# Indice

1. Traccia

2. Analisi

3. Analisi Preliminare

4. Analisi Integrazione

5. MODIFICHE AD ALTRE STRUTTURE

6. FIlepc.h/.cpp

7. AlberoNarrio.h/.cpp

8. DirectoryFiles

## tRACCIA

Si richiede di:

1. Integrare nel progetto di base denominato “**base2501**” le funzionalità sviluppate da Caropreso Fabiana e Di Serio Federica.
2. Implementare una nuova classe denominata “Computer” al fine di facilitare l'istanziazione di vari computer all'interno del gioco. Tale classe dovrà prevedere l’istanza di un computer localizzato nel luogo “Ufficio”.
3. Verificare eventuali problematiche legate al corretto funzionamento delle funzioni **save** e load.

## aNALISI

### aNALISI pRELIMINARE

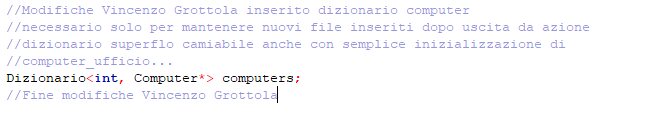
1. Il computer usa una struttura a Lista per la gestione e il salvataggio di file, per quanto semplice e rapida come implementazione, pecca di “profondità”, data l’impossibilità di avere Directory ed una gestione “più file per directory” come nei computer veri e propri. Ritengo che una struttura ad Albero come nei recenti computer sia più realista e dinamica per la gestione di file.
2. Ho notato che i siti web conservati nel computer siano difficili da accedere, in quanto una persona non sa (come nella realtà) quali siti effettivamente siano accessibili/presenti, in questo caso visto che si tratta di un pc simulato la mancanza di una interfaccia chiara che stampa i siti presenti renda tutta la “struttura” Browser quasi inutilizzabile.

### aNALISI INTEGRAZIONE

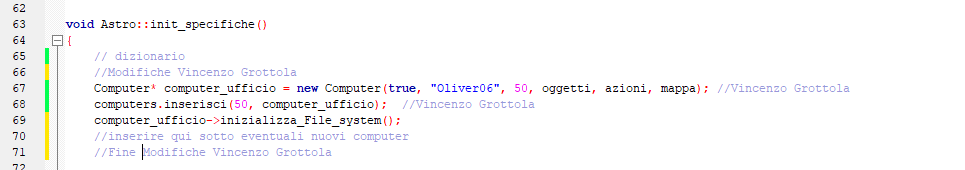
L’integrazione riguarda l’aggiunta, nel luogo “ufficio”, di un nuovo oggetto “computer”, azioni “guarda computer”, “usa computer”, “guarda appunti” e “leggi appunti”. Queste azioni non hanno finalità nel completamento del gioco, ma potranno aumentare o diminuire il tempo a disposizione.

1. L’uso di un dizionario in Astro.h non è assolutamente necessaria per l’utilizzo di computer, basterebbe anche solo creare un puntatore a singolo computer siccome;  
   Tuttavia per maggiore semplicità di future implementazioni ed il possibile “catalogamento” di tutti i computer nel gioco (anche se servirebbe una riscrittura di tutti i computer nel gioco sostituendoli con class Computer) ho deciso di aggiungere classe Dizionario<Computer>.

Astro.h:



Astro.cpp:



1. **Le Funzioni di Computer:**

Nessuna modifica:

* 1. ConfermaUtente;
  2. loginEseguito;
  3. loginFallito;
  4. interagisciComputer;
  5. apriBrowser

