

AI를 활용한 수업자료 자동 생성 프레임워크

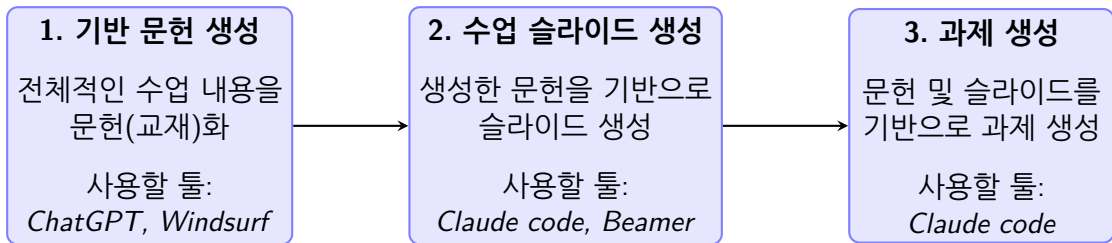
강의 슬라이드 및 과제 생성 자동화

전민석

DGIST

2025년 11월 4일

수행방법: 3단계 프로세스



1단계: 기반 문헌 생성

1단계: 기반 문헌 생성

목표

수업에서 배울 내용들을 정리한 문헌을 생성

사용 도구

(1) ChatGPT

- 대화형 인공지능
- 글쓰기, 아이디어 발상

(2) Windsurf

- AI 기반 개발 환경
- 코드 작성, 실행, 디버깅
- 마크다운(.md) 형태로 문헌 생성

핵심 프로세스

1. ChatGPT로 로드맵 생성
2. Windsurf에서 구체적인 내용 작성
3. 단원별 학습 포인트 정리
4. 상세 내용 생성

1.1 ChatGPT를 사용한 로드맵 생성

예시: 자료구조 수업

ChatGPT에 프롬프트를 입력하여 수업의 전체 로드맵을 마크다운(.md) 형식으로 생성

사용한 프롬프트

"I want to learn what data structures are and how we can use data structures. Would you please create a Learning Roadmap for me? Please describe the Roadmap using markdown format."

1.1 ChatGPT를 사용한 로드맵 생성

- 결과물 (Roadmap.md): 전체 수업 구조, 단원별 주제, 학습 목표, 학습 순서

```
# 📖 Data Structures Learning Roadmap
```

```
## 1. **Foundations**
```

```
Before diving into data structures, build a strong foundation.
```

```
- **Prerequisites**
```

- Learn a programming language (C, C++, Java, or Python recommended).
- Understand basic programming concepts:
 - Variables, loops, conditionals
 - Functions & recursion
 - Arrays and strings
 - Memory concepts (stack vs heap)

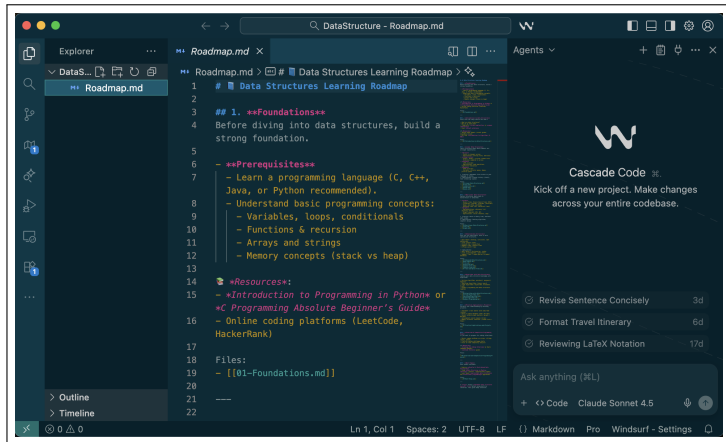
```
📚 *Resources*:
```

- **Introduction to Programming in Python** or **C Programming Absolute Beginner's Guide**
- Online coding platforms (LeetCode, HackerRank)

1.2 Windsurf에서 구체적인 내용 생성

Step 1: 로드맵 불러오기

ChatGPT가 생성한 Roadmap.md를 Windsurf에서 불러옴



1.2 Windsurf에서 구체적인 내용 생성

Step 2: 단위별 파일 생성

AI 채팅에 프롬프트를 입력하여 단위별로 배워야 할 포인트들이 작성된 파일들을 생성

사용한 프롬프트 1

"This is a learning roadmap report. I need you to do the following:

Step 1. Based on the learning roadmap, create a list of markdown files for each topic.

