



Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National
Laboratory
dbiersach@bnl.gov



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace SimpleEvents
11 {
12     public partial class Form1 : Form
13     {
14         Person person = new Person();
15
16         public Form1()
17         {
18             InitializeComponent();
19             person.FirstName = "Christian";
20             person.LastName = "Pano";
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             person.MainColor = textBox1.Text;
26         }
27     }
28 }
```

Session 22
Search Algorithms,
Adjacency Matrix

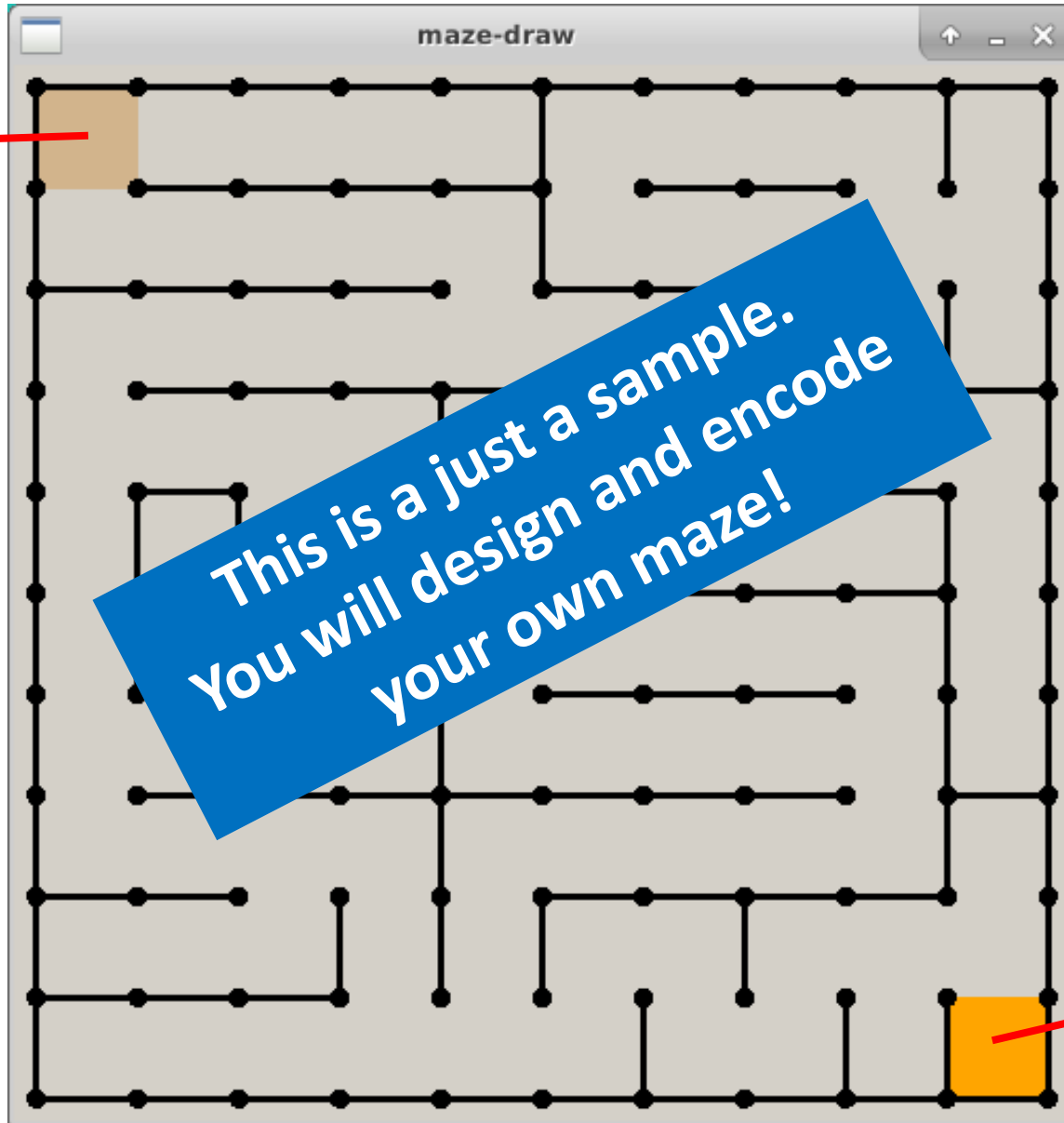
Session Goals

- Create a **2D maze** on graph paper
- Understand how to **encode** the cell walls in **base 2**
- Learn how **bitwise operators** can decode a wall value
- Perform **file input / output** using CSV and binary formats
- Appreciate backtracking in **depth-first** search algorithm
- Implement breadcrumbs using a **stack** data structure
- Create an **adjacency matrix** to improve search efficiency

Maze Solver

- Write a program to navigate maze of **10 x 10** square cells
- The program must find a **path** from the **entrance** to the **exit**
- The maze **perimeter** must not have any holes
- There must not be any "one-way" doors
- The program must run autonomously during the search
- Each cell should be described using a simple **encoding**

Entrance



Exit

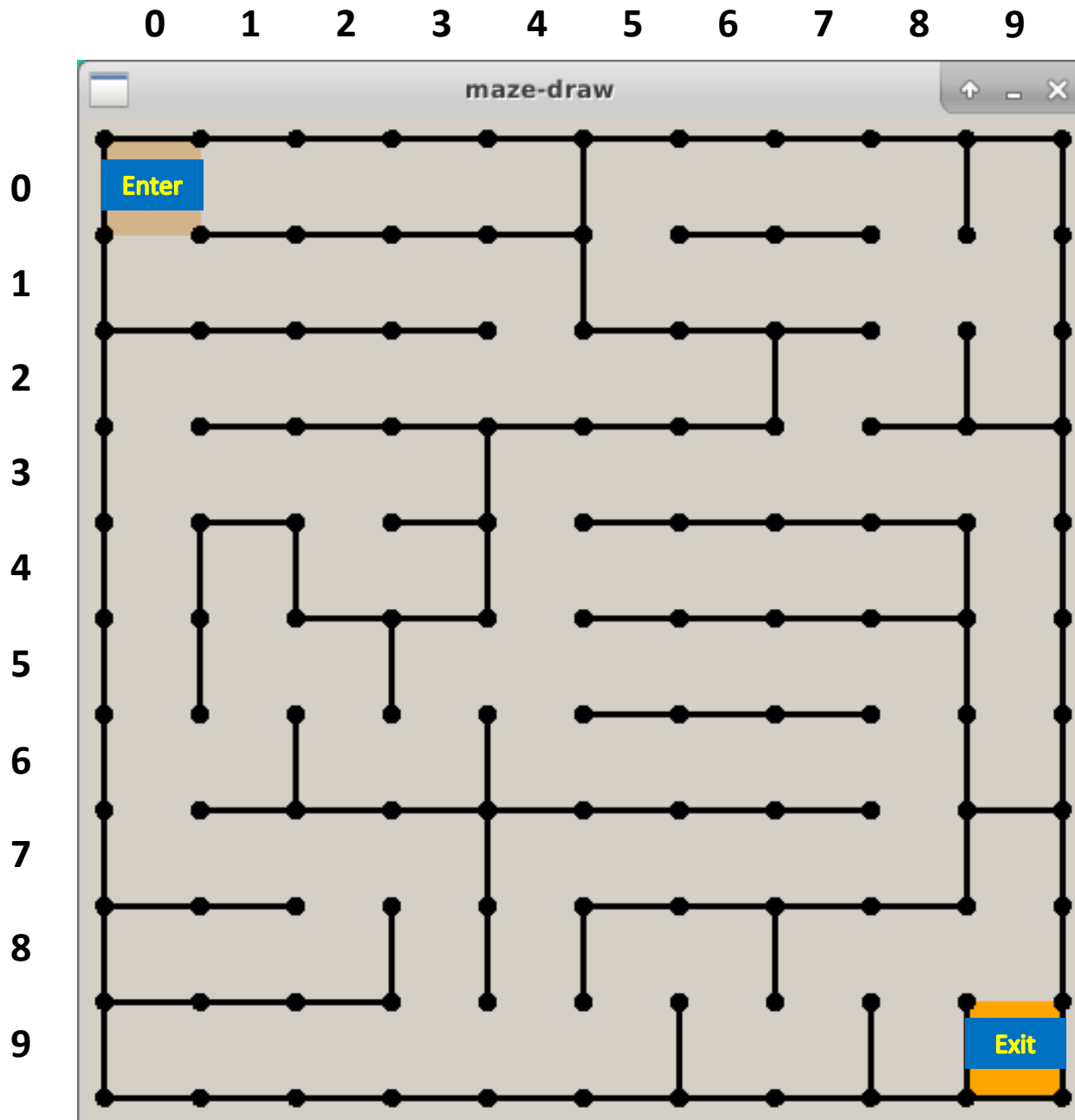
About C++ Arrays

- Matrices are named first by **row**, *then* by **column**
 - A (**3 x 2**) matrix has **3** rows and **2** columns
- In **Cartesian** coordinates
 - The abscissa **X** is listed first, followed by the ordinate **Y**: (**X**, **Y**)
 - Abscissa++ moves the world point **right**
 - Ordinate++ moves the world point **up**
- In **Array** coordinates
 - The **row** is listed first, followed by the **column**: [**Row**] [**Col**]
 - Row++ moves **downwards** (inverted like screen coordinates!)
 - Col-- moves **leftwards** to the previous element in the current Row

About C++ Arrays

	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	0,0	0,1	0,2	0,3	0,4
Row 1	1,0	1,1	1,2	1,3	1,4
Row 2	2,0	2,1	2,2	2,3	2,4

C++ arrays are 0-based