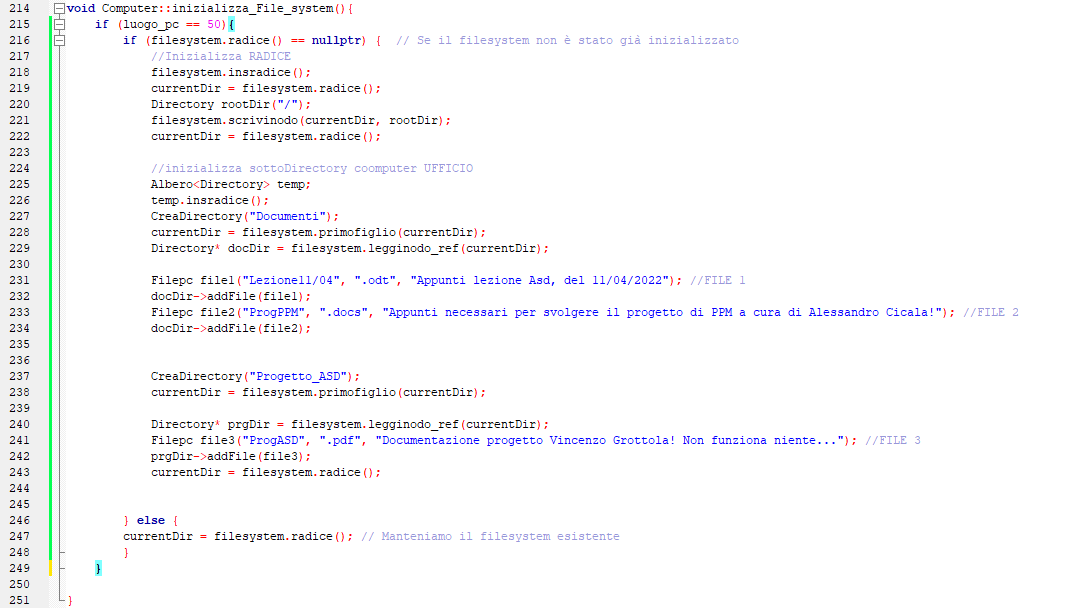
Queste funzioni per l’utilizzo effettivo del computer e della gestione di password/accesso funzionavano già bene e non è stata necessaria nessuna modifica.

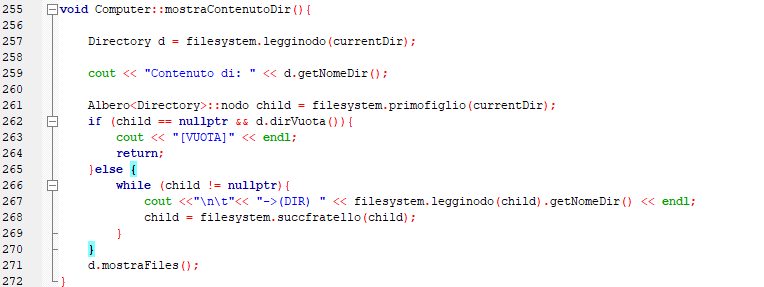
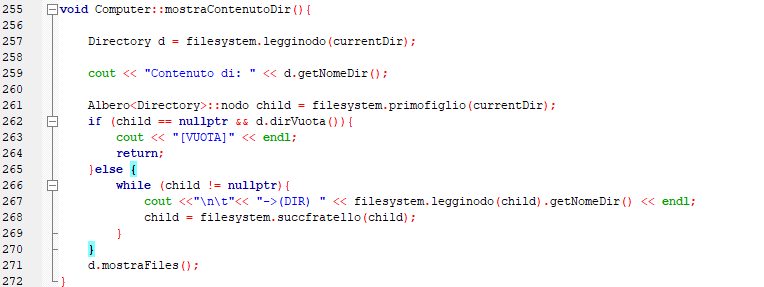
Poche modifiche:

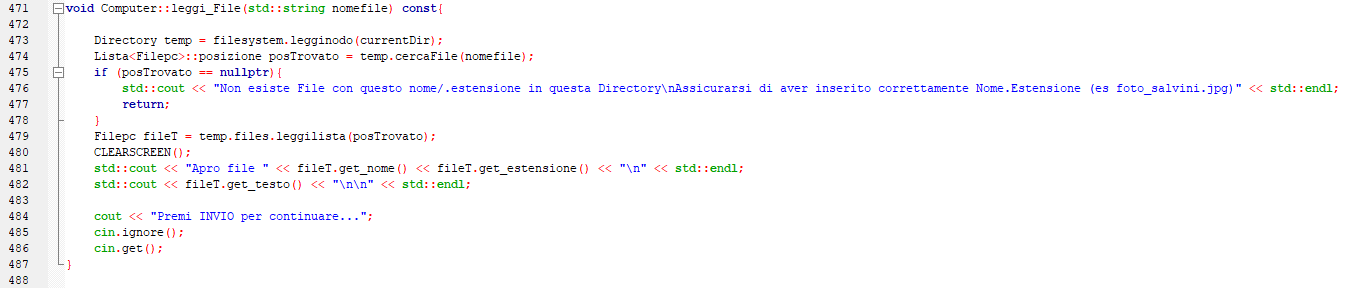
* 1. **inserisciSiti** 🡪 aggiunti altri siti esempio da visualizzare nel Dizionario;
  2. **gestisciBrowser** 🡪 aggiunta funzione mostraPagineWeb() per migliorare leggibilità interfaccia;
  3. **menu** 🡪 modificato case 1: con nuova funzione VUindows() che sostituisce gestioneFile;
  4. **mostraInterfaccia** 🡪 rimossi casi luoghi diversi è possibile aggiungerli in caso si voglia inserire nuoi pc con particolari interfacce
  5. **Costruttore Computer::Computer** 🡪 rimossa inizializzazione struttura Lista<FilePc> (ora l’inizializzazione FONDAMENTALE di filesystem Albero (insradice, etc..) e spostata in inizializzaFileSystem per chiarezza e facilità di lettura e rimossi casi luoghi diversi;

