// ^ id\_stanza

// \* oggetti

// % direzioni

**VARIABILI GLOBALI**:

keywords[MAX\_KEYWORD] (tabella\_simboli) tabella che contiene le parole chiave del gioco

ident[MAX\_KEYWORD] (tabella\_simboli) tabella che contiene il comando inserito in input dal giocatore, viene controllato che l’input corrisponda ad una delle parole chiave contenute in keywords

james (personaggio)

stanza\_attuale (ambiente)

**All’avvio del gioco**:

1. **inizializzazione\_parole\_chiave**: viene creata la tabella dei simboli (keywords), ovvero la tabella in cui sono presenti le parole chiave e i rispettivi simboli (es. prendi TKE, …)

inizia il ciclo da 0 fino alla lunghezza di ogni stringa prelevata, viene presa una riga per volta dal file “parole\_chiave.txt”, per ogni riga viene presa una parola per volta attraverso la funzione scan e inserita nella variabile di supporto token, se la variabile check ha valore 0, significa che il token è una parola (es. prendi) e viene inserita nella tabella keywords, se invece check ha valore 1, significa che la parola è già stata inserita nella tabella e bisogna quindi inserire il simbolo (es, TKE) e viene aumentato di uno il numero di parola contenute nella tabella perché una volta inserito il simbolo è stato inserito un intero comando nella tabella

2. **inizializzazione\_file**: viene copiato il contenuto di “storia\_originale.txt” (una riga per volta) nel file “storia.txt”

(storia\_originale.txt: contiene la storia originale (presa dall’analisi)

storia.txt: contiene la storia che sta giocando il giocatore (viene quindi modificata per esempio quando si prendono e lasciano degli oggetti))

3. **inizializzazione\_personaggio**: vengono assegnati i valori iniziali ai campi di personaggio:

- posizione: 1 (ovvero la stanza con id 1)

- punti\_vita: MAX\_HEALTH (il massimo, ovvero 100)

- inventario: attraverso la funzione *inizializzazione\_inventario* viene assegnato l’oggetto vuoto ad ogni slot dell’inventario (10 slot = MAX\_SLOTS)

4. **descrivere\_ambiente**: inizialmente, viene stampata la descrizione della stanza con id = 1, presa dal file “storia.txt”

**Durante il gioco**:

Fin quando i punti vita del giocatore sono maggiori o uguali a 0 e fin quando il giocatore si trova nella stanza 15 vengono analizzati ed eseguiti i comandi inseriti in input:

1. **analizzare\_comando**: riceve in input il comando inserito in input dall’utente; viene innanzitutto passato l’input alla funzione scan in modo da controllare una parola alla volta, viene poi controllato che la parola sia formata da soli caratteri minuscoli : se è così, la parola viene accettata e inserita nella tabella degli identificatori (tabella parole chiave) e passata in input alla funzione check\_parola\_chiave che verifica (dalla tabella keywords) se la stringa in input è una parola chiave o meno e restituisce il relativo simbolo (se si, restituisce il relativo simbolo, altrimenti il simbolo avrà valore EMP, cioè vuoto – no simbolo): se il simbolo è diverso da EMP, nella tabella ident viene inserito il simbolo corrispondente, altrimenti si inserisce come simbolo OBJ per indicare che si tratta di un oggetto generale e non di una parola chiave; se la parola non è accettata dall’automa, si finisce nello stato pozza e si assegna nella tabella ident 0 sia come parola e sia come simbolo.

