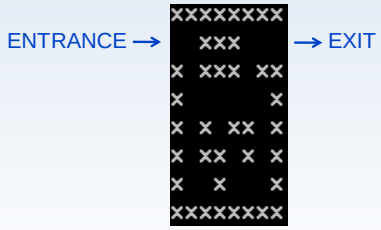
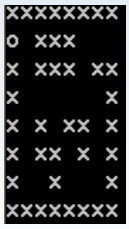
Data Structures

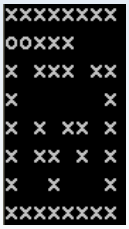
Stack Applications

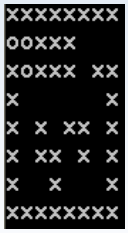
Finding a Path in a Labyrinth

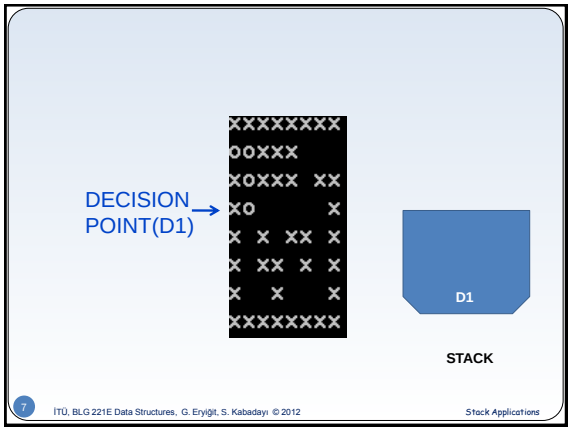
- The problem of finding a path in a labyrinth can be solved using the stack data structure:
  - While traversing the labyrinth, the states at decision points where going in more than one direction is possible get pushed onto a stack.
- We select one of the possible directions and proceed in that direction.
- If the choice made is not a correct choice, and we cannot find the exit of the labyrinth in this way, we go back to the last decision point (by popping the last state from the stack) and continue to search for the exit in the other untraversed directions.
- In the example labyrinth below, the first four steps are shown.
- In this representation, x's represent walls, the empty spaces represent paths, and o's represent traversed positions.

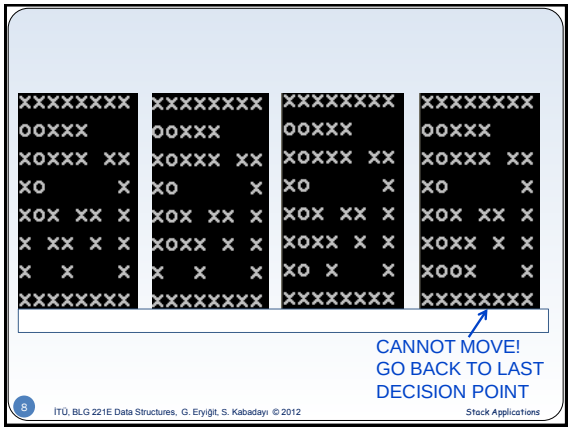


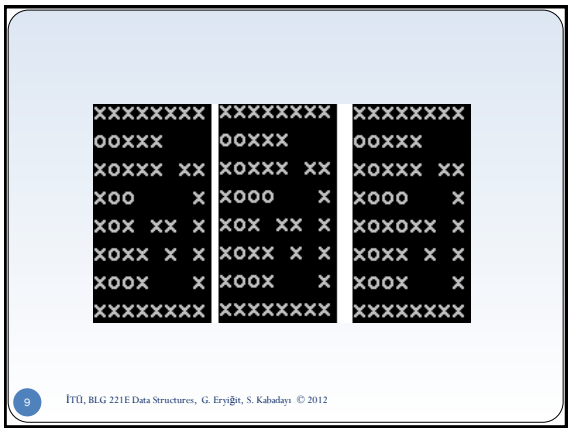













```
typedef struct d{
  int x;
  int y;
  int right;
  int left;
  int down;
  int up;
  int camefrom;
}StackDataType, position;

 struct Node{
  StackDataType data;
  Node *next;
};
```

