

**Dipartimento di Scienze e Innovazione Tecnologica**

**CORSO DI LAUREA IN INFORMATICA**

**PROGETTO DI: Reti 1**

**Progetto: Traccia A**

ANNO ACCADEMICO 2016/2017

Docente: Maurizio Matteo Munafò Gruppo 4.

Riccardo Pidello

Matricola: 10033872

Carlo Daniele Giavazzi

Matricola: 20013889

**Server:**

**Variabili Globali**:

* MESSAGE\_WELCOME: Stringa di benvenuto che il server invia al Client una volta stabilita la connessione.
* MESSAGE\_MINOR: Stringa inviata al Client nel caso il numero ricevuto sia minore del numero da indovinare.

* MESSAGE\_MAJOR: Stringa inviata al Client nel caso il numero ricevuto sia maggiore del numero da indovinare.

* MESSAGE\_CORRECT: Stringa inviata per avvisare di aver indovinato il numero.

* MESSAGE\_ERROR: Stringa inviata per segnalare di aver ricevuto un numero non valido.
* MESSAGE\_ERROR\_TENTATIVI: Stringa inviata per segnalare al Client di aver esaurito i tentativi per indovinare il numero.
* TENTATIVI­\_MAX: costante per indicare il numero massimo dei tentativi oltre cui il server chiude la connessione con un opportuno messaggio.

**Funzione main di tipo int**:

Variabili:

* random: variabile in cui risiede il numero da indovinare generato casualmente.

* tentativo: contatore dei tentativi.
* valore\_client: variabile in cui risiede il valore inserito dall’utente.
* end: flag per terminare il ciclo di identificazione del valore.
* buffer: stringa dove salvare il valore inviato dal client.
* compare: stringa di appoggio per rilevare eventuali caratteri invalidi.

Esecuzione:

1. Il server si mette in ascolto sulla porta passata come argomento.
2. Tramite la funzione socket stabilisce una connessione con client e salva il descrittore in “Simple\_socket”, esegue un controllo su “Simple\_socket” per capire se la connessione è stabilità, in caso contrario stampa un messaggio di errore e torna in ascolto al passo 1, altrimenti invia il messaggio di benvenuto contenuto in MESSAGE\_WELCOME al client.
3. Modifica il seme della funzione random e salva nella variabile random il numero casuale compreso da 1 a 100 da indovinare.
4. Inizia il ciclo while che sarà terminato con l’apposito flag “end”, la prima cosa che fa il ciclo è svuotare tramite la funzione memset le due stringhe “buffer” e “compare”, tramite una read legge il valore mandato dal client e lo salva in buffer dopo di che effettua un controllo per evitare la lettura del comando di invio da tastiera.
5. Il contenuto di buffer è quindi trasformato in intero tramite la funzione atoi e salvato in “valore\_client”, il cui contenuto viene scritto in “compare”, tramite la strcmp le due stringhe “buffer” e “compare” sono confrontate e il risultato inserito in un controllo per evitare di interpretare lettere come numeri, oltre a questo viene controllato che il numero rispetti gli estremi forniti all’utente, il non rispetto di una qualsiasi di queste condizioni termina il ciclo e invia il messaggio di errore contenuto in “MESSAGE\_ERRORE”, e riporta il server in ascolto al passo 1.
6. Viene controllato l’esito della read e il flag “end” il non superamento del controllo porta alla terminazione del ciclo e alla chiusura della comunicazione rimettendo il server in ascolto al passo 1, superato ciò viene effettuato un controllo su la quantità di tentativi utilizzati, e se anche questo controllo viene superato si può accedere ai controlli per determinare la correttezza del tentativo o nel caso di fallimento la tipologia di aiuto da inviare.
7. Se il valore inserito non è uguale a quello contenuto in random si passa alle casistiche: se il valore è maggiore di 0 e minore di “random” allora si invia il contenuto di “MESSAGE\_MINOR”, se il numero inserito è maggiore di “random” e minore o uguale a 100 allora si invia il contenuto di “MESSAGE\_MAJOR”, infine viene incrementato il numero di “tentativi”.
8. Il ciclo si ripete fino ad indovinare oppure fino all’utilizzo di tutti i tentativi messi a disposizione, fatto questo il server si rimette in ascolto.