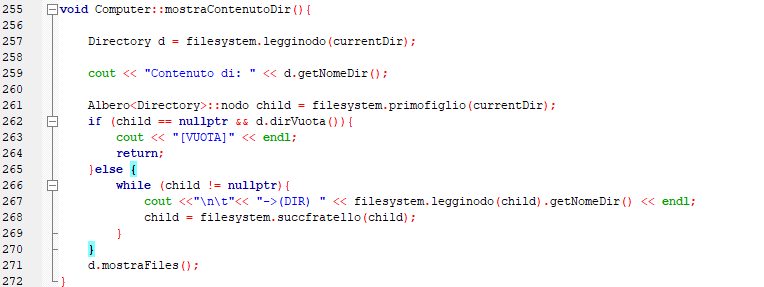
**Funzioni sostituite:**

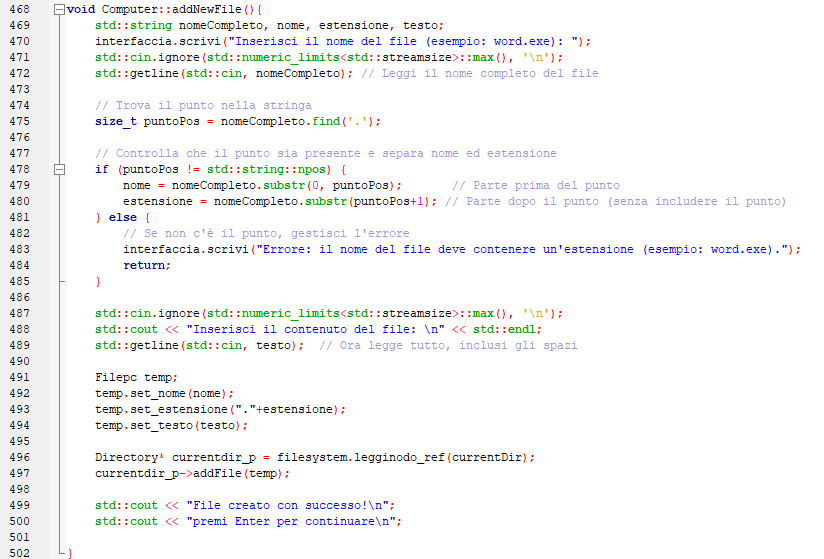
* Sostituito **inizializzaFile** 🡪 con **inizializza\_File\_system**

È stato inserito anche un controllo if (filesystem.radice() == nullptr) per evitare che i file venissero sovrascritti ad ogni accesso

* Sostituito **mostraFile** 🡪 con **mostraContenutoDir**
* Sostituito **rinominaFile** 🡪 con **rinominaFile(nuovo)**



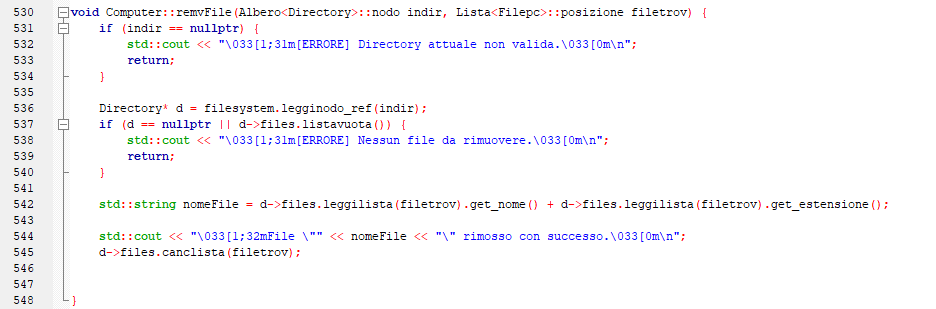
* Sostituito **creaNuovoFile** 🡪 con **addNewFile:** utilizzaun controllo sulla stringa in input simile a creaFile cambia il fatto che ci si riferisce ad una Directory currentDir;