0/1: cannot go/can go

Labyrinth.cpp

```
#define RIGHT   1
#define LEFT    2
#define UP      3
#define DOWN    4

char lab[8][8]={{'x','x','x','x','x','x','x','x'},
                {' ',' ','x','x','x',' ',' ',' '},
                {'x',' ','x','x','x',' ','x','x'},
                {'x',' ',' ',' ',' ',' ',' ','x'},
                {'x',' ','x',' ','x','x',' ','x'},
                {'x',' ','x','x',' ','x',' ','x'},
                {'x',' ',' ','x',' ',' ',' ','x'},
                {'x','x','x','x','x','x','x','x'}};
```



```
void printlab(char l[8][8]) {
     for (int i = 0; i < 8; i++) {
          for (int j = 0; j < 8; j++)
               cout << l[i][j];
          cout << endl;
     }
     cout << endl << endl;
}
```

```
int main(){
     Stack s;
     s.create();
     position entrance = {0,1,0,0,0,0,0};
     position exit = {7,1,0,0,0,0,0};
     position p = entrance;
     p.camefrom = LEFT;
     printlab(lab);
     bool goback = false;
```



```
while (p.x != exit.x || p.y != exit.y) {
  lab[p.y][p.x]='o';
  printlab(lab);
  //first find in how many directions we can move
  if (!goback) { //if not calculated before
        p.right = 0; p.left = 0; p.down = 0; p.up = 0;
        if (p.x<7 && lab[p.y][p.x+1]!='x') p.right=1;//right
        if (p.x>0 && lab[p.y][p.x-1]!='x') p.left=1;//left
        if (p.y<7 && lab[p.y+1][p.x]!='x') p.down=1;//down
        if (p.y>0 && lab[p.y-1][p.x]!='x') p.up=1;//up
  }
  else goback = false;
```



```
//here, one of the possible moves is selected
bool moved = true;
position past = p;
if (p.down && p.camefrom != DOWN)
  {p.y++; p.camefrom = UP; past.down = 0;}
else if (p.up && p.camefrom != UP)
  {p.y--; p.camefrom = DOWN; past.up = 0;}
else if (p.left && p.camefrom != LEFT)
  {p.x--; p.camefrom = RIGHT; past.left = 0;}
else if (p.right && p.camefrom != RIGHT)
  {p.x++; p.camefrom = LEFT; past.right = 0;}
else moved = false;//one direction (the
  minimum) is open, but this is the direction
  we came from
```

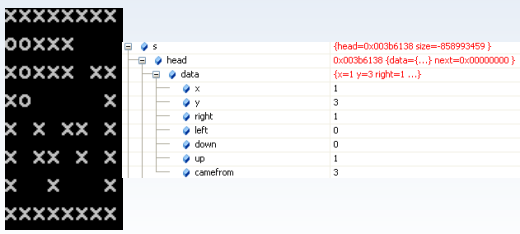


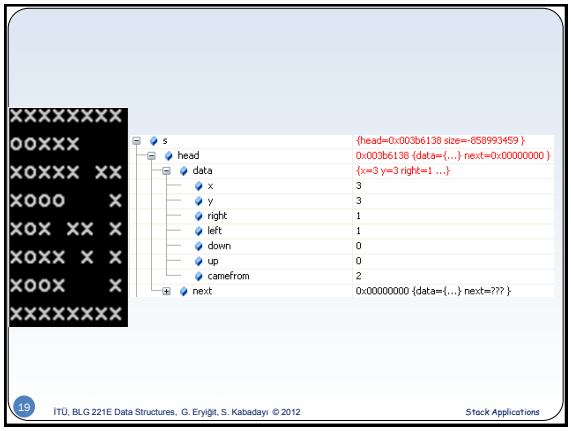
```
if (p.x != exit.x || p.y != exit.y) {
  if ( (p.down + p.up + p.right + p.left) > 2) {
  //there is more than one choice, push onto stack and
  //continue in that chosen direction. Let the choices
  //you have not selected remain marked on the stack.

      s.push(past);
  }
  if (!moved) { // has to go back
      if ( !s.isempty() ) {
            p = s.pop();
            goback = true;
      }

  }
}
```

