ESERCIZIO 1 (Visibilità delle variabili + Passaggio parametri per copia, per copia con puntatori, per reference)
Simulare l'esecuzione di ciascuna delle 5 chiamate a sottoprogrammi, limitandosi alla descrizione dei valori stampati tramite cout << ….

```
#include <iostream.h>
#include <stdlib.h>
int varA = 12, varB = 25;
int proc1(int *par) {
      *par = *par - 7;
      return(23);
}
void proc2(int *varX, int varY) {
      int varB;

      *varX = varY;
      varB = *varX + varY;
}
int proc3(int varA) {
      int varZ = 17;

     varA = varA + varZ;
      return(varA);
}
int proc4(int &varA) {
      int varZ = 17;

     varA = varA + varZ;
      return(varA);
}
int main() {
   int varC = 63;         /* situazione iniziale */

      varC = proc1(&varA);   /* passo 1 */
      cout << "\n1) varA = " << varA;
      cout << ", varB =  " << varB << ", varC = " << varC;

      proc2(&varA, varC);    /* passo 2 */
      cout << "\n2) varA = " << varA;
      cout << ", varB = " << varB << ", varC = " << varC;

      varB = proc3(varC);    /* passo 3 */
      cout << "\n3) varA = " << varA;
      cout << ", varB = " << varB << ", varC = " << varC;

      varB = proc1(&varB);   /* passo 4 */
      cout << "\n4) varA = " << varA;
      cout << ", varB = " << varB << ", varC = " << varC;

      varB = proc4(varC);   /* passo 5 */
      cout << "\n5) varA = " << varA;
      cout << ", varB = " << varB << ", varC = " << varC;
      system("PAUSE");
      return 0;
}
```

Variabili visibili dal main: **varA** **varB** **varC**

situazione iniziale **12** **25** **63**

dopo *varC = proc1(&varA);* **5** **25** **23**

```
int proc1(int *par) {
    *par = *par - 7;
    return(23);
}
```

dopo *proc2(&varA, varC);* **23** **25** **23**

```
void proc2(int *varX, int varY) {
    int varB;

    *varX = varY;
    varB = *varX + varY;
}
```

dopo *varB = proc3(varC);* **23** **40** **23**

```
int proc3(int varA) {
    int varZ = 17;

    varA = varA + varZ;
    return(varA);
}
```

dopo *varB = proc1(&varB);* **23** **23** **23**

```
int proc1(int *par) {
    *par = *par - 7;
    return(23);
}
```

dopo *varB = proc1(varC);* **23** **40** **40**

```
int proc4(int &varA) {
    int varZ = 17;

    varA = varA + varZ;
    return(varA);
}
```

Esercizio 2 (Allocazione dinamica dei vettori)

Allocazione dinamica di un vettore: richiedere all'utente la dimensione di un vettore di interi; allocarlo dinamicamente; leggere dallo *stdin* il contenuto del vettore stesso e ristamparlo a video.

```
#include <iostream>
#include <stdlib.h>
void main(){

  double *p=NULL;
  int n,i;

  cout << "Inserisci la dimensione del vettore " << endl;

  cin >> n;
  p = new double[n];

  cout << "Inserisci le " << n << " componenti del vettore "<< endl;
  for (i=0; i < n;i++){
    cin >> p[i];
  }
  cout << "Puntatore alla prima componente " << p << endl;
  cout << "Componenti del vettore ";

  for (i=0;i < n;i++){
    cout << " " << p[i];
  }
  cout  << endl;

  delete[] p;

  system("PAUSE");
  return;

}
```

Esercizio 3 (Allocazione dinamica matrici)