10 Rows
10 Cols

Y = Rows
X = Cols

**Positive Y
is down!**

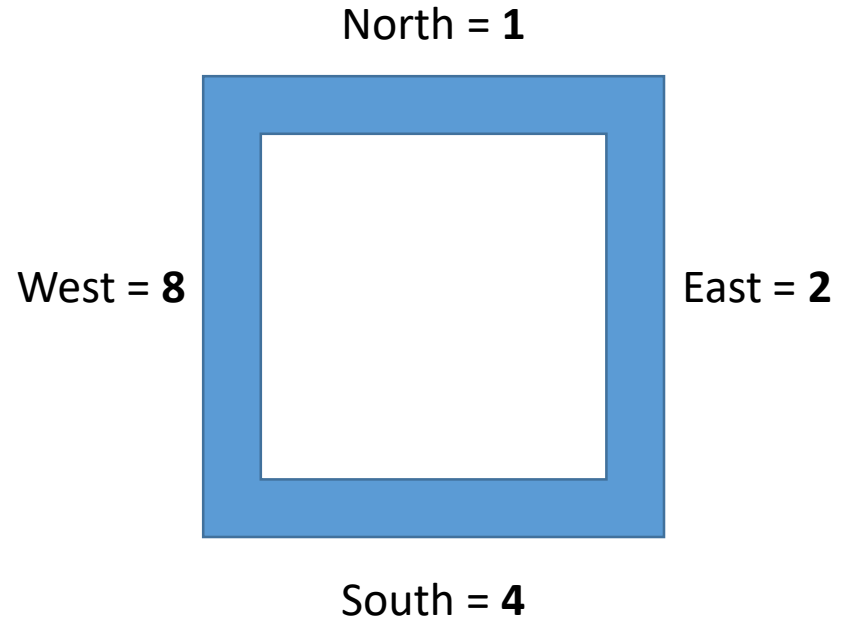
Entrance
= [0][0]

Exit
= [9][9]

How do we encode a maze?

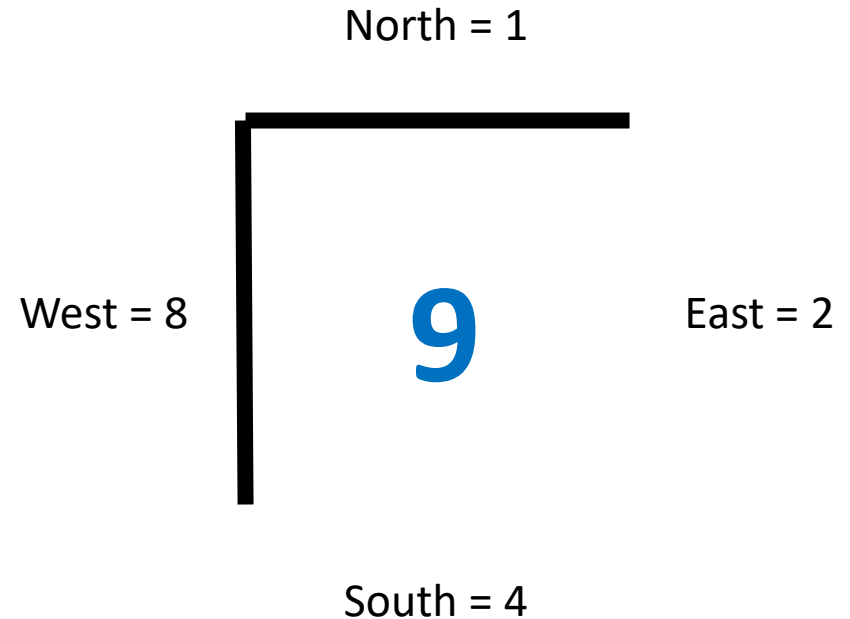
- Consider how to describe each individual square (cell) within the maze
- **How can we indicate for each individual cell if a wall exists to the North, East, South, or West direction?**

Encode each wall position as an increasing **power of 2**



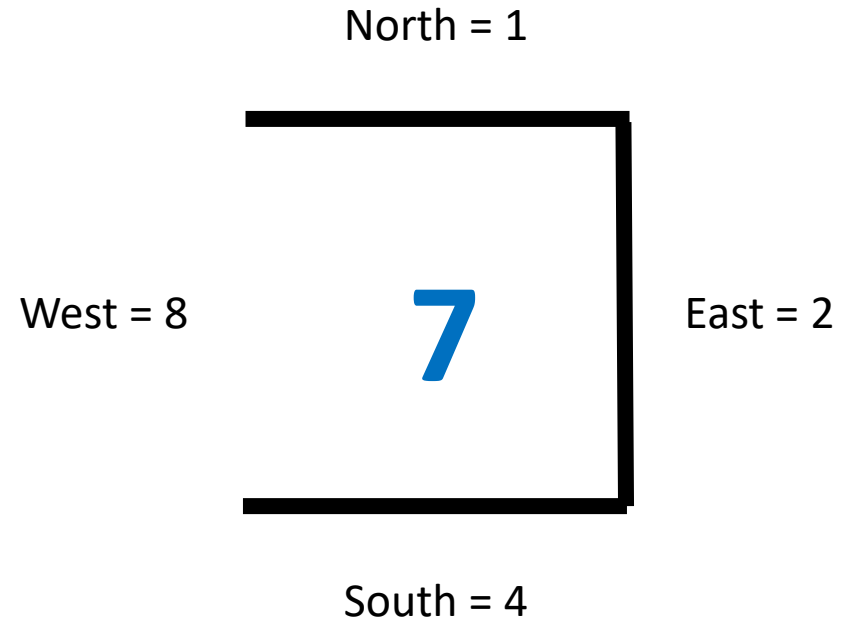
How do we encode a maze?

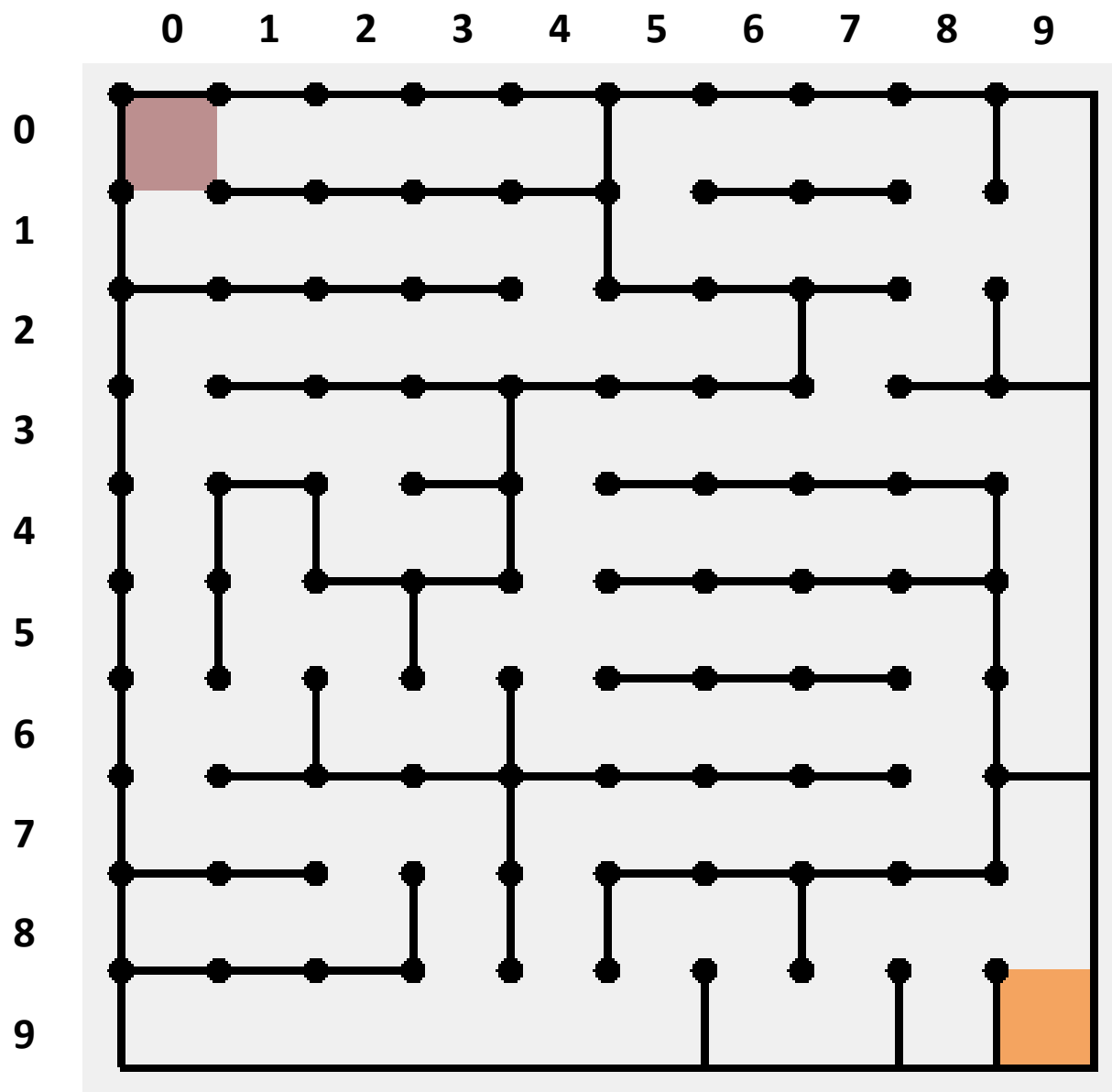
- If a wall exists in a given direction, add the value of that direction to the ***total*** cell value

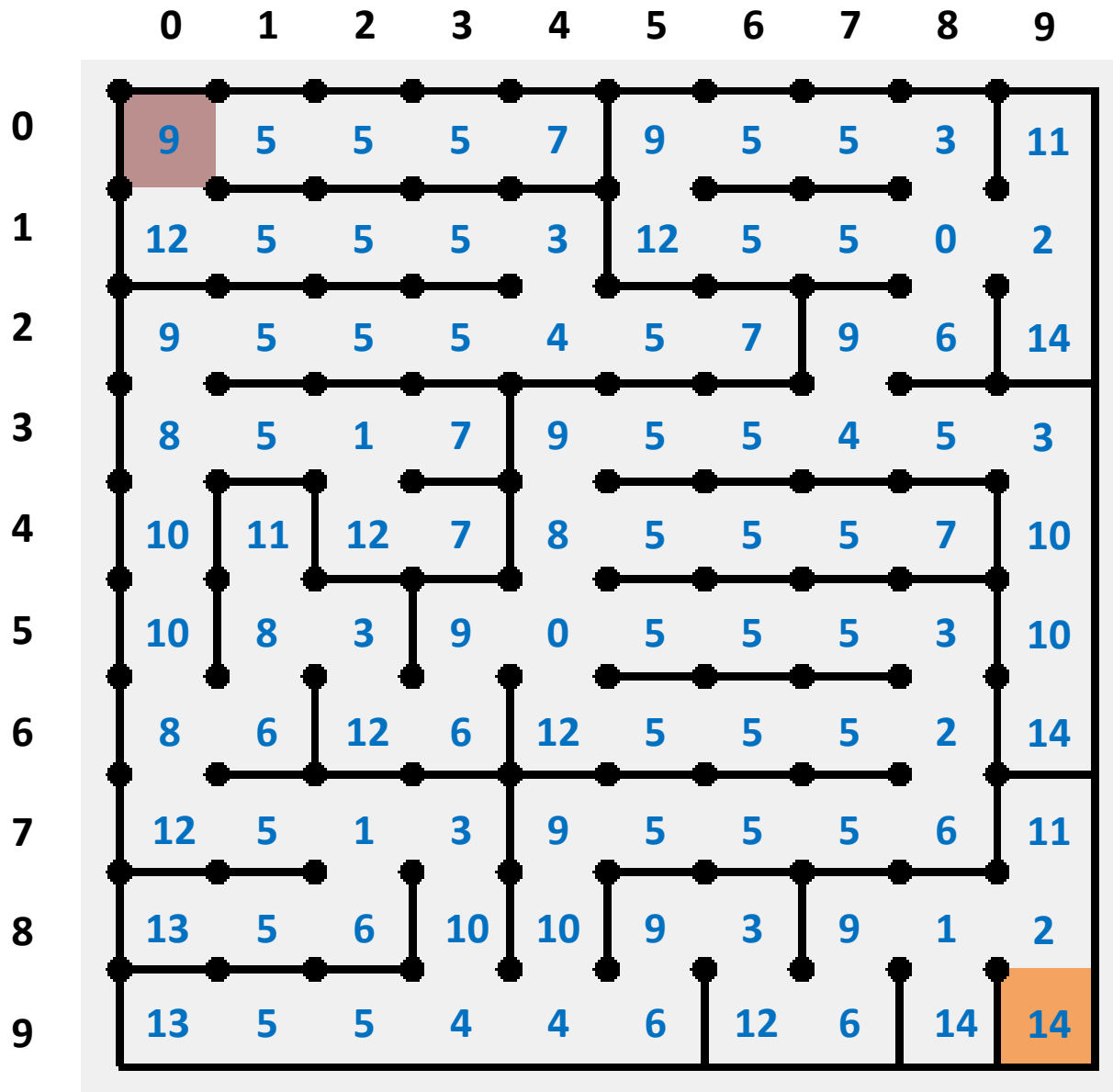


How do we encode a maze?

- Using a **power of 2** for each wall value produces an **unambiguous** encoding of all wall permutations
- $0 \leq \text{cell value} \leq 15$
- No walls = zero (0)
- 15 = a totally “walled-in” cell that is unreachable (*Hint: don't make a cell = 15*)







Drawing the 2D Maze

- Given a maze initialized with Base 2 wall encodings, how can we draw it?
- We could use a long series of `if()` statements to test all **16** possible values for a cell, and draw the required walls
- However, this would be inefficient in code size and run time
- We can take advantage of **bitwise AND** to figure out what walls to draw for a given cell

Bitwise Operators

- Logical operators use double ampersand or pipe characters
 - Boolean X **and** Y conditions: **if** (X && Y)
 - Boolean X **or** Y conditions: **if** (X || Y)
- **Bitwise** operators use single ampersand or pipe character
 - Bitwise X **and** Y values: X & Y
 - Bitwise X **or** Y values: X | Y
- Logical operators only return **true** or **false**
- **Bitwise** operators return a **integer value**
 - You normally only perform bitwise operations on **integers**

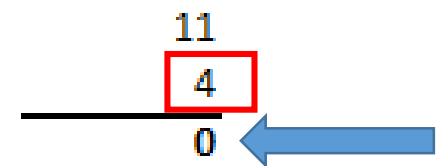
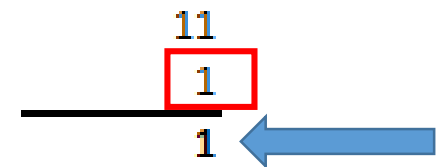
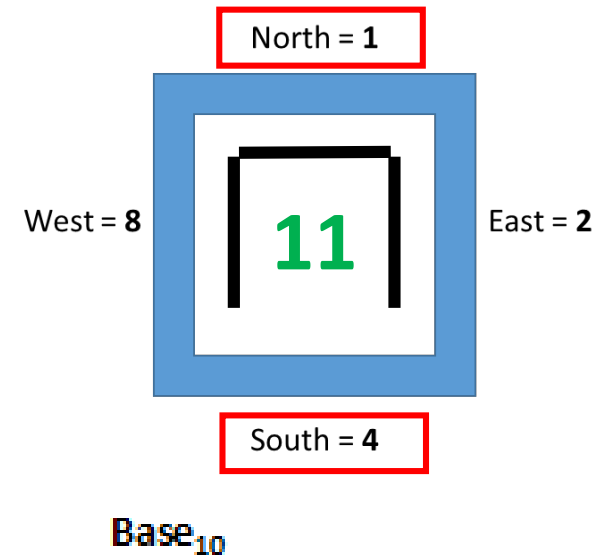
Bitwise AND

Binary (Base₂) Encoding

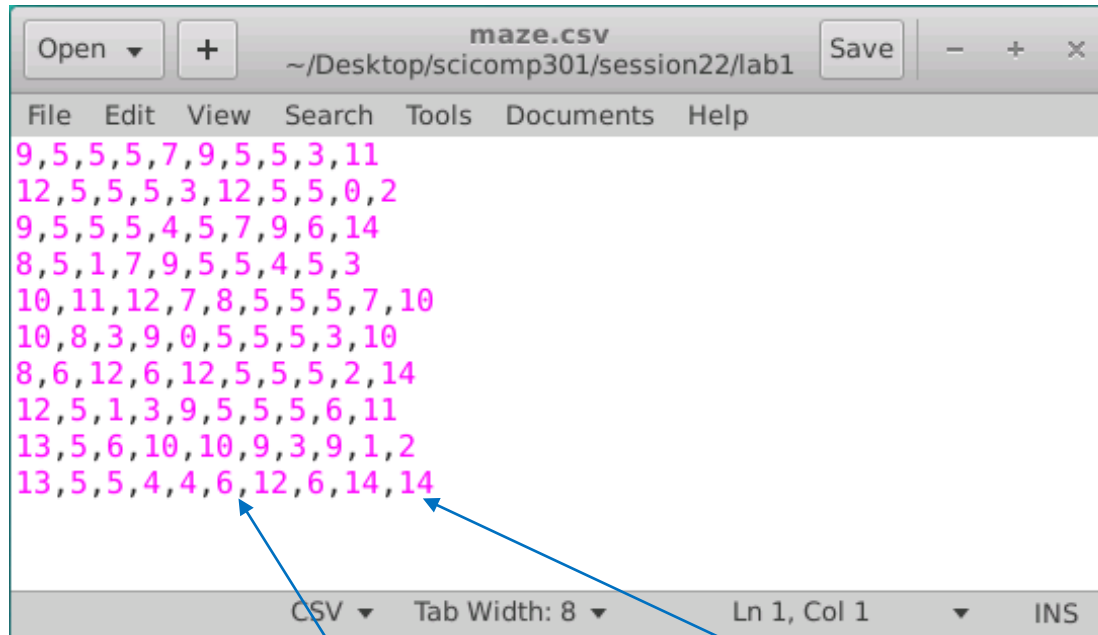
Position	3	2	1	0
Value	8	4	2	1

	1	0	1	1
AND	0	0	0	1
	0	0	0	1

	1	0	1	1
AND	0	1	0	0
	0	0	0	0



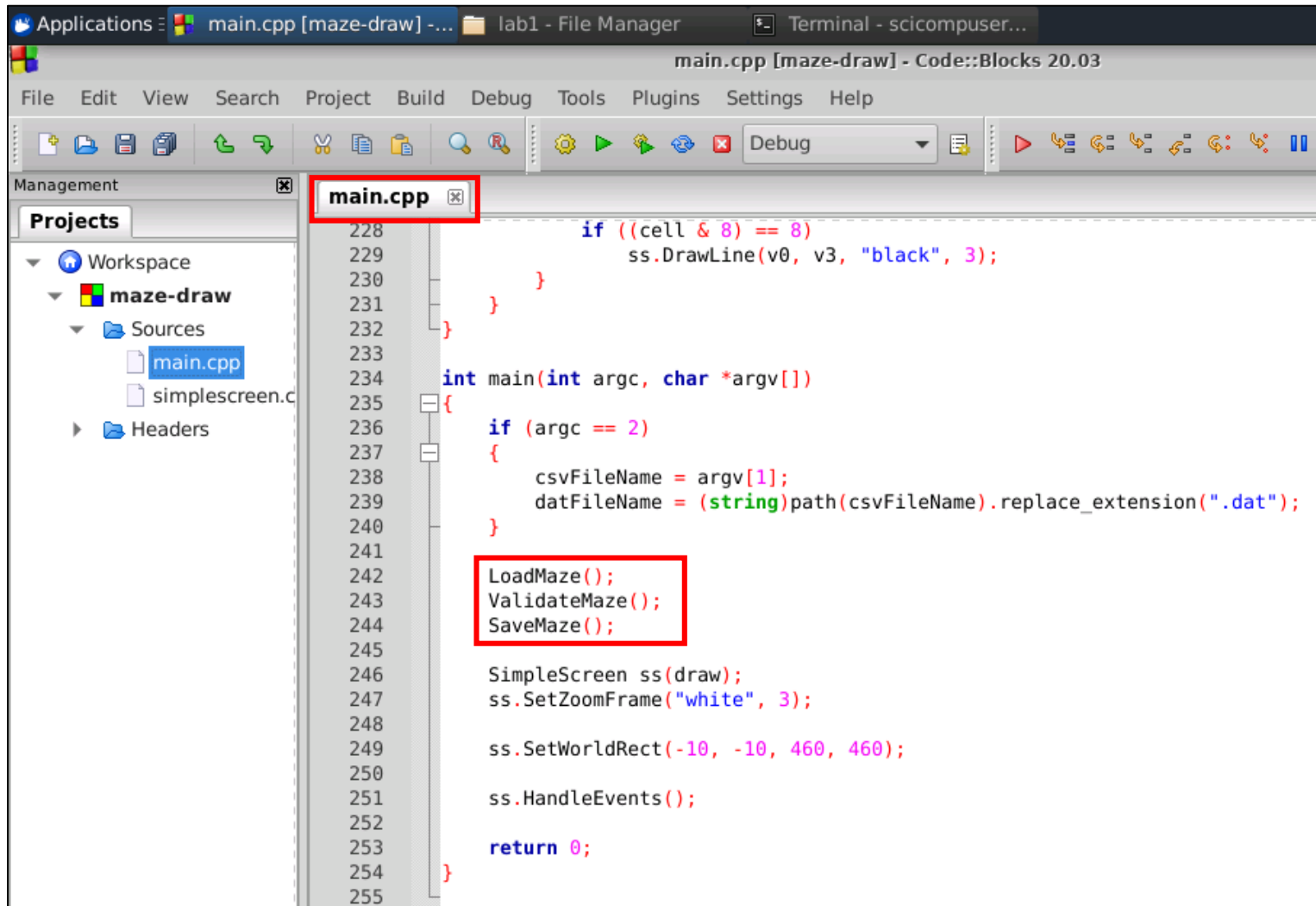
Encoding the Maze walls



