

RELAZIONE L.S.S. VOLTERRA DI CIAMPINO

Scuola secondaria di secondo grado - AI a supporto della programmazione

Laura Giardoni (2SA), Flavio Pizzi (2SA), Matteo Furcas (2SA), Damiano Zampetti (2SE)

L'Intelligenza Artificiale come Partner Epistemico nella Programmazione: Dal Debug alla Progettazione Modulare

L'evoluzione recente delle tecnologie di Intelligenza Artificiale generativa ha inaugurato una fase di profonda trasformazione nel paradigma dello sviluppo software, ridefinendo le modalità con cui i programmatori affrontano la risoluzione dei problemi e la progettazione algoritmica. La nostra relazione si propone di indagare criticamente tali dinamiche, illustrando attraverso una selezione di casi studio tratti in parte dai problemi delle Olimpiadi di Informatica 2025 come l'AI possa assumere il ruolo di partner cognitivo, capace di intervenire non solo nella correzione di errori tecnici, ma anche nel raffinamento strutturale e stilistico del codice.

Il nucleo dell'analisi risiede nell'esame di soluzioni inizialmente errate o subottimali, caratterizzate da punteggi bassi o nulli nei test ufficiali, e nella loro successiva riformulazione mediante un'interazione iterativa con l'AI. La tesi sostenuta è che l'intelligenza artificiale non debba essere concepita come un semplice generatore automatico di output, bensì come uno strumento di debug assistito e un facilitatore della modularizzazione, in grado di potenziare la capacità progettuale del programmatore umano.

Attraverso nove esempi approfonditi, è stato delineato un percorso metodologico rigoroso articolato in tre direttrici principali:

1. Identificazione dei bug

L'AI si è dimostrata particolarmente efficace nell'individuare criticità spesso insidiose, come il rischio di *integer overflow* in operazioni su larga scala (Esempio 1) o errori sintattici e di punteggiatura che impedivano la compilazione (Esempio 2). La rapidità con cui tali anomalie vengono rilevate evidenzia il valore dell'AI come strumento diagnostico.

2. Ottimizzazione algoritmica

In diversi casi, l'AI ha suggerito strategie computazionali più efficienti, come l'adozione della tecnica *sweep-line* per la gestione di intervalli temporali complessi, riducendo la complessità a $O(N \log N)$ (Esempio 3). Tali interventi non si limitano alla correzione, ma introducono un miglioramento sostanziale nella qualità algoritmica delle soluzioni.

3. Modularità e leggibilità del codice

In ogni esempio è stata richiesta una ristrutturazione del codice secondo il principio DRY (Don't Repeat Yourself). L'AI ha quindi proposto la creazione di funzioni specifiche e riutilizzabili (`calcola_studenti()`),

ipToNumber(), *risolviCaso()*) che hanno reso la logica complessiva più trasparente, manutenibile e aderente alle buone pratiche di ingegneria del software.

Oltre alla dimensione tecnica, la relazione discute criticamente i commenti generati dall'AI, utilizzati per ricostruire le cause profonde dei fallimenti iniziali: violazioni dei vincoli di memoria, errori logici nella gestione dei cicli, scorretta interpretazione dei vincoli del problema. È emerso come la spiegazione del *perché* un codice sia errato costituisca un valore epistemico fondamentale, spesso più formativo della soluzione stessa.

Conclusioni

La nostra relazione intende offrire una prospettiva articolata sull'uso consapevole e metodologicamente fondato degli strumenti di intelligenza artificiale nello sviluppo software. La sinergia tra l'intuizione progettuale del programmatore e la capacità analitica della macchina non produce soltanto codice corretto, ma favorisce la realizzazione di soluzioni eleganti, modulari e performanti. In questa collaborazione, l'AI si configura come un vero e proprio *partner epistemico*, capace di ampliare l'orizzonte cognitivo del processo di programmazione.

Introduzione

L'esperimento si basa nell'usare l'AI per risolvere esercizi diversi tra loro per difficoltà e per tipologie di errori. Infatti questi esercizi presi prevalentemente dal testo delle gare di olimpiadi di informatica non sono stati somministrati già corretti all'AI, bensì sono stati inseriti degli errori nello svolgimento, si controlla che l'IA abbia riconosciuto gli errori corretti. Indifferentemente della difficoltà si chiede anche di commentare il codice e di renderlo più modulare aggiungendo funzioni, per esempio. Negli ultimi esercizi si chiede anche di fare programmi dall'inizio anche con conversioni tra linguaggi diversi e anche aggiungendo interfaccia grafica. Alla fine c'è una tabella riassuntiva dell'esperimento, si trova nella sezione esempio 9 *.

Ogni esercizio ha come struttura: Testo(se presente il testo dell' esercizio), Soluzione Proposta(se presente la soluzione errata), Prompt Sottoposto(cioè che si scrive nella riga di inserimento dell'AI), Risposta Ottenuta(cioè che risponde l'AI senza modifiche), Commenti Alla Soluzione(cosa si può dedurre dalla soluzione).

*A sinistra il documento (.pdf) ha una struttura per la migliore comprensione del testo, cliccando ci si può trasferire da un punto del documento all'altro.

ESEMPIO 1

TESTO:

Giorgio, famoso professore universitario, domani dovrà sorvegliare il suo esame di Programmazione Competitiva. Infatti, sebbene gli studenti siano solitamente molto onesti, c'è sempre qualcuno che cerca di copiare. Per impedirlo, Giorgio sta studiando una disposizione degli studenti in aula che massimizzi la distanza tra ciascuno di loro.

L'aula d'esame è composta da $R \times C$ posti disposti a formare un rettangolo di R file e C colonne. Giorgio vuole che anche i suoi studenti siano disposti in file e colonne, ma saltando alcune file e alcune colonne per distanziarli. Dato un fattore di distanza K , ogni studente deve trovarsi ad almeno K posti di distanza ($K - 1$ posti intermedi) da quelli della sua fila e colonna. Aiutalo a trovare il numero massimo di studenti che possono essere contenuti nell'aula.

SOLUZIONE PROPOSTA:

C/C++

```
#include <assert.h>

#include <stdio.h>

long long R, C, K;

int main()
{
    freopen("input0.txt", "r", stdin);
    freopen("output0.txt", "w", stdout);

    assert(3 == scanf("%lld %lld %lld", &R, &C, &K));

    int r1 = C % K;

    int q1;
```

```
if(r1 == 0)
    q1 = C / K;
else
    q1 = (C / K) + 1;

int r2 = R % K;
int q2;

if(r2 == 0)
    q2 = R / K;
else
    q2 = (R / K) + 1;

int S = q1 * q2;

printf("%lld\n", S);

return 0;
}
```

Questo codice ottiene 0/100 punti, vediamo cosa non va:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento.

Il tuo ragionamento logico di base è assolutamente corretto: per massimizzare il numero di studenti, è necessario disporli calcolando quanti ne entrano in una singola riga e in una singola colonna, per poi moltiplicare questi due valori.

Tuttavia, il codice presenta un **bug critico legato alla gestione della memoria (Integer Overflow)** che ti farà fallire i test case più grandi.

I Problemi del tuo Codice

1. **Integer Overflow:** Il testo del problema ti avvisa esplicitamente: "The answer may be too large for a 32-bit integer: use 64-bit integers such as long long". Nel tuo codice hai dichiarato R , C e K come `long long`, ma poi hai salvato i risultati $q1$, $q2$ e il totale S come semplici `int`. Dato che i vincoli arrivano fino a 10^9 , se $K=1$, S potrebbe arrivare a 10^{18} . Un `int` a 32 bit può contenere al massimo circa $2 \cdot 10^9$, quindi il valore di S andrà in overflow e produrrà un numero errato o negativo.
2. **Comportamento indefinito nel printf:** Stai cercando di stampare S (che è un `int`) usando il formato `%lld` (che si aspetta un `long long`). Questo in C causa un comportamento anomalo.
3. **Lettura file (Opzionale):** Hai forzato la lettura da `input0.txt`. Nelle gare di programmazione competitive moderne (come le OIS), di solito si legge direttamente dallo Standard Input (`stdin`), a meno che non sia richiesto esplicitamente di aprire un file.

Codice Riscritto, Modulare e Ottimizzato

Possiamo semplificare enormemente il calcolo dell'arrotondamento per eccesso (ceiling) utilizzando una formula matematica standard in C/C++: $(A + B - 1) / B$. Questo ci permette di eliminare tutti gli `if/else` e rendere il codice molto più pulito.

C/C++

```
#include <stdio.h>

#include <assert.h>

// Funzione modulare per calcolare quanti studenti entrano in una determinata
// lunghezza

// Utilizza il tipo long long per prevenire qualsiasi overflow

long long calcola_studenti(long long dimensione, long long distanza) {

// La formula (dimensione + distanza - 1) / distanza simula l'arrotondamento per
// eccesso
```

```

// (equivalente alla funzione ceil() matematica, ma lavorando solo con interi)
    return (dimensione + distanza - 1) / distanza;
}

int main() {
    // R = righe, C = colonne, K = distanza minima

    long long R, C, K;

    // Lettura da Standard Input (commenta freopen se sottometti
    su piattaforme standard)
    // freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);

    assert(3 == scanf("%lld %lld %lld", &R, &C, &K));

    // Calcoliamo gli studenti per righe e colonne separatamente
    long long studenti_per_riga = calcola_studenti(C, K);
    long long studenti_per_colonna = calcola_studenti(R, K);

    // Il numero massimo di studenti è il prodotto (ora al sicuro
    dall'overflow)
    long long totale_studenti = studenti_per_riga * studenti_per_colonna;

    // Stampa del risultato
    printf("%lld\n", totale_studenti);

    return 0;
}

```

Spiegazione del funzionamento

- **L'approccio geometrico:** Il problema ci chiede di distanziare gli studenti di un fattore K in una griglia $R \times C$. Immagina di piazzare il primo studente all'inizio di una riga. Il prossimo studente potrà sedersi solo dopo aver saltato $K-1$ posti vuoti, quindi occuperà l'iesimo posto $+ K$. Questo equivale esattamente a dividere la lunghezza della riga per K arrotondando verso l'alto.
- **Il trucco dell'arrotondamento intero:** In C/C++, la divisione tra numeri interi tronca sempre verso il basso (es. $5 / 2 = 2$). Per arrotondare verso l'alto senza usare i numeri decimali (la funzione `ceil()` richiede conversioni in `double` che possono far perdere precisione con numeri giganteschi), si usa la formula $(A + B - 1) / B$.
 - Esempio con $C=6$, $K=2$: $(6 + 2 - 1) / 2 \rightarrow 7 / 2 \rightarrow 3$. Esattamente 3 studenti.
 - Esempio con $C=3$, $K=2$: $(3 + 2 - 1) / 2 \rightarrow 4 / 2 \rightarrow 2$. Esattamente 2 studenti.
- **Modularità:** Aver spostato questa formula nella funzione `calcola_studenti()` riduce le ripetizioni del codice (il principio DRY: Don't Repeat Yourself), rende la logica chiara al primo sguardo ed elimina del tutto la necessità delle variabili di supporto `r1`, `r2`, `q1`, e `q2`.

Il codice scritto dall'AI ottiene 100/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

L'AI ha controllato il giusto comportamento del codice, trovando i seguenti errori:

- Integer overflow
- Comportamento indefinito nella `printf`
- Lettura file (opzionale)

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `long long calcola_studenti()`

ESEMPIO 2

TESTO:

Marco sta cercando di hackerare la rete della sua università. Per fare ciò, deve inviare un pacchetto dannoso a ciascun computer che potrebbe essere connesso alla stessa rete. Per essere sicuro, Marco ha deciso di inviare il pacchetto a un intervallo di indirizzi IP. Un indirizzo IP è un insieme di quattro "ottetti" separati da punti (ogni "ottetto" è un numero intero nell'intervallo da 0 a 255 incluso). Ad esempio, 127.0.0.1, 192.168.2.3 e 255.255.255.255 sono indirizzi IP validi, mentre qualcosa come 10.20.300.4000 non lo è. Calcola, dato un intervallo valido di indirizzi, quanti pacchetti Marco deve inviare.

SOLUZIONE PROPOSTA:

C/C++

```
#include <iostream>
#include <stdio.h>
#include <assert.h>
using namespace std

int main(

{

long long  A[4], B[4]
unsigned long long ris=0;

assert(4 = scanf("%lld.%lld.%lld.%lld", &A[0], &A[1], &A[2], &A[3]));
assert(4 == scanf("%lld.%lld.%lld.%lld" &B[0], &B[1], &B[2], &B[3]));

if (A[0]!=B[0] || A[1]!=B[1] || A[2]!=B[2] || A[3]!=B[3])
```

```
{  
  
B[0] = B[0]-A[0]  
B[1] = B[1]-A[1];  
B[2] = B[2]-A[2];  
B[3] = B[3]-A[3];  
  
ris += (B[0]*16777216)  
  
ris += (B[1]*65536);  
  
ris += (B[2]*256);  
  
ris = ris+ B[3];  
}  
  
cout << ris+1  
  
return 0  
}
```

Questo codice ottiene 0/100 punti, vediamo cosa non va:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento.

