Project 1

Game: Mastermind

RCC

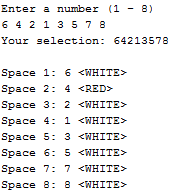
CSC-7-42645

David Covell

# **Introduction**

Title: Mastermind

The object of the game is to try and break the code that is generated by the computer.  
The computer will create a code based on your difficulty selection of 4, 6, or 8. It will then ask for the users input to begin guessing the sequence of numbers within the difficulty chosen.

Example:

The example above is based on a difficulty of 8 which is the hardest.

The user is given 10 attempts to break the computers code.

# **Summary**

Project size: about 329 lines  
Number of variables: about 15  
Number of methods: 8

This project took an estimated time of 1 week to complete while working no more than 5 hours a day.

I believe that this program has more possibilities of incorporating code to perform more complex problems.

# **Description**

The program serves to challenge the user to guess the computers sequence by using deduction.

# **Pseudo Code**

*Initialize*

*Get difficulty  
Set difficulty based on user input*

*Set computer code*

*User input to guess computer code*

*Compares computer vs user input*

*If user input = computer sequence  
 Then user wins game*

*If partial user input = partial computer sequence  
 Then display which column guess exists in*

*Repeat 10 times if not guessed within the 10 given attempts.  
 Else wins if guessed < 10 attempts*

# **Program**

#include <cstdlib>

#include <iostream>

#include <ctime>

#include "SortedList.h"

#include "Links.h"

using namespace std;

void chooseData(Links \*, int);

void printData(Links \*, int);

void setCompData(Links \*, int);

int setDifficulty();

int main(int argc, char\*\* argv) {

int size = 4, min = 1, max = 8, dif = 0, games = 0;

bool end = false, win = false;

srand(static\_cast<unsigned int>(time(0)));

SortedList<int> sortedLink;

dif = setDifficulty();

Links comp(dif);

setCompData(&comp, dif);

Links player(dif);

do {

SortedList<int> sortedLink;

int inc = 0, del = 0;

chooseData(&player, dif);

cout << "Your selection: ";

printData(&player, dif);

cout << "\n\n";

for(int i = 0; i < dif; i++) {

if(comp.getData(i) == player.getData(i)) {

sortedLink.push(player.getData(i), i, "RED");

del++;

inc++;

if(inc == dif) {

end = true;

win = true;

}

}

else {

for(int k = 0; k < dif; k++) {

if (comp.getData(i) == player.getData(k)) {

sortedLink.push(player.getData(k), k, "WHITE");

del++;

}

}

}

}

sortedLink.display();

games++;

if(games == 10)

end = true;

cout << "\n\n";

} while(end == false);

if(end == true || games == 10) {

if(win == true)

cout << "YOU WIN" << endl;

else

cout << "YOU LOSE" << endl;

cout << "# of games: " << games << endl;

cout << "Computer selection: ";

printData(&comp, dif);

}

return 0;

}

void printData(Links \*user, int dif) {

for(int i = 0; i < dif; i++) {

cout << user->getData(i);

}

}

void chooseData(Links \*user, int dif) {

int temp;

cout << "Enter a number (1 - 8)" << endl;

for(int i = 0; i < dif; i++) {

cin >> temp;

while (temp < 1 || temp > 8) {

cin.clear();

cin.ignore(10000, '\n');

cout << "Renter a VALID number (1 - 8)" << endl;

cin >> temp;

}

user->setData(i, temp);

}

cin.clear();

cin.ignore(10000, '\n');

}

void setCompData(Links \*user, int dif) {

//initialize computer sequence

for(int i = 0; i < dif; i++) {

user->setData(i, rand()%8+1);

if(i > 0) {

for(int k = 0;k < i; k++) {

while(user->getData(i) == user->getData(k)) {

user->setData(i, rand()%8+1);

k = 0;

