Progetto TennisTournament (Esame di Informatica I)

di Silvio Moioli e Davide Gottini per l'Università degli Studi di Bergamo

Specifiche di progetto

(documento Info01_01)

v. 1.0 del 28 Gennaio 2005

Autore	Matricola	Corso di Laurea
Silvio Moioli	46598	Ingegneria Informatica
Davide Gottini	44659	Ingegneria Informatica

1.0	07-02-03	First issue	SM		
Rev. Index	***Date***	***Revision description***	***Issued by***	***Checked***	***Approved***

DOCUMENT IDENTIFICATION		

Specifiche iniziali

Dal file http://www.unibg.it/dati/corsi/8451/8492-Progetti2005.pdf si legge:
"Programma per gestire un torneo di tennis a eliminazione diretta. Il programma deve gestire un ipotetico torneo di tennis, basato sul principio della eliminazione diretta. Ad ogni turno risultano iscritti alcuni giocatori, ogni partita fa incontrare due giocatori ed uno solo dei due vince la partita; i vincitori di un turno si incontrano al turno successivo, e così via, fino ad arrivare alla finale che proclama il vincitore. Si imposti il programma per non avere limitazioni sul numero dei turni né sul numero dei giocatori (alcuni giocatori possono entrare nel torneo anche in turni successivi al primo); inoltre, il programma deve consentire di ricostruire il percorso di un singolo giocatore, nonché un singolo turno nonché l'intero torneo."

Questo testo si riferisce alla stesura di un programma in linguaggio C++ operante su file e memoria dinamica (ossia che rifletta i contenuti del corso). E' inoltre richiesto:

"La dimensione del programma dovrebbe essere (indicativamente) non superiore alle 1000 linee di codice (compresi i commenti), ma il superamento di tale limite non comporta nessuna penalizzazione. Il programma sorgente e il relativo programma eseguibile devono essere consegnati su un dischetto o CD, insieme ad una relazione."

Nel corso di alcuni colloqui successivi con il docente è inoltre emerso che:
- Non è opportuno che sia utilizzato il paradigma di programmazione ad oggetti per la creazione del programma, poichè l'OOP non fa parte dei contenuti del corso;
- Per la stessa ragione è inopportuno fare uso di alcune delle caratteristiche "avanzate" del linguaggio quali template e overload degli operatori;
- E' consigliabile evitare quanto più è possibile l'uso delle librerie del C: specialmente per le operazioni di I/O si deve fare uso delle nuove librerie del C++ della famiglia *stream;
- Non si può fare uso di nessuna libreria non-standard (per esempio per le interfacce utente) e non si può fare uso della STL;
- E' possibile fare uso di strumenti per la documentazione automatica a partire dai commenti per stendere parte della relazione.

Specifiche ulteriori, stabilite dal gruppo di lavoro

Dopo alcune discussioni in merito alle modalità di realizzazione del progetto, gli autori hanno deciso di imporre le seguenti specifiche aggiuntive:
- Dal momento che operiamo su piattaforme differenti (Davide Gottini lavora in ambiente Windows, Silvio Moioli in Linux), il codice deve essere cross-platform e il progetto deve poter essere compilato ed eseguito correttamente in entrambi i contesti senza modifiche. Questo, fra le altre cose, implica l'impossibilità di usare librerie non-standard (anche se rappresentano "standard de facto" su certa una piattaforma) e l'aderenza del codice agli standard ANSI;
- Si è decisa la seguente politica di implementazione, in base al "limite superabile" delle 1000 righe di codice: dapprima si stenderà il programma con le sole funzionalità di base richieste dal testo, cercando per quanto possibile di non "gonfiare" inutilmente le dimensioni del programma. Se al termine dello sviluppo non si sarà superato il "limite" di molto, si procederà all'implementazione di caratteristiche aggiuntive. In caso contrario, si cercherà di snellire il codice, pur mantenendo come primi obiettivi la funzionalità, la leggibilità e l'usabilità. Nota di fine progetto: in realtà lo sviluppo ha richiesto molto più codice del previsto, per varie ragioni. La prima versione stabile si aggirava attorno alle

3000 righe, e pertanto non abbiamo proceduto nell'implementazione di caratteristiche non richieste. Dopo un attento lavoro di pulizia e riordino, il numero di righe ora è attorno alle 1800, che è ancora molto sopra il “limite”, ma ci sembra comunque ragionevole;
- Dal momento che le dimensioni del progetto non saranno in ogni caso molto grandi, si è deciso di concentrare l'attenzione sulla qualità del codice prodotto. Molte delle scelte tecniche adottate in fase di progettazione e realizzazione (discusse nei prossimi paragrafi) sono state dettate da questa decisione, nonché dalla già citata esigenza di limitare il numero di righe di codice.

Scelte tecniche effettuate durante la progettazione e la realizzazione di TennisTournament

Strumenti utilizzati

Viste le specifiche di cui sopra, e visti gli strumenti utilizzati durante le lezioni di laboratorio, la scelta del compilatore non poteva che ricadere sullo GNU GCC. La disponibilità gratuita, la qualità del compilatore e l'aderenza agli standard ne fanno uno strumento valido per questo genere di progetti. Inoltre è disponibile una buona quantità di IDE che lo integrano insieme agli altri strumenti della GNU toolchain sulle varie piattaforme, tra questi abbiamo scelto il Dev-C++ di Bloodshed (abbiamo usato sia la versione stabile 4 che la beta 5, http://www.bloodshed.net/devcpp.html), Eclipse dell'Eclipse Foundation (http://www.eclipse.org/) gli Eclipse C/C++ Development Tools (CDT, http://www.eclipse.org) e KDevelop (http://www.kdevelop.org/). Nota di fine progetto: TennisTournament compila correttamente in tutti questi ambienti.
Abbiamo utilizzato anche altri strumenti, tra cui meritano particolare attenzione Valgrind (http://valgrind.kde.org/), Artistic Style (http://astyle.sourceforge.net/) e Doxygen (http://www.doxygen.org/). Una descrizione più dettagliata dell'uso di questi strumenti è data nel prossimo paragrafo.

Scelte orientate alla qualità del codice

Come prima cosa, si sono poste fin dal principio le opzioni più restrittive possibili per il compilatore: questo ci ha permesso di scrivere codice migliore e più portabile. In particolare, sono stati attivati gli switch “*-pedantic -pedantic-errors -Wall -Werror -std=c++98 -ansi*” per il GCC. Questo significa che il compilatore si rifiuterà di produrre file oggetto anche in presenza di un solo warning, e sarà particolarmente attento a segnalare costrutti non aderenti allo standard ANSI C++ 98.
Inoltre è stato posto un buon numero di convenzioni di codifica interne per migliorare la leggibilità del codice. Queste includono: un modello di indentazione standardizzato per tutto il progetto (ricontrollato infine con Artistic Style per assicurarne l'uniformità), convenzioni sul nome e l'ordine dei parametri per funzioni simili, convenzioni sui nomi delle funzioni stesse (prefisso in base al file di appartenenza, lettere maiuscole in stile Java), nonché sui nomi delle variabili, suddivisione in librerie, ecc.
Tutti i commenti “esterni” alle funzioni (descrizione dei parametri, dell'uso, delle strutture, ecc.) sono stati uniformati alla sintassi di Doxygen, un programma per generare documentazione di alta qualità automaticamente (in maniera analoga a Javadoc).
Particolare cura è stata inoltre posta nella fase di debug, attraverso più di un procedimento. Per prima cosa, durante l'implementazione delle singole librerie sono stati creati diversi programmi di prova (uno per libreria) per testarne singolarmente il buon funzionamento. Inoltre lungo l'intero programma è stato fatto abbondante uso delle assertions, che

controllavano i parametri di ingresso delle funzioni nonchè il valore di alcune variabili critiche a runtime. Solo quando TennisTournament è stato giudicato completo e stabile è stata creata la versione "Release" del codice, senza assertions e programmi di prova. Inoltre il testing/debugging dell'applicazione è sempre avvenuto sia su Windows che su Linux, in modo da individuare eventuali problemi legati ad una piattaforma specifica.
Altra azione intrapresa è stata quella di testare il funzionamento di TennisTournament sotto Valgrind, un debugger/profiler della memoria dinamica. Questo ha permesso di risolvere bug altrimenti di difficile individuazione quali *double free*, *memory leak*, *out-of-bounds array access*, ecc.
Infine, avendo scelto il formato CSV per file di dati (vedi il prossimo paragrafo), è stato possibile creare facilmente con altri strumenti (quale, ad esempio, OpenOffice.org Calc) file particolarmente pesanti e quindi testare TennisTournament in condizioni critiche.

Scelte riguardanti la gestione dei dati

Memoria di massa

Una delle scelte principali che hanno guidato lo sviluppo di TennisTournament è stata la politica di gestione dei dati su file. La scelta del formato è ricaduta sui file di testo, sia per la semplicità di implementazione tramite gli **fstream*, che per la facilità nel debug (è sicuramente più semplice leggere un file di testo nel proprio editor preferito piuttosto che passare tempo ad interpretare codici esadecimali). In quest'ottica, si è deciso di rappresentare i dati nel formato CSV (Comma Separated Values), per molteplici ragioni fra cui: la veloce implementazione, la possibilità di leggere i CSV anche da applicazioni già esistenti (quali OpenOffice.org Calc o Microsoft Excel) e, appunto, l'immediata leggibilità anche con un semplice *cat*.
Inoltre è stato deciso di effettuare qualsiasi operazione sui dati del Torneo direttamente su file (a parte un piccolo insieme di operazioni intermedie o temporanee). Questo apporta notevoli vantaggi: per esempio non è necessario rappresentare gli stessi dati già residenti su file anche in memoria centrale "risparmiando" ore di sviluppo che sarebbero occorse per mantenere sincronizzate le due tipologie di memoria. Inoltre i dati restano sempre aggiornati sui file, anche immediatamente dopo un'operazione. D'altra parte, vista la natura del problema e il contesto didattico che ha portato alla creazione di questo progetto, sembra evidente che la dicitura "*Si imposti il programma per non avere limitazioni sul numero dei turni né sul numero dei giocatori*" si riferisce al fatto che un'implementazione del tipo "tutto in memoria, salva su file in chiusura" non fosse consigliabile. Mantenendo più dati possibile su file, non abbiamo quasi nessuna limitazione sulla dimensione dei dati stessi.
A questo proposito, l'implementazione fornita nella libreria *file.h* è stata particolarmente curata: non ci sono limitazioni neanche sulla lunghezza del singolo dato, anche se questo ha complicato e reso leggermente meno efficiente l'accesso ai file. L'efficienza infatti non è stata uno dei criteri base del progetto: sapendo di avere a che fare con insiemi di dati tutto sommato sempre piccoli, abbiamo preferito la semplicità del codice e la leggibilità alle performance.
Infine, qualche nota sulla strutturazione dei file: la libreria *file.h* mette a disposizione funzioni ad alto livello per l'accesso "array-like" a un file CSV: è in pratica possibile fare I/O su un file CSV come se si trattasse di una matrice bidimensionale in C++. Gli indici sono numerati da 0 a n-1 se n è il numero di elementi, esattamente come i normali array. In quest'ottica, i file sono strutturati come fossero delle "tabelle", con righe e colonne. Ecco una descrizione sintetica dell'organizzazione dei file in TennisTournament:
1. Esistono due file, associati a due tabelle;

2. Chiamiamo il primo file “giocatori”, che consiste in una tabella con una sola colonna, contenente il nome e cognome del giocatore. Ogni giocatore corrisponde a una riga della tabella;
3. Definiamo l'id del giocatore come la posizione y (riga) che occupa nel file “giocatori”, partendo da 0 fino a n-1;
4. Chiamiamo il secondo file “partite”, che consiste in una tabella con cinque colonne. Ogni partita corrisponde a una riga della tabella. Ognuna delle colonne conterrà un valore intero positivo, indicante nell'ordine: il numero del turno (maggiore o uguale a 1), l'id del primo concorrente, l'id del secondo concorrente, il risultato del primo concorrente e il risultato del secondo concorrente;
5. C'è un'eccezione alla regola 4: l'ultima riga del file partite, anzichè essere una quintupla, potrebbe anche essere solamente un “*”. Questo ci permette di rappresentare il fatto che il turno precedente è stato chiuso, ma non è ancora stata giocata nessuna patita del turno corrente.

Ulteriori indicazioni possono essere trovate nella documentazione HTML fornita insieme a questo file.

Memoria centrale

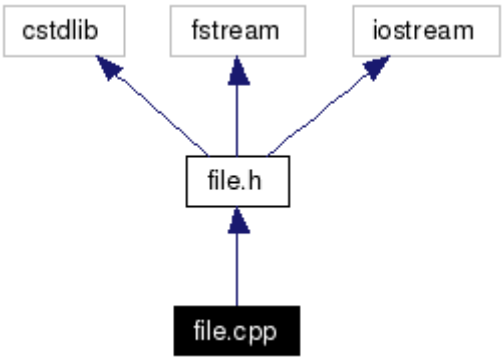
Come già anticipato, le strutture dati in memoria centrale sono state ridotte al minimo indispensabile. L'elemento più importante in questa categoria resta però un'implementazione di una lista (nella libreria *liste.h*). Il problema di fondo era però l'implementazione: fin dalle prime fasi della progettazione è stato chiaro che in qualche occasione sarebbe stato necessario l'uso di liste, anche con tipi di dato differenti. Non potendo usare i template però (e volendo di proposito evitare una libreria di sole macro), restavano due sole opzioni: duplicare il codice delle liste per ogni tipo di dato occorrente oppure tenere traccia solamente dei puntatori agli elementi della lista, senza conoscere nulla del loro contenuto. Abbiamo optato per la seconda soluzione, implementando una lista di *void**. L'unico effetto collaterale della scelta (a parte qualche cast in più durante l'estrazione degli elementi dalla lista) è stata una gestione della memoria dinamica decisamente poco elegante. Infatti, per poter deallocare una lista i cui elementi debbano essere anch'essi deallocati, è necessario “scorrerla” due volte: la prima per chiamare *delete* sui contenuti della lista, e la seconda per distruggere la lista stessa. Non siamo riusciti a pensare a una soluzione migliore (neanche parlando con il docente di laboratorio), perciò abbiamo accettato queste “controindicazioni”.
A parte questo, abbiamo semplicemente definito un altro paio di strutture (che riflettono la strutturazione nei file) per un più agevole passaggio dei parametri tra funzioni.
Per maggiori informazioni sulle strutture o sul funzionamento della libreria *liste.h* si rimanda alla documentazione HTML fornita insieme a questo file.

