**GUESS MAGIC NUMBER GAME USER GUIDE**

**Attention**

Please read this manual thoroughly before use my product

* DO NOT input double number or character when you are guessing because it will break the program.
* Follow the feedback of program when you are playing.

**Introduction**

**The game’s Objective**: Generates a random number with 4 digits. The player has to guess what the number is. After the player guess the correct number, it will calculate the lucky ratio of player and show five players had highest lucky ratio in history.

**Technical Details**: I chose C as the programming language because it is very fast and versatile. Furthermore, it has some useful library I can use to implement my program. The game will run base on Visual Studio Code Software.

**Functionality Overview**: These are some function that I use to implement the game.

* int random(minN, maxN): the function has two argument. Value return is a random number with it scope [minN, maxN].
* void insertBeginList(head, data): the function insert new player into the head of linked list.
* void insertAfter(node\_player, data): the function insert new player after a node player.
* void insertSorted(head, data): the function insert new player according to the decrease order of lucky ratio.
* void printList(head): the function displays five highest players in history.
* void deallocate(head): the function free the memory.
* int \* arrayNumber(number): the function convert argument number to an array with 4 elements is 4 digits.

**Instruction**

**Step 1**: The player starts the game by double left mouse click into application “Mock\_Le\_Minh\_Huy”.

**Step 2**: The program automatic generates a random magic number with 4 digits.

**Step 3**: The program will request the player enter player’s name. They can start guess the number after they enter a valid name.

**Step 4**: The program compares guess number to magic number. And show the result on the screen. If there are matched number, the program prints out the result and the magic number (which guested number in correct position, the remaining are ‘-‘). Example: - 4 - 7. Loop until the player guess a number that all digits is in correct position.

**Step 5**: After all digits is in correct position. The program calculates lucky ratio of player. Save data of player in structural form and then write this data to file List\_player.bin.

**Step 6**: The program gets all data in file List\_player.bin and insert into a linked list according to decrease order of lucky ratio. Loop until all data in file List\_player.bin has been loaded.

**Step 7**: After loop, the program displays five highest player in history.

**Step 8**: The program asks player: “Do you want to replay?(y/n)”

**Step 9**: If the player type ‘y’. The program will start at Step 2 again. If the player type ‘n’, they will exit the game.

**Link Git**

<https://github.com/huyuser1591999/Project_Final.git>

**Source Code**

/\* C program Guess Magic Number Game\*/

#include <stdio.h

#include <stdlib.h>

#include <string.h>

#include <time.h>

// Declare struct with name "Player" and use typedef to defining an alias "player"

typedef struct Player

{

    int magic\_number;

    float lucky\_ratio;

    char name[24];

    struct Player\* next;

} player;

// Function check the filePointer is valid

void checkFilePointer(FILE\* filePointer);

// Function check the pointer of struct player is valid

void checkPlayerPointer(player\* pointer);

// Function add a new player in the begining of list

void insertBeginList(player\*\* root, int magic\_number, float lucky\_ratio, char\* name);

// Function add a new player after a player

void insertAfter(player\* node, int magic\_number, float lucky\_ratio, char\* name);

// Function add a new player according to the decrease order of the lucky ratio

void insertSorted(player\*\* root, int magic\_number, float lucky\_ratio, char\* name);

// Function display list of player, with 5 highest player

void printList(player\*\* root);

// Function free the memory of each player in the list

void deallocate(player\*\* root);

// Function return a random number with minN <= number <= maxN

int random(int minN, int maxN);

// Function return a pointer point to an array. This array is array of 4 digits number.

