1. Analisi

1.1 Descrizione del Sistema

Il sistema riproduce le funzionalità di una piattaforma di gestione dati immobiliari. Il programma dovrà permettere di leggere da file e visualizzare i dati dei clienti, professionisti ed immobili, consentire quindi l'aggiunta di nuove offerte relative agli immobili. Dovrà inoltre consentire la ricerca tra gli immobili presenti in memoria in basi a criteri quali prezzo o località e infine dovrà anche permettere la visualizzazione dei risultati di vendita ottenuti dell'agenzia secondo un certo intervallo di tempo.

Ogni entità (clienti, professionisti e immobili) è stata definita con i loro attributi caratteristici.

Sono state implementate diverse estensioni aggiuntive come:

- Gestione completa di tutte le entità trattate direttamente nel software, quindi aggiunta e/o eliminazione dei dati relativi ai clienti e professionisti (oltre che agli immobili) in maniera guidata, senza obbligo di modifica file

- Salvataggio dei record nel file in ordine alfabetico

- Ricerca di duplicati con possibilità di risoluzione del conflitto direttamente all'interno del programma.

Oltre a ciò sono state aggiunte anche funzioni "utils" preposte a diversi compiti importanti "sotto il cofano" quali:

- gestione corretta dell'input con relativa validazione dei dati inseriti come `readString`, `readInteger` e `readDouble`()

- modifica dinamica del titolo della finestra in base alla posizione dell'utente nei menu

- applicazione colori all'output grazie all'utilizzo degli escape codes ANSI

- richiesta di conferma da parte dell'utente standardizzata: `askConfirm`()

1.2 Requisiti Funzionali

Codice	Nome	Descrizione
R01	Visualizzazione menu	Il programma deve mostrare all'utente un menu iniziale dove vengono visualizzate le opzioni disponibili.
R02	Caricamento dati da file	Il programma deve caricare i dati da file per ogni entità.
R03	Ordinamento alfabetico dei File	Il programma deve garantire il corretto ordinamento dei record presenti relativi alle diverse entità.
R04	Ricerca duplicati	Il programma deve cercare eventuali conflitti tra i dati presenti e risolverli.
R05	Ricerca immobili	Il programma deve visualizzare gli immobili che rispettano determinati criteri decisi dall'utente.
R06	Aggiornamento file	Il programma deve aggiornare i file relativi alle varie entità con le informazioni aggiornate.
R07	Aggiunta dati	Il programma deve poter aggiungere nuovi immobili/clienti/professionisti con dati inseriti dall'utente.
R08	Controllo clienti scaduti	Il programma deve controllare la data e permettere la cancellazione dell'utente se scaduto.
R09	Visualizzazione dati	Il programma deve mostrare le informazioni sugli immobili, clienti e professionisti.

1.3 Strumenti di Sviluppo

Il software è stato realizzato utilizzando un:

- un laptop con CPU Intel i7-7500U e 16GB di RAM
- un desktop con CPU Intel i7-4771 a 3.5 GHz e 16GB di RAM

Entrambi i computer montano come sistema operativo Windows.

L'ambiente di sviluppo adottato è stato Eclipse, corredato dal plugin **Eclox** per documentazione generata da **Doxygen**.

Per eseguire il software è necessario avere un computer con sistema operativo Windows, però con alcune modifiche, relative perlopiù alla funzione `pause()`, è possibile compilare ed utilizzare il software anche su macOS o Linux.

2. Progettazione

2.1 Progettazione dei tipi di dato, delle strutture dati

Nome	Tipologia	Descrizione	Tipi / Campi / Valori
`client`	`struct`	Tipo di dato che descrive le caratteristiche di un cliente.	**ID**: `char[MAX_STRING_SIZE]` **Nome**: `char[MAX_STRING_SIZE]` **Cognome**: `char[MAX_STRING_SIZE]` **Tipo cliente**: `clientType` **Nome azienda:** `char[MAX_STRING_SIZE]` **Budget**: `unsigned int` **Data reg:** `time_t` **Tipo immobile ricercato**: `buildingType` **Da eliminare**: `bool`
`professional`	`struct`	Tipo di dato che descrive le caratteristiche di un professionista.	**ID**: `char[MAX_STRING_SIZE]` **Nome**: `char[MAX_STRING_SIZE]` **Cognome**: `char[MAX_STRING_SIZE]` **Area di competenza**: `char[MAX_STRING_SIZE]` **Numero di telefono**: `char[MAX_STRING_SIZE]`

