Project Management (Reverse)
Project Details
Choosing Your Variables
Same problem/solution for Project Maze

- Enumeration
- Backtracking
- Systematic

The 8 queens problem solved by

CSI 310: Lecture 24 (2005)
The gbd/ddc debugger's print or display commands reveal an object's value:

```
mychar = 's';
```

as the corresponding argument. E.g., `ctn>`mychar; functions with a non-constant reference parameter: Called with the object.

Assignment operators: With the object on the LHS. E.g., `mychar = s'`;

```
value = contents = state
```

The value is set and modified by:

```
Each object has a runtime (dynamic)
```

```
OBJECT = VARIABLE = INSTANCE
```

In C++ terminology, these words are synonyms:
array: (Could be an array of anything.)

definitions or declarations.

struct or class: Programmer uses library defined by struct/class type

scalar: ALL pointers, char, int, double, bool, enum, etc.

There are 3 categories of type:

(characters)

type, that set has 256 elements in it. Only about 1/3 of it is printable
for the 8-bit char.
The computer can do with it, and the set of possible values. For each object has a compile time (static) type, which determines what operations
object after the lifetime of that object has expired.

DANGER: A dangling pointer value is an address when used to access an
OR, that memory might remain unused. (You cannot predict)

If stream or other library.

After the lifetime expires, that memory resource might be recycled, which

Like a lease which give you the right to occupy an apartment.

The lifetime is the period of time during which the memory resource that

Each object (synonym variable) has a LIFETIME (or extant).

University at Albany Computer Science Dept.
A constructor is called when execution flows into the definition of an
same variable name.
allocated in the execution stack, so multiple variables exist (are alive) with the
NEW for C1310: When a function recursively, multiple activation records for it are
when the corresponding function activation activation
The lifetime ENDS when execution leaves the containing block; in particular,
Taught first in C1201. Most common and elementary:

```c
{ 
  . .
  int HeresAnAutomaticArray[1039];
}

main() {
  int *anotherAuto;
  MakeSomething;
  Automatic: (or Local Extant) (NOT Local scope)
  In C++, there are 3 basic kinds of Lifetime (which came from C):
```
Garbage collection, but leaves memory in C++. Coding logically like (new Thing()->dostuff()); is OK in Java because of allocated.

You need an additional pointer type variable to hold the address returned.

Example of code that allocates one useless instance of a Maze and never deletes.

People who prefer a confusing terminology (Dynamical variables are allocated in the "free store" (called "heap") by some returned by the corresponding new.

It begins when new is run, and ends when delete is called on the address.

(In C, library functions malloc() and free() are used instead.)

This lifetime is controlled by explicitly coded new and delete operations.

The 2nd basic kind of lifetime is DYNAMIC.
have local scope except those declared extern.

Another confusing feature of C is static inside a function body makes the
the variable's name have the scope instead of global scope.

Another favorite feature of classical programs, like Linus Torvalds, is the
classical/C style code often has static variables shared by several functions.

Be static.

Defining a variable declared outside of all function bodies makes that variable
The static lifetime (logically) spans the entire run of the program.

The 3rd basic kind of lifetime is static.
instance) containing them. So, all 3 kinds are possible.

All other data members have the same lifetime as the particular object (i.e.,

(only one instance for that class).

static data members have STATIC lifetime. They are called "class variables."

What about data members of classes/structs?
void printPath(pathListNode *node) {
    if (node == NULL) return

    // indices is an empty hexagon inside the maze.
    // bool isEmpty(int i, int j) returns true iff
    // tool is Empty (i.e., maze[i][j] == 0, a maze that holds our maze.
    // invariant: maze[0..N-1][0..N-1] holds our maze.
    // int N; // input: sx, sy, tx, ty; // input data storeage.

    private:
    } }

class Maze {
    public:
        // include "pathListNode.h"

#include "maze.h"
#include "pathListNode.h"
define Maze included
define Maze included

#include "Maze.h"

Implementation is functionality and access its private data members.
Automaton variable belonging to main(), and use function members to
The choice main/switch illustrated most was: Make the key class instance be an
This is a software design choice.

So, what lifetime should I choose for the Maze?
Now, let's resume the design of class *Maze*.