Suddivisione in sottoprogrammi, funzioni, loro parametri e librerie, sorgenti

Per tutti questi argomenti si raccomanda la visione della documentazione HTML fornita insieme a questo file (la quale grazie alla struttura di ipertesto è più facilmente consultabile). Viene qui fornita una versione ridotta per una panoramica delle funzioni nei file più importanti.

file file.cpp

Grafo delle dipendenze di inclusione per file.cpp:



Funzioni

void fileCrea (const char *nomeFile)
Crea un nuovo file (vuoto).

bool fileVuoto (const char *nomeFile)
Determina se il file è vuoto (lunghezza 0 caratteri).

bool filePrimoCampo (const char *nomeFile)
Controlla se il prossimo campo da aggiungere è il primo del suo record.

int fileContaRecordInt (ifstream *file)
Conta il numero di record all'interno del file (versione interna, presuppone il file già aperto in lettura).

void filePosizionaRecord (ifstream *file, int y)
Posiziona lo stream all'inizio dell y-esimo record.

void filePosizionaCampo (ifstream *file, int x, int y)
Posiziona lo stream all'inizio dell x-esimo campo nell'y-esimo record.

int lunghezzaProssimoCampo (ifstream *file)
Legge e ritorna il numero di caratteri da questa posizione del file alla prossima virgola o fine riga (lunghezza del prossimo campo, se lo stream è posizionato all'inizio del campo stesso).

void fileAggiungiCampo (const char *nomeFile, const char *s)
Aggiungere un campo in coda al file.

void fileAggiungiCampoUltimoChar (const char *nomeFile, const char *s)
Aggiungere un campo in coda al file, sovrascrivendo l'ultimo carattere dell'ultimo record presente.

void fileNuovoRecord (const char *nomeFile)
Chiude il record corrente.

int fileContaRecord (const char *nomeFile)
Conta il numero di record all'interno del file.

bool fileEsiste (const char *nomeFile)
Determina l'esistenza di un file.

char fileUltimoCarattere (const char *nomeFile)

Legge e ritorna l'ultimo carattere in un file, prima dell'eof.

char * fileLeggiCampo (const char *nomeFile, int x, int y)

Legge e ritorna l'x-esimo campo nell'y-esimo record del file in formato stringa.

Documentazione delle funzioni

void fileAggiungiCampo (const char * *nomeFile*,
const char * *s*
)

Aggiungere un campo in coda al file.

Per "campo" si intende una stringa memorizzata in un file CSV.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file a cui aggiungere il campo, completo di percorso.

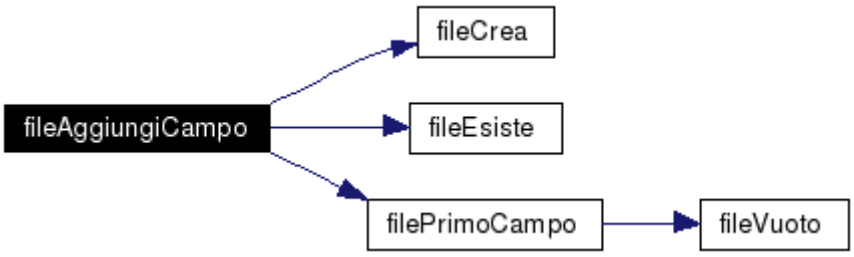
s stringa terminata da ' \0' con il campo da inserire

Definizione alla linea 160 del file file.cpp.

Riferimenti fileCrea(), fileEsiste(), e filePrimoCampo().

Referenziato da fileAggiungiCampoUltimoChar(), giocNuovoGiocatore(), parNuovaPartita(), e parNuovoTurno().

Questo è il grafo delle chiamate per questa funzione:



void fileAggiungiCampoUltimoChar (const char * *nomeFile*,
const char * *s*
)

Aggiungere un campo in coda al file, sovrascrivendo l'ultimo carattere dell'ultimo record presente.

Per "campo" si intende una stringa memorizzata in un file CSV.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file a cui aggiungere il campo, completo di percorso.

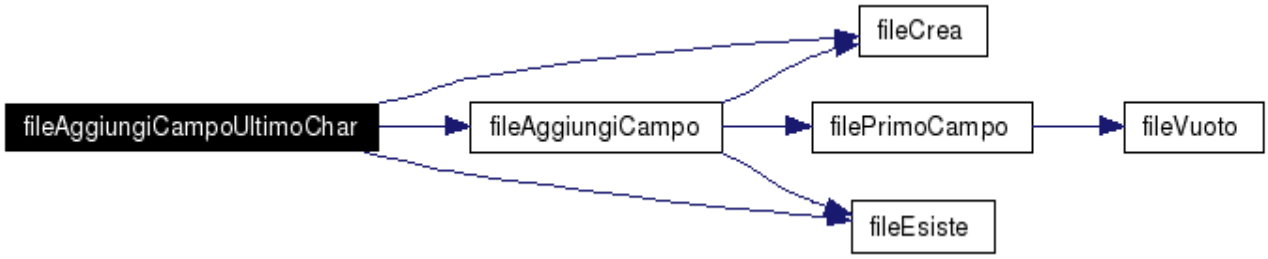
s stringa terminata da ' \0' con il campo da inserire

Definizione alla linea 176 del file file.cpp.

Riferimenti fileAggiungiCampo(), fileCrea(), e fileEsiste().

Referenziato da parNuovaPartita().

Questo è il grafo delle chiamate per questa funzione:



int fileContaRecord (const char * *nomeFile*)

Conta il numero di record all'interno del file.

Per "record" si intende una riga di un file CSV, composta da uno o più "campi".

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file, completo di percorso.

Restituisce:

il numero di record presenti nel file

Definizione alla linea 204 del file file.cpp.

Riferimenti fileContaRecordInt().

Referenziato da giocGiocatoriLiberi(), menuCarrieraGiocatore(), menuGiocatoriTorneo(), menuNuovaPartita(), parContaPartite(), parContaPartiteTurno(), parListaPartiteGiocate(), e parTurnoCorrente().

Questo è il grafo delle chiamate per questa funzione:

int fileContaRecordInt (ifstream * *file*)

Conta il numero di record all'interno del file (versione interna, presuppone il file già aperto in lettura).

Parametri:

file un puntatore a un ifstream già aperto

Restituisce:

il numero di record presenti nel file

Definizione alla linea 70 del file file.cpp.

Referenziato da fileContaRecord().

void fileCrea (const char * *nomeFile*)

Crea un nuovo file (vuoto).

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file da creare, completo di percorso.

Definizione alla linea 11 del file file.cpp.

Referenziato da fileAggiungiCampo(), fileAggiungiCampoUltimoChar(), e fileNuovoRecord().

bool fileEsiste (const char * *nomeFile*)

Determina l'esistenza di un file.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file da controllare, completo di percorso.

Restituisce:

true se il file esiste, false altrimenti.

Definizione alla linea 212 del file file.cpp.

Referenziato da fileAggiungiCampo(), fileAggiungiCampoUltimoChar(), fileNuovoRecord(), giocGiocatoriLiberi(), menuCarrieraGiocatore(), menuGiocatoriTorneo(), menuNuovaPartita(), parContaPartite(), parContaPartiteTurno(), parNuovaPartita(), e parTurnoCorrente().

char* fileLeggiCampo (const char * *nomeFile*,
int *x*,
int *y*
)

Legge e ritorna l'x-esimo campo nell'y-esimo record del file in formato stringa.

Note: le coordinate x ed y funzionano come negli array multidimensionali in C, sono numerate quindi da 0 a n-1, dove n è il numero degli elementi. Per "campo" si intende una stringa memorizzata in un file CSV. Per "record" si intende una riga di un file CSV, composta da uno o più "campi".

Parametri:

nomeFile	stringa terminata da ' \0' con il nome del file da cui leggere il campo, completo di percorso.
x	l'indice del campo
y	l'indice del record

Restituisce:

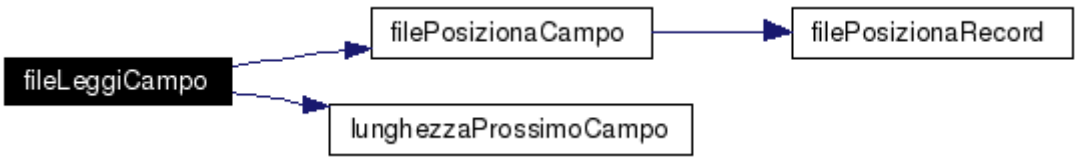
una stringa appena allocata contenente il dato letto

Definizione alla linea 233 del file file.cpp.

Riferimenti filePosizionaCampo(), e lunghezzaProssimoCampo().

Referenziato da giocDettagliGiocatore(), giocNomeGiocatore(), parContaPartite(), parContaPartiteTurno(), parDettagliPartita(), parListaPartiteGiocate(), e parTurnoCorrente().

Questo è il grafo delle chiamate per questa funzione:



void fileNuovoRecord (const char * *nomeFile*)

Chiude il record corrente.

La prossima operazione di fileAggiungiCampo avverrà su un nuovo record. Per "campo" si intende una stringa memorizzata in un file CSV. Per "record" si intende una riga di un file CSV, composta da uno o più "campi".

Parametri:

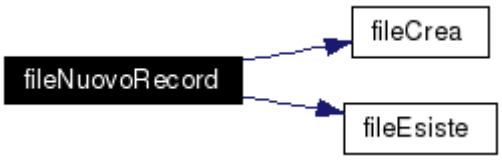
nomeFile	stringa terminata da ' \0' con il nome del file a cui aggiungere il record, completo di percorso.

Definizione alla linea 193 del file file.cpp.

Riferimenti fileCrea(), e fileEsiste().

Referenziato da giocNuovoGiocatore(), e parNuovaPartita().

Questo è il grafo delle chiamate per questa funzione:



void filePosizionaCampo (ifstream * *file*,
int *x*,
int *y*
)

Posiziona lo stream all'inizio dell x-esimo campo nell'y-esimo record.

Note: le coordinate x ed y funzionano come negli array multidimensionali in C, sono numerate quindi da 0 a n-1, dove n è il numero degli elementi. Per "campo" si intende una stringa memorizzata in un file CSV. Per "record" si intende una riga di un file CSV, composta da uno o più "campi".

Parametri:

file un puntatore a un ifstream già aperto
x l'indice del campo
y l'indice del record

Restituisce:

il numero di byte dall'inizio del file al primo carattere dell'x-esimo campo nell'y-esimo record

Definizione alla linea 124 del file file.cpp.

Riferimenti filePosizionaRecord().

Referenziato da fileLeggiCampo().
Questo è il grafo delle chiamate per questa funzione:

void filePosizionaRecord (ifstream * *file*,
int *y*
)

Posiziona lo stream all'inizio dell y-esimo record.

Note: le coordinate x ed y funzionano come negli array multidimensionali in C, sono numerate quindi da 0 a n-1, dove n è il numero degli elementi. Per "campo" si intende una stringa memorizzata in un file CSV. Per "record" si intende una riga di un file CSV, composta da uno o più "campi".

Parametri:

file un puntatore a un ifstream già aperto

y l'indice del record

Definizione alla linea 98 del file file.cpp.

Referenziato da filePosizionaCampo().

bool filePrimoCampo (const char * *nomeFile*)

Controlla se il prossimo campo da aggiungere è il primo del suo record.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file da controllare, completo di percorso.

Restituisce:

true se il prossimo campo aggiunto sarà il primo del suo record, false altrimenti.

Definizione alla linea 46 del file file.cpp.

Riferimenti fileVuoto().

Referenziato da fileAggiungiCampo().

Questo è il grafo delle chiamate per questa funzione:

char fileUltimoCarattere(const char * *nomeFile*)

Legge e ritorna l'ultimo carattere in un file, prima dell'eof.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file, completo di percorso.

Restituisce:

l'ultimo carattere nel file

Definizione alla linea 224 del file file.cpp.

Referenziato da parNuovaPartita(), e parTurnoCorrente().

bool fileVuoto(const char * *nomeFile*)

Determina se il file è vuoto (lunghezza 0 caratteri).

Parametri:

nomeFile stringa terminata da '\0' con il nome del file da controllare, completo di percorso.

Restituisce:

true se il file è vuoto, false altrimenti

Definizione alla linea 28 del file file.cpp.

Referenziato da filePrimoCampo().

int lunghezzaProssimoCampo (ifstream * *file*)

Legge e ritorna il numero di caratteri da questa posizione del file alla prossima virgola o fine riga (lunghezza del prossimo campo, se lo stream è posizionato all'inizio del campo stesso).

Parametri:

file un puntatore a un ifstream già aperto

Restituisce:

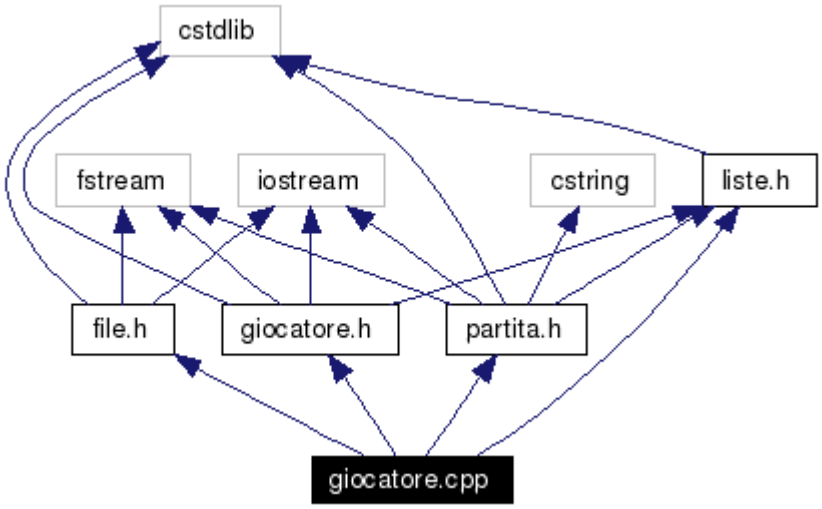
la lunghezza del prossimo campo (in caratteri)

Definizione alla linea 144 del file file.cpp.

Referenziato da fileLeggiCampo().

File giocatore.cpp

Grafo delle dipendenze di inclusione per giocatore.cpp:



Funzioni

Giocatore * giocDettagliGiocatore (const char *nomeFile, int id)
Ritorna una struttura che descrive il giocatore identificato dall'id specificato.

int giocStatoGiocatore (const char *nomeFile, int id)
Ritorna una delle costanti LIBERO, ELIMINATO o GIOCATO che descrivono lo stato di un giocatore nel Torneo.

