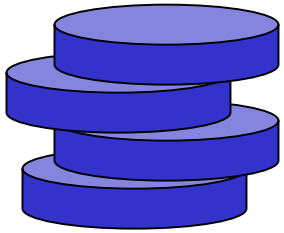
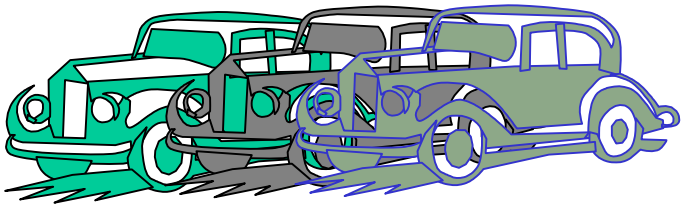




# Lecture5: 스택(5.1절), 큐(5.2절), 데크(5.3절)

김 강 희
khkim@ssu.ac.kr

# 요약

- 스택 (stack)
  - 임의의 객체들을 삽입한 순서대로 저장하는 자료 구조
  - 삭제는 Last In, First Out 방식으로 수행됨
  - 최근성(recency)가 중요한 상황에서 많이 사용됨
    - 웹 페이지 방문 기록, 에디터의 undo, …
- 큐 (queue)
  - 임의의 객체들을 삽입한 순서대로 저장하는 자료 구조
  - 삭제는 First In, First Out 방식으로 수행됨
  - 시간 순서(time order)가 중요한 상황에서 많이 사용됨
    - 대기자 명단, 생산자-소비자 관계 (프린터), …
- 데크 (deque: double-ended queue)
  - 임의의 객체들을 큐 형태로 저장하되, 큐의 앞과 뒤에서 각기 삽입과 삭제가 가능한 자료 구조
  - 데크를 스택 또는 큐처럼 사용할 수 있음

# 5.1절 스택

```
template <typename E>
class Stack  {
public:
   int size() const;
   bool empty() const;
   const E& top() const throw(StackEmpty);
   void push(const E& e);
   void pop() throw(StackEmpty);
}
```

# 활용예: 산술식 계산기

Slide by Matt Stallmann included with permission.