2. **eseguire\_comando**: una volta controllato che il comando inserito in input dall’utente sia stato scritto nel modo corretto, si procede con l’esecuzione del comando che avviene in base al simbolo preso dalla tabella\_simboli ident e in base all’esito l’esecuzione andrà a buon fine oppure si genererà un errore;

se il simbolo inserito nella tabella\_simboli ident è uguale a:

- *TKE*: viene richiamata la funzione prendere\_oggetto che riceve in input la parola della tabella\_simboli ident scritta subito dopo il simbolo; avrà come esito 0 se la funzione è andata a buon fine, -8 se l’inventario è pieno, -7 se l’oggetto non è presente nella stanza (o non può essere preso);

- *LVE*: viene richiamata la funzione lasciare\_oggetto che riceve in input la parola della tabella\_simboli ident scritta subito dopo il simbolo; avrà come esito 0 se la funzione è andata a buon fine, -6 se l’oggetto non è presente nell’inventario;

- *VAI*: viene innanzitutto prelevata la parola scritta subito dopo il simbolo e in base a quale direzione si tratta viene assegnato un carattere alla variabile direction (n= nord, s= sud, o= ovest, e= est, a= su, b= giu, se la parola non è uguale a nessuna di quelle possibilità si avrà esito -2 (direzione non capita)); una volta fatto questo controllo, viene richiamata la funzione muovere\_giocatore; avrà come esito 0 se la funzione è andata a buon fine e, in questo caso, viene stampata la descrizione dell’ambiente con id uguale alla nuova posizione del personaggio, -3 se la direzione non è consentita, -11 se la porta è chiusa;

- *LOK*: bisogna prima fare un controllo preliminare sulla parola scritta subito dopo il simbolo LOK: se la parola è arma viene prima verificato se il giocatore sta impugnando quell’arma, se si vengono stampate le informazioni sull’arma impugnata dal giocatore, se esito = -10 non sta impugnando nulla; se la parola è uomo e la posizione del personaggio è 14 viene richiamata la funzione *indovino\_indovinello*; se la parola è diversa dalle precedenti opzioni, viene richiamata la funzione *guardare\_oggetto* che avrà esito 0 se andrà a buon fine, -7 se l’oggetto non è presente nella stanza, -9 se non è un oggetto\_guardabile;

- *OPN*: bisogna controllare se la parola scritta subito dopo il simbolo è uguale a porta, se è così bisogna fare il controllo delle direzioni, se la parola non è uguale a nessuna di quelle possibilità si avrà esito -2 (direzione non capita), altrimenti viene richiamata la funzione *aprire\_porta* che avrà esito 0 se è andata a buon fine, -13 se la porta è già aperta, -3 se la direzione in cui aprire la porta non è consentita,

-12 se non si ha la chiave giusta per aprire quella porta; se la parola non è porta, viene richiamata la funzione *aprire\_oggetto* che avrò esito 0 se andrà a buon fine, -13 se l’oggetto è già aperto, -12 se non si ha la chiave giusta per aprire quell’oggetto, -14 se non si ha un oggetto di tipo chiave;

- *EAT*: viene richiamata la funzione *mangiare\_cibo* che riceve in input la parola della tabella\_simboli ident scritta subito dopo il simbolo; avrò esito 0 se va a buon fine, -4 se il giocatore ha già i punti vita al massimo valore (il cibo non viene mangiato), -5 se l’oggetto non è di tipo cibo, -6 se l’oggetto non è presente nell’inventario;

- *SLT*: viene stampato il valore attuale dei punti vita del personaggio;

- *INV*: viene richiamata la funzione *stampare\_inventario* che stampa i nomi degli oggetti presenti nell’inventario (tranne quelli vuoti);

- *HEL*: viene stampato un elenco della sintassi da utilizzare per i comandi;

- *SLV*: viene richiamata la funzione *salvataggio\_partita* che riceve in input le informazioni relative al personaggio e genera due file output (storia\_salvataggio.txt che contiene la descrizione della storia, e personaggio\_salvataggio.txt che contiene le informazioni sui campi di personaggio (posizione, se ha l’arma, le informazioni sull’arma, inventario se gli id degli oggetti presenti in ogni slot sono diversi da 0)