```
maze.csv
~/Desktop/scicomp301/session22/lab1
File Edit View Search Tools Documents Help
9,5,5,5,7,9,5,5,3,11
12,5,5,5,3,12,5,5,0,2
9,5,5,5,4,5,7,9,6,14
8,5,1,7,9,5,5,4,5,3
10,11,12,7,8,5,5,5,7,10
10,8,3,9,0,5,5,5,3,10
8,6,12,6,12,5,5,5,2,14
12,5,1,3,9,5,5,5,6,11
13,5,6,10,10,9,3,9,1,2
13,5,5,4,4,6,12,6,14,14
CSV Tab Width: 8 Ln 1, Col 1 INS
```

CSV = Comma Separated Values
A “formatted” text file

Open Lab 1 – Maze Draw



```
228         if ((cell & 8) == 8)
229             ss.DrawLine(v0, v3, "black", 3);
230     }
231 }
232
233
234 int main(int argc, char *argv[])
235 {
236     if (argc == 2)
237     {
238         csvFileName = argv[1];
239         datFileName = (string)path(csvFileName).replace_extension(".dat");
240     }
241
242     LoadMaze();
243     ValidateMaze();
244     SaveMaze();
245
246     SimpleScreen ss(draw);
247     ss.SetZoomFrame("white", 3);
248
249     ss.SetWorldRect(-10, -10, 460, 460);
250
251     ss.HandleEvents();
252
253     return 0;
254 }
255
```

Reading data in TEXT format (CSV)

input file
stream

```
void LoadMaze()
{
    ifstream mazeFile(csvFileName, ios::binary);
    if (!mazeFile)
    {
        cout << "Cannot open " << csvFileName << endl;
        exit(-1);
    }

    string line{};
    const regex comma(",");
    for (int r{}; r < 10; r++)
    {
        getline(mazeFile, line);
        vector<string> row(
            sregex_token_iterator(line.begin(), line.end(), comma, -1),
            sregex_token_iterator());
        for (int c{}; c < 10; c++)
            maze[r][c] = stoi(row.at(c));
    }
}
```

string to integer

Open + maze.csv

1	9	5	5	5	7	9	5	5	3	11
2	12	5	5	5	3	12	5	5	0	2
3	9	5	5	5	4	5	7	9	6	14
4	8	5	1	7	8	5	5	4	5	3

		0	1	2	3	4	5	6	7	8	9
		A	B	C	D	E	F	G	H	I	J
0	1	9	5	5	5	7	9	5	5	3	11
1	2	12	5	5	5	3	12	5	5	0	2
2	3	9	5	5	5	4	5	7	9	6	14

Writing data in BINARY format (DAT)

output file
stream

```
void SaveMaze()  
{  
    ofstream mazeFile(datFileName, ios::binary);  
    for (int r = 0; r < 10; r++)  
        for (int c = 0; c < 10; c++)  
            mazeFile.write((char *)&maze[r][c], sizeof(char));  
}
```

maze.csv



Lab 1



maze.dat

```

void draw(SimpleScreen& ss)
{
    // Draw maze (rows by cols)
    for (int r = 0; r < 10; r++) {
        double y0 = (9 - r) * 45;
        double y1 = (9 - r) * 45 + 45;
        for (int c = 0; c < 10; c++) {
            double x0 = c * 45;
            double x1 = c * 45 + 45;

            Point2D v0(x0, y0); // Lower-left vertex
            Point2D v1(x1, y0); // Lower-right vertex
            Point2D v2(x1, y1); // Upper-right vertex
            Point2D v3(x0, y1); // Upper-left vertex

            // Draw entrance cell
            if (r == 0 && c == 0)
                ss.DrawRectangle("tan", v0.x, v0.y, 45, 45, 1, true);
            // Draw exit cell
            if (r == 9 && c == 9)
                ss.DrawRectangle("orange", v0.x, v0.y, 45, 45, 1, true);

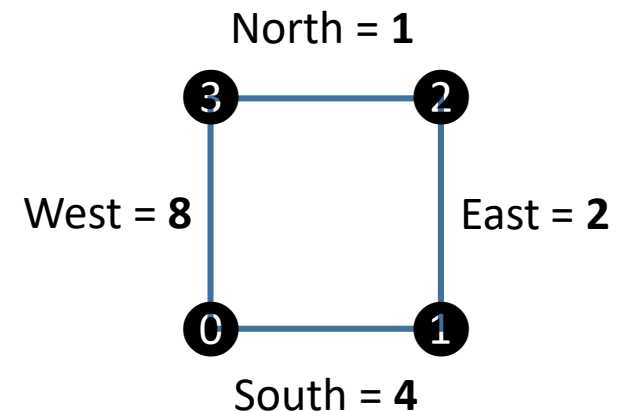
            // Draw cell corner circles
            ss.DrawCircle(v0.x, v0.y, 2, "black", 5);
            ss.DrawCircle(v1.x, v1.y, 2, "black", 5);
            ss.DrawCircle(v2.x, v2.y, 2, "black", 5);
            ss.DrawCircle(v3.x, v3.y, 2, "black", 5);

            int cell = maze[r][c];
            // Draw north wall if required
            if ((cell & 1) == 1)
                ss.DrawLine(v2, v3, "black", 3);
            // Draw west wall if required
            if ((cell & 2) == 2)
                ss.DrawLine(v1, v2, "black", 3);
            // Draw south wall if required
            if ((cell & 4) == 4)
                ss.DrawLine(v0, v1, "black", 3);
            // Draw east wall if required
            if ((cell & 8) == 8)
                ss.DrawLine(v0, v3, "black", 3);
        }
    }
}

