**Errori nel codice iniziale:**

1. Errore nello “switch” (gestione del “defaul”):

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Il codice non gestisce input non validi (come 'D' o altre lettere) nel ciclo `switch`. Bisogna aggiungere un caso ”default” o simili per gestire questi scenari.

1. Errore nell'input della scelta:

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

- Nella funzione “main”, si usa “scanf("%d", &scelta);” per leggere un carattere. Tuttavia, “%d” è un formato per gli interi, bisogna usare “ %c” per leggere un carattere.

1. Errore in “moltiplica”:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

- E’ stato usato “scanf("%f", &a);” per leggere un “short int”, il che è incorretto perché “%f” è per i float. Bisognerebbe usare “%hd” per i “short int”.

- La moltiplicazione è corretta, ma ci sono problemi con l'input.

1. Errore in “dividi”:
2. Immagine che contiene testo, schermata, Carattere, numero

   Descrizione generata automaticamente

E’ stato usato l'operatore modulo “%” anziché l'operatore di divisione “/”, che non è corretto se si vuole dividere due numeri.

Potrebbe esserci anche un problema se “b” è zero, poiché ciò causerebbe un errore di divisione per zero.

1. Potenziale problema “ins\_string”:
2. Immagine che contiene testo, Carattere, linea, schermata

   Descrizione generata automaticamente

- La funzione “ins\_string” legge una stringa, ma non gestisce i casi in cui l'input supera la dimensione dell'array “stringa[10]”. Si potrebbe considerare l'uso di “fgets” per una lettura più sicura.

**Rischio attacco BOF**

Ho riscontrato un possibile attacco di Buffer Overflow (BOF) nel codice iniziale, specificatamente nella funzione `ins\_string`. Nella versione originale del programma, la funzione utilizzava un array `char stringa[10]` per memorizzare l'input dell'utente. L'input veniva letto mediante `scanf("%s", &stringa);`, metodo non sicuro perché ”scanf” con “%s” non impone un limite al numero di caratteri letti, accettando input di qualsiasi lunghezza fino al primo spazio bianco o alla fine dell'input. Questo avrebbe potuto facilmente causare un overflow se l'utente avesse inserito una stringa di lunghezza superiore a 9 caratteri (10 meno il carattere nullo di terminazione), potenzialmente permettendo l'esecuzione di codice arbitrario o causando il crash del programma.

Per mitigare questo rischio, ho modificato l'implementazione della funzione “ins\_string” nel modo seguente: ho incrementato la dimensione dell'array a 50 caratteri per consentire una maggiore flessibilità nell'input dell'utente, riducendo il rischio di overflow per input comuni. Inoltre, ho sostituito l'uso di “scanf” con “fgets(stringa, sizeof(stringa), stdin);”. “fgets” è notevolmente più sicuro in quanto limita il numero di caratteri letti alla dimensione dell'array, proteggendo così da overflow indipendentemente dalla lunghezza dell'input fornito dall'utente. Inoltre, “fgets” legge fino alla newline inclusa, che ho gestito rimuovendo il carattere newline dalla stringa risultante con “stringa[strcspn(stringa, "\n")] = '\0';”. Questo non solo previene il buffer overflow, ma mantiene anche l'integrità dei dati inseriti dall'utente, evitando problemi nelle operazioni di stringa successive che potrebbero aspettarsi una stringa ben formattata senza newline al suo interno.

**Programma corretto:**

**Immagine che contiene schermata

Descrizione generata automaticamente**

**Immagine che contiene testo, schermata, design

Descrizione generata automaticamente**

**Avvio programma su kali:**

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

**Spiegazione programma corretto:**

Questo programma è un assistente digitale semplice che offre tre funzioni principali:

1 Moltiplicare due numeri: l’utente inserisce due numeri e il programma calcola il prodotto

2 Dividere due numeri: l’utente inserisce un numeratore e un denominatore e il programma calcola il quoziente, controllando che il denominatore non sia zero.

3 Inserire una stringa: L’utente può inserire una stringa di testo e il programma la memorizza e la visualizza

1)Il programma è iniziato con l’inclusione delle librerie:

**#include <stdio.h>** (Libreria standard che permette di utilizzare funzioni come “printf”, “scanf”

**#include <string.h**>(Libreria utilizzata per gestire stringhe come “strcspn”, utilizzata per manipolare stringhe di caratteri.

2)Successivamente sono passato alla dichiarazione delle funzioni che saranno utilizzate più avanti nel codice, permettendo al programma di riconoscerle prima della loro definizione effettiva:

**void menu();**

**void moltiplica();**

**void dividi();**

**void ins\_string();**

3)Poi sono passato alla funzione principale da cui ogni programma C inizia l’esecuzione

E successivamente ho dichiarato una variabile di tipo carattere per memorizzare la scelta dell’utente nel menu.

**int main() {**

**char scelta;**

4)A seguire abbiamo un ciclo che esegue un blocco di codice fino a quando la condizione del while è vera:

**do {**

**menu();**

**scanf(" %c", &scelta);**

**getchar();**

Abbiamo anche la chiamata della funzione menù che visualizza le opzioni disponibili.

