Appello del 21 gennaio 2021
Soluzioni parte in C

Domanda 1

Dato un numero intero, scrivere un programma che restituisca una stringa in cui le cifre sono scritte al contrario.

Ad esempio:
- input: 12345 (intero)
- output: "54321" (stringa)

Completare il codice indicato qui sotto. Attenzione: non modificate la firma della funzione di conversione.

Soluzione

La difficoltà maggiore di questo esercizio è nel fatto che non è noto a priori la dimensione in byte della stringa che deve contenere la rappresentazione dell'intero (indipendentemente dall'ordine delle cifre). Vari approcci sono possibili.
La prima soluzione è quella di utilizzare un buffer sovradimensionato. In particolare, il valore massimo rappresentabile in un intero è MAX_INT. Questo valore può essere rappresentato utilizzando log10(MAX_INT) cifre. In caso di valore negativo, è necessario prevedere anche un carattere necessario a rappresentare il segno meno. In aggiunta, è necessario prevedere un carattere per conservare il terminatore di stringa. L'allocazione del buffer può quindi essere fatta come:

```
char buffer[log10(MAX_INT) + 2];
```

Questa allocazione viene risolta già a tempo di compilazione, poiché il compilatore conosce il valore di MAX_INT.
Un secondo approccio, del tutto analogo, è quello di calcolare la dimensione del buffer sull'intero effettivo che viene passato come input alla funzione:

```
char buffer[log10(numero)+2];
```

Queste soluzioni sono tutte accettabili per la prova d'esame.
Si può anche utilizzare un buffer estremamente sovradimensionato, il che è comunque una soluzoine meno elegante:

```
char buffer[1024];
```

Alcune soluzioni online propongono di utilizzare la funzione di libreria "standard" `itoa()` per effettuare la conversione. Tali suggerimenti vengono da chi non conosce la libreria standard, poiché `itoa()` non è una funzione di libreria presente nello standard C, pertanto non tutte le implementazioni della libreria standard la forniscono (ad esempio, glibc, l'implementazione più comune della libreria standard su Linux, giustamente, non la implementa). Utilizzare `itoa()`, quindi, rende il codice non portabile, il che non è accettabile.

Una soluzione tecnicamente più avanzata è quella di "simulare" la conversione da int a stringa. La conversione da intero a stringa viene tipicamente affidata alla funzione `snprintf()`. Il manuale della funzione recita:

> *Concerning the return value of snprintf(), SUSv2 and C99 contradict each other: when snprintf() is called with size=0 then SUSv2 stipulates an unspecified return value less than 1, while C99 allows str to be NULL in this case, and gives the return value (as always) as the number of characters that would have been written in case the output string has been large enough. POSIX.1-2001 and later align their specification of snprintf() with C99.*

Questo vuol dire che, dal C99 in poi, è possibile utilizzare questa funzione per "simulare" conversioni verso stringhe ed ottenere la dimensione in byte del buffer di destinazione.

Una volta ottenuta la dimensione del buffer, è possibile effettuare un'allocazione dinamica (infatti, la firma della funzione richiede di restituire un char *, rendendo perfettamente lecito l'utilizzo di malloc() all'interno della funzione) per restituire il valore numerico invertito. La conversione a stringa dell'intero può essere effettuata su stack, in una variabile locale (la dimensione in cifre dell'intero sicuramente non crea problemi rispetto alla crescita dello stack).

Una volta ottenuto il numero in formato stringa, la soluzione dell'esercizio richiede semplicemente di scandire la stringa dalla fine all'inizio, copiando un carattere alla volta.

Un'implementazione completa possibile è la seguente.

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

char *conversione(int numero)
{
	size_t len = snprintf(NULL, 0, "%d", numero) + 1;
	char buffer[len];
	char *cur = buffer;
	char *ret = malloc(len);
	ret += len - 1;


	sprintf(buffer, "%d", numero);
	*ret = '\0';
	while(*cur)
		*--ret = *cur++;

	return ret;
}

int main(void)
{
	int numero = 123456;
	char *stringa = conversione(numero);
	printf("%s\n", stringa);
}
```

Domanda 2

Implementare una funzione che calcoli:

$$f(n) = \sum_{k=1}^{n} k^2 = 1^2 + 2^2 + \ldots + n^2$$

Attenzione: non modificare la firma della funzione che calcola il valore della serie.

Soluzione

Risolvere la serie richiesta si riduce ad implementare un ciclo in cui si accumulano somme in una variabile locale che alla fine verrà restituita.
Ovviamente, la *programmazione difensiva* ci richiede qualche accorgimento maggiore. In particolare, stiamo sommando ripetutamente valori in una variabile di tipo intero. Quindi, è possibile che si verifichi un overflow. In questo caso, dobbiamo notificare il chiamante, informandolo che il risultato esatto non è calcolabile.
Una possibile implementazione è quella che segue.

