

**RELAZIONE PROGETTO ALGORITMI**

**E**

**STRUTTURE DATI**

**Anno Accademico 2020/2021**

**Sessione d’esame Invernale**

**Studente: Cossi Simone**

**Nr. Matricola 290796**

[**s.cossi2@campus.uniurb.it**](mailto:s.cossi2@campus.uniurb.it)

**Corso tenuto da: Valerio Freschi**

**1) Specifica del problema:**

Si supponga di dover progettare un programma per la gestione dei dati delle CPU di un datacenter.

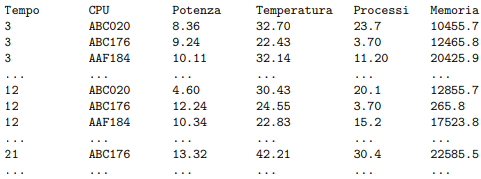
Il sistema di monitoraggio permette di rilevare, una volta all’ora per ogni CPU:

* La potenza richiesta in [W]
* La temperatura del chip in [°C]
* Il numero dei processi attivi
* La quantità di memoria RAM occupata in [MB]

L’intero log viene poi scritto (una volta al giorno) su un file in formato testo, secondo il seguente formato (si assumano campi separati da tabulazione o spazio):

* **Tempo:** un numero intero ∈ [1, 24] che rappresenta l’ora del giorno a cui si riferisce il rilevamento.
* **CPU:** un codice alfanumerico che risulta dalla concatenazione di tre lettere e tre numeri che identificano la singola CPU.
* **Potenza:** un numero reale che rappresenta il consumo di potenza (medio) nell’ora di rilevamento
* **Temperatura:** un numero reale che rappresenta la temperatura (media) nell’ora di rilevamento
* **Processi:** un numero reale che rappresenta il numero (medio) di processi attivi nell’ora di rilevamento
* **Memoria:** un numero reale che rappresenta la quantità (media) di memoria RAM richiesta nell’ora di rilevamento

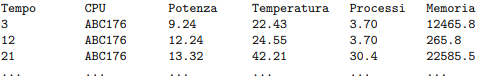
**Esempio**

****

Si scriva un programma ANSI C che esegue le seguenti elaborazioni:

1. Acquisisce il file e memorizza le relative informazioni in una struttura dati di tipo albero.
2. Ricerca e restituisce i dati relativi alle rilevazioni di una data CPU nel giorno. Ad esempio: se l’utente chiede i dati relativi alla CPU ABC176, il programma deve restituire le informazioni contenute nelle righe corrispondenti:

**Esempio**



Il programma deve inoltre prevedere una modalità che implementa le stesse funzionalità utilizzando un array al posto di un albero.

Per quanto riguarda l’analisi teorica si deve fornire la complessità dell’algoritmo di ricerca per la versione basata su albero e per quella basata su array.

Oltre all’analisi teorica della complessità si deve effettuare uno studio sperimentale della stessa per le operazioni di ricerca.

Come suggerimento si può operare generando un numero *N* di rilevazioni casuali (dove *N* rappresenta il numero di CPU monitorate).

L’analisi sperimentale deve quindi valutare la complessità al variare di *N* e confrontare l’algoritmo di ricerca che lavora su albero con il corrispondente che lavora su array.

**2) Analisi del problema:**

**Input:**

L’input è composto da un file che contiene le rilevazioni delle prestazioni per ogni ora di *N* CPU differenti e da alcuni comandi impartiti dall’utente nella fase di generazione del file e nella fase di esecuzione del programma.

I comandi che l’utente può impartire sono:

* Inserire il numero di CPU
* Scegliere tra una struttura ad albero o un array
* Scegliere il nome della CPU di cui si vorranno vedere le rilevazioni
* Scegliere se andare a vedere o meno le rilevazioni di un’altra CPU dopo aver visualizzato quelle di quella precedentemente scelta

**Output:**

L’output è composto dai dati che il generatore andrà a scrivere su file e da delle stampe a video in fase di esecuzione del programma, tra cui:

* Avvisi di eventuali errori di incompatibilità tra le scelte possibili e quelle effettuate dall’utente
* Avvisi di eventuali errori avvenuti durante l’apertura del file contenente i dati
* Indicazioni su cosa e come l’utente deve inserire per interagire col programma
* Stampa di una lista contenente tutti i nomi delle CPU contenute nel file
* Stampa di tutte le informazioni riguardanti la CPU scelta
* Stampa del conteggio dei passi eseguiti dall’algoritmo per ottenere la stampa delle informazioni

**Relazioni input-output:**

Partendo dal generatore, quest’ultimo genera e sovrascrive (se già dovessero essere presenti) dati riguardanti un numero *N* di CPU scelte dall’utente su di un file.

