**Головачев Д.А. Y2231 Лабораторная работа номер 17.**

Задание: Составить программу, отыскивающую проход по лабиринту. Визуализировать (простейшим способом, с помощью вывода символов) лабиринт и перемещение по нему (найденный путь).

**Node \*top = 0; -** указатель на вершину стека.

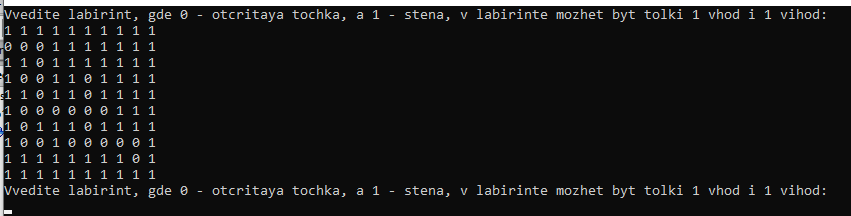
**int ary[N][N],n,m,prohodimost;** - все переменные, что я использую имеют тип int.

ary[N][N], потому что матрица нужна всего 10 на 10 и элементы там, это 0, 1 и 4.

n,m – переменные, которые нужны для определения точки входа и для дальнейшего перемещения по лабиринту, то есть просто индексы(от 0 до 9).

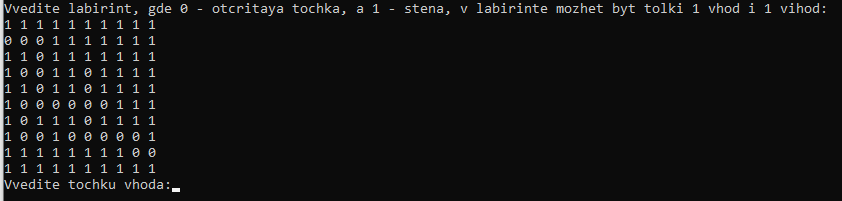
prohodimost **–** переменная, которая нужна для контроля количества «дыр» в стенах лабиринта, по начальному условию должно быть только 2 «дыры» - вход и выход.

**Алгоритм работы программы будет комментариями в коде.** Но если полный алгоритм вам не сильно интересен, то: Сначала программа просит ввести матрицу(лабиринт) из нулей и единиц, потом проверяет введенное на наличие входа/выхода и если входов/выходов больше или меньше 2, то:

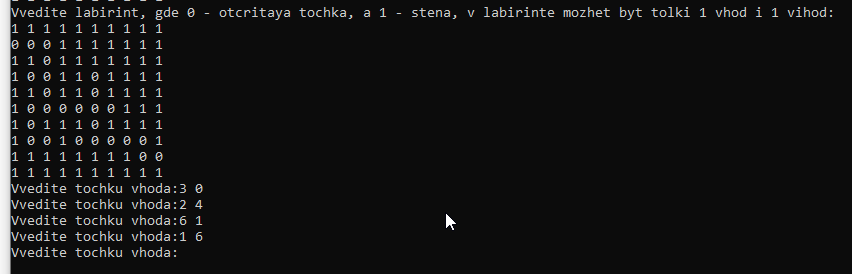


(на этом скриншоте только 1 точка, куда можно войти/выйти)

А если есть вход и выход, то просит ввести координаты точки входа:

