Strutture

```
typedef struct el
{
     int Info;
     struct el *Prox;
} elemlista;

typedef elemlista *listaint;
```

Lista di interi.
Per le griglie di attacco/difesa vale la seguente legenda per il campo info:
0: Cella non chiamata - sfondo acquatico;
1: Cella chiamata - nave colpita;
2: Cella chiamata - nave affondata;
3: Cella chiamata – acqua;
4: Cella non chiamata - contiene una nave.
Per le matrici di posizionamento ogni intero rappresenta l'Id di una nave.

```
typedef struct
{
     int NonAffondato;
     int Id;
} sommergibile;
```

Struttura per la nave sommergibile. NonAffondato viene inizializzato alla lunghezza della nave (2); se vale 0 la nave è stata distrutta. Id è un identificatore univoco della nave.

```
typedef struct
{
     int NonAffondato;
     int Id;
} incrociatore;
```

Struttura per la nave incrociatore. NonAffondato viene inizializzato alla lunghezza della nave (3); se vale 0 la nave è stata distrutta. Id è un identificatore univoco della nave.

```
typedef struct
{
     int NonAffondato;
     int Id;
} torpediniera;
```

Struttura per la nave torpediniera. NonAffondato viene inizializzato alla lunghezza della nave (4); se vale 0 la nave è stata distrutta. Id è un identificatore univoco della nave.

```
typedef struct
{
     int NonAffondato;
     int Id;
} portaereo;
```

Struttura per la nave portaerei. NonAffondato viene inizializzato alla lunghezza della nave (5); se vale 0 la nave è stata distrutta. Id è un identificatore univoco della nave.

```
typedef struct
{
     sommergibile *Sommergibili;
     int NumSommergibili;
     incrociatore *Incrociatori;
     int NumIncrociatori;
     torpediniera *Torpediniere;
     int NumTorpediniere;
     portaereo *Portaerei;
     int NumPortaerei;
     int NaviNonAffondate;
 } flotta;
```

Struttura per la flotta di un giocatore. Per ogni tipo di nave c'è un intero che rappresenta il numero di navi disponibile per il tipo (viene valorizzato in base alle impostazioni) ed un vettore di navi di quel tipo, di dimensioni pari all'intero corrispondente. L'intero NaviNonAffondate vale, inizialmente, quanto il numero totale di navi disponibili per il giocatore; quando vale 0 la flotta è stata distrutta.

```
typedef struct
{
     int NumGiocatori;
     int NumRighe;
     int NumColonne;
     int NumSommergibili;
     int NumIncrociatori;
     int NumTorpediniere;
     int NumPortaerei;
     int Audio;
     int PrimoGiocatore;
} settaggi;
```

Struttura per salvare le impostazioni. Viene memorizzato il numero di giocatori, il numero di righe e di colonne del campo ed il numero di navi per ogni tipo. Il flag Audio può valere 0 se il giocatore ha attivato gli effetti sonori o 1 se li ha disabilitati. Il flag PrimoGiocatore indica chi sarà il giocatore a fare la prima mossa durante una partita e può valere 0, 1, 2 o 3 (0: Random; 1: Giocatore1; 2: Giocatore2; 3: Manuale).

```
typedef struct
{
     char Nome[25];
     int Punteggio;
     flotta Flotta;
     listaint GrigliaAttacco;
     listaint GrigliaDifesa;
     listaint MatricePosizionamento;
} giocatore;
```

Struttura per salvare le informazioni di un giocatore. Consta del nome (univoco), del suo punteggio, della flotta, della lista dove tiene traccia dei suoi attacchi, di quella dove vengono mostrati i colpi che sta subendo e della lista che tiene traccia di dove sono posizionate le sue navi.

```
typedef struct
{
     giocatore *Giocatore1;
     giocatore *Giocatore2;
     giocatore *Vincitore;
} scontro;
```

Struttura utilizzata per gestire una partita. Contiene l'indirizzo dei due giocatori che si stanno sfidando e l'indirizzo del vincitore (che sarà uno dei due sfidanti).

```
typedef struct
{
     int PartitaDaGiocare;
     int TotalePartite;
     scontro *Match;
} calendario;
```

Struttura utilizzata per svolgere le partite del torneo. L'intero PartitaDaGiocare vale inizialmente 1 e viene incrementato, al termine di ogni partita, fino a raggiungere l'intero TotalePartite (che viene valorizzato a NumGiocatori – 1, vedi nota sotto). Il vettore Match ha dimensioni pari a TotalePartite; il vincitore dell'ultimo Match risulta essere il vincitore del torneo.
N.B.: Con 2 partecipanti il torneo sarà composto da 1 partita (finale).
Con 4 partecipanti il torneo sarà composto da 3 partite (2 semifinali + 1 finale).
Con 8 partecipanti il torneo sarà composto da 7 partite (4 quarti + 2 semifinali + 1 finale).
Con 16 partecipanti il torneo sarà composto da 15 partite (8 ottavi + 4 quarti + 2 semifinali + 1 finale).
Vale dunque l'equivalenza: TotalePartite = NumGiocatori – 1.

```
typedef struct
{
     giocatore *Giocatore1;
     giocatore *Giocatore2;
     int TurnoCorrente;
} salvataggio;
```