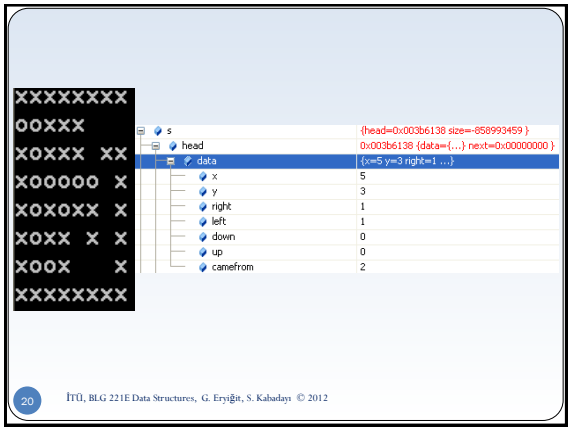


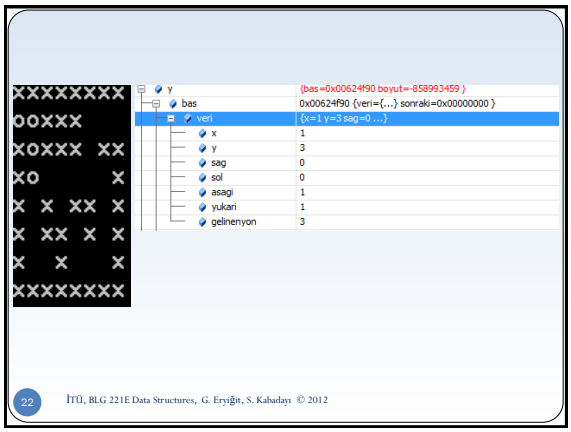
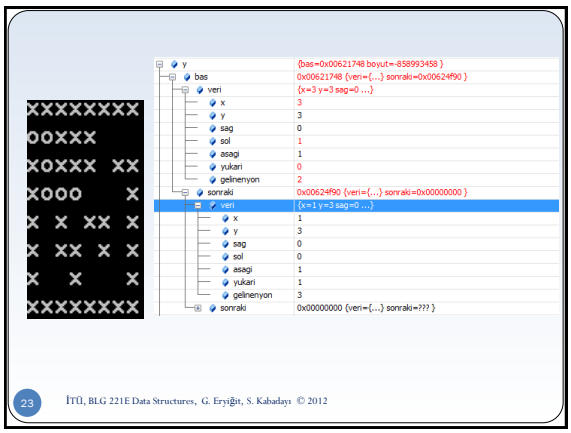
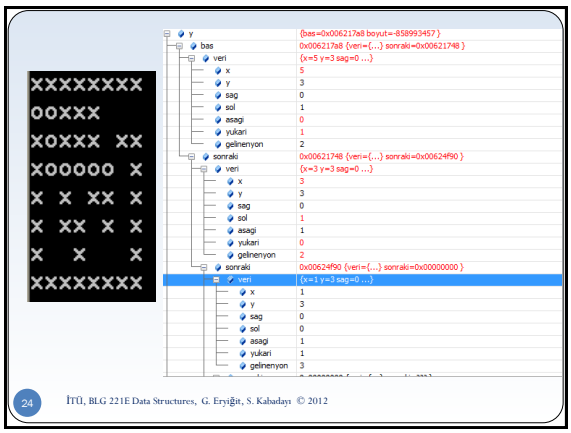
```
  }//end of while
  lab[p.y][p.x] = 'o';
  printlab(lab);
  cout << "PATH found" << endl;
  s.close();

  return EXIT_SUCCESS;
}
```


- Assuming that the code works step-by-step, the state of the stack after every stack operation (push and pull) is shown below.







- In the part of the code where a direction is selected, as a result of preference being given to going right, the state of the stack after every stack operation (push and pull):

```
if(p.right && p.camefrom != RIGHT)
   {p.x++;p.camefrom=LEFT;past.right=0;}
else if(p.down && p.camefrom != DOWN)
   {p.y++;p.camefrom=UP;past.down=0;}
else if(p.up && p.camefrom != UP)
   {p.y--;p.camefrom=DOWN;past.up=0;}
else if (p.left && p.camefrom != LEFT)
   {p.x--;p.camefrom=RIGHT;past.left=0;}
else moved = false;
```