第一题

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int FillList[10] = { 0,9,5,4,1,2,3,6,8,7 }; //初始化填充序列，按照此循序尝试填充数独单元

struct Sudoku //定义数独

{

int Map[10][10] = {0}; //9\*9数独，空出i=0||j=0的单元,简化代码

bool Occupied[10][10][10] = {0}; //用于标记Map[i][j可否填充k

int X = 1; //准备填充单元位置，默认（1,1）

int Y = 1;

void ToNext() //将准备填充单元移动到下一个

{

X = X + 1; //经过一行

if (X == 10)

{

X = 1;

Y = Y + 1; //从下一行第一个开始

}

};

bool IsDead() //判断准备填充位置不能被填充（同行、同列1-9都有了）

{

for(int i = 1; i <= 9; i++)

{

if (Occupied[Y][X][i] == false)

{

return false;

}

}

return true;

};

bool IsOK(int Num) //判断准备填充位置是否能被Num填充

{

if (Occupied[Y][X][Num] == false) //没有被占用，就可以填充

{

return true;

}

return false;

};

};

void RandomFillList() //随机化填充数组

{

bool Occupy[10] = {0}; //标记这个数字是否已经在数组内

srand(time(0)); //初始化随机

for (int i = 1; i <= 9; i++) //准备从1开始向数组放

{

int R = 0;

do

{

R = rand() % 10; //随机出一个位置

} while (Occupy[R] == true || R == 0); //如果这个位置空的，且不是0

FillList[i] = R; //写入并标记

Occupy[R] = true;

}

}

int GetSudoku(Sudoku S) //递归获得数独

{

if (S.IsDead()) //如果当前单元格没法填，返回

{

return 0;

}

if (S.Y == 10) //如果填够了，已经到10行，输出，返回

{

for (int i = 1;i <= 9; i++)

{

for (int j = 1; j <= 9;j++)

{

printf("%d ", S.Map[i][j]); //输出

}

printf("\n");

}

getchar();

printf("\n");

return 0;

}

for (int i = 1; i <= 9; i++) //按照填充数组顺序尝试填充

{

int R = FillList[i];

if (S.IsOK(R) == true) //如果能填充

{

Sudoku STemp = S;

STemp.Map[S.Y][S.X] = R; //填充

for (int j = 1; j <= 9; j++)

{

STemp.Occupied[S.Y][j][R] = true; //同列、同行标记这个数不能再次用

STemp.Occupied[j][S.X][R] = true;

}

STemp.ToNext(); //去下一个准备填充单元

GetSudoku(STemp); //尝试填充

}

}

return 0;

}

int main()

{

Sudoku S; //初始化数独

RandomFillList(); //随机填充数组

GetSudoku(S); //计算

}

第二题

#include <stdio.h>

#include <string.h>

#include<exception>

#define LenOfBuffer 15 //密码缓冲区大小，输入超了会崩

int ComparePosAndColor(char A[], char B[], int Len) //比较颜色对、位置对

{

int Count = 0;

for (int i = 0; i < Len; i++) //直接循环对比每一位并计数

{

if (A[i] == B[i])

{

Count++;

}

}

return Count;

}

int CompareColor(char A[], char B[], int Len) //比较颜色是对的

{

int Count = 0;

for (int i = 0; i < Len; i++) //对于B[i]一位，看看有没有任何一个A[j]可以匹配

{

for (int j = 0; j < Len; j++)

{

if (B[i] == A[j])

{

Count++; //并计数

break;

}

}

}

return Count;

}

int main()

{

char PSWD[LenOfBuffer] = "0";

printf("请输入4位密码，每位为1-8\n"); //获得密码

gets\_s(PSWD, LenOfBuffer);

printf("下面请尝试猜密码，4位密码，每位为1-8\n");

for (int i = 1; i <= 12; i++) //尝试十二次

{

char TryPSWD[LenOfBuffer] = "0";

gets\_s(TryPSWD, LenOfBuffer); //输入尝试的密码

int PosAndColorRight = ComparePosAndColor(PSWD, TryPSWD, 4); //分别得到位置颜色对、颜色对的个数

int OnlyColorRight = CompareColor(PSWD, TryPSWD, 4);

printf("%d 个位置颜色正确，%d 个只有颜色正确\n", PosAndColorRight, OnlyColorRight - PosAndColorRight);

if (PosAndColorRight == 4) //输出，并判断是不是四个都是对的

{

printf("密码破解者胜利\n");

return 0;