Con scanf(“ %c”, &scelta) legge un carattere dall’input dell’utente e lo memorizza in “scelta”. Mentre lo spazio prima di %c ignora qualsiasi spazio bianco prima del carattere.

Poi getchat(); legge e scarta il carattere newline rimanente nel buffer di input dopo la lettura della scelta.

Il ciclo while, continua fino a quando la scelta dell’utente non è una delle opzioni valide, mentre con il printf viene informato l’utente che la scelta non è valida

**while (scelta != 'A' && scelta != 'a' &&**

**scelta != 'B' && scelta != 'b' &&**

**scelta != 'C' && scelta != 'c') {**

**printf("Scelta non valida. Riprova.\n");**

**menu();**

**scanf(" %c", &scelta);**

**getchar();**

**}**

5)Successivamente passiamo allo switch:

Abbiamo una struttura di controllo che esegue un blocco di codice diverso in base al valore di “scelta”.

Con Case”X” viene specificato un blocco di codice da eseguire per un particolare valore di “scelta”

In base alla scelta vengono richiamate le funzioni, mentre con break termina l’esecuzione del blocco “switch”

**switch (scelta) {**

**case 'A': case 'a':**

**moltiplica();**

**break;**

**case 'B': case 'b':**

**dividi();**

**break;**

**case 'C': case 'c':**

**ins\_string();**

**break;**

**}**

6)Dopo lo switch abbiamo la richiesta per l’utente, se desidera continuare o uscire dal programma, ripetendo il ciclo se l’utente risponde “Y” o “y”

**printf("Hai bisogno di altro? (Y per continuare, N per uscire): ");**

**scanf(" %c", &scelta);**

**getchar();**

**} while (scelta == 'Y' || scelta == 'y');**

7)Poi abbiamo un messaggio di arrivederci, nel caso l’utente non volesse più continuare, con retur 0; che è un segno di terminazione senza errori.

**printf("Arrivederci!\n");**

**return 0;**

**}**

8)A seguire abbiamo le funzioni che vengono definite successivamente e gestiscono rispettivamente la visualizzazione del menu, della moltiplicazione, la divisione e l’inserimento di una stringa.

**void menu() {**

printf("Benvenuto, sono un assistente digitale, posso aiutarti a sbrigare alcuni compiti.\n");

printf("A >> Moltiplicare due numeri\nB >> Dividere due numeri\nC >> Inserire una stringa\n");

}

**void moltiplica() {**

int a, b;

printf("Inserisci il primo numero da moltiplicare: ");

scanf("%d", &a);

getchar();

printf("Inserisci il secondo numero da moltiplicare: ");

scanf("%d", &b);

getchar();

long long int prodotto = (long long) a \* b;

printf("Il prodotto tra %d e %d è: %lld\n", a, b, prodotto);

}

Spiegazione:

1)Vengono dichiarate due variabili intere, “a” e “b”, che serviranno a memorizzare i numeri da moltiplicare.

2)Viene richiesto all'utente di inserire il primo numero. Il valore viene letto e memorizzato nella variabile “a”.

3)getchar() rimuove il carattere newline residuo dopo l'input per evitare conflitti con successive letture.

4)Analogamente al primo, viene richiesto e letto il secondo numero, memorizzato in “b”.

5)Ancora una volta, “getchar()” viene utilizzato per pulire il buffer.

6)Si calcola il prodotto di “a” e “b”, utilizzando il tipo “long long int” per gestire grandi numeri.

**void dividi() {**

int a, b;

printf("Inserisci il numeratore: ");

scanf("%d", &a);

getchar();

printf("Inserisci il denominatore: ");

scanf("%d", &b);

getchar();

if (b != 0) {

double divisione = (double) a / b;

printf("La divisione tra %d e %d è: %.4f\n", a, b, divisione);

} else {

printf("Errore: Il denominatore non può essere zero.\n");

}

}

Spiegazione:

1) Vengono dichiarate le variabili “a” e “b” per il numeratore e il denominatore.

2)Si legge il numeratore.

3)Si rimuove il newline dal buffer.

4)Si legge il denominatore.

5)Si rimuove il newline.

6)Si controlla se il denominatore è diverso da zero. In caso positivo, si effettua la divisione e si stampa il risultato con quattro cifre decimali. Se è zero, si segnala un errore.

**void ins\_string() {**

char stringa[50];

printf("Inserisci la stringa: ");

fgets(stringa, sizeof(stringa), stdin);

stringa[strcspn(stringa, "\n")] = '\0';

printf("Hai inserito: %s\n", stringa);

}

Spiegazione:

1)Vengono allocati 50 caratteri per memorizzare la stringa.

2)Viene richiesta una stringa all'utente. “fgets()” legge fino a 49 caratteri (per lasciare spazio al terminatore nullo) o fino a quando non trova un newline.

3)”strcspn()” trova la posizione del primo newline e lo sostituisce con un terminatore di stringa null (“\0”), garantendo che la stringa sia ben formattata.

4)Viene mostrato all'utente ciò che ha inserito.