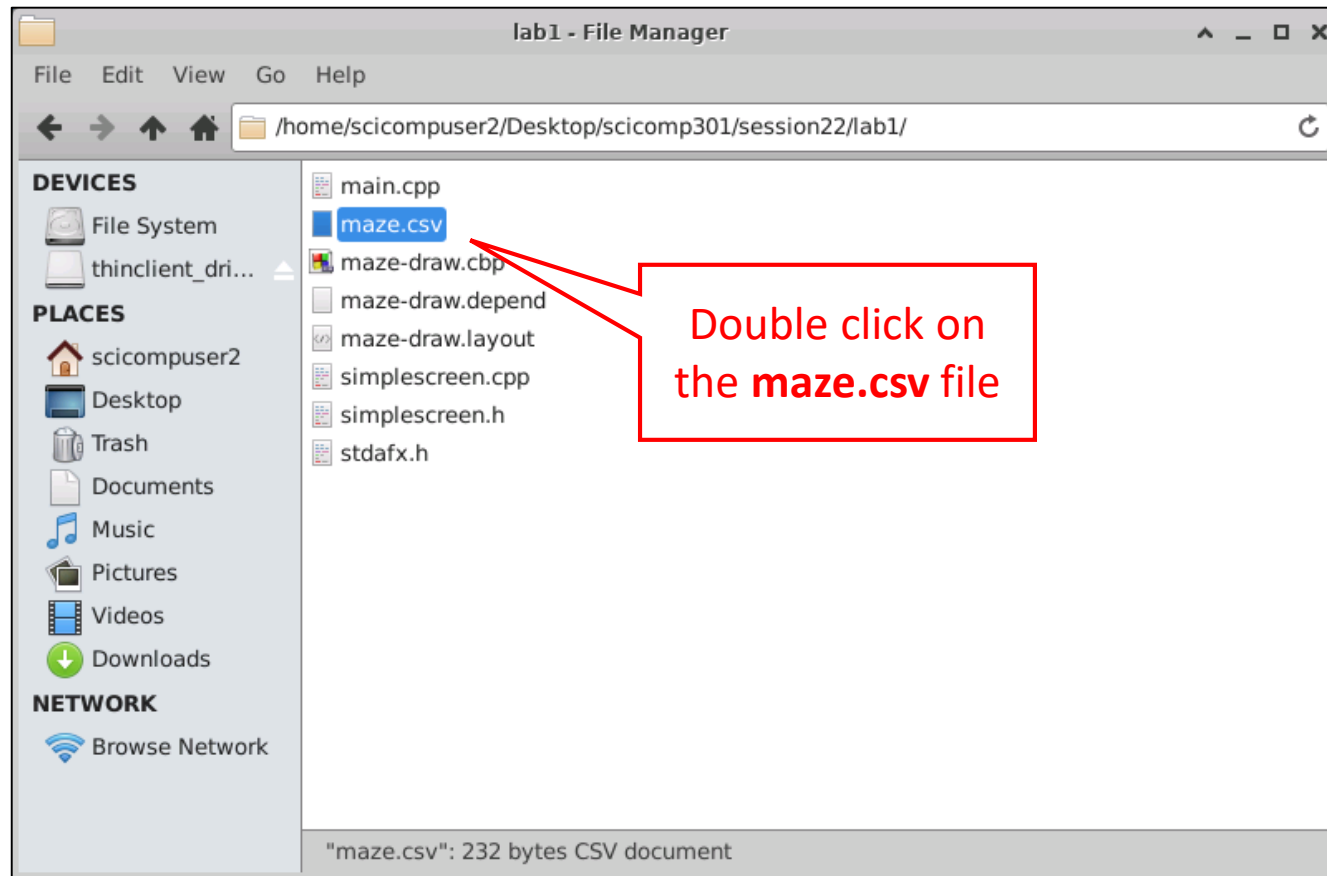
```

Lab 1

Maze Draw



Encoding the Maze walls



Encoding the Maze walls

Text Import - [maze.csv]

Import

Character set: Unicode (UTF-8)

Language: Default - English (USA)

From row: 1

Separator Options

☐ Fixed width ☒ Separated by

☒ Tab ☒ Comma ☒ Semicolon ☐ Space ☐ Other

☐ Merge delimiters

Text delimiter: "

Other Options

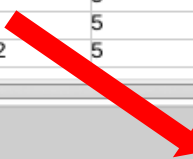
☐ Quoted field as text ☐ Detect special numbers

Fields

Column type:

	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard
1	9	5	5	5	7	9	5	5	3
2	12	5	5	5	3	12	5	5	0
3	9	5	5	5	4	5	7	9	6
4	8	5	1	7	9	5	5	4	5
5	10	11	12	7	8	5	5	5	7
6	10	8	3	9	0	5	5	5	3
7	8	6	12	6	12	5	5	5	2
8	12	5	1	5	6	5	5	5	5

Help OK Cancel



Encoding the Maze walls

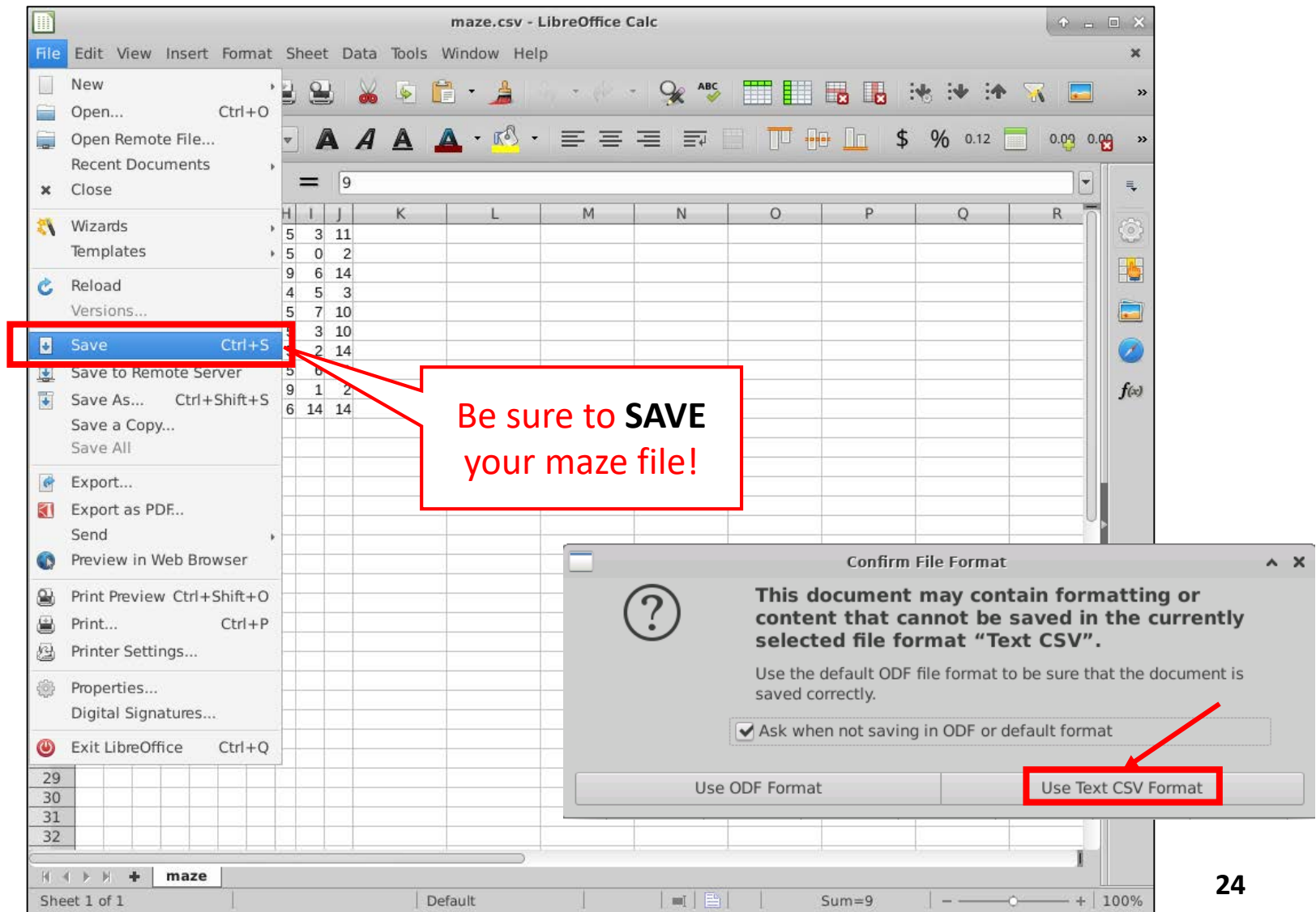
LibreOffice Calc window titled "maze.csv". The spreadsheet shows a 10x10 matrix of values representing maze walls. The matrix is located in the range A1 to J10. The values are as follows:

	A	B	C	D	E	F	G	H	I	J
1	9	5	5	5	7	9	5	5	3	1
2	12	5	5	5	3	12	5	5	0	2
3	9	5	5	5	4	5	7	9	6	4
4	8	5	1	7	9	5	5	4	5	3
5	10	11	12	7	8	5	5	5	7	0
6	10	8	3	9	0	5	5	5	3	0
7	8	6	12	6	12	5	5	5	2	4
8	12	5	1	3	9	5	5	5	6	1
9	13	5	6	10	10	9	3	9	1	2
10										14

A red box highlights the matrix area with the text: "Ensure you maintain a 10 x 10 matrix as you enter your own maze cell values".

The status bar shows "Sum=9".

Encoding the Maze walls



Run Lab 1 – Maze Draw

The screenshot shows a C++ IDE with the following components:

- main.cpp** (line 218-226):

