**CPU 스케줄링 알고리즘**

**팀 명: 모도리 (운영체제 13조)**

**팀원1: 2017152008 김지은 (FCFS 알고리즘)**

**팀원2: 2017156037 정수경 (Priority-Based 알고리즘)**

**팀원3: 2017154043 권한길 (Round-Robin 알고리즘)**

**목차**

**1. 팀 프로젝트 진행 일정………………………………………………….p4**

**2. 스케줄링 알고리즘 선택 이유………………………………………...p4**

**3. FCFS 알고리즘 (김지은)………………………………………………...p5**

**a. 스케줄링 알고리즘 설명**

**b. 사용한 자료 구조 설명**

**c. 구현 방법 (함수 및 변수 설명)**

**d. 소스코드 부착**

**e. 데모 결과 캡쳐 및 설명**

**4. Priority-Based 알고리즘 (정수경)…………………………………p19**

**a. 스케줄링 알고리즘 설명**

**b. 사용한 자료 구조 설명**

**c. 구현 방법 (함수 및 변수 설명)**

**d. 소스코드 부착**

**e. 데모 결과 캡쳐 및 설명**

**5. Round-Robin 알고리즘 (권한길)…………………………………..p37**

**a. 스케줄링 알고리즘 설명**

**b. 사용한 자료 구조 설명**

**c. 구현 방법 (함수 및 변수 설명)**

**d. 소스코드 부착**

**e. 데모 결과 캡쳐 및 설명**

**5. 분석 및 결론……………………………………………………………..p48**

**a. 사용한 스케줄링 알고리즘의 정규화된 반환시간 비교 분석**

**b. 결론 및 총평**

**1. 팀프로젝트 진행 일정**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 활동명 | 시작일 | 완료일 | 기간 | 일정 | | | | |
| 1주 | 2주 | 3주 | 4주 | 5주 |
| 1 | 미팅 | 19.04.29 | 19.05.30 | 31일 |  |  |  |  |  |
| 2 | 구현 | 19.04.29 | 19.05.19 | 20일 |  |  |  |  |  |
| 3 | 보고서 작성 | 19.05.19 | 19.05.30 | 12일 |  |  |  |  |  |
| 4 | PPT 제작 | 19.05.27 | 19.05.29 | 3일 |  |  |  |  |  |
| 5 | 발표 시뮬레이션 | 19.05.30 | 19.06.03 | 5일 |  |  |  |  |  |
| 6 | 보고서 최종본 완성 | 19.06.03 | 19.06.012 | 10일 |  |  |  |  |  |

**2. 스케줄링 알고리즘 선택 이유**

**김지은(FCFS 알고리즘)**

FCFS 스케줄링 알고리즘은 가장 기본적인 알고리즘으로 모든 스케줄링 알고리즘 방식 중에서 근간이 되므로 FCFS 스케줄링 알고리즘을 구현해보면 다른 스케줄링 알고리즘들도 응용해서 구현을 할 수 있을 것이라 생각했기 때문에 FCFS 스케줄링 알고리즘을 선택하게 되었다.

**정수경(Priority-Based 알고리즘)**

우선순위 기반 알고리즘을 선택한 이유는 수업시간에 교수님이 CPU 스케줄링 알고리즘에 대해 설명하실 때, 이 알고리즘의 원리가 제일 이해하기 쉬웠기 때문이다. 또한 과제를 받았을 때, 머릿속으로 어떻게 짜야 할지 직관적으로 그려볼 수 있었다. 그래서 팀원들과 업무 분담을 할 때, 자신 있게 이 알고리즘을 선택하였다.

**권한길(Round-Robin 알고리즘)**

라운드 로빈 알고리즘을 선택한 이유는 TIME QUANTUM 이라는 특이점과 공평하게 시간을 할당하여 프로세스를 처리하는 알고리즘이 매력적이었습니다. 또한 TIME QUANTUM이 너무 짧다면 문맥교환으로 오버헤드가 증가하고 TIME QUANTUM이 너무 길면 FCFS와 비슷한데 적정한 TIME QUANTUM을 이 과제를 통해서 발견할 수 있지 않을까 하는 생각에 선택했습니다.

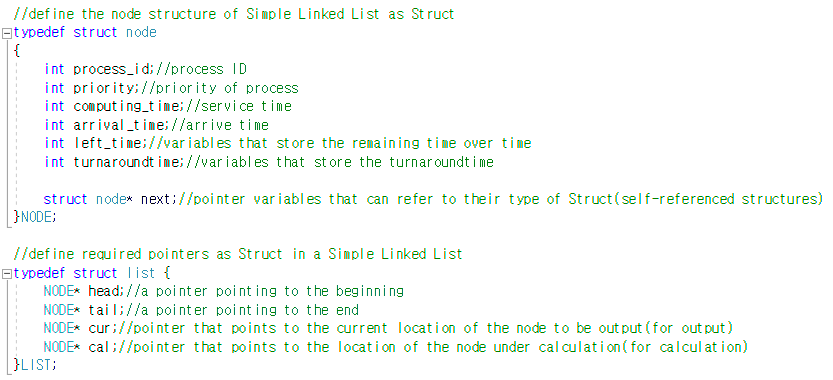
**3. FCFS 알고리즘(김지은)**

**a. 스케줄링 알고리즘 설명**

FCFS 스케줄링은 First-Come, First-Served의 약자로 선입 선처리 스케줄링, 선입선출 스케줄링으로 불리기도 한다. 가장 단순한 스케줄링이며 비선점 스케줄링(한 프로세스가 CPU를 할당 받았을 때 CPU를 스스로 반납할 때까지 계속 사용하도록 허용하는 방식) 방식 중 한 가지로 먼저 사용신청을 한 프로세스부터 차례로 CPU를 할당한다. CPU를 독점하여 사용하기 때문에 긴 프로세스가 실행될 경우에는 뒤에 있는 프로세스들의 대기시간이 길다는 단점이 있어 대화식 시스템에는 적합하지 않다.

**b. 사용한 자료 구조 설명**

**① 구조체**



FCFS 스케줄링 알고리즘에서는 두개의 구조체를 정의하여 사용하였다.

1. 단순 연결리스트의 노드 구조를 구조체로 정의

프로그램의 출력 필드에 사용되는 변수 이외에도 스케줄링 하는데에 필요한 연관된 데이터들인 arrival\_time과 left\_time 변수와 구조체 포인터 변수인 next를 추가하여 그룹화 하였다.

* 변수 arrival\_time 추가

turn around time 계산을 위해 도착시간을 나타내는 변수가 필요하여 변수 arrival\_time을 추가해주었다.

* 변수 left\_time 추가

각 프로세스를 스케줄링할 때 시간 경과에 따른 남은 수행시간을 저장하는 변수로 각 프로세스의 스케줄링 완료 여부를 확인하기 위해 left\_time을 추가해주었다.

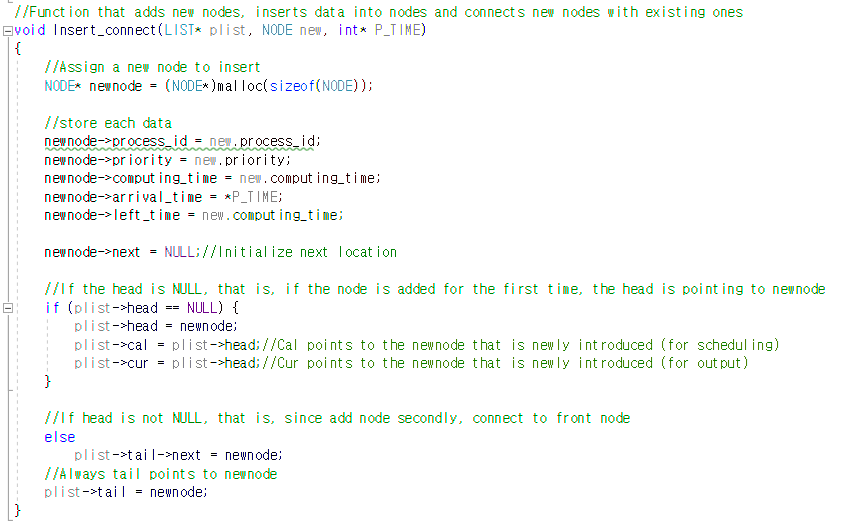
* 구조체 포인터 next 추가

연결리스트를 구현하기 위해 필요한 다음 노드를 가리키는 구조체 포인터를 추가해주었다.