Lista * giocGiocatoriLiberi (const char *nomeFileGiocatori, const char *nomeFilePartite)
Ritorna una lista nella quale ogni elemento è una struttura giocatore con i dettagli relativi a un giocatore libero.

char * giocNomeGiocatore (const char *nomeFile, int idGiocatore)
Ritorna il nome del giocatore avente l'id specificato.

void giocNuovoGiocatore (const char *nomeFile, char *nomeGiocatore)
Inserisce un nuovo giocatore nel Torneo.

int giocHaGiocato (const char *nomeFileGiocatori, const char *nomeFilePartite, int idGiocatore, int idPartita)
Fornisce informazioni sul ruolo di un giocatore in una data partita.

Documentazione delle funzioni

Giocatore* giocDettagliGiocatore (const char * *nomeFile*,
int *id*
)

Ritorna una struttura che descrive il giocatore identificato dall'id specificato.

Parametri:

nomeFile stringa terminata da '\0' con il nome del file dei giocatori, completo di percorso

id l'id del giocatore

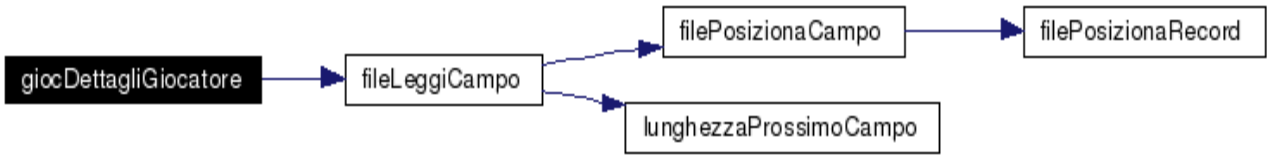
Restituisce:

una struttura Giocatore con i dettagli

Definizione alla linea 12 del file giocatore.cpp.

Riferimenti fileLeggiCampo(), Giocatore::id, e Giocatore::nome.

Referenziato da giocGiocatoriLiberi().
Questo è il grafo delle chiamate per questa funzione:



Lista* giocGiocatoriLiberi(const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Ritorna una lista nella quale ogni elemento è una struttura giocatore con i dettagli relativi a un giocatore libero.

Parametri:

nomeFileGiocatori stringa terminata da '\0' con il nome del file dei giocatori, completo di percorso.

nomeFilePartite stringa terminata da '\0' con il nome del file delle partite, completo di percorso.

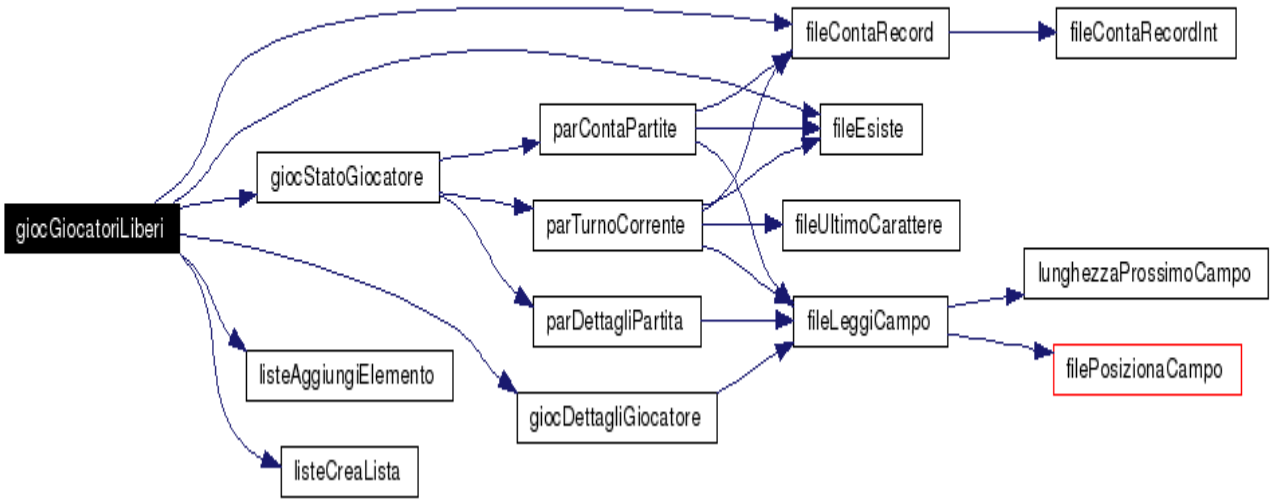
Restituisce:

una Lista con i dettagli di tutti i giocatori liberi nel Torneo.

Definizione alla linea 51 del file giocatore.cpp.

Riferimenti fileContaRecord(), fileEsiste(), giocDettagliGiocatore(), giocStatoGiocatore(), LIBERO, listeAggiungiElemento(), e listeCreaLista().

Referenziato da menuCambiaGiocatore(), e menuNuovaPartita().
Questo è il grafo delle chiamate per questa funzione:



int giocHaGiocato (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
int *idGiocatore*,
int *idPartita*
)

Fornisce informazioni sul ruolo di un giocatore in una data partita.

Ritorna una delle costanti simboliche HAVINTO, HAPERSO o NONGIOCA a seconda se il giocatore risulti vincitore, perdente o non abbia preso parte a una partita.

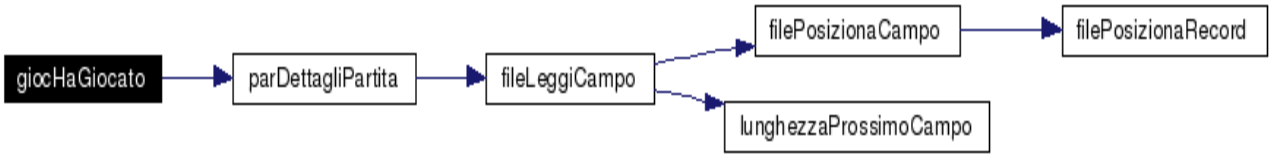
Parametri:

nomeFileGiocatori	stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da ' \0' con il nome del file delle partite, completo di percorso
idGiocatore	il giocatore di cui si vuole conoscere il ruolo
idPartita	la partita da esaminare

Definizione alla linea 82 del file giocatore.cpp.

Riferimenti HAPERSO, HAVINTO, Partita::idG1, Partita::idG2, NONGIOCA, parDettagliPartita(), Partita::risG1, e Partita::risG2.

Referenziato da menuVediCarrieraGiocatore().
Questo è il grafo delle chiamate per questa funzione:



char* giocNomeGiocatore (const char * *nomeFile*,
int *idGiocatore*
)

Ritorna il nome del giocatore avente l'id specificato.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso.

idGiocatore id del giocatore

Restituisce:

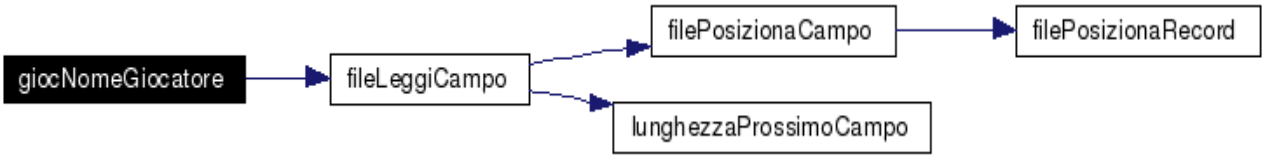
il nome del giocatore, in formato stringa

Definizione alla linea 73 del file giocatore.cpp.

Riferimenti fileLeggiCampo().

Referenziato da menuCarrieraGiocatore(), menuGiocatoriTorneo(), menuNuovaPartitaMostra(), menuPartiteTurno(), e menuVediCarrieraGiocatore().

Questo è il grafo delle chiamate per questa funzione:



void giocNuovoGiocatore (const char * *nomeFile*,
char * *nomeGiocatore*
)

Inserisce un nuovo giocatore nel Torneo.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso.

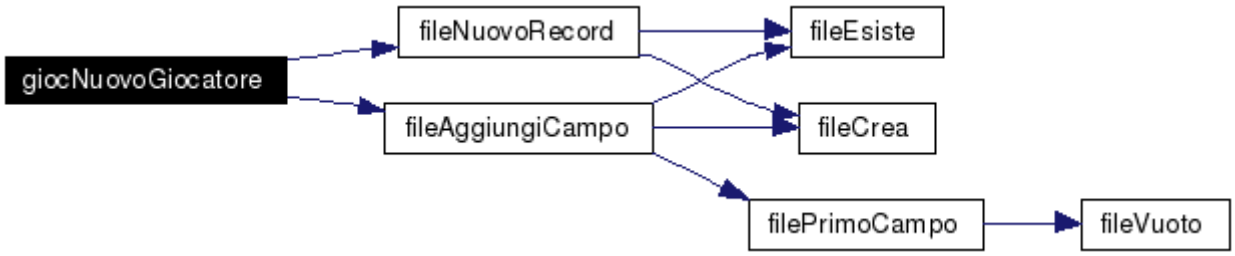
nomeGiocatore il nome del giocatore da inserire

Definizione alla linea 77 del file giocatore.cpp.

Riferimenti fileAggiungiCampo(), e fileNuovoRecord().

Referenziato da menuAggiungiGiocatore().

Questo è il grafo delle chiamate per questa funzione:



int giocStatoGiocatore (const char * *nomeFile*,
int *id*
)

Ritorna una delle costanti LIBERO, ELIMINATO o GIOCATO che descrivono lo stato di un giocatore nel Torneo.

In particolare:

- lo stato LIBERO corrisponde a un giocatore che ha la possibilità di giocare in questo turno o in turni successivi (non è mai stato eliminato e non ha ancora giocato in questo turno);
- lo stato ELIMINATO indica che il giocatore è già stato battuto in un turno precedente a questo;
- lo stato GIOCATO indica che il giocatore ha preso parte a una partita nel turno corrente.

Parametri:

nomeFile stringa terminata da '\0' con il nome del file delle partite, completo di percorso.

id il numero identificativo del giocatore

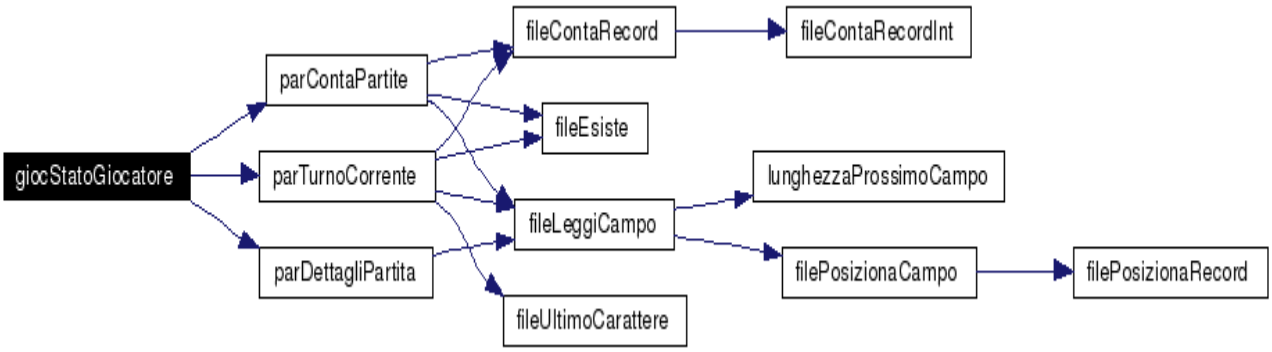
Restituisce:

LIBERO, ELIMINATO o GIOCATO

Definizione alla linea 19 del file giocatore.cpp.

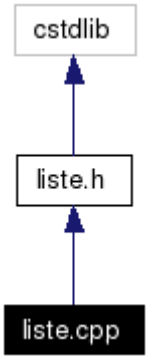
Riferimenti ELIMINATO, GIOCATO, Partita::idG1, Partita::idG2, LIBERO, parContaPartite (), parDettagliPartita(), parTurnoCorrente(), Partita::risG1, Partita::risG2, e Partita::turno.

Referenziato da giocGiocatoriLiberi().
Questo è il grafo delle chiamate per questa funzione:



File liste.cpp

Grafo delle dipendenze di inclusione per liste.cpp:



Funzioni

Lista * listeCreaLista (const void *primo)
Crea una nuova lista, e opzionalmente aggiunge il primo elemento.

void listeDistruggiLista (Lista *l)
Dealloca una lista e i rispettivi elementi.

void listeAggiungiElemento (Lista *l, const void *elemento)
Accoda un nuovo elemento alla lista.

const void * listeLeggiElemento (Lista *l, int i)
Legge l'i-esimo elemento e ne ritorna il puntatore.

void listeScriviElemento (Lista *l, int i, const void *elemento)
Sovrascrive l'i-esimo elemento della lista.

Documentazione delle funzioni

void listeAggiungiElemento(Lista * *l*,
const void * *element o*
)

Accoda un nuovo elemento alla lista.

Parametri:

l la lista a cui aggiungere l'elemento

elemento l'elemento da aggiungere

Definizione alla linea 40 del file liste.cpp.

Riferimenti Lista::coda, ElemLista::elemento, Lista::n, ElemLista::prossimo, e Lista::testa.

Referenziato da giocGiocatoriLiberi(), main(), menuCambiaGiocatore(), menuGiocatori(), menuNuovaPartitaMostra(), menuPartite(), e parListaPartiteGiocate().

Lista* listeCreaLista(const void * *primo*)

Crea una nuova lista, e opzionalmente aggiunge il primo elemento.

Parametri:

primo il primo elemento oppure NULL per una lista vuota

Restituisce:

la nuova Lista appena creata

Definizione alla linea 9 del file liste.cpp.

Riferimenti Lista::coda, ElemLista::elemento, Lista::n, ElemLista::prossimo, e Lista::testa.

Referenziato da giocGiocatoriLiberi(), main(), menuCambiaGiocatore(), menuGiocatori(), menuNuovaPartitaMostra(), menuPartite(), e parListaPartiteGiocate().

void listeDistruggiLista(Lista * *l*)

Dealloca una lista e i rispettivi elementi.

N.B.: **NON** dealloca i contenuti della lista!

Parametri:

l la lista da deallocare

Definizione alla linea 27 del file liste.cpp.

Riferimenti Lista::n, ElemLista::prossimo, e Lista::testa.

Referenziato da main(), menuCambiaGiocatore(), menuGiocatori(), menuNuovaPartita(), menuNuovaPartitaMostra(), menuPartite(), e menuPartiteTurno().

const void* listeLeggiElemento(Lista * *l*,
int *i*
)

Legge l'i-esimo elemento e ne ritorna il puntatore.

Gli indici della lista sono trattati come negli array monodimensionali del C: il primo elemento ha indice 0, l'ultimo n-1 (con n elementi).

Parametri:

l la lista da cui leggere

i l'indice dell'elemento da leggere

Restituisce:

il puntatore all'elemento letto

Definizione alla linea 57 del file liste.cpp.