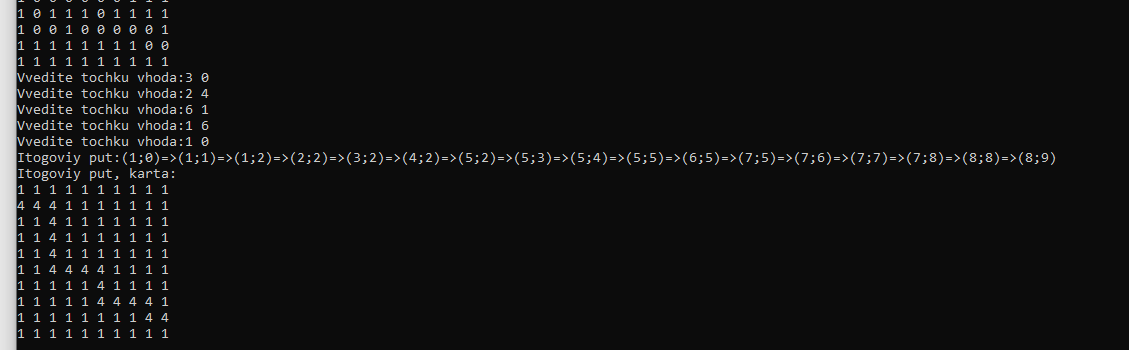


После этого конечно проверяет точку на возможность через ней войти и если войти нельзя, то просит ввести заново:



(в данном примере, как точка входа, может подойти только 1 0 и 8 9)

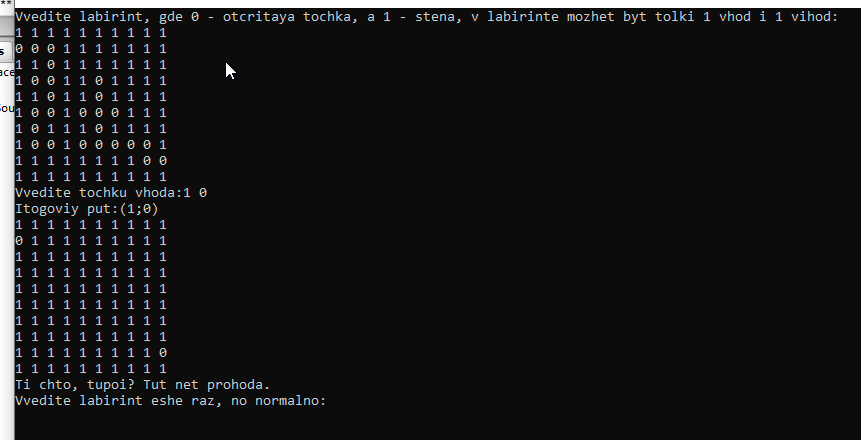
Если ввести нормальную точку, то программа переходит к следующему этапу.



(в этом случае, так как других проблем не было, программа просто выполнила свою работу, об альтернативных вариантах делее)

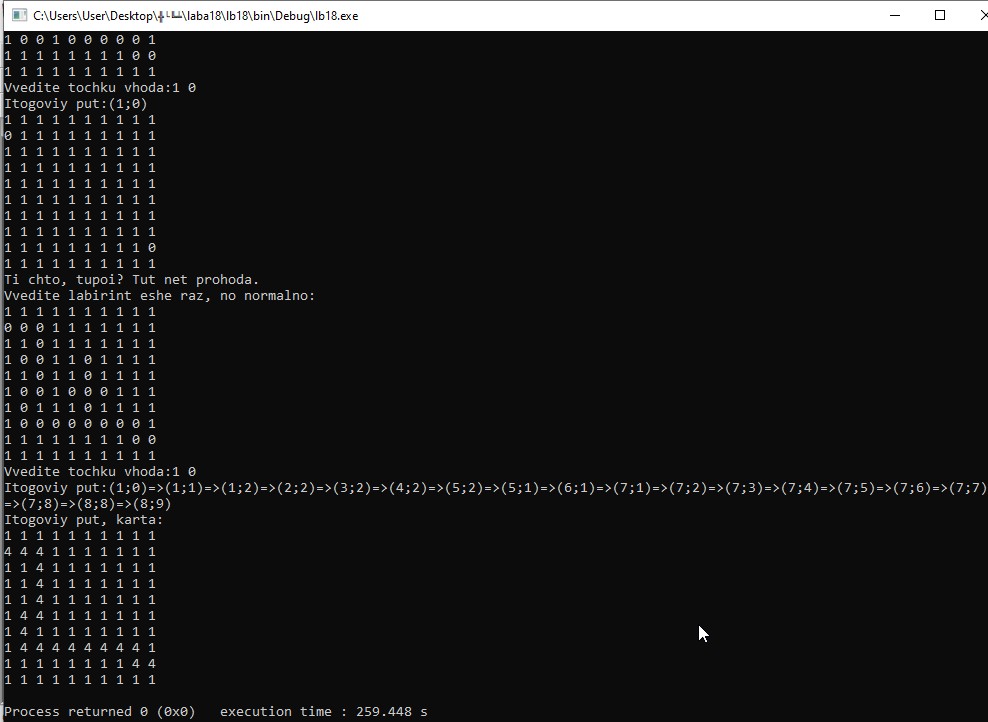
После проверки точки входа программа убирает из лабиринта все тупики, если они есть. (Это место лучше посмотреть в коде)