1. 단순 연결리스트에서 필요한 포인터들을 구조체로 정의

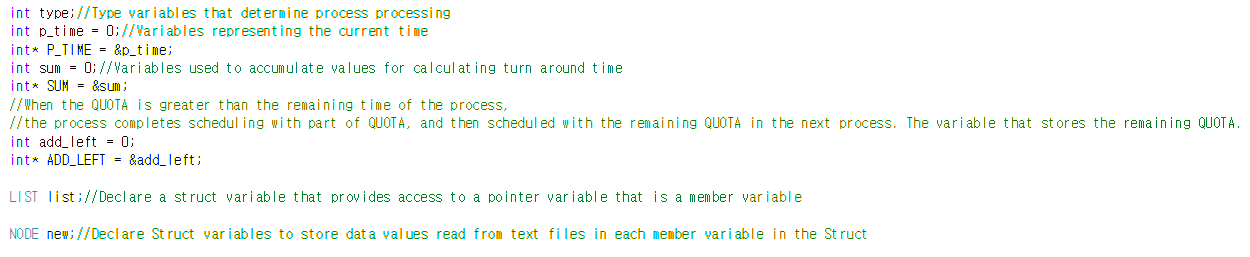
연결리스트에서 필요한 포인터 변수들을 그룹화하여 사용되는 각각의 포인터 변수가 아닌 구조체 통째를 함수의 매개변수로 건네줌으로써 함수들의 매개변수를 간결하게 하였다.

**② 단순연결리스트**



Insert\_connect() 함수에서 단순연결리스트가 사용되었음을 쉽게 알 수 있다. Insert\_connect() 함수는 새로운 노드를 추가하고 노드에 데이터 삽입 및 기존 노드와 새로 들어온 노드를 연결해주는 함수인데 프로세스(노드)가 들어온 시간, 즉 arrival\_time을 기준으로 앞에서부터 프로세스(노드)를 삽입하여 반복해서 새 노드를 추가해준다. 데이터 필드에는 데이터 값을, 링크 필드에는 새 노드의 시작 주소를 저장해서 기존의 노드와 새로 추가된 노드를 연결해주어 연결리스트를 구현하게 된다.

**c. 구현 방법 (함수 및 변수 설명)**

**① 주요 변수**  

**type** : 프로세스의 처리를 결정하는 변수

type이 1일 때 – 시간할당량이 다되어 스케줄 됨

type이 0일 때 – 프로세스 생성 및 데이터 값 입력받음

type이 -1일 때 – 데이터의 입력이 완료되었으니 남은 프로세스들 스케줄링처리

**p\_time** : 현재 시간을 나타내는 변수로 값을 누적시키기 위해 0으로 초기화시킴

**P\_TIME** : p\_time 변수의 주소를 저장하는 포인터 변수

**sum** : turn around time을 계산할 때 필요한 데이터 값을 누적시키기 위해 사용되는 변수로 값을 누적시키기 위해 0으로 초기화시킴

**SUM** : sum 변수의 주소를 저장하는 포인터 변수

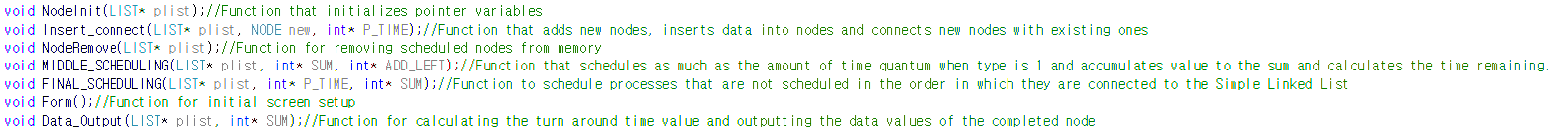
**add\_left** : 시간할당량의 값이 해당 프로세스의 남은시간보다 커서 해당 프로세스에서 QUOTA의 일부를 가지고 스케줄링을 완료시킨 후, 다음 프로세스에서 앞노드에서 스케줄링하고 남은 QUOTA로 스케줄링하게 되는데 이 때 남은 QUOTA를 저장하는 변수

**ADD\_LEFT** : add\_left 변수의 주소를 저장하는 포인터 변수

**list** : 멤버 변수인 포인터 변수들에 접근하게 해주는 구조체 변수

**new** : 텍스트파일에서 읽은 데이터 값들을 구조체 안 각각의 멤버 변수들에 저장하기 위해 사용하는 구조체 변수

**② 사용한 함수**



**(1) void NodeInit(LIST\* plist);**

**함수 명** : NodeInit

**함수 기능** : 포인터 변수의 쓰레기 값을 방지하기 위해서 구조체 안 포인터 변수를 NULL로 초기화 시켜주는 함수

**매개변수 의도**

**plist :** 각 노드들의 연결, 계산, 출력에서 사용되는 포인터 변수들을 구조체로 묶어 놓았기에 구조체 안 포인터 변수들을 초기화시키기 위해서 매개변수로 입력 받음

**(2) void Insert\_connect(LIST\* plist, Node new, int \* P\_TIME);**

**함수 명** : Insert\_connect

**함수 기능** : 도착시간을 기준으로 새로 들어온 노드에 데이터를 삽입하여 추가하고 기존 노드와 새로 들어온 노드를 연결해주는 함수

**매개변수 의도**

**plist :** 각 노드들의 연결, 계산, 출력에서 사용되는 포인터 변수들을 구조체로 묶어 놓았기에 노드끼리의 연결(cal, cur 포인터는 타 함수에서 계산, 출력에 이용됨)에 사용될 포인터들을 이용하기 위해 매개변수로 입력받음

**new** : 새로 할당 받은 노드에 텍스트파일에서 읽어서 저장한 값들을 전달하기 위해 매개변수로 입력받음

**P\_TIME** : 현재 시간의 경과를 전달받기 위해 매개변수로 입력받음

**(3) void NodeRemove(LIST\* plist);**

**함수 명** : NodeRemove

**함수** **기능** : 스케줄링 완료한 노드를 메모리에서 제거하기 위한 함수

**매개변수 의도**

**plist** : Data\_Output()함수에서 스케줄링 완료한 노드(프로세스)를 메모리에서 제거하기 위해서는 구조체 안에 저장된 포인터 변수인 cur이 사용되므로 매개변수로 입력받음

**(4) void MIDDLE\_SCHEDULING(LIST\* plist, int\* SUM, int\* ADD\_LEFT);**

**함수 명** : MIDDLE\_SCHEDULING

**함수 기능** : 읽어들인 type의 값이 1일 때 시간할당량만큼 스케줄링 시켜주고 turn around time 계산을 위해 sum에 값을 누적해주는 함수, 만약 해당 프로세스(노드)의 스케줄링이 끝났을 때엔 Data\_Output()함수를 호출하여 turn around time 값을 계산 및 데이터 값들을 출력해주고, 현재 노드에서 QUOTA만큼 스케줄링이 불가할 경우, 즉 left\_time<QUOTA인 경우에는 add\_left 값을 계산해주어 다음 노드에서 add\_left만큼 스케줄링 할 수 있도록 한다.

