REPORT Esercizio S2L5

Traccia: 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:

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

SVOLGIMENTO

Il programma in questione emula un assistente digitale che permette di fare alcune operazioni: moltiplicare due numeri, fare la divisione tra due numeri, inserire una stringa. È progettato per dare inizialmente in output un menu principale (chiamato con la funzione **menu()**) in cui è possibile scegliere l'operazione da fare attraverso l'inserimento di un carattere:

- A → richiama la funzione **moltiplica()**, che permette la moltiplicazione tra due numeri;
- B → richiama la funzione **dividi()**, che fa la divisione tra due numeri;
- C → richiama la funzione **ins_string()**, che permette l'inserimento di una stringa.

Queste tre funzioni sono di tipo **void**. Le funzioni di tipo void non restituiscono nessun risultato, inoltre non devono contenere necessariamente l'istruzione **return**.

INDIVIDUAZIONE DI EVENTUALI ERRORI E CASISTICHE NON STANDARD

Di seguito verranno inseriti gli screen degli errori individuati nel codice e le relative spiegazioni, comprese di casistiche non gestite.

```
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          }
```

ERRORE 1
Errore di sintassi: Nella riga 12 la variabile **scelta** è stata definita di tipo **char** ma, successivamente, in riga 14 è stata utilizzata una sintassi sbagliata per definirla nella funzione scanf(), ovvero **“%d”**, che viene utilizzata per definire le variabili di tipo **int**.

ERRORE 2
Casistica non gestita: nell’utilizzare l’istruzione condizionale **switch()** in riga 16 è stata omessa la casistica **default.** Tale casistica viene messa quando la variabile *scelta* non assume nessuno dei valori specificati dalle casistiche proposte **case**(case A, case B, case C), allora viene definito un blocco di istruzioni di default.

ERRORE 3
Casistica non gestita: nel caso in cui l’utente inserisce come input i caratteri “a”, ”b”, “c”, il programma non procede con l’esecuzione. Questo avviene perché le uniche casistiche ammissibili in questo caso sono “A”, “B”, “C” (maiuscole).

```
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 }
```

ERRORE 4

Errore di sintassi: Nella funzione **moltiplica()**, alla riga 45 le variabili “a” e “b” sono dichiarate di tipologia **short int**, ma nella riga 47-48, nella funzione scan(), vengono identificate in maniera sbagliata con “%f” e “%d” rispettivamente.

ERRORE 5
Errore logico: nella riga 52, nella funzione printf(), le variabili “a”, “b” e “prodotto” sono identificate in modo sbagliato, con “%d” entrambe.

```
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 }
```

ERRORE 6
Errore di sintassi: in riga 64 la variabile “divisione” è stata dichiarata di tipologia **int**, ma essendo una divisione il risultato potrebbe non essere un numero intero, ma con la virgola.

ERRORE 7
Errore logico: in riga 66 la variabile “divisione”, essendo che può essere un numero con la virgola, non può essere identificata con “%d”.

ERRORE 8
Errore logico: in riga 64 l’operatore “%” non fa la divisione tra due numeri ma restituisce il resto della divisione.

ERRORE 9
Casistica non gestita: nel caso in cui l’utente immetta come denominatore il numero 0, il programma restituirà un errore. È indicato inserire una condizione che non permetta l’inserimento del numero 0 al denominatore.

```
73 void ins_string ()
74 {
75         char stringa[10];
76         printf ("Inserisci la stringa:");
77         scanf ("%s", &stringa);
78 }
```

ERRORE 10
Errore logico: in riga 77, nella funzione scanf(), avendo identificato la variabile “stringa” con “%s”, non risulta necessario inserire l’operatore “&” vicino la variabile “stringa”. Quando si utilizza **scanf()** per leggere una stringa, non è necessario utilizzare l'operatore di indirizzo (&) per l'array di caratteri, in quanto gli array in C sono già puntatori alla loro prima posizione di memoria.

ERRORE 11
In riga 77 si trova anche un errore legato ad un eventuale overflow. Siccome non viene specificato alcun limite nella lunghezza della stringa da prendere in input, l’utente potrebbe inserire una stringa più lunga di 10 caratteri, causando un overflow del buffer.

