Documentazione Progetto

Cantini Simone, Kanaan Michelangelo, Marcolan Daniele, Vian Luca

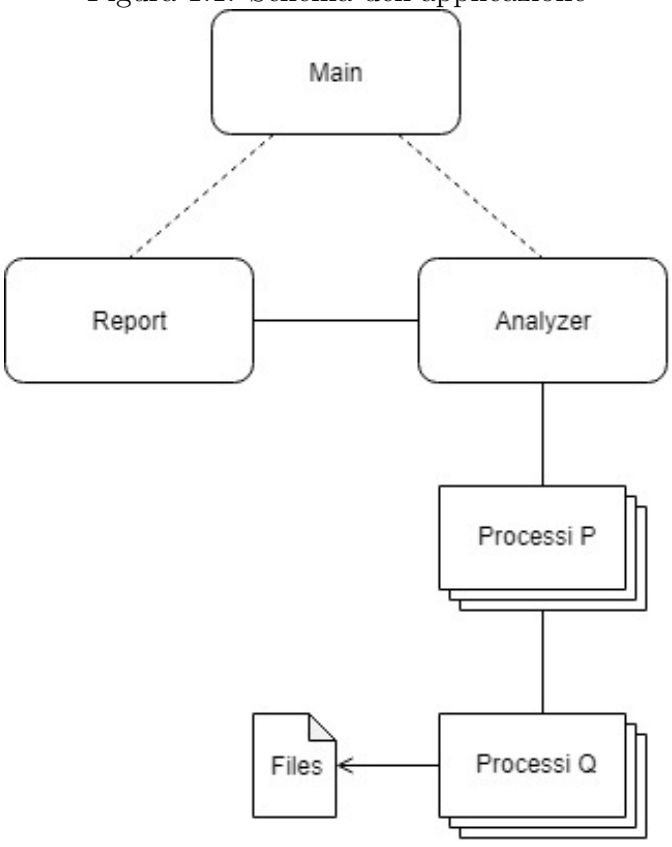
[Data]

Indice

Capitolo 1

Readme

Figura 1.1: Schema dell'applicazione



1.1 Makefile

Avendo come directory di lavoro la cartella dell'applicazione, digitare nel terminale:

- `make help`, per stampare a video delle breve indicazioni per avviare l'applicazione.
- `make build`, per effettuare la compilazione dell'applicazione.
- `make clean`, per resettare la cartella dell'applicazione allo stato iniziale.

1.2 Avviare l'applicazione

Per prima cosa bisogna estrarre dalla zip la cartella dell'applicazione. Dopidiché aprire il terminale, spostare la directory di lavoro sulla cartella estratta e inserire nel terminale il comando `make build`.

L'applicazione può essere utilizzata in due modalità:

- "Main"
- "Analyzer e Report"

1.3 Modalità "Main"

Avendo come directory di lavoro la cartella dell'applicazione, digitare nel terminale `bin/main`. Ora:

1. Viene richiesto il numero di processi P da creare; inserire un valore intero maggiore di 0 e premere invio.
2. Viene richiesto il numero di processi Q da creare; inserire un valore intero maggiore di 0 e premere invio.
3. Viene richiesto l'inserimento dei percorsi ai file o alle cartelle da analizzare; inserire un percorso e premere invio; quest'ultimo procedimento può essere ripetuto finché non viene digitato "`end`" e premuto invio.
4. Viene avviato il processo dell'Analyzer (vedere "Descrizione" nel capitolo "Analyzer") e viene stampato a video un resoconto dell'input e la suddivisione dei files rilevati nei processi P creati.
5. Viene avviato il processo del Report (vedere "Descrizione" nel capitolo "Report") e viene stampato a video un menù che indica le diverse opzioni per visualizzare i risultati dell'analisi sui file (vedere "Funzionalità" nel capitolo "Report" per ulteriori dettagli).
6. Per tornare al Main inserire 0; ora viene stampato a video un menù che presenta diverse opzioni per la modifica dei parametri e l'avvio di nuove esecuzioni (vedere "Modifiche on-the-fly" in questo capitolo).
7. Inserendo un ulteriore 0 è possibile chiudere l'applicazione.

1.4 Modalità "Analyzer e Report"

Aprire due terminali e impostare in entrambi come directory di lavoro la cartella dell'applicazione.