Riferimenti ElemLista::elemento, ElemLista::prossimo, e Lista::testa.

Referenziato da menuCambiaGiocatore(), menuNuovaPartita(), menuPartiteTurno(), e uiStampaMenu().

```
void listeScriviElemento (  Lista *        l,
                            int            i,
                            const void *   element
                                           o
                          )
```

Sovrascrive l'i-esimo elemento della lista.

Gli indici della lista sono trattati come negli array monodimensionali del C: il primo elemento ha indice 0, l'ultimo n-1 (con n elementi). N.B.: **NON** dealloca l'elemento sovrascritto!

Parametri:

l la lista a cui cambiare l'elemento

i indice dell'elemento da sovrascrivere

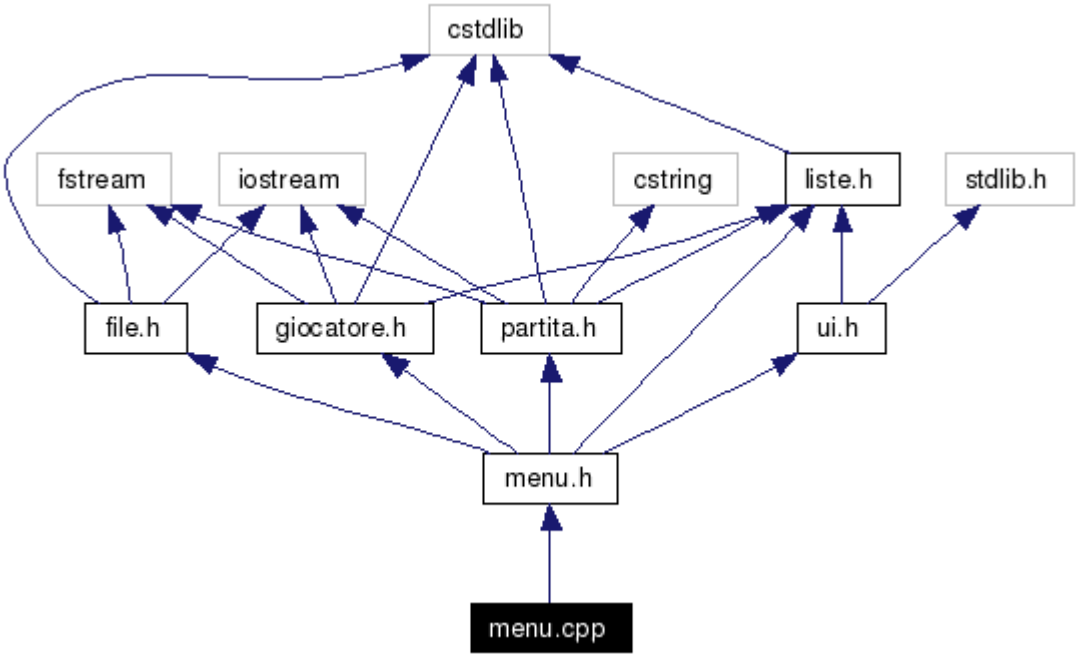
elemento il nuovo elemento

Definizione alla linea 65 del file liste.cpp.

Riferimenti ElemLista::elemento, ElemLista::prossimo, e Lista::testa.

File menu.cpp

Grafo delle dipendenze di inclusione per menu.cpp:



Funzioni

int menuCambiaGiocatore (const char *nomeFileGiocatori, const char *nomeFilePartite, Partita *p, int daCambiare)
Permette di cambiare un giocatore in una partita (se possibile) con un altro libero.

int menuCambiaRisultato (Partita *p, int daCambiare)
Permette di cambiare un risultato in una partita.

bool menuNuovaPartitaMostra (const char *nomeFileGiocatori, const char *nomeFilePartite, Partita *p)
Visualizza un'ipotesi di nuova partita e permette all'utente di cambiare concorrenti e risultati (ove possibile).

void menuNuovaPartita (const char *nomeFileGiocatori, const char *nomeFilePartite)
Controlla se sussistono le condizioni per iniziare una nuova partita.

void menuPartiteTurno (const char *nomeFileGiocatori, const char *nomeFilePartite, int turno)
Stampa la lista delle partite giocate in un turno, con i rispettivi risultati.

void menuChiudiTurno (const char *nomeFileGiocatori, const char *nomeFilePartite, int turno)
Chiude (se possibile) il turno corrente.

void menuTurniPassati (const char *nomeFileGiocatori, const char *nomeFilePartite)
Chiede all'utente di specificare un Turno (se necessario) di cui verranno stampate le partite.

void menuAggiungiGiocatore (const char *nomeFileGiocatori, const char *nomeFilePartite)
Chiede all'utente il nome di un giocatore da aggiungere e lo aggiunge al file.

void menuGiocatoriTorneo (const char *nomeFileGiocatori, const char *nomeFilePartite)
Stampa l'elenco dei giocatori nel Torneo.

void menuVediCarrieraGiocatore (const char *nomeFileGiocatori, const char *nomeFilePartite, int giocatore)
Stampa l'elenco degli avversari di un giocatore nel Torneo.
void menuCarrieraGiocatore (const char *nomeFileGiocatori, const char *nomeFilePartite)
Chiede all'utente di specificare un giocatore di cui mostrare la carriera (elenco degli avversari contro cui ha giocato e risultati).
void menuPartite (const char *nomeFileGiocatori, const char *nomeFilePartite)
Visualizza un menu con le seguenti possibili scelte:.
void menuGiocatori (const char *nomeFileGiocatori, const char *nomeFilePartite)
Visualizza un menu con le seguenti possibili scelte:.
void menuRiepilogoTorneo (const char *nomeFileGiocatori, const char *nomeFilePartite)
Visualizza un riepilogo dell'intero Torneo, con le partite giocate ad ogni turno.

Documentazione delle funzioni

void menuAggiungiGiocatore (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Chiede all'utente il nome di un giocatore da aggiungere e lo aggiunge al file.

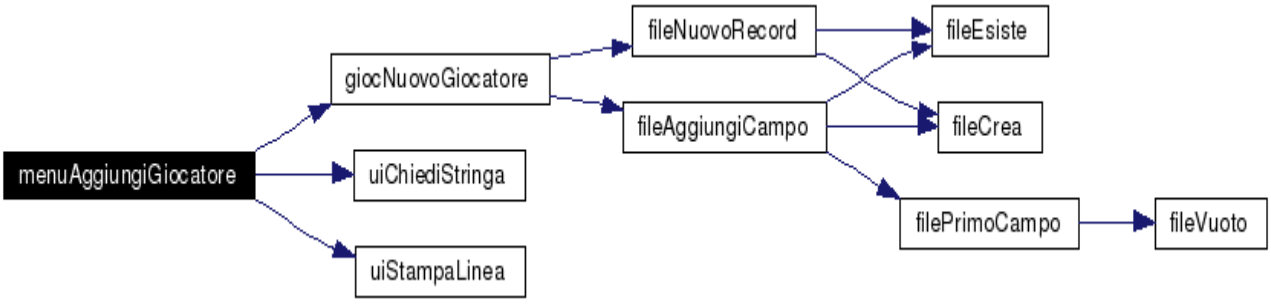
Parametri:

nomeFileGiocatori	stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 319 del file menu.cpp.

Riferimenti giocNuovoGiocatore(), uiChiediStringa(), e uiStampaLinea().

Referenziato da menuGiocatori().
Questo è il grafo delle chiamate per questa funzione:



int menuCambiaGiocatore (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
Partita * *p*,
int *daCambiare*
)

Permette di cambiare un giocatore in una partita (se possibile) con un altro libero.

Chiede una scelta all'utente in merito, se necessario.

Parametri:

nomeFileGiocatori	stringa terminata da '\0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da '\0' con il nome del file delle partite, completo di percorso
p	dettagli della partita a cui cambiare il giocatore
daCambiare	1 o 2, a seconda se il giocatore da cambiare è il primo o il secondo concorrente.

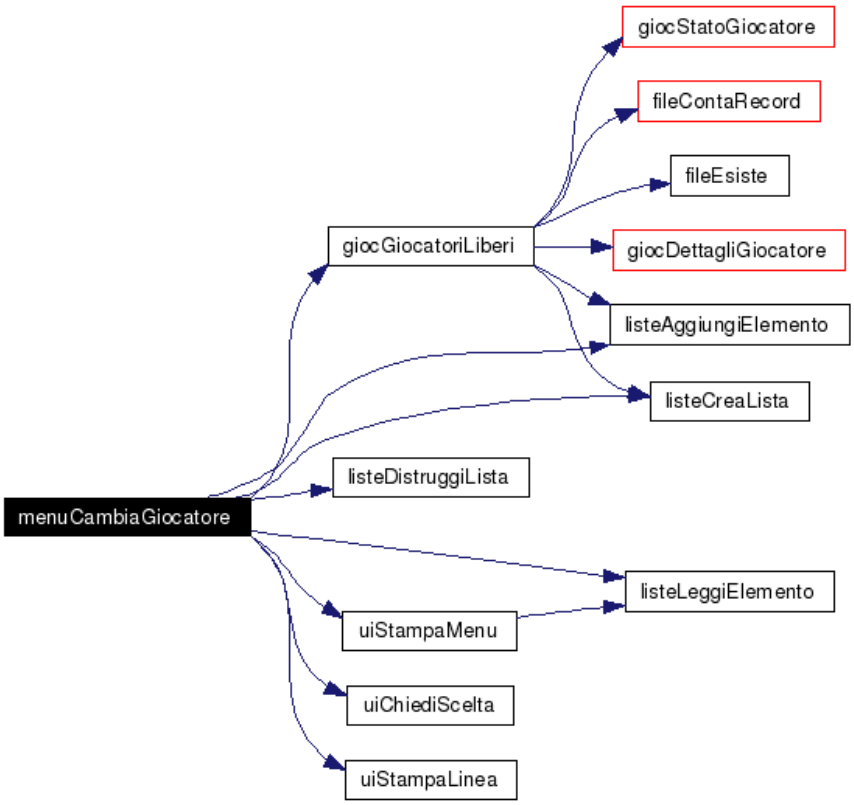
Restituisce:

l'id del nuovo giocatore

Definizione alla linea 18 del file menu.cpp.

Riferimenti giocGiocatoriLiberi(), Giocatore::id, Partita::idG1, Partita::idG2, listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), listeLeggiElemento(), Lista::n, Giocatore::nome, uiChiediScelta(), uiStampaLinea(), e uiStampaMenu().

Referenziato da menuNuovaPartitaMostra().
Questo è il grafo delle chiamate per questa funzione:



int menuCambiaRisultato (Partita * *p*,
int *daCambiare*
)

Permette di cambiare un risultato in una partita.

Chiede una scelta all'utente in merito.

Parametri:

p dettagli della partita a cui cambiare il risultato

daCambiare 1 o 2, a seconda se il giocatore cui cambiare il risultato da cambiare è il primo o il secondo concorrente.

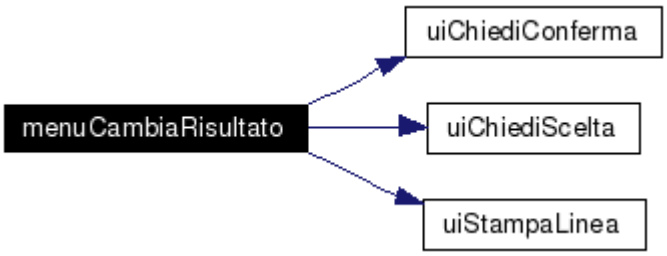
Restituisce:

il nuovo risultato

Definizione alla linea 77 del file menu.cpp.

Riferimenti Partita::risG1, Partita::risG2, uiChiediConferma(), uiChiediScelta(), e uiStampaLinea().

Referenziato da menuNuovaPartitaMostra().
Questo è il grafo delle chiamate per questa funzione:



void menuCarrieraGiocatore (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Chiede all'utente di specificare un giocatore di cui mostrare la carriera (elenco degli avversari contro cui ha giocato e risultati).

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

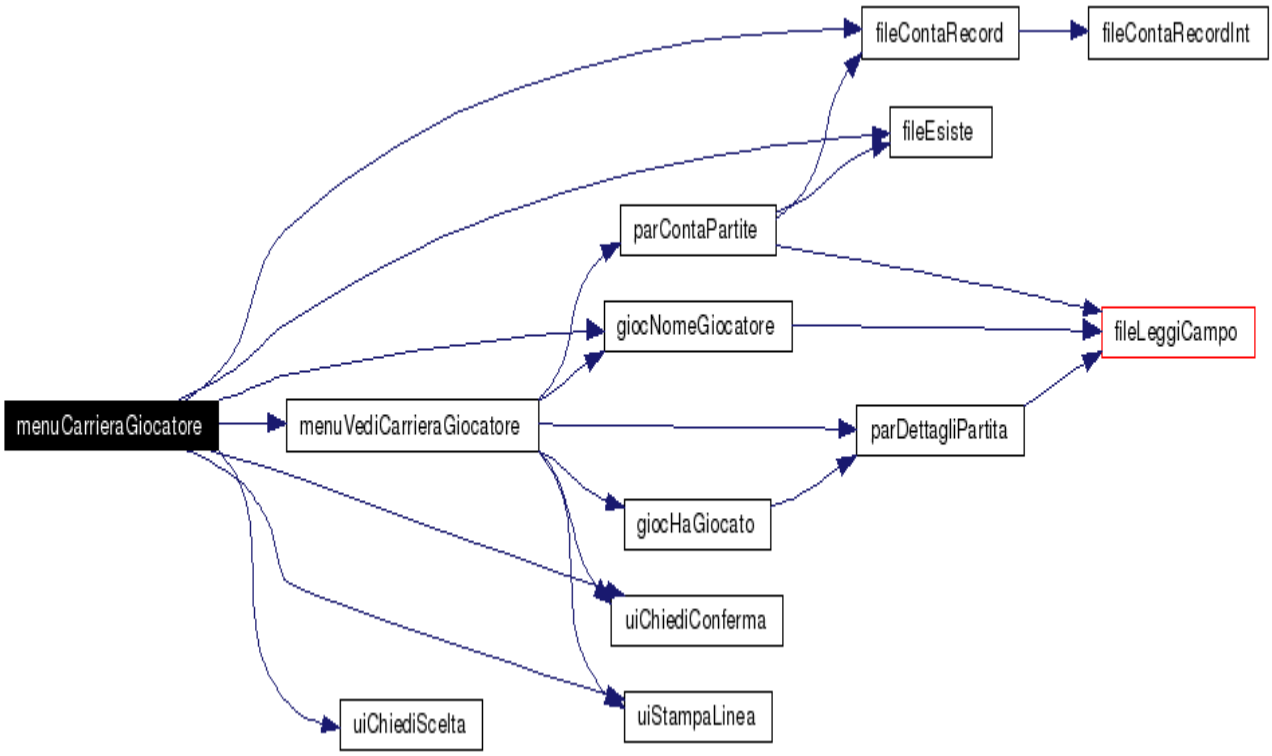
nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 408 del file menu.cpp.