Il tuo ragionamento logico per risolvere il problema è assolutamente corretto. L'approccio di convertire gli indirizzi IP in numeri interi, sottrarli e aggiungere 1 per calcolare i pacchetti totali (poiché gli estremi sono inclusivi) è il modo più efficiente per affrontare l'esercizio.

Tuttavia, il codice che hai scritto presenta diversi **errori di sintassi** che gli impediscono di essere compilato ed eseguito correttamente.

Gli errori nel tuo codice originale

1. **Punti e virgola mancanti:** Ne mancano svariati (dopo `using namespace std`, alla fine delle dichiarazioni delle variabili, dopo alcune assegnazioni e alla fine di `return 0`).
2. **Parentesi del main:** Hai scritto `int main({` invece di `int main() {`.
3. **Assegnazione invece di uguaglianza:** Nel primo `assert`, hai scritto `4 = scanf(...)`. L'operatore corretto per il confronto è `==`.
4. **Virgola mancante nello scanf:** Nel secondo `scanf`, manca la virgola per separare la stringa di formattazione dagli argomenti ("`%lld.%lld.%lld.%lld`" &B[0]).
5. **Controllo if non necessario:** Il blocco `if (A[0]!=B[0] ...)` non è strettamente necessario. Se gli IP sono identici, le differenze daranno zero, e il risultato finale sarà giustamente $0 + 1 = 1\$$.

Il tuo codice corretto

Ecco la tua versione originale con gli errori di sintassi corretti, mantenendo esattamente la tua logica:

```
C/C++  
  
#include <iostream>  
  
#include <stdio.h>  
  
#include <assert.h>  
  
using namespace std; // Aggiunto punto e virgola  
  
int main() // Aggiunte parentesi tonde  
{  
  
    long long A[4], B[4]; // Aggiunto punto e virgola  
  
    unsigned long long ris = 0;  
  
    // Corretto operatore == (era =)  
  
    assert(4 == scanf("%lld.%lld.%lld.%lld", &A[0], &A[1], &A[2], &A[3]));  
  
    // Aggiunta la virgola mancante prima di &B[0]  
  
    assert(4 == scanf("%lld.%lld.%lld.%lld", &B[0], &B[1], &B[2], &B[3]));
```

```

if (A[0]!=B[0] || A[1]!=B[1] || A[2]!=B[2] || A[3]!=B[3])
{
    B[0] = B[0] - A[0]; // Aggiunto punto e virgola
    B[1] = B[1] - A[1];
    B[2] = B[2] - A[2];
    B[3] = B[3] - A[3];

    ris += (B[0] * 16777216); // Aggiunto punto e virgola
    ris += (B[1] * 65536);
    ris += (B[2] * 256);
    ris = ris + B[3];
}

cout << ris + 1 << endl; // Aggiunto punto e virgola e ritorno a capo

return 0; // Aggiunto punto e virgola
}

```

Versione C++ Modulare e Ottimizzata

Per rendere il codice più leggibile, scalabile e in puro stile C++, possiamo rimuovere le dipendenze dal C (come `stdio.h` e `scanf`), estrarre il calcolo in una funzione separata ed eliminare i calcoli manuali per ogni ottetto usando un approccio matematico diretto.

C/C++

```
#include <iostream>
```

```
using namespace std;
```

```
/**
```

* Funzione modulare per convertire un indirizzo IP in un numero intero a 64-bit.

* Tratta l'IP come un numero in base 256.

*/

```
long long ipToNumber(long long o1, long long o2, long long o3, long long o4) {  
    // LL assicura che la moltiplicazione avvenga a 64-bit per evitare overflow  
    return (o1 * 16777216LL) + (o2 * 65536LL) + (o3 * 256LL) + o4;  
}
```

```
int main() {  
    long long start1, start2, start3, start4;  
    long long end1, end2, end3, end4;  
    char dot; // Variabile di appoggio per ignorare i punti '.' durante l'input  
  
    // Legge il primo IP (es. 192.168.1.0)  
    cin >> start1 >> dot >> start2 >> dot >> start3 >> dot >> start4;  
  
    // Legge il secondo IP (es. 192.168.1.255)  
    cin >> end1 >> dot >> end2 >> dot >> end3 >> dot >> end4;  
  
    // Converte entrambi gli IP in numeri interi  
    long long startNumeric = ipToNumber(start1, start2, start3, start4);  
    long long endNumeric = ipToNumber(end1, end2, end3, end4);  
  
    // Calcola i pacchetti: la differenza tra il range di IP, più 1 per includere  
    // l'estremo finale  
    long long packets = endNumeric - startNumeric + 1;  
  
    cout << packets << "\n";  
  
    return 0;  
}
```

}

Come funziona (Spiegazione Matematica)

Un indirizzo IP IPv4 è composto da quattro "ottetti" separati da punti. Ogni ottetto rappresenta un numero compreso tra 0 e 255. Questo significa che la struttura di un IP equivale di fatto a un sistema numerico in **base 256**.

Per sapere quanti indirizzi ci sono tra un punto A e un punto B, la mossa migliore è "appiattare" l'indirizzo IP in un unico grande numero decimale. Per un IP `A.B.C.D`, la conversione in decimale puro è:

$$\text{\$Valore} = (A \times 256^3) + (B \times 256^2) + (C \times 256) + D\text{\$}$$

Una volta trasformati i due IP in numeri interi (assicurandosi di usare un tipo di dato abbastanza grande da contenere il numero, come `long long` in C++), la risposta si trova con una semplice sottrazione (`\$IndirizzoFinale - IndirizzoIniziale + 1\$\`). Il "+1" è necessario perché il primo pacchetto va inviato anche al computer con l'IP di partenza.

Ti andrebbe di esplorare come leggere l'input in modo più robusto nel caso in cui il formato degli indirizzi IP fornito non fosse sempre "perfetto" o in caso ci fossero spazi di troppo?

Il codice scritto dall'AI ottiene 100/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

L'AI ha controllato il giusto comportamento del codice, trovando i seguenti errori:

- ' ; ' mancanti
- parentesi tonde del main ()
- assegnazione al posto dell'uguaglianza
- ' , ' mancante nello scanf()
- controllo if() non necessario

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `long long ipToNumber()`

ESEMPIO 3

TESTO:

Compiere gli anni d'estate ha i suoi vantaggi, ma spesso trovare una data in cui tutti sono liberi per festeggiare il compleanno risulta difficile, se non addirittura impossibile. Andrea, che compie gli anni in pieno agosto, è stanco di festeggiarlo sempre da solo; quest'anno ha deciso di portarsi avanti, chiedendo ai

suoi N amici quando saranno in vacanza, così da trovare il giorno perfetto per organizzare la sua festa.

Andrea ogni anno durante il giorno della sua festa.

Ogni amico i ($0 \leq i < N$) è in vacanza dal giorno A_i al giorno B_i . Gli estremi sono da considerarsi inclusi, quindi l'amico è disponibile fino al giorno $A_i - 1$ e dal giorno $B_i + 1$. Andrea vuole organizzare la sua festa tra il giorno X e il giorno Y (estremi inclusi).

Aiutalo a trovare il numero massimo di amici che possono venire alla sua festa!

SOLUZIONE PROPOSTA:

C/C++

```
#include <vector>

#include <map>

#include <algorithm>

using namespace std;

int helperFunzione(int x);

int organizza(int N, int X, int Y, vector<int> A, vector<int> B) {

    map<int, int> variazioni;
```

```
int c = 0;

for (int i = 0; i <= N; i++)
{
    if (A[i] < X && B[i] > X)
        c--;

    if (A[i] >= X && A[i] < Y)
        variazioni[A[i]]--;

    if (B[i] > X && B[i] <= Y)
        variazioni[B[i]+2]++;
}

int m = helperFunzione(N);

for (auto const& p : variazioni)
{
    c -= p.second;
    m = min(m, c);
}

return m;
}
```

Questo codice ottiene 0/100 punti, vediamo cosa non va:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento.

RISPOSTA OTTENUTA DALL'AI:

Analizzando il testo del problema e il codice che hai fornito, ci sono diverse criticità logiche e di implementazione che impedirebbero al programma di funzionare correttamente o di rispettare i limiti di tempo.

Analisi delle Criticità

- Logica degli Intervalli:** Il problema chiede il **numero massimo di amici liberi**. Un amico è libero se il giorno della festa *non* cade nell'intervallo $[A_i, B_i]$. Il tuo codice sembra tentare di contare chi è in vacanza, ma la gestione dei limiti X e Y è confusa.
- Efficienza:** L'uso di una `std::map` ha una complessità $O(N \log N)$. Sebbene accettabile per $N=100.000$, l'approccio migliore per questo tipo di problemi di "copertura" è spesso l'utilizzo di un **Difference Array** (o tecnica degli eventi) su vettori, data l'ampiezza dei giorni fino a 10^9 (che richiede però la compressione delle coordinate o solo l'analisi dei punti di cambio).
- Errori nel Loop:** Il ciclo `for (int i = 0; i <= N; i++)` causa un *out-of-bounds* perché i vettori A e B sono di dimensione N (indici 0 a $N-1$).
- Helper Inesistente:** La funzione `helperFunzione(N)` non è definita nel frammento.

Soluzione Ottimizzata e Modulare

L'approccio più pulito consiste nel:

- Contare quanti amici sono **inizialmente liberi** al giorno X .
- Registrare solo i "cambiamenti": quando un amico **inizia** una vacanza (diventa non disponibile) e quando la **finisce** (torna disponibile).
- Considerare solo gli eventi che accadono tra X e Y .

Ecco la versione riscritta, modulare e commentata:

```
C/C++
#include <vector>

#include <algorithm>

#include <map>

using namespace std;

/**
 * Struttura per gestire gli eventi di disponibilità.
 * Un amico i è NON disponibile nel range [A[i], B[i]].
 */
```

```

*/
int organizza(int N, int X, int Y, vector<int> A, vector<int> B) {
    // Mappa degli eventi: giorno -> variazione amici liberi
    // Usiamo una map per gestire giorni molto grandi (fino a 10^9)
    // registrando solo i punti in cui la disponibilità cambia.
    map<int, int> variazioni;
    int amici_liberi_correnti = 0;

    for (int i = 0; i < N; ++i) {
        // Verifichiamo lo stato iniziale al giorno X
        if (X < A[i] || X > B[i]) {
            // L'amico è libero se il giorno X è fuori dalle sue vacanze
            amici_liberi_correnti++;
        }

        // Registriamo i cambiamenti che avvengono DOPO il giorno X e fino a Y

        // 1. L'amico inizia le vacanze: da libero diventa occupato
        if (A[i] > X && A[i] <= Y) {
            variazioni[A[i]]--;
        }

        // 2. L'amico finisce le vacanze: dal giorno B[i]+1 torna disponibile
        if (B[i] >= X && B[i] < Y) {
            variazioni[B[i] + 1]++;
        }
    }

    int massimo_amici = amici_liberi_correnti;
}

```

```
// Iteriamo attraverso i giorni in cui avviene almeno un cambiamento
for (auto const& [giorno, variazione] : variazioni) {
    amici_liberi_correnti += variazione;
    massimo_amici = max(massimo_amici, amici_liberi_correnti);
}

return massimo_amici;
}
```

Spiegazione del Funzionamento

1. **Stato Iniziale:** Invece di calcolare ogni singolo giorno (impossibile con un miliardo di giorni), partiamo dal giorno X . Controlliamo quanti amici sono già liberi in quel momento.
2. **Gestione Eventi:**
 - Se un amico inizia le vacanze al giorno A_i (e A_i è entro il range della festa), il numero di amici liberi **diminuisce** (--).
 - Se un amico termina le vacanze al giorno B_i , dal giorno $B_i + 1$ torna a essere libero, quindi il numero **aumenta** (++) .
3. **Sweep-line (Scansione):** La mappa tiene i giorni ordinati. Scorrendola, aggiorniamo il conteggio degli amici disponibili e salviamo il valore massimo mai raggiunto.
4. **Efficienza Spaziale e Temporale:**
 - **Tempo:** $O(N \log N)$ per l'inserimento nella mappa.
 - **Spazio:** $O(N)$ per memorizzare gli eventi. Questo permette di superare anche il Subtask 6 senza problemi di memoria o tempo.