- *LOA*: viene richiamata la funzione *caricare\_partita* che riceve in input il personaggio di cui si vogliono caricare le informazioni. Viene prelevata una riga alla volta da storia\_salvataggio.txt e copiata in storia.txt (il file su cui gioca il personaggio), viene inizializzato l’inventario, viene prelevata una riga alla volta dal file personaggio\_salvataggio e assegnato al giusto campo di personaggio, se la riga inizia con & si tratta dell’arma impugnata dal personaggio, se la riga inizia con \* allora si tratta di un oggetto che deve essere inserito nell’inventario

- *IMP*: viene richiamata la funzione impugnare\_arma che avrò esito 0 se va a buon fine, -10 se non stai impugnando nulla, -6 se l’oggetto che si vuole impugnare non è presente nell’inventario;

- se il simbolo non è uguale a nessuna delle precedenti opzioni si avrà esito -1 (non ho capito).

3. **descrivere\_ambiente**: questa funzione riceve in input il numero di una stanza

- viene prelevata una riga per volta attraverso la variabile *prelievo* e viene controllato il primo carattere di ogni riga:

- se è ^ allora quel carattere viene salvato nella variabile *token* (contiene una parola alla volta di ogni riga, ovvero *prelievo*)*,* e passato in input alla funzione *convert\_sequence\_char\_to\_int* perché potrebbe trattarsi del numero identificativo della stanza; successivamente quel numero viene assegnato a *stanza\_attuale.id* e vengono inizializzati gli altri campi di stanza\_attuale (il vettore oggetti, count\_obj, count\_dir, count\_look);

- fin quando il primo carattere di prelievo è diverso da \_ e se il primo carattere è diverso da /, &, $, \*, %, viene copiata la descrizione della stanza in *stanza\_attuale.descrizione;*

- se il primo carattere di prelievo è &, allora si tratta di un nemico e quindi vengono prelevati i dati dei punti vita, del danno e della durezza dell’arma (es. & 100 15 3), ogni numero viene convertito in intero, assegnato alla giusta variabile e successivamente vengono impostati i valori di hp\_flag, danno\_flag e durezza\_flag a true; viene poi assegnata l’arma al nemico e prelevata la successiva riga dal file: se inizia con $ allora si tratta di un combattimento e viene quindi stampata la descrizione e poi richiamata la funzione combattimento;

- se è / allora si tratta di un oggetto\_guardabile: viene assegnato false ad ogni variabile booleana (campo di oggetto\_guardabile) per indicare se il campo è già stato riempito oppure no, viene poi controllata ogni parola presente nella riga (*prelievo*) e in base a cosa si tratta viene assegnata al giusto campo di oggetto (perché noi conosciamo la struttura del file quindi sappiamo in che ordine vengono prelevati i dati); alla fine l’oggetto viene assegnato a *stanza\_attuale.looked*

- se è \* allora si tratta di un oggetto: viene assegnato false ad ogni variabile booleana (campo di oggetto) per indicare se il campo è già stato riempito oppure no, viene poi controllata ogni parola presente nella riga (*prelievo*) e in base a cosa si tratta viene assegnata al giusto campo di oggetto (perché noi conosciamo la struttura del file quindi sappiamo in che ordine vengono prelevati i dati); alla fine l’oggetto viene assegnato a *stanza\_attuale.elenco\_oggetti* e anche gli altri dati vengono assegnati ai giusti campi

- se è % allora si tratta dell’elenco delle direzioni possibili verso cui si può andare da quella stanza; ogni direzione viene poi assegnato all’array di direzioni *stanza\_attuale.direzioni\_possibili*

4. **gestione\_ambiente**: solo funzioni di accesso al tipo di dato ambiente

5. **gestione\_combattimento**: è una funzione che genera valori random, alla fine della funzione viene rimosso il nemico dal file attraverso la funzione *remove\_enemy\_from\_file* (nel codice)