Step 2. For each topic, write an introduction and list of knowledge points. Each knowledge point title should be written in [[]]"

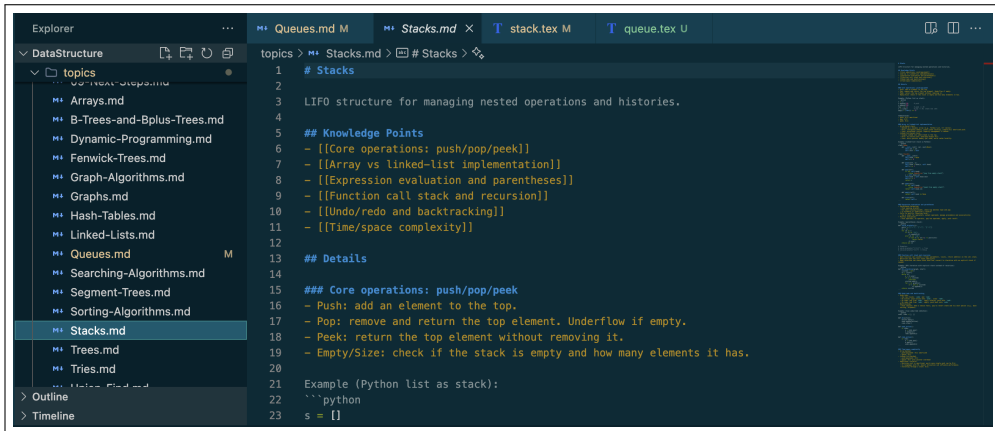
사용한 프롬프트 2

"I need you to explain the knowledge points in detail"

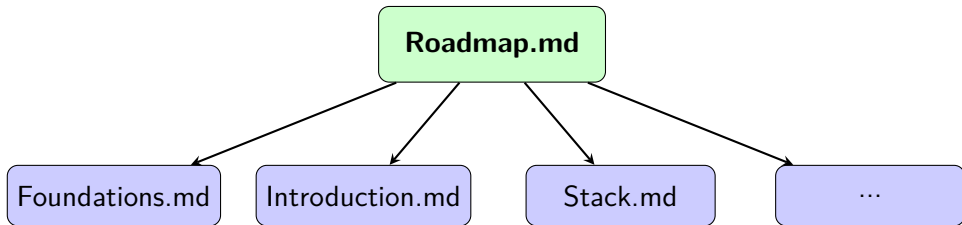
1.2 Windsurf에서 구체적인 내용 생성

최종 결과

단원별로 체계적으로 정리된 기반 문헌 파일들이 완성됨



기반 문헌 생성 결과



디렉토리 구조

```
DataSeture/  
|- Roadmap.md  
|- topics/  
    |- Foundations.md  
    |- ...
```

2단계: 슬라이드 생성

2단계: 슬라이드 생성

목표

생성한 문헌을 기반으로 수업 슬라이드 생성하기

사용 도구

Beamer

- LaTeX 기반 문서 클래스
- 학술 및 전문 발표용 슬라이드 제작
- **코드로 슬라이드를 생성**

Claude Code

- Claude AI 기반 코딩 보조 도구
- 코드 작성, 수정, 설명, 디버깅 지원

작업 순서

1. slides/ 폴더 생성
2. Beamer 템플릿 선택
3. Claude Code 실행
4. /init 명령어로 코드베이스 이해
5. 슬라이드 생성 프롬프트 입력