Il codice scritto dall'AI ottiene 100/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

L'AI ha controllato il giusto comportamento del codice, trovando i seguenti errori:

- Logica degli Intervalli
- Efficienza
- Errori nel Loop

- Helper Inesistente

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `int organizza()`

ESEMPIO 4

TESTO:

Laura, Daria e Noemi sono grandi appassionate di carte da collezione. Finalmente sono riuscite a risparmiare abbastanza per comprare un po' di bustine e adesso devono decidere come distribuirle. Ognuna di loro ha infatti dei gusti particolari in tema di carte: Laura ama le carte laminate, Daria va invece pazza per i mostri d'aria e Noemi ha un debole per le carte con lo sfondo nero. Le ragazze hanno comprato M carte, ogni carta può essere sia laminata, che contenere un mostro d'aria, che avere lo sfondo nero, oppure una qualsiasi combinazione di queste caratteristiche. Per non scontentare nessuno ciascuna ragazza prenderà quelle carte che possiedono solamente la qualità da loro prediletta. Vale a dire che Laura prenderà le carte laminate ma non d'aria né nere, Daria quelle d'aria ma non laminate né nere e Noemi quelle nere ma non laminate né d'aria. Le ragazze vogliono capire quante carte avanzano dopo averle distribuite, in modo da poterle rivendere a Edoardo. Aiutale a calcolare il numero di carte avanzate

SOLUZIONE PROPOSTA:

C/C++

```
int scarta(int M, vector<bool> L, vector<bool> D, vector<bool> N)
{
    long long ris = 0;

    ris = ris + x;    // 'x' non esiste

    M = "ciao";

    if (L[0] == "true")
        ris++;

    ris += L[1] + "errore";
```

```

    int k = somma(L, D, N);

ifstream f("config.txt");

    int tmp;

    f >> tmp;

ifstream g("/usr/local/data/input.txt");

    for (int i = 0; i <= M; i++)    // <= invece di <

        if (L[i] && D[i])

            ris++;

    return ris;

}

```

Questo codice ottiene 0/100 punti, vediamo cosa non va:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento

RISPOSTA OTTENUTA DALL'AI:

L'esercizio che hai proposto contiene diversi errori concettuali e di sintassi che ne impediscono la compilazione e il corretto funzionamento secondo le specifiche del problema "Bustine di carte".

Analisi degli errori nel codice originale

1. **Variabili non definite e tipi errati:** Tenti di usare una variabile `x` non dichiarata e assegna una stringa "ciao" a un intero `M`.
2. **Errori di tipo (Type Mismatch):** `L[0]` è un `bool`, quindi non può essere confrontato con la stringa "true". Allo stesso modo, non puoi sommare una stringa a un valore numerico (`L[1] + "errore"`).

3. **Violazione della consegna:** Il problema chiede di implementare solo la funzione `scarta`. Leggere file esterni come `config.txt` o `/usr/local/data/input.txt` non è richiesto e causerebbe errori nel sistema di correzione (grader).
4. **Logica errata:** Il codice originale conta le carte laminata e d'aria, mentre il problema chiede di contare le carte che **avanzano** dopo che Laura, Daria e Noemi hanno preso le loro.
5. **Indici fuori limite:** Il ciclo `for (int i = 0; i <= M; i++)` causerebbe un errore perché l'ultimo indice valido di un vettore di dimensione M è $M-1$.

Soluzione Corretta e Spiegazione

Secondo le regole:

- **Laura** prende la carta se è **solo** laminata ($L=true, D=false, N=false$).
- **Daria** prende la carta se è **solo** d'aria ($L=false, D=true, N=false$).
- **Noemi** prende la carta se è **solo** nera ($L=false, D=false, N=true$).
- Tutte le altre carte (quelle con più caratteristiche o nessuna) vengono **scartate** e date a Edoardo.

Ecco il codice riscritto in modo modulare e leggibile:

C/C++

```
#include <vector>

using namespace std;

/**
 * Calcola il numero di carte che avanzano per Edoardo.
 * Una carta avanza se NON viene scelta da nessuna delle tre ragazze.
 */
int scarta(int M, vector<bool> L, vector<bool> D, vector<bool> N) {
    int carte_avanzate = 0;

    for (int i = 0; i < M; i++) {
        // Controlliamo se la i-esima carta viene presa da una delle ragazze

        // Laura la prende se: L è true, D è false, N è false
```

```

bool presa_da_laura = (L[i] == true && D[i] == false && N[i] == false);

// Daria la prende se: L è false, D è true, N è false

bool presa_da_daria = (L[i] == false && D[i] == true && N[i] == false);

// Noemi la prende se: L è false, D è false, N è true

bool presa_da_noemi = (L[i] == false && D[i] == false && N[i] == true);

// Se nessuna delle tre la prende, la carta avanza per Edoardo

if (!presa_da_laura && !presa_da_daria && !presa_da_noemi) {
    carte_avanzate++;
}

return carte_avanzate;
}

```

Spiegazione della logica

- **Il Ciclo:** Iteriamo esattamente M volte, da 0 a $M-1$, per analizzare ogni singola carta comprata.
- **Le Condizioni:** Per ogni ragazza, verifichiamo che la caratteristica preferita sia presente (`true`) e che **contemporaneamente** le altre due siano assenti (`false`).
- **Il Conteggio:** Usiamo l'operatore di negazione `!` per dire: "se la carta NON è stata presa da Laura E NON è stata presa da Daria E NON è stata presa da Noemi", allora incrementiamo il contatore delle carte avanzate.

Il codice scritto dall'AI ottiene 100/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

L'AI ha controllato il giusto comportamento del codice, trovando i seguenti errori:

- Variabili non definite e tipi errati
- Errori di tipo (Type Mismatch)
- Violazione della consegna
- Logica errata
- Indici fuori **limite**

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `int scarta()`

ESEMPIO 5

TESTO:

Il nuovo e popolare gioco di parole Wordle si sta diffondendo così rapidamente che la sua contagiosità ha raggiunto anche Luca. Per chi non ne avesse ancora sentito parlare, in Wordle bisogna indovinare una parola di 5 lettere in un numero limitato di tentativi. Dopo ogni tentativo si scopre, per ciascuna posizione, se la lettera inserita è corretta e nella posizione giusta, se è presente nella parola ma in una posizione sbagliata, oppure se è completamente errata. Incuriosito dalle caratteristiche di questo gioco, Luca ha deciso di creare una sua versione più generica. In questa versione, bisogna indovinare una parola di N lettere. Ogni possibile sequenza di lettere dell'alfabeto inglese costituisce un tentativo valido, ma è garantito che la parola da indovinare non contiene lettere ripetute. Ti trovi nel mezzo di una partita: hai già indovinato alcune lettere, ma devi ancora scoprirne altre, che nell'input sono indicate con un trattino basso “_”. Ti chiedi: quante parole potrebbero essere una soluzione valida del gioco, date le lettere che conosci?

SOLUZIONE PROPOSTA:

C/C++

```
#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

int N;

long long sum, ris, let;

vector<char> L;

int main()
{
    cin >> N;

    char c;

    L.resize(N);

    for (int i = 0; i < N; i++)
        cin >> L[i];

    for (int i = 0; i < N; i++)
        if (L[i] != '_')
            sum++;

    c = L.begin();
```

```

let = 26 - sum;

ris = 1;

for (int i = 0; i < (N - sum); i++)
    ris = ris * (let - i);

char z = L[N];

    int* p = nullptr;
*p = 10;    // crash quasi certo

ifstream f("config.txt");

int tmp;

f >> tmp;

cout << ris << endl;

return 0;

}

```

Questo codice ottiene 0/100 punti, vediamo cosa non va:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento.

RISPOSTA OTTENUTA DALL'AI:

Il codice che hai fornito presenta diversi errori critici (sia logici che di programmazione pura) che ne impediscono il corretto funzionamento. Ecco un'analisi dei problemi principali e una versione riscritta, modulare e spiegata.

Analisi degli errori nel codice originale

1. **Crash intenzionale:** Hai inserito un puntatore nullo (`int* p = nullptr; *p = 10;`) che causa un **Segmentation Fault** immediato.
2. **Errore di tipo:** `c = L.begin();` tenta di assegnare un iteratore a un `char`, il che non è valido.
3. **Accesso fuori dai limiti:** `char z = L[N];` prova a leggere una posizione che non esiste (gli indici vanno da 0 a $N-1$).
4. **Logica di input:** La lettura del vettore `L` e il calcolo del numero di underscore non è gestita in modo robusto rispetto agli spazi o al formato dell'input Wordle descritto.

Versione corretta e modulare

Questa versione risolve il problema utilizzando il calcolo combinatorio (disposizioni semplici). Dato che le lettere **non possono ripetersi**, ogni underscore può essere riempito scegliendo tra le lettere dell'alfabeto (26) meno quelle già presenti nella parola.

```
C/C++
#include <iostream>

#include <vector>

#include <string>

using namespace std;

/**
 * Calcola il numero di possibili soluzioni per il Wordle generico.
 * La logica si basa sulle disposizioni semplici:
 * n = lettere disponibili (26 - lettere già note)
 * k = posti da riempire (numero di underscore)
 */
long long calcolaPossibilita(int N, const vector<char>& parola) {
    int lettereNote = 0;
    int underscore = 0;

    // 1. Contiamo quante lettere sono già fissate e quanti sono i buchi
    for (char c : parola) {
        if (c != '_') {
```

```

        lettereNote++;
    } else {
        underscore++;
    }
}

// 2. Lettere rimaste nell'alfabeto (26 totali)
// Il problema garantisce che non ci siano duplicati tra le note.
int lettereDisponibili = 26 - lettereNote;

// 3. Calcolo delle disposizioni:  $D(n, k) = n * (n-1) * \dots * (n-k+1)$ 
long long risultato = 1;
for (int i = 0; i < underscore; i++) {
    risultato *= (lettereDisponibili - i);
}

return risultato;
}

int main() {
    // Ottimizzazione I/O
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int N;

    if (!(cin >> N)) return 0;

    vector<char> parola(N);

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

```

```

        cin >> parola[i];
    }

    // Stampa il risultato finale
    cout << calcolaPossibilita(N, parola) << endl;

    return 0;
}

```

Spiegazione del funzionamento

Il problema ci chiede di trovare quante sequenze di lettere possono completare una parola di N caratteri, sapendo che:

1. **Nessuna lettera può ripetersi** all'interno della parola.
2. Abbiamo 26 lettere a disposizione nell'alfabeto inglese.

Il calcolo matematico

Se abbiamo una parola di lunghezza N dove K lettere sono già state inserite (es. HOUS_), abbiamo:

- **Lettere già usate:** K (nel caso di HOUS_, sono 4: H, O, U, S).
- **Lettere rimanenti tra cui scegliere:** $26 - K$ (ovvero 22 lettere disponibili).
- **Posti da riempire:** $N - K$ (nel caso di HOUS_, 1 posto solo).

Il numero di modi per riempire i posti vuoti è dato dal prodotto decrescente delle lettere disponibili. Ad esempio, se avessimo $N=3$ e nessun carattere noto (___):

- Per il primo posto abbiamo **26** scelte.
- Per il secondo abbiamo **25** scelte (tutte meno quella usata prima).
- Per il terzo abbiamo **24** scelte.
- Totale: $26 \times 25 \times 24 = 15.600$.

Se invece abbiamo A__ (come nel secondo esempio fornito):

- Lettere note: 1 ('A'). Lettere disponibili: $26 - 1 = 25$.
- Primo underscore: **25** scelte.
- Secondo underscore: **24** scelte.
- Totale: $25 \times 24 = 600$.

Il codice scritto dall'AI ottiene 100/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

L'AI ha controllato il giusto comportamento del codice, trovando i seguenti errori:

- Segmentation Fault ,causa crash intenzionale
- Errore di tipo
- Accesso fuori dai limiti
- Logica di input

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `long long` `calcolaPossibilita()`

ESEMPIO 6

TESTO:

Luca ha superato la paura del buio, e deve spegnere tutte le N lampadine, numerate da 1 a N , nel lungo corridoio della sua casa prima di andare a dormire! Nel corridoio sono presenti $N-1$ interruttori numerati da 1 a $N-1$. Il loro funzionamento è un po' particolare! Le lampadine i e $i+1$ sono collegate all' i -esimo interruttore. Premendo un interruttore entrambe le lampadine ad esso collegate *cambieranno di stato*: una lampadina *cambiando di stato* si accende se spenta e si spegne se accesa. Questo rende un po' complicato spegnere tutto, quindi Luca ha deciso di non andare troppo per il sottile, e se serve svitare direttamente alcune lampadine. Luca può quindi fare le seguenti azioni:

- *svita lampadina i* : svitare l' i -esima lampadina, spegnendola. Una volta svitata, la lampadina non potrà più essere accesa né riavvitata.
- *premi l'interruttore i* : premere l'interruttore i -esimo, cambiando lo stato delle lampadine i e $i+1$ (se e solo se non sono svitate).

Luca è molto stanco e vorrebbe finire al più presto il suo lavoro: trova il minimo numero di azioni necessarie per portare il buio nel corridoio! Conta come una azione sia svitare una lampadina che premere un interruttore.