}

}

}

}

}

int setDifficulty() {

int temp;

cout << "Please choose which difficulty you would like to play:\n(Easy = 4), (Medium = 6), (Hard = 8)" << endl;

cin >> temp;

switch(temp) {

case 4: return temp;

break;

case 6: return temp;

break;

case 8: return temp;

break;

default: cout << "Difficulty did not exist, please reenter your selection." << endl;

setDifficulty();

}

}

#ifndef LINKS\_H

#define LINKS\_H

class Links {

private:

int \*user;

public:

Links();

Links(int);

~Links();

int getData(int i) {

return user[i];

}

void setData(int i, int data);

};

#endif /\* LINKS\_H \*/

#include "Links.h"

#include <cstdlib>

#include <iostream>

Links::Links() {

}

Links::Links(int size) {

//Set comp array to new size of 4

user = new int[size];

}

Links::~Links() {

delete [] user;

}

void Links::setData(int i, int data) {

this->user[i] = data;

}

#ifndef LISTS\_H

#define LISTS\_H

#include <new>

#include <cstdlib>

#include <iostream>

using namespace std;

template <class T>

class SortedList {

private:

struct Node{

T data;

T space;

string color;

Node \*prev;

Node \*next;

};

Node \*head;

Node \*tail;

int size;

void memError(){

cout << "ERROR:Cannot allocate memory.\n";

exit(EXIT\_FAILURE);

}

void subError(){

cout << "ERROR: Subscript out of range.\n";

exit(EXIT\_FAILURE);

}

public:

SortedList();

virtual ~SortedList();

void push(T data, T space, string color);

void pop();

T at(int index);

T operator[](int index);

void display();

bool find(T space);

};

template <class T>

SortedList<T>::SortedList(){

head = NULL;

tail = NULL;

size = 0;

}

template <class T>

SortedList<T>::~SortedList(){

if (head) { //if head is set

do {

tail = head; //set head to tail

head = head->next; //set the head to be the old head link

delete tail; //delete tail

} while (head);

}

}

template <class T>

void SortedList<T>::push(T data, T space, string color){

if(!head){

try{

Node \*node = new Node;

node->data = data;

node->space = space;

node->color = color;

node->next = NULL;

node->prev = NULL;

head = node;

tail = head;

size++;

} catch(bad\_alloc){

memError();

}

} else {

Node \*working = head;

Node \*node = new Node;

node->data = data;

node->space = space;

node->color = color;

node->next = NULL;

node->prev = NULL;

bool inserted = false;

do{

if(working->space > node->space) { //so put node in front

node->prev = working->prev;

node->next = working;

if(working == head){

head = node;

} else {

(working->prev)->next = node;

}

working->prev = node;

inserted = true;

break;

} else if (working->next == NULL){ //then it is the biggest element add to rear

node->prev = working;

working->next = node;

inserted = true;

break;

}

} while((working = working->next));

if(inserted) {

size++;

} else {

cout << "Failed to insert element: " << space << endl;

}

}

}

template <class T>

void SortedList<T>::pop(){

Node \*current = head;

do{

if(current->next == NULL){

break;

}

current = current->next;

} while(true);

tail = current;

(current->prev)->next = NULL;

delete tail;

}

template <class T>

void SortedList<T>::display(){

Node \*current = head;

do{

cout << "Space " << current->space +1 << ": " << current->data << " <" << current->color << ">" << endl;

} while((current = current->next));

cout << endl;

}

template <class T>

bool SortedList<T>::find(T space){

Node \*current = head;

do{

if(current->data == space){

return true;

}

} while((current = current->next));

return false;

}

template <class T>

T SortedList<T>::at(int index){

if(index < 0 || index < size){

Node \*temp = head;

for(int i = 0; i < index; i++){

temp = temp->next;

}

return temp->space;

} else {

subError();

}

}

template <class T>

T SortedList<T>::operator [](int index) {

return at(index);

}

#endif /\* LISTS\_H \*/