}

}

printf("密码设置者胜利\n"); //循环12次之后，设密码胜利

return 0;

}

第三题

#define ENABLE\_QUICK\_EDIT\_MODE

#include <stdio.h>

#include <windows.h>

#include <math.h>

#define LeftOffset 8 //windows自带边框，需要补偿一定的边缘坐标

#define TopOffset 28

//本程序直接点击窗口放棋子，而且还支持面板大小无限制调节，胜利规则无限制调节

#define LenOfChain 5 //五子棋，或多子

#define MapLen 9 //面板大小

#define StartChess -1 //先手棋子1白 -1黑

POINT WaitClick() //等待窗口被点击，然后返回点击位置相对于窗口的坐标

{

HWND WindowHandle = GetConsoleWindow(); //得到控制带句柄

POINT MousePos; //两个结构体对应鼠标位置，窗口位置

RECT WindowRect;

while (1)

{

GetCursorPos(&MousePos);

GetWindowRect(WindowHandle, &WindowRect); //刷新鼠标、窗口位置

if (GetAsyncKeyState(VK\_LBUTTON)) //如果检测到点击事件

{

while (GetAsyncKeyState(VK\_LBUTTON)); //消除抖动（等待点击按键抬起）

if (MousePos.x <= WindowRect.right && MousePos.y <= WindowRect.bottom && MousePos.x >= WindowRect.left + LeftOffset && MousePos.y >= WindowRect.top + TopOffset)

{ //判断是否在窗口内

POINT Temp; //定义Point返回

Temp.x = MousePos.x - WindowRect.left - LeftOffset;

Temp.y = MousePos.y - WindowRect.top - TopOffset;

return Temp;

}

}

}

}

void NoQuickEdit() //阻止控制台启用快速编辑（点击变成选择操作）

{

HANDLE HANDLEtoConsole = GetStdHandle(STD\_INPUT\_HANDLE); //得到句柄

DWORD Mode;

GetConsoleMode(HANDLEtoConsole, &Mode);

Mode &= ~ENABLE\_QUICK\_EDIT\_MODE; //移除快速编辑模式

Mode &= ~ENABLE\_INSERT\_MODE; //移除插入模式

Mode &= ~ENABLE\_MOUSE\_INPUT;

SetConsoleMode(HANDLEtoConsole, Mode); //施加属性

}

void SetConsolPos(int Left,int Top, int Length,int Height) //设置窗口左侧、顶部、宽、高

{

MoveWindow(GetConsoleWindow(), Left, Top, Length, Height, true);

}

void PrintMap(int Map[MapLen + 1][MapLen+1]) //输出棋盘格子

{

for (int i = 1; i <= MapLen; i++)

{

for (int j = 1; j <= MapLen;j++) //双循环输出

{

switch (Map[i][j]) //每一个格子三种情况，0为空，1为黑，2为白

{

case 0: printf("%s", "□");break;

case -1: printf("%s", "○");break;

case 1: printf("%s", "●");break;

}

}

printf("\n");

}

}

void PrintInfo(int ChessToPut, int Round) //输出回合信息

{

printf("%d回合,棋子:", Round); //输出第几回合

switch (ChessToPut)

{

case 0: printf("%s", "□");break;

case -1: printf("%s", "○");break;

case 1: printf("%s", "●");break; //输出现在下哪种棋子

}

printf("\n");

}

int Is5Chain(int Num[MapLen + 1]) //判断Num中是否有LenOfChain个连续的1或-1

{

int Count = 0;

for (int i = 1; i <= MapLen; i++)

{

if (Num[i] == 0) //一个是0，Count归零

{

Count = 0;

}

else if (Count \* Num[i] < 0) //变号了，那就新的有一个

{

Count = Num[i];

}

else

{

Count += Num[i]; //没变号，正常加减

}

if (abs(Count) == LenOfChain) //如果足够LenOfChain，就输出结果

{

return Count / abs(Count);

}

}

return 0;