****

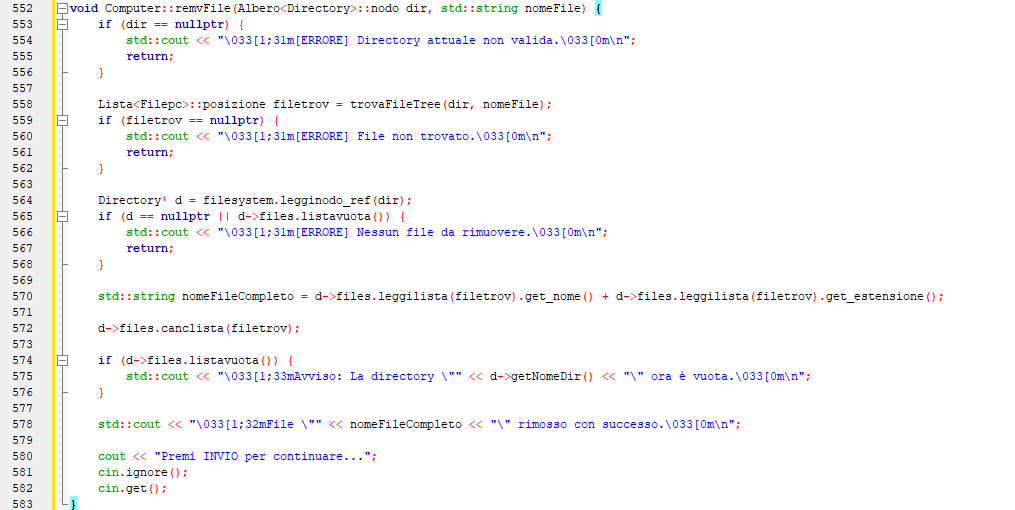
* Sostituito **rimuoviFile** 🡪 con **remvFile:** utilizza la funzione “trovaFileTree” per il passaggio della variabile currentDirTemp di cui ha bisogno.

**(debugging)**: attualmente quando si cerca di rimuovere un file in una directory “figlia” di currentDir si potrebbe andare in contro a crush a causa probabilmente di problemi nel passaggio dell valore di currentDirTemp tra le funzioni, implementando direttamente trovaFileTree in remvFile si risolverebbe il problema a discapito di costo operazionale (quindi ho optato per permettere la rimozione di un file solo se si è in curentDir [modificabile facilmente chiamando trovaFileDir all’interno di remvDir così da evitare problemi nel passaggio del puntatore currentDirTemp]);

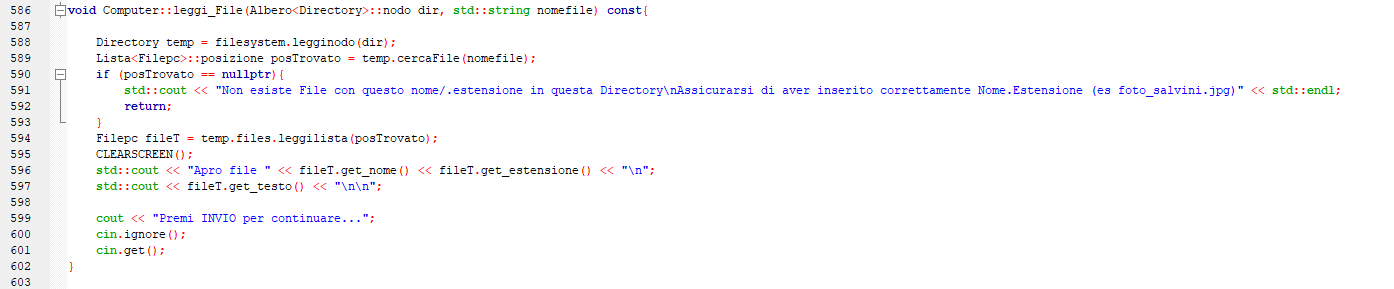
**versione “normale”**:



**Versione con trovaFileTree non testata (solo per dare un’idea):**

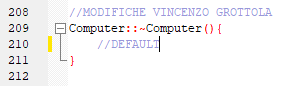
****

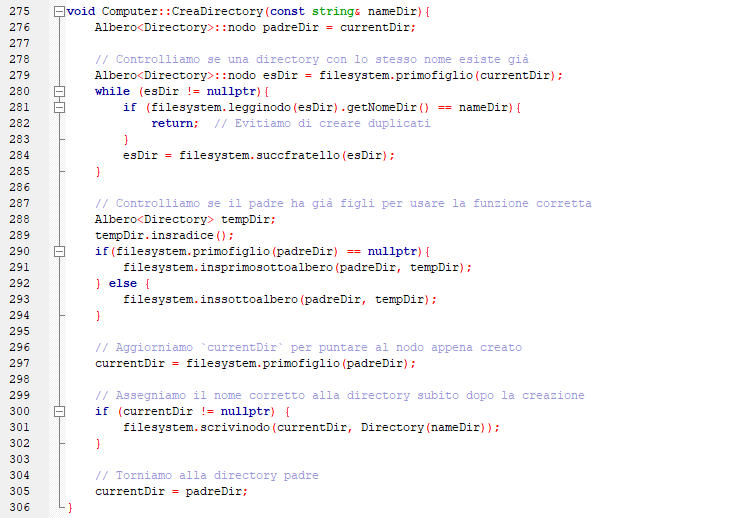
* **Sostituito gestioneFile con VUindows;**
* **Sostituito visualizzaContenutoFile con leggiFile:**

****

**Funzioni nuove inserite:**

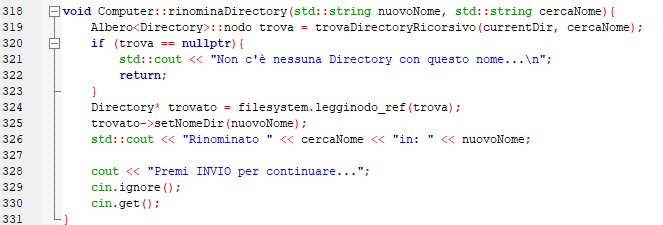
* Distruttore anche se “default” era necessario



* **CreaDirectory**: crea nuova Directory con un nome in currentDir, qui si effetua il controllo per insPRIMOsottoalbero se è presente solo la radice rootDirectory;



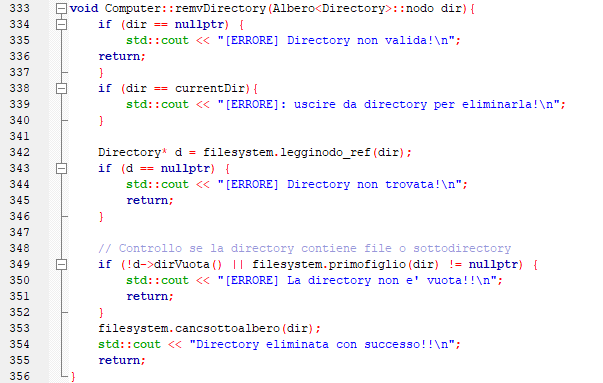
* **rinominaDirectory**: questa funzione sfrutta la funzione ricorsiva: trovaDirectoryRicorsivo per trovare una directory con nome “cercaNome” e cambiarlo con un nuovo nome “nuovoNome”;



* **remvDirectory: rimuove directory, che per regola imposta, solo se VUOTE;**

**questa è una scelta “modificabile”, ho voluto semplicemente rimuovere la possibilità di cancellare un INTERO sotto albero, perché non necessaria alla fine dato il piccolo volume di file e per rendere piu’ solida la struttura e togliere la responsabilità all’utente.**

**(ATTENZIONE):** ovviamente bisogna stare attenti a non eliminare la currentDir del “sistema operativo”, c’è comunque un controllo per evitare che eliminazioni accidentali di currentDir avvengano**.**

****

* **trovaDirectory e trovaDirectoryRicorsivo:** ho scelto di dividere trovaDirectory da trovaDirectoryRicorsivo in 2 funzioni separate per avere un codice piu’ pulito, infatti trovaDirectory serve per interazioni con utente e viene anche richiamata nel menù di scelta;

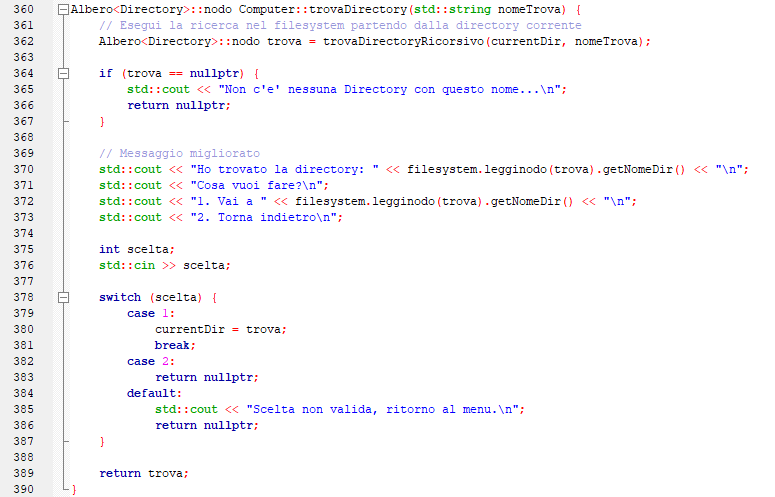
**mentre trovaDirectoryRicorsivo ha funzione di servizio (anche per** trovaDirectory al cui interno viene chiamata),