SOLUZIONE PROPOSTA:

```
C/C++
#include <fstream>

#include <iostream>

#include <string>

#include <vector>

using namespace std;

int main() {

    // decommenta le due righe seguenti se vuoi leggere/scrivere da file
    //ifstream cin("input.txt");
```

```
//ofstream cout("output.txt");

int t;

cin >> t;

int T[t];

for (int test = 1; test <= t; ++test) {

    int N;

    cin >> N;

    vector<long long> A(N);

    for (int i = 0; i < N; ++i)

        cin >> A[i];

    int ris = 0;

    for (int i = 0; i < N; i++)

        {

            if (A[i]==1)

                {

                    ris++;

                    i++;

                }

            else

                if (A[i+1]==1)

                    {

                        ris++;

                    }

        }

}
```

```

        }

        else

            i++;

    }

    T[test]=ris;

}

for (int test = 1; test <= t; ++test)
{

    cout << "Case #" << test << ": ";

    cout << T[test] << endl;

}

return 0;

}

```

Questo codice ottiene 15/50 punti, vediamo cosa non va:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento.

RISPOSTA OTTENUTA DALL'AI:

Ciao! Ho analizzato il tuo codice. L'idea di base per risolvere il problema c'è, ma ci sono alcuni **errori logici e tecnici** che rischiano di far fallire il programma (crash, errori di segmentazione o risultati sballati).

Ecco i problemi principali che ho individuato:

1. **Out of Bounds sull'array T:** Hai dichiarato `int T[t];`, che crea un array di dimensione `t` con indici da `0` a `t-1`. Tuttavia, il tuo ciclo `for` va da `1` a `t` incluso, e assegna il risultato a `T[test]`. Quando `test == t`, stai scrivendo fuori dalla memoria dell'array.

2. **Array a dimensione variabile (VLA):** Dichiarare `int T[t]`; dove `t` è una variabile letta a runtime non è C++ standard. Anche se alcuni compilatori (come GCC) lo permettono, è considerata una cattiva pratica. Meglio usare un `std::vector` o stampare direttamente il risultato.
3. **Out of Bounds sul vector A:** All'interno del ciclo `for` usi `A[i+1]`. Quando `i` arriva all'ultimo elemento (`N-1`), `A[i+1]` cerca di leggere un valore che non esiste (fuori dal vector), causando un comportamento indefinito.
4. **Errore logico nel conteggio:** Nel ramo `else if (A[i+1]==1)`, incrementi `ris` ma non fai `i++`. Questo significa che al ciclo successivo, `i` avanzerà normalmente di 1, troverà di nuovo quell' `1`, e incrementerà `ris` una seconda volta. Stai contando due volte!

Il Codice Riscritto (Modulare e Pulito)

Ho riscritto il tuo codice risolvendo tutti questi problemi. L'ho reso **modulare** (spostando la logica di un singolo caso di test in una funzione separata), in modo che sia molto più facile da leggere e mantenere.

C/C++

```
#include <iostream>

#include <vector>

using namespace std;

// Funzione che gestisce la logica di un singolo caso di test (testcase)
// Ritorna il numero di mosse necessarie

int risolviCaso() {

    int N;

    cin >> N;

    vector<int> A(N);

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

        cin >> A[i];

    }

    int ris = 0;

    // Scorriamo le lampadine da sinistra a destra

    for (int i = 0; i < N; i++) {
```

```

    if (A[i] == 1) {
        // Se troviamo una lampadina accesa (1), dobbiamo spegnerla.
        ris++;

        // Dato che un'azione spegne questa lampadina e la successiva,
        // possiamo saltare la prossima lampadina facendo i++.
        i++;
    }

    // Se A[i] == 0, non facciamo nulla. Il ciclo for farà i++
    // portandoci automaticamente alla lampadina successiva.
}

return ris;
}

int main() {
    // Ottimizzazione per velocizzare l'input/output in C++
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // decommenta le due righe seguenti se vuoi leggere/scrivere da file
    // freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);

    int t;

    if (cin >> t) {
        for (int test = 1; test <= t; ++test) {
            // Calcoliamo e stampiamo direttamente il risultato
            // senza bisogno di salvarlo in un array intermedio

```

```

        int risultato = risolviCaso();

        cout << "Case #" << test << ": " << risultato << "\n";

    }

}

return 0;

}

```

Come funziona questa versione:

1. **Approccio "Greedy" (Ingordo):** Assumendo che le regole del problema dicano che una mossa spegne due lampadine adiacenti, la strategia migliore è scorrere da sinistra verso destra. Appena incontriamo un 1, *dobbiamo* fare una mossa per forza (nessun'altra mossa futura potrà spegnere questa lampadina senza riaccendere quelle precedenti). Incrementiamo le mosse (`ris++`) e saltiamo la lampadina successiva (`i++`) perché verrà spenta (o cambiata) assieme a questa.
2. **Nessun else complicato:** Se `A[i]` è 0, semplicemente lo ignoriamo. Non ci serve guardare avanti (`A[i+1]`) rischiando di uscire dall'array. Ci penserà il ciclo al giro successivo a controllare se la prossima lampadina è 1 o 0.
3. **Stampa Immediata (Memoria Ottimizzata):** Invece di creare un array `T` per salvare tutti i risultati e stamparli alla fine, calcolo il risultato di un testcase e lo stampo *subito*. Questo fa risparmiare memoria, evita problemi di array out-of-bounds ed è lo standard nella programmazione competitiva.
4. **`ios_base::sync_with_stdio`:** Ho aggiunto una piccola ottimizzazione all'inizio del `main`. Serve a rendere `cin` e `cout` molto più veloci, utilissimo se hai file di input di grandi dimensioni.

Questa logica si basa sull'ipotesi che la regola del problema sia "un interruttore inverte lo stato di due lampadine consecutive". Questo approccio rispetta esattamente le regole della piattaforma in cui ti stai allenando.

Il codice scritto dall'AI ottiene 50/50.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

L'AI ha controllato il giusto comportamento del codice, trovando i seguenti errori:

- Out of Bounds sull'array `T[]` ed `A[]`
- Array a dimensione variabile (VLA)
- Errore logico nel conteggio

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `int risolviCaso()`

ESEMPIO 7

TESTO:

Davide è stanco di lamerare i problemi e ha deciso che da ora in avanti scriverà solo soluzioni oneste. La terapia però non è così facile da affrontare e spesso Davide inizia a comportarsi in modo irrazionale, infatti durante una crisi isterica causata dall'astinenza dalla funzione clock() inizia a disegnare compulsivamente cerchi. Una volta finita la crisi isterica egli nota che ha disegnato esattamente N Cerchi, ognuno con il centro nelle coordinate intere (X_i, Y_i) . Inoltre tutti i cerchi passano per l'origine degli assi, ovvero il punto $(0,0)$. A questo punto Davide si pone la seguente domanda: quali sono i cerchi non completamente inclusi in altri? Purtroppo Davide senza la possibilità di lamerare il problema non riesce a trovare la risposta, aiutalo tu!

SOLUZIONE PROPOSTA:

Non presente

vediamo come lo svolge l'AI:

PROMPT SOTTOPOSTO ALL'AI:

Risolvi l'esercizio in modo leggibile e modulare, commenta e spiega il funzionamento, adotta la soluzione più rapida ed efficace.

RISPOSTA OTTENUTA DALL'AI:

Per risolvere il problema "Chi cerchia trova", dobbiamo identificare quali cerchi **non sono completamente inclusi** in altri. Poiché tutti i cerchi passano per l'origine $(0,0)$, la condizione di inclusione si semplifica notevolmente grazie alle proprietà geometriche della **trasformazione per inversione**.

Analisi del Problema

Un cerchio C_i con centro (X_i, Y_i) che passa per l'origine ha raggio $R_i = \sqrt{X_i^2 + Y_i^2}$. L'equazione del cerchio è:

$$(x - X_i)^2 + (y - Y_i)^2 = X_i^2 + Y_i^2 \implies x^2 + y^2 - 2X_ix - 2Y_iy = 0$$

Un cerchio C_i è contenuto in C_j se e solo se la distanza tra i centri più il raggio del cerchio interno è minore o uguale al raggio del cerchio esterno:

$$\text{dist}(O_i, O_j) + R_i \leq R_j$$

Dato che entrambi i cerchi passano per l'origine, questa condizione è equivalente a dire che il centro O_i deve trovarsi sul segmento che collega l'origine al centro O_j , e $R_i \leq R_j$. In termini di coordinate, ciò significa che:

1. I centri (X_i, Y_i) e (X_j, Y_j) devono essere **collineari con l'origine** e trovarsi nello **stesso quadrante** (stessa direzione rispetto all'origine).
2. La distanza del centro O_i dall'origine deve essere minore o uguale a quella di O_j ($R_i \leq R_j$).

Strategia Risolutiva

Per trovare i cerchi non inclusi, dobbiamo scartare ogni cerchio i il cui raggio sia "coperto" da un cerchio j nella stessa direzione.

1. **Normalizzazione**: Rappresentiamo la direzione di ogni centro (X_i, Y_i) riducendo le coordinate tramite il massimo comune divisore: $dx = X_i / \text{gcd}(|X_i|, |Y_i|)$ e $dy = Y_i / \text{gcd}(|X_i|, |Y_i|)$.
2. **Raggruppamento**: Usiamo una mappa (o una tabella hash) dove la chiave è la direzione normalizzata (dx, dy) e il valore è la distanza massima (raggio) trovata in quella direzione.
3. **Selezione**: Solo i cerchi che hanno la distanza massima per la loro specifica direzione sono "non inclusi".

Implementazione C++

C/C++

```
#include <vector>

#include <numeric>

#include <map>

#include <cmath>

using namespace std;

/**
 * Funzione per calcolare il Massimo Comune Divisore
 */

long long gcd(long long a, long long b) {
    return b == 0 ? a : gcd(b, a % b);
}

/**
 * Risolve il problema identificando i cerchi non inclusi.
 * Un cerchio i è incluso in j se i centri sono allineati con l'origine
```

```

* e  $R_i < R_j$ .
*/
vector<int> trovacerchi(int N, vector<int> X, vector<int> Y) {
    // Mappa per memorizzare la distanza massima per ogni direzione (dx, dy)
    // key: {dx, dy}, value: {distanza_massima_quadrata, indice_cerchio}
    map<pair<long long, long long>, pair<long long, int>> max_circles;

    for (int i = 0; i < N; ++i) {
        long long xi = X[i];
        long long yi = Y[i];

        // Calcoliamo la direzione normalizzata (vettore ridotto)
        long long common = gcd(abs(xi), abs(yi));
        long long dx = xi / common;
        long long dy = yi / common;

        // Usiamo la distanza al quadrato per evitare approssimazioni con sqrt
        long long distSq = xi * xi + yi * yi;

        pair<long long, long long> direction = {dx, dy};

        // Se non abbiamo ancora questa direzione o se il cerchio corrente è più
        grande
        if (max_circles.find(direction) == max_circles.end() || distSq >
max_circles[direction].first) {
            max_circles[direction] = {distSq, i};
        }
    }

    // Estraiamo gli indici dei cerchi superstiti

```

```
vector<int> result;

for (auto const& [dir, data] : max_circles) {
    result.push_back(data.second);
}

return result;
}
```

Spiegazione Tecnica

- **Complessità Temporale:** $O(N \log N)$ dovuta all'inserimento nella `std::map` e al calcolo del GCD per N elementi.
- **Complessità Spaziale:** $O(N)$ per memorizzare le direzioni uniche.
- **Precisione:** L'uso di `long long` e della distanza al quadrato previene errori di precisione tipici dei numeri in virgola mobile (`double`), garantendo correttezza anche con coordinate fino a 10^9 .
- **Perché funziona:** Poiché tutti i cerchi passano per l'origine, un cerchio può essere contenuto in un altro solo se "si espande" nella stessa direzione ma arriva meno lontano. Identificando il cerchio più lontano per ogni raggio d'azione, eliminiamo tutti quelli contenuti.

Il codice scritto dall'AI ottiene 0/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI non ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

Nell'esecuzione si riscontrano i seguenti errori:

- TLE (time limit exceeded)
- output errato

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo funzioni denominate:

- `long long gcd()`
- `vector<int> trovacerchi()`

SECONDO PROMPT SOTTOPOSTO ALL'AI:

L'esercizio presenta errori di tipo TLE ed errori di output, risolvi e spiega.

SECONDA RISPOSTA OTTENUTA DALL'AI:

Il problema principale nella logica precedente è un'interpretazione geometrica incompleta: non basta considerare solo i cerchi sulla stessa retta passante per l'origine.

L'Errore Geometrico (Perché avevi Output Errato)

Un cerchio C_i è contenuto in C_j se e solo se la distanza tra i loro centri $d(O_i, O_j)$ sommata al raggio R_i è minore o uguale al raggio R_j . Poiché entrambi passano per l'origine, questa condizione si semplifica: il cerchio C_i è contenuto nel cerchio C_j se e solo se il punto O_i (centro del cerchio C_i) si trova **all'interno** del cerchio C_j .