Riferimenti fileContaRecord(), fileEsiste(), giocNomeGiocatore(), menuVediCarrieraGiocatore(), uiChiediConferma(), uiChiediScelta(), e uiStampaLinea().

Referenziato da menuGiocatori().

Questo è il grafo delle chiamate per questa funzione:



void menuChiudiTurno(const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
int *turno*
)

Chiude (se possibile) il turno corrente.

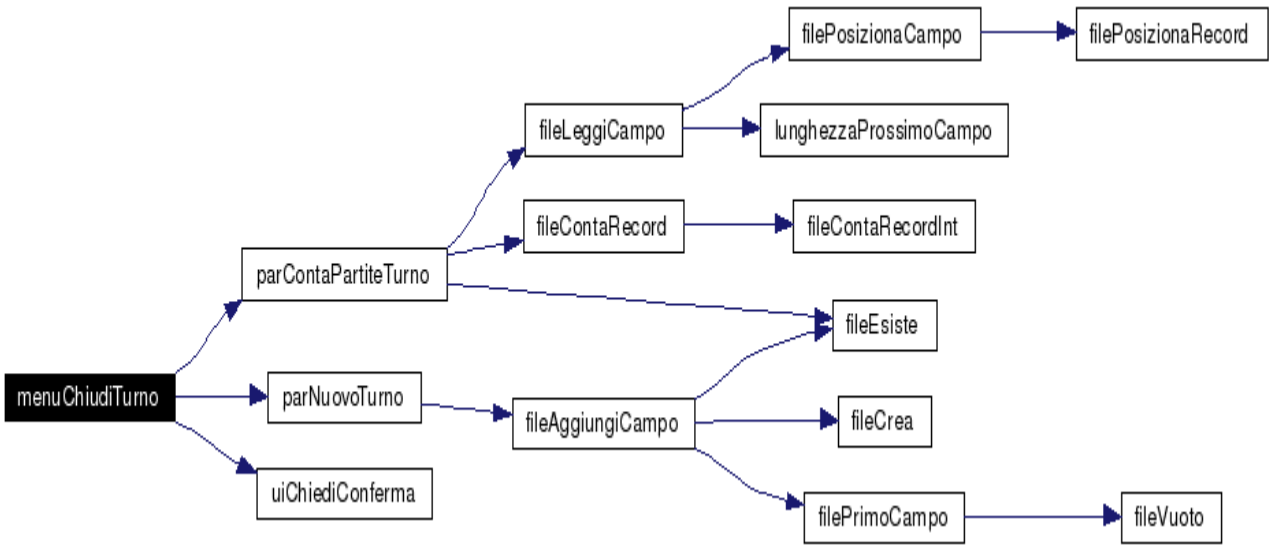
Parametri:

nomeFileGiocatori	stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da ' \0' con il nome del file delle partite, completo di percorso
turno	il numero del turno corrente

Definizione alla linea 265 del file menu.cpp.

Riferimenti parContaPartiteTurno(), parNuovoTurno(), e uiChiediConferma().

Referenziato da menuPartite().
Questo è il grafo delle chiamate per questa funzione:



void menuGiocatori (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Visualizza un menu con le seguenti possibili scelte:.

- Visualizza l'elenco dei giocatori in torneo
- Visualizza le partite e i risultati di un singolo giocatore
- Nuovo Giocatore
- Torna al Menu Principale

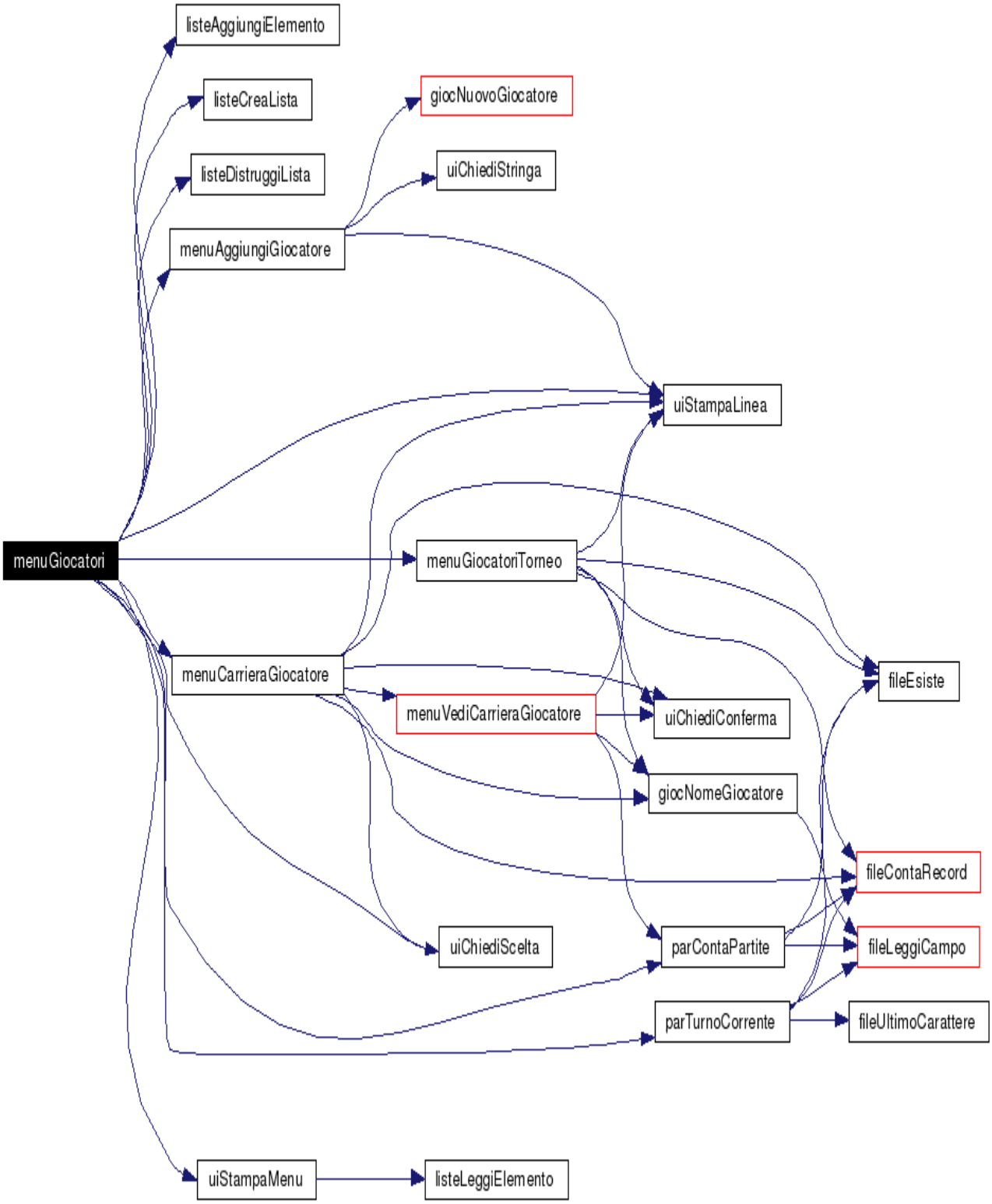
Parametri:

nomeFileGiocatori	stringa terminata da '\0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da '\0' con il nome del file delle partite, completo di percorso

Definizione alla linea 484 del file menu.cpp.

Riferimenti listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), menuAggiungiGiocatore(), menuCarrieraGiocatore(), menuGiocatoriTorneo(), parContaPartite(), parTurnoCorrente(), uiChiediScelta(), uiStampaLinea(), e uiStampaMenu().

Referenziato da main().
Questo è il grafo delle chiamate per questa funzione:



void menuGiocatoriTorneo(const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Stampa l'elenco dei giocatori nel Torneo.

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

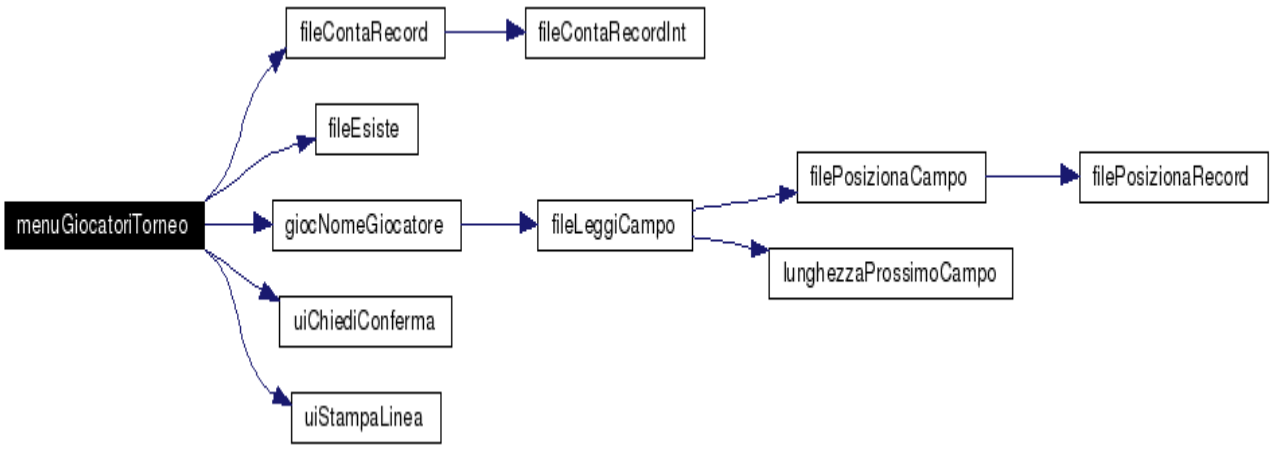
nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 337 del file menu.cpp.

Riferimenti fileContaRecord(), fileEsiste(), giocNomeGiocatore(), uiChiediConferma(), e uiStampaLinea().

Referenziato da menuGiocatori().

Questo è il grafo delle chiamate per questa funzione:



void menuNuovaPartita (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Controlla se sussistono le condizioni per iniziare una nuova partita.

In caso positivo, crea una partita ipotetica e la propone all'utente per i suoi cambiamenti, altrimenti avvisa dell'impossibilità di creare una nuova aprtita.

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

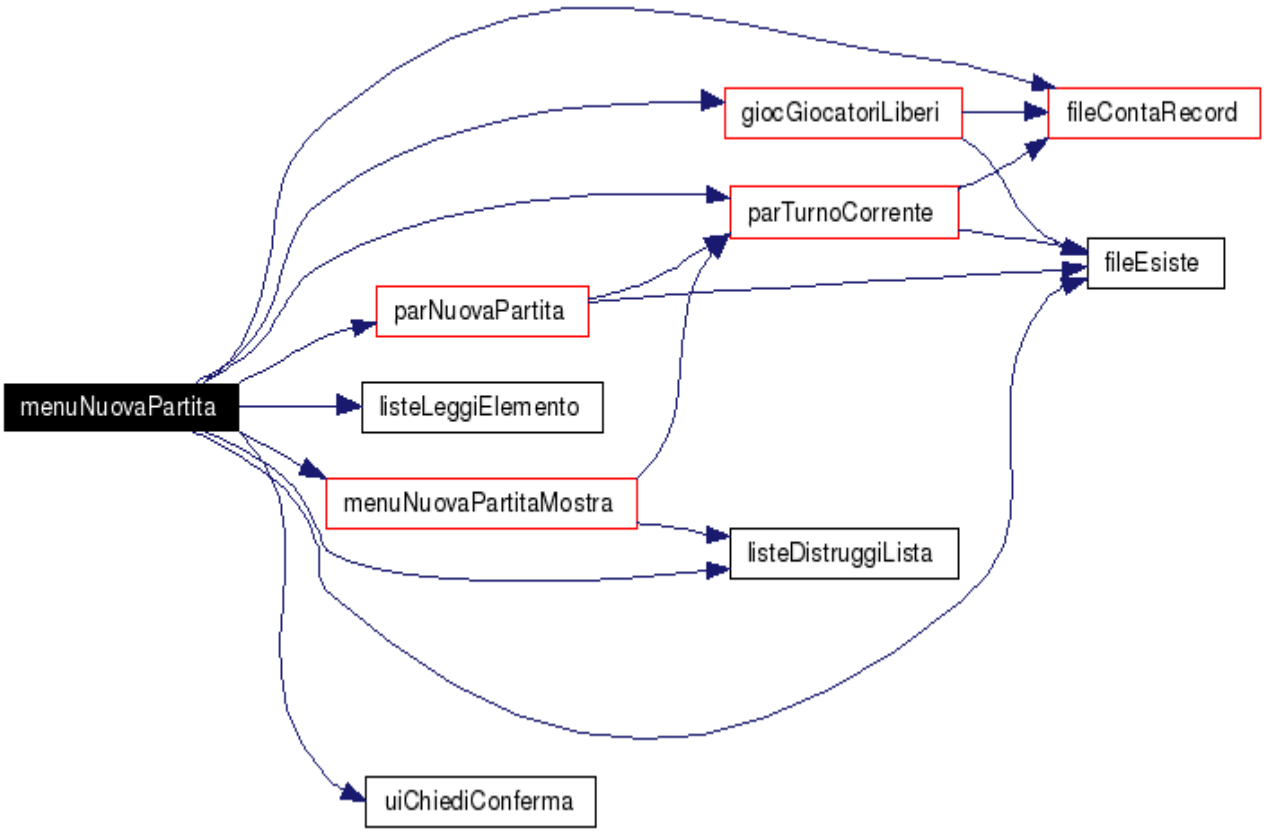
nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 174 del file menu.cpp.

Riferimenti fileContaRecord(), fileEsiste(), giocGiocatoriLiberi(), Giocatore::id, Partita::idG1, Partita::idG2, listeDistruggiLista(), listeLeggiElemento(), menuNuovaPartitaMostra(), Lista::n, Giocatore::nome, parNuovaPartita(), parTurnoCorrente(), Partita::risG1, Partita::risG2, Partita::turno, e uiChiediConferma().

Referenziato da menuPartite().

Questo è il grafo delle chiamate per questa funzione:



bool menuNuovaPartitaMostra (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
Partita * *p*
)

Visualizza un'ipotesi di nuova partita e permette all'utente di cambiare concorrenti e risultati (ove possibile).

