Esercitazione 2 Aprile

Template di funzioni

Un po’ di teoria

Un po’ di teoria [Prova pratica del 14 giugno 2019]

- Completare il codice del seguente template di funzione C++ e del programma che lo utilizza.
- *Esempio*: dato l’array di caratteri *s* = “abcde” (*n* = 6, compreso il carattere nullo), allora il template di funzione *dimezza*, istanziato con il tipo concreto *char* e chiamato con *q* = 1, restituisce un puntatore all’array di caratteri *w* = “bd”. L’array risultante contiene, cioè, soltanto gli elementi che nell’array originale si trovavano in una posizione di indice dispari.

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

_________________ <typename T>
T* dimezza (T v[], int n, int q){
        if (n%2 != 0) return NULL;
        _________________ p = new T[n/2];
        int k = 0;
        if (q != 0) k = 1;
         _________________ (k < n) {
                p[k/2] = v[k];
                k += 2; }
        return p;
}
```

```
int main() {
    const int dim = 6;
                char s[dim] = "abcde";
                char* w = dimezza(s, dim, 1) ;
                for (_________; i < dim/2; i++)
                                cout << w[i] << endl;
                _________________ w;
                return 0;
}
```

Un po’ di teoria

Un po’ di teoria [Prova pratica del 14 giugno 2019]

- Completare il codice del seguente template di funzione C++ e del programma che lo utilizza.
- *Esempio*: dato l’array di caratteri *s* = “abcde” (*n* = 6, compreso il carattere nullo), allora il template di funzione *dimezza*, istanziato con il tipo concreto *char* e chiamato con *q* = 1, restituisce un puntatore all’array di caratteri *w* = “bd”. L’array risultante contiene, cioè, soltanto gli elementi che nell’array originale si trovavano in una posizione di indice dispari.

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

template <typename T>
T* dimezza (T v[], int n, int q){
        if (n%2 != 0) return NULL;
        T* p = new T[n/2];
        int k = 0;
        if (q != 0) k = 1;
        while (k < n) {
                p[k/2] = v[k];
                k += 2; }
        return p;
}
```

```
int main() {
    const int dim = 6;
                char s[dim] = "abcde";
                char* w = dimezza(s, dim, 1) ;
                for (int i = 0; i < dim/2; i++)
                                cout << w[i] << endl;
                delete[] w;
                return 0;
}
```

[Esercizio C041] Template di funzioni

- Si scriva il template di funzione C++ *contaDistinti* che riceva come parametri un array *a* di elementi di tipo T e la sua dimensione *n* (un numero intero), calcoli e restituisca come valore di ritorno il numero di elementi distinti contenuti nell'array *a* (un numero intero).
- Si scriva quindi un programma per verificare il corretto funzionamento del template di funzione.
  Il programma chiederà all'utente di inserire da tastiera i valori per un array di 10 numeri interi, per un array di 10 numeri reali e per una stringa contenente al massimo 9 caratteri, chiamerà *contaDistinti* per ciascuno dei tre array e stamperà a video i tre valori di ritorno. Tali operazioni potranno essere ripetute finché l'utente lo desidera.
- *Nota*: per conoscere la dimensione effettiva della stringa inserita dall'utente, si può usare la funzione *strlen* disponibile nella libreria *cstring*.
- *Esempio*: dato l'array *a* = {1, 3, 5, 6, 5, 2, 1, 5, 3, 6} (*n* = 10), la funzione restituirà 5. L'array *a* contiene cioè 5 valori distinti (per la precisione si tratta dei valori: 1, 2, 3, 5, 6).

[Esercizio C042] Template di funzioni

Da fare a casa: estendere il template di funzione *contaDistinti* in modo che riceva come parametro un ulteriore array *f* di numeri reali, della stessa dimensione dell'array *a*. La funzione assegnerà a ciascun elemento di *f* la frequenza del corrispondente elemento di *a*. La frequenza di un elemento di un array è definita come il numero di volte in cui l'elemento compare nell'array diviso per la dimensione dell'array. Nel caso dell'esempio di prima, l'array *f* sarà dunque il seguente:

f = {0.2, 0.2, 0.3, 0.2, 0.3, 0.1, 0.2, 0.3, 0.2, 0.2}.

L'elemento 1 compare cioè 2 volte su 10, l'elemento 3 compare 2 volte su 10, l'elemento 5 compare 3 volte su 10 e così via.

[Esercizio C044] Template di funzioni e puntatori

- Implementare un programma C++ che utilizza template delle funzioni per lo scambio di valori e la stampa di array di interi e float:
  - definire un template di funzione **swapValues** che scambi i valori a cui puntano due puntatori dello stesso tipo generico T
  - definire un template di funzione **printArray** che stampi un array composto da elementi di tipo generico T
  - Nel **main**, dichiarare due array di dimensione 5, uno di interi e uno di float.
    - Utilizzare la funzione **swapValues** per invertire l'ordine degli elementi all'interno di ciascun array.
    - Stampare gli elementi degli array di interi e float utilizzando le rispettive funzioni **printArray**.

