UNIVERSITA' DEGLI STUDI DI VERONA – A.A. 2019/20
CORSO DI LAUREA IN INFORMATICA

ELABORATO ASSEMBLY – ARCHITETTURA DEGLI ELABORATORI

GIUSEPPE OSENDA - VR422465
ROMAN CICOLIN - VR447097
SEROYIZHKO NAZARIY - VR446033

VARIABILI UTILIZZATE:

- **A, B, C**: etichette per identificare i settori del parcheggio
- **newLine**: etichetta per controllare il fine riga di testin.txt

FUNZIONI IMPLEMENTATE

Nel file parking.c è stata inserita una funzione di tipo extern void chiamata a sua volta parking, la quale passa i vettori bufferin e bufferout_asm come parametri alla funzione assembly. Tale funzione si occupa di gestire, secondo la specifica data, il funzionamento del parcheggio riempiendo opportunamente il vettore bufferout_asm.

CONTENUTO DEL PROGETTO

- **makefile**: permette di velocizzare le funzioni di compilazione, linking e in più l'opzione –gstabs che permette di inserire nel file oggetto, e quindi nell'eseguibile, le informazioni necessarie al debugger (**gdb**).
- **parking-funz.s** (funzione esterna): funzione per la gestione dell'intero parcheggio
- **parking.c**: codice per la gestione degli input/output
- **testin.txt:** file di testo usato dal file parking.c per riempire il vettore bufferin
- **testout.txt**: stampa del bufferout_asm in un file di testo
- **trueout.txt**: file di confronto

FUNZIONAMENTO GENERALE

Il codice parking.c (#C) si occupa di gestire e di riempire il vettore bufferin, mentre il file parking-funz.s (Assembly) si dedica alla gestione e riempimento del vettore bufferout_asm.

Nella fase preliminare, un ipotetico operatore ha il compito di aggiornare **manualmente** il numero di automobili presenti nel parcheggio, il quale rimane libero durante la notte consentendo a chiunque di entrare o uscire. Così la funzione esterna si occupa inizialmente di analizzare il file di prova testin.txt (in particolare le prime tre righe dedicate), verificando innanzi tutto quale settore si presenta per poi riempirlo con il relativo numero di automobili presenti.

Dal momento in cui tale operazione è stata compiuta, il sistema inizia a lavorare **autonomamente**. Si parte, quindi, dalla quarta riga di testin.txt e si riprende l'analisi dei primi caratteri. In base a una determinata sequenza di caratteri, il codice si comporta nel seguente modo:

- Se trova esattamente la sequenza "IN-(A,B,C)" allora significa che si vuole entrare in uno dei tre settori
- Se trova la sequenza "OUT-(A,B,C)" allora significa che si vuole uscire da uno dei tre settori
- In tutti gli altri casi la stringa viene considerata "corrotta" (si veda la parte "scelte progettuali" per un maggior chiarimento) e non più valida, dovendo così passare al controllo della riga successiva (ciò vale anche per i settori A, B, C)

Una volta che si è verificato il tutto, si andrà ad inserire il settore (sempre stando attenti alla sintassi) per poi controllare se questo è pieno o meno. In caso affermativo allora si inizia a riempire il bufferout_asm secondo la sintassi data in trueout.txt tramite implementazione in codice assembly dell'algoritmo itoa, chiudendo di conseguenza la sbarra in entrata, altrimenti si incrementerà il numero di automobili presenti