Allocazione dinamica di una matrice.
Dichiarare una matrice statica x[2][3] di double contenente per ogni cella un numero incrementale: a partire da zero e procedendo da sinistra a destra e dall'alto in basso.
Allocare dinamicamente una matrice identica alla precedente, assegnarne i valori come nel caso precedente; stampare a video la matrice ottenuta e de-allocarla.

```
#include <iostream.h>
#include <stdlib.h>
void main(){
  int i,j;
  // Matrice allocata staticamente
  double x[2][3]={ 0.0,  1.0, 2.0,
                   3.0,  4.0, 5.0  };
  cout << " Valori della matrice: " << endl;
  for (j=0; j < 2; j++) {
    cout << endl;
    for (i=0; i < 3; i++) {
       cout << x[j][i] << " ";
    }
  }
  cout << endl;

  // Matrice allocata dinamicamente
  int nrow,ncol;
  double **matrix;

  nrow = 2 ;  ncol=3;
  matrix = new double*[nrow];

  for (i=0; i < nrow; i++){
    matrix[i] = new double[ncol];
  }

  for (i=0; i < nrow; i++){
    cout << endl;
    for (int j=0; j < ncol; j++){
      matrix[i][j] = i*ncol+j;
      cout << " " << matrix[i][j];
    }
  }
  for (i=0; i < nrow; i++){
    delete[] matrix[i];
  }
  delete[] matrix;

  system("PAUSE");
}
```

ESERCIZIO 3 **(Scomposizione in sottoprogrammi + definizione di sottoprogrammi con più di un "parametro di ritorno" [passaggio per reference])**

Scrivere un programma in linguaggio C che legga dallo standard input due sequenze di interi positivi ognuna terminata dal numero 0 (escluso).
Relativamente alla sequenza più lunga si deve stampare a terminale: l' elemento massimo con il suo numero d'ordine e l'elemento minimo con il suo numero d'ordine e infine l'intera sequenza in ordine inverso.

La soluzione del problema richiede l'utilizzo di array monodimensionali e l' esecuzione dei seguenti passi:

1. Acquisizione della prima sequenza di interi e calcolo della sua lunghezza
2. Acquisizione della seconda sequenza di interi e calcolo della lunghezza
3. Calcolo del massimo fra le due lunghezze
4. Calcolo dell'elemento massimo in una sequenza e del suo numero d'ordine
5. Calcolo dell'elemento minimo in una sequenza e del suo numero d'ordine
6. Stampa a terminale di una sequenza in ordine inverso

per evitare la duplicazione di pezzi di codice utilizzeremo:

- una funzione per la lettura delle due sequenze che restituirà la lunghezza della sequenza acquisita e al contempo **do**vrà anche memorizzare tale sequenza in un array dichiarato nel programma chiamante.
  N.B.: quest'ultima operazione sarà effettuata prevedendo un vettore di interi come primo argomento della funzione e un parametro di tipo intero passato per *reference* (per fare esercizio) per ottenere la lunghezza del vettore:

  void LeggiSeq(int A[], int& len);

  Si ricorda, come a seguito della dichiarazione di un vettore per es. **int A[100];** si ha che:

  - la variabile A viene trattata dal compilatore come un puntatore costante di tipo intero cioè come si fosse dichiarato: const int *A;
  - A è equivalente a &a[0];
  - A[i] è equivalente a *(A + i)

