Esercizio 1 – Scambio lettere

Scrivere la funzione

```
void scambioLettere(char *dest, char *lettere, int p_o_d)
```

che modifichi la stringa destinazione (dest), sostituendone i caratteri pari o dispari (a seconda del valore di `p_o_d`, 0 o 1 rispettivamente), in sequenza, con quelli contenuti nella stringa lettere, fino al loro esaurimento.

Esempio:
`dest` = "parolagigante", `lettere` = "PROVA"
`dest` (dopo l'invocazione di tipo "pari") = "pPrRlOgVgAnte"

Soluzione:

```
void scambioLettere(char* dest, char* lettere, int p_o_d) {
   int i, j;
   if (strlen(dest) < strlen(lettere) / 2)
     return;
   i = 1; j = 0;
   while (i < strlen(lettere) {
      if ( ((i % 2 == 1 && p_o_d == 0) || (i % 2 == 0 && p_o_d == 1))
           && j < strlen(lettere) ) {
        dest[i] = lettere[j];
        i += 2;
        j += 1;
      } else {
        i += 1;
      } // end else-if
   } // end while
} // end scambioLettere
```

Esercizio 2 – memoria dinamica. Media Mobile

Scrivere un programma che, data una sequenza arbitraria di numeri in virgola mobile immessi uno alla volta dall'utente, stampi man mano la media degli ultimi n valori immessi, dove anche n viene immesso dall'utente all'avvio del programma.
In ogni momento l'utente può scegliere se

a) aggiungere un valore alla sequenza,
b) modificare il valore di n, oppure
c) terminare il programma.

I numeri sono immessi fino all'immissione del valore "sentinella" 0 che termina il programma.

Ogni volta che è immesso un numero il programma mostra la media mobile aggiornata. Quando, all'inizio, sono stati inseriti meno di n numeri, o quando n viene aumentato, la media viene comunque calcolata sui valori disponibili.
Si badi a trattare correttamente anche il caso in cui n viene diminuito.

Suggerimenti:
• Si usi come "*buffer circolare*" un *array* di n elementi allocato dinamicamente per memorizzare gli ultimi n valori inseriti: i valori sono aggiunti uno dopo l'altro, ripartendo dall'inizio quando si raggiunge la fine dell'*array*.
• Si scomponga il problema in sottofunzioni

Soluzione:

```
#include <iostream>
using namespace std;

char menu(void) {
    char c;

    cout << endl << "Premi 'a' - per inserire un valore";
    cout << endl << "Premi 'b' - per modificare il valore di n";
    cout << endl << "Premi 'c' - per terminare il programma";
    cout << endl;
    do {
      cin.get(c);
    } while (c != 'a' && c != 'b' && c != 'c');
    cout << endl;
    return c;
}

struct circularBuffer {
   double *buffer;
   int length_buffer;
   int num_elements;
   int current_index; // indice della cella in cui scrivere il prossimo valore
};
```

```
circularBuffer* initializeCircularBuffer(int length_buffer) {

    circularBuffer* cb = new circularBuffer;

    cb->buffer = new double[length_buffer];
    cb->length_buffer = length_buffer;
    cb->idx_last = 0;
    cb->idx_first = 0;

    return cb;
}

circularBuffer* insertCircularBuffer(circularBuffer* cb, int num) {

   cb->idx_last = (cb->idx_last  + 1) % cb->length_buffer;
   cb->buffer[cb->idx_last] = num;
   if (cb->idx_first >= cb->idx_last)
     cb->idx_first = (cb->idx_first  + 1) % cb->length_buffer;

   return cb;
}

circularBuffer* resizeCircularBuffer(circularBuffer*cb, int new_size){

  if (new_size < 1 || new_size == cb->length_buffer) return cb;
  double* aux = new double[new_size];

  int idx_scrittura, idx_lettura;

  idx_lettura = cb->idx_first;
  idx_scrittura = 0;
  count = 0;
  while ( idx_lettura <= cb->idx_last ) {
       aux[idx_scrittura] = cb->buffer[idx_lettura];
       idx_lettura = (idx_lettura + 1) % cb->num_elements;
       idx_scrittura += 1;
       count++;
  }

  cb->idx_first = 0;
  cb->idx_last = count;
  cb->length_buffer = new_size;

  delete[] cb->buffer;
  cb->buffer = aux;

  return cb;
}
```

```
int main() {

  int n = 0, num;

  do {
      cout << endl << "Numero di elementi su cui mediare (>=1), n =";
      cin >> n;
  } while (n < 1);

  circularBuffer* cb = initializeCircularBuffer(n);
  char choice;

  do {

    choice = menu();
    switch(choice) {

      case 'a' :
                 cout << "\n Inserisci un numero intero: ";
                 cin >> num;
                 if (num != 0)
                   cb = insertCircularBuffer(cb, num);
                 break;

      case 'b' :
                 cout << "\n Inserisci nuovo valore (>=1), n: ";
                 cin >> n;
                 cb = resizeCircularBuffer(cb, n);
                 break;

      case 'c' :

      default  : break;
    }

    double avg = 0.0;
    for (int i = 0; i < cb->num_elements; ++i) {
      cout << cb->buffer[i];
      avg += cb->buffer[i];
    }
    avg /= cb->num_elements;
    cout << "\n media (ultimi " << n << " elementi): " << avg << endl;

  } while (num != 0 && choice != 'c');

  delete[] cb;
  return 0;
} // end main
```