Struttura utilizzata per memorizzare le informazioni su una partita (non su un torneo) in svolgimento. Contiene gli indirizzi dei giocatori che si stanno sfidando e il turno del giocatore che ricomincerà a giocare.

```
typedef struct
{
     settaggi Impostazioni;
     giocatore *Partecipanti;
     calendario Calendario;
     int FlagPartitaInCorso;
     salvataggio PartitaInCorso;
} torneo;
```

Struttura per un torneo. Contiene le impostazioni, un vettore di giocatori (di dimensioni pari al numero di giocatori), un calendario e un flag che indica se una partita è in corso (e va terminata prima di poter proseguire con le altre del calendario). Se il flag vale 1, la struttura PartitaInCorso conterrà le informazioni necessarie per terminare il match in corso. Il flag può essere valorizzato a -1 se c'è un errore nell'apertura di un salvataggio o se viene premuto ESC durante il posizionamento delle navi (situazioni di errore).

Funzioni.h

```
int CentraCursore ( int );
```
Riceve in input un intero (es: lunghezza di una stringa) e restituisce un intero pari alla metà di (80-input). In questo modo una stringa di lunghezza pari a quella in input se stampata a partire dalla colonna numericamente uguale all'output risulterà centrata in una finestra di 80 colonne.

`void StampaPattern ( int );`
Stampa il 177esimo carattere ASCII (scelto per lo sfondo della finestra), un numero di volte pari all'intero in input.

`void ImpostaDimensioniFinestra ( void );`
Fissa le dimensioni della finestra a 25 righe e 80 colonne.

`void CreaImpostazioniDefault ( void );`
Crea il file ImpostazioniDefault.ini contenente una struttura impostazioni con i valori in default.

`void CambiaNomeFinestra ( void );`
Imposta il titolo della finestra a "Battaglia Navale".

`void ImpostaColore ( int );`
Riceve in input un numero corrispondente ad una coppia di colori (es: ImpostaColore(0x2F)): dopo la chiamata tutti i caratteri stampati saranno del colore scelto (secondo numero/lettera dopo "x"), su un sfondo del colore scelto (primo numero/lettera dopo "x") (nell'esempio saranno di colore bianco su sfondo verde). Elenco dei colori: 0=Nero; 1=Blu scuro; 2=Verde; 3=Verde acqua; 4=Bordeaux; 5=Viola; 6=Verde oliva; 7=Grigio chiaro; 8=Grigio; 9=Blu acceso; A=Verde limone; B=Azzurro; C=Rosso; D=Fucsia; E=Giallo; F=Bianco.

`void PosizionaCursore ( int, int );`
Posiziona il cursore nelle coordinate date in input.

`void NascondiCursore ( void );`
Nasconde il cursore nella finestra: in nessun punto si può vedere l'underscore lampeggiante.

`void MostraCursore ( void );`
Ritorna a mostrare il cursore nella finestra: si può vedere l'underscore lampeggiante.

`void FrecciaMenu ( int * );`
In base all'input da tastiera del giocatore sposta in su o in giù la freccia nel menu. Quando viene premuto INVIO chiama la funzione corrispondente alla voce dove si trova la freccia. La variabile intera segnala se deve essere ristampato il menu oppure no.

`void StampaSfondoMenu ( void );`
Stampa tutti gli elementi costanti della finestra menu.

`void AcquisisciNome ( char *, int, int );`
Acquisisce da tastiera un nome alfanumerico di massimo 24 caratteri e lo memorizza nella stringa in input. Le coordinate sono quelle da cui iniziare a mostrare il nome che si sta digitando.

`void CancellaImpostazioniTemporanee ( void );`
Cancella, se esistono, le impostazioni temporanee.

`int ControlloPresenzaFile ( char * );`
Controlla se il file in input è presente nella cartella dell'eseguibile.

NuovaPartita.h

`void NuovaPartita ( void );`
Inizializza le strutture dei giocatori, richiedendo anche i nomi dei partecipanti, e avvia una serie di partite, seguendo il calendario del torneo (i giocatori si sfidano a coppie: 1° vs 2°, 3° vs 4°, 5° vs 6°, ecc…) e secondo le impostazioni correnti.