- In uno digitare `bin/analyzer n m path` per avviare il processo dell'Analyzer.
  - `n`: numero di processi P
  - `m`: numero di processi Q
  - `path`: percorso ad un file.txt o a una cartella; è possibile inserire multipli path divisi da uno spazio, es. "`path1 path2 path3`".

- Nell'altro digiatre `bin/report` per avviare il processo del Report.

Se si avvia per primo l'Analyzer, questo esegue l'analisi dei percorsi dati in input e poi attende l'avvio del processo Report. Avviato il processo Report, l'Analyzer gli passa i risultati dell'analisi e si chiude. Se, invece, si avvia prima il Report, questo attende i dati dell'analisi dell'Analyzer.

In entrambi i casi il Report, dopo avere ricevuto i dati dall'Analyzer, mostra un menù che indica le diverse opzioni per visualizzare i risultati dell'analisi sui file (vedere "Funzionalità" nel capitolo "Report" per ulteriori dettagli). Digitando 0 nel terminale del Report è possibile terminale l'esecuzione del Report.

1.5 Modifiche on-the-fly

Il menù della sezione "Modifiche on-the-fly" permette le seguenti azioni:

- Digitando 1 è possibile modificare il numero di processi P inserendo un nuovo valore maggiore di 0; dopodiché viene nuovamente mostrato il menù.
- Digitando 2 è possibile modificare il numero di processi Q inserendo un nuovo valore maggiore di 0; dopodiché viene nuovamente mostrato il menù.
- Digitando 3 è possibile aggiungere ulteriori percorsi da analizzare oltre a quelli già inseriti; la modalità di inserimento è la stessa indicata nel punto 3 della sezione Modalità "Main"; dopodiché viene nuovamente mostrato il menù.
- Digitando 4 viene stampata a video la lista dei percorsi inseriti; dopodiché viene nuovamente mostrato il menù.
- Digitando 5 viene eseguita l'analisi con i parametri attuali.
- Digitando 6 è possibile resettare l'esecuzione del Main.
- Digitando 0 è possibile chiudere l'applicazione.

Capitolo 2

Analyzer

2.1 Funzionalità

Il processo "Analyzer" ha il compito di calcolare le statistiche relative al contenuto dei file di testo passati. Per eseguire il suo compito utilizza dei sottoprocessi P e Q, la cui quantità è specificata dall'utente. Al termine dell'esecuzione "Analyzer" scrive a "Report" i dati calcolati.

2.2 Descrizione

2.2.1 Controllo argomenti passati

L'analyzer appena viene eseguito controlla che sia stato chiamato con gli argomenti opportuni.

Alla riga 7 controlla che il numero di argomenti sia ≥ 4, ossia che gli sia stato passato almeno un valore per `n` (che rappresenta il numero di processi P), uno per `m` (numero dei processi Q) e un file di testo da analizzare. Se `argc` < 4 stampa un messaggio d'errore (dalla riga 8) per poi uscire dal costrutto if-else (arrivando quindi direttamente alla riga 376 e ritornando 0); Se invece il numero di argomenti è corretto inizialmente controlla che `n` e `m` siano valori validi (rispettivamente alle righe 20-39 e 43-62): verifica la validità dell'input usando la funzione `isValidNumber(2)` (che ritorna `true` se il valore passato è una stringa convertibile in intero positivo). Se l'input è valido effettua la conversione, in caso contrario richiede un nuovo valore a video all'utente finché questo non risulta avere valori ammissibili.

Dopo aver controllato e inizializzato `n` e `m` l'analyzer controlla i path che gli sono stati passati (dalla riga 64, vedere sottosezione "Analisi paths").

Terminata la verifica sui parametri viene stampato un loro resoconto minimale alla riga 82. I file di testo vengono elencati se > 0 (da riga 97), dopo essere stati salvati precedentemente in un vettore di stringhe `filesName[x]` (riga 104). E' possibile che non vi siano file di testo da leggere, ad esempio nel caso sia stato passato un file inesistente o non valido. In questo caso stampa un messaggio di feedback per poi continuare normalmente la sua esecuzione (riga 110).

2.2.2 Analisi paths

I paths passati all'applicazione sono analizzati dalla funzione `scanPath(2)`. Tale funzione necessita di due argomenti:

- Un puntatore ad una struttura `FilePaths`.
- Un path rappresentato in una stringa.

