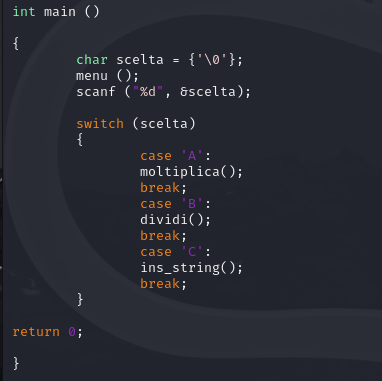
Questo programma scritto in linguaggio C è un assistente digitale che può svolgere 3 funzioni diverse: può moltiplicare due numeri, dividere due numeri e permette anche di inserire una stringa.

FUNZIONE MAIN

Nel costrutto del ciclo switch manca la parte in cui si dice al codice come comportarsi nel caso l’utente inserisca una scelta diversa da quelle proposte (A, B, C).

Inseriamo il **default** che andrà a rappresentare il caso predefinito, cioè quello che viene eseguito se l’utente inserisce qualcosa che non corrisponde a nessun caso.

Il codice però funziona solo se l’utente inserisce in maiuscolo le lettere a, b e c. per accettare in input anche il formato minuscolo delle lettere corrispondenti alle possibili opzioni, possiamo inserire la **funzione toupper**.

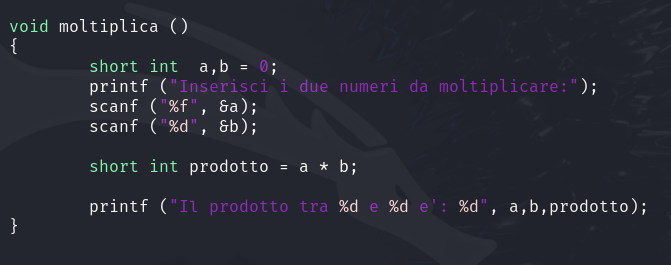
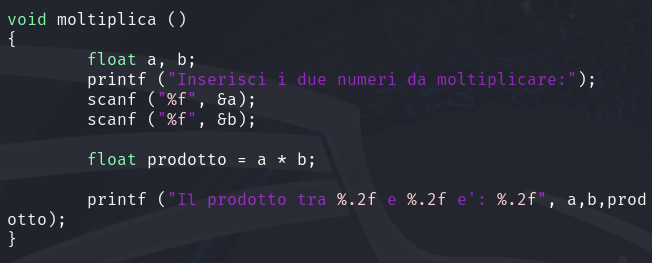
La funzione toupper, contenuta nella libreria ctype andrà a convertire l’input in maiuscolo. Questo serve per far sì che l’input dell’utente sia valido anche se dovesse inserire le lettere a, b o c in minuscolo. In alternativa, avrei potuto aggiungere altri tre casi al costrutto switch in cui si trattano le lettere minuscole: case ‘a’, case’b’, case ‘c’.

La variabile **scelta\_valida** mi serve per trattare le varie scelte ed è una conseguenza dell’inserimento della funzione toupper: se al posto della funzione toupper avessi scritto le varie casistiche, nelle condizioni del ciclo while avrei potuto elencare le varie combinazioni con l’&& con anche le lettere minuscole. Con la funzione toupper non posso farlo perché questa mi converte solo l’input in maiuscolo ma non mi dice che ‘a’ è uguale ad ‘A’

Utilizzo “ **%c**” perché in input andrò a leggere un dato di tipo char e inserisco uno spazio tra le “ e % perché così la scanf eviterà eventuali spazi o ritorni a capo dell’utente. Se l’utente in un inserimento precedente ha poi premuto invio o spazio, nel buffer degli input viene conservato questo input o questo invio e sarà quello che andrà a leggere la scanf come prima cosa, quindi per evitare che legga solo quello, inseriamo uno spazio.

Inserisco il **ciclo do-while** per dare la possibilità all’utente di reinserire una scelta nel caso avesse inserito un input non valido. ! è l’operatore logico not e mi dice che il ciclo si ripeterà finché la variabile scelta\_valida sarà falsa, sarà cioè uguale a 0. Quando scelta\_valida = 1 il programma esce dal ciclo do while.

