주제/단락	내용
부동소수점 표현 정의	지수로 소수점 위치를 이동시켜 넓은 범위를 표현하는 수 표현 방법이다.
부동소수점 일반식	N = (−1)^S × M × B^E 형태로, S는 부호, M은 가수, B는 기수, E는 지수이다.
10진 부동소수점 예	274,000,000,000,000 = 2.74 × 10^14, 0.00000000000274 = 2.74 × 10^−12로 표현한다.
2진 부동소수점 기수	2진 부동소수점의 기수 B는 2이다.
2진 부동소수점 예	11.101 = 0.11101 × 2^2, 0.00001101 = 0.1101 × 2^−4로 표현한다.
단정도 정의	단일 정밀도는 32비트이며 C 언어의 float 형식에 해당한다.
배정도 정의	복수 정밀도는 64비트이며 C 언어의 double 형식에 해당한다.
필드 비트수 예	S: 1비트, E: 8비트, M: 23비트로 구성된다.
지수 비트 증가 효과	지수(E) 비트 수를 늘리면 표현 가능한 수의 범위가 확장된다.
가수 비트 증가 효과	가수(M) 비트 수를 늘리면 정밀도가 증가한다.
단정도 범위	0.5 × 2^−128 ~ 0.5 × 2^127 ≈ 1.47 × 10^−39 ~ 1.7 × 10^38 범위의 수를 표현한다.
고정소수점 비교	32비트 고정소수점의 경우 1.0 × 2^−31 ~ 1.0 × 2^31 ≈ 2.0 × 10^−9 ~ 2.0 × 10^9 범위를 갖는다.
동치 표현 다중성	같은 값이 여러 형태로 표현될 수 있다(예: 0.1101 × 2^5, 11.01 × 2^3, 0.001101 × 2^7).
정규화 개념	표현을 한 가지로 통일하기 위해 소수점 아래 첫 비트가 1이 되도록 지수를 조정한다 (±0.1bbb… × 2^E).
정규화 예	0.1101 × 2^5가 정규화된 표현이다.
비트 필드 예시	0.1101 × 2^5의 경우 S=0, E=00000101, M=1101 0000 0000 0000 0000 000이다.
히든 비트	소수점 왼쪽의 1은 항상 1이므로 저장하지 않는 hidden bit로 간주하며, 23비트 가수로 24자리까지 표현 가능하다.

주제/단락	내용
0 표현 문제	0을 표현하려면 가수는 반드시 0이어야 하고 0×2^E=0이므로 지수와 무관하지만, 간단한 0-검사를 위해 지수를 0으로 만드는 방식이 필요하다.
바이어스 지수	지수에 일정값을 더해 저장한다(예: excess-127은 127을, excess-128은 128을 더한다).
바이어스 지수 용도	모든 비트를 0으로 두어 0을 쉽게 검사하고, 지수 필드를 부호 없는 정수로 취급해 크기 비교와 정렬을 용이하게 한다.
8비트 바이어스 지수값	8비트 지수에 바이어스를 적용한 값들의 범위를 정의한다(개별 매핑은 바이어스 값에 따라 결정).
예: −13.625의 32비트 표현	13.625=1101.101=0.1101101 × 2^4, S=1, E=00000100+10000000=10000100(excess-128), M=10110100000000000000000이다.
가수 최소값(32비트)	M의 최소 저장값은 000…0이며 수 형태는 0.1000 0000 0000 0000 0000 0000이다.
가수 최대값(32비트)	M의 최대는 1 − 2^−24 = 0.1111 1111 1111 1111 1111 1111 ≈ 0.99999994039535522460937 5이다.
표현 가능 수 범위	양수: 0.5 × 2^−128 ~ (1 − 2^−24) × 2^127, 음수: −(1 − 2^−24) × 2^127 ~ −0.5 × 2^−128이며, 가수가 24비트를 초과하면 제외된다.
제외되는 범위	−(1 − 2^−24) × 2^127보다 작은 음수는 음수 오버플로우, −0.5 × 2^−128보다 크면 음수 언더플로우, 0, 0.5 × 2^−128보다 작은 양수는 양수 언더플로우, (1 − 2^−24) × 2^127보다 큰 양수는 양수 오버플로우이다.
IEEE 754 표준	부동소수점 표현 방식을 국제 표준으로 정의한다.
IEEE 단정도 형식	N = (−1)^S × (1.M) × 2^(E−127)이며, 가수는 부호화-크기 표현을 사용하고 지수는 바이어스 127을 사용한다.
IEEE 배정도 형식	N = (−1)^S × (1.M) × 2^(E−1023)이다.
예: −13.625의 IEEE 단정도	13.625=1101.101=1.101101 × 2^3, S=1, E=00000011+01111111=10000010(excess-127), M=10110100000000000000000이다.
단정도 예외 규칙	E=255, M≠0이면 NaN; E=255, M=0이면 (−1)^S∞(오버플로우); 0<E<255이면 (−1)^S(1.M)2^(E−127); E=0, M≠0이면 (−1)^S(0.M)2^−126(언더플로우); E=0, M=0이면 (−1)^S0(0)이다.

