Armando Battaglino
05/11/2022

Week 2 - Esercizio TEST

BUG HUNTING: Controllo e correzione di un programma scritto in C

Consegna

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

Esercizio_10_Epicode.c

Procedimento

Domanda 1: Cosa fa il programma ancora prima di eseguirlo?
Il programma in allegato è un programma che chiede un input e in base ad esso, esegue determinate istruzioni.

Le istruzioni sono eseguite attraverso delle funzioni che sono scritte in precedenza.

Gli input possibili previsti dal programma sono 3:

- **A >>** Moltiplicazione tra due numeri:
  - Richiesta di input a terminale dei numeri da moltiplicare
  - Restituzione del risultato della divisione
- **B >>** Divisione tra due numeri:
  - Richiesta di input a terminale dei numeri da dividere
  - Restituzione del risultato della divisione
- **C >>** Inserimento di una stringa:
  - Richiesta di input a terminale di una stringa

Domanda 2: Individuazione delle casistiche non Standard non gestite

1. La prima casistica non gestita dal programma è l'input iniziale, ovvero quello in cui viene richiesto l'inserimento di uno dei tre caratteri tra **'A'**, **'B'** oppure **'C'**.

   In questo caso non viene previsto un controllo sull'input dell'utente, ad esempio se l'utente non va ad inserire uno di questi 3 caratteri oppure inserisce semplicemente un numero.

   Inoltre anche se prevedessimo il controllo attraverso **default** all'interno dello **switch**, non viene data una seconda possibilità di inserire un carattere dopo aver sbagliato l'inserimento dell'input. Dunque è necessario attivare questo ciclo attraverso il un **do while** oppure un semplice **while**

   Infine il programma ci da solo 3 opzioni, senza prevedere l'opzione di terminare il gioco, che va aggiunta.

Ecco dunque il codice che ho creato per risolvere questa casistica:

```c
int main () {

        char scelta = '\0';
        int continua_gioco = 1;
        menu();

        do{
                scanf("%c", &scelta);

                switch (scelta){
                        case 'A':
                                continua_gioco = 1;
                                moltiplica();
                                break;
                        case 'B':
                                continua_gioco = 1;
                                dividi();
                                break;
                        case 'C':
                                continua_gioco = 1;
                                ins_string();
                                break;
                        case 'F':
                                printf("\nTermino il programma ... ");
                                break;
                        default:
                                printf("\nInserisci solo A B C o F!\n\n");
                                continua_gioco = 0;
                                rewind(stdin);

                }

        } while(continua_gioco == 0);

return 0;

}
```

2. La seconda casistica non prevista è in tutte le funzioni **void** che sono state create.

   Essenzialmente il programmatore del codice non ha previsto nuovamente il controllo sull'input:

   - **moltiplica()**
     In questa funzione non viene controllato se nell'input sono inseriti caratteri e non è controllata la grandezza del numero inserito in input.

     Dunque se ad esempio vengono inseriti numero troppo grandi per la casella di memoria che abbiamo dichiarato, potrebbe esserci la possibilità che ci sia un Overflow.

     Nel caso in cui l'utente sbagli ad inserire l'input, non viene data la possibilità di reinserimento (risolviamo con un **do while**).

```c
void moltiplica () {
        int controllo_uno, controllo_due;
        short int a,b,prodotto = 0;

        printf ("Inserisci i due numeri da moltiplicare\n(I numeri decimali saranno arrotondati per DIFETTO):\n");

        do {

                rewind(stdin);
                printf("Numero 1: ");
                controllo_uno = scanf ("%hd", &a);

                rewind(stdin);
                printf("Numero 2: ");
                controllo_due = scanf ("%hd", &b);
                printf("\n");

                if(controllo_uno == 0 || controllo_due == 0) {
                        printf("Perfavore inserisci solo NUMERI!\n\n");
                        rewind(stdin);
                } else {
                        prodotto = a * b;
                        printf ("Il prodotto tra %hd e %hd e': %hd", a, b, prodotto);
                }

        } while (controllo_uno == 0 || controllo_due == 0);

}
```

   Sarebbe opportuno aggiungere un ulteriore controllo sulla grandezza dei numeri in input rispetto alle allocazioni di memoria dichiarate

- **dividi()**
  In questa funzione non viene controllato se nell'input dell'utente sono presenti caratteri, se il denominatore è minore o uguale a zero e infine non è controllata la grandezza del numero inserito in input.