Il programma invece, utilizzando il file, mostrerà all’utente una lista contenente tutti i nomi delle CPU di cui sono state effettuate le rilevazioni, per poi andare a mostrare le informazioni riguardanti una o più CPU scelte.

Per fare ciò utilizza una struttura ad albero o un array.

L’utente è dunque responsabile di scegliere:

* Il numero di CPU di cui fare le rilevazioni e dunque anche il numero di CPU che verranno stampate nella lista al momento di scegliere
* Se andare a utilizzare una struttura ad albero o un array per la memorizzazione delle informazioni prese da file e per la ricerca di quest’ultime.
* Se vedere o meno le rilevazioni di altre CPU

**3) Progettazione dell’algoritmo:**

**Strutture dati:**

Le strutture utilizzate sono quelle previste dalla specifica del progetto:

* **Albero**: le informazioni vengono memorizzate in una struttura ad albero binario di ricerca, quest’ultimo prevede che ogni nodo abbia al massimo due figli e che data una chiave *c* appartenente ad un nodo, ad ogni nodo del suo sottoalbero destro appartiene una chiave di valore ≥ *c*, mentre ad ogni nodo del suo sottoalbero sinistro appartiene una chiave di valore < *c*.

La struttura ad albero utilizzata è ordinata in base ai nomi delle CPU, è consentito, anche se molto improbabile, l’inserimento di una CPU con lo stesso nome di un’altra, quando ciò avviene quella rilevata successivamente viene inserita come figlio destro.

* **Array:** questa struttura è un insieme di elementi disposti linearmente, questo tipo di struttura viene utilizzato sia nel programma principale come richiesto dalla specifica di progetto sia nel programma che genera i dati nel file .txt.

**Scelte di progetto:**

La generazione dei nomi e dei dati di *N* CPU scelte dall’utente è stata affidata ad un programma separato dal resto. Questa scelta è stata effettuata per evitare conflitti tra dati e semplificare il lavoro in fase di sviluppo oltre al fatto di ritrovarsi con un generatore di dati su file separato.

Il resto del progetto è invece contenuto in un altro programma che utilizza lo stesso file del programma descritto precedentemente e utilizza i dati contenuti per delle ricerche.

Ricerche che vengono effettuate o con una struttura ad albero binario di ricerca (scelta per le sue buone prestazioni in fase di inserimento e ricerca) o con un array come previsto dalle specifiche di progetto.

Il programma chiede all’utente di scegliere il tipo di struttura da utilizzare per effettuare le ricerche all’interno del file, la quale non potrà essere cambiata durante l’esecuzione del programma.

Successivamente viene stampata una lista contenente tutti i nomi delle CPU contenuti nel file.

L’utente procede con la scelta di una delle CPU e il programma restituirà tutti i dati riguardanti la CPU scelta.

Inoltre, è stata inserita una funzione che permette all’utente di cercare altre CPU dopo averne già cercate alcune. Questa funzione è stata implementata poiché nonostante ci siano delle validazioni strette sugli input l’utente potrebbe inserire un nome di CPU accettabile ma non presente all’interno del file, in questa situazione l’utente si ritroverebbe senza alcun dato mentre, inserendo questa funzione si da la possibilità di cercare altre CPU senza dover eseguire dall’inizio il programma.

Ad ogni input inserito da tastiera viene effettuata una validazione stretta.

**Algoritmi:**

**Inserimento:** l’algoritmo di inserimento viene utilizzato per l’inserimento dei dati, contenuti nel file, all’interno dell’albero binario di ricerca. Ogni volta che è necessario inserire un nuovo elemento bisogna scorrere tutto l’albero fino ad arrivare ad un nodo foglia. Durante lo scorrimento dell’albero avvengono anche dei confronti tra i dati già inseriti e quello da inserire, poiché se il nuovo elemento ha valore superiore di quello già inserito lo scorrimento sarà vero il sottoalbero destro, viceversa con il sinistro.