Una struttura `FilePaths` contiene un array di paths a singoli file testuali (quindi un array di stringhe) e altri valori per la sua gestione, ovvero, `next` che indica il prossimo indice libero nell'array e `size` che indica la quantità di elementi nell'array. La funzione `constructor` permette di inizializzare la struttura `FilePaths`.

La funzione `scanPath` utilizza la funzione `popen()` per invocare una shell ed eseguire il seguente comando in essa:

```
1 find [path] -type f -exec grep -Iq . {} \; -print
```

dove `[path]` va sostituito con un path da analizzare. Gli elementi di tale comando sono:

- `find`: ci permette di cercare file, con determinate caratteristiche, in un path.
- `-type f`: opzione di `find` che indica di cercare entità che rappresentato un file.
- `-exec grep -Iq . {} \;` : questa riga di comando permette di filtrare i risultati ottenuti da find, mantenendo solamente i file di testo.
- `-print`: opzione di `find` che permette di stampare il path assoluto di un file trovato seguito da un simbolo di newline.

Il comando completo da passare alla shell viene costruito utilizzando la funzione `concat(3)` che concatena 3 stringhe. Nel nostro caso le tre stringhe che vengono concatenate sono le seguenti (nello stesso ordine):

1. `"find "`
2. la stringa `path` passata come argomento a `scanPath(2)`
3. `"-type f -exec grep -Iq . {} \; -print"`

Ogni path ad un file di testo trovato da questo comando viene salvato all'interno della struttura `FilePaths`.

2.2.3 Calcolo suddivisione dei file

Prima di tutto alla riga 119 viene creata una matrice di pipe per permettere la comunicazione tra C e P. Il processo principale (chiamato "C") continua nella sua esecuzione calcolando i file che ogni sottoprocesso "P" dovrà analizzare. Per fare questo crea la variabile intera `remainder` (riga 122), contenente il numero di processi che dovranno lavorare con un documento in più in caso questi non fossero divisibili per `n`, e un vettore `filesPerProcess[n]` contenente effettivamente il numero di file da analizzare per ogni processo. `filesPerProcess[n]` viene popolato dalla riga 129 con valore `x/n` eventualmente incrementato di 1 secondo il valore del `remainder`. Notare come il ciclo for della riga 129 (che cicla `n` volte) non termini subito dopo il popolamento di `filesPerProcess[n]`, bensì alla riga 338. Il codice descritto successivamente verrà quindi eseguito una volta per ogni processo P creato.

2.2.4 Creazione delle strutture necessarie

Sono molte le inizializzazioni precedenti alle chiamate di `fork()` eseguite dal processo principale C.

Sono stati infatti creati e popolati dalla riga 139 alla 161:

1. `filesPerProcessName[filesPerProcess[p]]` contenente il path dei file che dovranno essere analizzati dal processo creato nello stesso ciclo for.
2. `filesPerProcessNumber[filesPerProcess[p]]` contenente una serie di numeri incrementali, ognuno associato ad un path di `filesPerProcessName`.
3. `filesPerP[filesPerProcess[p]][SIZE]` che sarà inizializzato successivamente leggendo dalla pipe tra P e Q. Per ogni processo P, `filesPerP` conterrà le statistiche dei file esaminati dai propri figli.

2.2.5 Primo forking

Il processo C esegue quindi la prima `fork()`. Dalla riga 166 a 317 avremo il comportamento dei figli P. Dalla riga 313 alla 334 continuerà l'esecuzione del padre C;

2.2.6 Comportamento di P

Il figlio P chiude la lettura della pipe con C (riga 169) in quanto tutte le informazioni relative ai file da analizzare sono già presenti nelle variabili `filesPerProcessName` e `filesPerProcessNumber` inizializzate prima della `fork()`. Se P non ha file da analizzare (caso in cui $\texttt{argc} - 3 < \texttt{n}$) verrà stampato a video un messaggio di errore (ora commentato per motivi di visibilità in esecuzione alle righe 188-196). Per ogni processo che avrà almeno un documento da analizzare il resoconto sarà stampato alle righe 172-187. Solo adesso viene creata una matrice di pipe per permettere la comunicazione tra P e Q.