			E-mail: char[MAX_STRING_SIZE] **Data reg.** time_t **Immobili venduti**: unsigned int **Da eliminare**: bool
potential	struct	Tipo di dato che descrive il potenziale di un professionista.	**ID**: char[MAX_STRING_SIZE] **Contenuto**: char[MAX_STRING_SIZE]
building	struct	Tipo di dato che descrive le caratteristiche di un immobile.	**ID**: char[MAX_STRING_SIZE] **Indirizzo**: char[MAX_STRING_SIZE] **Numero civico**: unsigned short int **Città**: char[MAX_STRING_SIZE] **Provincia**: char[MAX_STRING_SIZE] **Data reg.** time_t **Prezzo**: double **Proprietario**: char[MAX_STRING_SIZE] **N. telefono proprietario**: char[MAX_STRING_SIZE] **Tipo contratto:** contractType **Tipo di immobile**: buildingType **Da eliminare**: bool
clientType	enum	Tipo di dato che descrivi i tipi di cliente.	family / single / company / government
buildingType	enum	Tipo di dato che descrivi i tipi di immobile.	flat / duplex/ house/ farmhouse / attic
contractType	enum	Tipo di dato che descrive il tipo di contratto dell'immobile	rent / sale

MAX_USER_BUDGET	costante	Costante utilizzata per memorizzare il numero massimo di budget.	1000000000
DAY_IN_SECONDS	costante	Costante utilizzata per memorizzare il numero di secondi in un giorno.	86400
CLIENT_EXPIRE_DAYS	costante	Costante utilizzata per memorizzare il numero di giorni prima della scadenza.	30
CLIENTS_FNAME	costante/ file	File utilizzato per salvare le info sui clienti. Nome file memorizzato nella costante.	clients.dat
BUILDINGS_FNAME	costante/ file	File utilizzato per salvare le info sugli immobili. Nome file memorizzato nella costante.	buildings.dat
PROS_FNAME	costante/ file	File utilizzato per salvare le info sui professionisti. Nome file memorizzato nella costante.	professionals.dat
PTS_FNAME	costante/ file	File utilizzato per salvare le info sul potenziale relativo ad ogni professionista. Nome file memorizzato nella costante.	pros_potential.dat

SHOW_TESTS_MAIN_MENU	costante	Costante booleana che definisce se una scelta per poter avviare i test manualmente dal menu principale deve essere mostrata	true / false
RUN_TESTS_AT_STARTUP	costante	Costante booleana che definisce se avviare i test in maniera automatica all'apertura del software.	true / false
ENABLE_COLORS	costante	Costante booleana che definisce se abilitare i colori nell'output grazie all'utilizzo degli ANSI Escape Codes, introdotti nella build di Windows 10 16257.	true / false

