("%d",&b);
   printf("\n Inserisci il numero c: ");
   scanf("%d",&c);
   printf("\n L'ordine voluto e': %d, %d, %d",a,b,c);
   return 0;
}
```

2.3 Soluzione 1 - analisi per casi

Un possibile approccio al problema è quello di valutare tutti i possibili ordinamenti dei numeri interi in input e, in base ad opportuni controlli, eseguire l'unica tra le istruzioni di output che utilizza l'ordinamento corretto.

```
#include <stdio.h> /* inclusione della libreria standard I/O */
int main( ) {
  int a,b,c;       /* dichiarazione delle variabili      */

  /* legge tre interi a,b,c dallo standard input         */
  printf("\n Inserisci il numero a: ");
  scanf("%d",&a);
  printf("\n Inserisci il numero b: ");
  scanf("%d",&b);
  printf("\n Inserisci il numero c: ");
  scanf("%d",&c);

  if (a < b) {
    if (b < c) {
      printf("\n L'ordine voluto e': %d, %d, %d",a,b,c);
    }
    else {
       if (a < c) {
         printf("\n L'ordine voluto e': %d, %d, %d",a,c,b);
      }
      else {
         printf("\n L'ordine voluto e': %d, %d, %d",c,a,b);
      }
    }
  }
   else {
    if (c < b) {
       printf("\n L'ordine voluto e': %d, %d, %d",c,b,a);
    }
    else {
       if (a < c) {
         printf("\n L'ordine voluto e': %d, %d, %d",b,a,c);
      }
      else {
         printf("\n L'ordine voluto e': %d, %d, %d",b,c,a);
      }
    }
   }
  return 0;
}
```

2.4 Soluzione 2 - ordinamento dell'input

Un secondo approccio alla risoluzione del problema prevede di fissare un solo output ed eseguire delle istruzioni utili a rendere i dati acquisiti coerenti nel loro ordinamento con quanto richiesto in output.

```
#include <stdio.h> /* inclusione della libreria standard I/O */
  int main( ) {
  int a,b,c,t; /*   dichiarazione delle variabili       */
  /* legge tre interi a,b,c dallo standard input        */
  printf("\nInserisci il numero a: ");
  scanf("%d",&a);
  printf("\nInserisci il numero b: ");
  scanf("%d",&b);
  printf("\nInserisci il numero c: ");
  scanf("%d",&c);
  /* ordinamento dei valori delle variabili a,b     */
  if (a > b) {
     /* Scambio dei valori delle due variabili a,b  */
     t = a;
     a = b;
     b = t;
  }
  /* ordinamento dei valori delle variabili a,c     */
  if (a > c) {
     /* Scambio dei valori delle due variabili a,c  */
     t = a;
     a = c;
     c = t;
  }
  /* la variabile a contiene ora sicuramente il     */
  /* valore più piccolo tra quelli inseriti.        */
  /* ordinamento dei valori delle variabili b,c     */
  if (b > c) {
     /* Scambio dei valori delle due variabili b,c  */
     t = b;
     b = c;
     c = t;
  }
  printf("\nL'ordine voluto e': %d, %d, %d",a,b,c);
  return 0;
}
```

3 Implementare il quadrato tramite somme successive

Si scriva un programma in linguaggio C che letto un numero intero positivo dallo standard input, visualizzi a terminale il quadrato del numero stesso facendo uso soltanto di operazioni di somma.

Si osservi che il quadrato di ogni numero intero positivo N può essere costruito sommando tra loro i primi N numeri dispari.

Esempio:

$$\begin{aligned} N &= 5 \\ N^2 &= 1+3+5+7+9=25 \end{aligned}$$

3.1 Approccio alla soluzione

3.1.1 Premessa

L'idea di soluzione è quella di scandire i primi N numeri dispari esprimendoli nella forma $(i + i + 1)$ al variare dell'intero $i \in [0; N - 1]$ e accumulare la loro somma man mano che si procede nella loro scansione in un'altra variabile.

$$\begin{aligned} N^2 &= \sum_{i=0}^{N-1} (2 \cdot i + 1) \\ &= \sum_{i=0}^{N-1} (i + i + 1) \\ &= (2{\cdot}0 + 1) + \ldots + (2{\cdot}i + 1) + \ldots + (2{\cdot}\,(N - 1) + 1) \end{aligned}$$

quindi si utilizzeranno almeno 3 variabili:

N numero in input

i contatore di iterazioni

S accumulatore

3.1.2 Pseudocodice (stesura informale dell'algoritmo)

1. Inizio dell'algoritmo
2. Leggi un numero intero positivo dallo standard input.
3. Inizializza un contatore i a 0
4. Inizializza un accumulatore S a 0
5. Finché il valore del contatore è minore del numero letto
   (a) Somma all'accumulatore il doppio del valore del contatore incrementato di 1
   (b) Incrementa il contatore
   (c) Torna al punto 5.
6. Stampa a terminale il valore dell' accumulatore
7. Fine dell' algoritmo

3.1.3 Blindare l'input

Quando viene richiesto che un numero in input rispetti determinate caratteristiche (in questo caso che sia positivo) è possibile scrivere il programma in modo che non accetti dati non conformi alle caratteristiche richieste e continui a chiedere all'utente di fornire dei dati fino a quando l'utente non inserisce dati corretti.

Questo è possibile farlo grazie a un ciclo do-while con controllo sul dato immesso.