2.2.7 Divisione dei caratteri per i processi Q

Ora il processo P deve dividere in parti uguali (o simili se i caratteri non fossero divisibili per m) ogni documento a lui assegnato, così da farlo analizzare in modo bilanciato ai processi Q. Viene istanziata una matrice di tipo `SPLIT` (riga 204) che conterrà le informazioni di ogni segmento dei file di testo: il path relativo al documento, il suo indice univoco, il carattere di inizio e di fine del frammento oltre alla sua dimensione totale. Il primo indice di `splitMatrix` è usato per indicare il file che è stato diviso in frammenti, il secondo indica il sottoprocesso Q al quale è destinato questo segmento. `splitsMatrix` verrà inizializzata in un for che cicla m volte (alle righe 211-245). La matrice di tipo `SPLIT` viene popolata in modo molto intuitivo, in modalità quasi analoghe al codice che si occupava della suddivisione dei file tra processi P, già analizzato in precedenza.

2.2.8 Funzione `fsize(1)`

Degna di nota è la funzione `fsize` utilizzata alla riga 218. Inizializza la variabile count con la dimensione in byte del file che si sta suddividendo (restituisce quindi il suo numero di caratteri). La funzione si basa sull'utilizzo della funzione `stat(2)` (in "`sys/stat.h`") usata per ritornare delle informazioni del file passato.

2.2.9 Secondo forking

Dopo aver aperto le pipe tra P e Q, ogni processo P genera `m` figli Q. I processi Q creati eseguiranno dalla riga 252 alla 267, il padre P dalla 272 alla 303.

2.2.10 Comportamento di Q

Q chiude il lato di lettura della propria pipe, crea il vettore split che conterrà le informazioni riguardanti il propri frammenti di testo, esamina con la funzione `stats(2)` la propria parte di documento prima di passare i dati calcolati al padre e uscire.

2.2.11 Funzione `stats(2)`

A `stats(2)` viene passato il frammento `SPLIT` da analizzare e un vettore dove poter salvare le statistiche. `stats(2)` apre il file (riga 491), si posiziona al primo carattere a lui destinato con `fseek` e, finché non termina di leggere il segmento passato, incrementa diverse celle dell'array in input a seconda dei caratteri affrontati. La posizione 0 dell'array rappresenta l'indice univoco del file di testo in questione e non viene quindi manipolato. La prima posizione viene incrementata alla lettura di ogni carattere, in quanto rappresenta proprio il numero di caratteri stampabili presenti nel frammento. Le successive 7 posizioni rappresentano cluster vari (in ordine: decimali, lettere in generale, lettere minuscole, lettere maiuscole, punteggiatura, spazi, altro), le 127 successive indicano i singoli valori ASCII.

2.2.12 Torniamo a P

Siamo alla riga 273. I processi P, dopo aver chiuso la stessa pipe dal lato di scrittura, si mettono in attesa che Q gli abbia passato i dati su `pipeMatrixPQ`. Le statistiche passate vengono salvate nel vettore `receivedSplit`. I dati in `receivedSplit` incrementano quelli in `filesPerP[f]` in modo che quest'ultimo array contenga i dati completi di un singolo file una volta ricevuti tutti i conteggi dai figli. Solamente quando `q == m-1` (riga 295) tutte le statistiche sui documenti in questione saranno complete e potranno essere scritte al processo principale C (`write` alla riga 297). P prima di terminare attende che tutti i figli abbiano finito di eseguire.

2.2.13 Torniamo a C

Con la riga 317, ricomincia il codice eseguito da C. C chiude il lato di scrittura di `pipeMatrixCP`, legge le statistiche inviate dal figlio che popoleranno attraverso un doppio ciclo for la matrice `files`, chiuderà la pipe e attenderà i figli (tutto questo tra la riga 318 e 343). Dalla riga 348 alla 358 viene creato il vettore total contenente le statistiche complessive sui file analizzati.

2.2.14 Passagio dei dati a report (Named Pipe)

L'analyzer passerà le statistiche calcolate attraverso una Named Pipe al report. La pipe viene creata dalla funzione `mkfifo(2)` (riga 362) alla quale viene passato il path della fifo stessa("`/tmp/myfifo`") e i suoi permessi ("`0666`" corrispondenti a `rw-rw-rw-`). La fifo viene aperta in sola scrittura passando a `open(2)` la costante `O_WRONLY` (riga 363). Verrà passato a report il numero di file analizzati, le statistiche dei singoli file, le statistiche totali e il nome dei singoli file.