Далее происходит процесс прохождения лабиринта и замены пройденных точек на 4 и вывода результата на экран, НО, если так получилось, что лабиринт невозможно пройти, то программа просит ввести лабиринт заново:



(В этом примере программа удаления тупиков приняла почти весь лабиринт за тупики, потому что изначально связи между входом и выходом не было)

После этого вы вводите лабиринт заново, а программа начинает работать с самого начала, в конце вам просто выводится готовый результат.



**Код:**

#include <iostream>

#include <stdio.h>

#define N 10

using namespace std;

**// описание структуры**

struct Node {

int d;

Node \*p;

};

void push(Node \*\*top, const int d)

{

Node \*pv = new Node;

pv->d = d;

pv->p = \*top;

\*top = pv;

}

int pop(Node \*\*top)

{

int temp = (\*top)->d; **// выборка**

Node \*pv = \*top;

\*top = (\*top)->p;

delete pv;

return temp;

}

int main()

{

Node \*top = 0; // стек пуст

int ary[N][N],n,m,prohodimost;

while (prohodimost!=2) { **//проверка лабиринта на то, возможно ли его пройти**

prohodimost=0;

cout << "Vvedite labirint, gde 0 - otcritaya tochka, a 1 - stena, v labirinte mozhet byt tolki 1 vhod i 1 vihod:"<< endl;

for (int i = 0; i < N; i++) { **//ввод с клавиатуры лабиринта**

for (int j = 0; j < N; j++) {

scanf("%d",&ary[i][j]);

}

}

for (int j = 0; j < N; j++) {

if (ary[0][j]==0||ary[9][j]==0) {

prohodimost++;

} else {continue;}

}

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

if (ary[i][0]==0||ary[i][9]==0) {

prohodimost++;

} else {continue;}

}

}

prohodimost=0; **//обнуление параметра проходимости, чтобы при новом вводе лабиринта не происходило ошибок**

cout << "Vvedite tochku vhoda:";

scanf("%d%d",&n,&m);

while (ary[n][m]!=0||(n!=0&&m!=0&&m!=9&&n!=9)) { **//проверка на то, что вход с краю и это не стена**

cout << "Vvedite tochku vhoda:";

scanf("%d%d",&n,&m);

}

push (&top, n); **//занесение длины точки в стек**

push (&top, m); **//занесение высоты точки в стек**

for (int i = 1; i < (N-1); i++) { **//это первый из 4-х циклов, которые полностью освобождают лабиринт от тупиков**

for (int j = 1; j < (N-1); j++) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

for (int i = N-2; i > 0; i--) {

for (int j = N-2; j > 0; j--) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

for (int i = 1; i < (N-1); i++) {

for (int j = 1; j < (N-1); j++) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

for (int i = N-2; i > 0; i--) {

for (int j = N-2; j > 0; j--) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

while (ary[n][m]!=4||((n!=0||n!=9)&&(m!=0||m!=9))) { **//цикл на прохождение лабиринта**

if (ary[n][m]==4) {

break;

}

if (ary[n+1][m]==0) { **//если можно идти вниз**

if (ary[n-1][m]!=0&&ary[n][m+1]!=0&&ary[n][m-1]!=0) {

ary[n][m]=4;