Esercizio 3 – Lista palindroma

Sia data una lista semplicemente concatenata i cui nodi contengano un singolo carattere. Verificare se la parola rappresentata dalla lista sia palindroma o meno.

Attenzione: esistono moltissimi modi di procedere, utilizzando variamente ogni combinazione di strutture dati ausiliarie o funzioni di supporto, magari combinando più scansioni della lista.
Si cerchi di trovare una soluzione ragionevolmente semplice ed efficace.

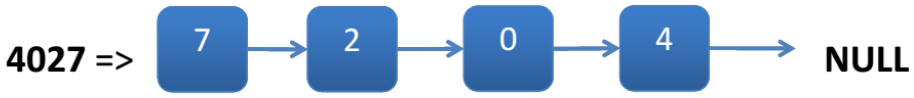
```
struct node {
   char info;
   node *next;
};

int lenList(node* head) {
   if (head == NULL) return 0;
   return 1+lenList(head->next);
}


bool IsPalindrom(node* head) {
   int  lung = lenList(head)-1;
   node* t, h = head;
   for (int i = 0; i < len/2; i++) {
      t = head;
      for (int j = 0; j < lung; j++)
         t = t->next;
      if (h->info != t->info)
        return false;
      h = h->next;
      lung = lung-1;
   }
   return true;
}
```

Esercizio 4 – Somma di interi (in lista)

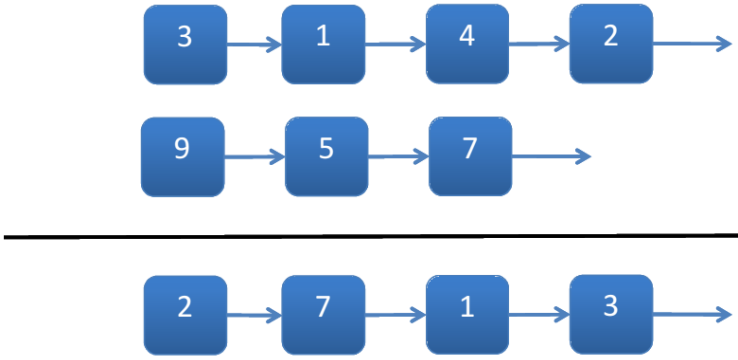
Dato un intero positivo espresso in base 10, possiamo convertirlo in una lista semplicemente concatenata, fatta di elementi che contengono solo interi compresi tra 0 e 9, che rappresentano le cifre decimali che lo codificano in modo che la cifra più significativa sia in coda alla lista e la meno significativa sia in testa:



1) Scrivere una funzione che converta un intero in una lista di cifre.
2) Scrivere una funzione che data una lista di cifre restituisce l'intero rappresentato
3) Scrivere una funzione di somma, che, date due liste, restituisca una terza lista, che rappresenti il numero ottenuto dalla somma delle prime due, simulando l'algoritmo di addizione in colonna.
4) Scrivere un sottoprogramma che riporti in una riga vuota di un *file* il numero intero corrispondente a quanto memorizzato nella lista.
5) Scrivere un sottoprogramma che legge i numeri interi memorizzati ciascuno su una diversa riga di un *file*, il cui nome è passato come parametro, ed effettua la loro somma utilizzando i sottoprogrammi 1) e 3).