**Tempa.** Last, print *Tempa*. Traverse the path once. Store the correct letter in the correct element of the *Tempa*. Fill it with the *ls* and *os*. Then, make the decision.

2. Use a temporary copy of a called *Tempa*. Search the path to decision for 0 values is whether to print 0 or a LETTER. Search the path to decision for 0 values is whether to print 0 or a LETTER. For each square, decide what to print. This

1. Loop through the subarray. For each square, decide what to print. This

2. Strategies: So, **both** the order N subarray of A and path from node must be traversed.

 major order (each row printed left to right, N different rows printed).

Output to the stdout stream **requires** the outputs must be done in row.

Assignment says display the maze and the path inside the maze. Let's think:

**Good idea:** Get a correct *printPath()* implementation early in the project.
as specified by the project assignment. //
the number of such paths is printed afterwards, //
if this linked list only contains 1 square position, //
(1.e., (p<->data.x,p<->data.y) to (fx, fy) (data members of maze), //
// square in the first node of the list located by p
post: printout done of all legal simple paths from
void printout() { post: printout of the above data.
// been read from stdin as specified by the assignment.
// postcondition: the N, A, sx, sy, tx, fy contents have
maze();
public:
```java
{ return 0;

Maze::printSolutions and their count.
Maze::printSolutions ( start )
{ / *test print of various paths*/ } (debug)
start = NULL;
start->next = Maze::next ( )
Maze->data->y = Maze::next ( )
Maze->data->x = Maze::next ( )
start = new PathListNode;
PathListNode *start;
Maze::print ( )
Maze::print ( )
// reads in and constructs the maze.
}

int main ( )

bool debug = true; // set to false for production

#include "Maze.h"
#include "PathListNode.h"
```
See below...

used by the key function's four-way loop.

Answer: The execution stack which contains the automatic variables named k,

Try an RIGHT, then DOWN, then LEFT, all four?

How does the computer figure out that after trying an UP step, it should then

Where is the stack of trial decisions?

algorithm.

The search algorithm follows the backtracking strategy of the n queens solution.
a square outside the $\$n$ by $\$n$ maze.

Each such continuation path does NOT use any square in the far and continuing to the goal square $(fx, fy)$ are printed.

Drift algorithm pseudo-code for

DRAFT
elif clause done

for loop finish

printInstructions (next) ;
next->data = this square;

} } } 

AND NOT in the path so far
AND the maze, AND is empty,

according to Rule[4]

If( the square one step to past->data

}

for (k=0; k<9; k++)

next->link = past;

PathListNode *next = new PathListNode;

int k; // automatic variable

else

{ print the path found so far; return

} // PathListNode *past

} 


Find and fix it yourself.

Whoops... There's a memory leak!
Let's get back to main ideas now.

return number of solutions printed.

3. Make output = int in PrintSolutions() and make it

2. Data member of Maze (accessible by member function).

1. Static lifetime variable.

Data design strategies:

Whoops... What about counting the paths?
can access the (private) the 2-dim array and the size variable.

empty square. These must be member functions of class Maze so they

an index pair is within the 0..n-1 range it corresponds to an

Test if a given index pair (denoting a square) is in a Linked List. (B) Test if
design helper methods to help implement its operations. For example, (A)
design the (PrintSolutions) method to

A. Analyze the pseudo-code in the outline of the PrintSolutions() method to

so far.

path to node to implement the Linked List of squares that is the path found

3. The Maze class uses some kind of Linked List node struct or class type

PrintSolutions()

may be accessed by any methods that need them. Such methods include

2. The 2-dim A array and size variable are private data members so they

reading it, printing it, printing paths, and printing all the solution paths.

1. Design a class named Maze to model one maze, and have methods for

Design Ideas
4. Write a skeleton implementation of the Maze.cpp which must include Maze.h.

You have in your mind.

and postconditions are consistent with each other and with the intentions

3. Check that the function declarations, data member declarations and pre- and postconditions to do.

2. Start writing the file Maze.h, and try to write pre- and postconditions to

1. Invent (or use our given) names for the data and function members.

Complete the Design of the Maze class.

deteects.
those that read in the maze data and print it out.

3. Finish coding, testing and debugging the skeleton functions, beginning with

2. You might want to write test drivers for some of the helper functions.

I. Code the main module: edit, compile and try to link it until all compilation

Implementation and Test