Laboratorio 3

Lo scopo è sviluppare delle funzioni che lavorano con vettori di interi. Sul sito si trova un file cpp che si chiama **lab3.cpp** che serve come punto di partenza dove trovate le seguenti funzioni già implementate:

- `alloca_vettore`: serve per allocare memoria per un vettore,
- `dealloca_vettore`: per liberare la memoria occupata da un vettore,
- `legge_vettore`: legge numeri dalla tastiera e li mette in un vettore,
- `stampa_vettore`: stampa un vettore a video,
- `numero_casuale`: genera un intero casuale.

Sono presenti due prototipi senza implementazione ma con qualche suggerimento:

- `vettore_casuale`: crea un vettore con numeri casuali,
- `somma_vettori`: calcola la somma di due vettori.

Ogni funzione deve essere testato nel modulo **main** di **lab3.cpp** (quindi si può lavorare con un singolo file aggiungendo le funzioni e modificando il modulo **main** per testarle).

Esercizio 1

Completare la funzione `vettore_casuale`**.** Testare la funzione nel **`main`**.

Esercizio 2

Realizzare le seguenti funzioni che effettuano semplici operazioni su vettori di interi:

1. somma di due vettori (completare la funzione che c'è già),
2. differenza di due vettori,
3. prodotto di uno scalare e un vettore,
4. prodotto scalare di due vettori,
5. calcolo della norma di un vettore.

Esercizio 3

Realizzare una funzione che dato un vettore v con n elementi interi e due indici inf e sup (tali che $0 \leq inf \leq sup \leq n-1$) restituisce l'indice dell'elemento massimo del sottovettore $\{v[inf], \dots, v[sup]\}$. Per esempio, se il vettore è (3,2,5,2,6,8,4,2,5,7,1,2) e inf e sup sono 4 e 9 allora la funzione deve restituire 5 perché nel sottovettore in questione (indicato qua in grassetto: (3,2,5,2,**6,8,4,2,5,7**,1,2)) l'elemento massimo è 8 e il suo indice nel vettore intero è 5.

Esercizio 4 (ordinamento per selezione ma una versione leggermente diversa rispetto a quella vista in aula)

Realizzare una funzione che ordina gli elementi di un vettore v con n elementi interi implementando i seguenti passi:

1. sia max il risultato della ricerca dell'indice dell'elemento massimo in $\{v[0], \dots, v[n-1]\}$; scambia $v[n-1]$ con $v[max]$
2. sia max il risultato della ricerca dell'indice dell'elemento massimo in $\{v[0], \dots, v[n-2]\}$; scambia $v[n-2]$ con $v[max]$

3. sia max il risultato della ricerca dell'indice dell'elemento massimo in $\{v[0], \dots, v[n-3]\}$; scambia $v[n-3]$ con $v[min]$
4. …

Per cercare l'indice del massimo utilizzare la funzione sviluppata precedentemente.

Studiare il tempo di esecuzione del programma variando la dimensione dell'array fra 100 e 200000 utilizzando lo schema

```
double start = clock();
selection_sort(n,v);
double end = clock();
double seconds = (end - start) / CLOCKS_PER_SEC;
std::cout << seconds;
```

dove `clock()` restituisce il numero di "tick" della CPU utilizzati dal processo sin da quando è partito e `CLOCKS_PER_SEC` è un costante utile per sapere quanti "tick" ci sono in un secondo. Per utilizzare questi elementi bisogna includere la libreria standard `ctime`.

Quante volte l'algoritmo confronta una coppia di elementi durante l'esecuzione nel caso di un vettore di n elementi? La crescita asintotica del numero di confronti è uguale alla crescita asintotica del tempo di esecuzione.

Esercizio 5

Realizzare una funzione che partizioni un segmento di un vettore secondo il primo elemento del segmento, chiamato *perno*. La funzione deve avere tre parametri: il vettore v, l'indice del primo elemento del segmento, inf, e l'indice dell'ultimo elemento del segmento, sup. Si sottintende che $0 \leq inf < sup \leq n-1$ dove *n* è il numero di elementi del vettore. La funzione deve riorganizzare gli elementi del segmento in modo tale che all'inizio del segmento si trovano gli elementi minori o uguali al perno, poi segue il perno, e infine tutti gli elementi maggiori del perno. La funzione deve restituire l'indice del perno dopo il partizionamento.

Per esempio, se il vettore è (8 19 7 **14 11 15 5 19 1 19 6 1 10** 4 2 7 12 4) e inf e sup sono 3 e 12 allora il partizionamento riguarda il segmento (14 11 15 5 19 1 19 6 1 10) e il perno è 14. La funzione deve riorganizzare il vettore in modo tale che il vettore diventi {8 19 7 **6 11 10 5 1 1 <u>14</u> 19 19 15** 4 2 7 12 4} e deve restituire 9.