14 – 3 * 2 + 7 = (14 – (3 * 2) ) + 7

Operator precedence

* has precedence over +/–

Associativity

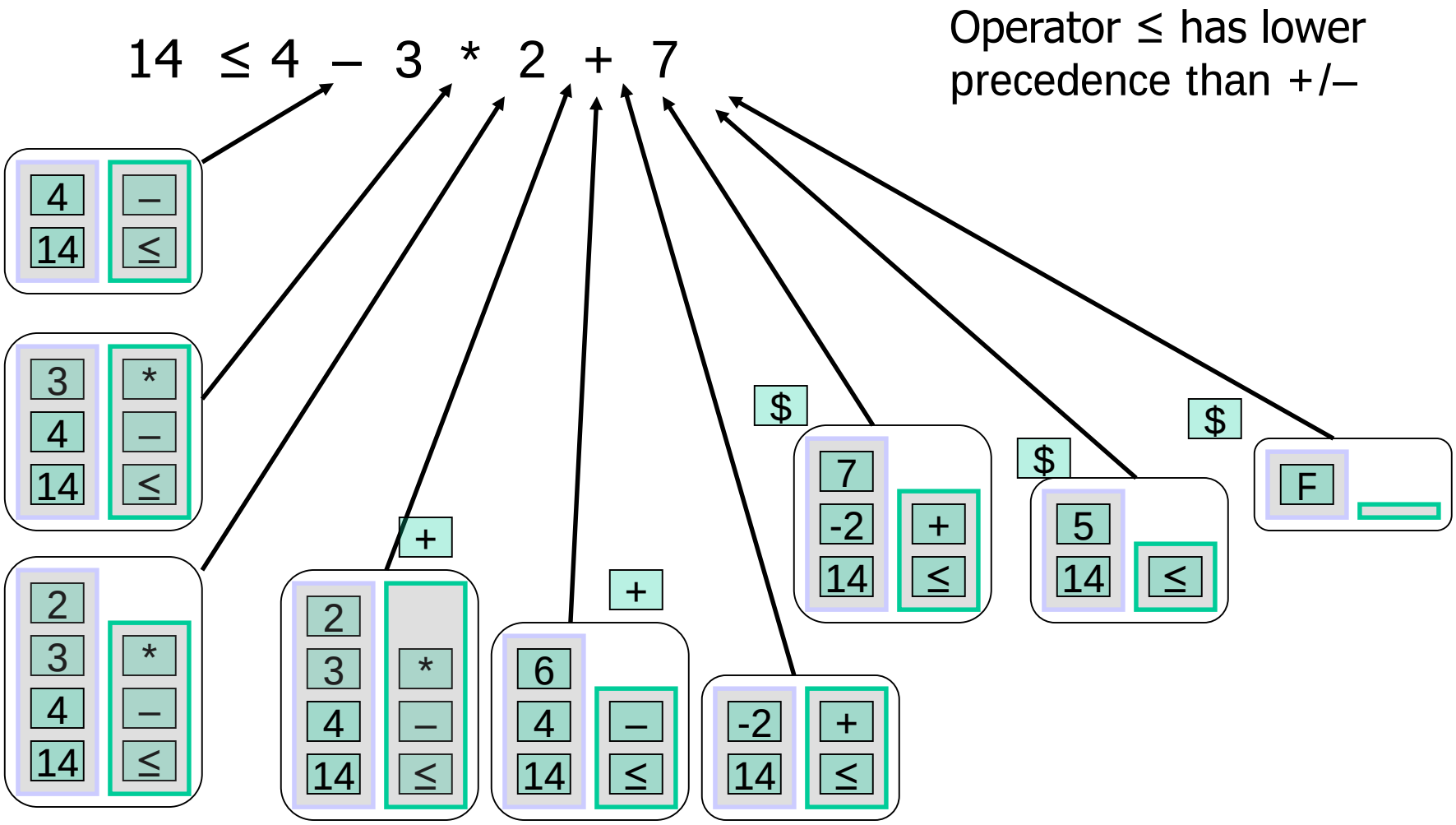
operators of the same precedence group evaluated from left to right

Example: (x – y) + z rather than x – (y + z)

Idea: push each operator on the stack, but first pop and perform higher and *equal* precedence operations.

# 활용예: 산술식 계산기

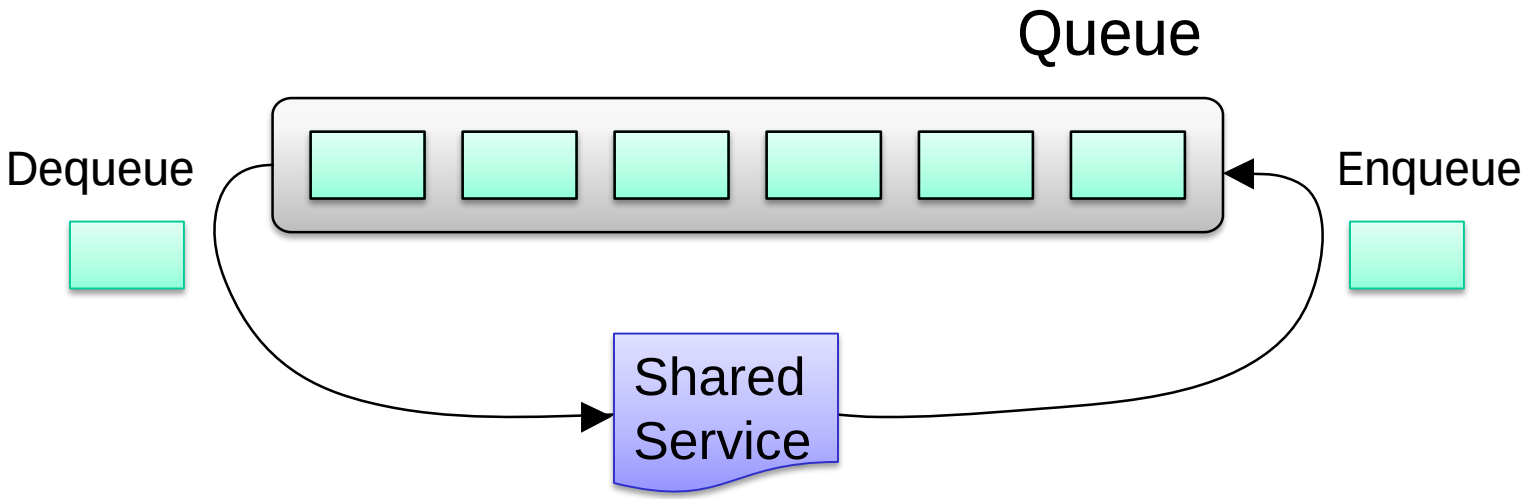
Slide by Matt Stallmann included with permission.



