

CoderFarm - Corso base

Lezione 2

Carlo Collodel, Francesco Cerroni

7 novembre 2022

Esercizi della scorsa volta

Trova la parità

Testo del problema

Topolino ha un numero N di cioccolatini nel suo zaino e vuole sapere se li può dividere con il suo amico, senza che ci rimanga male. In particolare, Topolino può dividere i cioccolatini col suo amico senza che nessuno rimanga male, solo se il numero N è pari. Aiuta Topolino scrivendo un programma che, preso in ingresso il numero N , stampa in uscita “pari” se il numero è pari, stampa “dispari” se il numero è dispari. Se N fosse uguale a 10, allora il tuo programma dovrebbe stampare “pari”.

Esercizi della scorsa volta

Trova la parità



```
#include <iostream>

using namespace std;

int main()
{
    int N;
    cin >> N;

    if (N % 2 == 0) {
        cout << "pari\n";
    } else {
        cout << "dispari\n";
    }
}
```

Esercizi della scorsa volta

Trova il massimo

Testo del problema

Topolino ha ricevuto in regalo una sequenza di N numeri interi. Puoi aiutarlo a trovare il numero più grande presente nella sequenza scrivendo un programma? Se N fosse uguale a 12 e la sequenza ricevuta da Topolino fosse la seguente:

-331	-341	389	349	-37	-287	441	-871	-913	-853	-617	-150
------	------	-----	-----	-----	------	-----	------	------	------	------	------

allora il tuo programma dovrebbe restituire 441.

Esercizi della scorsa volta

Trova il massimo

```

#include <iostream>

using namespace std;

int main()
{
    int N;
    cin >> N;

    int massimo = -1000; /* -1000 < S[i] < 1000 */
    for (int i = 0; i < N; ++i) {
        int s;
        cin >> s; /* leggo il prossimo valore della lista */
        if (massimo < s) {
            /* se supero il massimo, ho trovato un nuovo massimo */
            massimo = s;
        }
    }

    cout << massimo << endl;
}
```

Esercizi della scorsa volta

Trova la somma massima

Testo del problema

Topolino ha ricevuto in regalo una sequenza di N coppie (a_i, b_i) di numeri interi. Ha deciso che per ogni coppia calcolerà la somma $a_i + b_i$, con l'obiettivo di cercare la somma più grande. Fin qui tutto normale, purtroppo però il dottore ha ordinato a Topolino di stare alla larga dai numeri dispari! Aiuta Topolino scrivendo un programma che, presi in ingresso N e la sequenza di coppie (a_i, b_i) , stampi in uscita la somma massima tra quelle pari. Se N fosse uguale a 5 e la sequenza di coppie ricevuta da Topolino fosse la seguente:

coppie:	(746, 985)	(168, 440)	(425, 940)	(72, 376)	(801, 264)
somme:	1731	608	1365	448	1065

allora il tuo programma dovrebbe stampare 608.

Esercizi della scorsa volta

Trova la somma massima

```
#include <iostream>

using namespace std;

int main()
{
    int N;
    cin >> N;

    int massimo = -1; /* 0 <= a[i], b[i] < 1000 */
    for (int i = 0; i < N; ++i) {
        int a, b;
        cin >> a >> b; /* leggo la prossima coppia */
        int somma = a + b; /* calcolo la somma */

        /* se ho una somma pari */
        if (somma % 2 == 0) {
            if (massimo < somma)
                massimo = somma;
        }
    }

    cout << massimo << endl;
}
```

Array e vettori

Array


E se fosse necessario memorizzare una *lista* di valori? Grazie agli array possiamo allora tanti byte di memoria allineati e memorizzare una lunga sequenza di valori.

attenzione

Gli array in C++ sono *0-based*, cioè gli elementi vengono contati a partire dallo 0. Ad esempio il terzo elemento dell'array è quello in posizione 2.

Array e vettori

Array



```
int main() {  
    // dichiarazione di un array  
    int lista[3] = {3, 7, -2};  
  
    int n = 4;  
    int arr[n];  
    for (int i = 0; i < n; i++) {  
        arr[i] = i;  
    }  
    // arr: {0, 1, 2, 3}  
}
```

Array e vettori

`std::vector`

Nella libreria standard del C++ esistono delle strutture dati chiamate *container*, una tra le più utili sono i vector. I vector sono array che supportano operazioni aggiuntive, tra le più utili troviamo:

- ▶ aggiungere e rimuovere un elemento alla fine del vector in tempo costante (senza dover allocare nuova memoria)
- ▶ ridimensionare il vector
- ▶ *range-based loops*

Array e vettori

std::vector



```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    // vector di interi lungo 3
    vector<int> v(3);
    cout << v.size() << endl; // 3
    // vector di interi lungo 2 e elementi uguali a 5
    vector<int> u(2, 5); // {5, 5}
    u.push_back(3); // {5, 5, 3}
    u.pop_back(); // {5, 5}
}
```

Array e vettori

std::vector



```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    // vector vuoto
    vector<int> v; // {}
    // ridimensionamento
    v.resize(5); // {0, 0, 0, 0, 0}
    v.resize(7, 6); // {0, 0, 0, 0, 0, 6, 6}
    v.assign(3, 4); // {4, 4, 4}

    for (int x: v) { // esegue un'iterazione per ogni elemento di v
        cout << x << "\n"; // x assume il valore dell'elemento corrente
    }
}
```

Array e vettori

`std::vector`

Ci saranno altri *contenitori* con tempi di accesso e complessità diverse. TODO: COMPLESSITÀ ASINTOTICA!