2.3 Osservazioni

Oss. 1

All'inizio dell'esecuzione dell'analyzer vengono fatti diversi controlli sugli argomenti passati. Questi controlli sarebbero superflui se l'eseguibile venisse richiamato dal `main` (che ha già implementato gli stessi controlli sull'input). E' stato scelto di verificare la validità dell'input entrambe le volte per evitare di avere un flag in più nei parametri dell'analyzer ad indicare se fosse stato chiamato dal main o in modalità autonoma (sicuramente fuorviante e poco intuitivo se eseguito manualmente e senza nessun tipo di gateway).

Oss. 2

Il comportamento di `fsize(1)` è stato considerato il più indicato per leggere il numero di caratteri poiché non necessità di aprire il file in questione, comportamento riservato solamente ai figli Q e non direttamente a P.

Oss. 3

Alla riga 503 è stato scelto di calcolare solamente le statistiche riguardanti caratteri stampabili per avere una maggiore chiarezza nella stampa dettagliata del report. Il valore '`\n`', equivalente al backspace (10 in valore ASCII), è l'unico carattere non printable ritenuto importante nel conteggio.

Capitolo 3

Report

3.1 Funzionalità

Il processo "Report" si occupa di stampare a video le statistiche riguardanti i files passati in input dall'Analyzer.

3.2 Descrizione

3.2.1 Passaggio dati (named pipe)

Il Report legge i dati passati dall'Analyzer attraverso una Named Pipe (dalla riga 4 alla riga 19).

La Named Pipe viene creata dalla funzione `mkfifo(2)` (riga 7) alla quale viene passato il path della fifo stessa("`/tmp/myfifo`") e i suoi permessi ("0666" corrispondenti a `rw-rw-rw-`). Report leggerà dalla fifo il numero di files di testo che l'analyzer ha ricevuto in input, le statistiche per ogni file con il rispettivo nome oltre alle statistiche complessive di tutti i files (quest'ultimo passaggio è necessario poiché report non è autorizzato a manipolare i dati. Non può quindi creare il vettore `total` autonomamente). Tutto avviene rispettivamente alle righe 11, 14, 16, e 18. La fifo viene aperta in sola lettura passando a `open(2)` la costante `O_RDONLY` (riga 8).

3.2.2 Visualizzazione delle statistiche

Se il numero di files x passati al report è 0 l'eseguibile mostra un messaggio di feedback per poi, o uscire dall'esecuzione, o attendere nuovi path on-the-fly (modalità main)(dalla riga 51).

Se il numero di files passati è positivo, il report darà diverse opzioni su come visualizzare le statistiche (a partire dalla riga 30) attraverso un menù costruito all'interno di un while (che viene quindi ristampato dopo ogni successiva interazione). Dal menù è possibile:

- uscire dal Report inserendo "0" (invoca un break su quest'ultimo while) (da riga 71).
- richiamare la funzione `resoconto(4)` con dei parametri opportuni (righe 72-83), inserendo un valore da "1" a "4".

- aprire un'interfaccia dalla quale scegliere il file del quale si desidererà il resoconto (da riga 84), inserendo "5".

Se l'input non è compreso tra 0 e 5 l'eseguibile restituirà un'errore (da riga 39).

3.2.3 `resoconto(4)`

In `resoconto(4)` è stato passato come primo argomento l'array del quale si desidera la stampa (viene tendenzialmente passato o l'array total contenente le statistiche generali o l'array riguardante i dati di un singolo file).

Il nome del file passato sarà il secondo argomento (eventualmente uguale a " " quando richiesto resoconto di total poiché la tabelle in output non prevede in questo caso nessuna intestazione).

"`fileFlag`" in 3a posizione sarà uguale a 0 se si desidera un resoconto totale, sarà uguale a 1 se riferito a un singolo file.

L'ultimo valore passato a resoconto sarà "`detailsFlag`" che indica se i dati richiesti saranno dettagliati o meno: se `detailsFlag==1` dopo la stampa dei cluster sarà chiamata la funzione `resocontoDettagliato(3)`, che si occuperà di stampare la frequenza dei singoli caratteri ASCII.