Matematicamente, il cerchio C_i è incluso in C_j se:

$$(X_i - X_j)^2 + (Y_i - Y_j)^2 \leq X_j^2 + Y_j^2$$

Espandendo e semplificando:

$$X_i^2 - 2X_iX_j + X_j^2 + Y_i^2 - 2Y_iY_j + Y_j^2 \leq X_j^2 + Y_j^2$$

$$X_i^2 + Y_i^2 \leq 2X_iX_j + 2Y_iY_j$$

La Soluzione Ottimale (Perché avevi TLE)

Con $N = 100,000$, un doppio ciclo $O(N^2)$ causa **Time Limit Exceeded (TLE)**. Dobbiamo usare la **Convex Hull (Inviluppo Convesso)**.

Se trasformiamo ogni cerchio in un punto tramite un'inversione circolare o osserviamo la condizione di inclusione sopra citata, notiamo che i cerchi "non inclusi" corrispondono ai punti (X_i, Y_i) che formano l'inviluppo convesso dell'insieme dei centri, includendo l'origine $(0,0)$ nel calcolo.

Tuttavia, c'è una scorciatoia più semplice: i cerchi non inclusi sono quelli i cui centri non sono "coperti" da altri. Dato che tutti passano per l'origine, i cerchi massimali sono quelli i cui centri sono i vertici del **Convex Hull** dei punti $\{(X_i, Y_i)\} \cup \{(0,0)\}$.

Implementazione Corretta (C++)

```
C/C++
```

```
#include <vector>

#include <algorithm>

using namespace std;

struct Point {
```

```

    long long x, y;

    int id;

};

// Prodotto vettoriale per determinare la rotazione (cross product)
long long cross_product(Point a, Point b, Point c) {
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}

vector<int> trovacerchi(int N, vector<int> X, vector<int> Y) {
    vector<Point> pts;

    for (int i = 0; i < N; i++) {
        pts.push_back({(long long)X[i], (long long)Y[i], i});
    }

    // Aggiungiamo l'origine come punto di riferimento
    pts.push_back({0, 0, -1});

    sort(pts.begin(), pts.end(), [](Point a, Point b) {
        if (a.x != b.x) return a.x < b.x;

        return a.y < b.y;
    });

    int n = pts.size();
    vector<Point> hull;

    // Costruzione Monotone Chain (Involucro Convesso)
    for (int i = 0; i < n; ++i) {
        while (hull.size() >= 2 && cross_product(hull[hull.size() - 2],
            hull.back(), pts[i]) <= 0) {

```

```

        hull.pop_back();
    }

    hull.push_back(pts[i]);
}

int lower_size = hull.size();

for (int i = n - 2; i >= 0; --i) {
    while (hull.size() > lower_size && cross_product(hull[hull.size() - 2],
hull.back(), pts[i]) <= 0) {
        hull.pop_back();
    }

    hull.push_back(pts[i]);
}

hull.pop_back(); // Rimuovi duplicato finale

vector<int> result;

for (auto p : hull) {
    if (p.id != -1) { // Non includere l'origine fittizia nel risultato
        result.push_back(p.id);
    }
}

return result;
}

```

Perché questa soluzione è efficace:

1. **Velocità ($O(N \log N)$):** L'ordinamento dei punti domina il tempo di esecuzione, permettendo di gestire 10^5 punti in meno di un secondo.
2. **Correttezza Geometrica:** I cerchi non inclusi sono quelli i cui centri "vedono" l'origine senza essere oscurati, il che equivale ai punti sul bordo esterno (Convex Hull).
3. **Robustezza:** L'uso di `long long` evita overflow nei calcoli del prodotto vettoriale con coordinate fino a 10^9 .

Il secondo codice scritto dall'AI ottiene 100/100.

COMMENTI ALLA SECONDA SOLUZIONE:

Si nota che la seconda risposta dell' AI ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

Nell'esecuzione non si riscontrano errori:

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo funzioni denominate:

- `long long cross_product()`
- `vector<int> trovacerchi()`

ESEMPIO 8

TESTO:

Alessandro sta programmando la sua nuova versione 3D di Pac-Man. Il tabellone di gioco è descritto da una griglia tridimensionale, alcune celle sono bloccate e le altre sono libere. Pac-Man e i fantasmi possono spostarsi in qualsiasi cella libera che condivida una faccia con la cella corrente. Alessandro ha istruito i fantasmi a muoversi in un modo molto semplice. Se un fantasma vuole spostarsi dalla cella A alla cella B, ripeterà la seguente procedura fino a raggiungere la destinazione o a fallire:

- Se il fantasma può diminuire la distanza lungo l'asse x (cioè $|A_x - B_x|$), lo farà spostandosi di una cella lungo l'asse x;
- altrimenti, se può diminuire la distanza lungo l'asse y (cioè $|A_y - B_y|$), lo farà spostandosi di una cella lungo l'asse y;
- altrimenti, se può diminuire la distanza lungo l'asse z (cioè $|A_z - B_z|$), lo farà spostandosi di una cella lungo l'asse z;
- altrimenti fallisce.

Alessandro ha tre array X_i , Y_i e Z_i indicizzati da 0 a $N - 1$ che descrivono le coordinate delle celle libere. Si chiede se la strategia sopra è abbastanza intelligente da controllare i fantasmi. Aiutalo scrivendo un programma che determini se per ogni coppia di celle A e B un fantasma riuscirà a raggiungere la cella B partendo dalla cella A!

SOLUZIONE PROPOSTA:

Non presente

vediamo come lo svolge l'AI:

PROMPT SOTTOPOSTO ALL'AI:

Risolvi l'esercizio in modo leggibile e modulare, commenta e spiega il funzionamento, adotta la soluzione più rapida ed efficace.

RISPOSTA OTTENUTA DALL'AI:

L'esercizio richiede di determinare se un fantasma, seguendo un movimento **greedy** (ovvero privilegiando la riduzione della distanza prima sull'asse x , poi y , infine z), possa raggiungere qualsiasi cella di destinazione B partendo da qualunque cella A in una griglia 3D di celle libere.

Analisi del Problema

Il fantasma segue questa gerarchia di movimento:

1. Prova a muoversi su X per avvicinarsi a B_x .
2. Se non può (perché $A_x = B_x$ o la cella è bloccata), prova su Y .
3. Se non può su Y , prova su Z .
4. Se è bloccato in tutte le direzioni che ridurrebbero la distanza, **fallisce**.

La Condizione di Successo

Affinché i fantasmi abbiano sempre successo, l'insieme delle celle libere deve essere **"convesso" rispetto al movimento greedy**. In termini semplici, per ogni coppia di punti, non devono esserci "ostacoli" che costringano il fantasma ad allontanarsi da un asse per poi ritornarvi, poiché l'algoritmo non prevede il backtracking o l'aggiramento.

Dato che il fantasma riduce sempre la distanza, il percorso sarà lungo esattamente:

$$D = |A_x - B_x| + |A_y - B_y| + |A_z - B_z|$$

Il problema si risolve verificando se la struttura delle celle libere permette questo movimento. Un'osservazione chiave per questo tipo di problemi di raggiungibilità "monotona" è che il set di celle deve formare un **iper-rettangolo** (o un'unione specifica di essi) dove, se esistono due punti, deve esistere anche il punto che ha le coordinate "miste".

Tuttavia, con $N = 100.000$, non possiamo simulare tutti i percorsi (N^2). La soluzione efficace consiste nel verificare che l'insieme dei punti sia esattamente il prodotto cartesiano dei valori unici di X , Y e Z . Se il numero di punti totali N è uguale al prodotto del numero di coordinate distinte per ogni asse, allora ogni combinazione esiste e il fantasma non rimarrà mai bloccato.

Soluzione Proposta (Python)

Questa soluzione utilizza i set per contare i valori unici e verifica la completezza della griglia.

Python

```
import sys

def solve():

    # Lettura dell'input

    try:

        line1 = sys.stdin.readline()

        if not line1:

            return

        n = int(line1.strip())

        # Caricamento delle coordinate X, Y, Z

        x_coords = list(map(int, sys.stdin.readline().split()))

        y_coords = list(map(int, sys.stdin.readline().split()))

        z_coords = list(map(int, sys.stdin.readline().split()))
```

```

except ValueError:

    return

# 1. Troviamo i valori unici per ogni asse

unique_x = set(x_coords)

unique_y = set(y_coords)

unique_z = set(z_coords)

# 2. Calcoliamo il numero di celle necessarie per formare un
# parallelepipedo completo senza "buchi" o forme irregolari
# che causerebbero il fallimento del movimento greedy.

required_cells = len(unique_x) * len(unique_y) * len(unique_z)

# 3. Se il numero di celle fornite (N) è uguale al prodotto delle
# dimensioni degli assi unici, allora ogni percorso greedy è garantito.
# In caso contrario, esiste almeno una "lacuna" che bloccherà il fantasma.

if n == required_cells:

    print("YES")

else:

    print("NO")

if __name__ == "__main__":

    solve()

```

Spiegazione del Funzionamento

1. **Analisi delle Coordinate:** Il programma estrae tutti i valori distinti di XX , YY e ZZ presenti nell'input.
2. **Logica di "Raggiungibilità Totale":** Il movimento descritto è estremamente rigido. Se manca anche una sola cella "intermedia" in una qualsiasi proiezione rettangolare, il fantasma fallirà perché non sa tornare indietro o cambiare asse se la gerarchia ($X \rightarrow Y \rightarrow Z$) glielo impedisce.
3. **Efficienza:**

- **Tempo:** $O(N)$ per leggere i dati e inserirli nei set.
- **Spazio:** $O(N)$ per memorizzare le coordinate.
- Questa soluzione è ottimale per il Subtask 6 (nessuna limitazione) poiché gestisce $N=100.000$ in frazioni di secondo.

Esempio pratico (dal testo)

Nel **secondo esempio**:

- Celle: $(1,0,0)$ e $(1,1,1)$.
- Se cerchiamo di andare da $A(1,0,0)$ a $B(1,1,1)$, il fantasma vede che $A_x = B_x$, quindi passa a Y. Se la cella $(1,1,0)$ è bloccata (non presente tra le N celle), il fantasma fallisce immediatamente. La formula $n == \text{len}(\text{unique}_x) * \text{len}(\text{unique}_y) * \text{len}(\text{unique}_z)$ intercetta esattamente questa mancanza di "pienezza" della griglia.

Il codice scritto dall'AI ottiene 0/100.

COMMENTI ALLA SOLUZIONE:

Si nota che l' AI non ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

Nell'esecuzione si riscontrano i seguenti errori:

- output errato

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo funzioni denominate:

- `def solve()`

SECONDO PROMPT SOTTOPOSTO ALL'AI:

L'esercizio presenta errori di output, risolvi e spiega.

SECONDA RISPOSTA OTTENUTA DALL'AI:

Hai ragione a notare che l'approccio basato sul semplice "prodotto delle coordinate" potrebbe essere troppo restrittivo o non catturare appieno la natura del movimento **greedy** descritto nel problema.

Il movimento del fantasma è **monotono**: può solo diminuire la distanza lungo x , poi y , poi z . Se una cella intermedia necessaria manca, il fantasma fallisce.

Ecco la risoluzione corretta e l'analisi dell'errore logico comune.

Il Problema Logico

Il fantasma fallisce se, nel tentativo di andare da A a B , incontra un "buco" in una posizione che la sua gerarchia di movimento lo costringe a visitare.

La condizione affinché il fantasma raggiunga **sempre** la destinazione per **ogni** coppia di celle è che l'insieme delle celle sia **chiuso rispetto alla proiezione degli assi**. In termini semplici: se esistono le celle (x_1, y_1, z_1) e (x_2, y_2, z_2) , allora per garantire il movimento greedy devono esistere anche tutte le combinazioni "intermedie" che il percorso prioritario toccherebbe.

Per N fino a 100.000 , la soluzione più efficace consiste nel verificare che l'insieme dei punti sia un **iper-rettangolo pieno**.

Codice Risolutivo (C++)

Dato che il problema proviene dalle OIS (Olimpiadi di Informatica a Squadre), il C++ è il linguaggio più indicato per gestire l'efficienza richiesta dal Subtask 6.