La funzione può essere realizzata secondo il seguente algoritmo:

1. Sia p il perno, cioè $p = v[inf]$.
2. Usiamo due indici, i e j, per indicare la parte del segmento che non è stato ancora partizionato. Cioè all'inizio $i = inf + 1, j = sup$.
3. Se una parte del segmento è da partizionare, cioè se $i \leq j$, allora
    a. se $v[i] \leq p$ allora incrementa i e torna al passo 3
    b. se $v[i] > p \wedge v[j] > p$ allora decrementa j e torna al passo 3
    c. se $v[i] > p \wedge v[j] \leq p$ allora scambia $v[i]$ con $v[j]$, incrementa i, decrementa j e torna al passo 3
4. Scambia $v[inf]$ con $v[j]$.
5. Restituisci j.

Per capire bene come funziona l'algoritmo, conviene simularlo per un vettore non grande seguendo passo per passo come cambiano le variabili e il vettore.

Esercizio 6 (quick-sort)

Realizzare un algoritmo di ordinamento ricorsivo utilizzando il partizionamento precedente. L'idea di **quick-sort** è la seguente:

1. Caso base: un vettore con meno di due elementi è ordinato e quindi non bisogna fare niente.
2. Meccanismo ricorsivo:
    1. Partiziona il vettore secondo il primo elemento. Sia p il valore restituito dalla funzione che effettua il partizionamento.
    2. Applica quick-sort stesso sulla prima parte del vettore, cioè sugli elementi che hanno indice minore di p.
    3. Applica quick-sort stesso sulla seconda parte del vettore, cioè sugli elementi che hanno indice maggiore di p.

Per realizzare l'idea precedente conviene definire una funzione col prototipo

```
void quick_sort(int v[], int inf, int sup);
```

dove v è il vettore in questione e inf e sup indicano la parte del vettore che deve essere ordinata. Per ordinare tutto il vettore basta poi fare una chiamata `quick_sort(v,0,n-1)` dove `n` è il numero di elementi in `v`.

Studiare il tempo di esecuzione di quick-sort variando la dimensione dell'array fra 100 e 200000 come nel caso di selection-sort.

Esercizio 7 (massima sequenza crescente)

Le funzioni da sviluppare nell'esercizio 7 e 8 devono seguire la seguente logica perché devono restituire un vettore cui numero di elementi non si conosce a priori. I passaggi sono:

1. si determina il numero di elementi, *n*;
2. si alloca memoria per un vettore di *n* elementi;
3. si riempie il vettore con gli elementi richiesti;
4. si restituisce il vettore restituendo un puntatore (il numero di elementi si restituisce con passaggio di parametro per riferimento).

Scrivere una funzione che individua la massima sequenza crescente di un vettore di interi e la restituisce utilizzando un puntatore. Il numero di elementi della massima sequenza crescente deve essere restituito con passaggio di parametro per riferimento. Per esempio, se il vettore in input è (5,3,6,7,2,5,8,9,5,8) la sua massima (o più lunga) sequenza crescente è (2,5,8,9). (Se la massima sequenza crescente non è unica la funzione deve restituire una delle sequenze. Per esempio, nel vettore (3,4,5,2,1,4,5,3) ci sono due sequenze crescenti di lunghezza 3 e la funzione può scegliere quale restituire.)

La funzione deve seguire lo schema seguente:

1. individuare l'indice del primo elemento e la lunghezza della massima sequenza crescente;
2. allocare memoria per un vettore che conterrà la massima sequenza crescente;
3. riempire il vettore del punto 2 con la massima sequenza crescente;
4. restituire un puntatore che punta al vettore che contiene la massima sequenza crescente (numero di elementi restituito con passaggio di parametro per riferimento).

Esercizio 8 ("differenza" fra due vettori)

Scrivere una funzione che, dati due vettori *v1* e *v2*, individua tutti gli elementi di *v1* che non sono presenti in *v2*. Per esempio, con *v1*=(4,8,3,6,4,9,10) e *v2*=(8,6,2,1,10) la "differenza" è (4,3,4,9). La differenza deve essere restituito in un vettore (utilizzando un puntatore).

Lo schema diventa:

1. determinare il numero di elementi della differenza;
2. allocare memoria per un vettore che conterrà la differenza;
3. riempire il vettore del punto 2 con la differenza;
4. restituire un puntatore che punta al vettore che contiene la differenza (numero di elementi restituito con passaggio di parametro per riferimento).