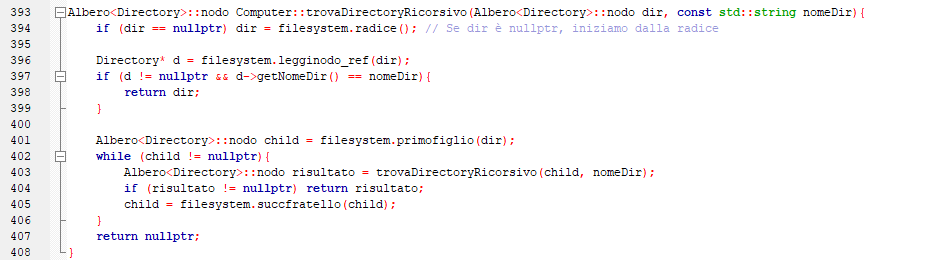
**per “uso di sistema” questa divisione è stata fatta per non “sovraccaricare” un'unica funzione (alla fine il lavoro di divisione è costato ai fini di realizzazione “quindi perché no!”) ;**

* **trovaDirectory() è la funzione pubblica, usata da altre parti del codice.**
* **trovaDirectoryRicorsivo() è la funzione ricorsiva privata, che esegue la ricerca vera e propria.**

**Questa divisione:**

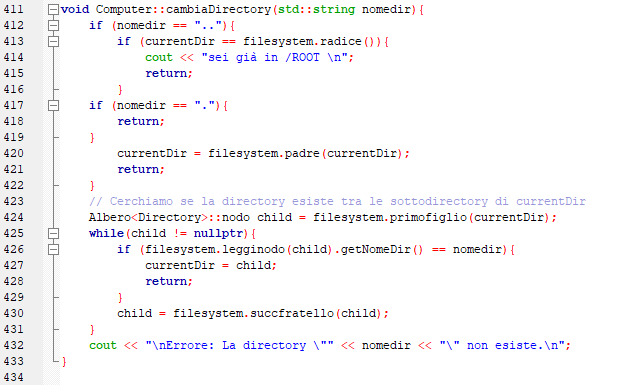
* Mantiene trovaDirectory() pulita e chiara.
* Evita che il chiamante debba sapere che la funzione è ricorsiva.

****

****

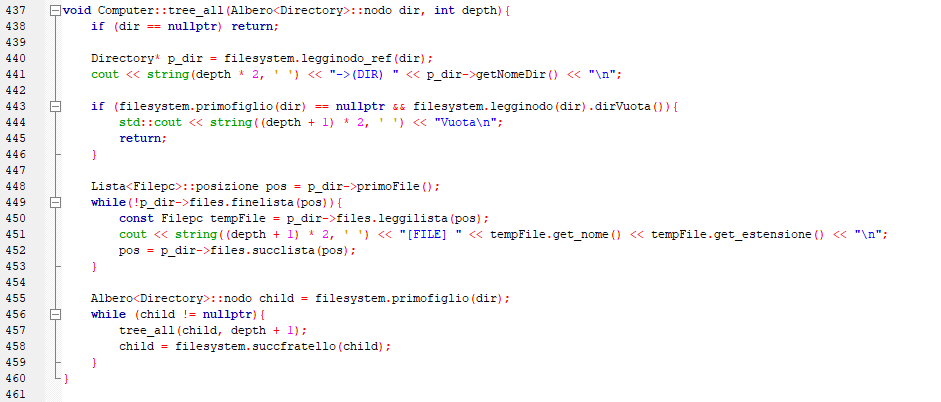
* **cambiaDirectory:** funzione strettamente necessaria per il movimento dell’utente all’interno dell’albero di sistema, per effettuare con piu’ precisione operazioni su file.