// Each element is 1 digit. "number" argument is the number that want to convert to array

int\* arrayNumber(int number);

// Main function of program

int main()

{

    // Declare variable to store option of player

    char options;

    // Declare variable to store name of player

    char name[24];

    // Declare variable to store magic number when program generates it

    int magic\_number;

    // Declare variable to store guess number from player

    int input\_number;

    // Declare a pointer point to an array of magic number 4 digits. Each element is a digit

    int\* arr\_magic\_num;

    // Declare a pointer point to an array of guess number 4 digits. Each element is a digit

    int\* arr\_guess\_num;

    // Declare variable to store the lucky ratio of the player

    float lucky\_ratio;

    // Declaring file pointer variable to store the value returned by fopen

    FILE\* filePointer;

    // Label "Replay".

    Replay:

    // Declare a head of linked list player

    player\* root = NULL;

    // Declare variable 'arr\_icon' and initialize it with "----" to display on screen when the game start

    char arr\_icon[4] = {'-','-','-','-'};

    // Declare variable to check that is 4 digit of magic number equal guess number

    int is\_full = 0;

    // Declare variable to count times when player guest

    int quantity\_guest = 0;

    // Dynamic allocated memory for pointer point to struct

    player\* my\_player = (player\*)malloc(sizeof(player));

    // Use function checkPlayerPointer() to check is pointer valid

    checkPlayerPointer(my\_player);

    // Dynamic allocated memory for pointer point to struct to get data from file

    player\* file\_player = (player\*)malloc(sizeof(player));

    // Use function checkPlayerPointer() to check is pointer valid

    checkPlayerPointer(file\_player);

    // Assigning value random of magic number to variable magic number using function radom()

    magic\_number = random(1000, 9999);

    // Using function arrayNumber() to assign each digit of magic number into array of magic number

    arr\_magic\_num = arrayNumber(magic\_number);

    printf("\n\t\tGUESS MAGIC NUMBER GAME\n");

    // Label "nameAgain"

    nameAgain:

    // Request the player enter name

    printf("\nEnter your name: ");

    scanf(" %[^\n]s", name);

    // Use for loop to check that if there are some invalid character in player name

    for (int i = 0; i < strlen(name); i++)

    {

        // Condition true when name has some invalid character

        if((name[i] < 65 && name[i] != 32) || (name[i] > 90 && name[i] < 97) || (name[i] > 122))

        {

            printf("\nInvalid character in your name!\nType again.\n");

            // Go to label "nameAgain" and request player enter name

            goto nameAgain;

        }

        else{

            // do nothing

        }

    }

    // Check that the name input from user is valid

    if(strlen(name) > 24)

    {

        printf("\nInvalid length name!\nType again.\n");

        // Go to label "nameAgain" and request player enter name

        goto nameAgain;

    }

    else{

        // do nothing

    }

    // Loop program when the player guess the magic number

    do

    {

        // Display 4 digits with content "----". If the player guess correct, it will be changed

        printf("\nMagic number: ");

        for(int i = 0; i < 4; i++)

        {

            printf("%c", arr\_icon[i]);

        }

        // Label "guessAgain"

        guessAgain:

        // Request the player guess the magic number

        printf("\n\nGuess integer number: ");

        scanf(" %d", &input\_number);

        // Increase this variable each times the player guess

        quantity\_guest++;

        // Check that the guess number is valid or invalid

        if(input\_number < 1000 || input\_number > 9999)

        {

            printf("\nInvalid guess number!\nThe number has 4 digits. Guess again.\n");

            // Go to label "guessAgain" and request player enter guess number

            goto guessAgain;

        }

        else{

            // do nothing

        }

        // Using funtion arrayNumber() to assign each digit of guess number into array of guess number

        arr\_guess\_num = arrayNumber(input\_number);

        // Reset is\_full every time the player has to guess again

        is\_full = 0;

        // Using for loop to get each digit in array of magic number and guess number, and then compare these

        for(int i = 0; i < 4; i++)

        {

            // Compare each digit

            if(arr\_guess\_num[i] == arr\_magic\_num[i])

            {

                // Assign each digit of guess number if condition of if is True. Need plus 48 and cast type char base on ASCII table

                arr\_icon[i] = (char)(arr\_guess\_num[i] + 48);

                // Increase 1 value if 1 digits has assigned.

                is\_full++;