6. **gestione\_nemico**: contiene le funzioni di accesso al tipo di dato nemico

7. **gestione\_oggetto**: contiene le funzioni di accesso al tipo di dato oggetto, cibo e mangiare\_cibo, arma e impugnare\_arma, oggetto\_apribile e aprire\_oggetto e aprire\_porta, oggetto\_guardabile;

- la funzione *mangiare\_cibo* riceve in input il personaggio e il nome del cibo che si vuole mangiare e avrà come output un esito che avrà valore 0 se tutto va a buon fine, -4 se i punti vita sono già al massimo valore, -5 se il nome dell’oggetto inserito in input non si riferisce ad un cibo, -6 se l’oggetto non è presente nell’inventario

- la funzione *impugnare\_arma* riceve in input il personaggio e il nome dell’arma che si vuole impugnare e avrà come output un esito che avrà valore 0 se tutto va a buon fine, -10 se non si sta impugnando nulla, -6 se l’oggetto non è presente nell’inventario

- la funzione *aprire\_oggetto* riceve in input il personaggio, il nome dell’oggetto che si vuole aprire e la stanza in cui è presente l’oggetto e avrà come output un esito che avrà valore 0 se tutto va a buon fine, -13 se l’oggetto è già aperto, -12 se non si ha la chiave giusta, -14 se non si ha un oggetto di tipo key

- la funzione *aprire\_porta* riceve in input il personaggio, il punto cardinale in cui è presente la porta che si vuole aprire e la stanza in cui è presente la porta e avrà come output un esito che avrà valore 0 se tutto va a buon fine, -13 se la porta è già aperta, -3 se la direzione non è consentita, -12 se non si ha la chiave giusta. Viene anche richiamata la funzione *open\_door\_from\_file* che riceve in input la direzione e la posizione del personaggio in modo da aggiornare il file storia.txt e impostare l’apertura di porta come true

8. **gestione\_parole\_chiave**: contiene la funzione *inizializzazione\_parole\_chiave*

9. **gestione\_personaggio**: contiene le funzioni di accesso al tipo di dato personaggio e inoltre le funzioni ordinare\_inventario, controllare\_inventario, prendere\_oggetto, lasciare\_oggetto, muovere\_personaggio, guardare\_oggetto, stampare\_inventario, inizializzazione\_inventario e inizializzazione\_personaggio

- la funzione *ordinare\_inventario* riceve in input il personaggio di cui si vuole ordinare l’inventario, cioè far sparire gli oggetti vuoti in modo da non avere slot vuoti

- la funzione *controllare\_inventario* riceve in input il personaggio di cui si vuole controllare se esiste una posizione vuota in cui non è inserito nessun oggetto

- la funzione *prendere\_oggetto* riceve in input il giocatore, il nome dell’oggetto che si vuole prendere e la stanza in cui si trova il pg e avrà come output un esito che avrà valore 0 se tutto è andato a buon fine, -8 se l’inventario è pieno, -7 se non c’è nessun oggetto con quel nome in quella stanza; l’oggetto viene poi rimosso dal file attraverso la funzione *remove\_object\_from\_file* (se l’oggetto è un libro accade una particolare cosa -> rivelazione passaggio segreto)

- la funzione *lasciare\_oggetto* riceve in input il giocatore e il nome dell’oggetto che si vuole lasciare e avrà come output un esito che avrà valore 0 se tutto è andato a buon fine, -6 se l’oggetto non è presente nell’inventario; l’oggetto viene poi inserito nel file attraverso la funzione *insert\_object\_into\_file*

- la funzione *muovere\_personaggio* riceve in input il personaggio da muovere, la stanza attuale in cui si trova il pg e la direzione che prende e avrà come output un esito che avrà valore 0 se tutto è andato a buon fine, -3 se la direzione inserita in input non è consentita, -11 se la porta è chiusa

- la funzione *guardare\_oggetto* riceve in input la stanza in cui si trova il personaggio e il nome dell’oggetto da guardare e avrà come output un esito che avrà valore 0 se andrà a buon fine, -7 se l’oggetto non è presente nella stanza, -9 se non è un oggetto\_guardabile