`giocatore * AvviaPartita ( giocatore *, giocatore *, torneo * );`
Disputa la partita tra due giocatori, secondo le impostazioni contenute nel torneo in input (i giocatori si alternano in turni di gioco fin quando uno non vince – o viene premuto ESC -, viene eventualmente aggiornata la top ten a fine partita). Restituisce l'indirizzo del giocatore vincente.

`void StampaMirino ( int, int );`
Riceve in input le coordinate del cursore e stampa nelle coordinate adiacenti (non diagonalmente) delle frecce, che si configurano come un mirino.

`void CancellaMirino ( int, int );`
Riceve in input le coordinate del cursore e stampa nelle coordinate adiacenti (non diagonalmente) il carattere "spazio", che sovrascrive i caratteri preesistenti.

`void PulisciLogBox ( void );`
Pulisce la parte destra dello schermo, dove vengono mostrati dei messaggi per l'utente durante il posizionamento delle navi.

`void ProceduraPosizionamento ( giocatore *, torneo * );`
Fa posizionare al giocatore la sua flotta nella griglia di difesa grazie ad un mirino che si muove sul campo. Le impostazioni del torneo determinano quante navi il giocatore deve posizionare.

`void InizializzaFlotta ( flotta *, settaggi * );`
Inizializza i campi della struttura flotta in input secondo le impostazioni in input e alloca spazio per i vettori delle navi.

`void CreaCalendario ( torneo * );`
Prepara gli incontri della prima fase del torneo, abbinando giocatore 1 vs giocatore 2, giocatore 3 vs giocatore 4, ecc…

`void StampaSfondoCampo ( void );`
Stampa tutti gli elementi costanti della finestra di gioco.

`void StampaSfondoCampoPosizionamento ( void );`
Stampa tutti gli elementi costanti della finestra del posizionamento delle navi.

`char OttieniCarattere ( listaint, int, int, int );`
In base al valore presente nella posizione determinata dalle coordinate in input restituisce un determinato carattere e cambia il colore della finestra.

`void StampaQuadranteAttacco ( listaint, int, int, int );`
Stampa sul lato sinistro della finestra dei pattern colorati in relazione alla griglia in input.

`void StampaQuadranteDifesa ( listaint, int, int, int );`
Stampa sul lato destro della finestra dei pattern colorati in relazione alla griglia in input.

`void ScriviElementoListaInteri ( listaint, int, int );`
Scrive sulla lista in input, nella posizione in input, l'intero in input.

`int LeggiElementoListaInteri ( listaint, int );`
Legge dalla lista in input, nella posizione in input, un intero e lo restituisce in output.

`listaint InizializzaListaInteri ( settaggi * );`
*Crea una lista di interi pari al numero di righe * il numero di colonne delle impostazioni.*

`listaint InserisciInTesta ( listaint , int );`
Inserisce in testa alla griglia in input l'intero in input e restituisce la lista aggiornata.

`listaint InizializzaLista ( );`
Restituisce NULL.

`int CellaIsolata ( int, int, listaint, int, int );`
Controlla che le celle circostanti alla cella con in input siano tutte vuote.

`int CellaValidaPosizionamento ( int, int, int, giocatore *, torneo * );`
Conta quante celle vuote consecutive ci sono sulla riga e sulla colonna della cella passata in input (si conta a partire dalla cella in input) e controlla, in base alla lunghezza della nave (passata in input) se una nave può essere posizionata.

`void PosizionaNave ( int, int *, int *, giocatore *, torneo *, int *, int *, int *, int );`
Posiziona una nave nella griglia di difesa del giocatore in input, assicurandosi che le celle della nave siano tra loro adiacenti e che la nave non tocchi altre già presenti.

`int CellaAdiacente ( int, int, int, int, listaint, int *, int );`
Controlla che la posizione in cui si vuole posizionare una cella della nave sia affiancata verticalmente o orizzontalmente, in base al parametro di input sull'orientamento, da una cella della stessa nave già posizionata precedentemente.

`int ContaCelleVuote ( int, int, int, int, listaint );`
Conta quante celle vuote circondano la cella passata in input. Se il numero delle celle vuote è pari al (numero totale delle celle che circondano quella in input - 1) restituisce vero (è utilizzata per vedere se una cella è adiacente ad un'altra, quindi una cella di quelle adiacenti all'input deve essere occupata dalle celle della nave che si sono già posizionate).

`void StampaSchermataAttesa ( giocatore *, giocatore * );`
Stampa una schermata di attesa tra il turno di un giocatore ed il turno dell'altro giocatore.

`void CelebrazioneVincitoreTorneo ( giocatore *, settaggi * );`
Stampa una schermata per comunicare quale giocatore ha vinto il torneo.

