Progetto S2L5

Risposta 1

Il programma svolge la funzione di un assistente digitale capace di svolgere le seguenti richieste:
-Moltiplicazione tra 2 numeri interi
-Divisione tra 2 numeri interi
-Inserimento di una stringa di caratteri.

Risposta 2

Ci sono varie casistiche che il codice non gestisce, che potrebbero limitare la sua efficacia o portare a comportamenti anomali.

Gestione dell'Overflow e input

-Non ci sono controlli per evitare l'overflow nelle operazioni di moltiplicazione e divisione. Inserendo numeri molto grandi, il risultato potrebbe superare il limite massimo dell'array e portare a comportamenti imprevisti del programma.

-Anche nella funzione ins_string non vi è un metodo di input sicuro, in questo modo inserendo un numero di caratteri che supera la dimensione dell'array, può

manifestarsi una vulnerabilità e creare problemi di sicurezza.

-Il codice non include controlli per verificare se l'input eseguito dall'utente è valido. Nella divisione per esempio, non c'è un controllo per assicurarsi che il denominatore non sia a 0, portando il programma, se digitato, a comportamenti imprevisti.

-Non c'è una gestione per casi in cui l'utente inserisca un carattere diverso dai suggeriti (A-B-C), il che porta il programma a comportamenti imprevisti come il termine di esso.

-Il programma non è composto da un ciclo che permette di eseguire più operazioni senza che esso termini l'esecuzione.

Risposta 3

Il codice ha vari errori di sintassi/logica

Errori logici

1°Nel menu iniziale vi è:

```
16  int main ()
17
18  {
19      char scelta = {'\0'};
20      menu ();
21      scanf ("%d", &scelta);
```

La riga 21 è sbagliata poiché %d è utilizzato per le variabili di tipo int, ma la variabile 'scelta' in questo caso è di tipo char, quindi la soluzione corretta è scrivere scanf("%c", &scelta);

2°Nella funzione moltiplica vi è:

```
50  void moltiplica ()
51  {
52      short int  a,b = 0;
53      printf ("Inserisci i due numeri da moltiplicare:");
54      scanf ("%f", &a);
55      scanf ("%d", &b);
56
57      short int prodotto = a * b;
58
59      printf ("Il prodotto tra %d e %d e': %d", a,b,prodotto);
60  }
```

Nella riga 52 solamente b equivale a 0, a potrebbe assumere qualsiasi altro valore, il modo corretto è short int a = 0, b = 0;
Nella riga 54 scan f utilizza %f per indicare a, ma a è una variabile short int, quindi bisogna utilizzare %hd, ugualmente nella riga 54, anche se %d è spesso compatibile con short int.
Nella riga 57 la variabile prodotto indicata con short int potrebbe portare a un overflow se il prodotto di **a** e **b** supera il range di short int, bisogna utilizzare int per evitare possibili comportamenti imprevisti.

Nella riga 59 bisogna sostituire %d %d delle variabili short int (a,b) con %hd %hd.

3°Nella funzione divisione vi è

```
63  void dividi ()
64  {
65          int  a,b = 0;
66          printf ("Inserisci il numeratore:");
67          scanf ("%d", &a);
68      printf ("Inserisci il denumeratore:");
69          scanf ("%d", &b);
70
71          int divisione = a % b;
72
73          printf ("La divisione tra %d e %d e': %d", a,b,divisione);
74  }
```

Nella riga 65 scriviamo int a = 0, b = 0 per le motivazioni sopra elencate.
Nella riga 71 il segno % equivale a un modulo, mentre è corretto utilizzare /. Inoltre se vogliamo che il risultato sia preciso, utilizziamo la variabile float, in modo da ricevere anche esiti decimali
float divisione = (float)a / b;
Nella riga 73 cambiamo la variabile %d che rappresenta la divisione in %f.
Se volessimo gestire la casistica in cui digitando 0 come denominatore, si visualizzi un messaggio di errore da parte del programma in cui sottolinea che non è possibile dividere x 0, bisognerebbe utilizzare questo codice

```
if (b == 0) {
        printf("Errore: il denominatore non può essere
zero.\n");
        return 1; // Termina il programma con un
messaggio di errore
    }
```

4°Nella funzione ins_string vi è:

```
80  void ins_string ()
81  {
82      char stringa[10];
83          printf ("Inserisci la stringa:");
84          scanf ("%s", &stringa);
85  }
```

Nella riga 84 troviamo un errore che può compromettere la sicurezza. L'utilizzo di scanf ("%s", &stringa); può portare ad un buffer overflow, poiché digitando una stringa più lunga di 9 caratteri, dato che non vi è un impedimento, potrebbe sovrascrivere altre parti della memoria, portando a comportamenti imprevisti e problemi di sicurezza.
La soluzione è utilizzare fgets, preferita rispetto a scanf nella lettura di stringhe poichè offre una maggiore sicurezza contro i buffer overflow, specificando il numero massimo di caratteri da leggere. In questo caso il codice sarebbe:
fgets(stringa, sizeof(stringa), stdin);