**매개변수 의도**

**plist** : 구조체 안에 저장된 포인터 변수인 cal을 계산중인 노드의 위치를 가리키는 포인터로 사용해야 하므로 매개변수로 입력받음

**SUM** : turn around time 계산을 위해서 값을 누적하기 위해 매개변수로 입력받음

**ADD\_LEFT** : QUOTA만큼 스케줄링을 해야 하는데 해당 프로세스의 left\_time의 값이 QUOTA보다 작을 경우에는 해당 프로세스에서 QUOTA의 일부(해당 프로세스의 스케줄링이 종료됨), 다음 프로세스에서 남은 QUOTA만큼 스케줄링되므로 다음 프로세스에서 스케줄링할 남은 QUOTA를 저장하기 위해 매개변수로 입력받음

**(5) void FINAL\_SCHEDULING(LIST\* plist, int\* P\_TIME, int\* SUM);**

**함수 명** : FINAL\_SCHEDULING

**함수 기능**: 스케줄링이 끝나지 않은 프로세스들에 대해 연결리스트에 연결된 순서대로 스케줄링하는 함수

**매개변수 의도**

**plist** : 구조체 안에 저장된 포인터 변수인 cal을 계산중인 노드의 위치를 가리키는 포인터로 사용해야 하므로 매개변수로 입력받음

**P\_TIME** : 현재 시간의 경과를 전달받기 위해 매개변수로 입력받음

**SUM** : turn around time 계산을 위해서 값을 누적하기 위해 매개변수로 입력받음

**(6) void Form();**

**함수 명** : Form

**함수 기능** : 초기 화면 셋팅을 위한 함수

**(7) void Data\_Output(LIST\* plist, int\* SUM);**

**함수 명**: Data\_Output

**함수 기능** : turn around time 값 계산 및 스케줄링 완료한 노드의 데이터 값들을 출력하기 위한 함수

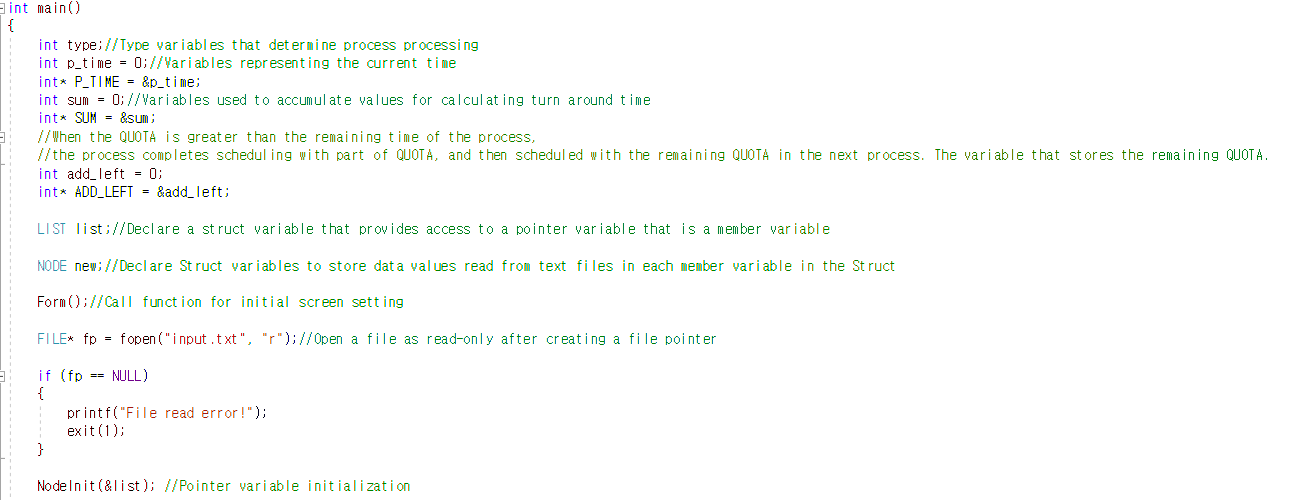
**매개변수 의도**

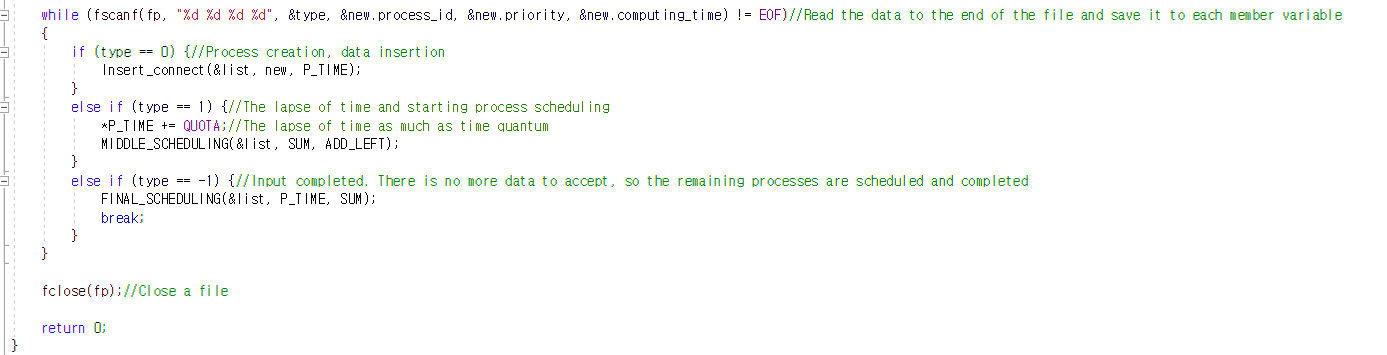
**plist** : 구조체 안에 저장된 포인터 변수인 cur이 가리키고 있는 노드의 turn around time 값 계산 및 데이터 값들을 출력해야 하므로 매개변수로 입력받음

**SUM** : 앞에 있는 MIDDLE\_SCHEDULING()함수와 FINAL\_SCHEDULING()함수에서 turn around time 계산을 위한 누적된 시간 값의 데이터를 받아오기 위해 매개변수로 입력받음