C/C++

```
#include <iostream>

#include <vector>

#include <set>

#include <algorithm>

using namespace std;

void solve() {

    int N;

    if (!(cin >> N)) return;

    vector<int> X(N), Y(N), Z(N);

    set<int> distinct_x, distinct_y, distinct_z;
```

```

for (int i = 0; i < N; i++) {
    cin >> X[i];
    distinct_x.insert(X[i]);
}
for (int i = 0; i < N; i++) {
    cin >> Y[i];
    distinct_y.insert(Y[i]);
}
for (int i = 0; i < N; i++) {
    cin >> Z[i];
    distinct_z.insert(Z[i]);
}

// La condizione necessaria e sufficiente affinché il fantasma non rimanga
// mai bloccato con questo algoritmo greedy è che la griglia sia un
// prodotto cartesiano completo dei valori unici presenti.

long long required_cells = (long long)distinct_x.size() * (long
long)distinct_y.size() * (long long)distinct_z.size();

// Se il numero di celle totali N è esattamente uguale al prodotto
// delle dimensioni dei tre assi, significa che non ci sono "buchi".
if ((long long)N == required_cells) {
    cout << "YES" << endl;
} else {
    cout << "NO" << endl;
}
}

```

```
int main() {  
  
    ios_base::sync_with_stdio(false);  
  
    cin.tie(NULL);  
  
    solve();  
  
    return 0;  
  
}
```

Perché questa è la soluzione corretta?

1. Analisi del Movimento Greedy

Il fantasma non ha intelligenza di percorso (pathfinding). Segue un ordine fisso:

1. **Priorità X:** Muove su x finché $A_x \neq B_x$.
2. **Priorità Y:** Muove su y solo se non può più muovere su x .
3. **Priorità Z:** Muove su z solo se non può muovere né su x né su y .

Se in uno di questi passaggi la cella successiva è "bloccata" (ovvero non è tra le N celle libere), il fantasma si ferma e fallisce.

2. La condizione di "Griglia Piena"

Perché il fantasma abbia successo tra ogni coppia A e B , non deve esistere alcuna combinazione di coordinate (x_i, y_j, z_k) mancante.

- Se esistesse una coordinata x in una cella, y in un'altra e z in una terza, ma non esistesse la cella che le combina tutte e tre, un fantasma che parte da una e deve arrivare a una destinazione che richiede quella combinazione rimarrebbe bloccato.
- Il numero totale di celle N deve quindi essere esattamente uguale al prodotto del numero di valori unici trovati in X, Y, Z .

3. Spiegazione dell'Esempio 2 (NO)

Nell'esempio 2, abbiamo celle come $(1,0,0)$ e $(1,1,1)$.

- Valori unici di X : $\{0, 1\}$
- Valori unici di Y : $\{0, 1\}$
- Valori unici di Z : $\{0, 1\}$
- Prodotto: $2 \times 2 \times 2 = 8$.
- Tuttavia, N è minore di 8. Questo significa che mancano celle fondamentali (come la $(1,1,0)$ necessaria per passare da x a z), quindi l'output è **NO**.

Il secondo codice scritto dall'AI ottiene 0/100.

COMMENTI ALLA SECONDA SOLUZIONE:

Si nota che la seconda risposta dell' AI non ha risolto correttamente l'esercizio.

RICERCA BUG ED ERRORI:

Nell'esecuzione si riscontrano i seguenti errori:

- output errato

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo funzioni denominate:

- `void solve()`

ESEMPIO 9

TESTO:

Creare un derivato di calcolatrice utilizzando linguaggio C.

SOLUZIONE PROPOSTA:

C/C++

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void rall(int millisecondi)
{
    clock_t start_time = clock();

    while (clock() < start_time + millisecondi * CLOCKS_PER_SEC /
1000);
}

void fstampastr(char str[])
{
    int const sec=0045;

    int s=0;

    while (str[s] != '\0')
    {
        printf("%c", str[s]);
    }
}
```

```
    rall(sec);  
  
    s++;  
}  
}
```

```
int main ()
```

```
{
```

```
char str40[] = "(Questo programma e' esclusivamente a scopo  
educativo.");
```

```
char str41[] = " Replica in parte le funzioni di alcune classiche  
calcolatrici digitali, sia nel funzionamento che nelle funzioni che  
offre: ";
```

```
char str42[] = " cioe' si puo' operare direttamente sul risultato di  
un calcolo precedente o azzerare il valore.);";
```

```
int const byte = 16, sec=0045;
```

```
int i, j, z, flag, flag1, flag2, esc, bin[byte], ris1, num1, num2,  
num3, num4, s;
```

```
float ris, num, n, percent;
```

```
i=0, flag=0, j=1, z=1, flag1=0, flag2=0, esc=0, s=0, num=0;
```

```
char str[] = "Buongiorno, premere 1 per iniziare.";
```

```
char str1[] = "Valore: ";
```

```
char str2[] = "Valore non consentito reinserire: ";
```

```
char str3[] = "Scegliere funzione: ";
char str4[] = "Funzioni di base: ";
char str5[] = "premere 0 per scegliere un numero, premere 1 per
addizione, premere 2 per sottrazione, premere 3 per moltiplicazione,
premere 4 per divisione.";
char str6[] = "Funzioni avanzate: ";
char str7[] = "premere 6 per trasformare valore in binario, premere 7
per calcolare percentuale, premere 8 per scegliere un numero
casuale.";
char str8[] = "Altre funzioni: ";
char str9[] = "premere 5 per azzerare valore, premere 9 per terminare
programma. ";
char str10[] = "Inserire addendo: ";
char str11[] = "Valore addendo: ";
char str12[] = "Inserire minuendo: ";
char str13[] = "Valore minuendo: ";
char str14[] = "Inserire sottraendo: ";
char str15[] = "Iserire moltiplicando: ";
char str16[] = "Valore moltiplicando: ";
char str17[] = "Inserire moltiplicatore: ";
char str18[] = "Inserire dividendo: ";
char str19[] = "Valore dividendo: ";
char str20[] = "Inserire divisore: ";
char str21[] = "somma = ";
char str22[] = "resto = ";
char str23[] = "prodotto = ";
char str24[] = "Scegliere numero a partire da: ";
char str25[] = "Fino a: ";
```

```
char str26[] = "Valore non consentito, reinserire valore maggiore: ";
char str27[] = "Numero scelto: ";
char str28[] = "Funzione non disponibile in questo momento, prova
altra funzione.";
char str29[] = "Valore ";
char str30[] = " in binario: ";
char str31[] = "Percentuale di ";
char str32[] = " da calcolare: ";
char str33[] = " % di ";
char str34[] = " = ";
char str35[] = " %";
char str36[] = "Premere 1 per continuare, premere 9 per terminare
programma.";
char str37[] = "Funzione: ";
char str38[] = "Inserire valore: ";
char str39[] = "Valore gia' presente, usare prima funzione 5 per
azzerare valore. ";
while(i==0)
{
    if (flag==0)
    {
        printf("\n");

        fstampastr (str40);

        printf("\n");

        fstampastr(str41);
```

```
printf("\n");
```

```
fstampastr(str42);
```

```
printf("\n\n");
```

```
fstampastr(str);
```

```
fstampastr(str1);
```

```
scanf ("%d", &i);
```

```
printf("\n");
```

```
flag=1;
```

```
}
```

```
while (i!=1)
```

```
{
```

```
fstampastr(str2);
```

```
scanf ("%d", &i);
```

```
printf("\n");
```

```
}
```

```
}
```

```
i=0, flag=0;
```

```
system("cls");
```

```
//calcolatore matematico
```

```
while(j==1)
```

```
{
```

```
    if (j==9)
```

```
        break;
```

```
    /* 1 calcolatrice di base :
```

```
        addizione, sottrazione, moltiplicazione, divisione.*/
```

```
    if(flag2==0)
```

```
        {
```

```
            fstampastr(str3);
```

```
            printf("\n\n");
```

```
            fstampastr(str4);
```

```
            printf("\n");
```

```
            fstampastr(str5);
```

```
            printf("\n\n");
```

```
fstampastr(str6);
```

```
printf("\n");
```

```
fstampastr(str7);
```

```
printf("\n\n");
```

```
fstampastr(str8);
```

```
printf("\n");
```

```
fstampastr(str9);
```

```
printf("\n\n");
```

```
}
```

```
else
```

```
{
```

```
printf("Scegliere funzione: \n\n");
```

```
printf("Funzioni di base: \n");
```

```
printf("premere 0 per scegliere un numero, premere 1  
per addizione, premere 2 per sottrazione, premere 3 per  
moltiplicazione, premere 4 per divisione.\n\n");
```

```
printf("Funzioni avanzate: \n");
```

```
printf("premere 6 per trasformare valore in binario,  
premere 7 per calcolare percentuale, premere 8 per scegliere un numero  
casuale.\n\n");
```

```
        printf("Altre funzioni: \n");

        printf("premere 5 per azzerare valore, premere 9 per
terminare programma.\n\n");
    }

fstampastr(str37);

scanf ("%d", &z);
printf("\n");

while (z<0 || z>9)
    {

        fstampastr(str2);

        scanf ("%d", &z);
        printf("\n");
    }

if (z==9)
    {
        system ("cls");
        break;
    }

if (z==0)
    if (flag1==0)
```

```
{  
    fstampastr(str38);  
  
    scanf ("%f", &num);  
    ris = num;  
    flag1=1;  
    printf("\n");  
}
```

else

```
{  
    fstampastr(str39);  
    printf("\n\n");  
}
```

if(z==1)

```
{  
    if (flag1==0)  
    {  
        fstampastr(str10);  
  
        scanf ("%f", &num);  
        ris = num;  
        flag1=1;  
    }  
}
```

```
else
    {
        fstampastr(str11);

        printf("%f \n",ris);
    }

fstampastr(str10);

scanf ("%f", &num);

ris = ris + num;

fstampastr(str21);

printf("%f ", ris);
printf("\n\n");
}

if(z==2)
    {
        if (flag1==0)
            {
                fstampastr(str12);

                scanf ("%f", &num);

                ris = num;

                flag1=1;
```

```
    }  
else  
    {  
        fstampastr(str13);  
  
        printf("%f \n",ris);  
    }  
  
    fstampastr(str14);  
  
    scanf ("%f", &num);  
    ris = ris - num;  
  
    fstampastr(str22);  
  
    printf("%f ", ris);  
    printf("\n\n");  
    }  
  
if(z==3)  
    {  
        if (flag1==0)  
            {  
                fstampastr(str15);  
  
                scanf ("%f", &num);
```

```
        ris = num;

        flag1=1;

    }

else

    {

        fstampastr(str16);

        printf("%f \n",ris);

    }

fstampastr(str17);

scanf ("%f", &num);

ris = ris * num;

fstampastr(str23);

printf("%f ", ris);

printf("\n\n");

}

if(z==4)

    {

        if (flag1==0)

            {

                fstampastr(str18);
```

```
scanf ("%f", &num);  
ris = num;  
flag1=1;  
}  
else  
{  
fstampastr(str19);  
  
printf("%f \n",ris);  
}  
  
fstampastr(str20);  
  
scanf ("%f", &num);  
ris = ris / num;  
  
fstampastr(str22);  
  
printf("%f ", ris);  
printf("\n\n");  
}  
  
if(z==8)  
{  
srand(time(NULL));
```

```
fstampastr(str24);
```

```
scanf ("%d", &num3);
```

```
fstampastr(str25);
```

```
scanf ("%d", &num4);
```

```
printf("\n");
```

```
while (num4<=num3)
```

```
{
```

```
    fstampastr(str26);
```

```
    scanf ("%d", &num4);
```

```
    printf("\n");
```

```
}
```

```
num4=num4-num3;
```

```
num2 = (rand()% num4)+num3;
```

```
if (ris==0 || num==0)
```

```
{
```

```
    ris = num2;
```

```
    flag1 = 1;
```

```
}
```

```
fstampastr(str27);
```

```
        printf("%d \n\n", num2);

        flag1 = 1;
    }

if (flag1==0)
    {
        fstampastr(str28);

        printf("\n\n");
    }
else
    {
        if(z==6)
            {
                num1= (int)ris;
                ris1 = (int)ris;
                for (i=byte-1; i>=0; i--)
                    {
                        bin[i] = ris1 %2;
                        ris1 = ris1 / 2;
                    }

                fstampastr(str29);

                printf("%d", num1);
```

```
fstampastr(str30);

for (i=0; i<byte; i++)
    {
        if(i % 4==0)
            printf(" ");

            printf("%d ", bin[i]);

        }
    ris=num1;

    printf("\n\n");
}

if(z==7)
    {
        fstampastr(str31);

        printf("%f", ris);

        fstampastr(str32);

        scanf ("%f", &n);

        percent = ris/100;
```

```
percent = percent*n;
printf ("%f", n);

fstampastr(str33);

printf("%f", ris);

fstampastr(str34);

printf("%f", percent);

fstampastr(str35);
printf("\n\n");
}
}
```

```
if(z==5)
{
flag1=0;
num=0;
}
```

```
s=0;
```

```
if (flag2==0)
{
fstampastr(str36);
```

```
        printf ("\n");
        flag2=1;
    }
else
    printf("Premere 1 per continuare, premere 9 per
terminare programma. \n");

fstampastr(str1);
scanf ("%d", &j);
printf("\n");

while (!(j==1 || j==9))
    {
        fstampastr(str2);

        scanf ("%d", &j);
        printf("\n");
    }
system ("cls");
}

return 0;
}
```

Vediamo come si può migliorare:

PROMPT SOTTOPOSTO ALL'AI:

Controlla il corretto funzionamento dell'esercizio successivamente se possibile riscrivilo in modo più leggibile e modulare, commenta e spiega il funzionamento, infine trasformalo in c++ e aggiungi un interfaccia grafica

RISPOSTA OTTENUTA DALL'AI:

Il tuo codice originale è un ottimo esercizio di logica per un principiante: implementa una calcolatrice "stateful" (che mantiene memoria del risultato precedente) e aggiunge un tocco estetico con l'effetto "macchina da scrivere".

Tuttavia, presenta alcune criticità tecniche:

1. **Costanti Ottali:** Hai scritto `0045`. In C, i numeri che iniziano con `0` sono interpretati come **ottali**. `0045` ottale è `37` decimale. Funziona, ma è un errore concettuale se intendevi 45 millisecondi.
2. **Busy Waiting:** La funzione `rall` consuma il 100% della CPU mentre aspetta.
3. **Mancanza di Modularità:** Il `main` è un unico blocco gigante, difficile da gestire e correggere.
4. **Sicurezza:** `scanf` con i numeri può causare loop infiniti se l'utente inserisce per sbaglio una lettera.

1. Versione C Migliorata (Leggibile e Modulare)

Qui ho diviso il codice in funzioni logiche e pulito la gestione delle stringhe.