Le comparazioni avvengono tramite due funzioni: una verifica se i valori sono uguali mentre l’altra verifica quale dei due è minore.

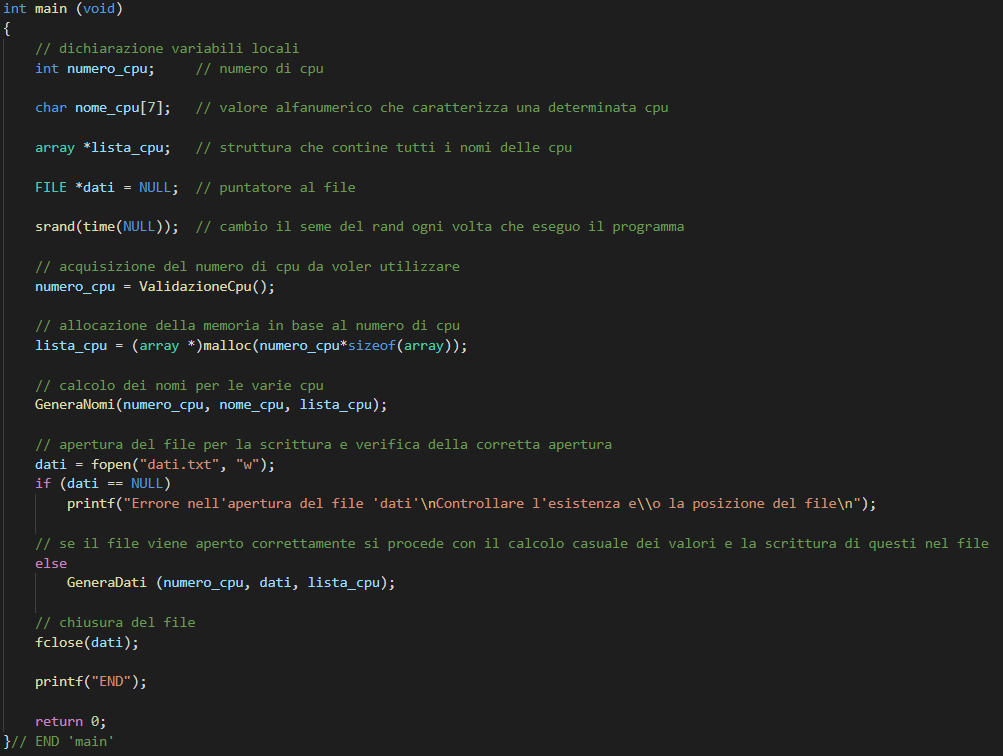
Ovviamente se l’albero dovesse essere vuoto il primo inserimento corrisponderà alla radice.

**Ricerca:** per la ricerca basta partire dalla radice e proseguire verso destra o sinistra in base al risultato del confronto tra il valore scelto e quello della chiave contenuta nel nodo.

Se il valore scelto è maggiore di quello della chiave del nodo si procederà verso destra altrimenti verso sinistra.

**4)** **Implementazione dell’algoritmo**

**GeneratoreFileASD.c**

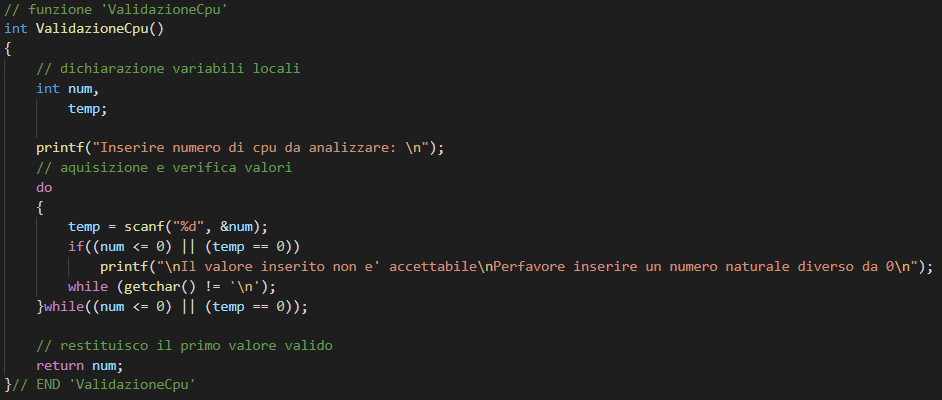
Inizialmente viene cambiato il seme della funzione rand() in modo tale da cambiare ogni volta la generazione randomica dei nomi e dei dati.