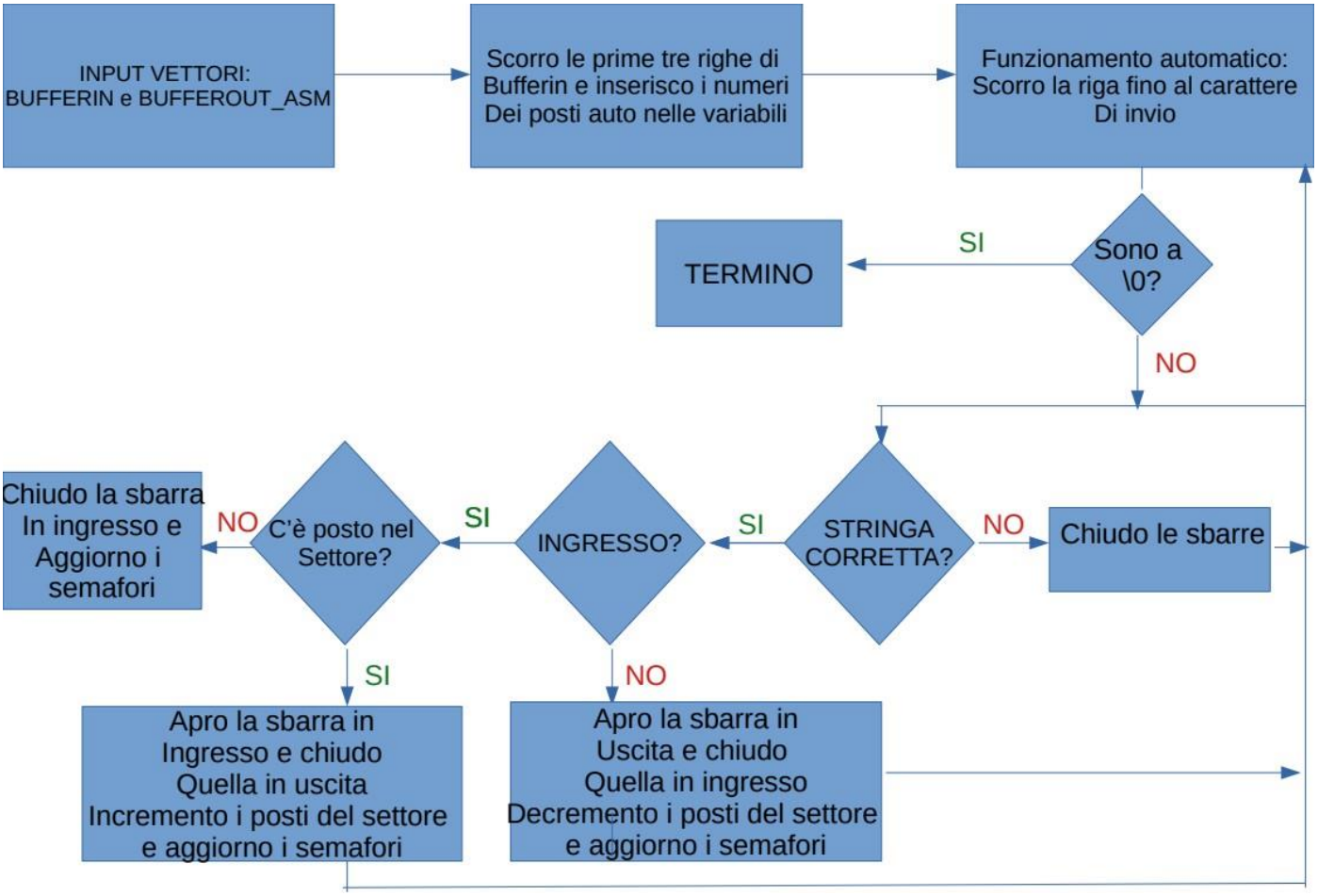
in quel determinato settore, per poi aggiornare correttamente il bufferout_asm e aprire la sbarra in entrata. Stesso discorso per chi esce dal parcheggio.

Dopo aver aperto/chiuso le sbarre in base alle condizioni sopra citate, si aggiorna la terna di valori rappresentanti lo stato di accensione di una luce in corrispondenza a ciascun settore per indicare che è pieno (1) o libero (0).

Tale procedura viene ripetuta fino a quando si confronterà il primo carattere di ogni riga con il valore 0 (NULL – ASCII) per indicare che il programma è finito, uscendo dalla funzione e restituendo il bufferout_asm completo.

Si veda il seguente diagramma di flusso:

DIAGRAMMA DI FLUSSO



SCELTE PROGETTUALI

Per svolgere l'elaborato abbiamo organizzato il programma su piu file. E' stata infatti implementata una funzione assembly esterna chiamata parking-funz.s che si occupa di gestire il funzionamento del parcheggio e il riempimento del vettore bufferout_asm. La funzione in questione è l'unica implementata. Per questo motivo presenta un numero elevato di righe di codice dovuto alla gestione dell'algoritmo di conversione itoa predisposto per convertire i numeri in stringhe. Oltre ad itoa abbiamo personalizzato l'algoritmo atoi basandolo sul risultato della divisione per 10 per capire se i numeri da inserire nelle variabili fossero a una o due cifre. Nel primo caso ci limitiamo a inserire il numero nella variabile mentre nel secondo l'algoritmo si occupa di moltiplicare per 10 la prima cifra e sommandole la seconda. In tal modo si inserisce sempre il numero corretto.

Per una corretta gestione del funzionamento automatico del parcheggio abbiamo deciso di scorrere le stringe carattere per carattere.
In base al confronto tra il carattere atteso e quello effettivamente letto abbiamo predisposto dei salti (jump) opportuni.
In particolare la gestione delle stringhe corrotte ha un funzionamento a parte: una volta rilevata l'anomalia la funzione salta subito all'etichetta predisposta alla gestione dell'errore. Tale etichetta provvede a chiudere le sbarre inserendo "CC-", dove C sta per "Close", all'inizio della stringa e successivamente si limita a completarla inserendo il valore dei parcheggi seguiti dallo stato dei semafori. Il risultato è una stringa sempre uguale alla precedente, se non per il CC all'inizio, finché il programma non rileva un inserimento corretto.
La gestione dei semafori è stata implementata in maniera tale che sia una sola etichetta a gestire l'inserimento di tutti i semafori cosi da ottimizzare al meglio il codice