Status progetto ASM 11/05/2021

Prototipo 2 (`postfix_calculator2.py`)

Fatto

- le funzioni `is_operator()`, `is_operand()`, `is_valid_char()`, `is_valid()` in Python dovrebbero essere facilmente "convertibili" in assembly e si occupano di verificare la validita' delle espressioni in input
- In fondo a `if __name__ == "__main__"` sono stati scritti dei test per queste funzioni

Da fare

- scrivere un algorimo per scorrere gli elementi della stringa in stile C/assembly

  > Ecco l'idea che mi e' venuta:
  >
  > - scorri i caratteri fino a raggiungere uno spazio (o il fine stringa) e memorizza la posizione dopo (per poter ricominciare dall'elemento successivo)
  > - parti dal carattere prima dello spazio e torna indietro carattere per carattere
  > - se trovi una cifra comincia a convertire il numero
  > - se trovi un carattere tra "+", "-", "*", "/", " " o sei a inizio stringa fermati: hai trovato o l'operatore oppure il "-" per il numero negativo oppure hai finito di leggere un numero positivo
  > - se muovendomi all'indietro ho subito trovato quei caratteri speciali ho trovato un operatore
  > - se prima di trovare quei caratteri speciali ci sono cifre ho trovato un operando
  > - se il carattere speciale trovato e' "-" so che l'operando trovato e' negativo

- scrivere un algoritmo per convertire i caratteri degli operandi in numeri (ricordando sempre che "-" davanti al numero indica numero negativo)
- Scrivere le funzioni `is_operator()`, `is_operand()`, `is_valid_char()`, `is_valid()` in assembly
- scrivere in assembly la parte in cui si calcolano le operazioni (la parte "operativa" di `postfix()`)
- scrivere in C i test che sono presenti in fondo al `if __name__ == "__main__"` per testare le funzioni assembly (unit tests)
- scrivere test per testare l'intero programma (end to end tests)

  > Piu' espressioni testiamo e piu' siamo sicuri che il programma funziona correttamente

- ottimizzare l'intero programma

  > sicuramente l'ottimizzazione piu' sostanziale e' usare un ciclo sia per verificare se l'input e' valido sia per effettuare i calcoli invece di usarne due.
  >
  > (Permetterebbe anche di cancellare la funzione `is_valid()`)

Note aggiuntive

- Ho modificato il prototipo in Python per cercare di scrivere la parte del controllo della validita' il piu' vicino possibile a assembly
- Sempre per avvicinare il prototipo al codice finale e' stato cambiato il nome della funzione principale assembly `main_asm()` a `postfix()`
- la funzione `ord()` serve per ottenere il valore corrispondente di un carattere alla tabella ASCII

  > non serve in assembly creare un equivalente

- Prima di ottimizzare e' meglio pensare a come fare il resto.
- Ho creato due funzioni per la verifica completa della validita' (`is_valid()` e `is_valid_char()`) per permettere facilmente il riutilizzo della funzione `is_valid_char()` quando ci occuperemo di ottimizzare il codice (e quindi cancelleremo `is_valid()` che scorre una volta la stringa solo per assicurarsi che i caratteri siano validi)