Se `resoconto(4)` avrà a che fare con la stampa dei dati di un singolo file (avrà quindi `fileFlag==1`) stamperà il nome del file in questione. In caso di un resoconto totale creerà un'intestazione a indicare la scelta di avere un riepilogo complessivo (rispettivamente riga 166 e 170). Se il file passato ha almeno un carattere stamperà la tabella di resoconto (da riga 179). In caso contrario avviserà l'utente della presenza del file vuoto (da riga 212). La tabella avrà come prima riga il numero totale dei caratteri (da riga 182); le successive righe in output sono stampate in modo leggermente differente: comprendono infatti il tipo di cluster con la relativa frequenza e percentuale all'interno del file (ciclando da 0 all'`ASCIIOFFSET==9`) (riga 196). A questo punto (riga 204) se il resoconto chiesto sarà dettagliato (`details==1`) verrà chiamata la funzione `resocontoDettagliato(3)`. Altrimenti verrà stampata la chiusura della tabella di resoconto.

3.2.4 `resocontoDettagliato(3)`

La funzione `resocontoDettagliato(3)` riceve in input tre argomenti (analoghi ai primi tre argomenti di `resoconto(4)`). Le statistiche verranno ora mostrate su due colonne (`cols==2` riga 220). La stampa avviene in modo analogo a quella di `resoconto(3)` (ora ciclando da `i==0` a `ASCII`). La gestione delle colonne è presente dalla riga 250.

3.2.5 Gestione input "5" nel menù principale

Se si sceglierà di vedere le statistiche per un singolo file verrà creata una tabella che associa ad ogni numero un documento specifico (da riga 86). L'input dato sarà esaminato e se corrispondente ad un file esistente verrà chiamato il resoconto dettagliato su quel file (riga 111). Il controllo dell'input (righe 108-110) valuta se l'input è un numero decimale attraverso la funzione `isValidNumber()` oppure se è uguale a 0. Viene anche verificato nello stesso `if` se il numero passato indichi effettivamente un file disponibile. Digitando "`end`" o "`list`" si uscirà dal resoconto del file specifico o verrà ristampata l'interfaccia da

dove vedere i singoli documenti (rispettivamente righe 147 e 125). Un messaggio d'errore è mostrato se la stringa in input non sarà corrispondente a nessun file o comando (da riga 140).

3.3 Osservazioni

- Spesso per creare le stringhe rappresentanti valori in ingresso è usato `strcspn()`. E' usato per mettere il carattere "`/0`" al termine di una stringa al posto di "`\n`" letto in input.
- `isValidNumber()` restituisce `false` con interi non positivi, per questo motivo è necessario un ulteriore controllo nella gestione dell'input alla riga 109.

Capitolo 4

Main

4.1 Funzionalità

Il processo "Main" costituisce un "gateway" tra i processi "Analyzer" e "Report". Implementa un'interfaccia grafica e permette di utilizzare l'applicazione tramite un'unico terminale. È "opzionale", nel senso che è possibile utilizzare l'applicazione anche senza il suo utilizzo, ottenendo gli stessi risultati.

4.2 Descrizione

Il processo Main richiede all'utente l'inserimento dei seguenti parametri:

- Il parametro `n`, ovvero il numero di processi P.
- Il parametro `m`, ovvero il numero di processi Q.
- Uno o più path a file o cartelle, fino all'inserimento di `"end"`.

Dopo aver ricevuto i parametri richiesti, richiama la funzione `mainFunction(2)`. Tale funzione riceve in input il numero di elementi `parc` dell'array di stringhe `pargv`, ovvero il secondo argomento della funzione, contenente i paramentri `n`, `m` e i paths.

Procede poi a creare 2 sottoprocessi (utilizzando `fork()`) che eseguiranno parallelamente l'analyzer e il report (tramite l'utilizzo di `execv`). Al termine della loro esecuzione il Main prosegue in background.

Inserendo nel menù del Report il valore 0 si ritorna al Main, che procede quindi a mostrare all'utente una shell interattiva per le modifiche on-the fly dei parametri, le opzioni della shell sono:

1. Modifica il numero di processi P
2. Modifica il numero di processi Q
3. Aggiungi path da analizzare
4. Visualizza i path da analizzare
5. Esegui con i parametri attuali
6. Esegui un nuovo Main

0. Chiusura dell'applicazione

4.3 Osservazioni

Oss. 1

Il Main utilizza la funzione `isValidNumber(2)` per verificare la validitià degli input dell'utente.

Oss. 2

L'input dell'utente avviene tramite degli `fgets` su stringhe di lunghezza massima `MAXINPUT` (1024).

Capitolo 5

Osservazioni generali

- In modalità "Analyzer e Report", dopo che un analyzer ha già inviato i dati al report, l'utilizzo di un nuovo analyzer non ha successo sul report precedente anche se risulta che l'analyzer abbia concluso.