**③ 핵심 알고리즘 설명**

1. **main문**
2. **코드**





1. **main 안의 큰 실행 흐름 3가지**

**1단계 :** 필요한 변수 선언 및 초기 화면 셋팅 함수 Form() 호출

**2단계 :** 파일을 읽으면서 각 type 값에 알맞은 함수 호출

type이 0일 때 – 프로세스 생성 및 데이터 값 입력받음 & Insert\_connect()함수 호출

type이 1일 때 – 시간할당량이 다되어 스케줄 됨 & p\_time 값 QUOTA만큼 증가 및 MIDDLE\_SCHEDULING()함수 호출

type이 -1일 때 – 데이터가 입력 완료되었으니 남은 프로세스들 스케줄링

처리 & FINAL\_SCHEDULING()함수 호출

**3단계:** 파일 닫기

1. **알고리즘 순서도(while문)**

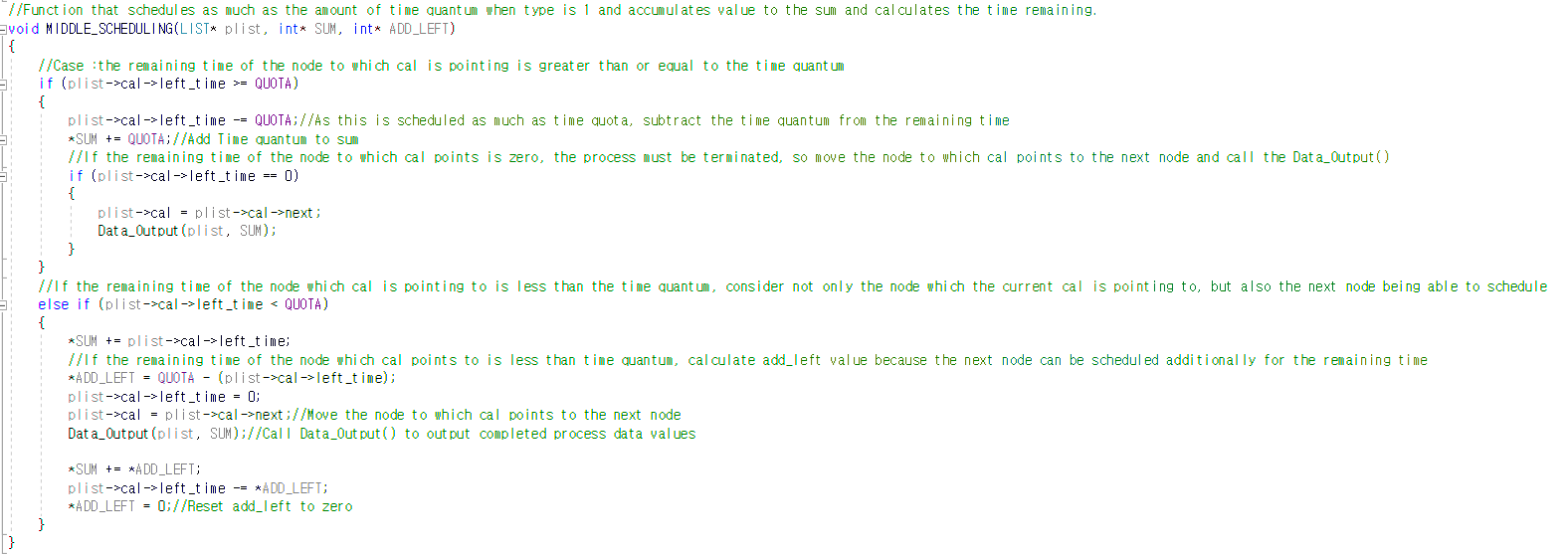
지도, 텍스트이(가) 표시된 사진

자동 생성된 설명

* fscanf() 통해서 파일 끝까지 한 줄씩 데이터를 읽어들여서 더 이상 읽어들일 수 있는 데이터가 없을 때까지 while()문을 실행시킨다.
* 한 줄씩 읽어들인 데이터의 값들 중 type이 0일 때엔 Insert\_connect()를 호출하여 프로세스를 생성하고 읽어들인 데이터의 값을 저장한다. 후에 head(첫번째로 연결될 때) 또는 기존노드와 새로들어온 노드를 연결한다. 노드끼리의 연결을 마치면 Insert\_connect()를 빠져나와 다시 파일 읽기를 시작한다..
* 한 줄씩 읽어들인 데이터의 값들 중 type이 1일 때엔 p\_time변수에 QUOTA만큼 더한 값을 다시 p\_time 변수에 대입하여 준다. 즉, 현재시간이 시간할당량만큼 지났음을 알리고 이후 들어오는 프로세스들의 도착시간 또한 변한다는 것을 알 수 있다. 후에 MIDDLE\_SCHEDULING()을 호출하여 시간할당량만큼 프로세스를 스케줄링 시켜준 후 함수 내에서의 모든 작업이 끝나면 함수를 탈출하여 다시 파일 읽기를 시작한다.
* 한 줄씩 읽어들인 데이터의 값들 중 type이 -1일 때엔

FINAL\_SCHEDULING()를 호출하여 스케줄링이 끝나지 않은 프로세스들에 대해 연결리스트에 연결된 순서대로 스케줄링을 시작하여 남은 프로세스들의 스케줄링을 끝냄과 동시에 turn around time 계산을 위한 sum(대기시간+실행시간)의 값을 누적시킨다. FINAL\_SCHEDULING()안에서의 모든 작업들이 끝나면 함수를 빠져나와 break를 통해 while문을 종료시킨다.

1. **MIDDLE\_SCHEDULING() 함수**
2. **코드**



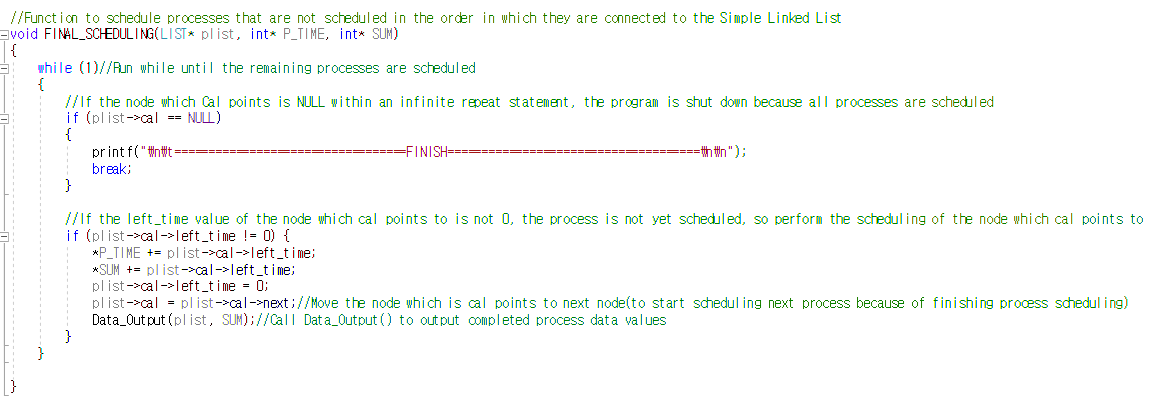
1. **알고리즘 순서도**

텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

* cal이 가리키는 노드의 남은 시간이 시간할당량보다 크거나 같을 경우에는 시간할당량만큼 스케줄링해야 하므로 left\_time에서 시간할당량만큼을 빼주고 sum에는 시간할당량만큼 누적시켜준다. 이 때 해당 프로세스의 left\_time이 0이라면 해당 프로세스를 종료시켜야 하므로 cal이 가리키는 노드를 다음 노드로 이동시켜준 후 데이터 출력함수인 Data\_Output()을 호출시킨다.
* cal이 가리키는 노드의 남은 시간이 시간할당량보다 작을 경우에는 현재 cal이 가리키는 노드뿐만 아니라 다음노드(일부 혹은 전체)까지 스케줄링이 가능한 것까지 고려하여야 한다. 현재 cal이 가리키는 노드를 QUOTA의 일부를 가지고 스케줄링을 완료시킨 후, cal이 다음 노드를 가리키게 하여 cal이 가리키는 노드를 앞노드에서 스케줄링하고 남은 QUOTA로 스케줄링하게 되는 원리이다. 일단 cal이 가리키는 노드의 남은 시간만큼 sum에 더해주고 add\_left 변수에는 QUOTA에서 cal이 가리키는 노드의 남은시간을 제외한 값을 넣어준다. cal이 가리키는 노드의 남은 시간을 0으로 초기화시키고 cal이 가리키는 노드를 다음노드로 이동시켜준 후 스케줄링을 완료한 프로세스의 데이터 값을 출력하기 위해 Data\_Output()을 호출해준다. sum에 앞 노드에서 스케줄링하고 남은 QUOTA만큼, 즉, add\_left만큼 더해준 후 left\_time에서 스케줄링할 add\_left만큼 빼준 값을 다시 left\_time에 대입하여 준다. 마지막으로 다음 변수 사용을 위해 다시 add\_left는 0으로 초기화시킨다.