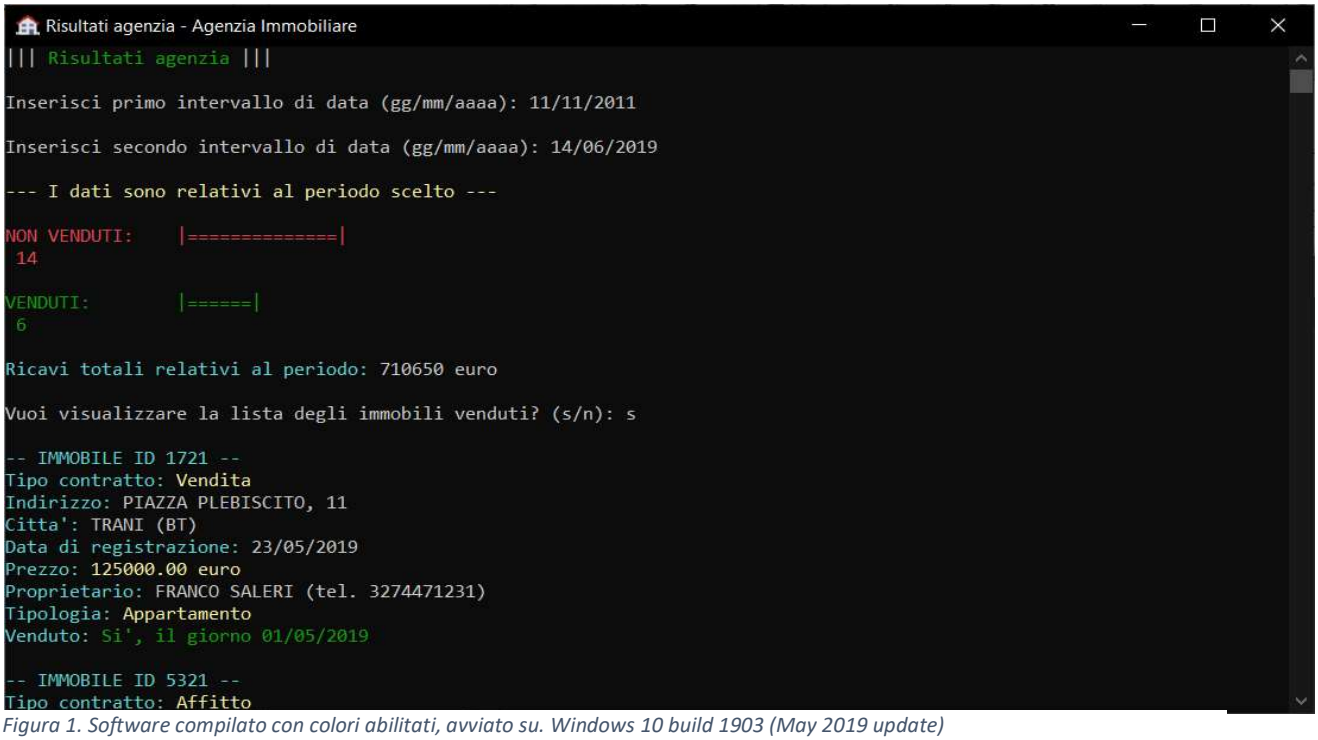

Figura 1. Software compilato con colori abilitati, avviato su. Windows 10 build 1903 (May 2019 update)

2.2 Progettazione delle librerie / funzioni

Libreria	Metodi

utils
 	
Contiene diverse estensioni relative alla gestione dell'output, all'acquisizione dell'input con validazione, etc... richiamate in altre funzioni.
	`void newLine();`
	`void clearScr();`
	
	`bool isNumber(char *str);`
	`bool isChar(char *str);`
	
	`void printFormattedDate(time_t epochTime);`
	`void printSectionName(char *string, bool isHome);`
	`time_t parseDate(char string[MAX_STRING_SIZE], bool errorCheck);`
	
	`void convertToUpperCase(char *s);`
	`void convertToLowerCase(char *s);`
	`bool strCompare(char *from, char *to);`
	`bool askConfirm();`
	
	`int readString(char *value, bool onlyAlpha, bool onlyNumbers);`
	`int readInteger();`
	`double readDouble();`
	
	`void resetColor();`
	`void setRedColor();`
	`void setYellowColor();`
	`void setGreenColor();`
	`void setCyanColor();`
	
	`void pause();`
	`void notFoundError();`
	`void setTitle(char *titleToSet);`
	`void dbEmptyError();`

sort
 	
Funzioni relative all'applicazione dell'ordinamento, tramite l'utilizzo dell'algoritmo di ordinamento per selezione, poiché i dati, potenzialmente, non sono già preordinati.
	`void sortPros(professional *pr, unsigned int rows);`
	`void sortClients(client *cl, unsigned int rows);`
	`void sortBuildings(building *bl, unsigned int rows);`

menus
Funzioni legate alla visualizzazione del menu principale e relativi sottomenu legati alle varie entità.
	`void mainMenu();`
	`int clientsMenu();`
	`int professMenu();`
	`int buildingsMenu();`

tests Funzione relativa all'avvio dei tests con CUnit.	`void startTests();`

file_utils
 	
Estensioni che facilitano la gestione e l'utilizzo dei file.
	`bool checkFile(FILE *filePtr);`
	`int countFileRows(char *name);`
	`void formattedDateToFile(FILE *filePtr, time_t *epochTime);`

agency Funzione relativa alla visualizzazione dei "risultati dell'agenzia".	`int resultsAgency(building *bl, unsigned int numBuildings);`