`void CelebrazioneVincitorePartita ( giocatore *, settaggi * );`
Stampa una schermata per comunicare quale giocatore ha vinto la partita.

```
void RipristinaFlotta ( flotta * );
```
Ripristina i valori di tutti gli interi della flotta in input a come erano stati impostati da InizializzaFlotta().

```
void StampaCalendario ( torneo * );
```
Stampa una schermata che mostra il calendario del torneo, evidenziando i cammini dei giocatori vincenti.

```
void SalvaPartita ( torneo *, giocatore *, giocatore *, int, int );
```
Salva su file tutti i dati sul torneo in input ed un'eventuale partita in corso. Può sovrascrivere un file precedente o crearne uno nuovo.

```
int NuovoNome ( torneo *, int );
```
Controlla che il nome nella posizione ricevuta in input nel vettore dei partecipanti sia diverso da tutti gli altri nomi presenti in tale vettore. Restituisce 1 se è stato inserito un nuovo nome, 0 se è stato inserito un nome già presente.

```
void DeallocaLista ( listaint );
```
Dealloca, nodo per nodo, la lista in input.

CaricaPartita.h

```
void CaricaPartita ( void );
```
Richiede all'utente quale partita caricare tra quelle presenti. Continua il torneo scelto, in funzione delle configurazioni salvate sul file.

```
void RicreaTorneo ( torneo *, char * );
```
Legge le informazioni salvate sul file in input e le salva nel torneo in input. Ripristina il vettore di scontri con i vincitori delle partite già giocate. Riconfigura le partite da giocare.

```
int NomeSalvataggioValido ( char * );
```
Controlla che il nome in input sia valido (alfanumerico, e minore di 25 caratteri).

```
int IndicizzaSalvataggiValidi ( void );
```
Verifica quanti file con estensione .tsg ci sono nella cartella del gioco, li indicizza in un file di testo e restituisce il numero di file indicizzati.

Impostazioni.h

```
void Impostazioni ( void );
```
Mostra un elenco di impostazioni all'utente che potranno essere modificate.

```
void FrecciaImpostazioni ( void );
```
In base all'input da tastiera del giocatore sposta in su o in giù la freccia tra le voci delle impostazioni. Quando viene premuto INVIO permette la modifica della voce corrispondente alla riga dove si trova la freccia.

```
void StampaImpostazione ( int, int, int, int );
```
Stampa nelle coordinate in input l'impostazione ricevuta in input.

```
int ModificaImpostazione ( int, int, int, int );
```
Permette all'utente di variare l'impostazione desiderata (primo parametro), stampando i valori in coordinate predefinite (secondo e terzo parametro), scorrendo tra un range di valori dipendente da una variabile e restituisce il valore dell'impostazione modificata.

```
void OttieniImpostazioniIniziali ( settaggi* );
```
Memorizza nella struttura in input le impostazioni presenti nel file "NuoveImpostazioni.ini". Se il file non esiste, le impostazioni saranno lette da "DefaultSetting.ini". Se non esiste neanche quest'ultimo, il file sarà prima creato e poi da esso saranno lette le impostazioni.

```
void StampaSfondoImpostazioni ( void );
```
Stampa tutti gli elementi costanti della finestra impostazioni.

TopTen.h

```
void TopTen ( void );
```
Mostra la topten attuale del tuo gioco.

```
void StampaSfondoTopTen ( void );
```
Stampa tutti gli elementi costanti della finestra top ten.

```
void StampaPunteggi ( void );
```
Stampa l'elenco dei giocatori presenti nella top ten ed i corrispondenti punteggi.

```
int NuovoRecord ( int );
```
Riceve in input il punteggio di un giocatore e restituisce 0 se è minore del punteggio più basso della top ten, 1 se è maggiore..

```
void CreaTopTenDefault ( void );
```
Crea il file TopTen.txt con dei punteggi in default.

```
void AggiornaTopTen ( giocatore * );
```
Memorizza il nome e il punteggio del giocatore ricevuto in input nella giusta posizione nel file Top Ten.

Esci.h

```
void Esci ( void );
```
Stampa un messaggio di saluto.

Istruzioni.h

```
void Istruzioni ( void );
```
Mostra un menu all'utente che potrà scegliere se visualizzare le regole del gioco, i parametri delle impostazioni o i comandi del gioco.

```
void FrecciaIstruzioni ( void );
```
In base all'input da tastiera del giocatore sposta in su o in giù la freccia tra le voci delle istruzioni. Quando viene premuto INVIO avvia la visualizzazione della voce corrispondente alla riga dove si trova la freccia.

```
void StampaSfondoIstruzioni ( void );
```
Stampa tutti gli elementi costanti della finestra istruzioni.

```
void RegoleGioco ( void );
```
Stampa le regole del gioco

```
void ParametriImpostazioni ( void );
```
Stampa informazioni su alcuni parametri delle impostazioni.

```
void ComandiGioco ( void );
```
Stampa informazioni sui comandi utilizzabili durante il gioco.