1. **FINAL\_SCHEDULING ()함수**
2. **코드**



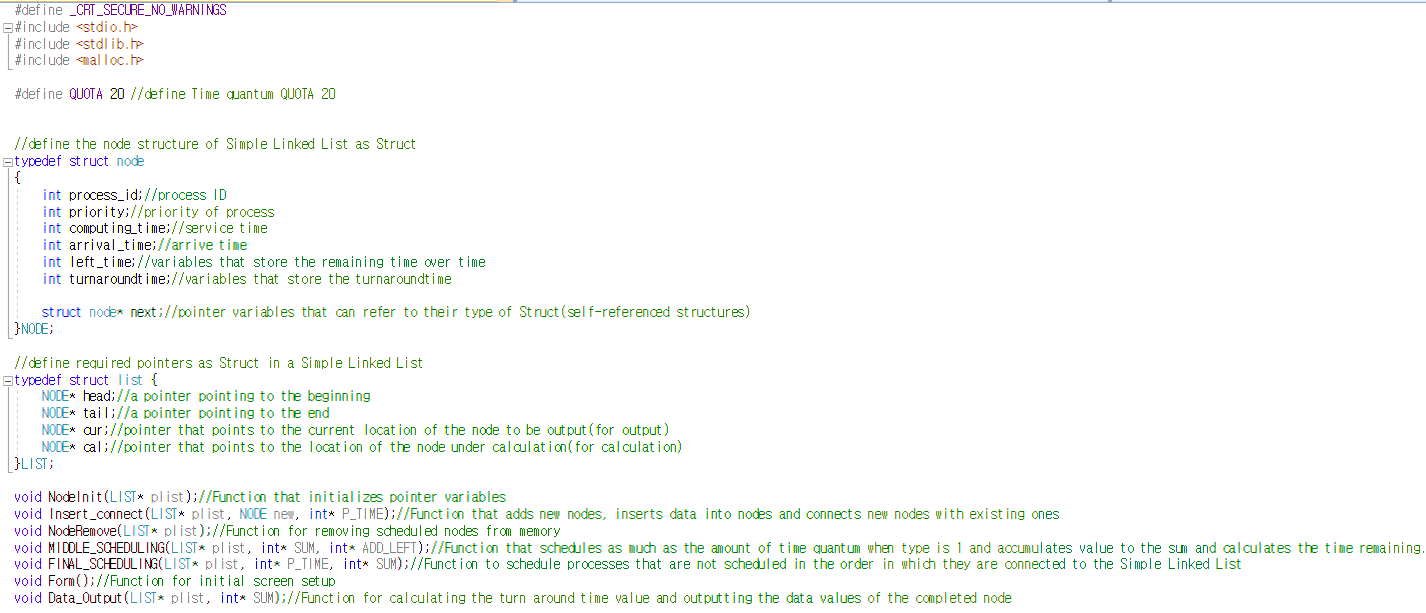
1. **알고리즘 순서도**

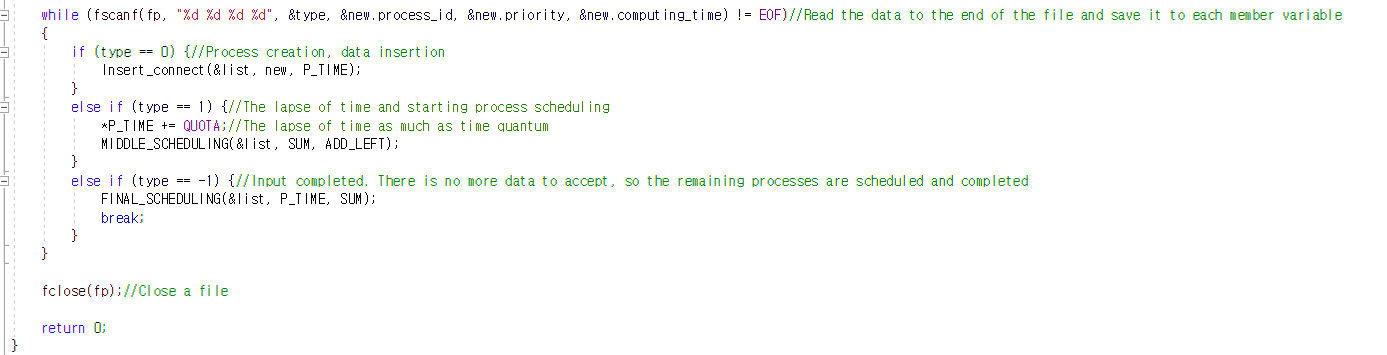
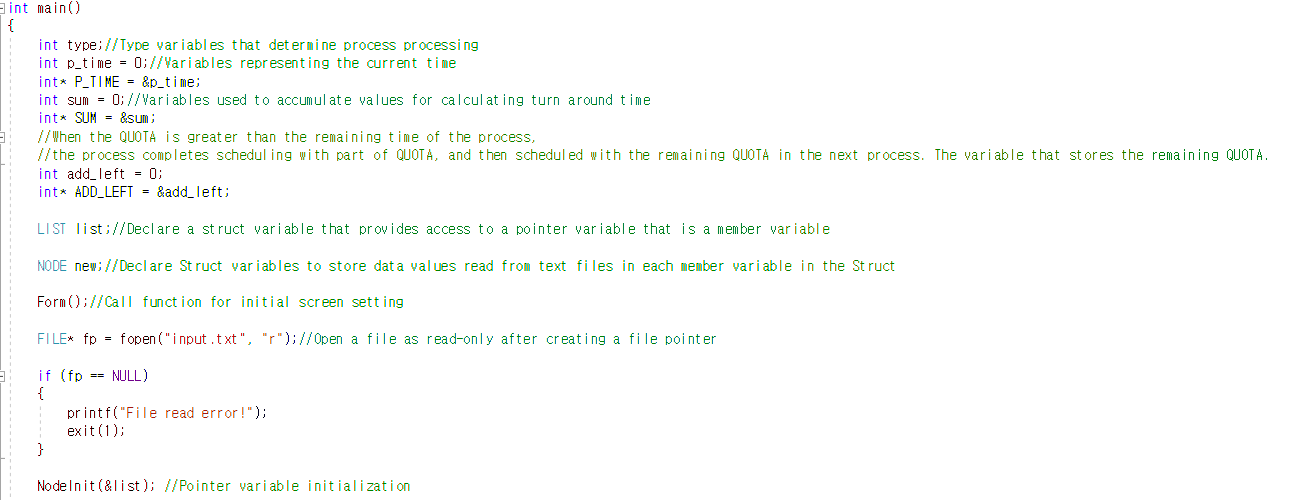
텍스트이(가) 표시된 사진