주제/단락	내용
배정도 예외 규칙	E=2047, M≠0이면 NaN; E=2047, M=0이면 (−1)^S∞(오버플로우); 0<E<2047이면 (−1)^S(1.M)2^(E−1023); E=0, M≠0이면 (−1)^S(0.M)2^−1022(언더플로우); E=0, M=0이면 (−1)^S0(0)이다.
사중정밀도 형식	128비트(quadruple)에서 N = (−1)^S(1.M)2^(E−16383)이며, 가수는 부호화-크기, 지수는 바이어스 16383을 사용하고 범위는 −16382~+16383이다.

항공사 대기자 우선순위 관리 시스템 (C++ & Python 통합 프로젝트)

📋 프로젝트 개요

본 프로젝트는 항공사의 대기 고객 우선순위 계산 문제를 해결하고, 학부 과제의 **기본 구현에 충실**하면서도 **고성능 자료구조 구현** 및 **전문적인 분석**을 통해 혁신성을 입증하는 것을 목표로 합니다.

핵심 차별화 요소 (혁신성)

1. **C++ 고성능 구현**: 핵심 로직(계산, 정렬, 검색)을 C++로 구현하여 실행 속도를 최적화했습니다.
2. **Red-Black Tree (STL `std::map`)**: 고객 이름 검색 인덱스에 Red-Black Tree 기반 자료구조를 사용하여 **O(log n)**의 검색 성능을 보장합니다.
3. **Python 성능 검증**: Python 스크립트가 C++ 엔진을 실행하고, 양쪽의 성능을 벤치마킹하여 **C++ 구현의 속도 우위**를 시각적으로 증명합니다.

🏗️ 시스템 아키텍처

파일	역할	기술/알고리즘
`waitlist_engine.cpp`	**핵심 엔진 (C++)**	우선순위 공식, `std::sort` (Timsort/Introsort), **`std::map` (Red-Black Tree)**, 성능 타이밍
`analysis_visualizer.py`	**분석 및 시각화 (Python)**	`subprocess` 실행, `pandas` 데이터 처리, `matplotlib/seaborn` 시각화
`Makefile`	**자동화**	`g++` 컴파일, 실행 자동화 (`make analyze`)

🧮 우선순위 계산 공식 (과제 요구사항)

$$ \text{우선순위} = A - \frac{B}{B_{MAX}} \times 10 - \frac{C}{C_{MAX}} \times 5 $$

- **A (접수 순번)**: 기본적인 순서 가중치 (높을수록 좋음).
- **B (마일리지)**, **C (가입 기간)**: 충성도 높은 고객일수록 **감산** 항목이 커져 우선순위가 낮아짐 (과제 공식에 따른 해석 및 구현).
- **정렬**: 우선순위 **내림차순** (우선순위 값이 높은 고객이 선순위).

🚀 실행 방법 및 결과 확인

터미널에서 `Makefile`을 사용하여 프로젝트를 실행합니다.

1. **C++ 엔진 빌드 및 실행:**

```
make run_cpp
```

2. **종합 분석 및 시각화 실행 (권장):**

```
make analyze
```

   - 이 명령어는 C++ 엔진을 실행하고, Python 분석기가 결과 파일(`정렬된_대기자명단_cpp.txt`)을 로드하여 **`analysis_report.png`** 시각화 파일을 생성합니다.

3. **결과물:**

   - `정렬된_대기자명단_cpp.txt`: 정렬된 최종 대기 명단 (C++ 출력).
   - `analysis_report.png`: 성능 비교 및 데이터 분석 그래프 (Python 출력).