# 5.2절 큐

```
template <typename E>
class Queue {
public:
   int size() const;
   bool empty() const;
   const E& front() const throw(QueueEmpty);
   void enqueue(const E& e);
   void dequeue() throw(QueueEmpty);
};
```

# 활용예: Round Robin Scheduler



# 5.3절 데크

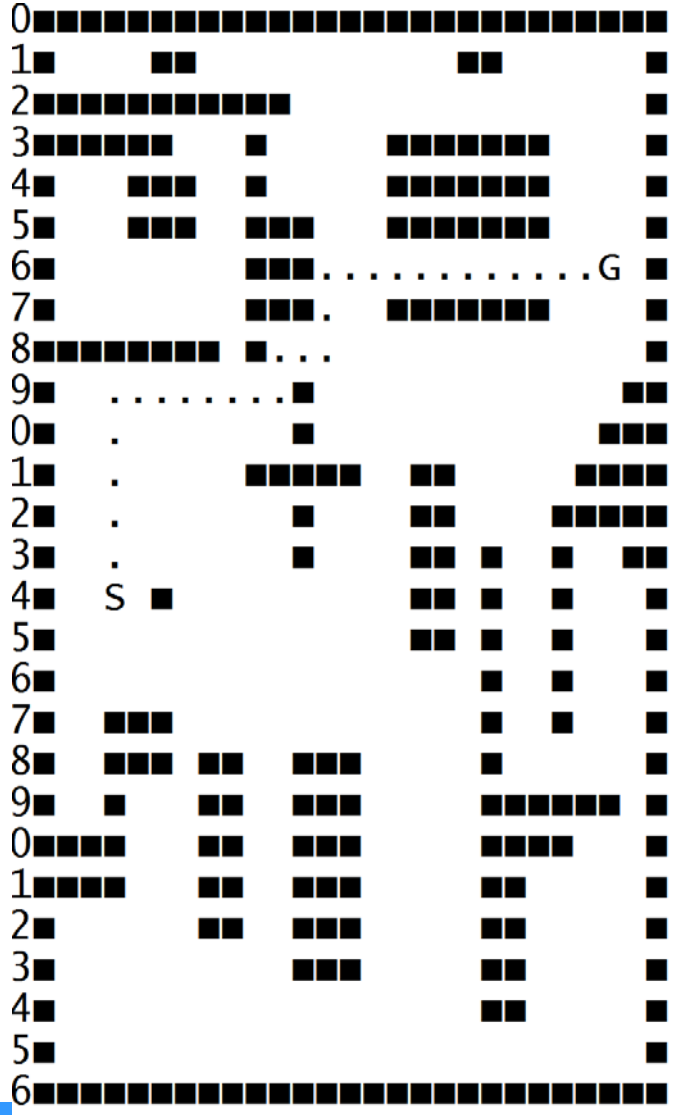
```
template <typename E>
class Deque {
public:
   int size() const;
   bool empty() const;
   const E& front() const  throw(DequeEmpty);
   const E& back() const  throw(DequeEmpty);
   void insertFront(const E& e);
   void insertBack(const E& e);
   void removeFront() throw(DequeEmpty);
   void removeBack()  throw(DequeEmpty);
};
```

# Deque 활용

- Stack → Deque
  - top() → front()
  - push() → insertFront()
  - pop() → removeFront()
- Queue → Deque
  - front() → front()
  - enqueue() → insertBack()
  - dequeue() → removeFront()