Viene richiesto all’utente quante CPU si vogliono analizzare.

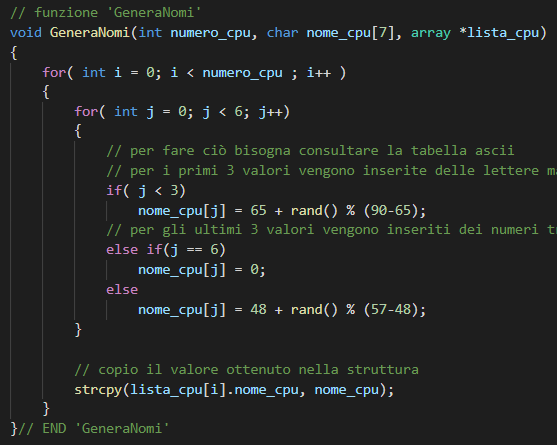
Viene allocata la memoria necessaria per l’acquisizione dei nomi delle CPU che vengono successivamente calcolati randomicamente.

Viene generato/aperto il file in modalità scrittura su cui andranno inseriti tutti i dati.

Vengono generati randomicamente all’interno di un intervallo tutti i dati mancanti delle CPU e vengono così scritte tutte le informazioni su file, alla fine della scrittura si procede con la chiusura del file.

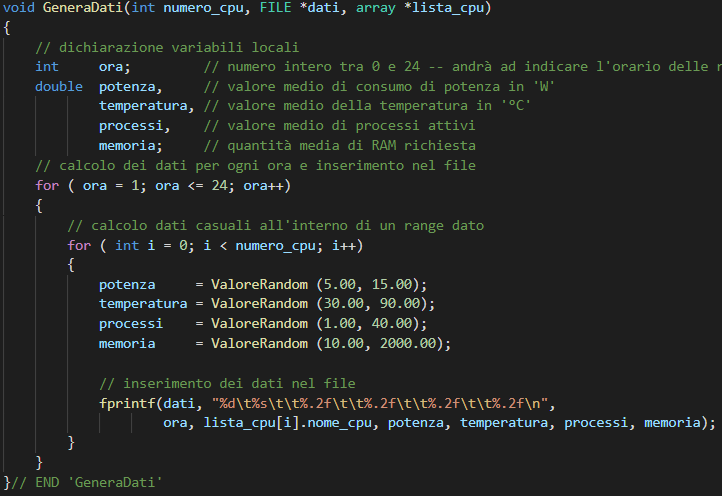
Funzione che chiede all’utente il numero di CPU, acquisisce la risposta e la verifica.

Nel caso l’input inserito dall’utente non dovesse essere compatibile con il tipo richiesto verrà comunicato e effettuata nuovamente la richiesta dell’input.

Funzione che genera randomicamente tutti i nomi delle CPU,

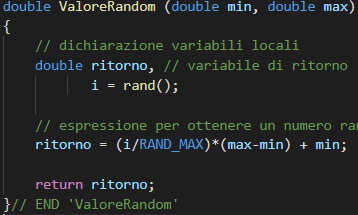
nel formato richiesto dalla specifica.

I nomi generati saranno formati da tre lettere maiuscole e tre numeri.

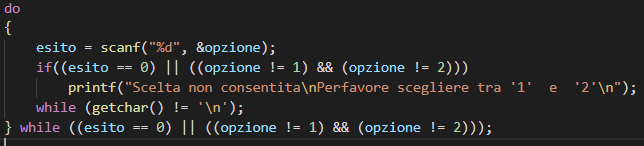
Funzione che genera i valori randomici e scrive su file tutti i dati in ordine:

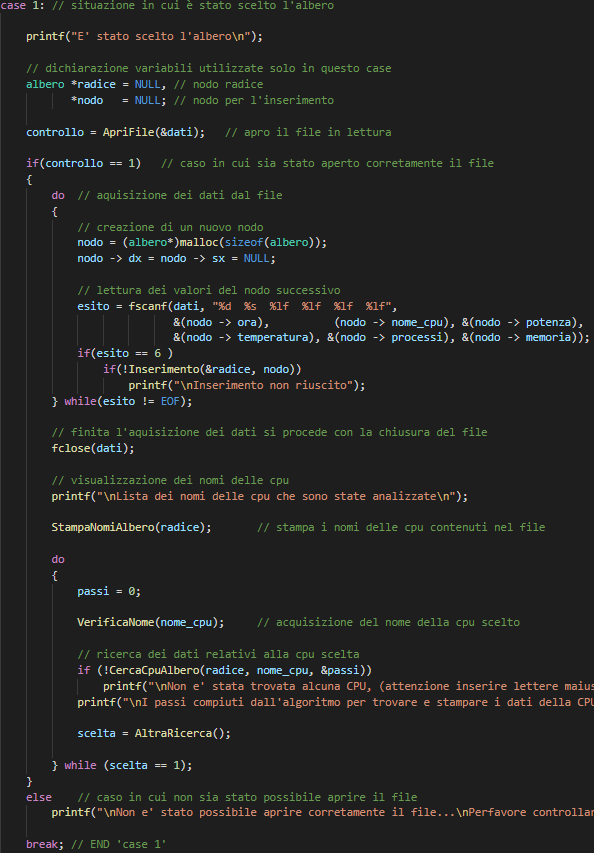
ora, nome, potenza, temperatura, processi e memoria.

Per il calcolo randomico dei valori utilizza un’altra funzione.

Funzione che contiene l’espressione per calcolare i valori randomici delle CPU utilizzando un intervallo.

**ProgettoASD.c**

Acquisizione della scelta tra albero o array da parte dell’utente con validazione stretta.

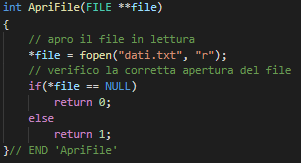
****­Nel caso l’utente dovesse scegliere di eseguire il programma utilizzando la struttura di tipo albero si procede con l’apertura del file in lettura e relativa verifica di corretta apertura.

Nel caso il file sia stato aperto correttamente si procede con l’acquisizione e l’inserimento di tutti i dati nell’albero.

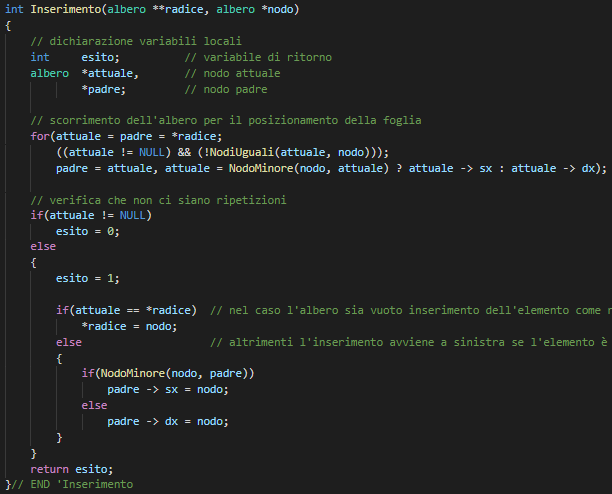
Stampa di tutti i nomi delle CPU.

Scelta della CPU e stampa di tutte le informazioni relative alla CPU scelta con relative verifiche riguardanti il nome.

Si conclude con la richiesta di voler effettuare un’altra ricerca o meno.

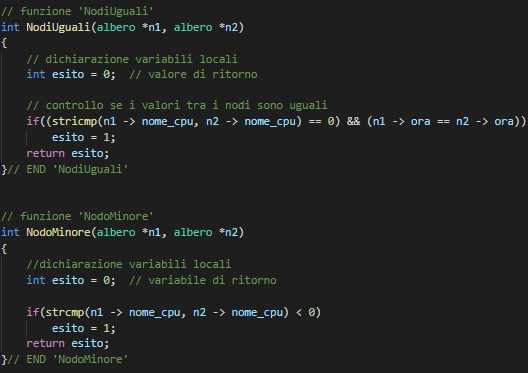
Funzione che verifica la corretta apertura del file.

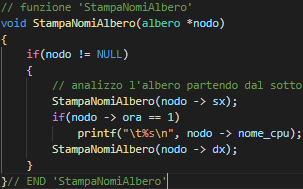
Se il file non dovesse venir aperto correttamente ritornerà un valore che verrà utilizzato nel main per avvertire l’utente di un problema con il file.