```
C/C++
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#ifdef _WIN32

    #include <windows.h>

    #define CLEAR "cls"

#else

    #include <unistd.h>

    #define CLEAR "clear"

#endif

// Funzione per attendere senza bruciare la CPU

void aspetta(int ms) {

    #ifdef _WIN32
```

```

        Sleep(ms);

    #else

        usleep(ms * 1000);

    #endif
}

// Effetto macchina da scrivere
void stampa_lento(const char *testo) {
    while (*testo) {
        printf("%c", *testo++);
        fflush(stdout);
        aspetta(30);
    }
}

void mostra_menu() {
    printf("\n--- MENU CALCOLATRICE ---\n");
    printf("0: Inserisci Numero | 1: + | 2: - | 3: * | 4: /\n");
    printf("5: Azzeri | 6: Binario | 7: %% | 8: Random | 9: Esci\n");
    printf("-----\n");
}

int main() {
    float risultato = 0, input_val;
    int scelta = 0, inizializzato = 0;

    stampa_lento("Benvenuto nella Calcolatrice Educativa v2.0\n");

    while (1) {

```

```
mostra_menu();

printf("Risultato attuale: %.2f\n", risultato);

printf("Scelta: ");

if (scanf("%d", &scelta) != 1) break;

if (scelta == 9) break;

if (scelta == 5) { risultato = 0; inizializzato = 0; continue; }

switch (scelta) {

    case 0:

        printf("Inserisci valore: ");

        scanf("%f", &risultato);

        inizializzato = 1;

        break;

    case 1:

        printf("Aggiungi: ");

        scanf("%f", &input_val);

        risultato += input_val;

        break;

    case 4:

        printf("Dividi per: ");

        scanf("%f", &input_val);

        if (input_val != 0) risultato /= input_val;

        else printf("Errore: Divisione per zero!\n");

        break;

    // ... altre operazioni ...

    default:

        printf("Funzione non valida.\n");

}
```

```
}  
  
return 0;  
  
}
```

2. Trasformazione in C++ (Object Oriented)

In C++ usiamo le **Classi** per incapsulare i dati e `std::bitset` per la conversione binaria, rendendo tutto molto più elegante.

C/C++

```
#include <iostream>
```

```
#include <vector>
```

```
#include <bitset>
```

```
#include <string>
```

```
#include <thread>
```

```
#include <chrono>
```

```
class Calcolatrice {
```

```
private:
```

```
    double record;
```

```
    bool attivo;
```

```
public:
```

```
    Calcolatrice() : record(0), attivo(false) {}
```

```
    void aggiungi(double val) { record += val; attivo = true; }
```

```
    void sottrai(double val) { record -= val; attivo = true; }
```

```
    void moltiplica(double val) { record *= val; attivo = true; }
```

```
    void dividi(double val) {
```

```
        if (val != 0) record /= val;
```

```

        else std::cerr << "Errore: Div per zero!\n";
    }

    void reset() { record = 0; attivo = false; }

    double getRisultato() const { return record; }

    std::string toBinary() const {
        return std::bitset<16>(static_cast<int>(record)).to_string();
    }
};

// Esempio di utilizzo logico
int main() {
    Calcolatrice miaCalc;

    miaCalc.aggiungi(10);

    miaCalc.moltiplica(2);

    std::cout << "Risultato: " << miaCalc.getRisultato() << std::endl;

    std::cout << "In binario: " << miaCalc.toBinary() << std::endl;

    return 0;
}

```

3. Interfaccia Grafica (GUI) in C++

Per creare una GUI in C++ non basta il linguaggio standard; serve una libreria esterna. La più professionale è **Qt**, ma per semplicità didattica ti mostro come apparirebbe la struttura utilizzando **Qt (QML/C++)**.

Struttura concettuale (pseudo-codice Qt)

```

C/C++
#include <QApplication>

#include <QPushButton>

```

```

#include <QVBoxLayout>

#include <QLineEdit>

#include <QWidget>

class GuiCalculator : public QWidget {
    Q_OBJECT

    QLineEdit *display;

    double currentVal = 0;

public:
    GuiCalculator() {
        auto *layout = new QVBoxLayout(this);

        display = new QLineEdit("0");
        display->setReadOnly(true);
        display->setAlignment(Qt::AlignRight);
        layout->addWidget(display);

        // Griglia di bottoni
        auto *grid = new QGridLayout();

        QStringList buttons = {"7", "8", "9", "/", "4", "5", "6", "*", "1", "2",
"3", "-", "0", "C", "=", "+"};

        int pos = 0;

        for (const QString& b : buttons) {
            auto *btn = new QPushButton(b);

            grid->addWidget(btn, pos / 4, pos % 4);

            pos++;

            // Connessione logica (Signal/Slot)
            connect(btn, &QPushButton::clicked, [this, b]() { handlePress(b); });

```

```

    }

    layout->addLayout(grid);
}

void handlePress(QString key) {

    // Logica per aggiornare il display e fare i calcoli

    display->setText(key);

}

};

int main(int argc, char *argv[]) {

    QApplication app(argc, argv);

    GuiCalculator calc;

    calc.show();

    return app.exec();

}

```

Spiegazione del funzionamento:

1. **Modularità:** Ho separato la "Logica" (la classe `Calcolatrice` che fa i conti) dalla "Visualizzazione" (la GUI).
2. **Event Driven:** A differenza del tuo codice dove il programma aspetta un input con `scanf`, nella GUI il programma "dorme" finché l'utente non clicca un bottone (evento).
3. **Conversione Binaria:** Invece di fare il ciclo con il modulo `%2`, in C++ usiamo `std::bitset` che è immediato e sicuro.

Se vuoi compilare la versione grafica, dovrai scaricare il **Qt Framework**. Se invece preferisci qualcosa di più leggero per imparare, ti consiglio di guardare la libreria **SFML** o **Dear ImGui**.

COMMENTI ALLA SOLUZIONE:

RICERCA BUG ED ERRORI:

Si nota che la risposta dell' AI non ha risolto correttamente l'esercizio.

1. Il codice C devono essere apportate modifiche

- non sono presenti tutte le funzioni

2. Il codice C++ devono essere apportate modifiche

- non sono presenti tutte le funzioni
- logica risposta: mancanza input

3. Il codice C++ GUI devono essere apportate modifiche

- non compila: test funzionamento non possibile

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo funzioni denominate:

1. Il codice C comprende:

```
void aspetta()
```

```
void stampa_lento()
```

```
void mostra_menu()
```

2. Il codice C++ comprende all'interno di una class:

```
void aggiungi()
```

```
void sottrai()
```

```
void moltiplica()
```

```
void dividi()
```

```
void reset()
```

```
double getRisultato()
```

```
std::string toBinary()
```

3. Il codice C++ GUI comprende:

```
void handlePress()
```

SECONDO PROMPT SOTTOPOSTO ALL'AI:

La compilazione non va a termine, risolvi e spiega.

SECONDA RISPOSTA OTTENUTA DALL'AI:

Non scoraggiarti, passare dal C procedurale (riga per riga) a un framework professionale come **Qt** è un bel salto! Se non compila o non si apre la finestra, il "colpevole" di solito è uno di questi tre motivi.

Controlliamo insieme questi passaggi:

1. Il file di progetto (.pro)

In Qt, il compilatore deve sapere che vuoi usare i "pezzi" della grafica (i Widgets).

- Apri il file che finisce con **.pro** (quello con l'icona blu nel navigatore a sinistra).
- Assicurati che all'inizio ci sia scritto esattamente questo:

```
C/C++
QT += widgets
```

- Se hai aggiunto questa riga ora, fai **tasto destro sul nome del progetto** e seleziona **"Run qmake"**.

2. Hai creato un progetto "Qt Quick" invece di "Widgets"?

Se vedi un file **main.qml**, probabilmente hai creato un progetto **Qt Quick / Kirigami**. Quel tipo di progetto si aspetta un linguaggio diverso (QML).

- **La soluzione rapida:** Crea un nuovo progetto andando su *File -> New Project -> Application (Qt) -> Qt Widgets Application*.
- Qt Creator creerà automaticamente un file **main.cpp** pulito. Incolla lì il codice che ti ho dato sopra.

3. Errori comuni nel codice

Guarda la finestra **"Issues"** (Problemi) in basso in Qt Creator.

- Se leggi **"C1083: Cannot open include file: 'QApplication': No such file or directory"**, allora il problema è il punto 1 (manca **QT += widgets**).
- Se vedi **"Multiple definition of main"**, significa che hai due file **.cpp** entrambi con una funzione **main()**. Devi tenerne solo uno.

Struttura corretta del progetto

Ecco come dovrebbe apparire il tuo navigatore a sinistra affinché tutto funzioni:

Un piccolo trucco per "resettare"

Se hai fatto modifiche e continua a non andare:

1. Vai nel menu in alto su **Build**.
2. Clicca su **Clean Project**.
3. Clicca su **Run qmake**.
4. Premi di nuovo il tasto **Play (Verde)**.

Cosa leggi esattamente nella scheda "Issues" (Problemi) in basso? Se mi incolli l'errore specifico, posso dirti al volo quale riga cambiare!

COMMENTI ALLA SECONDA SOLUZIONE:

RICERCA BUG ED ERRORI:

File. pro non presente

LEGGIBILITÀ:

Non richiesta

MODULARITÀ:

Non richiesta

TERZO PROMPT SOTTOPOSTO ALL'AI:

Non trovo la sezione .pro, risolvi e spiega.

TERZA RISPOSTA OTTENUTA DALL'AI:

Probabilmente il tuo progetto non usa il file `.pro` perché è un progetto **CMake**. In Qt Creator moderno, **CMakeLists.txt** (lo vedi in cima alla lista nel tuo screenshot) ha preso il posto del vecchio file `.pro`.

Ecco come far funzionare la calcolatrice nel tuo progetto attuale:

1. Apri il file giusto

Nella colonna di sinistra (sotto *Source Files*), fai doppio clic su `main.cpp`.

- **Attenzione:** Non incollare il codice in `mainwindow.ui` (quello che hai aperto ora, che sembra codice HTML/XML) né in `Main.qml`.

2. Sostituisci il contenuto di `main.cpp`

Cancella tutto quello che c'è dentro `main.cpp` e incolla il codice che ti ripropongo qui sotto, leggermente adattato per essere sicuri che funzioni con il tuo sistema CMake:

```
C/C++
```

```
#include <QApplication>
```

```
#include <QPushButton>

#include <QVBoxLayout>

#include <QGridLayout>

#include <QLineEdit>

#include <QWidget>

class CalcolatriceGrafica : public QWidget {

public:

    QLineEdit *display;

    CalcolatriceGrafica() {

        setWindowTitle("Calcolatrice C++");

        auto *layout = new QVBoxLayout(this);

        display = new QLineEdit("0");

        display->setReadOnly(true);

        display->setAlignment(Qt::AlignRight);

        display->setStyleSheet("font-size: 24px; height: 50px;");

        layout->addWidget(display);

        auto *grid = new QGridLayout();

        QStringList bottoni = {"7", "8", "9", "/", "4", "5", "6", "*", "1", "2",
"3", "-", "0", "C", "=", "+"};

        int riga = 0, col = 0;

        for (const QString &testo : bottoni) {

            QPushButton *btn = new QPushButton(testo);

            btn->setMinimumSize(50, 50);

            grid->addWidget(btn, riga, col);
```

```

// Logica base per i tasti

connect(btn, &QPushButton::clicked, [this, testo]() {
    if (testo == "C") display->setText("0");
    else if (testo == "=") display->setText("Calcolo...");
    else {
        if (display->text() == "0") display->setText(testo);
        else display->setText(display->text() + testo);
    }
});

col++;

if (col > 3) { col = 0; riga++; }
}

layout->addLayout(grid);
}
};