```
218 if ((cell & 2) == 2)
219     ss.DrawLine(v1, v2, "black", 3);
220 // Draw south wall if required
221 if ((cell & 4) == 4)
222     ss.DrawLine(v0, v1, "black", 3);
223 // Draw east wall if required
224 if ((cell & 8) == 8)
225     ss.DrawLine(v0, v3, "black", 3);
226
```
- Terminal** (maze-draw):

	0	1	2	3	4	5	6	7	8	9
0	9	5	5	5	7	9	5	5	3	11
1	12	5	5	5	3	12	5	5	0	2
2	9	5	5	5	4	5	7	9	6	14
3	8	5	1	7	9	5	5	4	5	3
4	10	11	12	7	8	5	5	5	7	10
5	10	8	3	9	0	5	5	5	3	10
6	8	6	12	6	12	5	5	5	2	14
7	12	5	1	3	9	5	5	5	6	11
8	13	5	6	10	10	9	3	9	1	2
9	13	5	5	4	4	6	12	6	14	14

Maze is valid!
- main window** (maze-draw): A 10x10 grid maze with a green start cell at (0,0) and an orange end cell at (9,9). The maze is drawn with black lines on a gray background.

Lab 1 – Maze Draw

maze-draw

File Edit View Terminal Tabs Help

	0	1	2	3	4	5	6	7	8	9
0	9	5	5	5	7	9	5	5	3	11
1	12	5	5	5	3	12	5	5	1	2
2	9	5	5	5	4	5	7	9	6	14
3	8	5	1	7	9	5	5	4	5	3
4	10	11	12	7	8	5	5	5	7	10
5	10	8	3	9	0	5	5	5	3	10
6	8	6	12	6	12	5	5	5	2	14
7	12	5	1	3	9	5	5	5	6	11
8	13	5	6	10	10	9	3	9	1	2
9	13	5	5	4	4	6	12	6	14	14

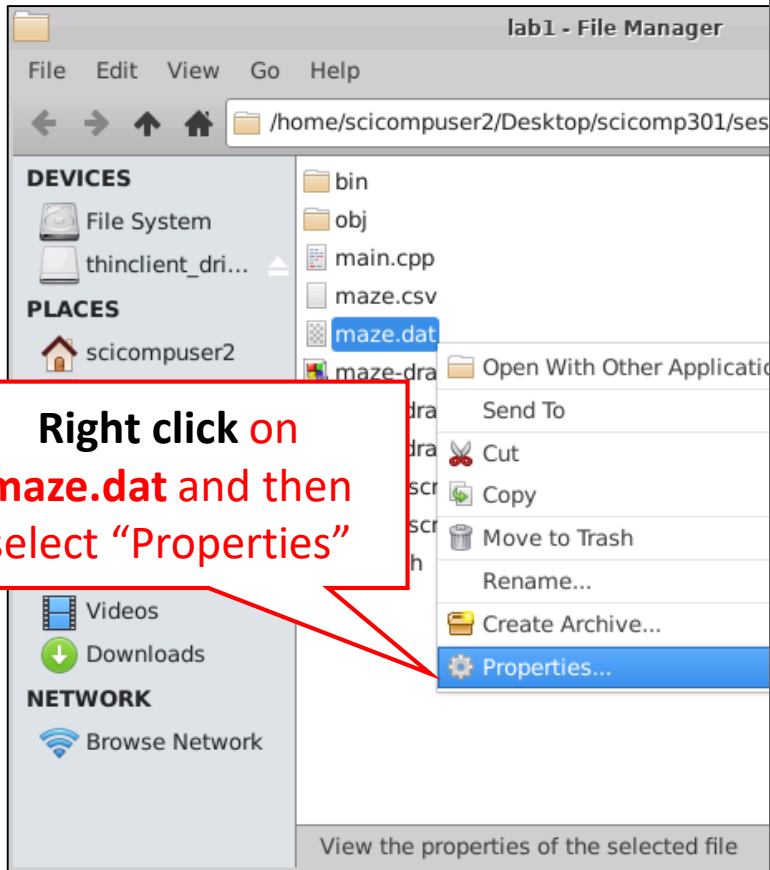
Cells (1,8) and (0,8) do not agree in direction 1

Process returned 255 (0xFF) execution time : 0.037 s
Press ENTER to continue.

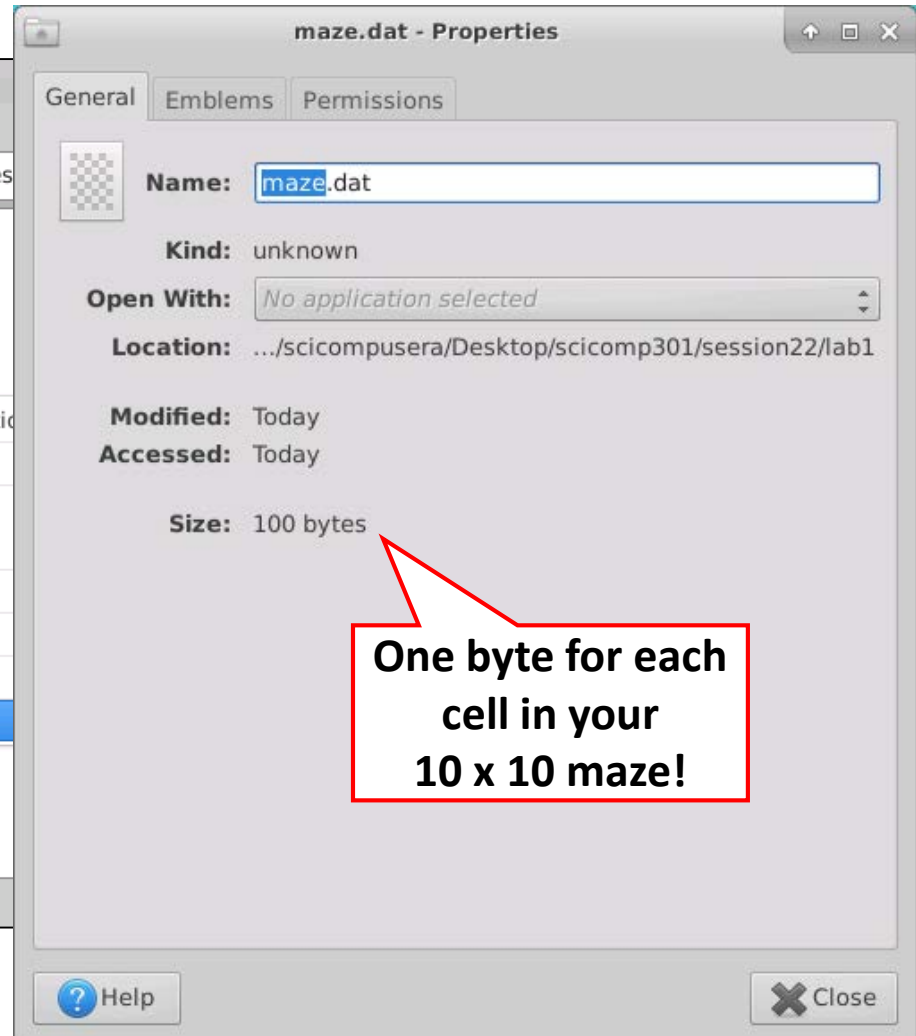
Adjacent cells must agree on their wall values

```
graph LR;
    1(( )) --- 1 --- 2(( ))
    2 --- 2 --- 3(( ))
    3 --- 3 --- 4(( ))
    4 --- 4 --- 1
```

Lab 1 – Binary Output File

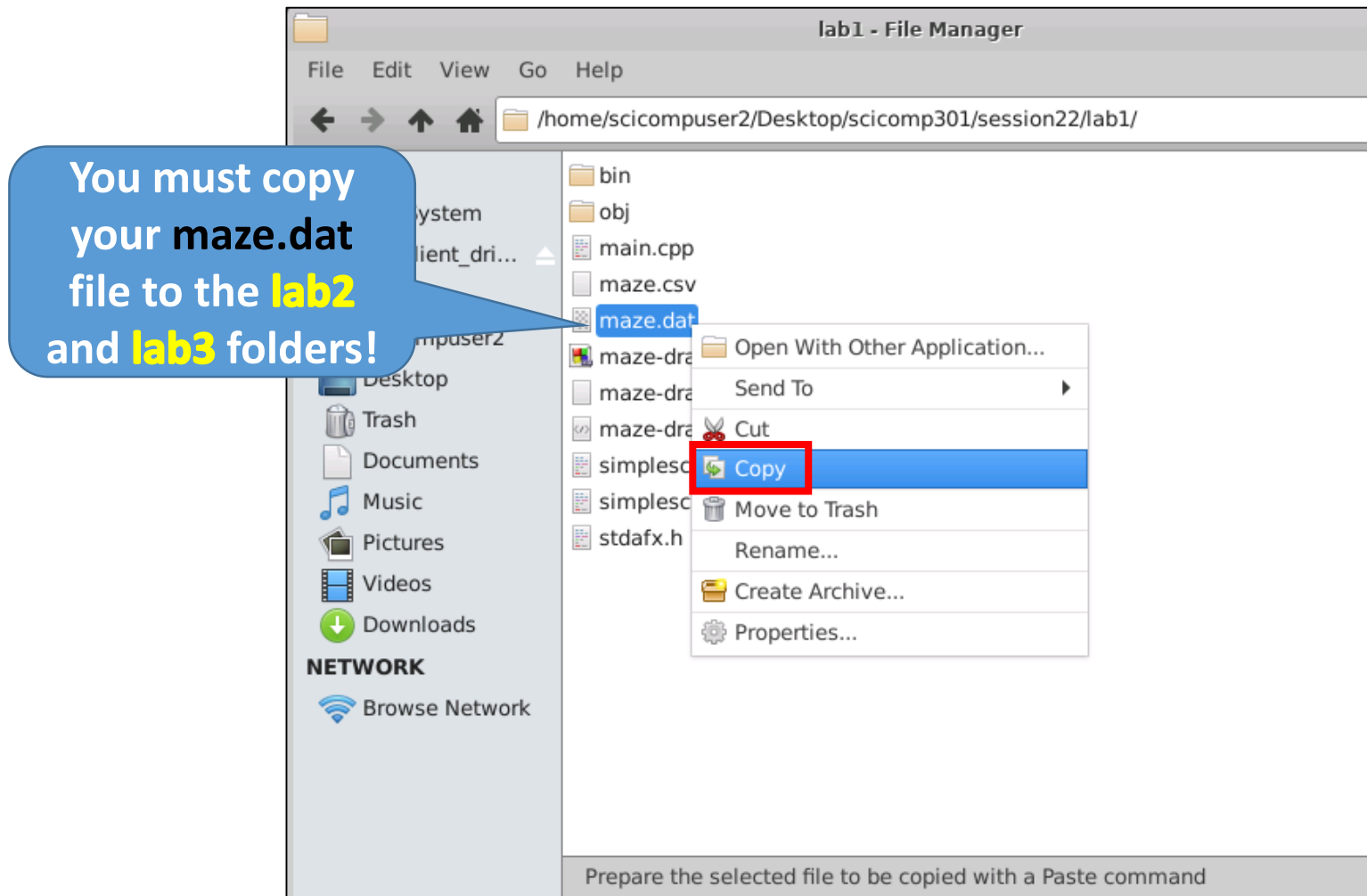


Right click on
maze.dat and then
select "Properties"



One byte for each
cell in your
10 x 10 maze!

Lab 1 – Binary Output File



Depth-First Search

- Depth-first is a sequential search algorithm
 - It is just you alone in the maze, you have no helpers
 - It is a zero *prior* knowledge, recursive, backtracking approach
 - You have **breadcrumbs** to mark your cell visitation history
- Order of search in each cell is **North, East, South, West**
 - We can only proceed in a direction if there is no wall in the path:

if (cell value & direction) != direction

North (1)	→	(Δ row = -1, Δ column = 0)
East (2)	→	(Δ row = 0, Δ column = 1)
South (4)	→	(Δ row = 1, Δ column = 0)
West (8)	→	(Δ row = 0, Δ column = -1)

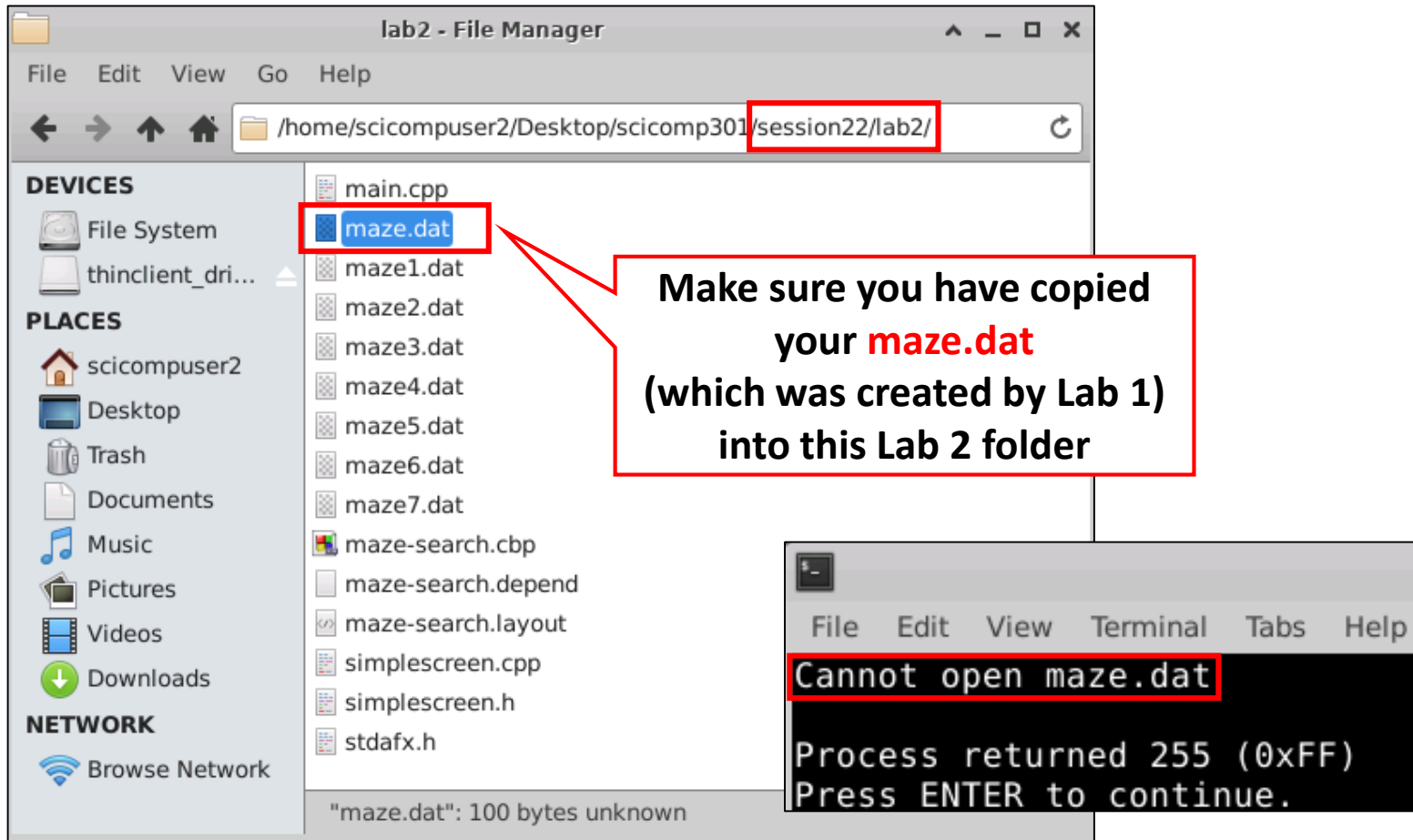
Depth-First Search Algorithm

1. Drop a **breadcrumb** as you enter each cell
2. Take a step in the very first direction that is **open**, and go to step #1
3. If there are no more open directions in the cell, retrace your steps **backwards** until you reach a cell with a breadcrumb where **the next open direction** is one you have not taken yet
4. Take a step in that new direction, and go to step #1
5. Stop with you reach the **exit square**

Depth-First Search Breadcrumbs

- A **breadcrumb** matrix (array) can contain a simple **bool** value to indicate if you have previously visited this cell
- Breadcrumbs prevent going around in endless circles and never finding the exit
- In this program we use an **int visitCount** array, so we can color the path according to the number of times we've visited each cell (1=**Blue**, 2=**Green**, 3=**Red**, 4=**Orange**)
- Lots of **red** and **orange** squares in the path indicates an *inefficient* search pattern, because you are visiting the same node too many times!

Lab 2 – Maze Search



The image shows a file manager window titled "lab2 - File Manager" with a menu bar (File, Edit, View, Go, Help) and a address bar containing "/home/scicompuser2/Desktop/scicomp301/session22/lab2/". The left sidebar shows "DEVICES" (File System, thinclient_dri...) and "PLACES" (scicompuser2, Desktop, Trash, Documents, Music, Pictures, Videos, Downloads). The main pane lists files: main.cpp, maze.dat (highlighted with a red box), maze1.dat, maze2.dat, maze3.dat, maze4.dat, maze5.dat, maze6.dat, maze7.dat, maze-search.cbp, maze-search.depend, maze-search.layout, simplescreen.cpp, simplescreen.h, and stdafx.h. A red callout box points to maze.dat with the text: "Make sure you have copied your **maze.dat** (which was created by Lab 1) into this Lab 2 folder". Below the file manager is a terminal window with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal displays the error "Cannot open maze.dat" (highlighted with a red box), followed by "Process returned 255 (0xFF)" and "Press ENTER to continue."

lab2 - File Manager

File Edit View Go Help

/home/scicompuser2/Desktop/scicomp301/session22/lab2/

DEVICES

- File System
- thinclient_dri...

PLACES

- scicompuser2
- Desktop
- Trash
- Documents
- Music
- Pictures
- Videos
- Downloads

NETWORK

- Browse Network

main.cpp
maze.dat
maze1.dat
maze2.dat
maze3.dat
maze4.dat
maze5.dat
maze6.dat
maze7.dat
maze-search.cbp
maze-search.depend
maze-search.layout
simplescreen.cpp
simplescreen.h
stdafx.h

"maze.dat": 100 bytes unknown

Make sure you have copied your **maze.dat** (which was created by Lab 1) into this Lab 2 folder

File Edit View Terminal Tabs Help

Cannot open maze.dat

Process returned 255 (0xFF)
Press ENTER to continue.

Lab 2 – Maze Search