**E per comodità ovviamente è implementato anche in trovaDirectory (ho trovato dir… vuoi andarci?) -> cambiaDirectory;**

****

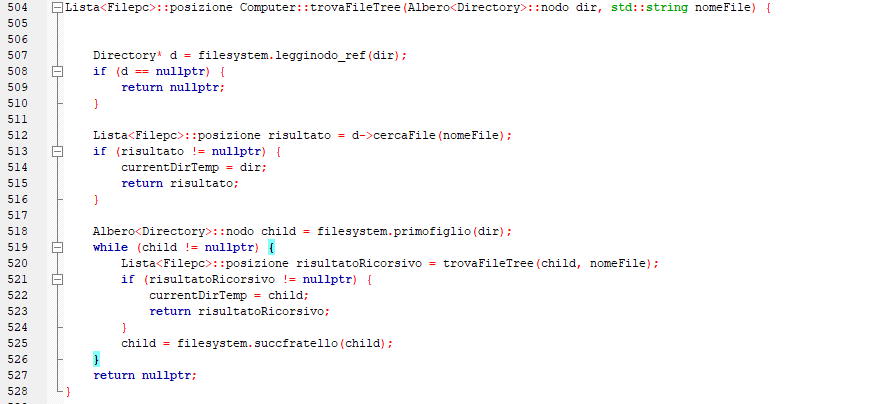
* **tree\_all (spunto da comando per prompt di windows): mostra l’intera ramificazione del file system (albero) in modo gerarchico di tutte le directory e dei loro relativi file.**

**Funziona in modo ricorsivo dato che per ogni directory (se non vuota) richiama se stessa per “andare in profondità” nell’albero.**

****

* **trovaFileTree:** come con trovaDirectory cerchiamo in maniera ricorsiva in ogni directory a partire da currentDir un file con nome e estensione uguali a nomeFile.

Conserva anche in una variabile di servizio del Computer CurrentDirTemp che salverà la directory del file trovato (in debugging).



## MODIFICHE AD ALTRE STRUTTURE

Sono state modificate alcune strutture già presenti in questo progetto ai fini di implementare bene class: Computer

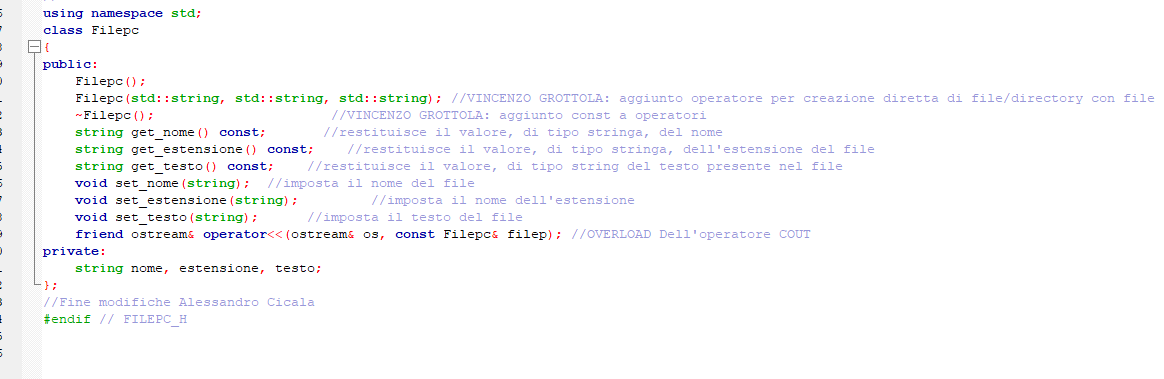
Le modifiche sono riportate di seguito:

### FIlepc.h/.cpp

Aggiunto **const alle funzioni : get\_nome, get\_estensione e get\_testo;**

**queste operazioni di sola lettura DEVONO essere const, per evitare problemi…**

**(relative modifiche anche in .cpp)**

****

### ALBERONARIO.h

Questa modifica è stata una decisione ponderata, resa necessaria da problemi relativi alle directory;

La struttura Albero non permette (giustamente) l’accesso, soprattutto per legginodo (un’operzione di lettura - const), alle posizioni effettive in memoria di T o in questo caso delle Directory;

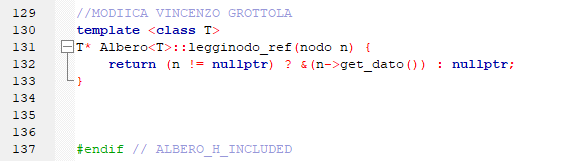
Il fatto che legginodo fornisse alla fine solo “una copia” della directory e quindi modifiche sarebbero non effettuabili (infatti così come dovrebbe essere giusto si dovrebbe usare scrivinodo per effettuare operazioni di scrittura), però, questo sistema, va in contrasto con l’idea di Computer che avevo in mente ( fornendo al “sistema operativo” più’ potere e avvicinarsi il piu’ possibile ad un accesso diretto), inoltre una struttura con piu’ livelli di astrazione come questa a creato non pochi problemi e Bug veri e propri in lettura e scrittura di file e la gestione delle Directory. Quindi ho pensato di alleggerire la struttura passando e lavorando direttamente sui puntatori;