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    CalcolatriceGrafica calc;

    calc.show();

    return app.exec();
}

```

3. Controlla il file CMakeLists.txt

Solo per sicurezza, fai doppio clic su CMakeLists.txt (il primo file in alto). Controlla che ci sia una riga simile a questa:

```
find_package(Qt6 REQUIRED COMPONENTS Widgets)
```

(Dovrebbe esserci già, visto che hai creato un progetto Widgets).

4. Avvia!

Premi il tasto **Play Verde** in basso a sinistra.

Perché prima vedevi errori?

Stavi guardando `mainwindow.ui`, che è un file di design "astratto". Il C++ non può essere eseguito lì dentro. Il "cuore" del programma deve stare sempre nei file `.cpp`.

Se premi Play e ti dà ancora errore, guarda la scheda **"Issues"** in basso e scrivimi cosa dice la prima riga di errore!

COMMENTI ALLA TERZA SOLUZIONE:

RICERCA BUG ED ERRORI:

Si nota che la risposta dell' AI non ha risolto correttamente l'esercizio.

- Il codice C++ GUI
 - stampa solo "Calcolo...",
 - non esegue nessuna delle funzioni originali

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

Non richiesta

QUARTO PROMPT SOTTOPOSTO ALL'AI:

Il progetto non esegue le funzioni richieste, risolvi e spiega.

QUARTA RISPOSTA OTTENUTA DALL'AI:

Ce l'abbiamo fatta! Vedere quella finestra apparire è il momento più soddisfacente della programmazione. Hai ufficialmente trasformato un programma testuale in "bianco e nero" in un'applicazione moderna con finestre e bottoni.

Tuttavia, come avrai notato cliccando, al momento è solo una "bella facciata": scrive solo "Calcolo..." quando premi l'uguale. Dobbiamo aggiungergli il **"cervello"** per renderla funzionale come la tua versione originale in C.

Ecco il codice finale completo per il tuo `main.cpp`. Ho aggiunto la logica per le operazioni, il calcolo della percentuale e la conversione in binario (proprio come nel tuo esercizio iniziale).

Il "Cervello" della Calcolatrice (Copia e sostituisci in `main.cpp`)

C/C++

```
#include <QApplication>

#include <QPushButton>

#include <QVBoxLayout>

#include <QGridLayout>

#include <QLineEdit>

#include <QWidget>

#include <QString>

#include <bitset>

class CalcolatriceUltimate : public QWidget {

    // Variabili per la logica (il "cervello")

    double primoNumero = 0;

    QString operazioneCorrente = "";

    bool attesaNuovoNumero = true;

public:

    QLineEdit *display;

    CalcolatriceUltimate() {

        setWindowTitle("Calcolatrice Ultimate C++");

        setFixedSize(350, 450); // Blocchiamo la dimensione per farla sembrare
professionale

        auto *layout = new QVBoxLayout(this);

        display = new QLineEdit("0");

        display->setReadOnly(true);

        display->setAlignment(Qt::AlignRight);

        display->setStyleSheet("font-size: 30px; height: 60px; background: #222;
color: #0f0; padding: 5px;");
```

```

layout->addWidget(display);

auto *grid = new QGridLayout();

// Layout bottoni: numeri + funzioni speciali del tuo esercizio originale
QStringList bottoni = {
    "7", "8", "9", "/",
    "4", "5", "6", "*",
    "1", "2", "3", "-",
    "0", "C", "=", "+",
    "%", "BIN", "RAND"
};

int riga = 0, col = 0;
for (const QString &testo : bottoni) {
    QPushButton *btn = new QPushButton(testo);
    btn->setMinimumSize(60, 60);
    btn->setStyleSheet("font-size: 18px; font-weight: bold;");
    grid->addWidget(btn, riga, col);

    connect(btn, &QPushButton::clicked, [this, testo]() {
gestisciPressione(testo); });

    col++;

    if (col > 3) { col = 0; riga++; }
}

layout->addLayout(grid);
}

void gestisciPressione(QString tasto) {

```

```
if (tasto >= "0" && tasto <= "9") {  
    if (attesaNuovoNumero) {  
        display->setText(tasto);  
        attesaNuovoNumero = false;  
    } else {  
        display->setText(display->text() + tasto);  
    }  
}  
  
else if (tasto == "C") {  
    display->setText("0");  
    primoNumero = 0;  
    operazioneCorrente = "";  
    attesaNuovoNumero = true;  
}  
  
else if (tasto == "+" || tasto == "-" || tasto == "*" || tasto == "/") {  
    primoNumero = display->text().toDouble();  
    operazioneCorrente = tasto;  
    attesaNuovoNumero = true;  
}  
  
else if (tasto == "=") {  
    double secondoNumero = display->text().toDouble();  
    double risultato = 0;  
  
    if (operazioneCorrente == "+") risultato = primoNumero +  
secondoNumero;  
  
    else if (operazioneCorrente == "-") risultato = primoNumero -  
secondoNumero;  
  
    else if (operazioneCorrente == "*") risultato = primoNumero *  
secondoNumero;
```

```

        else if (operazioneCorrente == "/") risultato = (secondoNumero != 0)
? primoNumero / secondoNumero : 0;

        display->setText(QString::number(risultato));

        attesaNuovoNumero = true;
    }

    else if (tasto == "BIN") {

        // Conversione Binaria come nel tuo codice C

        int val = display->text().toInt();

        std::string binario = std::bitset<16>(val).to_string();

        display->setText(QString::fromStdString(binario));

        attesaNuovoNumero = true;
    }

    else if (tasto == "%") {

double secondoNumero = display->text().toDouble();

double calcoloPercentuale;

if (operazioneCorrente == "+" || operazioneCorrente == "-") {

    // Se sto sommando o sottraendo, calcolo la percentuale del primo numero
    // Es: 100 + 10% -> calcola (100 * 10) / 100 = 10

    calcoloPercentuale = (primoNumero * secondoNumero) / 100.0;
} else {

    // Se sto moltiplicando o dividendo, o non c'è operazione, divido
semplicemente per 100

    // Es: 50 * 10% -> calcola 50 * 0.1

    calcoloPercentuale = secondoNumero / 100.0;
}

display->setText(QString::number(calcoloPercentuale));

// IMPORTANTE: non resettiamo l'operazioneCorrente,

```

```

        // così se premi "=" subito dopo, somma/sottrae il valore appena
        calcolato!
    }

    else if (tasto == "RAND") {
        display->setText(QString::number(rand() % 100));

        attesaNuovoNumero = true;
    }
}

};

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    CalcolatriceUltimate calc;

    calc.show();

    return app.exec();
}

```

Cosa abbiamo aggiunto?

Per far funzionare una calcolatrice serve una **Macchina a Stati**. Ecco il concetto logico che abbiamo implementato:

1. **Le Variabili di Memoria:** Abbiamo aggiunto `primoNumero` e `operazioneCorrente`. Senza di esse, il computer "dimenticherebbe" il primo numero inserito non appena premi `+`.
2. **attesaNuovoNumero:** È un flag (un interruttore). Se è vero, la calcolatrice sa che deve cancellare lo schermo prima di scrivere il prossimo numero (perché hai appena premuto un operatore).
3. **Conversione Binaria:** Invece di quel lungo ciclo `for` con `%2` che avevi nel codice C, qui usiamo `std::bitset<16>`, che è un modo molto più veloce e "professionale" per gestire i bit in C++.

Prossima sfida? Ora che la logica funziona, potresti provare a cambiare i colori nel `setStyleSheet` per personalizzarla, oppure aggiungere un bottone per calcolare la radice quadrata!

COMMENTI ALLA QUARTA SOLUZIONE:

RICERCA BUG ED ERRORI:

Si nota che la risposta dell' AI ha risolto correttamente l'esercizio.

LEGGIBILITÀ:

L'AI ha riscritto il codice, aggiungendo i commenti richiesti in fucsia:

- `// ESEMPIO`

MODULARITÀ:

L'AI ha riscritto il codice, aggiungendo una funzione denominata:

- `void gestisciPressione()`

TABELLA RIASSUNTIVA

Esempio Numero	Titolo Problema	Linguaggio di Programmazione	Errori Identificati dall'AI	Funzioni Modulari Aggiunte	Punteggio Ottenuto (0-100)	Risolto
1	Massimizzare la distanza tra studenti in aula	C / C++	Integer Overflow (gestione q1, q2 e S come int invece di long long); Comportamento indefinito nel printf; Lettura file forzata (input0.txt)	long long calcola_studenti()	100/100	Sì
2	Calcolo intervallo indirizzi IP (Hackeraggio)	C / C++	Punti e virgola mancanti; Parentesi del main errate; Operatore di assegnazione invece di uguaglianza nell'assert; Virgola mancante nello scanf; Controllo if non necessario	long long ipToNumber()	100/100	Sì
3	Festa di compleanno (Massimo amici liberi)	C / C++	Logica degli intervalli errata; Inefficienza (complessità $O(N \log N)$ con map); Errore out-of-bounds nel loop; Helper function non definita	int organizza()	100/100	Sì
4	Bustine di carte da collezione	C / C++	Variabili non definite (x); Type mismatch (bool vs string); Violazione della consegna (lettura file esterni); Logica errata nel conteggio; Indici fuori limite	int scarta()	100/100	Sì
5	Wordle Generico (Calcolo combinatorio)	C / C++	Crash intenzionale (puntatore nullo); Errore di tipo (iteratore a char); Accesso fuori dai limiti (LN); Logica di input poco robusta	long long calcolaPossibilita()	100/100	Sì
6	Spegnere lampadine (Corridoio)	C / C++	Array out-of-bounds nel ciclo Case; Array a dimensione variabile (VLA); Accesso oltre il limite nel vector A[i+1]; Errore logico nel conteggio (doppio incremento)	int risolviCaso()	50/50	Sì
7	Chi cerchia trova (Inclusione cerchi)	C / C++	Interpretazione geometrica incompleta; TLE (Time Limit Exceeded); Errore di output	long long gcd(); vector trovacerchi(); long long cross_product()	100/100	Sì
8	Pac-Man 3D (Movimento Greedy)	Python / C++	Approccio basato sul prodotto delle coordinate incompleto; Mancanza di gestione della pienezza della griglia	def solve(); void solve()	0/100	No
9	Calcolatrice derivata	C / C++ / C++ GUI (Qt)	Costanti interpretate come ottali (0045); Busy waiting (rall); Mancanza di modularità; Sicurezza input (scanf); GUI non funzionale inizialmente	void aspetta(); void stampa_lento(); void mostra_menu(); void aggiungi(); void gestisciPressione()	Not in source	Sì

VIDEO RIASSUNTIVO

Un_Partner_per_Programmare.mp4

CONCLUSIONI

IL nostro esperimento dimostra l'evoluzione dell'AI generativa da generatore di output a partner epistemico nella programmazione. L'analisi di nove casi studio evidenzia le capacità diagnostiche e risolutive dell'AI.

Differenze nell'efficacia dell'AI:

- **Errori Sintattici e di Compilazione:** L'AI rileva con precisione errori formali (punti e virgola, parentesi, operatori) e risolve rapidamente, superando l'analisi manuale.
- **Gestione della Memoria e Tipi di Dato:** L'AI identifica rischi di integer overflow, suggerendo tipi a 64-bit (long long) invece di 32-bit (Esempio 1).
- **Ottimizzazione Algoritmica:** In problemi complessi (Olimpiadi di Informatica), l'AI propone salti di paradigma, migliorando l'efficienza (Esempi 3 e 7).
- **Modularizzazione e Leggibilità:** L'AI applica il principio DRY, ristrutturando codici monolitici in funzioni specifiche (es. `calcola_studenti()`, `ipToNumber()`), migliorando manutenibilità e trasparenza.

Prospettiva Formativa:

L'AI offre un supporto prezioso nella programmazione, migliorando la qualità del codice e l'efficienza del programmatore.

Il valore aggiunto sta nella spiegazione educativa dei commenti. Spiegare perché un approccio fallisce (es. violazione dei vincoli di memoria o accesso out-of-bounds) trasforma l'interazione con la macchina in un'opportunità di apprendimento per lo studente. L'intuizione umana e l'analisi dell'AI superano i limiti della programmazione tradizionale, offrendo soluzioni eleganti e performanti.

(Abbiamo utilizzato diverse AI, tra cui Gemini, NotebookLM, Claude, ChatGPT e Copilot, esclusivamente a scopo di ricerca. Per non alterare il contenuto di output e non manipolare alcuna risposta, abbiamo lasciato i testi generati dall'IA invariati, copiando ed incollando direttamente. Di conseguenza, alcuni testi o video generati dall'AI potrebbero contenere errori sintattici o lessicali.)