2.1 슬라이드 생성을 위한 세팅

디렉토리 구조 설정

```
DataSetructure/  
  |- Roadmap.md  
  |- topics/  
    |- Foundations.md  
    |- ...  
  |- slides/  
    |- template/
```

Claude Code 초기화

1. Root 폴더(DataStructure/)에서 Claude Code 실행
2. `/init` 명령어 실행

2.2 Claude Code로 슬라이드 생성

프롬프트

Currently, I am teaching Data Structure course based on the markdown files in the 'topics/' folder.

In the lecture, I will use slides that are constructed based on the markdown files in the 'topic/' folder.

I will use Beamer to generate slides, and the template I will use is placed in the 'slides/BeamerTemplate/' folder.

I need you to generate a folder 'slides/Stack/' and write a Beamer LaTeX file stack.tex based on the contents in 'topics/Stack.md' and referring to the template in 'slides/BeamerTemplate'.

2.2 Claude Code로 슬라이드 생성

결과

AI에 의해 자동으로 생성된 전문적인 강의 슬라이드

Data Structures: Stacks

Data Structures Course
Week 1
October 3, 2023

Contents

1. Introduction to Stacks
2. Core Operations
3. Implementation Approaches
4. Applications
5. Complexity Analysis
6. Summary

Introduction to Stacks

What is a Stack?

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle.

Key Characteristics:

- Elements are added and removed from the same end (top).
- Only the top element is accessible.
- Perfect for managing nested operations and function calls.
- Natural structure for function calls and recursion.

Stack Structure Diagram

Core Operations

Essential Stack Operations

Stack Operations:

- Push:** Add element to the top.
- Pop:** Remove and return top element.
- Peek/Top:** Return top element without removing.
- Empty:** Check if stack is empty.
- Size:** Get number of elements.

Time Complexity:

All operations are $O(1)$ - constant time (amortized for push in dynamic array).

Stack Operations Visualization

Initial Stack: [10, 20, 30] After Push(40): [10, 20, 30, 40] After Pop(): [10, 20, 30]

Return: 40 Return: 30 Return: 20 Return: 10

Implementation Approaches

Array-based vs Linked List Implementation

Array-based Stack:

- Pros:
 - Contiguous memory.
 - Great cache locality.
 - Simple implementation.
 - $O(1)$ amortized push.
- Cons:
 - Wasted space (if not full).
 - Capacity management.

Linked List-based Stack:

- Pros:
 - No waste (dynamic growth).
 - Always $O(1)$ operations.
 - Dynamic size.
- Cons:
 - Extra pointer memory.
 - More complex.

Linked List Stack Implementation

```
class Node {
    int data;
    Node* next;
};

class Stack {
    Node* top;
};

void push(Stack* s, int x) {
    Node* new_node = new Node(x);
    new_node->next = top;
    top = new_node;
}
```

Applications

Expression Evaluation: Parentheses Matching

Check if parentheses, brackets, and braces are properly balanced.

Algorithm:

1. Push opening brackets onto stack.
2. For closing brackets:
 - Check if stack is empty.
 - Check if top matches type.
 - Pop if match, return false if not.
3. Stack should be empty at end.

Function Call Stack and Recursion

Call Stack Diagram

Recursion: Function calls create a stack frame. Each frame contains parameters, local variables, return address, and return value.

Call Stack Diagram

Undo/Redo Operations

Undo/Redo Stack Implementation:

- Undo Stack: Store previous actions.
- Redo Stack: Store actions undone.

Operations:

- Undo: pop from undo, clear redo.
- Redo: pop from redo, apply to redo.

Complexity: $O(1)$ per operation.

Complexity Analysis

Time and Space Complexity

Implementation	Push	Pop/Peek	Space
Array-based	$O(1)$	$O(1)$	$O(N)$
Linked List-based	$O(1)$	$O(1)$	$O(N)$ (plus pointer overhead)

Key Takeaways:

- Array-based: Amortized $O(1)$ push, due to resizing.
- Linked List-based: Guaranteed $O(1)$ for all operations.