Parametri:

nomeFileGiocatori	stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da ' \0' con il nome del file delle partite, completo di percorso
p	dettagli della partita di default

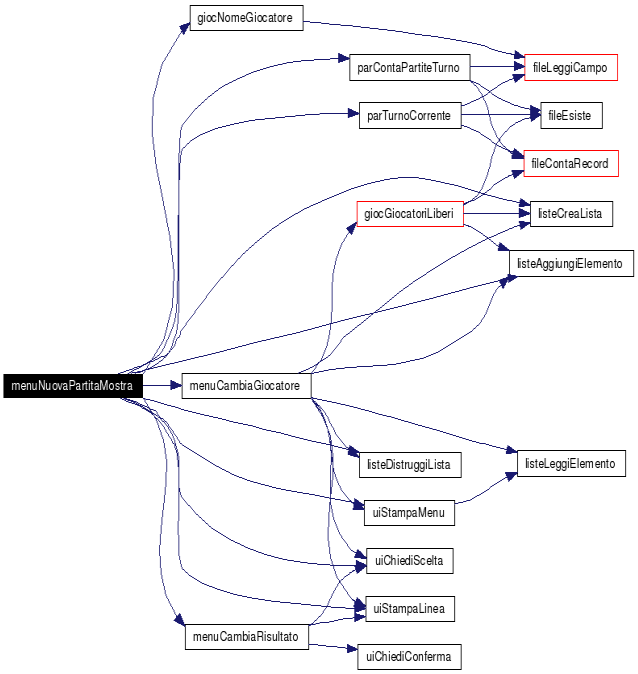
Restituisce:

true se la proposta della partita è stata accettata dall'utente, false altrimenti

Definizione alla linea 109 del file menu.cpp.

Riferimenti giocNomeGiocatore(), Partita::idG1, Partita::idG2, listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), menuCambiaGiocatore(), menuCambiaRisultato(), parContaPartiteTurno(), parTurnoCorrente(), Partita::risG1, Partita::risG2, uiChiediScelta(), uiStampaLinea(), e uiStampaMenu().

Referenziato da menuNuovaPartita().
Questo è il grafo delle chiamate per questa funzione:



void menuPartite (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Visualizza un menu con le seguenti possibili scelte:.

- Nuova partita
- Visualizza le partite giocate in questo turno (e i risultati)
- Chiudi il turno
- Visualizza i dettagli di un turno passato
- Torna al Menu Principale

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

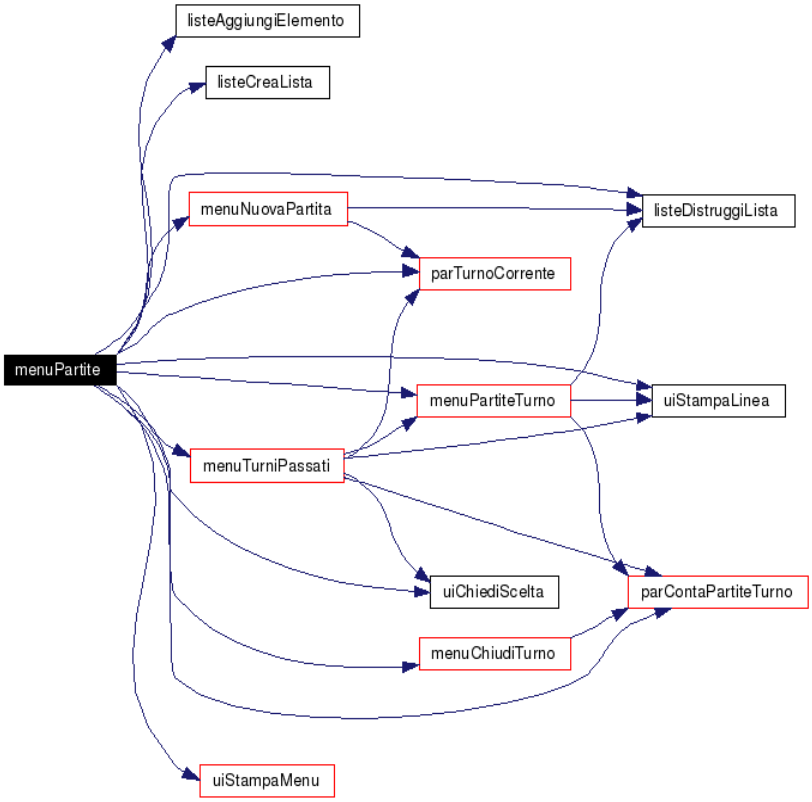
nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 444 del file menu.cpp.

Riferimenti listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), menuChiudiTurno(), menuNuovaPartita(), menuPartiteTurno(), menuTurniPassati(), parContaPartiteTurno(), parTurnoCorrente(), uiChiediScelta(), uiStampaLinea(), e uiStampaMenu().

Referenziato da main().

Questo è il grafo delle chiamate per questa funzione:



void menuPartiteTurno(const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
int *turno*
)

Stampa la lista delle partite giocate in un turno, con i rispettivi risultati.

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

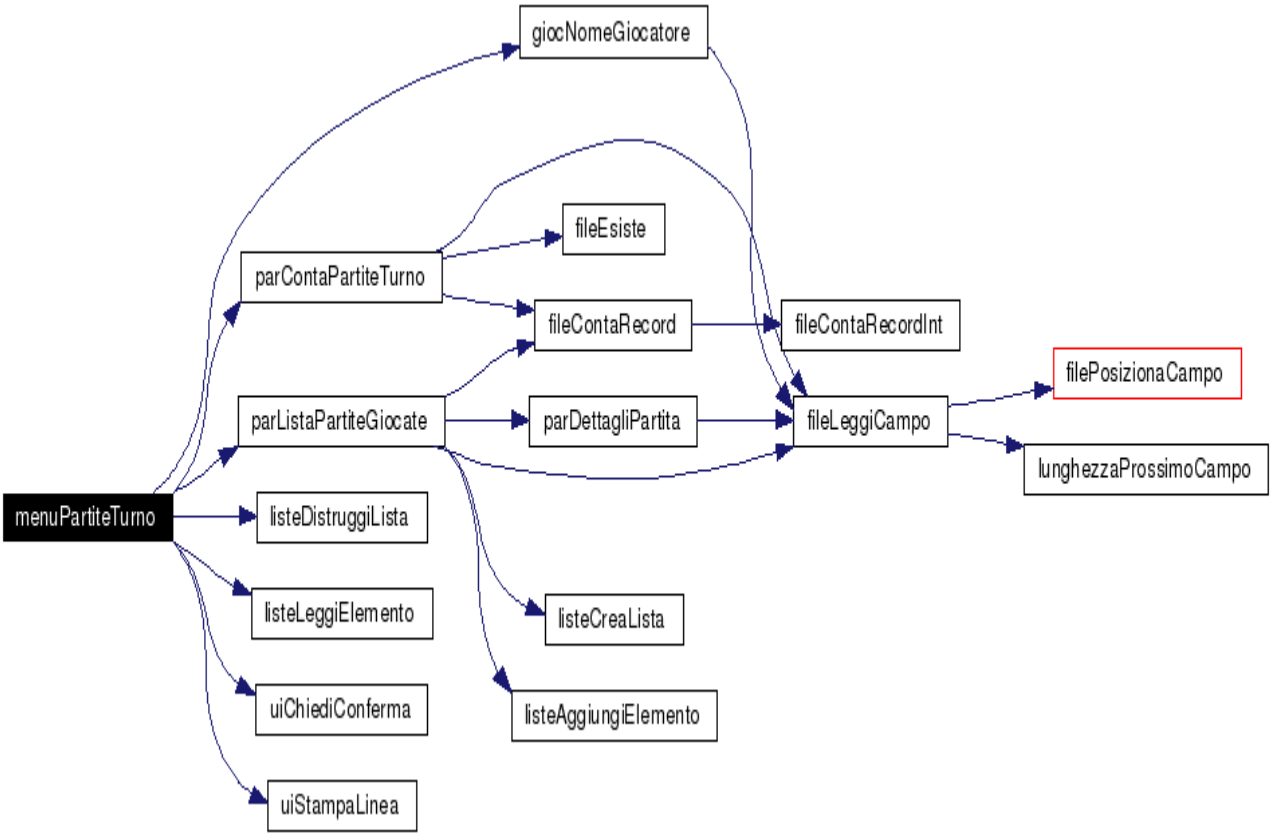
turno il numero del turno di cui stampare le partite

Definizione alla linea 227 del file menu.cpp.

Riferimenti giocNomeGiocatore(), Partita::idG1, Partita::idG2, listeDistruggiLista(), listeLeggiElemento(), Lista::n, parContaPartiteTurno(), parListaPartiteGiocate(), Partita::risG1, Partita::risG2, uiChiediConferma(), e uiStampaLinea().

Referenziato da menuPartite(), menuRiepilogoTorneo(), e menuTurniPassati().

Questo è il grafo delle chiamate per questa funzione:



void menuRiepilogoTorneo (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Visualizza un riepilogo dell'intero Torneo, con le partite giocate ad ogni turno.

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

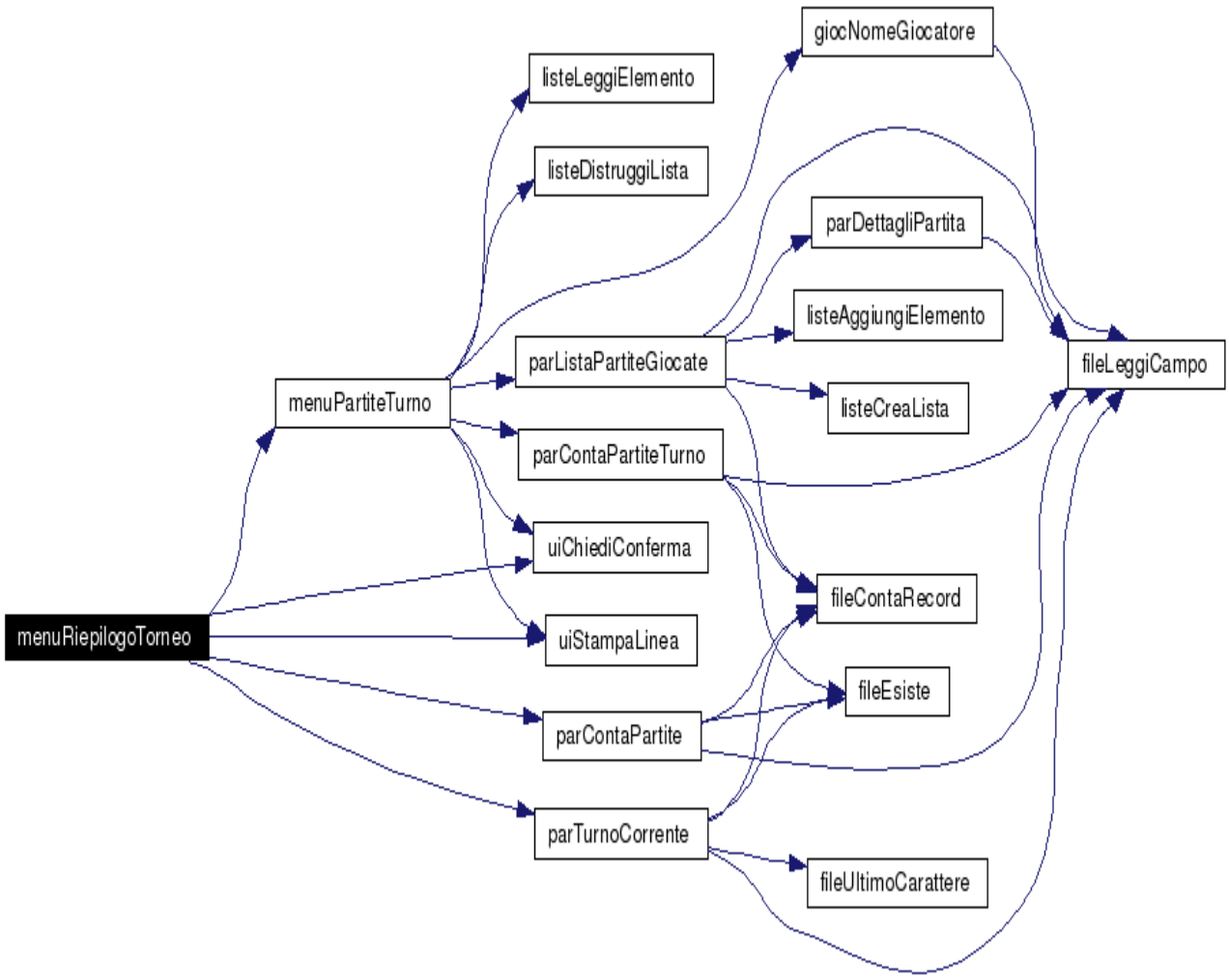
nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 519 del file menu.cpp.

Riferimenti menuPartiteTurno(), parContaPartite(), parTurnoCorrente(), uiChiediConferma(), e uiStampaLinea().

Referenziato da main().

Questo è il grafo delle chiamate per questa funzione:



void menuTurniPassati (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*
)

Chiede all'utente di specificare un Turno (se necessario) di cui verranno stampate le partite.