Funzione che inserisce i valori letti da file all’interno dell’albero scorrendolo e andando a decidere la posizione dei dati da inserire.

Per decidere la posizione dei dati da inserire si avvale di due funzioni che controllano se il nome della CPU sia minore o maggiore di quello già presente.

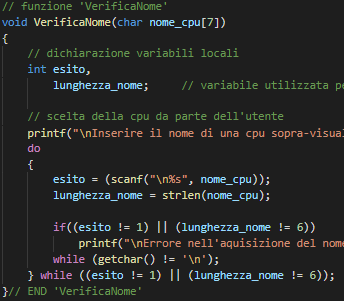
Lo scorrimento avviene come già detto verso destra se dovesse essere maggiore altrimenti verso sinistra.

Funzioni che mi permettono di confrontare due nodi e comunicare quale sia il maggiore o il minore e se dovessero essere uguali.

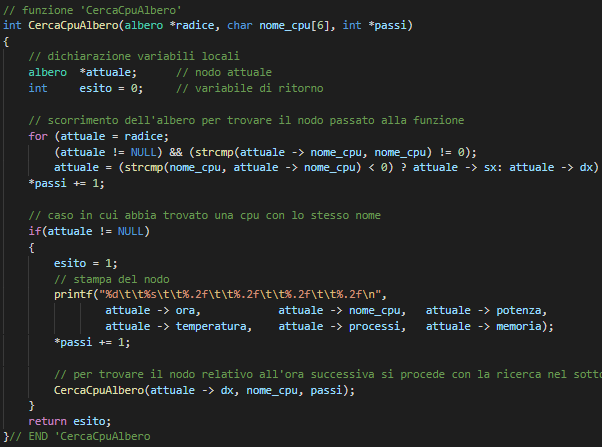


Funzione che Stampa a video tutti i nomi delle CPU utilizzando la struttura ad albero.

Sviluppata in modo da non ripetere nomi già presenti ed è ricorsiva per andare ad analizzare tutti i sottoalberi.

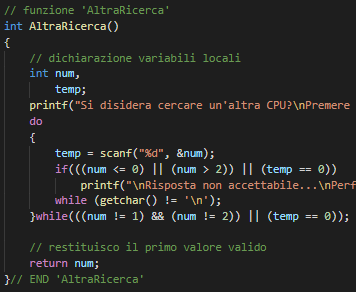
Funzione che chiede all’utente di inserire il nome di una CPU di cui si vogliono vedere le informazioni.

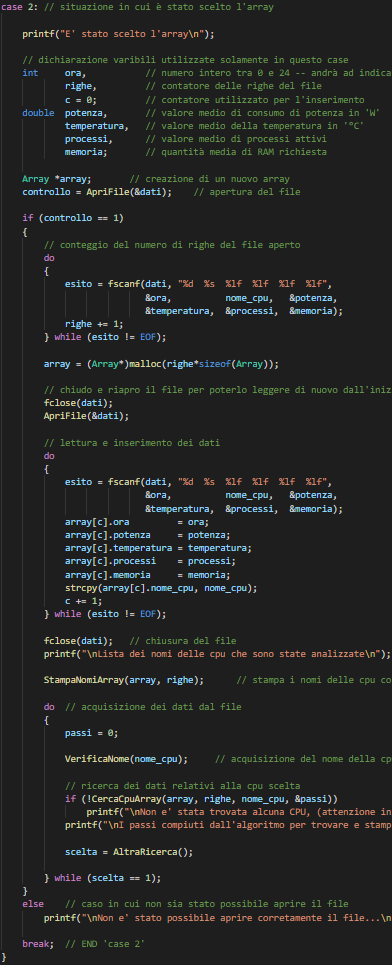
Esegue anche tutti i controlli possibili sul nome.

Funzione ricorsiva che serve per scorrere l’albero finché non viene trovata la CPU richiesta, trovata la CPU ne stampa tutti i valori.

Successivamente richiama se stessa e va ad analizzare il sottoalbero destro poiché per come è stato sviluppato l’albero deve trovarsi per forza lì.

Ha un valore di ritorno perché nel caso non dovesse essere stato inserito un nome presente nella lista lo comunicherà all’utente.

Funzione che acquisisce una scelta tra 2 possibilità con verifica dell’input e comunica la scelta.