  E quindi la dichiarazione di un parametro formale di tipo vettore nel prototipo di una funzione (o procedura) fa sì che al momento dell'invocazione del sottoprogramma il parametro attuale coincida con l'indirizzo della prima cella del vettore. All'interno del corpo del sottoprogramma si farà poi riferimento alle celle del vettore definito nel programma chiamate nel solito modo (es:A[i]) grazie all'equivalenza implicita sopra riportata.
- una procedura per il calcolo del massimo e del suo numero d'ordine in una sequenza che dovrà prevedere come parametri in ingresso il vettore su cui effettua il controllo e la sua lunghezza (passaggio per valore) e come parametri d'uscita l'elemento massimo della sequenza e il suo numero d'ordine (passaggio per reference).
- Una procedura analoga alla precedente per il calcolo del minimo.
- Infine si può prevedere anche una procedura per la stampa a terminale e una procedura per l'inversione di una sequenza, perché essendo questa operazione ben definita è sempre buona norma, quando possibile, utilizzare sottoprogrammi, per aumentare la leggibilità, la modularità e la verifica del programma.

```
#include <iostream.h>
#include <stdlib.h>
#define MAX_LEN 100

void leggi_seq(int v[], int& len);
void scrivi_seq(int v[], int len);
void trova_max(int v[], int len, int& max, int& index);
void trova_min(int v[], int len, int& min, int& index);
void inverti_seq(int len, int v_in[], int v_out[]);

int main() {


    int a[MAX_LEN],b[MAX_LEN],c[MAX_LEN];
    int len_a, len_b, len_c;
    int max,min;
    int ind_max, ind_min;

    leggi_seq(a, len_a);
    leggi_seq(b, len_b);

    if (len_a > len_b) {

        trova_max(a,len_a,max,ind_max);
        trova_min(a,len_a,min,ind_min);
        inverti_seq(len_a,a,c);
        len_c = len_a;

    }
    else {

        trova_max(b,len_b,max,ind_max);
        trova_min(b,len_b,min,ind_min);
        inverti_seq(len_b,b,c);
        len_c = len_b;

    }
    cout << "\n la sequenza piu' lunga ha: \n";
    cout << "\n massimo = “ << max << “ in posizione “ << ind_max;
    cout << "\n minimo  = “<< min << “ in posizione " << ind_min;
    cout << "\n la sequenza invertita e': \n";
    scrivi_seq(c,len_c);

    system(“PAUSE”);
    return 0;
}
```

```
void leggi_seq(int v[], int& len) {

     int i = 0;

     cout << "\nInserire una seq. di interi terminata con 0.";
     do {
         cout << "\n inserisci elemento: ";
         cin >> v[i];
          i++;
     } while ((v[i-1]!=0)&&(i < MAX_LEN));

    if (i == MAX_LEN) len = MAX_LEN;
    else len = i-1;
}


void scrivi_seq(int v[], int len) {

     int i;

     cout << "\nla sequenza e': \n";
     for (i = 0; i < len; i++)
          cout << " " << v[i];
}




void trova_max(int v[], int len, int& max, int& index) {

     int i;

     index = 0;
     max = v[0];
     for (i = 1; i < len; i++) {
          if (v[i] > aux) {
             max = v[i];
             index = i;
          }
     }
}
```

```
void trova_min(int v[], int len, int& min, int& index) {

     int i;

     index = 0;
     min = v[0];
     for (i = 1; i < len; i++) {
         if (v[i] < aux) {
            min = v[i];
            index = i;
          }
     }
}

void inverti_seq(int len, int v_in[], int v_out[]) {

      int i;
      for (i = len-1; i>=0; i--) {
           v_out[len-1-i] = v_in[i];
      }
}
```

ESERCIZIO 4 **(Definizione di funzioni + vettori in passaggio parametri + manipolazione stringhe + algoritmi)**

Scrivere un programma in linguaggio C che definendo opportune funzioni, risolva il seguente problema. Date due stringhe S1 e S2, stabilire se S2 compare come sottosequenza di S1 e, in caso affermativo, a partire da quale indice. Creare quindi una nuova stringa S3 costruita come S1 meno la prima occorrenza della stringa S2. Non è permesso utilizzare le funzioni di libreria del linguaggio relative alla manipolazione di stringhe.

Esempio:

S1 = “melograno” ,
S2 = “grano”;

restituisce 5, perché S2 è inclusa in S1 a partire dal quinto carattere e

S3 = “melo”.

Se fosse S1 = “melogranograno”,
S2 = “grano”;
il risultato sarebbe ancora 5, ma S3 = “melograno”.