  Se viene inseto uno zero a denominatore il programma riscontrerebbe problemi a livello matematico.

  Se ad esempio vengono inseriti numero troppo grandi per la casella di memoria che abbiamo dichiarato, potrebbe esserci la possibilità che ci sia un Overflow.

  Inoltre anche qui nel caso in cui l'utente sbagli ad inserire l'input, non viene data al possibilità di reinserimento (anche in questo caso risolviamo con un **do while**).

```c
void dividi () {
        float  a,b,divisione;
        int controllo_uno, controllo_due;

        printf ("Inserisci i due numeri da dividere\n(Puoi utilizzare anche numeri decimali):\n");

        do {

                rewind(stdin);
                printf("Numeratore: ");
                controllo_uno = scanf ("%f", &a);

                rewind(stdin);
                printf("Denominatore: ");
                controllo_due = scanf ("%f", &b);
                printf("\n");

                if(controllo_uno == 0 || controllo_due == 0) {
                        printf("Perfavore inserisci solo NUMERI!\n\n");
                } else if (b <= 0){
                        printf("Non puoi dividere per 0 o minore di zero!\n\n");
                } else {
                        divisione = a / b;
                        printf ("La divisione tra %0.2f e %0.2f e': %f", a,b,divisione);
                }

        } while (controllo_uno == 0 || controllo_due == 0 || b <= 0);
}
```

- **ins_string()**
  In questa funzione non c'è un controllo sulla lunghezza della stringa immessa dall'utente.
  Per ovviare a questo basta utilizzare la funzione **fgets()**.

```
void ins_string () {
        char stringa[10];
        printf ("Inserisci la stringa: ");
        rewind(stdin);
        fgets(stringa,10,stdin);

        printf("\nHai inserito la seguente stringa: %s", stringa);
}
```

Nel caso in cui la stringa inserita sia più lunga dell'allocazione di memoria predisposta nella dichiarazione, sarebbe anche opportuno segnalarlo all'utente ed eventualmente dargli una seconda possibilità (con un ciclo **do while)** per inserire una nuova stringa.

Domanda 3: Correzione degli errori logici o di sintassi

```
char scelta = {'\0'};
```

Uno dei primi errori che ho notato è questo, non è esattamente sbagliato, ma generalmente questa notazione viene utilizzata per gli array o stringhe (vettori) per fare in modo che ogni valore sia inizializzato con il valore presente all'interno delle parentesi graffe.

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

In questo caso è sbagliato il %d poiché noi ci aspettiamo una variabile di tipo carattere e non una variabile di tipo integrale

```
switch (scelta)
{
        case 'A':
        moltiplica();
        break;
        case 'B':
        dividi();
        break;
        case 'C':
        ins_string();
        break;
}
```

Non è presente il **default** all'interno dello switch

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

Ho notato che nel codice iniziale le variabili sono dichiarate come tipo **short int** dunque nei rispettivi **scanf** per andare a prendere gli input sono non sono corretti i rispettivi tipi.
Per il tipo **short int** andiamo a sostituire **%f** e **%d** con **%hd**
(ho eseguito stessa cosa ovviamente nel printf sottostante)

```
short int prodotto = a * b;
```

Sempre nella funzione **modifica()** ho notato questo tipo di dichiarazione e pur non essendo completamente sbagliato, ho preferito dichiarare prima la variabile prodotto e successivamente eseguire un prodotto tra le variabili **a** e **b**

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

Nella funzione **dividi()** sono andato a sostituire il tipo **int** nelle variabili dichiara che c'erano prima e ho sostituito con un float, per fare in modo che nella divisione fosse accettata anche una divisione che abbia come risultato un numero decimale (esempio: ½)

```c
int divisione = a % b;
```

Anche qui ho dichiarato la variabile prima e ho sostituito il modulo con "/" la divisione