            }

            else{

                //do nothing

            }

        }

        // Check that is 4 digits has assigned enough.

        if(is\_full == 4)

        {

            // Calculate the lucky ratio of player

            lucky\_ratio = ((float)1 / quantity\_guest) \* 100;

            printf("\nCorrect number!\nYour magic number is %.4d.\n", magic\_number);

            // Free the memory

            free(arr\_magic\_num);

            free(arr\_guess\_num);

            // Assign data of player into struct player

            my\_player->lucky\_ratio = lucky\_ratio;

            my\_player->magic\_number = magic\_number;

            strcpy(my\_player->name, name);

            // Opening the file in write mode on binary file. Use mode "ab" to continues store data of player

            filePointer = fopen("List\_Player.bin", "ab");

            // Check if the filePointer is valid

            checkFilePointer(filePointer);

            // Write data to binary file

            fwrite(my\_player, sizeof(player), 1, filePointer);

            // Close file and save

            fclose(filePointer);

            // Free the memory

            free(my\_player);

            // Open file for reading and get data of previous player

            filePointer = fopen("List\_Player.bin", "rb");

            // Check if the pointer is valid

            checkFilePointer(filePointer);

            // Using while loop to get each data of player from file "List\_Player.bin"

            while (fread(file\_player, sizeof(player), 1, filePointer) == 1)

            {

                // Assign data to pointer of struct "my\_player"

                lucky\_ratio = file\_player->lucky\_ratio;

                magic\_number = file\_player->magic\_number;

                strcpy(name, file\_player->name);

                // Insert data from pointer "my\_player" into linked list according to the increase order of lucky ratio

                insertSorted(&root, magic\_number, lucky\_ratio, name);

            }

            // Close the file

            fclose(filePointer);

        }

        else{

            // do nothing

        }

    } while (is\_full != 4); // Check that is 4 digits has assigned enough. If not, continues do{} while() loop

    // Show the lucky ratio history of the program, with 5 highest player

    printf("\n---------------------------------------------------------------------------------\n");

    printf("\n\t\tThe lucky ratio history of the player\n");

    // Using function printList() to display each player in the list

    printList(&root);

    printf("\n---------------------------------------------------------------------------------\n");

    printf("\nDo you want to replay?(y/n) ");

    scanf(" %c", &options);

    if(options == 'y')

    {

        goto Replay;

    }

    else{

        // do nothing

    }

    // Free the memory of the list player

    deallocate(&root);

    return 0;

}

// Function check the filePointer is valid.

void checkFilePointer(FILE\* filePointer)

{

    // Check that is filePointer point to NULL

    if(filePointer == NULL)

    {

        // Display warning

        perror("Error opening file.\n");

        // Warning exit program failure because filePointer point to NULL

        exit(1);

    }

    else {

        //do nothing

    }

}

// Function check the pointer of struct player is valid

void checkPlayerPointer(player\* pointer)

{

    // Check that is pointer point to NULL

    if (pointer == NULL)

    {

        // Display warning

        printf("Memory not allocated.\n");

        // Warning exit program failure because pointer point to NULL

        exit(1);

    }

    else {

        //do nothing

    }

}

// Function add a new player in the begining of list

void insertBeginList(player\*\* root, int magic\_number, float lucky\_ratio, char\* name)

{

    // Dynamic allocate memory for new\_player

    player\* new\_player = (player\*)malloc(sizeof(player));

    // Use function checkPlayerPointer() to check is new\_player is valid

    checkPlayerPointer(new\_player);

    // Assign data from argument into node new\_player

    new\_player->lucky\_ratio = lucky\_ratio;

    new\_player->magic\_number = magic\_number;

    strcpy(new\_player->name, name);

    new\_player->next = \*root;

    // Assign head of list is new\_player

    \*root = new\_player;

}

// Function add a new player after a player

void insertAfter(player\* node, int magic\_number, float lucky\_ratio, char\* name)