Summary

Key Takeaways

Stack Fundamentals:

- LIFO data structure with top-only access.
- Essential operations: push, pop, peek, empty, size.
- All operations are $O(1)$ time complexity.

Implementation Approaches:

- Array-based: Simple, efficient, $O(1)$ amortized push.
- Linked List-based: Guaranteed $O(1)$, more memory.

Applications:

- Expression evaluation and parentheses matching.
- Function call management and recursion.
- Undo/redo functionality.

Thank You!

Questions?

3단계: 과제 생성

3단계: 과제 생성

목표

생성한 문헌을 기반으로 과제 생성하기

디렉토리 구조

```
DataStructure/  
  |- Roadmap.md  
  |- topics/  
  |- slides/  
  |- assignments/
```

생성 과정

1. assignments/ 디렉토리 생성
2. Claude Code에 프롬프트 입력
3. 문헌 및 슬라이드 기반 과제 생성

3단계: 과제 생성

프롬프트

Currently, I am teaching Data Structure course based on the markdown files in the 'topics/' folder.

In the lecture, I will give the students assignments about implementing each data structure.

I need you to generate 'assignments/stack' folder and implement 'stack.py' that implements the core functions of stack data structure, and also write an application code 'application.py' that imports and uses the functions in stack.py.

After you implement the core functions in stack.py, I will hide them and make the student implement them as their assignments.

3단계: 과제 생성

결과

자동으로 생성된 체계적인 과제

README.md	update	2 weeks ago
application.py	update	2 weeks ago
stack.py	update	2 weeks ago
test_stack.py	update	2 weeks ago

README.md

Stack Assignment

This assignment contains a complete implementation of stack data structures and their applications.

```
class ArrayStack:
    """
    Array-based stack implementation using Python list.

    Provides O(1) amortized time complexity for all operations.
    """

    def __init__(self):
        """Initialize an empty stack."""
        self._data = []

    def push(self, item):
        """
        Add an item to the top of the stack.

        Args:
            item: The item to be added to the stack

        Time Complexity: O(1) amortized
        """
        self._data.append(item)
```

AI 기반 수업자료 생성 시스템의 핵심

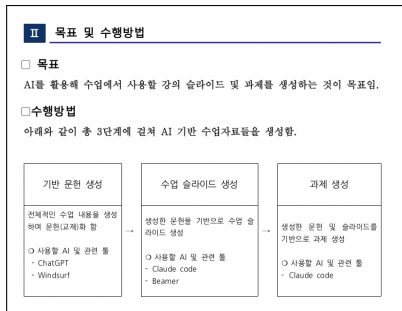
1. **기반 문헌 생성**: ChatGPT + Windsurf로 체계적인 교재 작성
2. **슬라이드 생성**: Claude Code + Beamer로 전문적인 프레젠테이션 제작
3. **과제 생성**: AI 기반 맞춤형 과제 자동 생성

핵심 가치

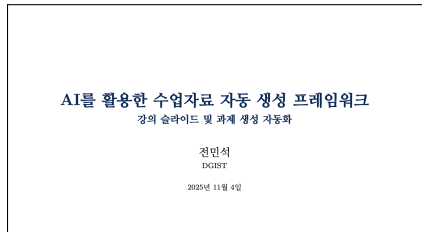
효율성 × 품질 × 확장성

확장성: 발표 슬라이드 생성

- 제출한 아이디어 결과서로부터 발표 자료 생성하는것 또한 가능



⇒
Claude code
+
Windsurf



오늘 발표에 사용한 슬라이드

제출한 아이디어 결과서

감사합니다!

생성한 코드: <https://github.com/MinseokJGit/DataStructure/>

생성한 과목 페이지: https://dgistpl.github.io/courses/ai_ds/2025/