FUNZIONE MOLTIPLICA

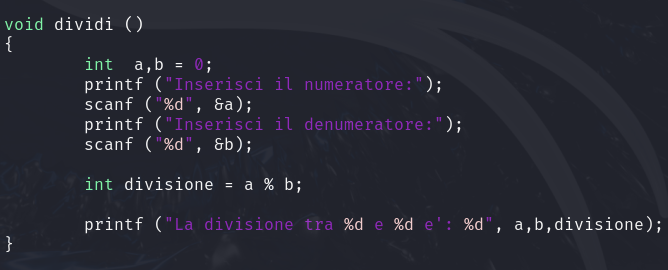
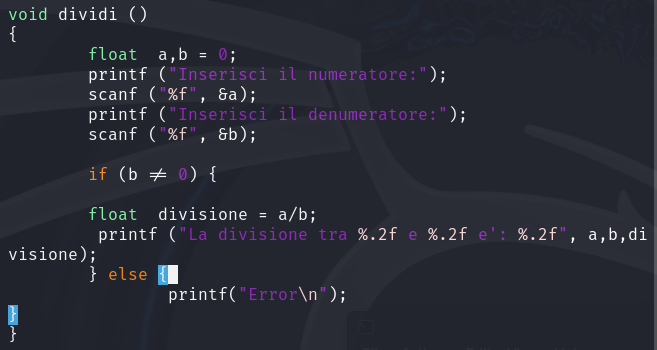
Short int è un tipo di variabile più piccolo di int, può rappresentare un intervallo di numeri più piccolo rispetto al tipo int e questo potrebbe causare un overflow se si inserissero numeri che vanno oltre questo intervallo.

In questo caso però le scanf si aspettano in input un intero o un **float**, quindi dichiariamo le due variabili come interi o float. Se dichiarassi le due variabili come intere, non potrei effettuare una moltiplicazione di numeri reali e per fare un assistente digitale che assista a tutti gli effetti, è giusto che questo sappia fare anche le moltiplicazioni con i numeri reali, quindi dichiaro come float le variabili e modifico la seconda scanf mettendo **“%f**” al posto di “%d”

Lo stesso ragionamento lo facciamo per la variabile prodotto, anche perché dichiarando le due variabili a e b come float, se non dichiarassimo anche la variabile prodotto come float, questa non leggerebbe un possibile numero reale come risultato.

Cambio gli specificatori di formato presenti nella printf da “%d” a **“%.2f**”, così mi stamperà a video solo due cifre dopo la virgola.

FUNZIONE DIVIDI

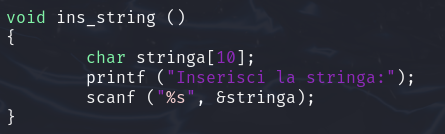
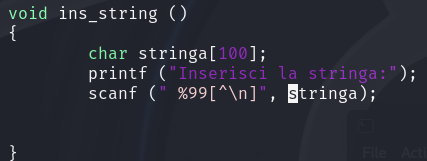
Stesso ragionamento fatto per la funzione moltiplica(). Voglio che il mio assistente sappia svolgere una divisione anche se si tratta di numeri reali.

Con **l’operatore %** si ottiene come risultato il resto della divisione eseguita, non il quoziente. L’operazione di divisione si effettua con l’operatore **/**

Inoltre, sappiamo che un numero non può essere diviso per zero e quindi bisogna trattare il caso in cui l’utente inserisca come divisore il numero 0. Lo facciamo con **un ciclo if- else**: nel caso il divisore (la variabile b) sia diverso da 0, effettua la divisione, altrimenti dà in output un messaggio di errore.

Cambio gli specificatori di formato nella printf da %d a %.2f.

FUNZIONE STRINGA

La stringa ha una dimensione massima di 10 caratteri. Se l’utente inserisse una stringa più lunga di 9 caratteri (perché uno è riservato al terminatore di stringa \0), si verificherebbe un **buffer overflow** andando a sovrascrivere lo spazio di memoria adiacente, perché la scanf andrebbe a scrivere dei caratteri anche oltre l’array di stringa. Per evitare questo, dovremmo assicurarci che la stringa abbia una dimensione sufficiente per contenere la stringa inserita dall’utente. Per fare in modo che l’utente non superi il limite di caratteri potremmo usare una fgets anziché la scanf. In questo caso però è richiesto di inserire solo una stringa e non un testo, quindi possiamo semplicemente aumentare la dimensione massima della stringa.

Aumentiamo la dimensione della stringa fino a 100 caratteri (quindi 99 disponibili) e diciamo che la scanf puà leggere al massimo 99 caratteri (con %99) o fino al raggiungimento di un carattere di nuova riga(^\n).

CODICE COMPLETO

