Esercizio 1

Esercizio 1 – Composizione del nome

- Al fine di stampare degli indirizzi su delle buste, è necessario comporre la prima parte dell'indirizzo come "Cognome Nome" o "Cognome N."
- Si realizzi una funzione che riceva come parametri:
  - cognome
  - nome
  - stringa per l'indirizzo
  - lunghezza massima della stringa indirizzo
- La funzione deve copiare/concatenare nell'indirizzo il cognome seguito dal nome, avendo cura di rispettare le dimensioni della stringa indirizzo. Ovvero, qualora la stringa indirizzo sia troppo piccola per contenere entrambi, la funzione provi a comporre la stringa come "Cognome N."
- Qualora neanche ciò sia possibile, la funzione ritorni un ***codice di errore opportuno***

Esercizio 1 - Soluzione (1)

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


#define TRUE    1
#define FALSE   0

#define RESULT_OK 1
#define RESULT_ADDRESS_TOO_LONG -1
#define RESULT_COMPRESSED_NAME -2

#define MAX 50

#define BOOLEAN int;
#define RESULT int;
```

Esercizio 1 - Soluzione (2)

```
RESULT componiIndirizzo(   char * cognome,
                           char * nome,
                           char * indirizzo,
                           int maxChars)
{
   int requiredChars;
   int size;
   BOOLEAN compressNome = FALSE;
   strcpy(indirizzo, ""); //inizializzazione...
   requiredChars = strlen(cognome) + 1 + strlen(nome) + 1;
   if (requiredChars > maxChars) {
       requiredChars = strlen(cognome) + 4;
       if (requiredChars > maxChars)
              return RESULT ADDRESS TOO LONG;
       else   compressNome = TRUE;
   }
   strcat(indirizzo, cognome);
   strcat(indirizzo, " ");
   …
```


Esercizio 1 - Soluzione (3)

```
    …
    if (!compressNome)
        strcat(indirizzo, nome);
    else {
        size = strlen(indirizzo);
        indirizzo[size] = nome[0];
        indirizzo[size+1] = '.';
        indirizzo[size+2] = '\0';
        // oppure, in maniera lievemente più astratta
        // strcpy(indirizzo+size+1, ".");
    }
    if (!compressNome)
        return RESULT_OK;
    else
        return RESULT_COMPRESSED_NAME;
}
```

Esercizio 1 - Soluzione (4)

```
int main(void)
{
   char indirizzo[MAX], resultStr[MAX];
   RESULT result;

   result = componiIndirizzo("Chesani", "Federico", indirizzo, MAX);
   if ((result == RESULT_OK) ||
        (result == RESULT_COMPRESSED_NAME))
   {
        printf("%s\n", indirizzo);
        printf("Lunghezza indirizzo: %d\n", strlen(indirizzo));
   }
   else
   {
        handleError(result, resultStr); //gestione dell'errore
        printf("Errore %s\n", resultStr);
   }
   system("PAUSE");
   return (0);
}
```


Esercizio 2: array di caratteri

Codice segreto nella pagina di un libro

- Sono date due stringhe, una denominata msg e una denominata secret (non più lunga di msg), di caratteri tutti minuscoli
- Si vuole sapere se ***tutti i caratteri di secret sono contenuti nello stesso ordine (ma eventualmente intervallati da altri caratteri)*** nella stringa msg
- In caso positivo, il programma deve restituire una copia del msg originale, dove però ad ogni lettera riconosciuta come facente parte di secret viene sostituita la lettera maiuscola
  - Es: msg = "ma che bel castello", secret = "cestello"
  - Risultato: SI e stringa "ma ChE bel caSTELLO"

Esercizio 2 - Soluzione

```
BOOLEAN identifica(     char * msg,
                   char * secret,
                   char * result)
{
  while ((*msg != '\0') && (*secret != '\0'))
  {
      if (*msg == *secret)
      {
            secret++;
            *result = *msg - 'a' + 'A';
      }
      else
            *result = *msg;
      msg++;
      result++;
  }
  return (! *secret);
}
```


Esercizio 3 - Soluzione (1)

```
int convertiBin(char* bin)
{
   int res = 0, i, segno = 1;
   char absBin[MAXSIZE];
   if(bin[0]=='1')
   {
       invertiSegno(bin, absBin);
       segno = -1;
   }
   else
       strcpy(absBin, bin);
   for(i = strlen(absBin)-1; i >= 0; i--)
   {
       if(absBin[i] == '1')
             res = res + pow(2, strlen(absBin)-i-1);
   }
   return segno * res;
}
```

Il numero è negativo, quindi derivo la codifica del valore assoluto facendone il modulo due

Il numero è positivo, quindi ho già la codifica del valore assoluto

Esercizio 3 - Soluzione (1)