Professionisti	

files_pr
 	
Caricamento file, parsing dati, scrittura su file, controllo duplicati relativamente i professionisti.
	`int loadProsFile(professional *allPros, char *filename);`
	`void parseProsFile(FILE *filePtr, professional *pr);`
	`int checkDuplicatePros(professional *allPros, potential *allPts, int rows);`
	`int appendProToFile(professional *pr, char *filename);`
	`void rewriteProsToFile(professional *allPros, unsigned int rows, char *filename);`

files_pts
 	
Caricamento file, parsing dati, scrittura su file relativamente i potenziali dei professionisti.
	`void findPotential(char id[], potential *pr, unsigned int num_profess);`
	`void parsePotentialsFile(FILE *filePtr, potential *pr);`
	`int loadPotentialsFile(potential *allPts, char *filename);`
	`int appendPtsToFile(potential *pt, char *filename);`
	`void rewritePtsToFile(potential *allPts, unsigned int rows, char *filename);`

misc_pr
 	
Aggiunta (con relativo controllo duplicati) ed eliminazione record professionisti.
	`int addPro(professional *allPros, potential *allPts, unsigned int numRecords);`
	`int requestProDeletion(professional *allPros, potential *allPts, unsigned int numRecords);`
	`int deletePro(professional *allPros, potential *allPts, unsigned int numRecords, char *toDeleteID, char *prosFile, char *ptsFile);`
	`void checkDuplicateProID(potential *po, professional *allPros, unsigned int numRecords);`

req_pr
 	
Funzioni per richiesta dati da aggiungere in un nuovo record di tipo “professionista”.
	`void reqProCF(professional *pr);`
	`void reqProName(professional *pr);`
	`void reqProSurname(professional *pr);`
	`void reqProArea(professional *pr);`
	`void reqProPhone(professional *pr);`
	`void reqProEmail(professional *pr);`
	`void reqProSoldBuildings(professional *pr);`
	`void reqProPotential(professional *pr, potential *pt);`

show_pr Funzioni per visualizzazione dati professionisti.	`void showProData(professional *pr, potential *allPts, unsigned int numRecords);` `int showAllPros(professional *allPros, potential *allPts, unsigned int numRecords);`

tests_pr
 	
Test suite CUnit relativa ai professionisti.
	`int initSuitePros();`
	`int cleanSuitePros();`
	`void testProsFileParse();`
	`void testProDeletion();`

Clienti	

`files_cl`
 	
Caricamento file, parsing dati, scrittura su file, controllo duplicati relativamente i clienti.
	`int loadClientFile(client *cl, char *filename);`
	`void parseClientFile(FILE *filePtr, client *cl);`
	`int rewriteClientsToFile(client *cl, unsigned int rows, char *filename);`
	`int appendClientToFile(client *cl, char *filename);`
	`int checkDuplicateClients(client *cl, unsigned int rows);`

`misc_cl`
 	
Aggiunta (con relativo controllo duplicati) ed eliminazione record clienti.
	`int addClient(client *allClients, unsigned int numClients);`
	`void saveLocalDate(client *cl);`
	`bool checkIfUserExpired(time_t epochTime, char id[]);`
	`void initClientsArray(client *cl, unsigned int size);`
	`int requestClientDeletion(client *allClients, unsigned int numClients);`
	`int deleteClient(client *allClients, unsigned int numClients, char *toDeleteID, char *filename);`
	`void checkDuplicateClientID(client *cl, client *allClients, unsigned int numClients);`

`req_cl`
 	
Funzioni per richiesta dati da aggiungere in un nuovo record di tipo “cliente”.
	`void reqID(client *cl);`
	`void reqCF(client *cl);`
	`void reqPIVA(client *cl);`
	`void reqName(client *cl);`
	`void reqSurname(client *cl);`
	`void reqType(client *cl);`
	`void reqCompanyName(client *cl);`
	`void reqBudget(client *cl);`
	`void reqPropertyType(client *cl);`

`show_cl`
Funzioni per visualizzazione dati clienti.
	`void showClientType();`
	`void showClientData(client *cl, bool checkExpiration);`
	`int showAllClients(client *cl, unsigned int numClients);`

`tests_cl`
 	
Test suite CUnit relativa ai clienti.
	`int initSuiteClients();`
	`int cleanSuiteClients();`
	`void testClientsFileParse();`
	`void testClientDeletion();`