```
void LoadMaze()
{
    ifstream mazeFile(datFileName, ios::binary);
    if (!mazeFile)
    {
        cout << "Cannot open " << datFileName << endl;
        exit(-1);
    }

    for (int r = 0; r < 10; r++)
    {
        for (int c = 0; c < 10; c++)
        {
            mazeFile.read((char *)&maze[r][c], sizeof(char));
            crumbs[r][c] = false;
        }
    }
}
```

Lab2 reads in the
binary file **maze.dat**

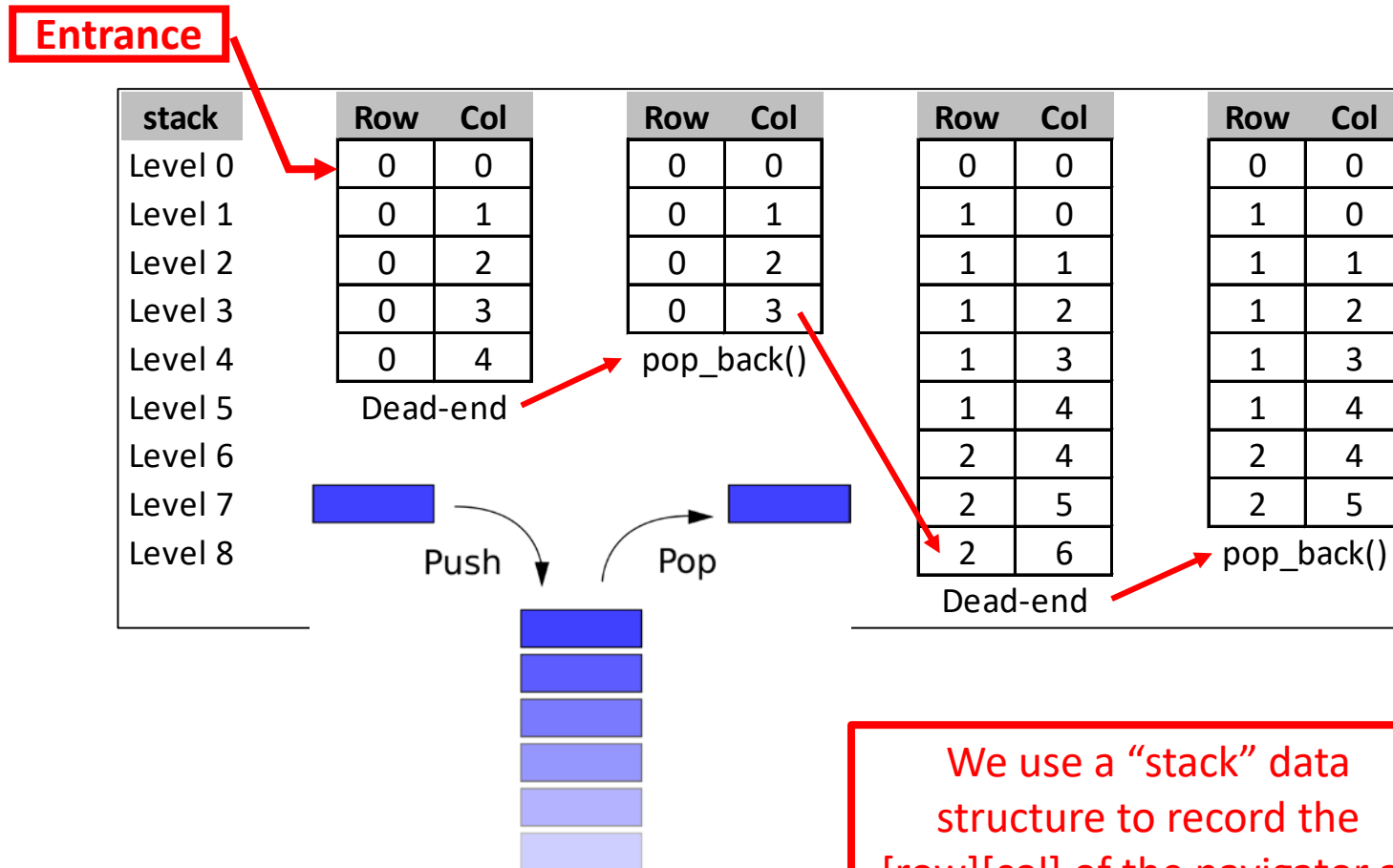
Lab 2

Maze Search

```
bool TakeStep()
{
    int r = get<0>(stack.back());
    int c = get<1>(stack.back());
    int r2 = r; int c2 = c;
    int dir = get<2>(stack.back());
    if (dir == 1) r2--;
    if (dir == 2) c2++;
    if (dir == 4) r2++;
    if (dir == 8) c2--;
    get<2>(stack.back()) *= 2;
    bool moved = false;
    if (((maze[r][c] & dir) != dir) &&
        visitCount[r2][c2] == 0)
    {
        tuple<int, int, int> cell(r2, c2, 1);
        stack.push_back(cell);
        moved = true;
    }
    if (dir == 16 && !moved) {
        stack.pop_back();
        r2 = get<0>(stack.back());
        c2 = get<1>(stack.back());
        moved = true;
    }
    if (moved) {
        visitCount[r2][c2]++;
        totalSteps++;
        if (r2 == 9 && c2 == 9)
            foundExit = true;
        return true;
    }
    return false;
}
```

The program tries to take a step in each direction, updating the **visitCount** array with each step

Lab 2 – Maze Draw



We use a “stack” data structure to record the [row][col] of the navigator at each step through the maze

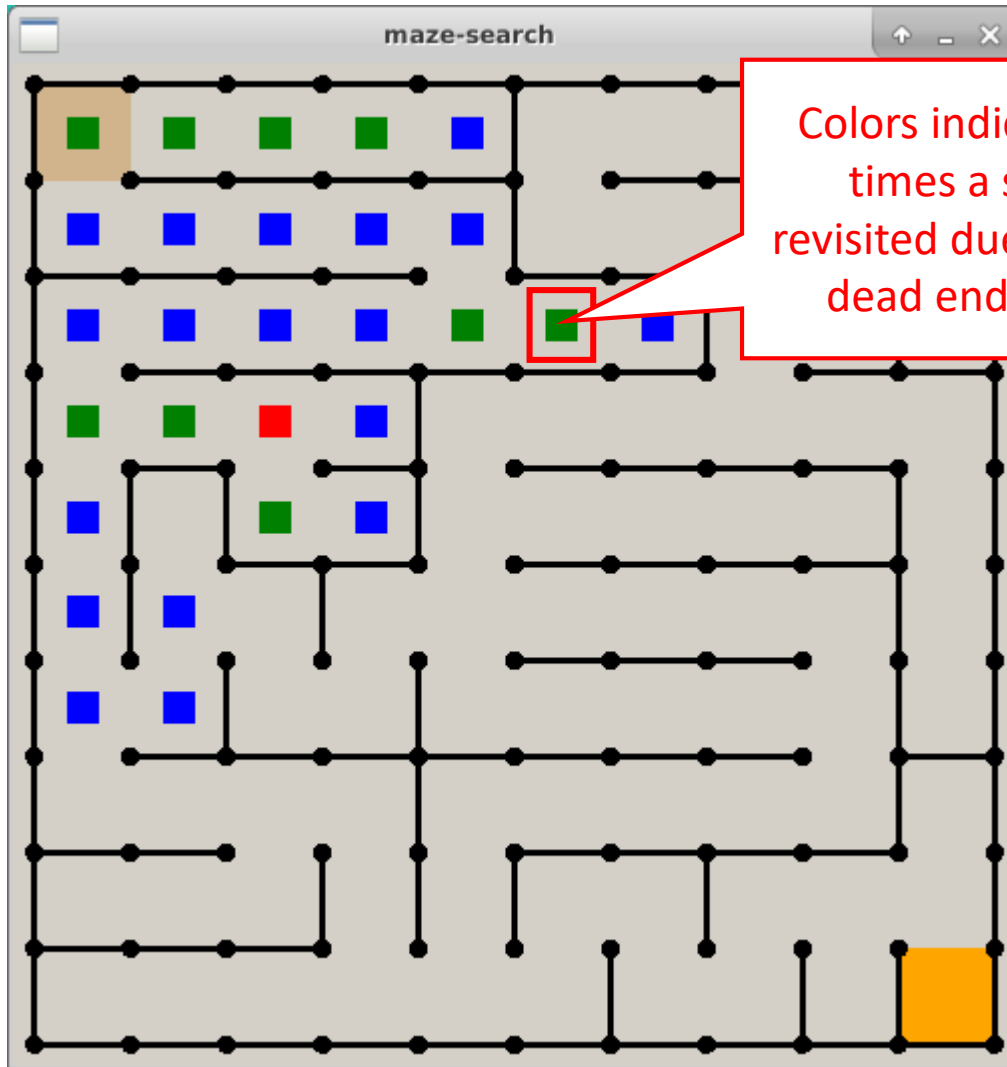
Lab 2 – Maze Search