Parametri:

nomeFileGiocatori stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso

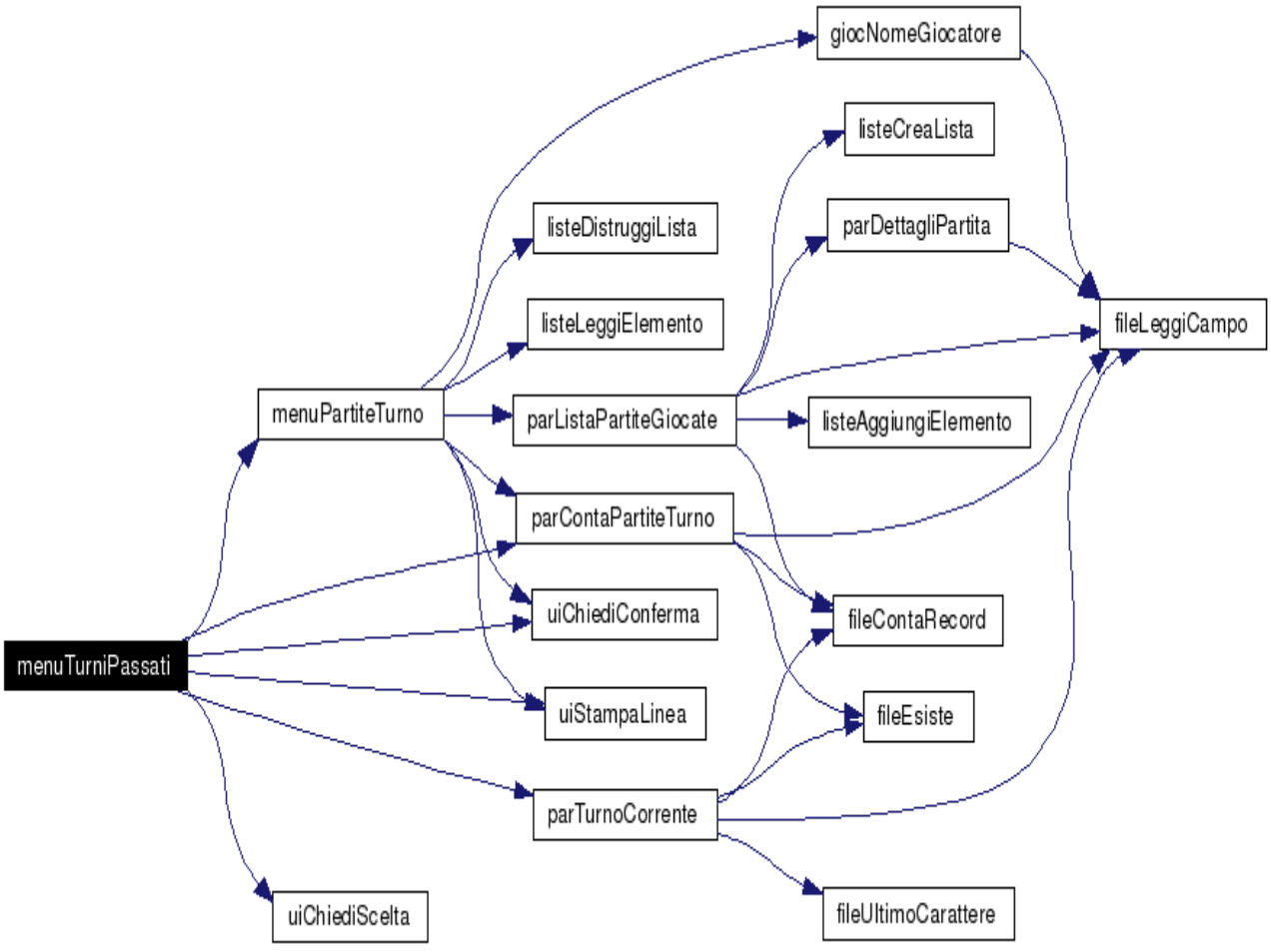
nomeFilePartite stringa terminata da ' \0' con il nome del file delle partite, completo di percorso

Definizione alla linea 287 del file menu.cpp.

Riferimenti menuPartiteTurno(), parContaPartiteTurno(), parTurnoCorrente(), uiChiediConferma(), uiChiediScelta(), e uiStampaLinea().

Referenziato da menuPartite().

Questo è il grafo delle chiamate per questa funzione:



void menuVediCarrieraGiocatore (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
int *giocatore*
)

Stampa l'elenco degli avversari di un giocatore nel Torneo.

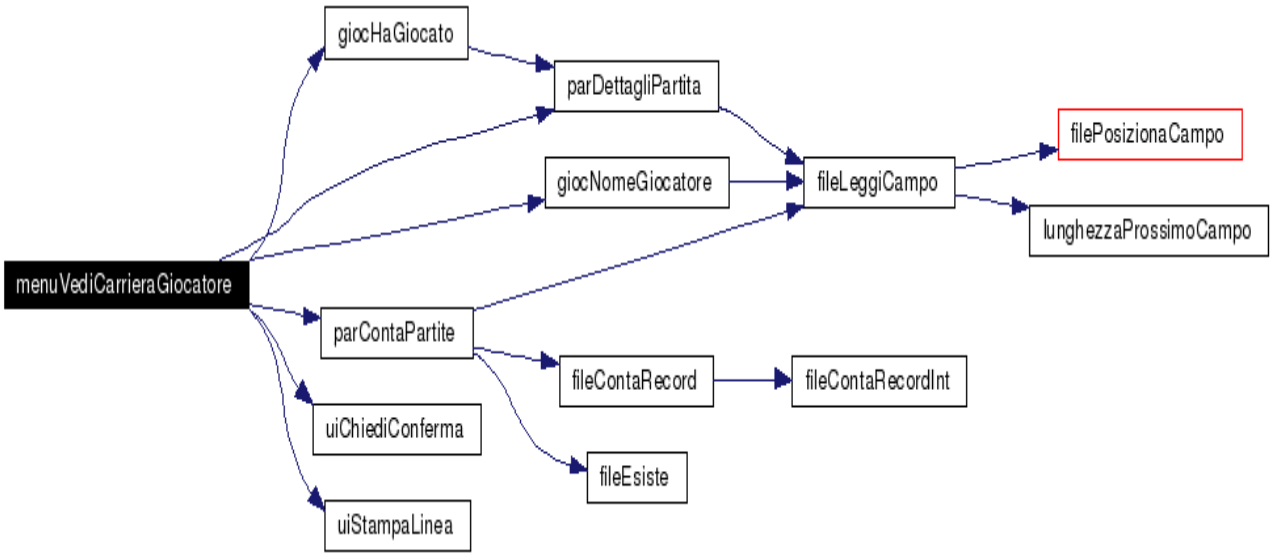
Parametri:

nomeFileGiocatori	stringa terminata da ' \0' con il nome del file dei giocatori, completo di percorso
nomeFilePartite	stringa terminata da ' \0' con il nome del file delle partite, completo di percorso
giocatore	l'id del giocatore da analizzare

Definizione alla linea 365 del file menu.cpp.

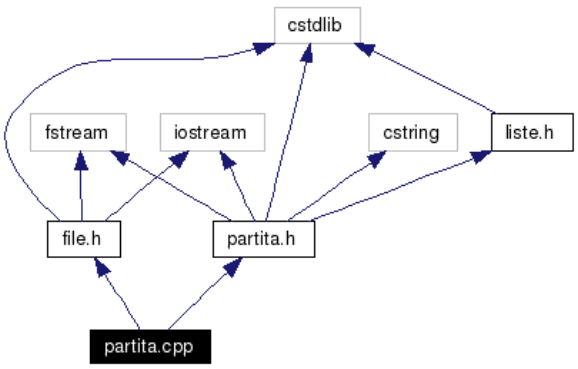
Riferimenti giocHaGiocato(), giocNomeGiocatore(), HAPERSO, HAVINTO, Partita::idG1, Partita::idG2, NONGIOCA, parContaPartite(), parDettagliPartita(), uiChiediConferma(), e uiStampaLinea().

Referenziato da menuCarrieraGiocatore().
Questo è il grafo delle chiamate per questa funzione:



File partita.cpp

Grafo delle dipendenze di inclusione per partita.cpp:



Vai al codice sorgente di questo file.

Funzioni

int parTurnoCorrente (const char *nomeFile)
Ritorna il numero progressivo di questo turno.

int parContaPartite (const char *nomeFile)
Ritorna il numero totale di partite giocate nel Torneo.

int parContaPartiteTurno (const char *nomeFile, int turno)
Ritorna il numero di partite giocate in questo turno.

Partita * parDettagliPartita (const char *nomeFile, int id)
Carica i dati relativi a una partita da un file a una struttura in memoria.

Lista * parListaPartiteGiocate (const char *nomeFile, int turno)
Carica i dati relativi a tutte le partite di un certo turno e li ritorna in una lista opportuna.

void parNuovaPartita (const char *nomeFileGiocatori, const char *nomeFilePartite, int idG1, int idG2, int risG1, int risG2)
Aggiunge una nuova partita al file.

void parNuovoTurno (const char *nomeFile)
Chiude il turno corrente aprendone uno nuovo.

Documentazione delle funzioni

int parContaPartite (const char * *nomeFile*)

Ritorna il numero totale di partite giocate nel Torneo.

Parametri:
nomeFile stringa terminata da '\0' con il nome del file

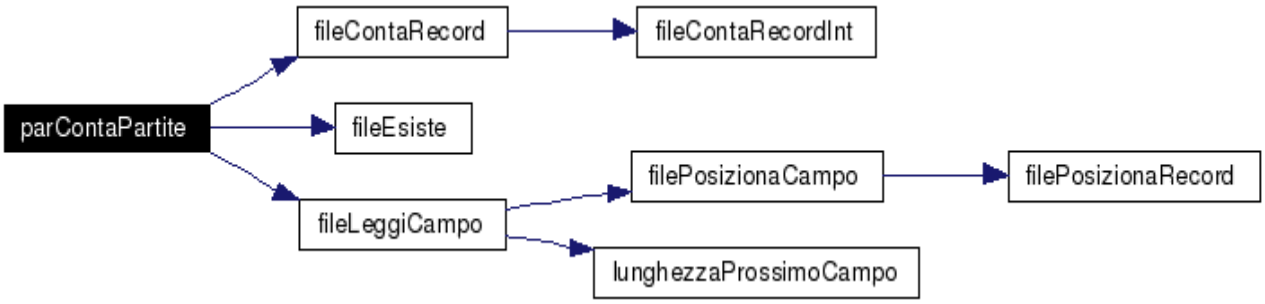
Restituisce:
il numero di partite

Definizione alla linea 26 del file partita.cpp.

Riferimenti fileContaRecord(), fileEsiste(), e fileLeggiCampo().

Referenziato da giocStatoGiocatore(), main(), menuGiocatori(), menuRiepilogoTorneo(), e menuVediCarrieraGiocatore().

Questo è il grafo delle chiamate per questa funzione:



int parContaPartiteTurno(const char * *nomeFile*,
int *turno*
)

Ritorna il numero di partite giocate in questo turno.

Parametri:

nomeFile stringa terminata da ' \0' con il nome del file a cui aggiungere il campo, completo di percorso.

turno il turno del quale si vogliono avere informazioni

Restituisce:

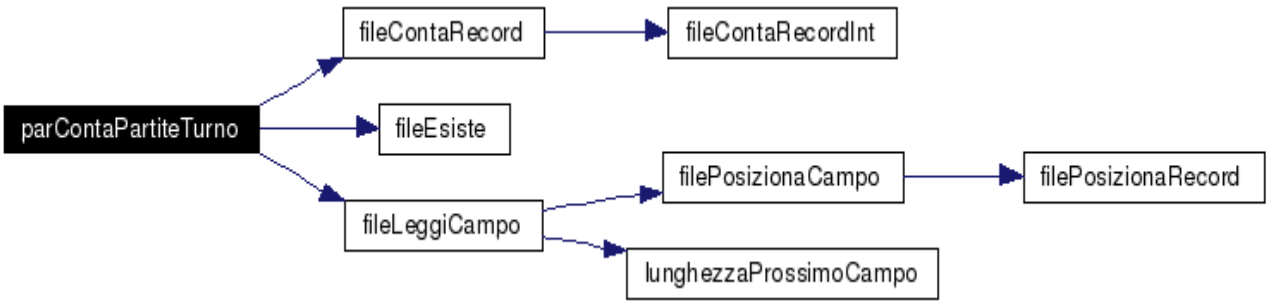
il numero di partite giocate nel turno

Definizione alla linea 45 del file partita.cpp.

Riferimenti fileContaRecord(), fileEsiste(), e fileLeggiCampo().

Referenziato da menuChiudiTurno(), menuNuovaPartitaMostra(), menuPartite(), menuPartiteTurno(), e menuTurniPassati().

Questo è il grafo delle chiamate per questa funzione:



Partita* parDettagliPartita (const char * *nomeFile*,
int *id*
)

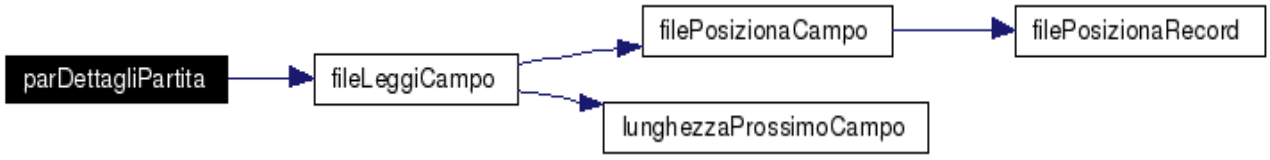
Carica i dati relativi a una partita da un file a una struttura in memoria.

Parametri:
nomeFile stringa terminata da '\0' con il nome del file da leggere
id numero identificativo della partita
Restituisce:
una struttura con i dettagli della partita richiesta

Definizione alla linea 68 del file partita.cpp.

Riferimenti fileLeggiCampo(), Partita::idG1, Partita::idG2, Partita::risG1, Partita::risG2, e Partita::turno.

Referenziato da giocHaGiocato(), giocStatoGiocatore(), menuVediCarrieraGiocatore(), e parListaPartiteGiocate().
Questo è il grafo delle chiamate per questa funzione:



Lista* parListaPartiteGiocate (const char * *nomeFile*,
int *turno*
)

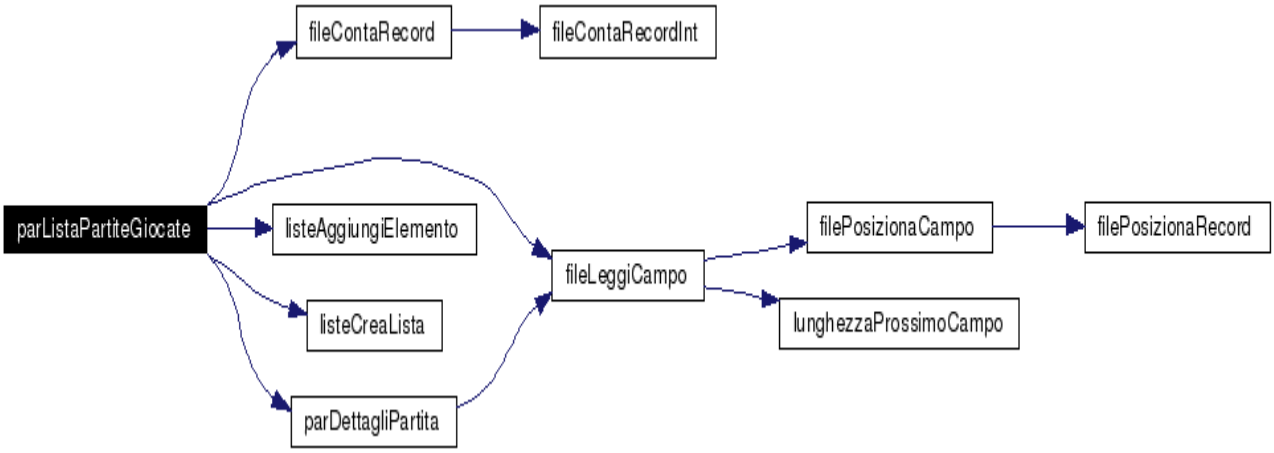
Carica i dati relativi a tutte le partite di un certo turno e li ritorna in una lista opportuna.