SOLUZIONI

La prima parte di codice è stata modificata con le giuste correzioni così come in figura.

```
12         char scelta = {'\0'};
13         menu ();
14         scanf (" %c", &scelta);
15
16         switch (scelta)
17         {
18                 case 'a':
19                 case 'A':
20                 moltiplica();
21                 break;
22                 case 'b':
23                 case 'B':
24                 dividi();
25                 break;
26                 case 'c':
27                 case 'C':
28                 ins_string();
29                 break;
30                 default:
31                 printf("Scelta non valida. Riprovare!");
32         }
```

- Nell'istruzione **switch()** sono state aggiunte le casistiche che tengono conto dell'inserimento di minuscole in input da parte dell'utente. È stato poi aggiunto il caso **default** per risolvere l'eventuale inserimento di un carattere diverso da quelli suggeriti.
- Nella riga 14 è stato cambiato "%d" con " %c" in quanto la variabile "scelta" è di tipo **char**, e quindi eve essere indicato il tipo di argomento giusto, ovvero "c"

- ACCORGIMENTI: in riga 12, per una questione di miglior leggibilità, si potrebbe dichiarare la variabile "scelta" evitando l'uso delle parentesi graffe, come di seguito:

```
12          char scelta = '\0';
```

Per quanto riguarda la funzione moltiplica, si sono inserite le seguenti correzioni.

```
48 void moltiplica ()
49 {
50          short int  a,b = 0;
51          printf ("Inserisci i due numeri da moltiplicare:");
52          scanf ("%hd", &a);
53          scanf ("%hd", &b);
54
55          short int prodotto = a * b;
56
57          printf ("Il prodotto tra %hd e %hd e': %hd", a,b,prodotto);
58 }
```

- Siccome le variabili "a", "b" e "prodotto" sono state definite di tipo **short int**, devono essere indicate con il tipo di argomento giusto, ovvero "%hd", invece che con "%f" ad esempio. Allo stesso modo anche in riga 57 sono stati cambiati i tipi di argomento nella funzione printf().

- ACCORGIMENTI: qui le variabili sono state definite di tipo **short int**, ma questa tipologia indica un numero di 2 byte che va da [-32768 : 32768]. Siccome c'è la possibilità di avere un prodotto maggiore, si può considerare l'idea di mettere le variabili di tipo **int** come nella seguente figura

```
48 void moltiplica ()
49 {
50          int  a,b = 0;
51          printf ("Inserisci i due numeri da moltiplicare:");
52          scanf ("%d", &a);
53          scanf ("%d", &b);
54
55          int prodotto = a * b;
56
57          printf ("Il prodotto tra %d e %d e': %d", a,b,prodotto);
58 }
```

Per quanto riguarda la funzione dividi(), si sono applicate le seguenti modifiche:

```
61 void dividi ()
62 {
63         int  a,b = 0;
64         printf ("Inserisci il numeratore:");
65         scanf ("%d", &a);
66
67         while ( b == 0){
68
69         printf ("Inserisci il denumeratore:");
70         scanf ("%d", &b);
71         if (b == 0){
72         printf("Il denominatore deve essere diverso da zero!\n");
73         }
74         }
75
76         float divisione = (float) a / b;
77
78         printf ("La divisione tra %d e %d e': %.2f", a,b,divisione);
```

- Nella funzione **dividi()** non è stata gestita la casistica di un denominatore uguale a zero (0). È stato quindi inserito un ciclo while che, nel caso di un numeratore pari a 0, chiede all'utente di reinserirlo facendo attenzione a mettere un numero diverso da 0.
- In riga 76, la divisione era definita con l'operatore **%**, che in realtà restituisce il resto della divisione, non il risultato. Si è quindi cambiato l'operatore inserendo " **/** ". Inoltre, siccome la divisione può dare come risultato un numero non intero, si è cambiata la tipologia di variabile di "divisione" da **int** a **float**. Per poter avere in output a schermo un numero con la virgola, in riga 78, si è specificato il tipo di variabile "divisione" con "**%.2f**", che oltre a identificare la variabile come float inserisce 2 cifre decimali.

Per la funzione int_string() le modifiche fatte sono quelle mostrate nella figura sottostante.

```
85 void ins_string ()
86 {
87         char stringa[10];
88         printf ("Inserisci la stringa:");
89         scanf ("%9s", stringa);
90         printf ("La stringa che hai inserito e': %s", stringa);
91 }
```

- In riga 89 è stato tolto l'operatore di indirizzo & vicino la variabile "stringa", in quanto la stringa è già un array e quindi non necessita di un operatore di indirizzo di memoria.
- In riga 89 si trova un errore di overflow che può essere evitato specificando la lunghezza massima della stringa nell'argomento di **scanf()**, si è infatti aggiunto "**%9s**".

- ACCORGIMENTI: È stata poi inserita una riga di printf() per poter mostrare in output la riga inserita in modo da farla vedere all'utente.

APPROFONDIMENTI: l’errore di overflow è una minaccia per la sicurezza del sistema. Un eventuale attaccante esterno potrebbe sfruttare un buffer overflow per sovrascrivere dati in memoria ed eseguire un proprio codice che gli può consentire di ottenere un accesso privilegiato al sistema, sfruttare vulnerabilità ed eseguire programmi comandi non autorizzati.