}

int IsOver(int Map[MapLen + 1][MapLen + 1]) //用于判断是否存在足够的连珠

{ //首先进行横竖向判断

int Num1[MapLen + 1] = {0}; //横向和竖向

int Num2[MapLen + 1] = {0};

for (int i = 1; i <= MapLen;i++)

{

for (int j = 1; j <= MapLen; j++)

{

Num1[j] = Map[i][j]; //把对应的列、行复制到新数组

Num2[j] = Map[j][i];

}

int Result1 = Is5Chain(Num1); //用Is5Chain判断

int Result2 = Is5Chain(Num2);

if (Result1 != 0) return Result1; //输出结果

if (Result2 != 0) return Result2;

}

//其次进行斜向判断

for (int i = 0; i <= MapLen \* 2 ;i++)

{

int Num3[MapLen + 1] = { 0 }; //对应三种不同情况

int Num4[MapLen + 1] = { 0 };

int Num5[MapLen + 1] = { 0 };

int i1 = 1, i2 = 1,i3 = 1;

for (int j = 1; j <= MapLen; j++)

{

for (int k = 1; k<= MapLen; k++)

{

if (j + k == i) //左下角到右上角

{

Num3[i1++] = Map[j][k];

}

if (j - k == i) //左上角到右下角（上半部分）

{

Num4[i2++] = Map[j][k];

}

if (k - j == i) ////左上角到右下角（下半部分）

{

Num5[i3++] = Map[j][k];

}

}

}

int Result3 = Is5Chain(Num3); //三种情况分别判断有没有连上的

int Result4 = Is5Chain(Num4);

int Result5 = Is5Chain(Num5);

if (Result3 != 0) return Result3; //如果有胜利就返回

if (Result4 != 0) return Result4;

if (Result5 != 0) return Result5;

}

return 0; //无人胜利

}

int main()

{

NoQuickEdit();

SetConsolPos(300,300,50 + MapLen \* 16,160 + MapLen \* 16); //初始化窗口

int Map[MapLen + 1][MapLen + 1] = {0}; //初始化对局信息

int ChessToPut = StartChess,Round = 1;

for (; ; )

{

if (Round == MapLen \* MapLen + 1) //格子全填满

{

PrintMap(Map); //输出对应文字

printf("平局\n");

system("pause");

exit(0);

}

int Result = IsOver(Map); //判断是否胜利

if (Result != 0)

{

PrintMap(Map); //输出对应文字

printf("%s方胜利\n", (Result > 0) ? "●" : "○");

system("pause");

exit(0);

}

PrintMap(Map); //输出当前局面

PrintInfo(ChessToPut, Round);

POINT ClickPos = WaitClick(); //得到点击操作位置

if (Map[ClickPos.y / 16 + 1][ClickPos.x / 16 + 1] != 0)

{

printf("该地方已经有棋子了\n点击继续"); //如果点的地方已经有棋子

WaitClick();

system("cls");

}

else

{

Map[ClickPos.y / 16 + 1][ClickPos.x / 16 + 1] = ChessToPut; //没问题就在此下棋

system("cls");

ChessToPut \*= -1;

Round++;

}

}

return 0;

}

第四题

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define MaxN 30 //矩阵最大大小

struct Matrix

{

float Map[MaxN + 1][MaxN + 1] = {0}; //从[1]开始调用

int Length = 0;

int Height = 0;

void GetIdentityMatrix(int N) //将目标设置为N大小标准阵

{

for (int i = 0;i <= MaxN;i++)

{

Map[i][i] = 1;

}

Length = N;

Height = N;

}

void ToPrint() //输出矩阵

{

for (int i = 1; i <= Height; i++)

{

for (int j = 1; j <= Length; j++)

{

printf("%5.1f", Map[i][j]); //双循环输出

}

printf("\n");

}

}

};

Matrix InputMatrix()

{

Matrix Temp;

printf("请输入矩阵高与长\n");

scanf\_s("%d%d", &Temp.Height, &Temp.Length);

for (int i = 1; i <= Temp.Height; i++)

{

for (int j = 1; j <= Temp.Length; j++)

{

scanf\_s("%f",&Temp.Map[i][j]);

}

}

return Temp;

}

Matrix InputMatrixFrom0()

{

Matrix Temp;

printf("请输入矩阵高与长\n");

scanf\_s("%d%d", &Temp.Height, &Temp.Length);

for (int i = 0; i < Temp.Height; i++)