```
do {

    printf("\nInserisci un numero positivo N: ");
    scanf("%d",&N);

} while (N <= 0);
```

Variante con messaggio di errore A volte può capitare che debbano essere inseriti più dati dello stesso tipo e il messaggio che invita l'utente a inserire i dati può non variare. In queste situazioni può essere utile segnalare quando si verifica un errore e non è stato accettato il dato.

```
do {

    printf("\nInserisci un numero positivo N: ");
    scanf("%d",&N);
    if (N <= 0) {


        printf ("\nErrore: il numero inserito non e' positivo.");
    }

} while (N <= 0);
```

Variante con ciclo while

```
printf("\nInserisci un numero positivo N: ");
scanf("%d",&N);
while (N <= 0) {


    printf ("\nErrore: il numero inserito non e' positivo.");
    printf("\nInserisci un numero positivo N: ");
    scanf("%d",&N);

}
```

3.2 Soluzione C

```
#include <stdio.h> /* inclusione della libreria standard  I/O */
int main( ) {

    /*        dichiarazione delle variabili                    */
    int i; /* contatore                                        */
    int N; /* variabile di cui si vuole calcolare il quadrato */
    int S; /* accumulatore per il risultato del calcolo        */
           /* ciclo di controllo che garantisce N > 0          */
    do {

        printf("\nInserisci un numero positivo N: ");
        scanf("%d",&N); /* legge N dallo standard input       */
    /* Finché il numero inserito non è positivo ripetere
       l'immissione dati.                                       */
    } while (N <= 0 );
    /* Il numero inserito è positivo                           */

    S = 0; /* inizializzazione della variabile di accumulo     */
    /* ciclo di scansione dei primi N numeri dispari           */
    i = 0;
    while(i < N) {