[Esercizio C046] Template di funzioni e puntatori

- Implementare un programma C++ che utilizza template, puntatori a funzione e puntatori a template di funzione per eseguire operazioni matematiche su due numeri:
  - definire un template di funzione **sum** che prende in input due numeri di tipo generico e restituisce la loro somma
  - definire un template di funzione **product** che prende in input due numeri di tipo generico e restituisce il loro prodotto
  - definire un template di funzione **executeOperation** che prende in input:
    - due valori di tipo generico T
    - un puntatore a funzione che prende due argomenti dello stesso tipo generico e restituisce un valore dello stesso tipo,
    - la funzione **executeOperation** restituisce il risultato dell'operazione specificata su quei numeri.
  - Nel **main**, dichiarare due numeri interi e due numeri reali e eseguire le seguenti operazioni utilizzando **executeOperation**:
    - somma dei due numeri
    - prodotto dei due numeri

[Esercizio C047] Template e matrici

- Si desidera realizzare un programma C++ per eseguire semplici elaborazioni su matrici quadrate di elementi di tipo generico. A tale scopo si implementino i seguenti template di funzione:
  - Il template di funzione ***somma*** che riceva come parametri un puntatore *pM* a un puntatore a un oggetto di tipo T e un numero intero *n* e restituisca come valore di ritorno un oggetto di tipo T. Il puntatore *pM* punta a una matrice quadrata di *n* righe e *n* colonne di oggetti di tipo T. La funzione calcola e restituisce come valore di ritorno la somma degli elementi della matrice puntata da *pM*.
  - Il template di funzione ***prodotto*** che riceva come parametri un puntatore *pM* a un puntatore a un oggetto di tipo T e un numero intero *n* e restituisca come valore di ritorno un oggetto di tipo T. Il puntatore *pM* punta a una matrice quadrata di *n* righe e *n* colonne di oggetti di tipo T. La funzione calcola e restituisce come valore di ritorno il prodotto degli elementi della matrice puntata da *pM*.
  - Il template di funzione ***traccia*** che riceva come parametri un puntatore *pM* a un puntatore a un oggetto di tipo T e un numero intero *n* e restituisca come valore di ritorno un oggetto di tipo T. Il puntatore *pM* punta a una matrice quadrata di *n* righe e *n* colonne di oggetti di tipo T. La funzione calcola e restituisce come valore di ritorno la traccia degli elementi della matrice puntata da *pM*. Si ricorda che la traccia di una matrice quadrata è data dalla somma degli elementi sulla diagonale principale.

- Si scriva quindi un programma C++ che operi come segue:
  - Chieda all'utente di inserire da tastiera la dimensione *d* (un numero intero) delle matrici quadrate che si vogliono elaborare.
  - Allochi dinamicamente una matrice di *d* x *d* numeri reali. A tale scopo, dichiari un puntatore a un puntatore a un numero reale *pData*, allochi dinamicamente un array di *d* puntatori a numeri reali e ne assegni il puntatore a *pData*, allochi dinamicamente *d* array di numeri reali e ne assegni i puntatori agli elementi dell'array puntato da *pData*.
  - Chieda all'utente di inserire da tastiera il valore di ciascun elemento della matrice così allocata.
  - Chiami *somma*, *prodotto* e *traccia* e stampi a video i loro valori di ritorno.
  - Deallochi la memoria precedentemente allocata e termini.

[Esercizio C048] [*Prova del 14 giugno 2019, Testo 1, Esercizio 5*]

- Si scriva il template di funzione *Sottocampiona* che riceva come parametri un array *a* di elementi di tipo generico T, la sua dimensione *n* (un numero intero positivo) e un secondo numero intero *m* tale che $0 < m \leq n / 2$. Il template di funzione restituisce come valore di ritorno un puntatore a un array di elementi di tipo generico T e opera in questo modo: calcola $k=\lceil n/m \rceil$ (cioè il quoziente della divisione di *n* per *m*, approssimato al numero intero immediatamente superiore), alloca dinamicamente un array *s* di *m* elementi di tipo generico T, copia in *s* un elemento ogni *k* dell'array *a*, partendo dal primo elemento di *a*.
  - Ad esempio, assumendo che T venga istanziato con il tipo concreto *int* e che siano dati *a* = {3, 2, 8, 5, 4, 9, 6, 1, 2, 1} (*n* = 10) e *m* = 3, il template di funzione allocherà un array *s* di 3 elementi e lo riempirà copiandovi un elemento ogni $k=\lceil 10/3 \rceil=4$ elementi di *a* (copierà quindi il primo, il quinto e il nono elemento di *a*), cioè si avrà *s* = {3, 4, 2}. Se il valore di *n* o *m* non è valido, il template di funzione restituisce NULL. Si supponga che il template di funzione venga istanziato con un tipo concreto dotato di operatore di assegnamento.

- Si scriva quindi un programma C++ che operi come segue: dichiari un array *v* di 10 numeri interi e chieda all'utente di inserirne i valori da tastiera, chieda all'utente di inserire un numero intero *t* compreso tra 1 e 5 (in caso di inserimento di un valore non valido, il programma chiederà di ripetere l'inserimento finché non pervenga un valore valido), chiami il template di funzione *Sottocampiona* passando l'array *v*, la sua dimensione e la dimensione del nuovo array *t* come parametri, stampi a video i valori contenuti nell'array puntato dal puntatore che il template di funzione restituisce come valore di ritorno (tale array avrà dimensione *t*), deallochi tale array e termini.