# 활용예: Maze

$ ./Maze.exe q map1.txt

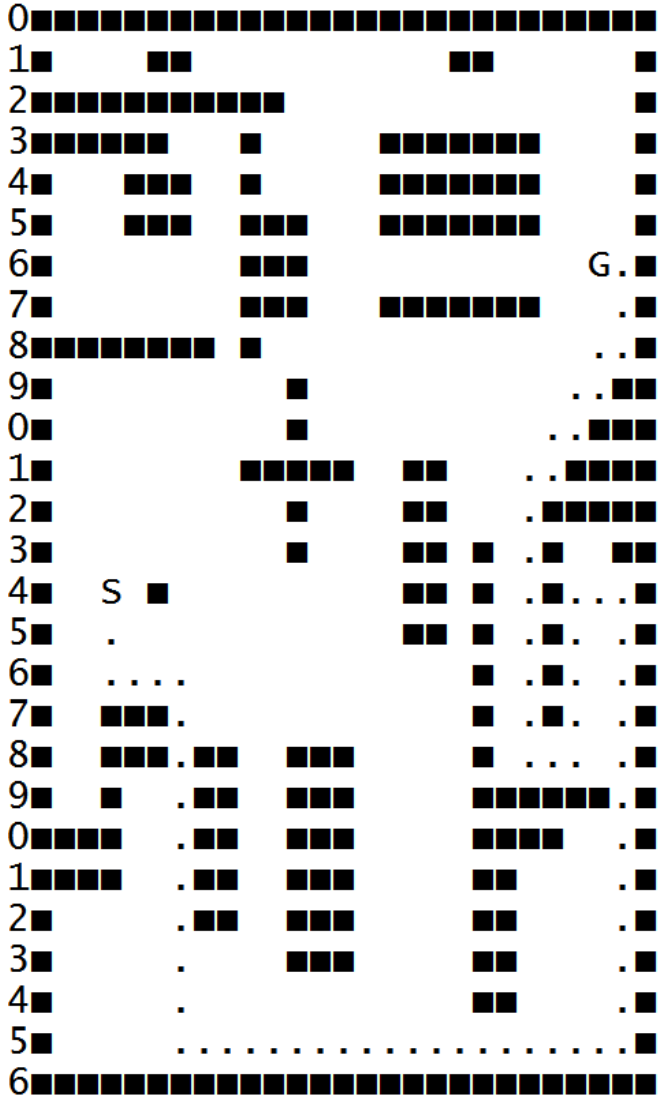


```
Complete path:
p(14,3) -> p(13,3) -> p(12,3) -> p(11,3) -> p(
10,3) -> p(9,3) -> p(9,4) -> p(9,5) -> p(9,6)
-> p(9,7) -> p(9,8) -> p(9,9) -> p(9,10) -> p(
8,10) -> p(8,11) -> p(8,12) -> p(7,12) -> p(6,
12) -> p(6,13) -> p(6,14) -> p(6,15) -> p(6,16
) -> p(6,17) -> p(6,18) -> p(6,19) -> p(6,20)
-> p(6,21) -> p(6,22) -> p(6,23) -> p(6,24)

Path Length (in points) : 30
Total Moves (in points) : 375
Maximum Reached Length of Open Queue: 22
Final Length of Open Queue (O): 9
Final Length of Closed Queue (C): 376
Total Number of Explored States (O+C): 385
```