```
void eventHandler(SimpleScreen& ss, ALLEGRO_EVENT& ev)
{
    if (ev.type == ALLEGRO_EVENT_KEY_CHAR) {
        if (ev.keyboard.keycode == ALLEGRO_KEY_S) {
            if (!foundExit) {
                while (!TakeStep());
            }
            if (foundExit) {
                cout << "Exit found!" << endl
                    << "Total steps = " << totalSteps << endl
                    << "Path steps = " << stack.size() - 1 << endl;
                ResetVisitCount();
                for (auto s : stack)
                    visitCount[get<0>(s)][get<1>(s)] = 1;
                ss.Clear();
                DrawMaze(ss);
            }
            ss.Redraw();
        }
    }
}
```

Press **S** to
take a
single step

When the exit is found,
your path will be shown
(after removing any
backup steps)

Lab 2 – Maze Search

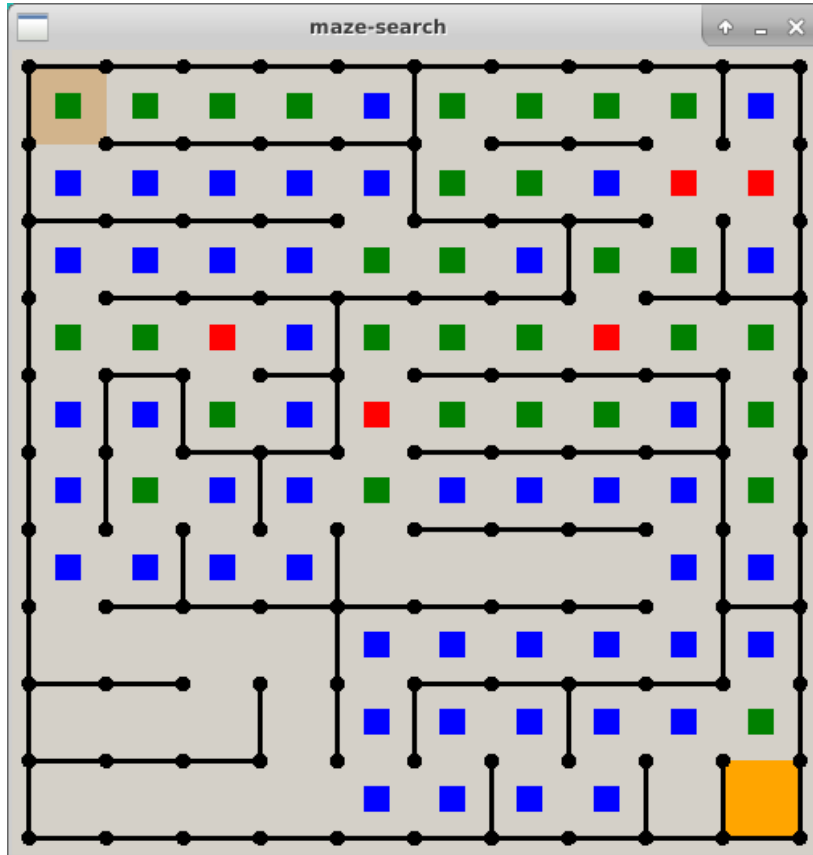


Press **S** to
take a
single step

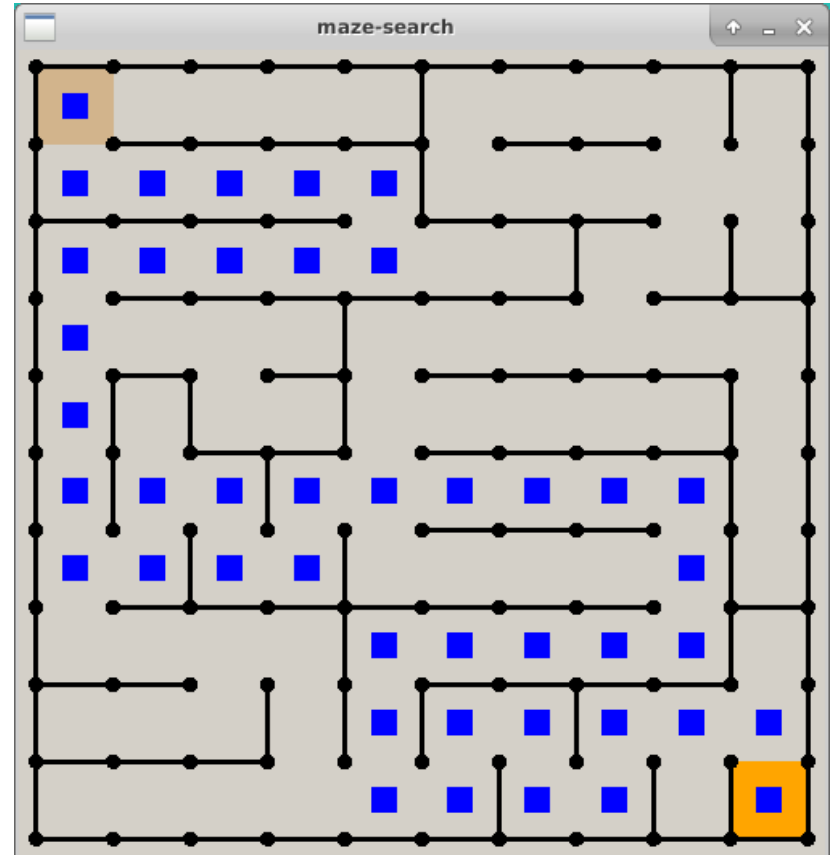
Press **F** to
full step to
finish

Lab 2 – Maze Search

Right *before* stepping into exit cell



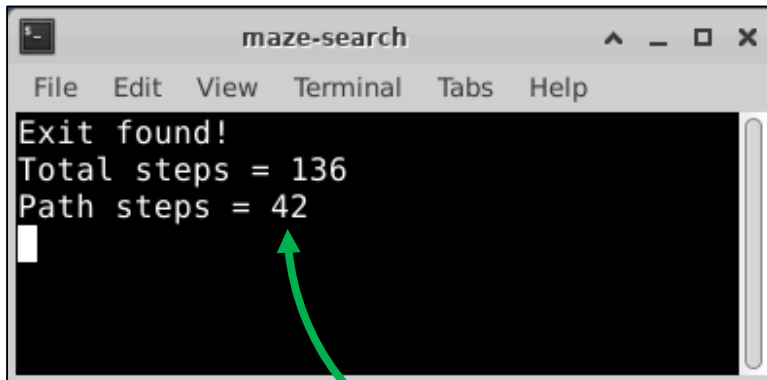
Right *after* stepping into exit cell



When the exit is found, your path will be shown
(after removing any backup steps)

Lab 2 – Maze Search

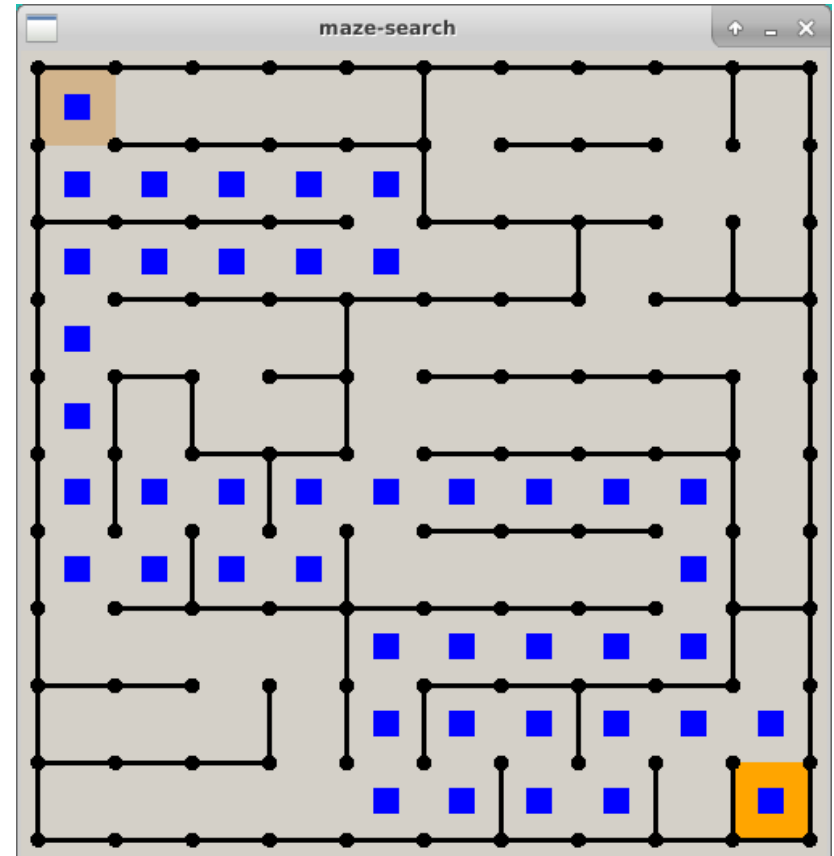
Right *after* stepping into exit cell



```
maze-search
File Edit View Terminal Tabs Help
Exit found!
Total steps = 136
Path steps = 42
```

A terminal window titled 'maze-search' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The output shows 'Exit found!', 'Total steps = 136', and 'Path steps = 42'. A green arrow points from the 'Path steps' line to the definition box below.

- **Total steps** = How many you had to take counting backup steps
- **Path steps** = The **best path you found** minus any backup steps



When the exit is found, your path will be shown
(after removing any backup steps)

Improving Depth-First Search Efficiency

- The unaided depth-first search spends considerable time exploring paths which clearly are not on the *optimal* route – it hits a lot of **dead ends**
- What if we could calculate the **shortest path length**, and start **backtracking** the instant our current path length \geq the *shortest* path length?
- It is possible to calculate the minimum number of steps from entrance to exit (*the* shortest path) ***without searching one cell or taking one step!***

Adjacency Matrix

3 x 3 maze
9 total cells

	Col 0	Col 1	Col 2
Row 0	9	7	11
Row 1	10	9	2
Row 2	12	6	14

Adjacency Matrix

Every maze square (cell) is represented along **both** the rows and columns of an adjacency matrix

Row	Col
0	0
0	1
0	2
1	0
1	1
1	2
2	0
2	1
2	2

Col	0	1	2	0	1	2	0	1	2
Row	0	1	2	3	4	5	6	7	8
0	1	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	0	0	1	0	0	0
3	1	0	0	1	0	0	1	0	0
4	0	0	0	0	1	1	0	1	0
5	0	0	1	0	1	1	0	0	1
6	0	0	0	1	0	0	1	1	0
7	0	0	0	0	1	0	1	1	0
8	0	0	0	0	0	1	0	0	1

A **true** (1) indicates you **can** reach the other cell in **just one step**
A **false** (0) means that you **cannot** reach the other cell in just one step

Adjacency Matrix

The main diagonal has all one values
because every cell can reach itself by definition

Col	0	1	2	0	1	2	0	1	2
Row	0	0	0	1	1	1	2	2	2
0	0	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	0	0	1	0	0	1	0	0	0
3	1	0	0	1	0	0	1	0	0
4	0	0	0	0	1	1	0	1	0
5	0	0	1	0	1	1	0	0	1
6	0	0	0	1	0	0	1	1	0
7	0	0	0	0	1	0	1	1	0
8	0	0	0	0	0	1	0	0	1

The whole matrix is symmetric about the main diagonal
because there are no “one-way” doors in the maze.

Reflexive Property: If you can get from cell A to cell B in one step,
then you can also get from cell B to cell A in one step

Adjacency Matrix

- To find the **shortest path**, we keep **AND**ing (using the *logical* operator **&&**) the adjacency matrix against itself until a **true** value appears in the matrix element that represents the **exit cell**
- The number of times we had to “multiply” (AND) the adjacency matrix by itself **equals the minimal # of steps** from entrance to exit
- **Ironically, we can know the # of steps in the shortest path, but not what the actual steps are!**

Adjacency Matrix

- If it is **true** you can get from **A to B**, and it is **true** you can get from **B to C**, then it must be **true** that you can get from **A to C** (**transitive property**)
- When “multiplying” **bool** adjacency matrices, if the **AND** of all the elements in (Row A x Col B) **== true** then the cell is **set to true**
 - We keep multiplying the adjacency matrix until a **true** appears in the cell that represents the exit square.
 - The total # of matrix multiplications required = **the shortest path length from entrance to exit**

Adjacency Matrix

- We can calculate the adjacency matrix **before** starting a depth-first search. We can then use this **shortest path length** to limit the current search path to improve the efficiency of the search
- Once the current **stack.size()** has more levels than the shortest path length calculated from the adjacency matrix, **start back tracking!**
- There is no reason to continue on a path which has a step count greater than the known shortest path
 - **it's best to backup and try a new direction**

Lab 3 – Maze Search

Lab 2 – Maze Search w/o Adj Matrix

```
int main()
{
    LoadMaze();
    ResetVisitCount();

    tuple<int, int, int> entrance(0, 0, 1);
    stack.push_back(entrance);
    visitCount[0][0] = 1;

    SimpleScreen ss(draw, eventHandler);
    ss.SetZoomFrame("white", 3);
    ss.SetWorldRect(-10, -10, 460, 460);

    DrawMaze(ss);

    ss.HandleEvents();

    return 0;
}
```

Lab 3 – Maze Search w Adj Matrix

```
int main()
{
    LoadMaze();
    ResetVisitCount();

