S2.L5 Compito settimanale in.C

Lo scopo dell'esercizio odierno era di dimostrare come riuscire ad essere un hacker professionista ragionando con un osservazione critica e con una certa unicità, pensando fuori dagli schemi. dato il seguente codice ci viene richiesto 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-scrivere sotto quali sono gli errori
4-screen del programma corretto con spiegazione.

Il programma menzionato serve come un assistente virtuale che ha il compito di eseguire operazioni matematiche come moltiplicazioni e divisioni, oppure di accettare l'input di una sequenza di caratteri.

```
 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;
30
31 }
```

dalla riga di codice 17 alla 27 ho rappresentato il blocco di codice switch rappresentandolo come un errore logico/sintattico

```
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 riga di codice 47-48-52 sono presenti degli errori di origine sintattica

```c
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);
}
```

nella riga 64 è presente un errore di origine sintattica

```c
void ins_string ()
{
        char stringa[10];
        printf ("Inserisci la stringa:");
        scanf ("%s", &stringa);
}
```

nella riga 77 è presente un errore di origine sintattica

Nella parte superiore della discussione sono stati evidenziati problemi relativi alla logica e alla sintassi all'interno di specifiche sezioni del codice sorgente

```c
scanf ("%f", &a);
scanf ("%d", &b);
```

È consigliabile inserire uno spazio all'interno della chiamata di funzione "scanf", come in "scanf (" *uno spazio* %f")". Questo approccio è considerato una buona pratica di programmazione in quanto facilita la distinzione dei caratteri e può aiutare a prevenire errori durante la fase di compilazione del codice.

```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("Scelta non valida\n")
                break;
        }

return 0;

}
```

Nella riga 14 del codice, è stato effettuato un cambiamento nel formato di lettura da '%d', utilizzato per numeri interi, a '%c', per accogliere un singolo carattere in input. Questa modifica è necessaria per consentire al programma di gestire l'input dell'utente come carattere anziché come numero.Per quanto riguarda il blocco "switch", sono stati inclusi casi aggiuntivi per ogni lettera minuscola ('a', 'b', 'c'), per permettere al programma di riconoscere e rispondere correttamente alle opzioni inserite dall'utente indifferentemente dalla loro capitalizzazione.In aggiunta, è stato implementato un ramo "default" che verrà eseguito nel caso in cui l'utente inserisca un carattere non previsto dal programma. Questo ramo emette un messaggio che informa l'utente di aver compiuto un'errata digitazione. Queste modifiche migliorano la robustezza e l'usabilità del programma gestendo una gamma più ampia di input dell'utente e fornendo un feedback diretto in caso di errori.

```c
char scelta = ' ';
```

Il codice mostrato istanzia una variabile di nome `scelta`, di tipo carattere, e le viene assegnato il valore '\0'. Questo carattere è comunemente usato per denotare la fine di una stringa in C/C++. La correttezza sintattica del codice non è in discussione, ma la sua idoneità può essere valutata solo conoscendo il contesto nel quale viene impiegato. Se si intende che `scelta` indichi un non-valore o un'opzione non scelta, allora l'uso di '\0' è pertinente. Diversamente, se `scelta` deve contenere un valore significativo che corrisponde a una scelta concreta, allora assegnare '\0' non sarebbe congruente con tale intento.

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

Le istruzioni presenti alle linee 47, 48 e 52 sono state modificate per operare con variabili di tipo SHORT INT, adottando l'operatore "%hd" nel formato delle funzioni scanf e printf: rispettivamente scanf("%hd", &a), scanf("%hd", &b) e printf("il prodotto tra %hd e %hd e': %hd", a, b, prodotto). È importante notare che il tipo di dato SHORT INT è un intero a 16 bit, che può variare da -32,768 a 32,767. Al contrario, un INT standard è un intero a 32 bit, con un range da -2,147,483,648 a 2,147,483,647. Pertanto, il codice menzionato è stato adattato per utilizzare il tipo di dato SHORT INT, come segue.

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

Nell’immagine sopradiacente mostro la correzione.
Una funzione in programmazione è un insieme di istruzioni che compie un'operazione definita e può essere invocata in altre sezioni del codice per realizzare quella stessa operazione. Per esempio, la funzione void moltiplica() ci consente di moltiplicare due numeri interi e visualizzare il risultato. Il termine "void" in questo contesto denota che la funzione non fornisce un valore di ritorno. In linguaggio C, quando una funzione è dichiarata come "void", significa che non restituirà alcun valore dopo la sua esecuzione. Questo tipo di funzioni è tipicamente impiegato per svolgere attività o manipolazioni che non necessitano di un output ritornato.

```
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 71 è stata fatta una modifica, sostituendo il simbolo % (che rappresenta l'operatore modulo) con quello della divisione (/). La funzione dividi() è progettata per dividere due numeri interi e mostrare il risultato sullo schermo. Personalmente, suggerirei l'uso di una variabile di tipo FLOAT o DOUBLE per gestire il risultato di una divisione, e raccomando di integrare un ciclo while che continua a richiedere l'input dell'utente finché non viene inserito un valore diverso da zero, evitando così errori di divisione per zero. Nell'esempio presentato ho optato per l'utilizzo di una variabile di tipo DOUBLE

```
63 void dividi ()
64 {
65          double  a,b = 0;
66          printf ("Inserisci il numeratore:");
67          scanf ("%lf", &a);
68          while(b==0)
69          {
70          printf ("Inserisci il denumeratore:");
71          scanf ("%lf", &b);
72          }
73
74          double divisione = a / b;
75
76          printf ("La divisione tra %lf e %lf e': %lf", a,b,divisione);
77 }
```

Nell’immagine sopradiacente mostro che per evitare errori logici nelle divisioni, è raccomandabile adottare variabili di tipo FLOAT o DOUBLE

```
80 void ins_string ()
81 {
82          char stringa[10];
83          printf ("Inserisci la stringa:");
84          scanf ("%9s", stringa);
85 }
```

La funzione void ins_string() è progettata per accettare l'input di una stringa. Per mitigare il rischio di overflow, che potrebbe verificarsi se l'utente inserisce una stringa più lunga di 9 caratteri, abbiamo impostato scanf per leggere al massimo 9 caratteri con il formato %9s. Questo approccio previene il sovrascrivimento della memoria oltre il buffer designato, una vulnerabilità che può portare a gravi problemi di sicurezza, inclusi gli attacchi di tipo buffer overflow, dove dati nocivi possono corrompere la memoria adiacente.

Nel codice, la sintassi per scanf è stata modificata in scanf("%9s", stringa), eliminando il bisogno dell'operatore di indirizzo '&' perché gli array in C sono già passati come indirizzi di memoria. Ricorda che in un array di dimensione 10, l'ultimo elemento è accessibile all'indice [9].
Per quanto riguarda gli overflow in generale, nel contesto dei linguaggi di programmazione come il C, si verificano quando i dati inseriti eccedono la capacità della variabile assegnata, causando spesso errori di esecuzione o malfunzionamenti del software. Questi possono essere sfruttati per eseguire codice malevolo.
CONSIDERAZIONI FINALI:
A livello di codice, sarebbe prudente rivedere alcune pratiche fondamentali per migliorare l'efficienza, come l'utilizzo di variabili di tipo SHORT INT e l'esecuzione di divisioni con variabili di tipo FLOAT o DOUBLE. Inoltre, è consigliabile adottare nomi di variabili più chiari e descrittivi per facilitare la comprensione e la manutenzione del codice.