1. Lavoro

Scrivi un programma in C++ che:

1. Definisca un nuovo tipo `lavoro` utilizzando `typedef` per rappresentare un'attività lavorativa.
2. Ogni attività (`lavoro`) deve contenere:
   - **ore**: un numero intero che rappresenta le ore lavorative.
   - **minuti**: un numero intero che rappresenta i minuti lavorativi (0-59).
   - **costoAlMinuto**: un numero in virgola mobile che rappresenta il costo per minuto.
3. Utilizzi un array per gestire un insieme di lavori.
4. Contenga le seguenti funzioni:
   - `riempiRnd(lavoro arr[], int n)`: assegna valori casuali ai campi di ogni elemento dell'array.
   - `costoTotale(const lavoro arr[], int n)`: calcola il costo totale di tutti i lavori dell'array.
   - `stampa(const lavoro arr[], int n)`: stampa i dettagli di ogni lavoro nell'array.

svolto Lavoro

```
#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <ctime>



using namespace std;



// Definizione del tipo "lavoro" tramite typedef

typedef struct {

    int ore;
```

```
    int minuti;
    float costoAlMinuto;
} lavoro;

// Prototipi delle funzioni
void riempiRnd(lavoro arr[], int n);
float costoTotale(const lavoro arr[], int n);
void stampa(const lavoro arr[], int n);

int main() {
    const int NUM_LAVORI = 5; // Numero di lavori da gestire
    lavoro lavori[NUM_LAVORI]; // Array di lavori

    // Inizializza il generatore di numeri casuali
    srand(time(0));

    // Riempi l'array con valori casuali
    riempiRnd(lavori, NUM_LAVORI);

    // Stampa i dettagli dei lavori
    stampa(lavori, NUM_LAVORI);

    // Calcola e visualizza il costo totale
    float totale = costoTotale(lavori, NUM_LAVORI);
```

```
    cout << "\nCosto totale di tutti i lavori: " << fixed << setprecision(2) << totale << endl;

    return 0;
}

// Funzione che riempie l'array con valori casuali
void riempiRnd(lavoro arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i].ore = rand() % 8; // Ore casuali tra 0 e 7
        arr[i].minuti = rand() % 60; // Minuti casuali tra 0 e 59
        arr[i].costoAlMinuto = (rand() % 300) / 100.0f; // Costo tra 0.00 e 2.99
    }
}

// Funzione che calcola il costo totale di tutti i lavori
float costoTotale(const lavoro arr[], int n) {
    float totale = 0.0f;
    for (int i = 0; i < n; i++) {
        int totaleMinuti = arr[i].ore * 60 + arr[i].minuti;
        totale += totaleMinuti * arr[i].costoAlMinuto;
    }
    return totale;
}
```

```
// Funzione che stampa i dettagli dei lavori

void stampa(const lavoro arr[], int n) {

    cout << left << setw(10) << "Ore" << setw(10) << "Minuti" << setw(15) <<
"Costo/Minuto" << "Costo Totale" << endl;

    cout << string(50, '-') << endl;


    for (int i = 0; i < n; i++) {

        int totaleMinuti = arr[i].ore * 60 + arr[i].minuti;

        float costo = totaleMinuti * arr[i].costoAlMinuto;


        cout << setw(10) << arr[i].ore

            << setw(10) << arr[i].minuti

            << setw(15) << fixed << setprecision(2) << arr[i].costoAlMinuto

            << fixed << setprecision(2) << costo << endl;

    }

}
```

2. massimo e media

scrivi una funzione che ricevuto come parametro un array di interi e la sua dimensione, restituisca una struct contenente il valore massimo, la media, il numero di coppie.

svolto massimo&media

```
#include <iostream>
```

```
using namespace std;

// Definizione della struct per contenere i risultati
struct Risultati {
    int valoreMassimo; // Valore massimo nell'array
    float media;       // Media dei valori dell'array
    int numDuplicati;  // Numero di valori duplicati
};

/* Definizione del tipo "lavoro" tramite typedef
typedef struct {
    int valoreMassimo; // Valore massimo nell'array
    float media;       // Media dei valori dell'array
    int numDuplicati;  // Numero di valori duplicati
} Risultati; */

// Prototipo della funzione
Risultati analizzaArray(const int arr[], int n);

// Funzione principale
int main() {
    const int DIM = 10;
    int array[DIM] = {4, 2, 8, 4, 5, 8, 2, 1, 3, 5}; //max 2 valori =

    // Chiamata alla funzione e visualizzazione dei risultati
    Risultati risultati = analizzaArray(array, DIM);

    cout << "Valore massimo: " << risultati.valoreMassimo << endl;
    cout << "Media: " << risultati.media << endl;
    cout << "Numero di coppie: " << risultati.numDuplicati << endl;

    return 0;
}

// Funzione che analizza l'array e restituisce i risultati
Risultati analizzaArray(const int arr[], int n) {
    Risultati risultati = {arr[0], 0.0f, 0}; // Inizializzazione della struct

    int somma = 0;
    int duplicati = 0;

    // Calcolo del massimo e della somma
    for (int i = 0; i < n; i++) {
        if (arr[i] > risultati.valoreMassimo) {
```

```
            risultati.valoreMassimo = arr[i];
        }
        somma += arr[i];
    }

    // Calcolo della media
    risultati.media = (float)(somma) / n;

    // Conteggio dei valori duplicati
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                duplicati++;
                break; // Evita di contare più volte lo stesso duplicato
            }
        }
    }
    risultati.numDuplicati = duplicati;

    return risultati;
}
```