```c
#include <stdio.h>

int f(int n)
{
	int accumulatore = 0;
	for(int i = 1; i <= n; i++) {
		accumulatore += i*i;
		if(accumulatore < 0) { // overflow!
			return -1;
		}
	}
	return accumulatore;
}

int main(void)
{
	printf("Il valore di f(%d) e' %d\n", 50, f(50));
}
```

Domanda 3
Implementare una funzione in C che accetti due stringhe come input e restituisca il prefisso comune più lungo (PCL) delle due stringhe.
Non vengono posti limiti sulla lunghezza massima delle stringhe fornite come input.

Esempi:
- date le stringhe "globale" e "glossario", il PCL è "glo"
- date le stringhe "dipartimento" e "dipartita", il PCL è "diparti"
- date le stringhe "casa" e "muro", il PCL è la stringa vuota ""

Attenzione: non è consentito modificare il prototipo della funzione per il calcolo del PCL.

Soluzione

L'esercizio richiede, come primo aspetto, di individuare la dimensione della stringa prefisso. Date due stringhe s1 ed s2, il prefisso potrà al più essere lungo come la *più breve* tra le due stringhe. La dimensione può quindi essere calcolata utilizzando la `strlen()`, ricordando che questa funzione calcola la lunghezza della stringa escluso il terminatore di stringa.
Determinata la dimensione massima del prefisso, la soluzione richiede di scandire contemporaneamente entrambe le stringhe, verificando fino a quando queste sono uguali, un carattere alla volta. Ogni volta che si individua un carattere comune tra le stringhe, questo deve essere copiato nella stringa di destinazione. Una volta che si individua un carattere non comune, sarà necessario inserire nella stringa che mantiene il prefisso un terminatore di stringa.
La firma della funzione, in maniera similare a tante funzioni della libreria standard (si veda ad esempio `strcat()`), richiede di memorizzare l'indirizzo del buffer allocato per mantenere il prefisso in una variabile fornita dal chiamante e di restituire lo stesso indirizzo.
Una possibile implementazione è quella che segue.

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

char *PCL(char *s1, char *s2, char **pcl)
{
	size_t s1_len = strlen(s1);
	size_t s2_len = strlen(s2);
	size_t len = (s1_len > s2_len ? s2_len : s1_len) + 1;
	size_t pos = 0;

	*pcl = malloc(len);
	if(!*pcl) {
		fprintf(stderr, "ENOMEM\n");
		exit(EXIT_FAILURE);
	}

	while(*s1 && *s2 && *s1 == *s2++)
		(*pcl)[pos++] = *s1++;
```

```
	(*pcl)[pos] = '\0';

	return *pcl;
}

int main(void)
{
	char *s1 = "supercalifragilistichespiralidoso";
	char *s2 = "superiormente";
	char *pcl;
	printf("%s\n", PCL(s1, s2, &pcl));
	return 0;
}
```

È chiaro che utilizzare la lunghezza della stringa più breve come stima per il buffer necessario a contenere il prefisso è comunque una sovrastima che può essere arbitrariamente grande (si pensi alle stringhe “supercalifragilisichespiralidoso” e “precipitevolissimevolmente”, che hanno come PCL la stringa vuota).
È possibile quindi adottare anche una soluzione simile in spirito al “vettore dinamico”, in cui il buffer contenente il prefisso viene ridimensionato con chiamate successive a realloc(), qualora lo spazio per contenere il prefisso sia esaurita. Per lo specifico problema, comunque, ritengo che sia un *overkill*, poiché rende la soluzione incredibilmente meno performante. Se lo spreco di memoria è un problema, è possibile effettuare una sola realloc() al termine dell’identificazione del prefisso.