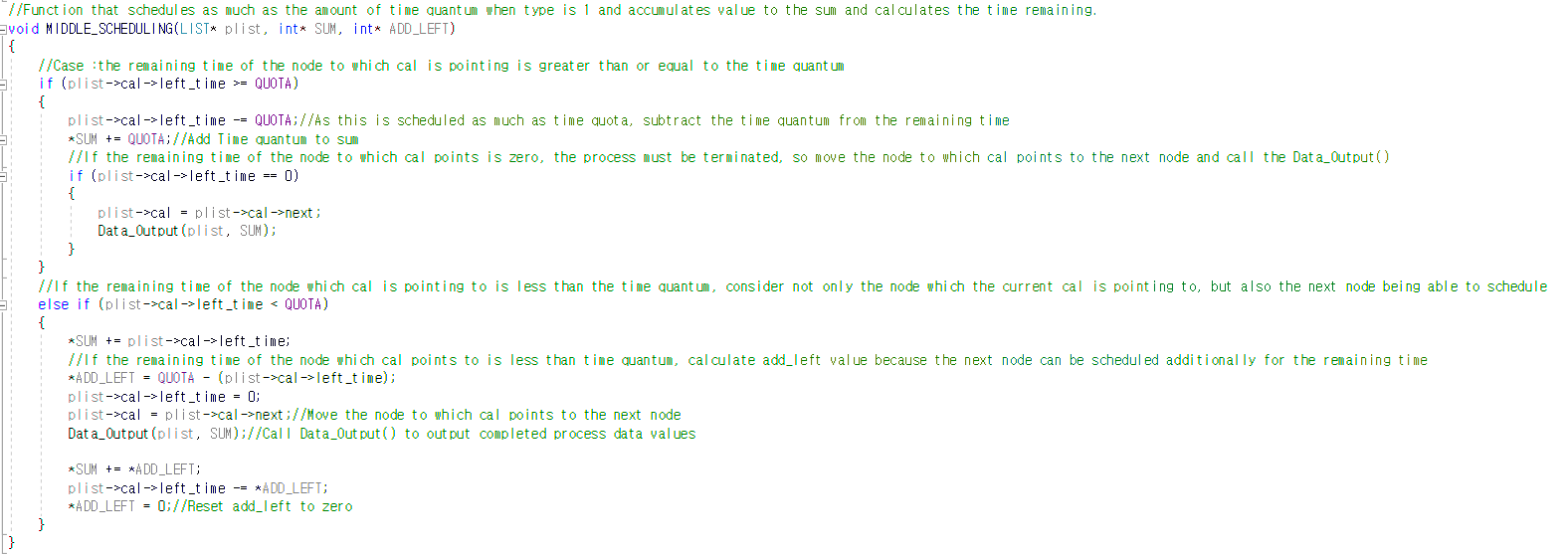
자동 생성된 설명

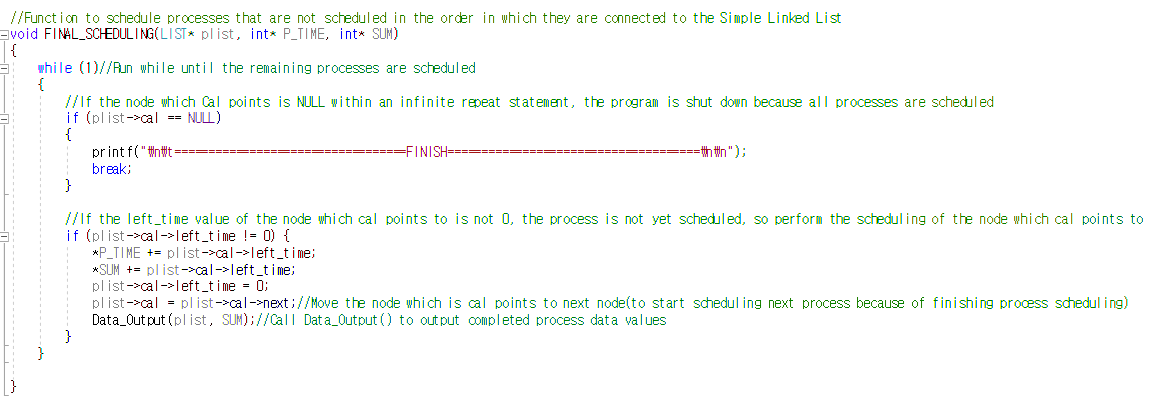
* while()문을 통해 남은 프로세스들의 스케줄링이 끝날 때까지 무한 반복문을 실행시킨다.
* 만약 cal이 가리키는 노드가 NULL이라면 모든 프로세스의 스케줄링이 완료된 것이므로 ‘------FINISH------‘의 문구를 출력 후 while()문을 종료시킨다.
* cal이 가리키는 노드의 left\_time의 값이 0이 아니라면, 즉, 스케줄링이 완료되지 않은 노드가 있다면 현재 시간을 나타내는 변수 p\_time과 실행시간+대기시간의 합을 나타내는 변수 sum에 cal이 가리키는 노드의 남은 시간 만큼을 더해주고 left\_time은 0으로 초기화 시켜준다. 후에 cal이 가리키는 노드를 다음 노드로 이동시켜주는데 이 과정은 cal이 가리키고 있는 프로세스의 스케줄링을 종료시켜주고 다음 노드로 이동하여 다음 프로세스의 스케줄링을 하기 위함이다. 마지막으로 완료시킨 프로세스의 데이터 값을 출력하기 위해 Data\_Output()을 호출시켜준다. Data\_Output()의 모든 작업이 완료되면 함수를 빠져나와 다시 while()문을 시작한다.

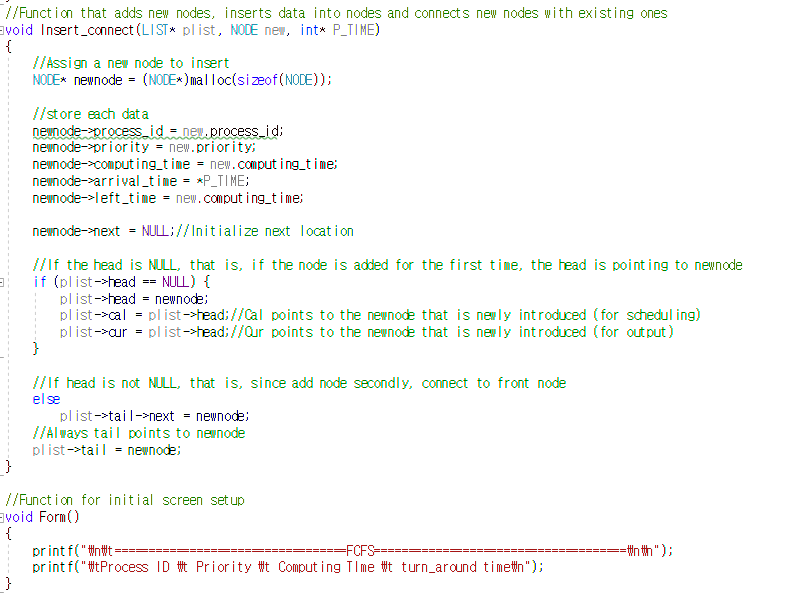
**d. 소스코드 부착**

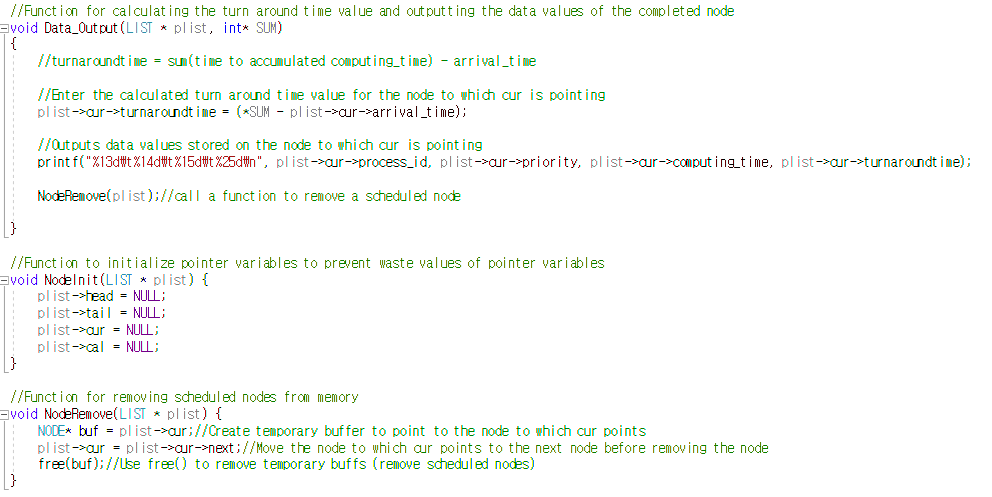












**e. 데모 결과 캡쳐 및 설명**



데모 결과 캡쳐화면을 보면 Process ID의 순서대로 스케줄링이 완료되어 turn\_around time이 계산되었음을 알 수 있는데, 이는 FCFS 알고리즘, 즉 선입선처리가 제대로 이루어졌음을 알 수 있다. 또한 출력된 turn\_around time의 값을 통해 도착시간이 시간할당량만큼 반영되어 도착시간의 변화에 따라 제대로 계산되었음을 알 수 있다.

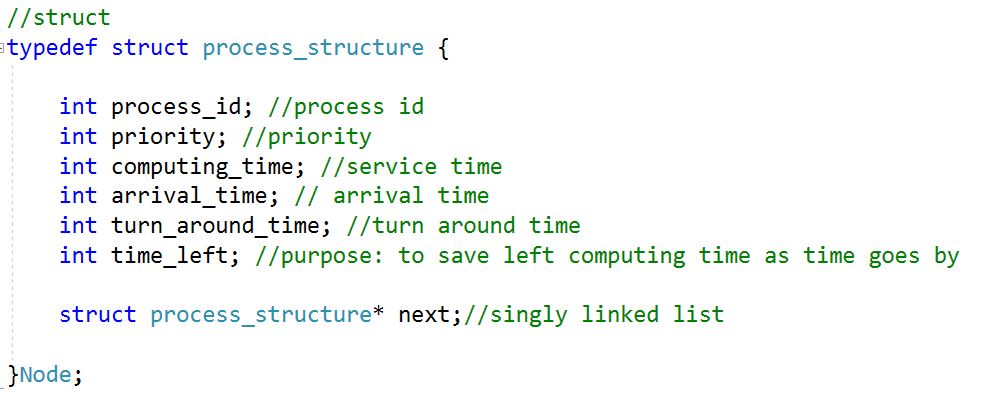
**4. Priority-Based 알고리즘(정수경)**

**a. 스케줄링 알고리즘 설명**

‘Priority-Based 알고리즘’이란 각각의 프로세스마다 우선순위를 부여해서 우선순위가 높은 프로세스를 먼저 실행시키는 CPU 스케줄링 알고리즘이다. 예를 들어 우선순위가 7인 프로세스와 우선순위가 2인 프로세스가 동시에 들어올 때, 우선순위가 2인 프로세스는 우선순위가 7인 프로세스가 종료되기 전에 결코 실행되지 못하며 실행 완료될 때까지 기다려야 한다. (우선순위 필드 값이 클수록 높은 우선순위를 부여한다고 가정)

**b. 사용한 자료 구조 설명**

**① 구조체**



프로그램 출력 필드를 보면 프로세스 id, 우선순위, 서비스 시간, turnaround time이 있다.