NOTA: Non «vale» (se volete giocare a questo gioco ☺) una soluzione "easy" che converta le liste in intero, esegua la somma, e la converta nuovamente in lista!

Esempio: 2413 + 759 = 3172



Soluzione:

```
struct node {
   unsigned int data;
   node *next;
};

node* intToList(int n) {
   if (n < 1) return NULL;

   node *head = NULL, *aux;
   do {
       aux = new node;
       aux->data = n % 10;
       aux->next = head;
       head = aux;

       n = n / 10;

   } while (n != 0);

   return head;
}

int listToInt(node *head) {
   unsigned int pot = 1, num = 0;
   while (head != NULL) {
     num = num + pot*head->data;
     pot = pot*10;

     head = head->next;
   }
   return num;
}
```

```
node *sumList(node *h1, node *h2) {
    if (h1 == NULL) return h2;
    if (h2 == NULL) return h1;

    node *aux, *head = NULL;
    unsigned int digit, carry = 0;

    while (h1 != NULL && h2 != NULL) {
      digit = (carry + h1->data + h2->data);
      carry = (h1->data + h2->data) / 10;

      aux = new node;
      aux->data = digit;
      aux->next = head;
      head = aux;

      h1 = h1->next; h2 = h2->next;
    }
    while (h1 != NULL) {
      digit = (carry+h1->data);
      carry = (carry+h1->data) / 10;

      aux = new node;
      aux->data = digit;
      aux->next = head;
      head = aux;

      h1 = h1->next;
    }
    while (h2 != NULL) {
      digit = (carry+h2->data);
      carry = (carry+h2->data) / 10;

      aux = new node;
      aux->data = digit;
      aux->next = head;
      head = aux;

      h2 = h2->next;
    }
    if (carry != 0) {
      aux = new node;
      aux->data = digit;
      aux->next = head;
      head = aux;
    }

    return head;
}
```

```
void dumpList(node* head, char fileName[]) {
   ofstream fout;
   fout.open(fileName, ios::app);
   if (fout.fail()) return;

   int n = listToInt(head);
   fout << n;

   fout.close();
}

void deleteList(node* & h) {
   node *aux;
   while (h != NULL) {
     aux = h;
     h = h->next;
     delete aux;
   }
}

node* sumFromFile(char fileName[]) {

  ifstream fin;
  fin.open(nomefile);
  if (fin.fail()) return 0;

  node *sum, *aux = NULL, curr;

  while (!fin.eof()) {
     fin >> n;
     curr = intToList(n);
     sum = sumList(aux, curr);
     deleteList(aux);
     aux = sum;
  }
  return sum;
}
```

Esercizio 5 – Print(f)-a-Ring-o-Roses

Dovendo intrattenere e tenere a bada dei bambini (un po' selvaggi), organizzate un semplice girotondo: ogni bimbo che arriva deve prender posto nel cerchio in ordine alfabetico, dando una mano al bambino immediatamente precedente e l'altra a quello immediatamente successivo. Quando un bambino se ne va, i suoi due vicini si danno la mano per chiudere il cerchio.
I bambini guardano sempre verso il centro del cerchio.

Nel momento in cui sono annunciati un nome e una direzione (destra/sinistra), tutti i bambini devono gridare (su stdout) il proprio nome, in ordine, a partire da quello chiamato e avanzando via via nel cerchio, passando di mano destra in mano destra (senso antiorario), o di mano sinistra in mano sinistra (senso orario).

- Scrivere un sotto-programma `...fun(…)` che implementi questo scenario, assumendo che il gruppo di bambini sia rappresentato con una lista dinamica doppiamente concatenata in cui ciascun nodo contiene il nome del bambino, un puntatore per il nodo a sinistra e un puntatore per nodo a destra e che la funzione riceva come parametri:
  la lista, il nome del bambino che parla per primo, e un numero intero per indicare il verso in cui devono parlare gli altri bambini (0 – sinistra, 1 – destra).

