S2L5

Per agire come un Hacker bisogna capire come pensare fuori dagli schemi. L'esercizio di oggi ha lo scopo di allenare l'osservazione critica. Dato il codice in allegato, si richiede allo studente di:

1. Capire cosa fa il programma senza eseguirlo.
2. Individuare dal codice sorgente le casistiche non standard che il programma non gestisce (esempio, comportamenti potenziali che non sono stati contemplati).
3. Individuare eventuali errori di sintassi / logici.
4. Proporre una soluzione per ognuno di essi.

Il programma parla di un assistente digitale dove è capace di fare moltiplicazioni e divisioni tra due numeri oppure di inserire una stringa di caratteri, ma guardando il codice notiamo alcuni errori.

Errori nel codice

```
 9 int main ()
10
11 {
12     char scelta = {'\0'};
13     menu ();
14     scanf ("%d", &scelta);
15
16     switch (scelta)
17     {
18         case 'A':
19         moltiplica();
20         break;
21         case 'B':
22                 dividi();
23                 break;
24         case 'C':
25                 ins_string();
26                 break;
27     }
28
29 return 0;
```

Nel main gli **errori logici** sono: riga 16.
Nel main gli **errori di sintassi** sono: riga 12 e 14.

```
43 void moltiplica ()
44 {
45     short int  a,b = 0;
46     printf ("Inserisci i due numeri da moltiplicare:");
47     scanf ("%f", &a);
48     scanf ("%d", &b);
49
50     short int prodotto = a * b;
51
52     printf ("Il prodotto tra %d e %d e': %d", a,b,prodotto);
53 }
```

Nella funzione moltiplica gli **errori logici** sono: riga 45, 50 e 52.

Nella funzione moltiplica gli **errori di sintassi** sono: riga 47 e 48.

```
56 void dividi ()
57 {
58         int  a,b = 0;
59         printf ("Inserisci il numeratore:");
60         scanf ("%d", &a);
61     printf ("Inserisci il denumeratore:");
62         scanf ("%d", &b);
63
64         int divisione = a % b;
65
66         printf ("La divisione tra %d e %d e': %d", a,b,divisione);
67 }
```

Nella funzione dividi gli **errori logici** sono: riga 58, 60, 62, 64, 66.

```
72 void ins_string ()
73 {
74     char stringa[10];
75         printf ("Inserisci la stringa:");
76         scanf ("%s", &stringa);
77 }
```

Nella funzione inserisci stringa gli **errori logici** sono: riga 76.

Spiegazione degli errori

- Riga 14 il codice corretto è " %c", preferibilmente inseriamo lo spazio perché alcuni compilatori senza lo spazio possono bypassare l'input dell'utente.
- Nella riga 16 lo switch non prevede la casistica default, inoltre, C è case sensitive quindi se l'utente inserisce la a minuscola lo switch non effettua la moltiplicazione.
- Riga 45, 50 e 52 gli short int sono troppo piccoli per la moltiplicazione
- Riga 47 f è per i float, serve d per int.
- Riga 58, 64 bisogna mettere float perché la divisione tra due numeri può risultare con la virgola, inoltre in riga 64 è inserito il modulo che ritorna il resto, bisogna inserire lo slash /.
- Riga 60, 62, 66 dobbiamo togliere la %d ed inserire la %f a questo punto per lo scanf stampiamo %.2f per maggior leggibilità, perché stamperà soltanto 2 cifre dopo lo 0.
- Riga 60, 61, 62 il problema è che non c'è un controllo per il denominatore sia 0, perché in quel caso la divisione non è possibile. Inoltre, c'è un errore di grammatica, ovvero il nome il "denumeratore".
- Riga 76 l'errore è per l'overflow quindi nello scanf inseriremo %9s così l'utente anche se inserirà troppi caratteri il programma prenderà soltanto i primi 9 caratteri. Secondo errore, non è necessario la & perché il puntatore punta alla prima posizione dell'array. Dato che il codice non prevedeva il ritorno a schermo del print della riga, l'ho inserito in un print.

Soluzione

Per risolvere l'input non previsto dai case bisogna mettere un default, invece per le lettere in minuscolo bisogna creare altri case con le lettere in minuscolo. Per impaginazione migliore ho portato la funzione moltiplica al livello delle altre istruzioni con un TAB.

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

    switch (scelta)
    {
        case 'A':
        case 'a':
                moltiplica();
        break;
        case 'B':
        case 'b':
                dividi();
                break;
        case 'C':
        case 'c':
                ins_string();
                break;

         default:
                printf ("Comando errato!\n");
         break;
    }
return 0;
}
```

Per non moltiplicare numeri grandi e sforare i byte per lo short int ho inserito il type int.

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

    int prodotto = a * b;

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

Due numeri interi possono generare una divisione float quindi ho cambiato e inserito il type float e per ovviare all'errore della divisione /0 ho inserito un ciclo while affinché il denominatore esca solo se sarà diverso da 0.

```c
void dividi ()
{
        float  a,b = 0;
        printf ("Inserisci il numeratore:");
        scanf ("%f", &a);
        while(b == 0)
        {
        printf ("Inserisci il denominatore:");
        scanf ("%f", &b);
        }
        float divisione = a / b;

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

Per evitare che l'utente inserisca troppi caratteri e quindi incappare nell'overflow ho inserito %9s così l'utente anche se inserirà troppi caratteri il programma prenderà soltanto i primi 9 caratteri.
Inoltre, ho implementato il codice con un printf che ti mostrava a schermo la stringa dato che mancava un output per l'utente.

```c
void ins_string ()
{
    char stringa[10];
        printf ("Inserisci la stringa:");
        scanf ("%9s", stringa);
        printf("La stringa inserita e': %s", stringa);
}

```