        /* Finché il contatore è minore del numero letto
           aggiorna il contenuto della variabile accumulatore  */
        S = S + (i+i+1);
        i = i + 1; /* incrementa il contatore                  */
    }
    printf("\n Il quadrato del numero inserito e': %d \n",S);
    return 0;
}
```

4 Numeri triangolari

Si definisce Triangolare un numero costituito dalla somma dei primi N numeri interi positivi per un certo N.

Esempio: per $Q = 10$ si ha $Q = 1 + 2 + 3 + 4$, da cui $N = 4$.

Scrivere un programma C che stabilisca se un numero intero positivo Q, letto dallo standard input, è un numero triangolare o meno, utilizzando soltanto operazioni tra numeri interi. In caso affermativo stampare a video il numero inserito e il massimo tra gli addendi che lo compongono.

4.1 Approccio alla soluzione

4.1.1 Pseudocodice (stesura informale dell'algoritmo)

Idea di soluzione: se Q–1–2–3–$\ldots$–i–$\ldots$–$N = 0$ per un certo N allora Q è Triangolare.

1. Leggi il numero positivo Q dallo standard input
2. Inizializza un contatore i a zero;
3. Memorizza in una variabile S il valore della variabile in ingresso.
4. Finché il numero S è maggiore di zero
   (a) Incrementa di 1 il valore del contatore
   (b) Sottrai a S il valore del contatore i
   (c) Torna a 4.
5. Se il valore residuo di S è zero allora
   (a) Il numero è triangolare
   (b) Il valore del massimo degli addendi è uguale al contatore i
   (c) La variabile Q contiene il valore della variabile in ingresso
6. Altrimenti il numero NON è triangolare.

4.1.2 Distruggere l'input

È buona norma, in generale, non modificare le variabili contenenti i dati in ingresso perché può accadere che sia necessario accedere a tali valori in diversi punti del programma.

Nel caso appena mostrato senza l'ausilio di un'altra variabile (S) non sarebbe possibile, al termine della computazione, stampare a video il valore del numero inserito, così come richiesto dalla traccia del problema.

4.2 Soluzione C

```
#include <stdio.h>/* inclusione della libreria standard input        */
int main( )
{                 /* dichiarazione delle variabili                  */

    int i;        /* variabile contatore                            */
    int Q;        /* variabile per il numero letto da tastiera      */
    int S;        /* variabile accumulatore                         */
    /* ciclo di controllo per l'immissione dati che garantisce Q > 0 */
    do {

       printf("\n Inserisci un numero positivo Q: ");
       scanf("%d",&Q);
    } while (Q <= 0);
    S = Q;        /* copia del valore del dato in ingresso          */
    i = 0;        /* inizializzazione del contatore                 */
    while (S > 0) {
       i = i + 1;
       S = S - i;
    }
    /* all'uscita dal ciclo il valore assunto dalla variabile S
       permette di procedere in base a due alternative              */
    if (S == 0) {

       /* il valore del contatore i contiene qui il valore del massimo
          degli addendi che compongono il numero triangolare inserito*/
       printf("\n %d = alla somma dei primi %d numeri positivi!", Q, i);
    } else {

       printf("\n Il numero %d non e' un numero triangolare! \n", Q);
    }
    return 0;
}
```

5 Conversione in binario

Dato un numero intero positivo Q,

- Scrivere la sua rappresentazione in binario naturale, applicando il tradizionale algoritmo per divisioni successive

  (per convenzione, in questo esercizio l'output si intende corretto se letto da destra a sinistra);

- Indicare anche il minimo numero di bit utilizzato.

Esempio:

Input 19_{10}

Output $\underleftarrow{11001_2}$

5.1 Approccio alla soluzione

5.1.1 LSB e MSB

LSB **L**ess **S**ignificant **B**it. Nelle rappresentazioni binarie è il bit con il peso minore. Nel sistema di numerazione posizionali è solitamente il la cifra più a destra di tutte (perché solitamente i numeri vengono letti da sinistra a destra). In questo esercizio ci viene chiesto di considerare corretto il risultato se letto da destra a sinistra. Il LSB sarà quindi il bit più a sinistra.

MSB **M**ost **S**ignificant **B**it. Cifra di valore maggiore nella rappresentazione di un numero binario. In questo esercizio sarà la cifra più a destra.

5.1.2 Indicazioni per gestire un ciclo

Per non sbagliare a impostare un ciclo, ecco alcune domande a cui è necessario dare risposta prima di scrivere codice

1. Da quali valori si deve partire?
2. Qual è la logica di ogni passo?
3. Come si aggiornano le variabili?
4. Quando si deve uscire dal ciclo?

Nello specifico caso di questo esercizio, ecco le risposte

1. Inizializzazioni:
   (a) contatore = 0
   (b) variabile da dividere = numero in input
2. Logica: divido il numero per la base (2) e scrivo il resto
3. aggiornamento variabili:
   (a) il contatore aumenta di 1
   (b) il numero viene diviso per due (approssimato all'intero inferiore)
4. Uscita dal ciclo: quando il numero raggiunge il valore 0

5.2 Soluzione C