Array e vettori

`std::array`

Un altro *container* molto utile è *array*, che rappresenta una versione leggermente migliorata degli array nativi. In particolare supporta:


- ▶ molte delle funzioni e proprietà condivise dai *container*
- ▶ *range-based loops*

attenzione

A differenza dei *vector*, `std::array` non supporta la possibilità di aggiungere e rimuovere elementi dal fondo in tempo costante.

Array e vettori

std::array



```
#include <iostream>
#include <array>

using namespace std;

int main() {
    // dichiarazione di un array di interi lungo 5
    array<int, 5> lista = {1, 2, 3, 4, 5};
    cout << lista.size() << "\n"; // output: 5
    for (int x: lista) {
        cout << x << " "; // output: 1 2 3 4 5
    }
    lista.fill(-1); // {-1, -1, -1, -1, -1}
}
```

Esercizio con gli array

easy3

<https://training.olinfo.it//task/easy3/statement>

Funzioni

introduzione

In matematica dati due insiemi A e B , si definisce una funzione che mette in relazione ogni elemento di A con un solo elemento B . Un esempio comune sono le funzioni $f : \mathbb{R} \rightarrow \mathbb{R}$, tipo:

$$h(x) = 2x$$

In questo caso la funzione si chiama h , x prende il nome di parametro e per ogni valore di x la funzione associa il doppio. Possiamo anche avere funzioni a più parametri:

$$g(x, y, z) = 3xz + y + x^2$$

Funzioni

introduzione

In generale i parametri non sono necessariamente numeri, ma anche stringhe, booleani, ecc.

esempio 1

$s : \text{stringa } x \mapsto \text{lunghezza di } x$

$s(\text{" esempio"}) = 7$

$s(\text{" aaa"}) = 3$

esempio 2


$c : (\text{stringa } x, \text{intero } n) \mapsto n\text{-esimo carattere di } x$

$c(\text{" parola"}, 3) = 'o'$

Funzioni

implementazione

In C++ una funzione viene implementata nel seguente modo:



```
tipo_di_ritorno nome(tipo_parametro nome_parametro, ...) {  
    // codice  
    return valore_di_ritorno;  
}  
  
int somma(int a, int b) {  
    return a + b;  
}
```

Funzioni

implementazione

Come chiamare una funzione:



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


int somma(int a, int b) {
    return a + b;
}

int main() {
    int n = somma(3, 4);
    std::cout << n << "\n"; // 7
}
```

Funzioni

implementazione

Prototipo di funzione: (se si vuole dichiarare questa dopo il main)



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

int somma(int, int);

int main() {
    int n = somma(3, 4);
    std::cout << n << "\n"; // 7
}

int somma(int a, int b) {
    return a + b;
}
```

Funzioni

implementazione

Alcune cose da notare:

- ▶ appena viene raggiunta un'istruzione *return* l'esecuzione della funzione termina
- ▶ se non si vuole restituire un valore basta usare come tipo *void*
- ▶ nelle funzioni senza valore di ritorno si può comunque utilizzare l'istruzione *return*; per terminare in anticipo la funzione

Funzioni

passaggio per reference

```
● ● ●

#include <iostream>
using namespace std;

// a f viene passato solo il valore di x
void f(int x) {
    x = 2;
}

// a g viene passata una variabile, non il suo valore
void g(int &x) {
    x = 2;
}

int main() {
    int n = 10;

    f(n); // n valore ancora 10
    cout << n << "\n"; // 10

    g(n); // n ha cambiato valore
    cout << n << "\n"; // 2
}
```

Funzioni

passaggio per reference

Alcune cose da notare sul passaggio per reference:

- ▶ Quando si chiama la funzione bisogna necessariamente passare una variabile e non un valore
- ▶ Di solito conviene passare i *container* per reference, perché altrimenti verrebbe completamente clonato rischiando di rallentare molto il programma
- ▶ Gli array nativi, non essendo *container* ma solamente indirizzi di memoria, non è necessario passarli per reference

Funzioni

implementazione

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

void f(vector<int> v) {
    v[0] = 1;
}

void g(vector<int> v) {
    v[0] = 1;
}

void h(int a[]) {
    v[0] = 1;
}

int main() {
    vector<int> v = {3, 4, 5};
    int a[3] = {3, 4, 5};

    f(v); // v = {3, 4, 5}
    g(v); // v = {1, 4, 5}
    g(a); // a = {1, 4, 5}
}
```

Fine

Ci vediamo alla prossima lezione! Vi lasceremo degli esercizi da fare a casa.

- ▶ **E-Mail:** `base@coderfarm.it`
- ▶ **Telegram:** T.B.D.