Le funzionalità che il programma deve possedere saranno realizzate definendo tre funzioni e precisamente:

```
int lung_str(char s[]);

int inclusione_str(char s1[], char s2[]);
// ritorna -1 se s2 non è contenuta in s1; altrimenti ritorna
// indice della cella in cui inizia l’occorrenza di s2 in s1;

void sottrai_str(char s1[], char s2[], char s3[]);
// s3 conterrà tutti i caratteri di s1 tranne la prima
// occorrenza della stringa s2.
```

```
#include <iostream.h>
#include <stdlib.h>
#define MAX_LEN 100

int lung_str(char s[]);
int inclusione_str(char s1[], char s2[]);
void sottrai_str(char s1[], char s2[], char s3[]);

int main() {

   char stringa1[MAX_LEN], stringa2[MAX_LEN];
   int res;

   cout << "\n inserisci la prima stringa:   ";
   cin >> stringa1;

   cout << "\n inserisci la seconda stringa: ";
   cin >> stringa2;

   res = inclusione_str(stringa1,stringa2);

   if (res == -1)  {
      cout << "\n la stringa < “ << stringa2 << “ > “;
     cout << “non e' contenuta nella stringa < “;
     cout << stringa1 << “ > !";
   }
   else {
      cout << "\n l'inizio della prima occorrenza di < “;
      cout << stringa2 << “ > in < “ << stringa1 << “ > e' “;
      cout << " a cominciare dal carattere n. " << 1+res;
     sottrai_str(stringa1,stringa2,stringa3);
     cout << "\nla stringa differenza e’: s3 = “ << stringa3;
   }
   system(“PAUSE”);
   return 0;
}


int lung_str(char s[]) {

  int i = 0;
  while (s[i] != '\0')
       i++;
  return i;
}
```

```
int inclusione_str(char s1[], char s2[]) {

   int len1,len2;
   int trovato;
   int i,j,new_i;

   len1 = lung_str(s1);
   len2 = lung_str(s2);

   if (len1 < len2)
      return -1;

   for (i = 0; i <= len1-len2; i++) {

      j = 0;
      trovato = 1;
      new_i = i;
       while ((j < len2)&&(trovato)) {

            trovato = (s1[new_i] == s2[j]);
            j++;
           new_i++;
       }
      if ((j == len2)&&(trovato))
         return i;
   }
   return -1;
}


void sottrai_str(char s1[], char s2[], char s3[]) {

    int index,len1,len2,i;

     index = inclusione_str(s1,s2);
     len1 = lung_str(s1);
    len2 = lung_str(s2);

     for (i = 0; i < index;i++)
        s3[i] = s1[i];

    for (i = index + len2; i < len1; i++)
        s3[i-len2] = s1[i];

    s3[i-len2] = s1[i]; /* per aggiungere il char '\0' */
}
```

Una sintassi alternativa nella scrittura dei sottoprogrammi precedenti sarebbe potuta essere la seguente:

Prototipi

```
int lung_str(char *s);
int inclusione_str(char *s1, char *s2);
void sottrai_str(char *s1, char *s2, char *s3);
```

definizioni

```
int lung_str(char *s) {

  int i = 0;

  while (*s != '\0') {
       s++;
       i++;
  }
  return i;
}
int inclusione_str(char *s1, char *s2) {

   int len1,len2;
   int trovato;
   int i,j,new_i;

   len1 = lung_str(s1);len2 = lung_str(s2);

   if (len1 < len2) return -1;

   for (i = 0; i <= len1-len2; i++) {

      j = 0;
      trovato = 1;
      new_i = i;
       while ((j < len2)&&(trovato)) {

            trovato = (*(s1+new_i) == *(s2+j));
            j++;
           new_i++;
       }
      if ((j == len2)&&(trovato))
         return i;
   }
   return -1;
}
```

```
void sottrai_str(char *s1, char *s2, char *s3) {

    int index,len1,len2,i;

     index = inclusione_str(s1,s2);
     len1 = lung_str(s1);len2 = lung_str(s2);

     for (i = 0; i < index;i++)
        *(s3+i) = *(s1+i);

     for (i = index + len2; i < len1; i++)
        *(s3+i-len2) = *(s1+i);

    *(s3+i-len2) = *(s1+i);  // per aggiungere il char '\0'
}
```