} else {ary[n][m]=4;}

n++; **//переход на 1 клетку вниз**

push (&top, n); **//занесение длины точки в стек**

push (&top, m); **//занесение высоты точки в стек**

if (n==0||n==9||m==0||m==9) {  **//проверка на то, является ли точка, в которую мы пришли, выходом**

ary[n][m]=4;

} else {continue;}

} else if (ary[n-1][m]==0) { **//если можно идти вверх**

if (ary[n+1][m]!=0&&ary[n][m+1]!=0&&ary[n][m-1]!=0) { **//проверка на то, есть ли проходы не по направлению движения**

ary[n][m]=4;

} else {ary[n][m]=4;}

n--; **//переход на 1 клетку вверх**

push (&top, n);

push (&top, m);

if (n==0||n==9||m==0||m==9) {

ary[n][m]=4;

} else {continue;}

} else if (ary[n][m+1]==0) { **//если можно идти вправо**

if (ary[n-1][m]!=0&&ary[n+1][m]!=0&&ary[n][m-1]!=0) {

ary[n][m]=4;

} else {ary[n][m]=4;}

m++; **//переход на 1 клетку вправо**

push (&top, n);

push (&top, m);

if (n==0||n==9||m==0||m==9) {

ary[n][m]=4;

} else {continue;}

} else if (ary[n][m-1]==0) { **//если можно идти влево**

if (ary[n-1][m]!=0&&ary[n][m+1]!=0&&ary[n+1][m]!=0) {

ary[n][m]=4;

} else {ary[n][m]=4;}

m--; **//переход на 1 клетку влево**

push (&top, n);

push (&top, m);

if (n==0||n==9||m==0||m==9) {

ary[n][m]=4;

} else {continue;}

} else { **//если нельзя никуда идти**

cout << endl;

for (int i = 0; i < N; i++) { **//вывод матрицы на экран**

for (int j = 0; j < N; j++) {

cout << ary[i][j] << " ";

}

cout << endl;

}

while (top) { **//полная отчистка стека от данных, так как они не действительны**

pop(&top);

}

cout<<"Ti chto, tupoi? Tut net prohoda."<<endl;

prohodimost=0;

while (prohodimost!=2) { **//цикл для ввода лабиринта и проверки его пригодности к поиску прохода**

prohodimost=0;

cout << "Vvedite labirint eshe raz, no normalno:"<< endl;

for (int i = 0; i < N; i++) { **//непосредственно ввод**

for (int j = 0; j < N; j++) {

scanf("%d",&ary[i][j]);

}

}

for (int j = 0; j < N; j++) { **//проверка параметра количества входов/выходов**

if (ary[0][j]==0||ary[9][j]==0) {

prohodimost++;

} else {continue;}

}

for (int i = 0; i < N; i++) { **//проверка параметра количества входов/выходов**

if (ary[i][0]==0||ary[i][9]==0) {

prohodimost++;

} else {continue;}

}

cout << "Vvedite tochku vhoda:";

scanf("%d%d",&n,&m); **//ввод точки входа в лабиринт**

while (ary[n][m]!=0||(n!=0&&m!=0&&m!=9&&n!=9)) { **//проверка на пригодность точки**

cout << "Vvedite tochku vhoda:";

scanf("%d%d",&n,&m);

}

push (&top, n);

push (&top, m);

for (int i = 1; i < (N-1); i++) { **//это первый из 4-х циклов, которые полностью освобождают лабиринт от тупиков**

for (int j = 1; j < (N-1); j++) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

for (int i = N-2; i > 0; i--) {

for (int j = N-2; j > 0; j--) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

for (int i = 1; i < (N-1); i++) {

for (int j = 1; j < (N-1); j++) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

for (int i = N-2; i > 0; i--) {

for (int j = N-2; j > 0; j--) {

if (ary[i][j]==0&&(ary[i+1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i-1][j]!=0&&ary[i][j+1]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j-1]!=0)||(ary[i+1][j]!=0&&ary[i-1][j]!=0&&ary[i][j+1]!=0)) {

ary[i][j]=1;