* 변수 arrival\_time 추가

-> 하지만 각 프로세스(노드)의 turn around time을 계산하려면 ‘도착시 간’ 멤버가 추가적으로 필요하다.

* 변수 time\_left 추가

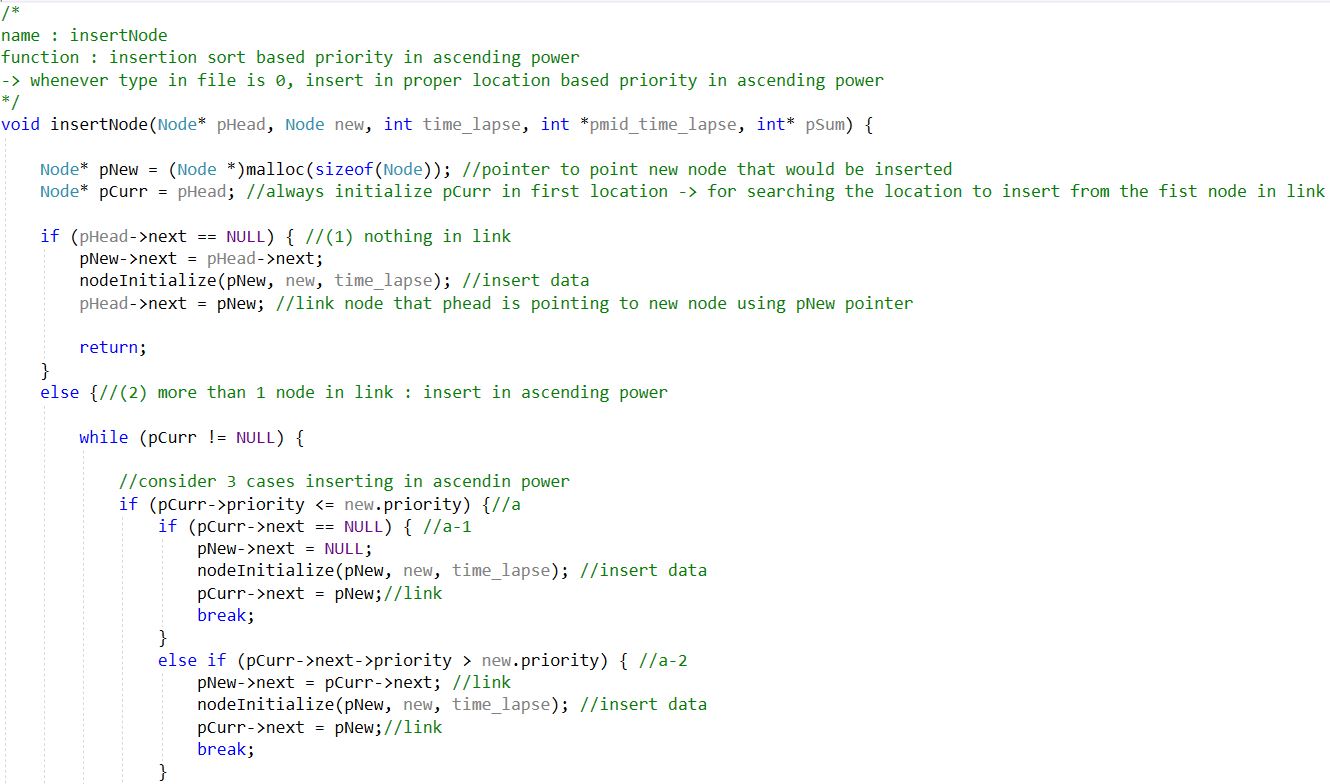
-> 또한 각 프로세스의 수행시간을 고려하여 스케줄링 완료 여부를 결정하려면 시간의 경과에 따라 남은 수행 시간을 따로 저장해야 할 필요가 있다.

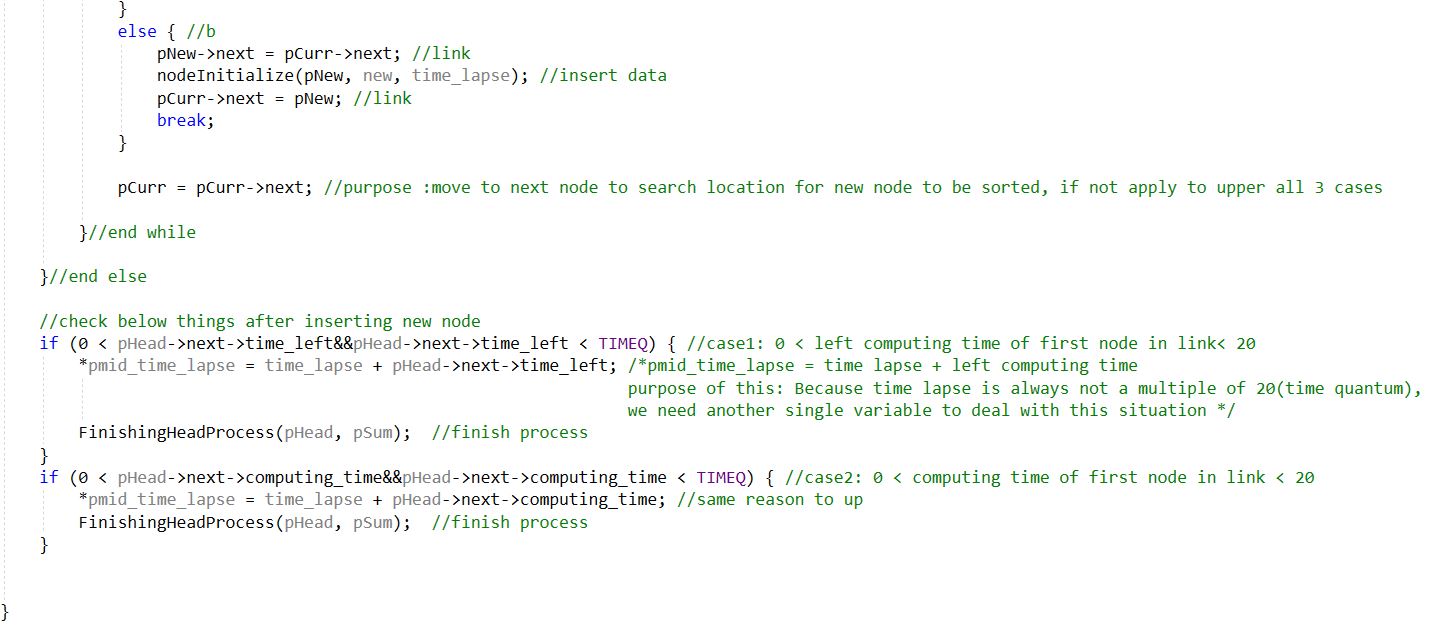
* 구조체 포인터 next 추가

->연결리스트를 구현하기 위해 다음 노드를 가리키는 구조체 포인터를 사용.