Nel caso in cui l’utente scegli di eseguire le ricerche utilizzando l’array come nell’altro caso si procede con l’apertura del file e con il controllare la corretta apertura.

Nel caso in cui venga aperto correttamente si procede con il conteggio delle righe del file per poi andare ad allocare la memoria necessaria.

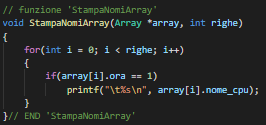
Viene poi chiuso e riaperto il file per ritornare all’inizio del file, qui non viene fatto un controllo aggiuntivo poiché non dovrebbe essere necessario data la corretta apertura precedente.

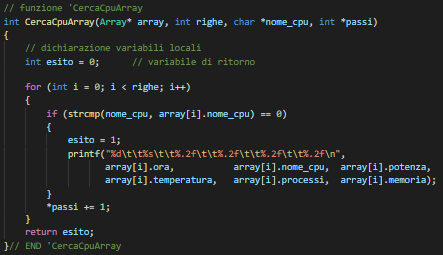
Una volta riaperto si procede con l’acquisizione delle informazioni. A fine acquisizione si chiude il file.

Si esegue la stampa dei nomi delle CPU e con l’acquisizione del nome della CPU da analizzare con tutte le verifiche come nel caso precedente.

Si esegue la ricerca della CPU scelta e la stampa delle informazioni riguardanti quest’ultima.

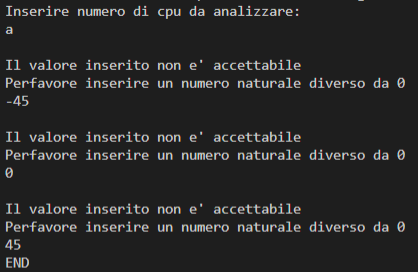
Si conclude come nell’altro caso chiedendo all’utente se vuole effettuare un’altra ricerca.

Funzione che scorre l’array e ne stampa e nomi senza ripeterli.

Funzione che cerca e stampa tutte le informazioni riguardanti la CPU scelta.

**5) Testing del programma:**

**Testing ‘GeneratoreFileASD.c’**

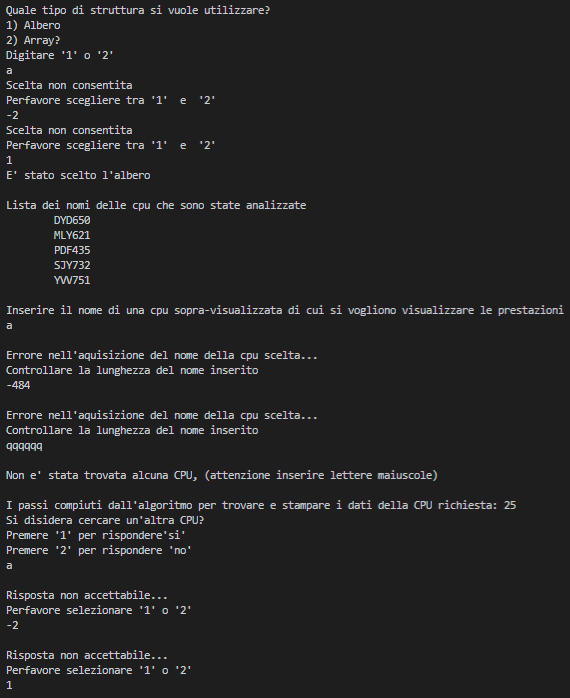


Viene chiesto dal programma quante CPU si vogliono analizzare.

Quando vengono inseriti dei valori non accettabili il programma ce lo comunica e ci chiede nuovamente di inserire dei valori.

Al primo valore accettabile che inseriamo il programma procede con l’esecuzione e termina.

**Testing ‘ProgettoASD.c’**



Inizialmente vien chiesto il tipo di struttura che si vuole utilizzare.

Se si vanno a inserire risposte non accettabili viene comunicato e va inserita nuovamente una risposta.