Soluzione:

```
struct node {
   char childName[20+1];
   node *left, *right;
};

node *findNode(node* girotondo, char name[]) {
   while (girotondo != NULL &&
          strcmp(girotondo->childName, name) != 0)
      girotondo = girotondo->next;
   return girotondo;
}

// direction: 0 sinistra, 1 destra
void fun(node* girotondo, char name[], int direction) {
    node* ptr, ptrStart = find(girotondo, name);

    if ( prtStart == NULL ) return;
    if (direction == 0)
      for (ptr = ptrStart; ptr != ptrStart; ptr = ptr->left)
         cout << ptr->childName << endl;
    else
      for (ptr = ptrStart; ptr != ptrStart; ptr = ptr->right)
         cout << ptr->childName << endl;
}
```

Esercizio TE - 1

L'agenzia viaggi GV - Grandi Viaggi vi commissiona l'implementazione della funzione `...AssegnaVolo(...)`

Tale funzione riceve due liste dinamiche, una riguardante i voli a disposizione, con i campi data, ora, numero di volo, aeroporto origine, aeroporto destinazione, posti disponibili per l'agenzia. La seconda lista descrive i clienti che hanno acquistato un pacchetto vacanze, con i campi nome, cognome, data partenza, luogo della vacanza, numero partecipanti, puntatore al volo; quest'ultimo vale `NULL` se al cliente non è stato ancora assegnato un volo.

La funzione `...AssegnaVolo(...)` deve associare per ciascun cliente l'opportuno volo, tenendo conto della data di partenza e della destinazione della vacanza. Per ciascun volo della lista voli con un numero di posti disponibili maggiore di zero, si assegneranno i viaggi non ancora assegnati, così che l'assegnazione a uno specifico volo terminerà quando i posti saranno tutti esauriti. L'assegnazione di un volo al cliente consiste nell'assegnare allo specifico campo nella lista dei clienti l'indirizzo del nodo che descrive il volo assegnato, decrementando il numero di posti disponibili nel nodo che descrive il volo. Il processo termina quando non ci sono più clienti senza volo assegnato o voli con posti disponibili.

Dopo aver dichiarato le opportune strutture dati, implementare la funzione richiesta in C/C++.

```
struct volo {
   char data[10+1], ora[5+1];
   unsigned int numVolo;
   char origine[40+1], destinazione[40+1];
   unsigned int postiDisponibili;
   volo* next;
};
struct cliente {
  char nome[40+1], cognome[40+1], dataPartenza[10+1];
  char luogo[40+1];
  unsigned int numPartecipanti;
  volo* aereo;
  cliente* next;
};
void AssegnaVolo(volo* lv, cliente* lc) {
   volo *v; cliente *c = lc; bool assegnato;
   while (c != NULL) {
      v = lv; assegnato = false;
      while (v != NULL && assegnato == false) {
         if (strcmp(c->dataPartenza, v->data) == 0  &&
             strcmp(c->luogo, v->destinazione) == 0 &&
             v->postiDisponibili >= c->numPartecipanti) {
           c->aereo = v;
           v->postiDisponibili -= c->numPartecipanti;
           assegnato = true;
         }
         v = v->next;
      }
      c = c->next;
   }
}
```

Esercizio TE - 2

Il magazzino della EmmePi SpA è suddiviso in settori, in base al reparto in cui i beni sono impiegati. Vi viene commissionata l'implementazione della funzione

`...GiacenzaElevata(...)`

che riceve come parametro il *nome del reparto* e *due liste*. *La prima lista* contiene il codice del prodotto (stringa di 15 caratteri), il nome del reparto di appartenenza e il prezzo del prodotto. *La seconda lista* contiene il codice del prodotto (stringa di 15 caratteri), il codice del magazzino (numerico) e la giacenza nel magazzino (Nota: un prodotto può essere presente in più magazzini).
La funzione restituisce il numero di prodotti del reparto indicato che hanno una giacenza complessiva superiore alle 100 unità e una giacenza minima **in ciascun magazzino** superiore alle 10 unità. Definire le strutture dati delle due liste e implementare la funzione in C/C++.

```
struct prodotto {
   char codice[15+1], reparto[15+1];
   double prezzo;
   prodotto *next;
};

struct magazzini {
   char codProdotto[15+1];
   unsigned int codMagazzino, giacenza;
   magazzini *next;
};

void GiacenzaElevata(char nomeReparto[], prodotto* lp, magazzini* lm){
  magazzini *m;
  prodotto *p = lp;
  unsigned int num = 0, totGiacenza = 0;
  bool flagGiacenzaMinima10;

  while (p != NULL) {
    if (strcmp(p->reparto, nomeReparto) == 0) {

     flagGiacenzaMinima10 = true;
     for (m=lm; m!=NULL && flagGiacenzaMinima10==true; m=m->next) {
        if (strcmp(p->codice, m->codProdotto)==0 && m->giacenza > 10)
          totGiacenza += m->giacenza;
        if (strcmp(p->codice, m->codProdotto)==0 && m->giacenza <= 10)
          flagGiacenzaMinima10 = false;
     }
     if (flagGiacenzaMinima10 == true && totGiacenza > 100) num++;
   }
   p = p->next;
 }
 return num;
}
```