**Client:**

**Funzione split\_message di tipo int:**

Variabili:

* source: il messaggio ricevuto dal server.

* dest\_keyword: puntatore alla stringa di destinazione della parola chiave.
* dest\_message: puntatore alla stringa di destinazione del messaggio.
* n: dimensione della stringa.
* split: carattere di split.
* terminazione: carattere di terminazione.
* pointer\_read: indice da cui partire a scansionare source.
* split\_counter: contatore che salva l'indice in cui si è effettuato lo split.

Esecuzione:

* 1. Effettua a cascata controlli sui puntatori del messaggio del server, e delle stringhe di destinazione, in caso puntino a NULL ritorna l’opportuno intero,

alla dimensione n della stringa viene sommato l’indice da cui partire a leggere il messaggio del server.

* 1. Inizia il ciclo che continuerà fino a che l’indice da cui leggere il messaggio sarà minore della dimensione n, viene eseguito un controllo sull’indice da leggere, in caso corrisponda al carattere di terminazione viene incrementato il contatore dell’indice da cui leggere e termina.
  2. Si effettua un controllo su “splitted” che avvisa di aver splittato il messaggio del server e si controlla di non incontrare il carattere di split in caso il controllo venga superato l’elemento del messaggio di indice “pointer\_read” viene copiato nella stringa di destinazione all’indice “pointer\_keyword ” +1 e viene incrementato il contatore di split.
  3. Se il prima citato controllo non viene superato si controlla se abbiamo già splittato un messaggio, se non è così mettiamo “splitted” a 1, incrementiamo “split\_counter” e inseriamo il carattere di terminazione in “dest\_keyword”, all’indice “pointer\_read”.
  4. Se non abbiamo ancora splittato il messaggio il carattere nell’indice “pointer\_read”, della stringa source viene copiato in “dest\_message” all’indice “pointer\_redad + 1”.
  5. Se al passo 3) abbiamo incontrato il carattere di terminazione incrementiamo “pointer\_read” e ritorniamo “split\_counter”.

**Funzione main di tipo int:**

Variabili:

* pointer\_read: indice per leggere il messaggio del server.
* return\_status: utilizzata per registrare la condizione di uscita delle funzioni.
* buffer e buffer\_in; stringhe usate per i messaggi del server.
* keyword;
* message;

Esecuzione:

* 1. Effettuati i controlli sul numero di argomenti passati all’applicativo stabilisce la connessione al server.
  2. Effettua una “read” su buffer per leggere il messaggio di benvenuto, e tramite la sopra presentata funzione di split spezza la parola chiave dal messaggio contenuto e ritorna il punto da cui iniziare la prossima lettura.
  3. Effettua un controllo su “keyword”, che dopo l’esecuzione della funzione di split contiene il codice del messaggio ”OK”, in caso sia andato tutto bene.
  4. Superato il controllo entra nel ciclo while che viene terminato con l’apposito flag “end” se posto a 1, il client stampa a video le istruzioni all’utente ed esegue a cascata controlli per evitare caratteri non consentiti.
  5. Tramite una write si invia la risposta contenente il tentativo al server, dopo di che sempre tramite la funzione split viene estrapolata la risposta del server, e tramite controlli si rientra in diversi rami a seconda dell’errore del tentativo oppure se si ha indovinato o se i tentativi massimi sono stati superati .
  6. Il client risponde guidando l’utente alla soluzione o lo avvisa di aver terminato il gioco senza riuscire ad indovinare, fatto questo chiude la connessione sulla porta del server.