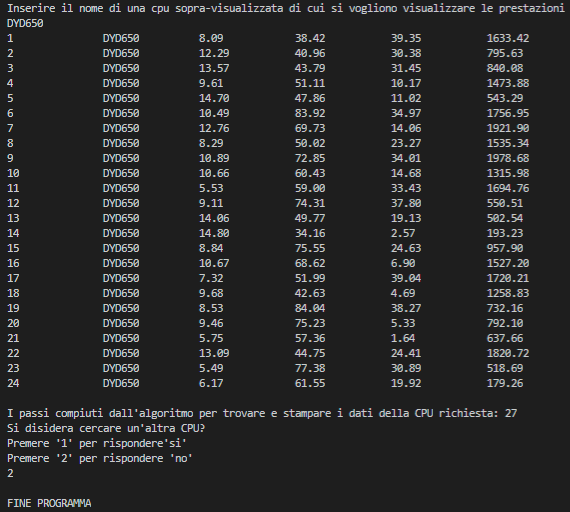
Al primo input accettabile si procede e viene visualizzata una stampa di tutti i nomi delle CPU disponibili.

Anche qui se vengono inseriti input non accettabili ci viene comunicato e va reinserito il nome di una CPU.

Quando inseriamo il nome di una CPU accettabile ma non presente il programma eseguirà le ricerche e non trovando alcuna CPU con il nome richiesto ci verrà comunicato.

Successivamente ci viene chiesto se si vuole effettuare un’altra ricerca e anche qui quando vengono inseriti input non accettabili ci viene comunicato e dovremmo inserire nuovamente un’altra risposta.

Alla prima risposta accettabile il programma procede con l’esecuzione.



All’inserimento di un nome accettabile e presente tra le opzioni il programma eseguirà la ricerca e stamperà a video tutte le informazioni possedute.

Ci verrà richiesto nuovamente se si vorrà effettuare un’altra ricerca, se vogliamo chiudere il programma ci basterà rispondere di non voler cercare altro.

Ci viene anche stampato il numero di passi che l’algoritmo esegue per darci ciò che abbiamo chiesto che ci servirà per analizzare gli algoritmi.

Il funzionamento del programma con l’array e non con l’albero per quanto riguarda l’utilizzo utente è il medesimo.

**6) Valutazione della complessità del programma:**

**Analisi teorica della complessità:**

**Algoritmo di ricerca con albero binario:**

Per trovare la CPU scelta dall’utente viene utilizzata la funzione ricorsiva ‘CercaCpuAlbero’.

Quest’ultima scorre l’albero grazie ad un ciclo, fin quando non trova l’elemento scelto o arriva alla fine dell’albero senza trovarne alcuno.

Quando ciò che cerca viene trovato vengono stampate le informazioni di quel nodo, tuttavia saranno le informazioni riguardanti solo un’ora, perciò la funzione chiama sé stessa.

La complessità di questo algoritmo è:

**In un caso ottimale (albero vuoto) risulta**:

T(𝑛) = 𝑂(1) complessità costante

**In un caso pessimo (albero non vuoto e cerca un elemento non presente, deve perciò essere percorso interamente) risulta:**

T(𝑛) = 𝑂(𝑛) complessità lineare

**Algoritmo di visita dell’array:**

Per trovare la CPU scelta dall’utente viene utilizzata la funzione ‘CercaCpuArray’.

Quest’ultima attraversa gli elementi dell’array linearmente, ovvero nell’ordine in cui sono stati memorizzati, mentre li controlla (verifica se quello che sta cercando e quello che sta controllando siano uguali).

In questa funzione viene utilizzata un’istruzione di selezione *if (𝛽) 𝑆1 else 𝑆2,* la cui complessità è:

N.B. l’istruzione else non è esplicitamente presente ma il funzionamento è il medesimo.

**Caso Ottimo (array nullo, viene subito eseguita l’istruzione *S2*):**

T(𝑛) = 𝑂(1) complessità costante

**Caso pessimo (array non nullo, viene eseguita l’istruzione *S1* fino a scorrere tutto l’array):**

T(𝑛) = 𝑂(𝑛) complessità lineare

**Analisi sperimentale della complessità**

Con l’analisi sperimentale andremo a verificare se quanto detto nell’analisi teorica risulti corretto meno.

In particolare, vedremo l’andamento dei due algoritmi all’aumentare delle CPU e li confronteremo.

|  |  |  |
| --- | --- | --- |
| CPU | ALBERO | ARRAY |
| 10 | 26 | 241 |
| 100 | 27 | 2401 |
| 500 | 28 | 12001 |
| 1000 | 30 | 24001 |
| 2500 | 30 | 60001 |
| 5000 | 33 | 120001 |
| 10000 | 31 | 240001 |