- la funzione *stampare\_inventario* riceve in input il personaggio di cui si vuole conoscere il contenuto dell’inventario e vengono stampati i nomi degli oggetti diversi dall’oggetto vuoto

10. **gestione\_tabella\_simboli**: contiene le funzioni di accesso del tipo di dato tabella\_simboli

11. **analizzatore\_lessicale**: contiene le funzioni relative all’automa (scan, per prendere una parola alla volta senza spazi e carattere nuova linea, analizzare\_comando per verificare se il comando è scritto in modo corretto, check\_parola\_chiave in cui viene controllato che la parola prelevata col token sia una parola chiave o meno (da keywords), eseguire\_comando per eseguire la funzione corretta in base al simbolo collegato al comando inserito in input)

12. **gestione\_storia**: contiene le funzioni descrivere\_ambiente, remove\_object\_from\_file, insert\_object\_into\_file, inizializzazione\_file, remove\_enemy\_from\_file, open\_door\_from\_file, indovino\_indovinello e convert\_sequence\_char\_to\_int

- la funzione *remove\_object\_from\_file* viene richiamata nel momento in cui viene preso un oggetto che quindi deve essere eliminato dal file poiché non esiste più nella stanza

- la funzione *insert\_object\_into\_file* viene richiamata nel momento in cui viene lasciato un oggetto che quindi deve essere inserito nel file in quanto viene aggiunto nella stanza insieme alle sue caratteristiche

- la funzione *remove\_enemy\_from\_file* viene richiamata nel momento in cui viene sconfitto un nemico che deve quindi essere eliminato dal file per non far ricominciare un nuovo combattimento se si rientra in quella stanza

- la funzione *open\_door\_from\_file* viene richiamata nel momento in cui viene aperta una porta perché bisogna impostare nel file che quella direzione ora è accessibile

- la funzione *convert\_sequence\_char\_to\_int* viene richiamata nel momento in cui viene letto un carattere dal file che deve essere convertito in intero per poter essere utilizzato

**CORRISPONDENZA OGGETTO- TIPO\_OGGETTO- SPECIAL\_OBJ**

La corrispondenza tra oggetto – tipo\_oggetto – special\_obj avviene in questo modo:

typedef struct {

cibo food;

arma weapon;

oggetto\_apribile key;

} special\_obj;

typedef struct {

int ID;

char nome[20];

bool afferrabile;

tipo\_oggetto tipo;

special\_obj oggetto\_speciale;

} oggetto;

typedef enum {

food = 'c',

weapon = 'a',

key = 'k'

} tipo\_oggetto;

**es. \*2 spada true a 23 5**

*\** : per identificare che si tratta di un oggetto

*2* : id dell’oggetto, oggetto.ID

*spada* : nome dell’oggetto, oggetto.nome

*true* : afferrabilità dell’oggetto, oggetto.afferrabile

*a* : tipo oggetto, oggetto.tipo (questo campo è di tipo tipo\_oggetto che a sua volta è di tipo enumerativo)

*23* : danno dell’arma, oggetto.oggetto\_speciale.weapon.danno (in questo modo, poiché l’oggetto in questione è un’arma, tramite il campo oggetto\_speciale possiamo assegnare i giusti campi)

*5* : durezza dell’arma, oggetto.oggetto\_speciale.weapon.durezza (in questo modo, poiché l’oggetto in questione è un’arma, tramite il campo oggetto\_speciale possiamo assegnare i giusti campi)

**es.**  **\*1 statue false**

\* : per indicare che si tratta di un oggetto

1 : id dell’oggetto, oggetto.ID

statue : nome dell’oggetto, oggetto.nome

false : afferrabilità dell’oggetto, oggetto.afferrabile

Gli altri campi di oggetto (tipo, oggetto\_speciale) non hanno valori perché si tratta di un oggetto comune