Immobili	

files_bl
 	
Caricamento file, parsing dati, scrittura su file, controllo duplicati relativamente gli immobili.
	`int loadBuildingsFile(building *bl, char *filename);`
	`void parseBuildingsFile(FILE *filePtr, building *cl);`
	`int rewriteBuildingsToFile(building *bl, unsigned int rows, char *filename);`
	`int checkDuplicateBuildings(building *bl, unsigned int rows);`
	`int appendBuildingToFile(building *bl, char *filename);`

misc_bl
 	
Aggiunta (con relativo controllo duplicati), modifica, eliminazione record immobili, aggiornamento stato di vendita.
	`void initBuildingsArray(building *bl, unsigned int size);`
	`int addBuilding(building *allBuildings, unsigned int numBuildings);`
	`int requestBuildingDeletion(building *allBuildings, unsigned int numBuildings);`
	`int deleteBuilding(building *allBuildings, unsigned int numBuildings, int toDeleteID, char *filename);`
	`int sellBuilding(building *allBuildings, unsigned int numBuildings);`
	`int editBuilding(building *allBuildings, unsigned int numBuildings);`
	`void checkDuplicateBuildingID(building *bl, building *allBuildings, unsigned int numBuildings);`

req_bl
 	
Funzioni per richiesta dati da aggiungere in un nuovo record di tipo “immobile”.
	`void genBuildingID(building *bl);`
	`void reqBuildingStreet(building *bl);`
	`void reqBuildingCity(building *bl);`
	`void reqBuildingProvince(building *bl);`
	`void reqBuildingPrice(building *bl);`
	`void reqBuildingOwner(building *bl);`
	`void reqBuildingPhone(building *bl);`
	`void reqBuildingType(building *bl);`
	`void reqContractType(building *bl);`
	`void reqBuildingSold(building *bl);`

search_bl
 	
Funzioni per ricerca immobili in base a criteri specifici.
	`void searchBuildingsForPrice(building *allBuildings, unsigned int numBuildings);`
	`void searchBuildingsForCity(building *allBuildings, unsigned int numBuildings);`
	`void searchBuildingsCtrType(building *allBuildings, unsigned int numBuildings);`
	`void searchBuildingsByType(building *allBuildings, unsigned int numBuildings);`
	`int searchBuilding(building *allBuildings, unsigned int numBuildings);`

show_bl
 	
Funzioni per visualizzazione dati immobili.
	`void showBuildingType();`
	`void showBuildingData(building *bl);`
	`int showAllBuildings(building *bl, unsigned int numBuildings);`
	`void printBuildingChoices();`
	`void showContractType(unsigned short int type);`
	`void printContractChoices();`

tests_bl
 	
Test suite CUnit relativa agli immobili.
	`int initSuiteBuildings();`
	`int cleanSuiteBuildings();`
	`void testBuildingsFileParse();`
	`void testBuildingDeletion();`

3. Codifica

È possibile consultare la documentazione prodotta da **Doxygen** aprendo il file **index.html** presente nella cartella **doxygen_docs/html**.

4. Testing

4.1 Definizione del Piano di Test

Codice Requisito	Codice Test	Nome	Descrizione Test	Eventuale Input	Risultato Atteso	Risultato Ottenuto
R01	1.1	Menu Iniziale	Scelta n.1	1	Caricamento del menu X	TRUE
R02	1.2	Menu iniziale	Scelta errata	100	Messaggio di Errore	ERRORE
R02	2.1	Caricamento Dati da File	File non esistente		Visualizzazione Messaggio di Errore e Creazione Nuovo File	ERRORE e Creazione del File
R02	2.2	Caricamento Dati da File	File Esistente		Caricamento dei File effettuato correttamente	TRUE
R03	3.1	Ordinamento File	Struttura in memoria		Ordinamento effettuato correttamente	TRUE
R04	4.1	Ricerca duplicati	Struttura in memoria	1	Trova eventuali duplicati e ne modifica il primo	TRUE
R05	5.1	Ricerca immobili	Località presente	barletta	Visualizza immobili	TRUE
R05	5.2	Ricerca immobili	Località non presente	barlstkt	Non restituisce immobili	ERRORE
R06	6.1	Riscrittura File	Struttura da scrivere in memoria		Riscrive il file correttamente	TRUE
R07	7.1	Aggiunta dati	Aggiunta corretta clienti (clienti in questo caso)	Dati cliente	Aggiunge correttamente il nuovo cliente	TRUE
R08	8.1	Controllo clienti scaduti	Presenza di clienti scaduti	s	Segnala il cliente scaduto e lo elimina	TRUE
R09	9.1	Visualizzazione dati	Dati presenti in memoria		Visualizza i dati correttamente	TRUE

4.2 Esiti del Piano di Test

Sono stati automatizzati tutti i test legati al caricamento, al parsing e all'eliminazione dei record relativamente ai file delle tre entità principali: **clients**, **buildings** e **professionals** grazie all'utilizzo di CUnit con la creazione delle suite di test e dei test methods necessari.