# 활용예: Maze

$ ./Maze.exe s map1.txt



```
Complete path:
p(14,3) -> p(15,3) -> p(16,3) -> p(16,4) -> p(16,5) -> p(16,6) -> p(17,6) -> p(1
8,6) -> p(19,6) -> p(20,6) -> p(21,6) -> p(22,6) -> p(23,6) -> p(24,6) -> p(25,6
) -> p(25,7) -> p(25,8) -> p(25,9) -> p(25,10) -> p(25,11) -> p(25,12) -> p(25,1
3) -> p(25,14) -> p(25,15) -> p(25,16) -> p(25,17) -> p(25,18) -> p(25,19) -> p(
25,20) -> p(25,21) -> p(25,22) -> p(25,23) -> p(25,24) -> p(25,25) -> p(24,25) -
> p(23,25) -> p(22,25) -> p(21,25) -> p(20,25) -> p(19,25) -> p(18,25) -> p(17,2
5) -> p(16,25) -> p(15,25) -> p(14,25) -> p(14,24) -> p(14,23) -> p(15,23) -> p(
16,23) -> p(17,23) -> p(18,23) -> p(18,22) -> p(18,21) -> p(17,21) -> p(16,21) -
> p(15,21) -> p(14,21) -> p(13,21) -> p(12,21) -> p(11,21) -> p(11,22) -> p(10,2
2) -> p(10,23) -> p(9,23) -> p(9,24) -> p(8,24) -> p(8,25) -> p(7,25) -> p(6,25)
 -> p(6,24)

Path Length (in points) : 70
Total Moves (in points) : 364
Maximum Reached Length of Open Queue: 115
Final Length of Open Queue (O): 54
Final Length of Closed Queue (C): 365
Total Number of Explored States (O+C): 419
```

# Maze.cpp

```
// Taken from https://github.com/steffanc/MazeBot
// Edited by khkim for readability
#include <cstdlib>
#include <iostream>
#include <math.h>
#include <string>
#include <fstream>
//#define __STL_DEQUE__
#ifdef __STL_DEQUE__
#include <deque> // Standard Template Library
#else
#include "LinkedDeque.h" // "Data Structures and Algorithms in C++ (2nd Edition)"
#define deque LinkedDeque
#define push_back insertBack
#define pop_back removeBack
#define push_front insertFront
#define pop_front removeFront
#endif
```

# Maze.cpp

```
class Point {
private:
  int depth;
  Point *parent;
public:
  int cy, cx;
  Point(): cy(0), cx(0), depth(0), parent(0) { }
  Point(int cy, int cx, int depth, Point * parent) { … }
  ~Point() { delete parent; }
  int getDepth() const { return depth; };
  Point *getParent() const { return parent; };
  friend ostream& operator<<(ostream& out, const Point& obj);
};
```

# Maze.cpp

```
int main(int argc, char *argv[]) {
  Point start, goal;
  InitMaps(argv[1], start, goal);
  bool found = FindRoute(start, goal);
  if (!found) {
    cout << "This maze has no solution!" << endl;
    return 1;
  }

  PrintRoute();
  //ClearMaps();
  FreeMaps();
  return 0;
}
```

# map1.txt

```
27 27
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1
1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1
1 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 9 0 1
1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1
1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 1 1
1 0 0 8 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1
1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1
1 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1
1 0 0 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 1
1 1 1 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 1
1 1 1 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1
1 0 0 0 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 27x27 행렬
- '1'→ 벽
- '0'→ 빈공간
- '8'→ 시작
- '9'→ 목표

# Maze.cpp

```
void InitMaps(char *fname, Point& start, Point& goal) {
  ifstream mapFile;
  mapFile.open(fname, ios::in);
  if (!mapFile.is_open()) {    cout << "Unable to open file";    exit(1);  }
  mapFile >> mapHeight;          //  '>>' operator는 콘솔 입력을 변수에 저장함
  mapFile >> mapWidth;
  mazeMap = new int*[mapHeight];
  mazeRoute = new int*[mapHeight];
  for (int y = 0; y < mapHeight; y++) {
    mazeMap[y] = new int[mapWidth];
    mazeRoute[y] = new int[mapWidth];
  }
  for (int y = 0; y < mapHeight; y++) {
    for (int x = 0; x < mapWidth; x++) {
      mapFile >> mazeMap[y][x];
      mazeRoute[y][x] = mazeMap[y][x];
      if (mazeMap[y][x] == START) {
          start.cy = y;
          start.cx = x;
      } else if (mazeMap[y][x] == GOAL) {
          goal.cy = y;
          goal.cx = x;
      }
    }
  }
  mapFile.close();
  return;
}
```

# Maze.cpp