{

for (int j = 0; j < Temp.Length; j++)

{

scanf\_s("%f", Temp.Map[i][j]);

}

}

return Temp;

}

Matrix Transpose(Matrix Input) //输入Input，输出转制后矩阵

{

struct Matrix Temp;

for (int i = 1; i <= Input.Height; i++)//循环写入

{

for (int j = 1; j <= Input.Length; j++)

{

Temp.Map[j][i] = Input.Map[i][j];//置换

}

}

Temp.Length = Input.Height;

Temp.Height = Input.Length;

return Temp;

}

Matrix OrgMatrix(Matrix Input)

{

Matrix VoidM;

Matrix Temp;

if (Input.Length != Input.Height) //首先判断是不是个正方的

{

printf("非方阵\n");

return VoidM; //不是正方返回空矩阵

}

Temp = Input;

Matrix Result; Result.GetIdentityMatrix(Input.Height); //初始化备用矩阵

float Max;

int k,j,temp;

for (int i = 1; i <= Input.Length; i++)

{

Max = Temp.Map[i][i]; //对对角线遍历寻找主元

k = i;

for (j = i + 1; j <= Input.Length; j++) //判断上方是否有大于主元

{

if (fabs(Temp.Map[j][i]) > fabs(Max))

{

Max = Temp.Map[j][i]; //更新

k = j;

}

}

if (k != i) //如果i行没有主元，每一行都进行交换

{

for (j = 1; j <= Input.Length; j++)

{

temp = Temp.Map[i][j];

Temp.Map[i][j] = Temp.Map[k][j];

Temp.Map[k][j] = temp;

temp = Result.Map[i][j]; //标准矩阵同时交换

Result.Map[i][j] = Result.Map[k][j];

Result.Map[k][j] = temp;

}

}

if (Temp.Map[i][i] == 0) //主元==0, 矩阵不满秩矩阵,没有逆矩阵

{

printf("不满秩矩阵\n");

return VoidM;

}

temp = Temp.Map[i][i]; //消去A的第i列除去i行以外的各行元素

for (j = 1; j <= Input.Length; j++)

{

Temp.Map[i][j] = Temp.Map[i][j] / temp; //对角线取1

Result.Map[i][j] = Result.Map[i][j] / temp; //标准矩阵同时计算

}

for (j = 1; j <= Input.Length; j++) //反向0-最大行

{

if (j != i) //除了i行都交换

{

temp = Temp.Map[j][i];

for (k = 1; k <= Input.Length; k++) //[j] - [i]\*[j][i]

{

Temp.Map[j][k] = Temp.Map[j][k] - Temp.Map[i][k] \* temp;

Result.Map[j][k] = Result.Map[j][k] - Result.Map[i][k] \* temp;

}

}

}

}

return Result;

}

Matrix Multiply(Matrix A, Matrix B)//计算A\*B

{

Matrix VoidM;

if (A.Length != B.Height || A.Height != B.Length) //判断行列相等

{

printf("行列不等\n");

return VoidM;

}

int Min = (A.Height < A.Length) ? A.Height : A.Length; //获得结果矩阵大小，和最长列长度

int Max = A.Height + A.Length - Min;

Matrix Result; //初始化结果阵列

Result.Height = Min;

Result.Length = Min;

for (int i = 1; i <= Min; i++) //遍历结果的每一项

{

for (int j = 1; j <= Min; j++)

{

int sum = 0;

for (int k = 1; k <= Max; k++)

{

sum += A.Map[i][k] \* B.Map[k][j]; //按照A行\*B列的方式进行求和

}

Result.Map[i][j] = sum;

}

}

return Result; //返回结果阵列

}

float determinant(float matrix[MaxN + 1][MaxN + 1], int order);

int laplace\_expansion(float matrix[MaxN + 1][MaxN + 1], int r, int c, int order)

{

int original\_i, original\_j, i, j;

float result = 0, cofactor[MaxN + 1][MaxN + 1];

for (i = 0;i < order;i++)

for (j = 0;j < order;j++)

{

original\_i = i;

original\_j = j;

if (i == r || j == c);

else

{

if (i > r)

i--;

if (j > c)

j--;

cofactor[i][j] = matrix[original\_i][original\_j];

i = original\_i;

j = original\_j;