I test possono eseguiti in automatico ad ogni avvio del software, impostando il booleano **`RUN_TESTS_AT_STARTUP`** su **`true`**, in **consts.h**

Non sono state riscontrate criticità degne di nota dopo l'esecuzione dei test, sia manuali che automatici.

5. Commenti ed eventuali miglioramenti attuabili

È importante notare che i file, anche se con estensione .dat, sono stati utilizzati in maniera testuale seguendo lo standard "**comma-separated values**" (**CSV**).

Si è deciso di modificare l'estensione da .csv a .dat **per scoraggiare l'eventuale modifica dei file** da parte dell'utente, poiché il software si aspetta come input da file dei record codificati in maniera precisa (es. stringhe in maiuscolo, etc..).

Si sottolinea anche il fatto che, nel file relativo alle potenzialità dei professionisti (pros_potential.dat) **il delimitatore non è una virgola, ma invece è il simbolo di barra verticale** (detto anche pipe). Questo perché, essendo il potenziale del testo, ci sono molte più possibilità che venga utilizzata una virgola, generando quindi problemi nel parsing.

Inoltre, in futuro, è possibile ottimizzare l'utilizzo di memoria utilizzando l'allocazione dinamica con **malloc**(), invece del ricaricamento dei file in un'altra porzione di memoria. (per la lettura delle nuove modifiche)