{

    // Dynamic allocate memory for new\_player

    player\* new\_player = (player\*)malloc(sizeof(player));

    // Use function checkPlayerPointer() to check is new\_player is valid

    checkPlayerPointer(new\_player);

    // Assign data from argument into node new\_player

    new\_player->lucky\_ratio = lucky\_ratio;

    new\_player->magic\_number = magic\_number;

    strcpy(new\_player->name, name);

    // Assign address of node next to new\_player is address of node next to node pass to function as an argument

    new\_player->next = node->next;

    // Assign address of node next to node pass to function as an argument is address of new\_player

    node->next = new\_player;

}

// Function add a new player according to the decrease order of the lucky ratio

void insertSorted(player\*\* root, int magic\_number, float lucky\_ratio, char\* name)

{

    // Check that if the list is empty or lucky\_ratio of player head is smaller than lucky\_ratio of new\_player

    if(\*root == NULL || (\*root)->lucky\_ratio <= lucky\_ratio)

    {

        // Use function insertBeginList() add a new player in the begining of list

        insertBeginList(root, magic\_number, lucky\_ratio, name);

        // Return function main()

        return;

    }

    // Assign address of player head list to pointer curr

    player\* curr = \*root;

    // Use while loop to loop throught the list. If curr->next == NULL, curr is the last player in list

    while (curr->next != NULL)

    {

        // Check that if there is a node player had lucky\_ratio smaller than lucky\_ratio of new\_player

        if(curr->next->lucky\_ratio <= lucky\_ratio)

        {

            // Exit while loop

            break;

        }

        // Assign address of current player is address of next player

        curr = curr->next;

    }

    // Use function insertAfter() add a new player after current player

    insertAfter(curr, magic\_number, lucky\_ratio, name);

}

// Function display list of player, with 5 highest player

void printList(player\*\* root)

{

    // Assign address of player head list to pointer curr

    player\* curr = \*root;

    // Declare variable "count" to get 5 highest player

    int count = 0;

    // Use while loop to loop throught the list. If curr == NULL, the program has loop throught all player of list

    while (curr != NULL)

    {

        // Display 5 highest player

        printf("\nMagic Number: %d\tPlayer: %-20s\tLucky ratio: %.2f %%\n", curr->magic\_number, curr->name, curr->lucky\_ratio);

        // Assign address of current player is address of next player

        curr = curr->next;

        // Increase variable "count" by 1 after a player has been displayed

        count++;

        // Check that if the program has display enough 5 player

        if(count == 5)

        {

            // Return function main()

            return;

        }

        else{

            // do nothing

        }

    }

}

// Function free the memory of each player in the list

void deallocate(player\*\* root)

{

    // Assign address of player head list to pointer curr

    player\* curr = \*root;

    // Use while loop to loop throught the list. If curr == NULL, the program has loop throught all player of list

    while (curr != NULL)

    {

        // Declare another pointer to point the address of current player

        player\* delete\_player = curr;

        // Assign address of current player is address of next player

        curr = curr->next;

        // Free the memory of node player

        free(delete\_player);

    }

    // After while loop. Assign head of list is NULL. List is empty

    \*root = NULL;

}

// Function return a random number with minN <= number <= maxN

int random(int minN, int maxN)

{

    // Initialize random number generator

    srand((int)time(0));

    return minN + rand() % (maxN + 1 - minN);

}

// Function return a pointer point to an array. This array is array of 4 digits number.

// Each element is 1 digit. "number" argument is the number that want to convert to array

int\* arrayNumber(int number)

{

    // Dynamic allocated memory for a pointer "arr"

    int\* arr = (int\*)malloc(4 \* sizeof(int));

    // Check that if pointer is invalid

    if(arr == NULL)

    {

        exit(1);

    }

    // Use for loop to assign each digit of "number" equal to each element in array

    for(int i = 3; i >= 0; i--)

    {

        arr[i] = number % 10;

        number /= 10;

    }

    // Return a pointer

    return arr;

}