```
#include <stdio.h> /* inclusione della libreria standard I/O */
int main( ) {

    int n; /* contatore */
    int Q; /* variabile per il numero letto da tastiera */
    /*ciclo di controllo per l'immissione dati che garantisce Q >= 0 */
    do {


        printf("\nInserisci un numero positivo o nullo Q: ");
        scanf("%d", &Q);

    } while (Q < 0);
    printf("\nIn binario: ");
    n = 0;
    do {

        printf("%d", (Q % 2));
        Q = Q / 2;
        n = n + 1;

    } while (Q != 0);
    printf("\nNumero di bit n = %d", n);
    return 0;

}
```

6 Conversione da decimale a binario

Dato un numero positivo Q, scrivere la sua rappresentazione in binario naturale, indicando anche il minimo numero di bit utilizzato.
Esempio:

Input 19_{10}

Output 5 bit, 10011_2

6.1 Approccio alla soluzione

È richiesto che l'output sia dato dalla codifica binaria naturale del numero in base 10 fornito in ingresso. È noto che per ottenere il numero in binario esiste un algoritmo per divisioni successive per 2 che fornisce il risultato in ordine inverso. Questo algoritmo può essere alla base di una soluzione oppure può essere modificato per fornire l'output in modo più conveniente.

Esistono diversi approcci a questo problema: di seguito ne sono riportati 3.

6.2 Soluzione 1 - Countdown to destruction!

Questa soluzione conta quante divisioni per 2 sono necessarie per azzerare Q. Questo numero coincide con il numero di bit che saranno necessari per rappresentare il numero in binario.

Per ogni nuovo bit si procede poi alla distruzione per divisioni successive di una copia del numero in input Q.

Listato 1: Conversione da decimale a binairo: Soluzione

```
1  #include <stdio.h>
2  int main () {
3    int Q, Qaux, current, i, n;
4    do {
5      printf("\nNumero intero : Q = ");
6      scanf("%d", &Q);
7    } while (Q < 0);
8    Qaux = Q;
9    n = 0;
10   do {
11     Qaux = Qaux / 2;
12     n += 1;
13   } while (Qaux != 0);
14   printf("\nCodifica di Q = %d con %d bit = ", Q, n);
15   for (current = n - 1; current >= 0; current--) {
16     Qaux = Q;
17     for (i = 0; i < current; i++)
18       Qaux = Qaux / 2;
19     printf("%d", Qaux % 2);
20   }
21   return 0;
22 }
```

6.3 Soluzione 2 - Raddrizzamento dell'output con algoritmo più semplice

Questa soluzione utilizza l'algoritmo per divisioni successive che fornisce l'output ordinato in ordine inverso e, tramite un accumulatore, raddrizza il risultato.

Listato 2: Conversione da decimale a binairo: Soluzione

```
1  #include <stdio.h>
2  int main () {
3    int dec, Q, n, base;
4    do {
5      printf("\nNumero intero Q non negativo : Q = ");
6      scanf("%d", &Q);
7    } while (Q < 0);
8    dec = 0;  /* accumulatore  */
9    n = 0;    /* posizione bit */
10   /* n a fine ciclo conterrà il # di bit */
11   base = 1; /* accumulatore delle potenze di 10 */
12   while (Q > 0){
13     dec = dec + (Q % 2) * base;
14     Q = Q / 2 ;
15     n = n + 1 ;
16     base = base * 10;
17   }
18   printf("\nNum. dec. emula la sequenza ");
19   printf("binaria di %d bit : %d ",n, dec);
20   return 0;
21 }
```

6.4 Soluzione 3 - Smash by power of 2

L'algoritmo da utilizzare è quello delle divisioni per decrescenti potenze di 2. Si inizia a dividere il numero per la più grande potenza di 2 che non eccede il numero in input. Si procede a dividere i resti della divisione intera per potenze di 2 decrescenti. L'output è dato dal quoziente della divisione così calcolato. L'algoritmo termina quando le potenze decrescenti di 2 arrivano a 2^0.

Il problema principale è identificare qual è la più grande potenza di due minore di un numero. Questa soluzione calcola la più piccola potenza di 2 maggiore di Q. Decrementando di uno è poi possibile calcolare la più grande potenza di 2 contenuta in Q.

Listato 3: Conversione da decimale a binairo: Soluzione