Algoritmi principali

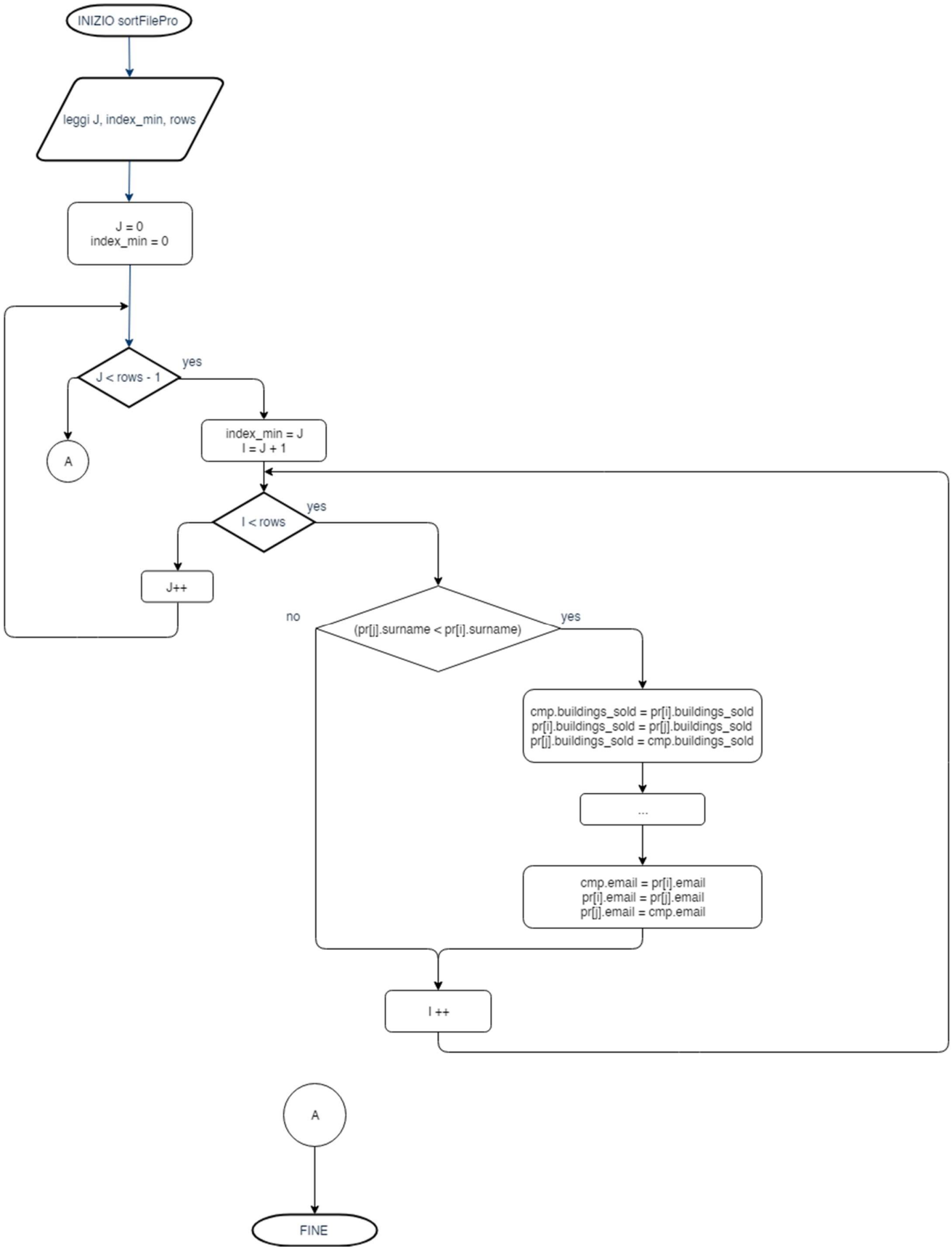

Figura 2. Ordinamento (selection sort)