Esercizio TE - 3

Si definisca la struttura dati che definisce una lista dinamica di nominativi (lunghezza massima di 100 caratteri), dove ogni nominativo ha associato un codice numerico (intero).
Si consideri anche un file che contiene un elenco di righe e ciascuna riga contiene 6 caratteri che rappresentano un codice identificativo, seguiti da un massimo di 100 caratteri che contengono il nominativo vero e proprio.
Si scriva quindi la funzione

`...CreazioneListaDaFile(...)`

che restituisce l'indirizzo della testa di una lista di nominativi e riceve due parametri: il *nome del file* da leggere e *una lettera*. La lista restituita conterrà solo i nominativi che **non** contengono la lettera ricevuta come secondo parametro nei suoi primi 10 caratteri.

```
struct node {
  char nominativo[100+1];
  unsigned int codice;
  node *next;
};

node* CreazioneListaDaFile(char fileName[], char c) {
    ifstream fin;
    fin.open(fileName);
    if (fin.fail()) return NULL;

    node* head = NULL, *aux;
    char codStr[6+1] = {'\0'}, nominativoStr[100+1] = {'\0'};

    while (!fin.eof()) {

       fin.getline(codStr, 6+1, '\n');
       fin.getline(nominativoStr, 100+1, '\n');

       int i = 0;
       while( i < 10 && nominativoStr[i] != '\0'
                     && c != nominativoStr[i] ) {
         i = i + 1;
       }

       if (i == 10 || nominativoStr[i] == '\0') {
         aux = new node;
         strcpy(aux->nominativo, nominativoStr);
         aux->codice = atoi(codStr);
         aux->next = head;
         head = aux;
       }
    }
    fin.close();
    return head;
}
```

Esercizio TE - 4

Si consideri la struttura dati che definisce una lista dinamica di nominativi (lunghezza massima di 100 caratteri). Si scriva quindi la funzione

… `CreazioneGuidataLista(…)`

che restituisce l'indirizzo della testa di una lista di nominativi. La funzione riceve tre parametri: una lista di nominativi da elaborare, un vettore di caratteri e la dimensione del vettore.
La funzione crea la nuova lista nel modo seguente: i nominativi nella nuova lista vengono presi dalla lista data in base alla lettera iniziale, secondo l'ordine delle lettere riportato nel vettore. Per esempio, se il vettore contiene i tre caratteri `'C', 'Z', 'c'` ; nella nuova lista ci saranno, nell'ordine, tutti i nominativi della lista data che iniziano con `'C'`, poi quelli che iniziano con `'Z'`, quindi quelli che iniziano con `'c'`.
La nuova lista deve mantenere l'ordine, quindi si deve effettuare un inserimento in coda.

```
struct node {
   char nominativo[100+1];
   node *next;
};

node *CreazioneGuidataLista(node *head, char v[], unsigned int len) {
   node *h, *newHead = NULL, *aux, *last;

   for (int i = 0; i < len; ++i) {
      h = head;
      while ( h!= NULL ) {
         if ( h->nominativo[0] == v[i] ) {
             aux = new node;
             strcpy(aux->nominativo, h->nominativo);
             aux = NULL;

             if ( newHead == NULL ) {
               newHead = aux;
               last = aux;
             } else {
                      last->next = aux;
                      last = last->next;
             }
         }
         h = h->next;
      }
   }
   return newHead;
}
```