Non ho voluto modificare la “logica” di albero quindi ho semplicemente aggiunto un operatore (minimizzando il più possibile l’impatto di una modifica simile ad una struttura) **T\* legginodo\_ref (nodo n), che non restituisce come output una copia della “directory” ma un puntatore ad essa**;

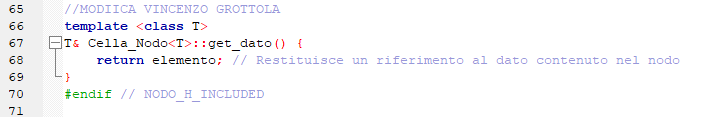
ovviamente come per Albero, è stata aggiunto un altro operatore con stessa **funzione T& get\_dato();**

questo ha permesso la creazioni di funzioni più leggere, piu’ semplici/leggibili, e perfettamente funzionanti, alla fine l’importante è dover gestire BENE questi puntatori per evitare crash o accesso illegali a memoria, e quindi è servita una maggiore attenzione in fase di creazione, ma a mio parere ne è totalmente valsa la pena.

AlberoNario.h

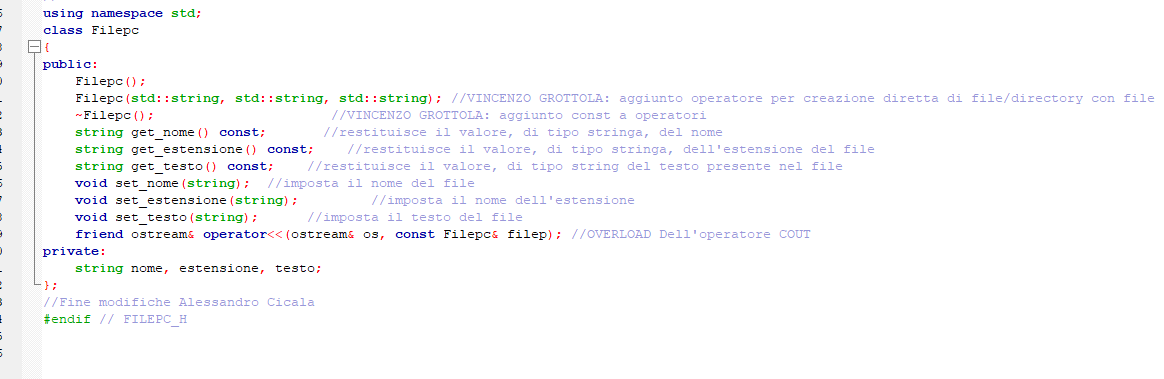


Nodo.h



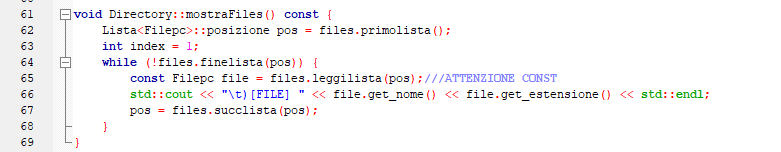
## directoryfiles.h/.cpp

E’ stato necessario anche per poter realizzare la struttura Albero<Directory>, l’aggiunto di un’ulteriore classe al progetto che ci permettesse di conservare altre ad una lista di file Lista<Filepc> anche il nome di una determinata Directory **nomeDir.**

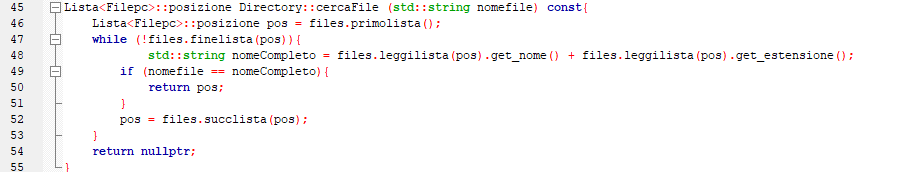
****

**Sfruttiamo anche DirectoryFiles** per contenere anche alcune funzioni più complesse, che verranno molto usate all’interno di Computer, come la stampa dei nomi di tutti i file in una directory**, DirVuota** per sapere se una directory contiene files oppure no **( listavuota )**

**(mostraFile):**

****

E anche di **cercaFile che controlla la lista dato una stringa nomefile come input e che se appartiene alla lista restituisce la sua posizione che viene successivamente passata ad altre funzioni di Computer.**

****