1)Analisi del codice.

Dalle prime righe del codice fornito possiamo notare che ci sono 4 funzioni di tipo void che servono rispettivamente (*moltiplicazione , divisione*) a eseguire operazioni aritmetiche basilari e stamparne il risultato a schermo insieme all'input ricevuto. La funzione *ins_string* invece permette di inserire una stringa in un array. Ed infine la funzione *menu* che non esegue nessuna operazione se non stampare delle frasi di "benvenuto" utili all'utente a capire come funziona il programma. Dalla riga 11 alla 30 invece c'è una porzione di codice atta a chiedere all'utente quale funzione di quelle citate in precedenza invocare (ad eccezione di *menu* che viene eseguita subito) per fare ciò oltre a chiedere con uno scanf un input, utilizza uno switch per selezionare la funzione.

2) Errori di sintassi e logici.

Prima di analizzare le vulnerabilità del codice ritengo sia più opportuno correggere la sintassi e gli eventuali errori logici. Nella fase di dichiarazione delle funzioni non si riscontrano errori di alcun tipo (riga da 3 al 6). Iniziamo dalla *scelta* :

```
char scelta = {'\0'};
menu ();
scanf ("%d", &scelta);
```

L'input richiesto da scanf è di tipo diverso e sbagliato rispetto alla dichiarazione della variabie che è di tipo char. Bisogna sostituire con *" %c".* Così facendo riceverà un carattere come input.

Non si riscontrano tali errori nel resto della porzione di codice *scelta* e nella funzione *menu*. Passiamo ora alla funzione *moltiplicazione*.

```
void moltiplica ()
{
        short int  a,b = 0;
        printf ("Inserisci i due numeri da moltiplicare:");
        scanf ("%f", &a);
        scanf ("%d", &b);

        short int prodotto = a * b;

        printf ("Il prodotto tra %d e %d e': %d", a,b,prodotto);
```

Notiamo subito un errore simile al precedente, la variabile a è di tipo int ma viene chiesto all'utente una variabile di tipo float. inoltre nella stringa di output viene di nuovo stampato un tipo intero. Correggiamo con "%d" e di conseguenza anche l'istruzione iniziale della funzione chiedendo all'utente di inserire 2 numeri **INTERI**. Inoltre non ha senso utlizzare la funzione *short* ne nelle variabili ne nel prodotto in quanto il programma è molto semplice, senza tener conto che porta ad errore nel caso in cui i numeri inseriti siano troppo lunghi o per esempio 1 poichè vengono codificati in complemento 2.

Passiamo ora alla prossima funzione:

```
void dividi ()
{
        int  a,b = 0;
        printf ("Inserisci il numeratore:");
        scanf ("%d", &a);
        printf ("Inserisci il denumeratore:");
        scanf ("%d", &b);

        int divisione = a % b;

        printf ("La divisione tra %d e %d e': %d", a,b,divisione);
```

Evidenziato nell'immagine vediamo che viene usato % al posto di /. Tale operatore infatti non restituisce la divisione ma solo il suo resto. Dobbiamo specificare all'utente che deve inserire numeri interi.(inoltre nel secondo printf notiamo che c'è scritto "denumeratore" al posto di "denominatore").
Non si riscontrano altri errori nel resto del codice.

3)Bug Hunting

Partiamo dalla parte di codice di scelta della funzione, notiamo subito che manca la condizione di ***default*** allo ***switch***, quindi dobbiamo aggiungerla così che se l'utente inserisse qualcosa di sbagliato (ovvero qualcosa non previsto dai casi preposti) ci segnali un messaggio di errore. Personalmente ritengo che utlizzare le lettere per la scelta sia abbastanza vincolante e spesso conduca ad errori di input (per esempio lettera maiuscola e minuscola) quindi ho cambiato la selezione con l'utilizzo di numeri al posto di lettere.
Passiamo alla funzione divisione,dobbiamo eseguire un controllo per evitare che venga inserito 0 come denominatore. In questo caso è stato usato un ciclo ***while***.
Per quanto riguarda la funzione di inserimento di una stringa in primis aggiungiamo un printf che stampi la stringa a schermo così da visualizzare che il funzionamento è corretto. Tale funzione presenta un problema di input in quanto un utente malevolo potrebbe causare un buffer overflow/run. Per ovviare a ciò si può alternativamente a scanf utilizzare fgets e specificare la dimensione dell'array da inserire. Oppure eseguire un controllo sull'input.