```
RESULT converti(int num, char *bin)
{
  int i, absNum;
  absNum = abs(num);
  if((int)log2(absNum) >= MAXSIZE-2)
     return SIZE_OVERFLOW;
  for(i = 0; i < MAXSIZE - 1; i++)
     bin[i] = '0';
  bin[MAXSIZE - 1] = '\0';
  i = 0;
```

Tolgo un posto per il bit di segno e un posto per il terminatore

Riempio tutte le celle a 0 (così non avrò posizioni scoperte)

Esercizio 3 - Soluzione (2)

```
  while(absNum != 0)
  {
     bin[i] = '0' + (absNum % 2);
     absNum = absNum / 2;
     i++;
  }
  inverti(bin);
  if(num < 0)
     invertiSegno(bin, bin);
  return OK;
}
```

Codifico il valore assoluto

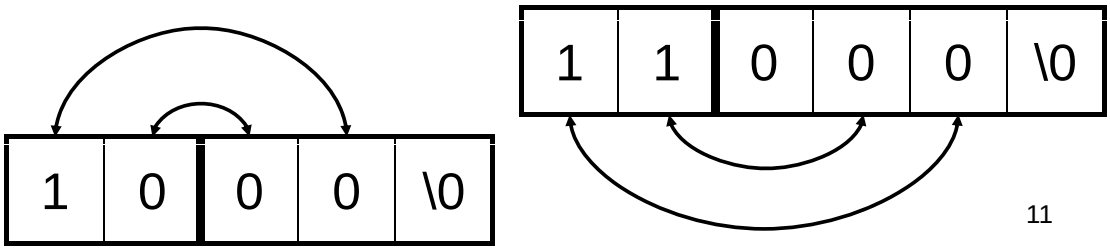
Devo ottenere la stringa “speculare” (oppure riempire il vettore bin a partire dal fondo - tanto ne conosciamo la dimensione)

Se il numero è negativo ne devo invertire il segno (modulo 2)

Esercizio 3 - Soluzione (3)

```
void inverti(char* str)
{
  int i;
  char appo;
  for(i = 0; i < strlen(str)/2; i++)
  {
      appo = str[i];
      str[i] = str[strlen(str) - i - 1];
      str[strlen(str) - i - 1] = appo;
  }
}
```

È l'intero inferiore rispetto alla metà (in caso di elementi dispari quello in mezzo rimane dov'è)



Esercizio 3 - Soluzione (4)

```
void invertiSegno(char *num, char *res)
{
  int i;
  char one[MAXSIZE];
  converti(1, one);
  for(i = 0; i < strlen(num); i++)
  {
      res[i] = '0' + (num[i] == '0' ? 1 : 0);
  }
  res[MAXSIZE - 1] = '\0';
  sum(res, one, res);
}
```

Inversione dei bit

Aggiunta di 1

Esercizio 3 - Soluzione (5)

```
RESULT sum(char *add1, char *add2, char *res)
{
  int i;
  int carry = 0;
  int curCarry = 0;
  int value;
  BOOLEAN overflow;
  if( strlen(add1) != strlen(add2))
      return GENERIC_ERROR;
  …
```

Verifico che le stringhe abbiano la stessa lunghezza

Esercizio 3 - Soluzione (6)

```
for(i = strlen(add1) - 1; i >= 0; i--)
   {
      curCarry = carry + (add1[i]-'0') + (add2[i]-'0') >= 2;
      res[i] = '0' + (carry != (add1[i]-'0' != add2[i]-'0' ?
                        1 : 0) ? 1: 0);
       if(i==0)
              overflow = curCarry != carry ? 0 : 1;
       carry = curCarry;
   }
   res[MAXSIZE - 1] = '\0';
   if(overflow)
       return SIZE_OVERFLOW;
   else
       return OK;
}
```

Procedo dal bit meno significativo fino al bit di segno

Prima il calcolo del nuovo carry se voglio poter sovrascrivere un addendo (se res è add1 o add2, in questo passo sovrascrivo l'i-mo bit)
NOTA: res[i] si calcola facendo xor tra i due bit i-mi degli addendi e il carry

Se i == 0 il carry corrente è quello del segno, quindi faccio x-or con il carry precedente per l'overflow

XOR

- L’algoritmo può diventare più chiaro introducendo una funzione che esegua l’exclusive or fra due bit

```
char xor(char bit1, char bit2)
{
     return bit1 == bit2 ? 0 : 1;
}
```

Esercizio: integrare la funzione di cui sopra nel codice…