}

}

}

}

}

}

cout << endl;

cout<< "Itogoviy put, karta:"<< endl;

for (int i = 0; i < N; i++) { **//непосредственно вывод итогового пути на карте**

for (int j = 0; j < N; j++) {

cout << ary[i][j] << " ";

}

cout << endl;

}

cout<< "Itogoviy put:"; **//вывод точки входа в лабиринт**

while (top) { **//вывод всех остальных точек**

cout << "(";

cout << pop(&top);

cout << ";";

cout << pop(&top);

cout << ")<=";

}

return 0;

}

**Результаты тестирования механизмов защиты были выше, так что я просто приведу несколько примеров лабиринтов**. (также во все лабиринты я буду заходить с обоих сторон)

1 1 1 1 1 1 1 1 1 1

0 0 0 1 1 1 1 1 1 1

1 1 0 1 1 0 0 0 0 1

1 0 0 1 1 0 1 1 0 1

1 1 0 1 1 0 1 1 0 1

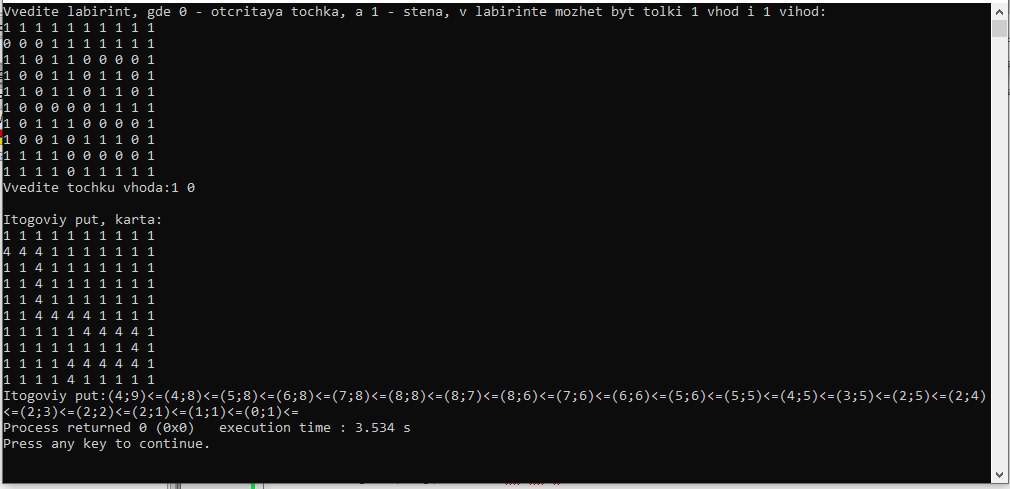
1 0 0 0 0 0 1 1 1 1

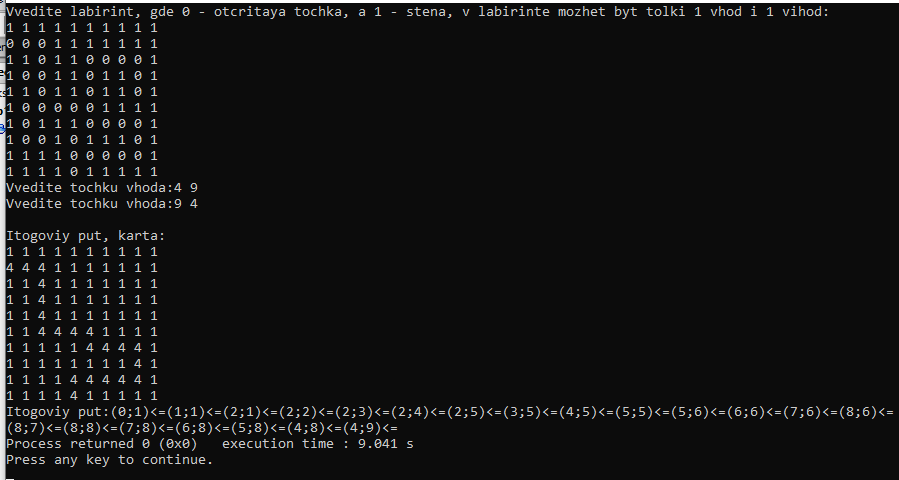
1 0 1 1 1 0 0 0 0 1

1 0 0 1 0 1 1 1 0 1

1 1 1 1 0 0 0 0 0 1

1 1 1 1 0 1 1 1 1 1





1 0 1 1 1 1 1 1 1 1

1 0 0 0 1 1 0 0 0 0

1 1 0 1 0 0 0 1 0 1

1 0 0 1 1 0 1 0 0 1

1 1 0 1 1 1 1 1 0 1

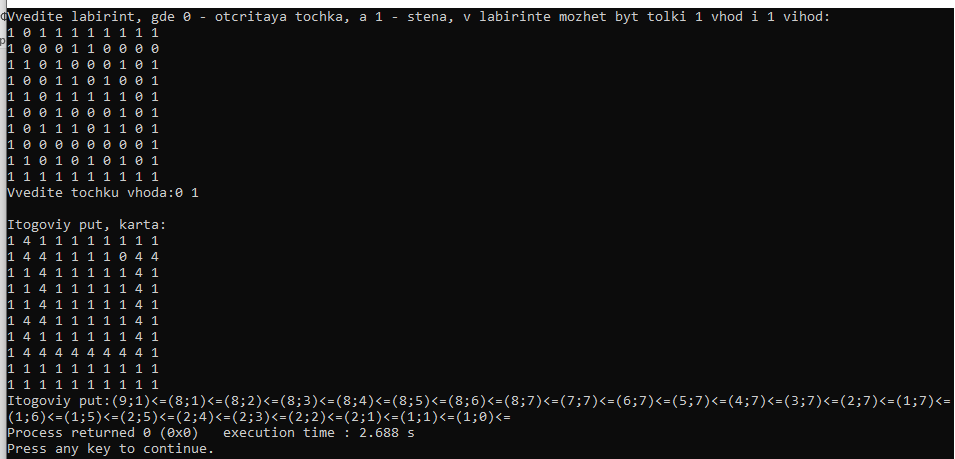
1 0 0 1 0 0 0 1 0 1

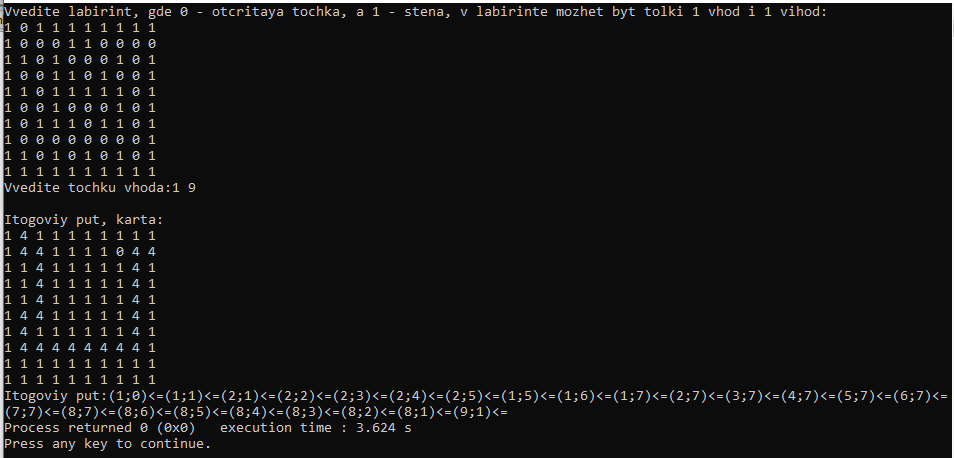
1 0 1 1 1 0 1 1 0 1

1 0 0 0 0 0 0 0 0 1

1 1 0 1 0 1 0 1 0 1

1 1 1 1 1 1 1 1 1 1





1 1 1 1 1 1 1 1 0 1

1 0 0 1 1 1 1 1 0 1

1 1 0 1 1 0 0 0 0 1

1 0 0 1 1 0 1 1 0 1

1 1 0 1 1 0 1 1 0 1