3. Intervalli lineari geometrici

Sviluppare alcune funzioni per gestire strutture lineari di tipo ”intervallo”.
Un intervallo è definito da due punti a, b tali per cui a <= b
funzioni:

a. bool isIntersect(.., ..) //i due intervalli sono intersecati
b. bool isDisjoint (.., ..) //i due intervalli sono disgiunti
c. bool contains (.., ..) // il primo intervallo contiene il secondo
d. bool isContained (.., ..) // il secondo è contenuto nel primo
e. bool isAntecedent (.., ..) // il secondo è antecedente al primo
f. bool isSubsequent;(.., ..) // il secondo è successivo al primo
g. interval create(double p1, double p2); //crea un intervallo a partire da 2 punti
h. double lenght(interval i); //restituisce la lunghezza dell’intervallo
i. interval translate(interval i, double amount); //trasla l’intervallo della quantit amount

svolto Intervalli lineari geometrici

```c
#include <stdio.h>

// Struttura per rappresentare un intervallo
typedef struct {
    double start;
    double end;
} Interval;

// Funzione che verifica se due intervalli sono intersecati
int isIntersect(Interval interval1, Interval interval2) {
    return (interval1.end >= interval2.start && interval1.start <= interval2.end);
}

// Funzione che verifica se due intervalli sono disgiunti
int isDisjoint(Interval interval1, Interval interval2) {
    return (interval1.end < interval2.start || interval1.start > interval2.end);
}

// Funzione che verifica se il primo intervallo contiene il secondo
int contains(Interval interval1, Interval interval2) {
    return (interval1.start <= interval2.start && interval1.end >= interval2.end);
}

// Funzione che verifica se il secondo intervallo e' contenuto nel primo
int isContained(Interval interval1, Interval interval2) {
    return contains(interval2, interval1);
}

// Funzione che verifica se il secondo intervallo e' antecedente al primo
int isAntecedent(Interval interval1, Interval interval2) {
    return (interval2.end < interval1.start);
}

// Funzione che verifica se il secondo intervallo e' successivo al primo
int isSubsequent(Interval interval1, Interval interval2) {
    return (interval2.start > interval1.end);
}

// Funzione che crea un intervallo a partire da due punti
Interval createInterval(double p1, double p2) {
    Interval interval;
    interval.start = (p1 < p2) ? p1 : p2;
    interval.end = (p1 < p2) ? p2 : p1;
    return interval;
```

```
}

// Funzione che restituisce la lunghezza dell'intervallo
double lengthInterval(Interval interval) {
    return interval.end - interval.start;
}

// Funzione che trasla l'intervallo della quantità specificata
Interval translateInterval(Interval interval, double amount) {
    interval.start += amount;
    interval.end += amount;
    return interval;
}

void stampa(Interval interval){
        printf("Intervallo: [%f, %f]\n", interval.start, interval.end);
}

int main() {
    Interval interval1 = createInterval(1, 5);
    Interval interval2 = createInterval(3, 7);
        stampa(interval1);
        stampa(interval2);
    printf("isIntersect: %d\n", isIntersect(interval1, interval2));    //
    printf("isDisjoint: %d\n", isDisjoint(interval1, interval2));      //
    printf("contains: %d\n", contains(interval1, interval2));          //
    printf("isContained: %d\n", isContained(interval1, interval2));    //
    printf("isAntecedent: %d\n", isAntecedent(interval1, interval2));  //
    printf("isSubsequent: %d\n", isSubsequent(interval1, interval2));  //

    double length_interval1 = lengthInterval(interval1);
    printf("lengthInterval: %f\n", length_interval1);                  //

    Interval translated_interval = translateInterval(interval1, 2);
    stampa(translated_interval);
    return 0;
}
```

4. Intervalli temporali

Gestire punti temporali costituiti di ore, minuti e secondi
prendendo spunto dall’esercizio 3 elaborare una serie di funzioni

5. dato

Scrivi un programma in C++ che:

1. Definisca una `struct` denominata `Dato`, contenente i seguenti campi:
   - `numero` (int): un numero intero.
   - `valore` (float): un numero in virgola mobile.
   - `simbolo` (char): un carattere.