}

}

if (order >= 2)

result = determinant(cofactor, order - 1);

return result;

}

float determinant(float matrix[MaxN + 1][MaxN + 1], int order)//n阶矩阵行列式不会，没法写，借鉴一下

{

int sign = 1, i;

float result = 0, cofactor;

if (order == 1)

result = matrix[0][0];

else

for (i = 0;i < order;i++)

{

cofactor = laplace\_expansion(matrix, i, 0, order);

result += sign \* matrix[i][0] \* cofactor;

sign \*= -1;

}

return result;

}

void PrintMenu() //输出菜单

{

printf(

"1 转置\n"

"2 逆矩阵\n"

"3 乘法\n"

"4 行列式\n"

"5 Quit\n"

"请输入选择\n"

);

}

void WaitAndCls() //等待获取按键，并清空屏幕

{

system("pause");

system("cls");

}

int GetNum(char Arr[])

{

int Temp;

printf("%s");

scanf\_s("%d", &Temp);

return Temp;

}

int main()

{

Matrix A1, A2, A3, A4;

Matrix B1, B2, B3, B4;

Matrix C1, C2, C3, C4;

for (;;)

{

PrintMenu(); //打印菜单

int Chose = 0;

scanf\_s("%d", &Chose);

switch (Chose) //判断对应选项并输出

{

case 1:

A1 = InputMatrix();

B1 = Transpose(A1);

B1.ToPrint();

WaitAndCls();

break;

case 2:

A2 = InputMatrix();

B2 = OrgMatrix(A2);

B2.ToPrint();

WaitAndCls();

break;

case 3:

A3 = InputMatrix();

B3 = InputMatrix();

C3 = Multiply(A3,B3);

C3.ToPrint();

WaitAndCls();

break;

case 4:

A4 = InputMatrixFrom0();

printf("结果为%f", determinant(A4.Map,A4.Height));

WaitAndCls();

break;

case 5:

exit(0);

break;

}

}

return 0;

}

第五题

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h> //输入输出，文本处理，输出控制都需要

#include <string.h>

#include <stdlib.h>

#define StudentMax 50 //最多50个学生，名字最长处64个字符，有三科

#define NameLen 64

#define ScoreLen 3

struct StudentInfo //定义学生结构体，名字，ID，分数若干

{

char Name[NameLen] = { 0 };

long ID = 0;

float Score[ScoreLen] = { 0 };

};

void SetStudent(StudentInfo Student[], int Index, StudentInfo StudentToAdd)//设置一个学生，将StudentToAdd中信息复制到Student[Index]中

{

Student[Index] = StudentToAdd;

}

void ShowStudent(StudentInfo Student[]) //输出所有学生

{

int Count = 0;

printf("Index\tName\t\t ID\t\tC1\t\tC2\t\tC3\t\tScore\t\tAvg\n"); //输出表头

for (int i = 0; i < StudentMax && Student[i].ID != 0;i++) //输出每一个学生

{

float TotalScore = Student[i].Score[0] + Student[i].Score[1] + Student[i].Score[2]; //记录总分

printf("%-8d%-8s\t%-8d\t%-8.3f\t%-8.3f\t%-8.3f\t%-8.3f\t%-8.3f\n",

i, Student[i].Name, Student[i].ID,

Student[i].Score[0], Student[i].Score[1], Student[i].Score[2],

TotalScore, TotalScore / 3);//输出

Count++;//统计学生数量

}

float CourseAvg[ScoreLen] = { 0 }; //计算平均分

for (int i = 0; i < StudentMax && Student[i].ID != 0;i++)

{

CourseAvg[0] += Student[i].Score[0];

CourseAvg[1] += Student[i].Score[1];

CourseAvg[2] += Student[i].Score[2];

}

printf("\n平均分\t\t\t\t\t%-8.3f\t%-8.3f\t%-8.3f\n",

CourseAvg[0] / Count, CourseAvg[1] / Count, CourseAvg[2] / Count); //输出平均分

}

void AddStudent(StudentInfo Student[]) //添加若干个学生

{

int Count = 0;

int Num = 0;

for (;Count < StudentMax && Student[Count].ID != 0;Count++); //获得当前学生总个数

printf("请输入要输入几个，目前有%d个\n", Count);

scanf("%d", &Num); //获取输入数量

while (Num + Count > StudentMax)