```
1  #include <stdio.h>
2  int main( ) {
3    int i, d, Q, n = 0;
4    do {
5      printf("\nInserisci un numero positivo Q: ");
6      scanf("%d", &Q);
7    } while (Q <= 0);
8    d = 1;
9    do {
10     n = n + 1;
11     d = d * 2;
12   } while (Q > d);
13 /* d è la più piccola potenza di 2 maggiore di Q */
14 /* n ha il valore dell'esponente della potenza di 2
       memorizzata in d, e coincide con il numero di bit
       minimo per rappresentare Q. */
15 /* (n-1) è il valore dell'esponente più significativo
       presente nella rappresentazione del numero Q */
16   printf("\n%d in decimale, con %d bit = ", Q, n);
17   n = n - 1;
18   d = d / 2;
19   do {
20     if (Q >= d) {
21       printf("1");
22       Q = Q - d;
23     } else {
24       printf("0");
25     }
26     n = n - 1;
27     d = d / 2;
28   } while ( n >= 0 );
29   return 0;
30 }
```

7 Intervallo di caratteri

Si scriva un programma in linguaggio C che risolva il problema seguente.

Ricevere dallo standard input due caratteri alfabetici, convertirli in maiuscolo e stampare a video ordinatamente tutti i caratteri dell'alfabeto fra essi compresi, estremi inclusi.

Esempio: dati 'g' e 'M' stampa a video la sequenza GHIJKLM.

7.1 Approccio alla soluzione

7.1.1 Input di due caratteri alfabetici

Occorre verificare che i caratteri inseriti siano caratteri alfabetici. Sfruttando la tabella della codifica ASCII è possibile identificare gli intervalli su cui fare il controllo.

```
1 do {
2   printf("Inserisci due caratteri alfabetici separati
       da spazio: ");
3   scanf("%c %c", &c1, &c2);
4 } while ((c1 < 'A') || (c2 < 'A') || (c1 > 'z') || (c2
     > 'z') || (c1 > 'Z' && c1 < 'a') || (c2 > 'Z' &&
    c2 < 'a'));
```

7.1.2 Lowercase to uppercase

Nota

In ASCII si hanno le seguenti corrispondenze tra i caratteri alfanumerici e le loro codifiche:

$'0' \div' 9' \;\; 48 \div 57$

$'A' \div' Z' \;\; 65 \div 90$

$'a' \div' z' \;\; 97 \div 122$

Per passare da minuscolo a maiuscolo bisogna sottrarre al carattere, la differenza tra i due range.

$$'a' -' A' = 97 - 65 = 32$$

```
if (c1 >= 'a') {
  c1 = c1 - 32;
}
```

Nel caso non ci fosse disponibile la tabella ASCII per calcolare la differenza tra i due intervalli, è possibile farla calcolare al programma.

```
const int diff = 'A' - 'a';
if (c1 < 'a' || c1 > 'z') {
  c1 = c1 + diff;
}
```

7.2 Soluzione C

Listato 4: Intervallo di caratteri