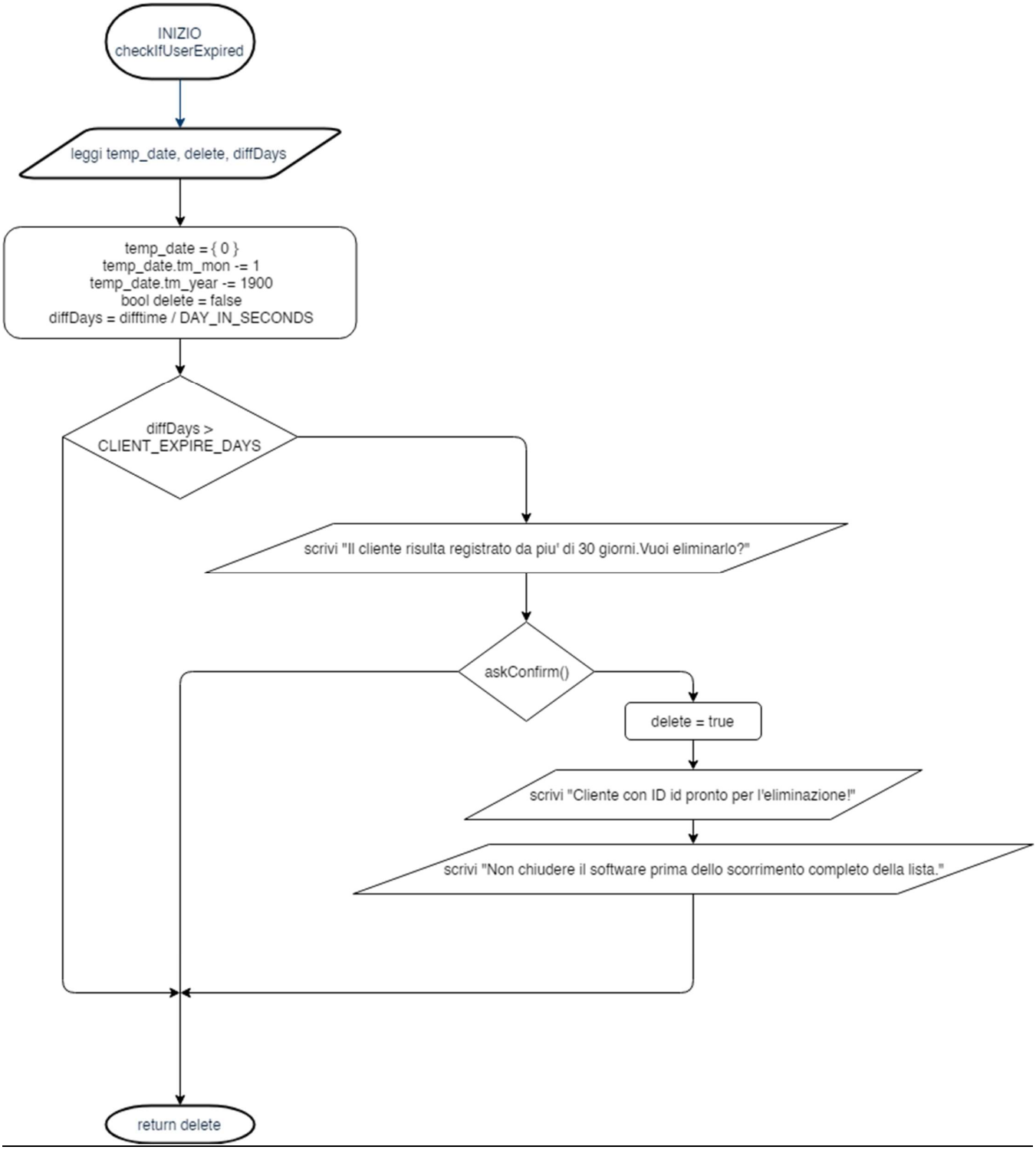

Figura 3. Controllo scadenza utente

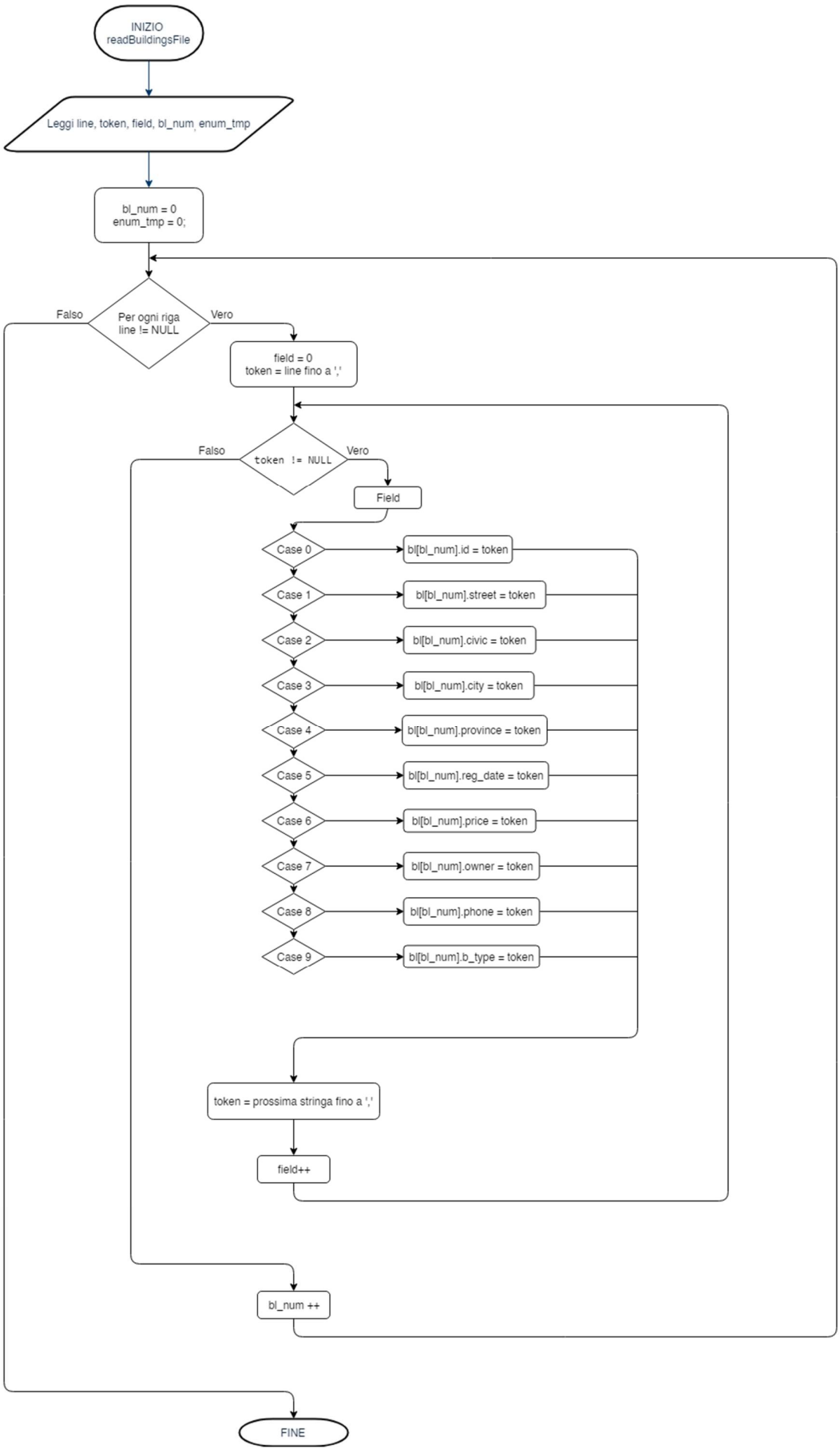

Figura 4. Lettura file (immobili in questo caso specifico)