2. Crei un array di `Dato` per gestire più elementi.

3. Implementi le seguenti funzioni:
   - `riempiArray(Dato arr[], int n)`: riempie l'array di `Dato` chiedendo i dati all'utente (o in modo casuale).
   - `trovaMassimo(const Dato arr[], int n)`: restituisce **l'indice** dell'elemento con il valore più alto nel campo `valore`.
   - `stampaArray(const Dato arr[], int n)`: stampa tutti gli elementi dell'array in formato leggibile.
   - `contaCar(const Dato arr[], char c):` restituisce il conteggio delle occorrenze del carattere c

svolto dato

```
#include <iostream>
#include <iomanip>
#define DIM 2 // Dimensione dell'array
using namespace std;

// Definizione della struct
struct Dato {
    int numero;    // Campo intero
    float valore;  // Campo float
    char simbolo;  // Campo char
};

// Prototipi delle funzioni
void riempiArray(Dato arr[], int n);
int trovaMassimo(const Dato arr[], int n);
void stampaArray(const Dato arr[], int n);
int contaCar(const Dato arr[], int n, char c);
```

```
// Funzione principale
int main() {
    Dato dati[DIM];    // Array di Dato

    // Riempimento dell'array
    riempiArray(dati, DIM);

    // Stampa degli elementi dell'array
    cout << "\nElementi dell'array:" << endl;
    stampaArray(dati, DIM);

    // Trova l'elemento con il valore massimo
    int indiceMassimo = trovaMassimo(dati, DIM);
    cout << "\nElemento con il valore massimo:\n";
    cout << "Numero: " << dati[indiceMassimo].numero
         << ", Valore: " << dati[indiceMassimo].valore
         << ", Simbolo: " << dati[indiceMassimo].simbolo << endl;

    // Conta il numero di occorrenze di un carattere
    char c;
    cout << "\nInserisci un carattere per contare le occorrenze: ";
    cin >> c;
    int occorrenze = contaCar(dati, DIM, c);
    cout << "Il carattere '" << c << "' appare " << occorrenze << " volte." << endl;

    return 0;
}

// Funzione per riempire l'array con input utente
void riempiArray(Dato arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << "Inserisci i dati per l'elemento " << i + 1 << ":\n";
        cout << "Numero (int): ";
        cin >> arr[i].numero;
        cout << "Valore (float): ";
        cin >> arr[i].valore;
        cout << "Simbolo (char): ";
        cin >> arr[i].simbolo;
    }
}

// Funzione per trovare l'indice dell'elemento con il valore massimo
int trovaMassimo(const Dato arr[], int n) {
    int indiceMassimo = 0;
```

```
    for (int i = 1; i < n; i++) {
        if (arr[i].valore > arr[indiceMassimo].valore) {
            indiceMassimo = i;
        }
    }
    return indiceMassimo;
}

// Funzione per stampare tutti gli elementi dell'array
void stampaArray(const Dato arr[], int n) {
    cout << left << setw(10) << "Numero"
         << setw(10) << "Valore"
         << setw(10) << "Simbolo" << endl;
    cout << string(30, '-') << endl;

    for (int i = 0; i < n; i++) {
        cout << setw(10) << arr[i].numero
             << setw(10) << fixed << setprecision(2) << arr[i].valore
             << setw(10) << arr[i].simbolo << endl;
    }
}

// Funzione per contare le occorrenze di un carattere nel campo simbolo
int contaCar(const Dato arr[], int n, char c) {
    int contatore = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i].simbolo == c) {
            contatore++;
        }
    }
    return contatore;
}
```

già visti:

6. frazioni
    a. vedi soluzione sullo stream di classroom

7. numeri complessi
    a. vedi soluzione su sullo stream di classroom (sia a coefficienti interi che a coefficienti frazionari)

8. ordinamenti
    a. Scrivi un programma in C++ che:
        i. Definisce una struct con i seguenti campi:
            1. iniziale nome
            2. iniziale cognome
            3. anno di nascita
    b. Generare un array di 10 elementi di tipo struct con valori casuali (caratteri casuali fra A e Z estremi compresi; numeri casuali fra 1990 e 2010 estremi compresi)
    c. Ordinare (e stampare) tale array in tre modi diversi:
        i. per cognome crescente
        ii. per anno di nascita decrescente
        iii. per nome decrescente e, a parità di nome, per cognome decrescente.

9. sol ordinamenti
    a. qsort

10. ticoprof struct 5

Definire una struct livelloLinguistico che memorizzi il livello del certificato linguistico di uno studente; il livello è definito da una lettera e da un numero (ad esempio: A1, B2, C2).
Creare una struct classe definita dall'anno e dalla sezione (5F, 3C, …).
Creare una struct studente con codice numerico studente, classe e livello linguistico.
Memorizzare 5 studenti (leggendo i dati da tastiera o casualmente) e verificare se c'è qualche studente di una classe inferiore che ha un livello linguistico più alto di uno studente di una classe superiore e nel caso stamparne i dati.