```
1  #include <stdio.h>
2  int main( ) {
3    char x, y; /* dati in ingresso  */
4    char t;    /* ausiliaria per scambiare */
5    int i;     /* contatore  */
6    do {
7      printf("\nInserisci il carattere alfabetico x: ");
8      scanf("%c", &x);
9    } while (!(((x >= 'a') && (x <= 'z')) || ((x >= 'A')
         && (x <= 'Z'))));
10   /* lowercase to uppercase */
11   if ((x >= 'a') && (x <= 'z')) {
12     x = x - ('a'-'A');
13   }
14   do {
15     printf("\nInserisci il carattere alfabetico y: ");
16     scanf("%c", &y);
17   } while (!(((y >= 'a') && (y <= 'z')) || ((y >= 'A')
         && (y <= 'Z'))));
18   /* lowercase to uppercase */
19   if ((y >= 'a') && (y <= 'z'))
20     y = y - ('a'-'A');
21   /* ordina i caratteri x, y in ordine crescente */
22   if (x > y) {
23     t = x;
24     x = y;
25     y = t;
26   }
27   printf("\nLa sequenza di caratteri richiesta e': ");
28   i = (int)x;
29   while (i <= (int)y) {
30     printf("%c", (char)i);
31     /* operatori di casting opzionali */
32     i = i + 1;
33   }
34   return 0;
35 }
```

8 Sequenze pari crescenti

Si codifichi un programma C che legge dallo standard input una sequenza (di lunghezza arbitraria) di interi positivi terminata dal valore 0 e, al termine della sequenza, visualizza su standard output un messaggio che indica quante terne di numeri consecutivi diversi e pari sono contenute nella sequenza.

Esempio:

input 2 50 13 16 8 6 4 6 18 6 6 16 4 1 25 0

output 4

Le sequenze sono 16-8-6 8-6-4 4-6-18 6-16-4

8.1 Approccio alla soluzione

8.1.1 La sentinella

Quando si tratta di prendere in input una sequenza di numeri, è necessario stabilire quanti numeri in input è necessario acquisire.

- È possibile che i numeri siano in un numero prestabilito noto a priori.
- È possibile chiedere all'utente la lunghezza della sequenza.
- È possibile che la sequenza abbia un terminatore che ne delimita la fine (la sentinella).

La sentinella può essere richiesta all'utente prima che l'utente inserisca il primo elemento oppure può essere nota a priori.

8.1.2 Flag

Quando una condizione è molto complessa e viene valutata in diversi punti del programma, può essere utile utilizzare per controlli una variabile ausiliaria. Questa variabile ausiliaria viene impostata come *attiva* o *disattiva* una sola volta, quando si verifica che la condizione sia vera o falsa. Per convenzione questo tipo di variabili ausiliarie vengono chiamate *flag*, bandiere di segnalazione, che si alzano e si abbassano a seconda che la condizione sia vera o falsa.

Di natura le flag sono di tipo booleano (solo vero o falso). In c99 non esiste un tipo che rappresenti una variabile booleana quindi si utilizzano variabili intere. Il valore 0 significa falso mentre un valore $\neq 0$ rappresenta il valore logico vero (di solito si utiliza 1).

8.2 Soluzione C

Listato 5: Conteggio terne positive

```
1  #include <stdio.h>
2  #define SENTINELLA 0
3  int main() {
4    int cont = 0; /* conteggio delle terne valide */
5    int prev2, prev1 = 1, cur = 1;
6    /* Le inizializzazioni di cur e prev1 a 1 sono
        innocue perché impostate con valori positivi
        dispari
7    cur, prev1, prev2: rappresentano rispettivamente il
        valore corrente e i due precedenti */
8    int cond = 0; /* flag */
9    printf("\nSeq. di int separati da spazi ");
10   printf("(termina con %d)\n\n", SENTINELLA);
11   do {
12     prev2 = prev1;
13     prev1 = cur;
14     scanf("%d", &cur);
15     cond = (prev2 > 0 && prev1 > 0 && cur >= 0);
16     cond = cond && !(prev2 == prev1 || prev2 == cur ||
           prev1 == cur);
17     cond = cond && (prev2 % 2 == 0 && prev1 % 2 == 0
          && cur % 2 == 0);
18     if (cond) {
19       cont++;
20       printf("\nNuova terna: ");
21       printf("(%d, %d, %d)", prev2, prev1, cur);
22     }
23   } while ( cur != SENTINELLA );
24   printf("\nHo trovato %d terne", cont);
25   return 0;
26 }
```

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. 2013/14.

Licenza beerware[4]

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.

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