```
bool FindRoute(const Point& start, const Point& goal) {
  int cy = 0, cx = 0;
  bool found = false;
  maxOpenQSize = 1;
  mazeRoute[start.cy][start.cx] = OPEN;
  openDeque.push_back(new Point(start.cy, start.cx, 0, 0));
  while(openDeque.size() != 0) { // Keep searching until an goal is determined or no solution is found
    cy = openDeque.front()->cy;
    cx = openDeque.front()->cx;
    mazeRoute[cy][cx] = CLOSED; // Current position has now been opened and explored
    nMoves++;
    closedDeque.push_back(openDeque.front()); // move the 1st element from openDeque to closedDeque
    Point *p = openDeque.front();
    openDeque.pop_front();

    if(mazeMap[cy][cx] == GOAL) { found = true;   break;     }
    else { StepNext(cy, cx); } // check surrounding positions

    cout << endl; PrintRoute(); cout << endl;
    getchar();
  }
  return found;
}
```

```
void StepNext(int cy, int cx) {
  if (data_struct == 'q') StepNext_w_Queue(cy, cx);
  else if (data_struct == 's') StepNext_w_Stack(cy, cx);
}
```

# Maze.cpp

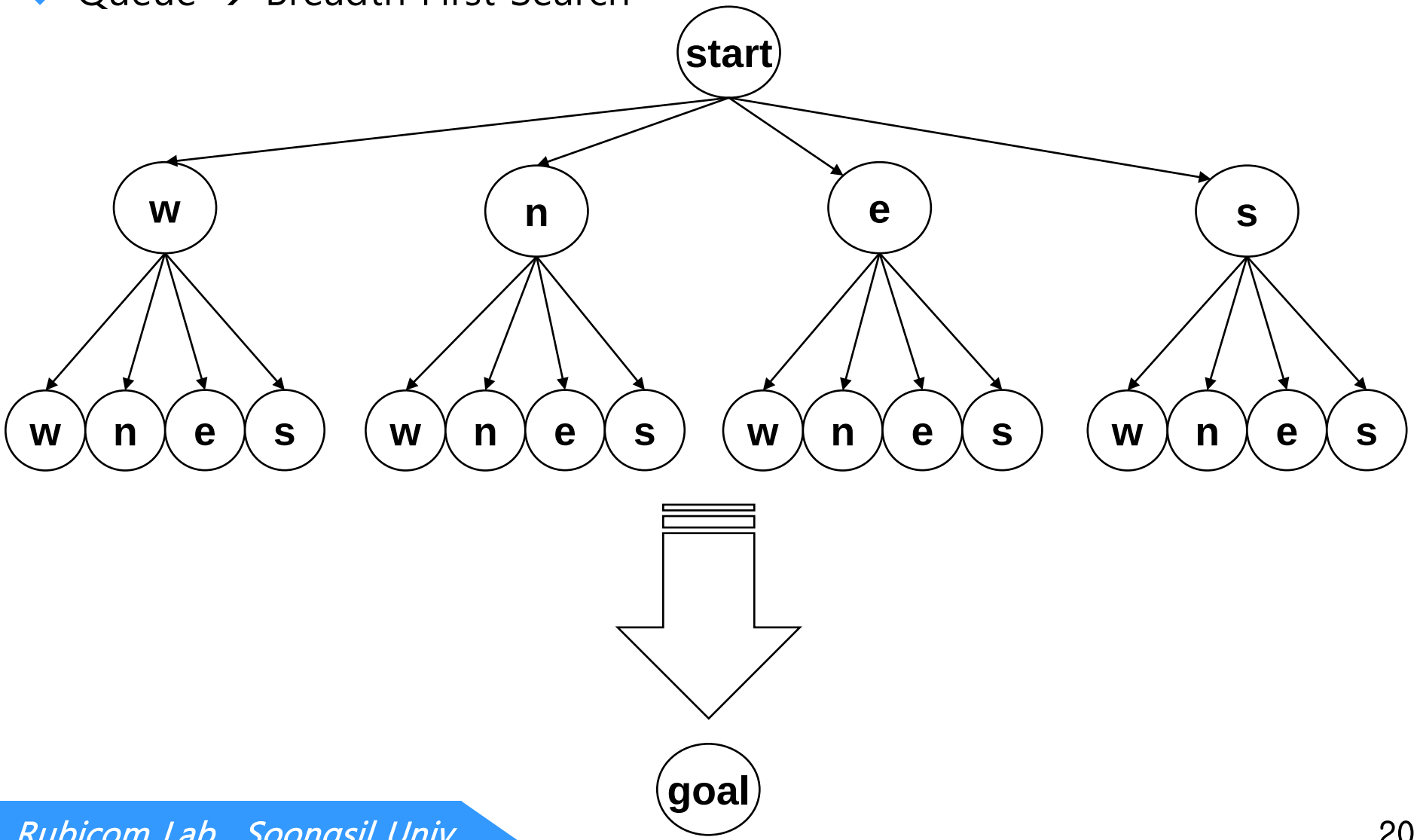
```
void StepNext_w_Queue(int cy, int cx) {
  int depth = (closedDeque.back()->depth)+1;
  Point *parent = closedDeque.back();
  if (never_visited(mazeRoute[cy][cx-1])) { // west
    mazeRoute[cy][cx-1] = OPEN;
    openDeque.push_back(new Point(cy, cx-1, depth, parent));
    cout << "openDeque.push_back(" << cy << "," << cx-1 << "," << depth << ")" << endl;
  }
  if (never_visited(mazeRoute[cy-1][cx])) { // north
    mazeRoute[cy-1][cx] = OPEN;
    openDeque.push_back(new Point(cy-1, cx, depth, parent));
    cout << "openDeque.push_back(" << cy-1 << "," << cx << "," << depth << ")" << endl;
  }
  if (never_visited(mazeRoute[cy][cx+1])) { // east
    mazeRoute[cy][cx+1] = OPEN;
    openDeque.push_back(new Point(cy, cx+1, depth, parent));
    cout << "openDeque.push_back(" << cy << "," << cx+1 << "," << depth << ")" << endl;
  }
  if (never_visited(mazeRoute[cy+1][cx])) { // south
    mazeRoute[cy+1][cx] = OPEN;
    openDeque.push_back(new Point(cy+1, cx, depth, parent));
    cout << "openDeque.push_back(" << cy+1 << "," << cx << "," << depth << ")" << endl;
  }
  maxOpenQSize = (openDeque.size() > maxOpenQSize) ? openDeque.size() : maxOpenQSize;
}
```

# Maze.cpp