    AdjMatrix adj;
    adj.Init(maze);
    minSteps = adj.MinSteps();

    tuple<int, int, int> entrance(0, 0, 1);
    stack.push_back(entrance);
    visitCount[0][0] = 1;

    SimpleScreen ss(draw, eventHandler);
    ss.SetZoomFrame("white", 3);
    ss.SetWorldRect(-10, -10, 460, 460);

    DrawMaze(ss);

    ss.HandleEvents();

    return 0;
}
```

The variable **minSteps** holds the # of steps in the shortest path from entrance to exit

Lab 3 – Maze Search

Lab 2 – Maze Search w/o Adj Matrix

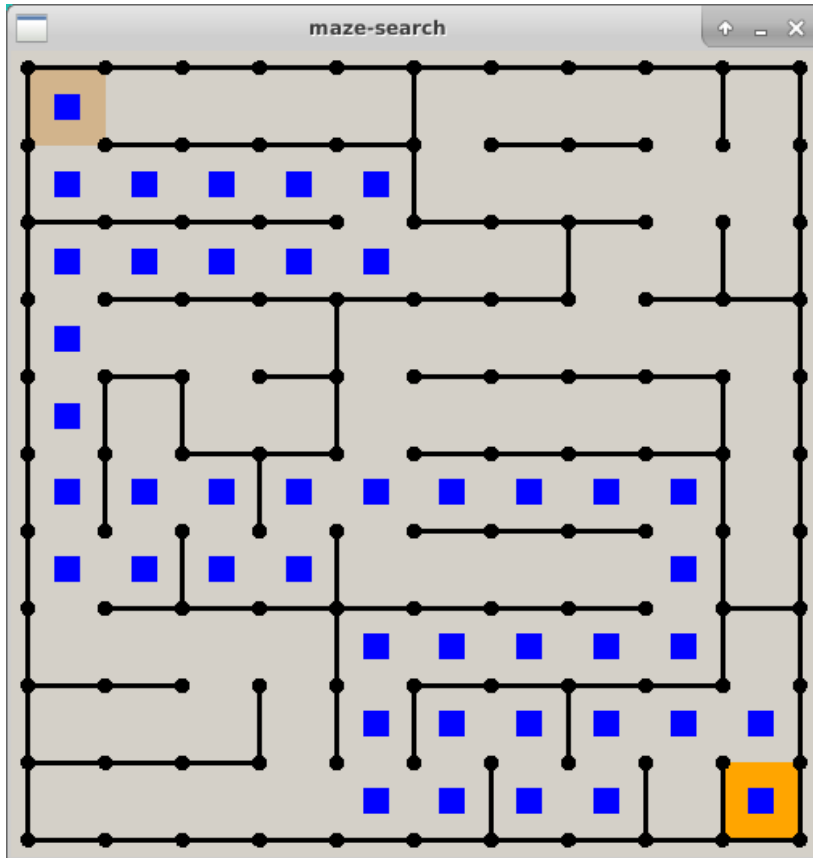
```
bool TakeStep()
{
    int r = get<0>(stack.back());
    int c = get<1>(stack.back());
    int r2 = r; int c2 = c;
    int dir = get<2>(stack.back());
    if (dir == 1) r2--;
    if (dir == 2) c2++;
    if (dir == 4) r2++;
    if (dir == 8) c2--;
    get<2>(stack.back()) *= 2;
    bool moved = false;
    if (((maze[r][c] & dir) != dir) &&
        visitCount[r2][c2] == 0)
    {
        tuple<int, int, int> cell(r2, c2, 1);
        stack.push_back(cell);
        moved = true;
    }
    if (dir == 16 && !moved) {
        stack.pop_back();
        r2 = get<0>(stack.back());
        c2 = get<1>(stack.back());
        moved = true;
    }
    if (moved) {
        visitCount[r2][c2]++;
        totalSteps++;
        if (r2 == 9 && c2 == 9)
            foundExit = true;
        return true;
    }
    return false;
}
```

Lab 3 – Maze Search w Adj Matrix

```
bool TakeStep()
{
    int r = get<0>(stack.back());
    int c = get<1>(stack.back());
    int r2 = r; int c2 = c;
    int dir = get<2>(stack.back());
    if (dir == 1) r2--;
    if (dir == 2) c2++;
    if (dir == 4) r2++;
    if (dir == 8) c2--;
    get<2>(stack.back()) *= 2;
    bool moved = false;
    if (((maze[r][c] & dir) != dir) &&
        (visitCount[r2][c2] == 0) &&
        (stack.size() <= minSteps))
    {
        tuple<int, int, int> cell(r2, c2, 1);
        stack.push_back(cell);
        moved = true;
    }
    if (dir == 16 && !moved) {
        stack.pop_back();
        r2 = get<0>(stack.back());
        c2 = get<1>(stack.back());
        moved = true;
    }
    if (moved) {
        visitCount[r2][c2]++;
        totalSteps++;
        if (r2 == 9 && c2 == 9)
            foundExit = true;
        return true;
    }
    return false;
}
```

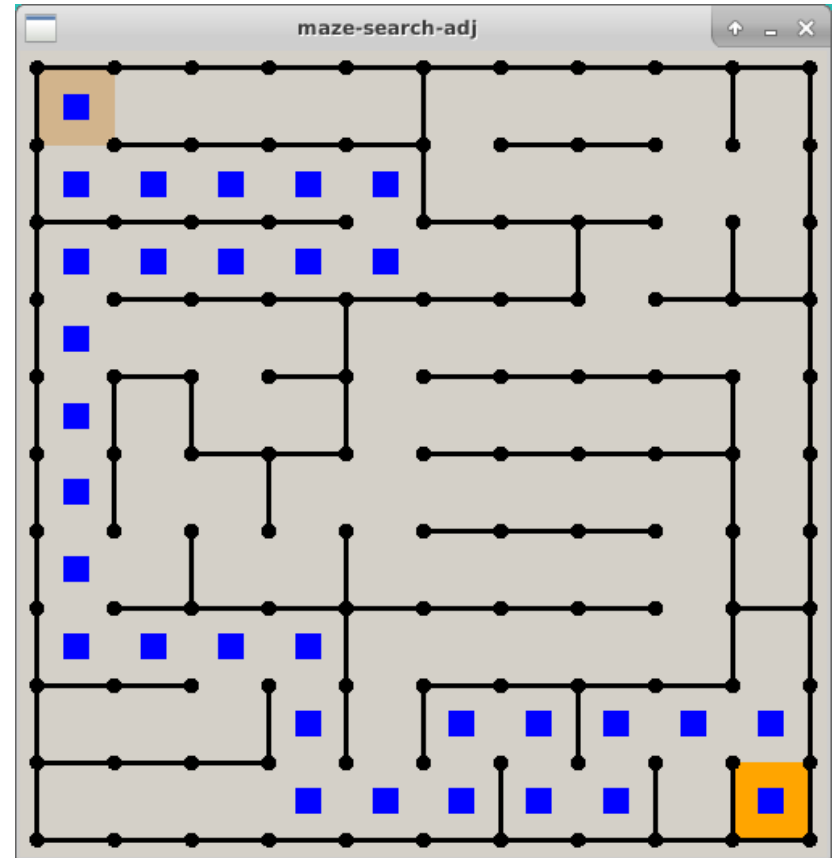

Lab 3 – Maze Search

Lab 2 – Maze Search w/o Adj Matrix



Path Steps = 42

Lab 3 – Maze Search with Adj Matrix



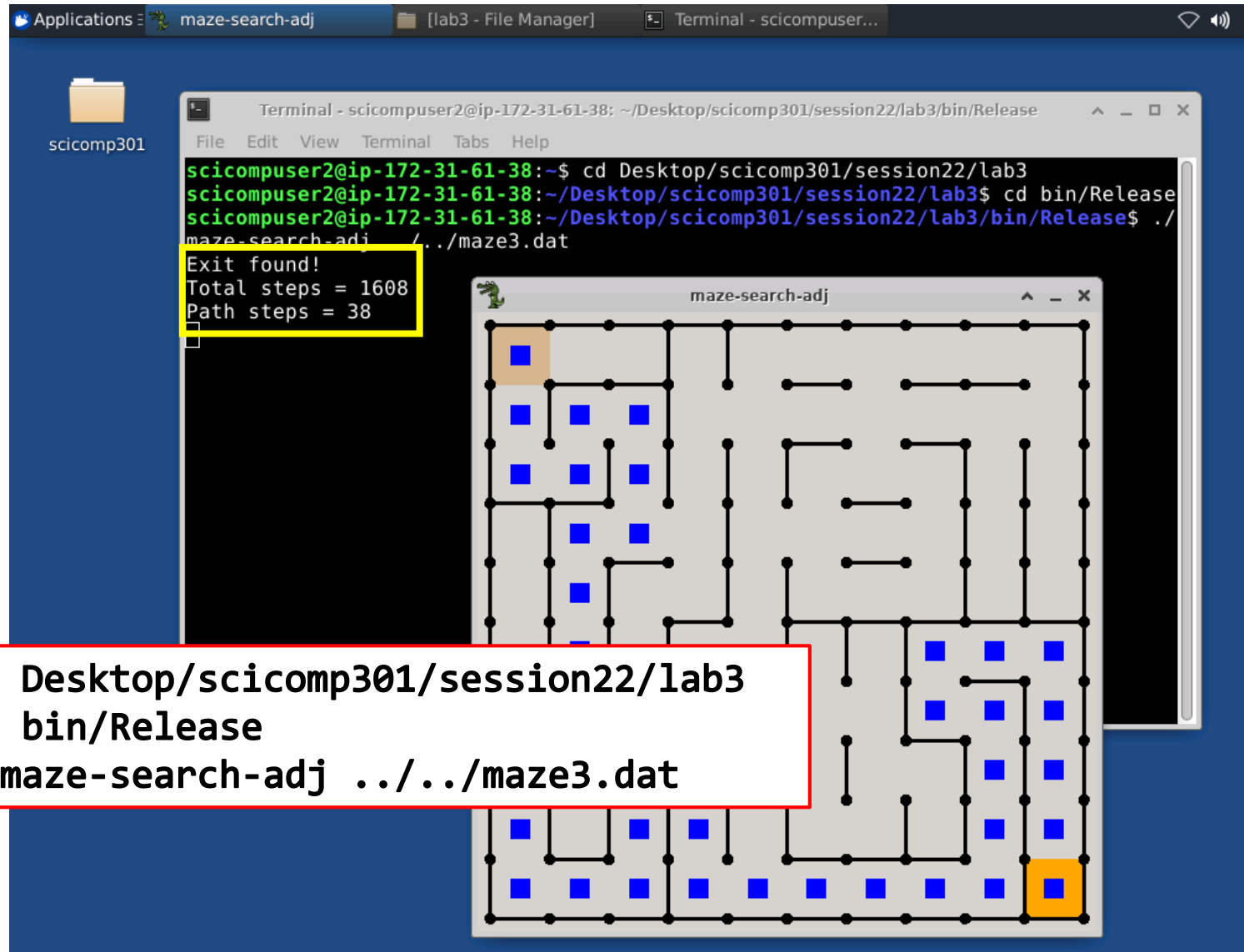
Path Steps = 30

Lab 3 – Maze Search

	maze-search	maze-search-adj	
Data File	Depth First	Shortest Path	% Reduction
maze1.dat	42	30	28.57%
maze2.dat	56	30	46.43%
maze3.dat	38	38	0.00%
maze4.dat	38	26	31.58%
maze5.dat	32	22	31.25%
maze6.dat	26	18	30.77%
maze7.dat	32	26	18.75%
Average:			26.76%

On average using an adjacency matrix reduces the steps in a depth-first search by **25%**

Lab 3 – Maze Search



```
scicompuser2@ip-172-31-61-38: ~/$ cd Desktop/scicomp301/session22/lab3
scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session22/lab3$ cd bin/Release
scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session22/lab3/bin/Release$ ./
maze-search-adj ../maze3.dat
Exit found!
Total steps = 1608
Path steps = 38
```

cd Desktop/scicomp301/session22/lab3
cd bin/Release
./maze-search-adj ../maze3.dat

Now you know...

- How to encode 2D maze walls in **base 2**
 - In C++ an **[Y][X]** matrix means there are **Y rows** and **X columns**
 - The bitwise AND (&) operator can decode wall values
- Depth-first search is implemented with recursion or a **stack**
 - You must use a **breadcrumbs** array to prevent infinite loops
- A **logical adjacency matrix** can be used to calculate the length of the shortest path from entrance to exit
 - However, the adjacency matrix will not identify the actual steps along that shortest path
 - Leveraging the adjacency matrix during a depth-first search will yield on average a **25% improvement** in the efficiency of the search