{

printf("太多了重新输入\n"); //避免超量

scanf("%d", &Num);

}

printf("请输入学生姓名，学号，三门课成绩\n");

for (int i = 1;i <= Num; i++)

{

StudentInfo Temp; //获取若干个学生信息

scanf("%s%ld%f%f%f", &Temp.Name, &Temp.ID, &Temp.Score[0], &Temp.Score[1], &Temp.Score[2]);

SetStudent(Student, Count++, Temp); //写入Student记录中

}

}

void DelStudent(StudentInfo Student[])//删除一个学生

{

int Count = 0;

int Num = 0;

for (;Count < StudentMax && Student[Count].ID != 0;Count++); //得到学生总数

printf("请输入要删除第几个，目前有%d个\n", Count);

scanf("%d", &Num); //获取要删除的序号

while (Num > Count || Num <= 0)

{

printf("输入错误重新输入\n"); //避免越界

scanf("%d", &Num);

}

for (int i = Num;i < Count - 1; i++) //用后一个覆盖前一个

{

SetStudent(Student, i, Student[i + 1]);

}

StudentInfo VoidStudent;

SetStudent(Student, Count - 1, VoidStudent); //设置最后一个为空

}

void SaveStudent(StudentInfo Student[])//保存学生表格

{

FILE\* Path;

Path = fopen("./OutPut.txt", "w"); //打开文件得到指针

if (Path == NULL)

{

printf("文件访问失败\n"); //错误输出

exit(0);

}

else

{ //原理同ShowStudent

int Count = 0;

fprintf(Path, "Index\tName\t\t ID\t\tC1\t\tC2\t\tC3\t\tScore\t\tAvg\n");

for (int i = 0; i < StudentMax && Student[i].ID != 0;i++)

{

float TotalScore = Student[i].Score[0] + Student[i].Score[1] + Student[i].Score[2];

fprintf(Path, "%-8d%-16s\t%-16d\t%-8.3f\t%-8.3f\t%-8.3f\t%-8.3f\t%-8.3f\n", i, Student[i].Name, Student[i].ID, Student[i].Score[0], Student[i].Score[1], Student[i].Score[2], TotalScore, TotalScore / 3);

Count++;//注意！文件保存时部分%s，%d要改变长度，否则缩进错误：%8d--->%16d

}

float CourseAvg[ScoreLen] = { 0 };

for (int i = 0; i < StudentMax && Student[i].ID != 0;i++)

{

CourseAvg[0] += Student[i].Score[0];

CourseAvg[1] += Student[i].Score[1];

CourseAvg[2] += Student[i].Score[2];

}

fprintf(Path, "\n平均分\t\t\t\t%-8.3f\t%-8.3f\t%-8.3f\n", CourseAvg[0] / Count, CourseAvg[1] / Count, CourseAvg[2] / Count);

fclose(Path); //关闭文件

}

}

void PrintMenu() //输出菜单

{

printf(

"1 Display all information\n"

"2 Add students\n"

"3 Delete a student\n"

"4 Save to file\n"

"5 Quit\n"

"请输入选择\n"

);

}

void WaitAndCls() //等待获取按键，并清空屏幕

{

system("pause");

system("cls");

}

void InitializeStudent(StudentInfo Student[]) //初始化部分学生信息

{

StudentInfo S1 = { "田所浩二" ,1145,{19,19,81} };

SetStudent(Student, 0, S1);

StudentInfo S2 = { "张三" ,1919,{99,59,88} };

SetStudent(Student, 1, S2);

StudentInfo S3 = { "卢本伟" ,9999,{20,2,17} };

SetStudent(Student, 2, S3);

}

int main()

{

StudentInfo Student[StudentMax]; //初始化学生队列

InitializeStudent(Student);

for (;;)

{

PrintMenu(); //打印菜单

int Chose = 0;

scanf("%d", &Chose);

switch (Chose) //判断对应选项并输出

{

case 1:

ShowStudent(Student);

WaitAndCls();

break;

case 2:

AddStudent(Student);

WaitAndCls();

break;

case 3:

DelStudent(Student);

WaitAndCls();

break;

case 4:

SaveStudent(Student);

WaitAndCls();

break;

case 5:

exit(0);

break;

}

}

return 0;

}