```
void StepNext_w_Stack(int cy, int cx) {
  int depth = (closedDeque.back()->depth)+1;
  Point *parent = closedDeque.back();
  if (never_visited(mazeRoute[cy][cx-1])) {  // west
    mazeRoute[cy][cx-1] = OPEN;
    openDeque.push_front(new Point(cy, cx-1, depth, parent));
    cout << "openDeque.push_front(" << cy << "," << cx-1 << "," << depth << ")" << endl;
  }
  if (never_visited(mazeRoute[cy-1][cx])) {  // north
    mazeRoute[cy-1][cx] = OPEN;
    openDeque.push_front(new Point(cy-1, cx, depth, parent));
    cout << "openDeque.push_front(" << cy-1 << "," << cx << "," << depth << ")" << endl;
  }
  if (never_visited(mazeRoute[cy][cx+1])) {  // east
    mazeRoute[cy][cx+1] = OPEN;
    openDeque.push_front(new Point(cy, cx+1, depth, parent));
    cout << "openDeque.push_front(" << cy << "," << cx+1 << "," << depth << ")" << endl;
  }
  if (never_visited(mazeRoute[cy+1][cx])) {  // south
    mazeRoute[cy+1][cx] = OPEN;
    openDeque.push_front(new Point(cy+1, cx, depth, parent));
    cout << "openDeque.push_front(" << cy+1 << "," << cx << "," << depth << ")" << endl;
  }
  maxOpenQSize = (openDeque.size() > maxOpenQSize) ? openDeque.size() : maxOpenQSize;
}
```

# Stack vs. Queue

- Stack → Depth First Search
- Queue → Breadth First Search



# 감사합니다!