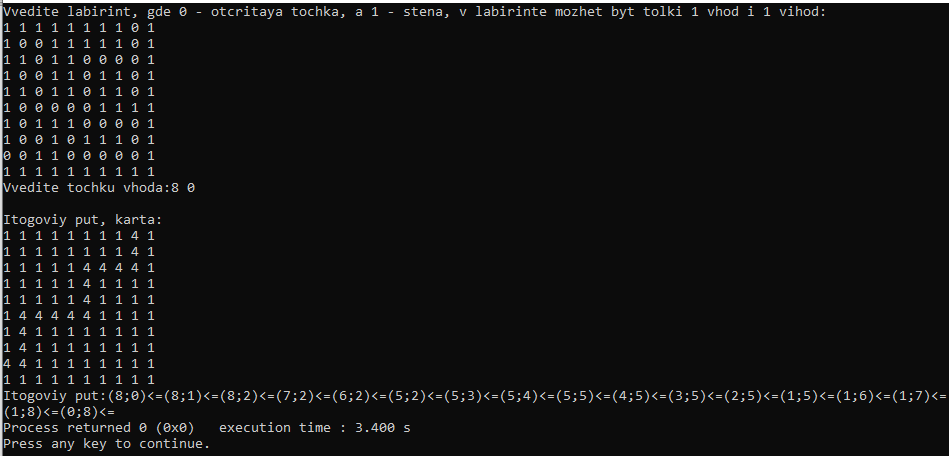
1 0 0 0 0 0 1 1 1 1

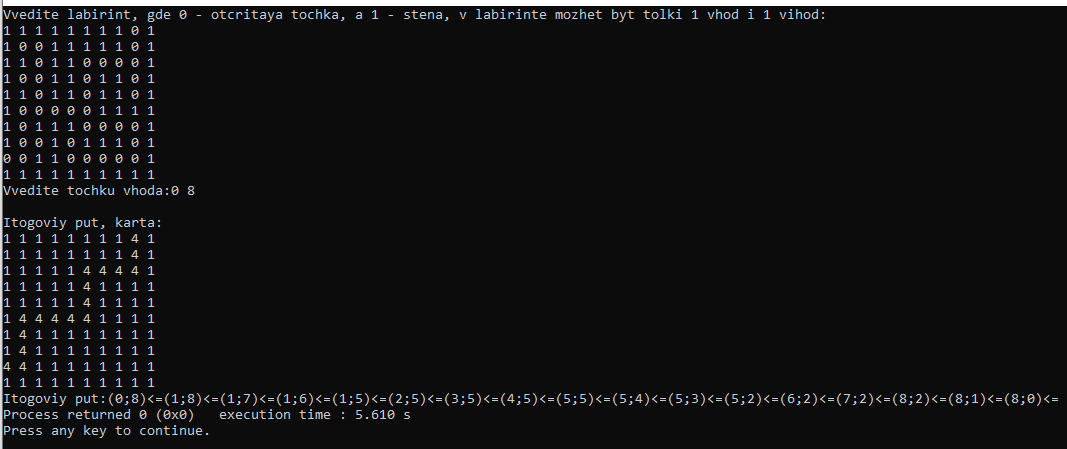
1 0 1 1 1 0 0 0 0 1

1 0 0 1 0 1 1 1 0 1

0 0 1 1 0 0 0 0 0 1

1 1 1 1 1 1 1 1 1 1





**Вывод:** если идти по лабиринту, изначально зная, где все тупики, то легко из него выйдешь, если будешь идти в любом возможном направлении кроме «назад»). Матрица находится в стеке, поэтому я решил с помощью неё же выводить итоговую карту пути. А координаты пути заносятся в стек, как отдельные переменные и объединяются в полноценные координаты при выводе. Динамический стек удобно использовать, потому что это не засоряет память (память отчищается во время вывода элементов оттуда).