Parametri:
nomeFile stringa terminata da '\0' con il nome del file da leggere
turno il numero del turno di cui caricare i dati delle partite
Restituisce:
una Lista in cui ogni elemento è una struttura Partita contenente i dettagli di una delle partite giocate in questo turno

Definizione alla linea 92 del file partita.cpp.

Riferimenti fileContaRecord(), fileLeggiCampo(), listeAggiungiElemento(), listeCreaLista(), e parDettagliPartita().

Referenziato da menuPartiteTurno().
Questo è il grafo delle chiamate per questa funzione:



void parNuovaPartita (const char * *nomeFileGiocatori*,
const char * *nomeFilePartite*,
int *idG1*,
int *idG2*,
int *risG1*,
int *risG2*
)

Aggiunge una nuova partita al file.

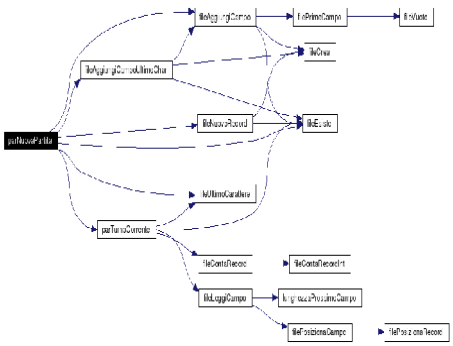
Parametri:

nomeFilePartite	stringa terminata da ' \0' con il nome del file a cui aggiungere la partita
nomeFileGiocatori	stringa terminata da ' \0' con il nome del file dei giocatori
idG1	numero identificativo del primo concorrente
idG2	numero identificativo del secondo concorrente
risG1	punteggio del primo giocatore a fine partita
risG2	punteggio del secondo giocatore a fine partita

Definizione alla linea 115 del file partita.cpp.

Riferimenti fileAggiungiCampo(), fileAggiungiCampoUltimoChar(), fileEsiste(), fileNuovoRecord(), fileUltimoCarattere(), e parTurnoCorrente().

Referenziato da menuNuovaPartita().
Questo è il grafo delle chiamate per questa funzione:



void parNuovoTurno(const char * *nomeFile*)

Chiude il turno corrente aprendone uno nuovo.

Parametri:

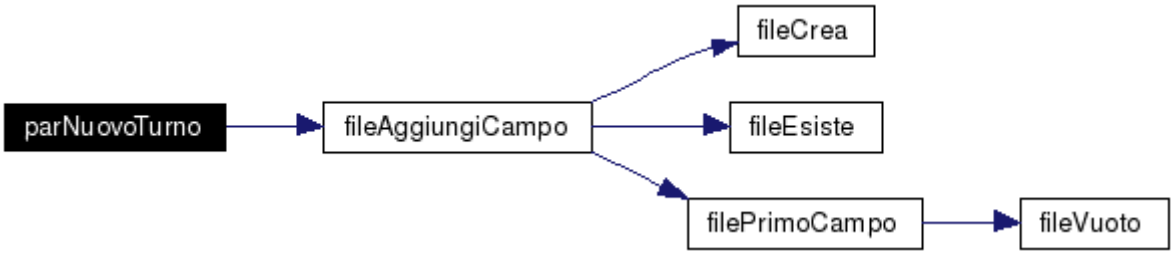
nomeFile stringa terminata da '\0' con il nome del file da leggere

Definizione alla linea 140 del file partita.cpp.

Riferimenti fileAggiungiCampo().

Referenziato da menuChiudiTurno().

Questo è il grafo delle chiamate per questa funzione:



int parTurnoCorrente(const char * *nomeFile*)

Ritorna il numero progressivo di questo turno.

Parametri:

nomeFile stringa terminata da '\0' con il nome del file a cui aggiungere il campo, completo di percorso.

Restituisce:

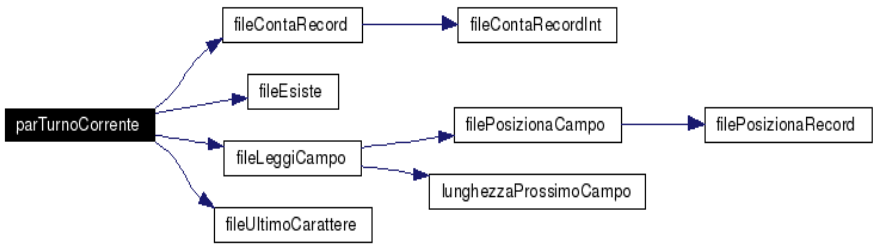
il numero del turno corrente

Definizione alla linea 10 del file partita.cpp.

Riferimenti fileContaRecord(), fileEsiste(), fileLeggiCampo(), e fileUltimoCarattere().

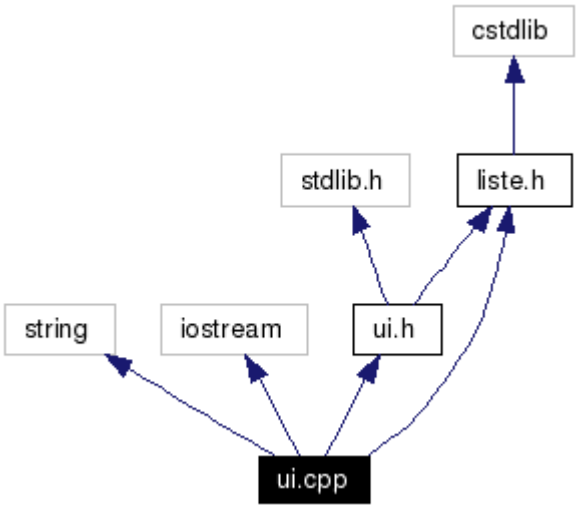
Referenziato da giocStatoGiocatore(), main(), menuGiocatori(), menuNuovaPartita(), menuNuovaPartitaMostra(), menuPartite(), menuRiepilogoTorneo(), menuTurniPassati(), e parNuovaPartita().

Questo è il grafo delle chiamate per questa funzione:



File ui.cpp

Grafo delle dipendenze di inclusione per ui.cpp:



Vai al codice sorgente di questo file.

Funzioni

void uiStampaLinea ()
Stampa una linea di 20 caratteri a video.

void uiStampaMenu (Lista *voci)
Stampa un elenco di voci numerate per un menu.

int uiChiediScelta (int sceltaMax)
Chiede all'utente di immettere un numero compreso tra 1 e sceltaMax.

void uiChiediConferma ()
Chiede all'utente di battere un tasto e quindi Invio.

char * uiChiediStringa ()
Chiede all'utente di digitare una stringa (lunghezza massima MAXCONSOLEINPUT compreso il '\0') e la ritorna.

void uiStampaBanner ()
Stampa il banner del programma.

Documentazione delle funzioni

void uiChiediConferma ( )

Chiede all'utente di battere un tasto e quindi Invio.

Questo purtroppo è l'unico modo concesso per chiedere una conferma dalle librerie standard del C++. Questa funzione potrebbe essere meglio reimplementata in un contesto non cross-platform con le funzioni del DOS (getch()) o di un*x (ncurses).

Definizione alla linea 44 del file ui.cpp.

Riferimenti MAXCONSOLEINPUT.

Referenziato da menuCambiaRisultato(), menuCarrieraGiocatore(), menuChiudiTurno(), menuGiocatoriTorneo(), menuNuovaPartita(), menuPartiteTurno(), menuRiepilogoTorneo(), menuTurniPassati(), e menuVediCarrieraGiocatore().

int uiChiediScelta(int *sceltaMax*)

Chiede all'utente di immettere un numero compreso tra 1 e sceltaMax.

Parametri:

sceltaMax massimo numero accettabile

Restituisce:

la scelta dell'utente

Definizione alla linea 25 del file ui.cpp.

Referenziato da main(), menuCambiaGiocatore(), menuCambiaRisultato(), menuCarrieraGiocatore(), menuGiocatori(), menuNuovaPartitaMostra(), menuPartite(), e menuTurniPassati().

char* uiChiediStringa( )

Chiede all'utente di digitare una stringa (lunghezza massima MAXCONSOLEINPUT compreso il '\0') e la ritorna.

Non ammette stringhe vuote. Usa esclusivamente funzioni "sicure" per evitare i buffer overflow.

Restituisce:

la stringa immessa dall'utente

Definizione alla linea 57 del file ui.cpp.

Riferimenti MAXCONSOLEINPUT.

Referenziato da main(), e menuAggiungiGiocatore().

void uiStampaBanner( )

Stampa il banner del programma.

Definizione alla linea 68 del file ui.cpp.

Referenziato da main().

void uiStampaLinea ( )

Stampa una linea di 20 caratteri a video.

Definizione alla linea 13 del file ui.cpp.

Referenziato da main(), menuAggiungiGiocatore(), menuCambiaGiocatore(), menuCambiaRisultato(), menuCarrieraGiocatore(), menuGiocatori(), menuGiocatoriTorneo(), menuNuovaPartitaMostra(), menuPartite(), menuPartiteTurno(), menuRiepilogoTorneo(), menuTurniPassati(), e menuVediCarrieraGiocatore().

void uiStampaMenu (Lista * *voci*)

Stampa un elenco di voci numerate per un menu.

Parametri:

voci una Lista contenente stringhe terminate da '\0' che rappresentano le voci di menu

Definizione alla linea 17 del file ui.cpp.

Riferimenti listeLeggiElemento(), e Lista::n.

Referenziato da main(), menuCambiaGiocatore(), menuGiocatori(), menuNuovaPartitaMostra(), e menuPartite().

Questo è il grafo delle chiamate per questa funzione:

struct ElemLista

Descrive un elemento della lista.

Diagramma di collaborazione per ElemLista:

Campi

const void * elemento
Puntatore al contenuto dell'elemento.

ElemLista * prossimo
Puntatore al prossimo elemento nella lista o NULL.

Descrizione Dettagliata

Descrive un elemento della lista.

Definizione alla linea 17 del file liste.h.

Documentazione dei campi

const void* ElemLista::elemento

Puntatore al contenuto dell'elemento.

Definizione alla linea 19 del file liste.h.

Referenziato da listeAggiungiElemento(), listeCreaLista(), listeLeggiElemento(), e listeScriviElemento().

ElemLista* ElemLista::prossimo

Puntatore al prossimo elemento nella lista o NULL.

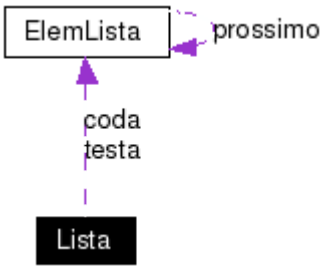
Definizione alla linea 21 del file liste.h.

Referenziato da listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), listeLeggiElemento(), e listeScriviElemento().

struct Lista

Descrive un'intera lista.

Diagramma di collaborazione per Lista:



Campi

ElemLista * testa
Puntatore al primo elemento della lista.

ElemLista * coda
Puntatore all'ultimo elemento della lista.

int n
Numero di elementi nella lista (sola lettura).

Descrizione Dettagliata

Descrive un'intera lista.

Definizione alla linea 25 del file liste.h.

Documentazione dei campi

ElemLista* Lista::coda
Puntatore all'ultimo elemento della lista.

Definizione alla linea 29 del file liste.h.

Referenziato da listeAggiungiElemento(), e listeCreaLista().

int Lista::n

Numero di elementi nella lista (sola lettura).

Definizione alla linea 31 del file liste.h.

Referenziato da listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), menuCambiaGiocatore(), menuNuovaPartita(), menuPartiteTurno(), e uiStampaMenu().

ElemLista* Lista::testa

Puntatore al primo elemento della lista.

Definizione alla linea 27 del file liste.h.

Referenziato da listeAggiungiElemento(), listeCreaLista(), listeDistruggiLista(), listeLeggiElemento(), e listeScriviElemento().

struct Giocatore

Struttura che descrive un giocatore del torneo.

Campi

int id
: Il numero identificativo del giocatore.

char * nome
: Il nome del giocatore, terminato da '\0'.

Descrizione Dettagliata

Struttura che descrive un giocatore del torneo.

Definizione alla linea 51 del file giocatore.h.

Documentazione dei campi

int Giocatore::id

Il numero identificativo del giocatore.

Definizione alla linea 53 del file giocatore.h.

Referenziato da giocDettagliGiocatore(), menuCambiaGiocatore(), e menuNuovaPartita().

char* Giocatore::nome

Il nome del giocatore, terminato da '\0'.

Definizione alla linea 55 del file giocatore.h.

Referenziato da giocDettagliGiocatore(), menuCambiaGiocatore(), e menuNuovaPartita().

struct Partita

Struttura che descrive una partita.

Campi

int turno
Il numero del turno in cui questa partita è giocata.
int idG1
idG1 numero identificativo del primo concorrente
int idG2
iidG2 numero identificativo del secondo concorrente
int risG1
risG1 punteggio del primo giocatore a fine partita
int risG2
risG2 punteggio del secondo giocatore a fine partita

Descrizione Dettagliata

Struttura che descrive una partita.

Definizione alla linea 23 del file partita.h.

Documentazione dei campi

int Partita::idG1

idG1 numero identificativo del primo concorrente

Definizione alla linea 27 del file partita.h.

Referenziato da giocHaGiocato(), giocStatoGiocatore(), menuCambiaGiocatore(), menuNuovaPartita(), menuNuovaPartitaMostra(), menuPartiteTurno(), menuVediCarrieraGiocatore(), e parDettagliPartita().

int Partita::idG2

iidG2 numero identificativo del secondo concorrente

Definizione alla linea 29 del file partita.h.

Referenziato da giocHaGiocato(), giocStatoGiocatore(), menuCambiaGiocatore(), menuNuovaPartita(), menuNuovaPartitaMostra(), menuPartiteTurno(), menuVediCarrieraGiocatore(), e parDettagliPartita().

int Partita::risG1

risG1 punteggio del primo giocatore a fine partita

Definizione alla linea 31 del file partita.h.

Referenziato da giocHaGiocato(), giocStatoGiocatore(), menuCambiaRisultato(), menuNuovaPartita(), menuNuovaPartitaMostra(), menuPartiteTurno(), e parDettagliPartita().

int Partita::risG2

risG2 punteggio del secondo giocatore a fine partita

Definizione alla linea 33 del file partita.h.

Referenziato da giocHaGiocato(), giocStatoGiocatore(), menuCambiaRisultato(), menuNuovaPartita(), menuNuovaPartitaMostra(), menuPartiteTurno(), e parDettagliPartita().

int Partita::turno

Il numero del turno in cui questa partita è giocata.

Definizione alla linea 25 del file partita.h.

Referenziato da giocStatoGiocatore(), menuNuovaPartita(), e parDettagliPartita().