**② 단순 연결리스트**

-> 다음은 프로세스(노드)를 새로 우선순위 기준으로 알맞은 위치에 새로 삽입할 때 사용한 코드다. 새 노드를 추가할 때마다 노드의 포인터(링크) 부분에 연결할 노드의 시작 주소를 저장하거나 삽입할 위치의 앞에 있는 노드의 포인터에 삽입할 새 노드의 시작 주소를 저장한다. 이로써 연결리스트를 구현한다.

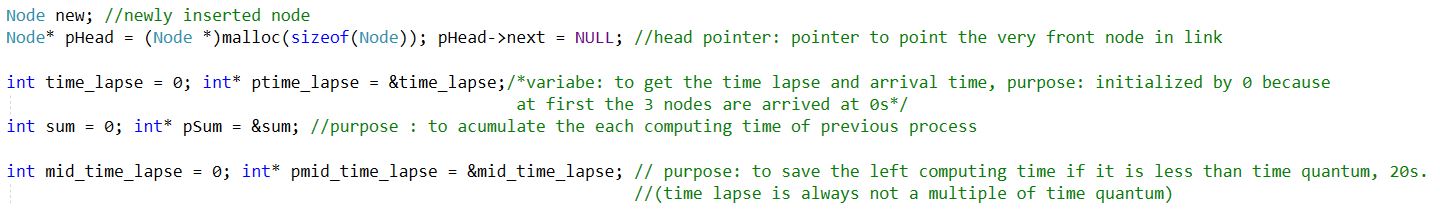




**c. 구현 방법 (함수 및 변수 설명)**

**① 주요 변수**





**TIMEQ**: 시간 할당량, time quantum으로 이번 설계과제의 예제에서는 타임 퀀텀 이 20이였지만 어떤 타임 퀀텀에서도 작성한 알고리즘이 작동할 수 있도록 매크 로 상수 로 선언을 하고 시간할당량이 바뀔 때 마다 코드의 맨 위에 위치한 매크 로 상수 의 값만 수정할 수 있도록 할 수 있게 하였다.

**new**: input 파일로부터 데이터를 읽어 들여 구조체에 저장할 때 사용되는 변수

**pHead**: 항상 연결리스트의 맨 앞 노드를 가리키고 있는 헤드 포인터

**time\_lapse**: 시간의 경과 정도를 저장하는 변수

**ptime\_lapse**: time\_lapse 변수를 가리키는 포인터 변수

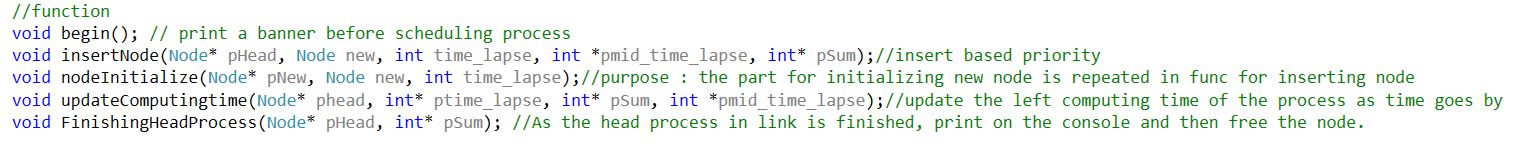
**sum**: 스케줄링 완료된 프로세스의 수행시간(compuing\_time)과 현재 맨 앞에 있 어 곧 완료될 프로세스의 수행시간을 더한 값 sum에 대한 포인터, 나중에 TAT를 계산하기 위해 설정함. TAT(턴어라운드타임) = sum – arrival\_time(도착시간)

**pSum**: sum 변수를 가리키는 포인터 변수

**mid\_time\_lapse**: 20초 단위로 시간 경과되지 않을 경우 그 시간을 저장해 놓을 변수, 즉 구체적으로 설명하자면 남은 수행 시간이 0초와 20초 사이여서 시간 경 과를 20초로 나누어 떨어지게 경과되지 않는 경우를 말함, 예를 들어 현재 20초 가 경과한 상태이고 time\_left가 10이여서 10초만 경과하면 맨 앞에 있는 프로세 스를 완료시킬 수 있는 상황이다. 그럴 경우 time\_lapse는 20초 단위로 시간이 경과하게끔 설정해 놔서 사용할 수 없다. 따라서 mid\_time\_lapse 변수에 30초를 저장하고 스케줄링을 완료시키는 것이다.

**pmid\_time\_lapse**: mid\_time\_lapse를 가리키는 포인터 변수

**② 사용한 함수**



**(1) void begin();**

**- 함수 명**: begin

**- 기능**: 화면의 배너 출력

**(2) void insertNode(Node\* pHead, Node new, int time\_lapse, int \*pmid\_time\_lapse, int\* pSum);**

- **함수 명**: insertNode

- **함수 기능**: 우선순위 기준으로 오름차순으로 삽입 정렬

type이 0일 때마다 뒤로 갈 수록 우선순위가 큰 것이 위치하도록 정렬하면서 삽입

- **매개변수 의도**

**pHead**: 연결리스트의 맨 앞에 어떤 노드가 있는지 알기 위해 헤드 포인터 필요

**new**: 매번 노드 새롭게 추가하기 위해 사용하는 구조체

**time\_lapse**: 현재 시간의 경과를 전달

**pmid\_time\_lapse**: mid\_time\_lapse에 대한 포인터(남은 수행 시간이 20초 단위의 경과시간보다 적어 20초 단위로 시간이 경과 되지 않을 경우를 말함, 예를 들어 time\_left가 10인데 다음 시간 경과는 40초일 경우 이 프로세스는 내가 정의한 ‘중간시간경과’ 30초에 스케줄링 완료가 되게 된다.)

**pSum**: 앞서 스케줄링 완료된 프로세스의 수행시간(compuing\_time)과 현재 맨 앞에 있어 곧 완료될 프로세스의 수행시간을 더한 값 sum에 대한 포인터, 나중에 TAT를 계산하는데 사용됨

**(3) void nodeInitialize(Node\* pNew, Node new, int time\_lapse);**

- **함수 명**: nodeInitialize

- **함수** **기능**: 삽입 정렬 inserNode()함수에서 노드 값을 초기화하는 코드가 반복되서 가독성과 효율성을 위해 nodeInitialize 함수를 새로 만들어 빼줌

- **매개변수 의도**

**pNew**: 새롭게 추가할 노드를 가리킬 포인터를 전달

**new**: 새롭게 추가할 노드의 구조체 전달

**time\_lapse**: 구조체 안의 도착시간(arrival\_time), 즉 프로세스의 도착시간에 20초 단위로 변화하는 시간의 경과를 넘겨주기 위해

**(4) void updateComputingtime(Node\* phead, int\* ptime\_lapse, int\* pSum, int \*pmid\_time\_lapse);**

- **함수 명**: updateComputingtime

- **함수 기능**: 시간의 경과를 반영하여 맨 앞에 있는 프로세스의 computing\_time 을계산해서 업데이트, 만약에 남은 처리시간이 0이라면 FinishingHeadProcess() 함수를 호출해서 스케줄링 완료시키기

- **매개변수 의도**

**ptime\_lapse와 pmid\_time\_lapse**: 20초 단위로 시간이 경과할 경우와 아닌 경우를 생각해서 ptime\_lapse를 가리키고 있는 포인터 pmid\_time\_lapse 두 개의 변수를 넘겨줌

**phead와 pSum**: 맨 앞에 있는 프로세스의 처리시간이 0이 되었을 경우 매개변수로 phead와 pSum을 가지는 FinishingHeadProcess()함수를 호출하기 위해서

**(5) void FinishingHeadProcess(Node\* pHead, int\* pSum);**

- **함수 명**: FinishingHeadProcess

- **기능**: 우선순위에 의해 맨 앞에 놓여있는 프로세스의 남은 처리시간이 0이므로 스케줄링 완료시키기

-> 1단계: TAT 계산

2단계: 맨 앞 프로세스(노드) 정보 출력

3단계: 동적할당 해제

- **매개변수 의도**

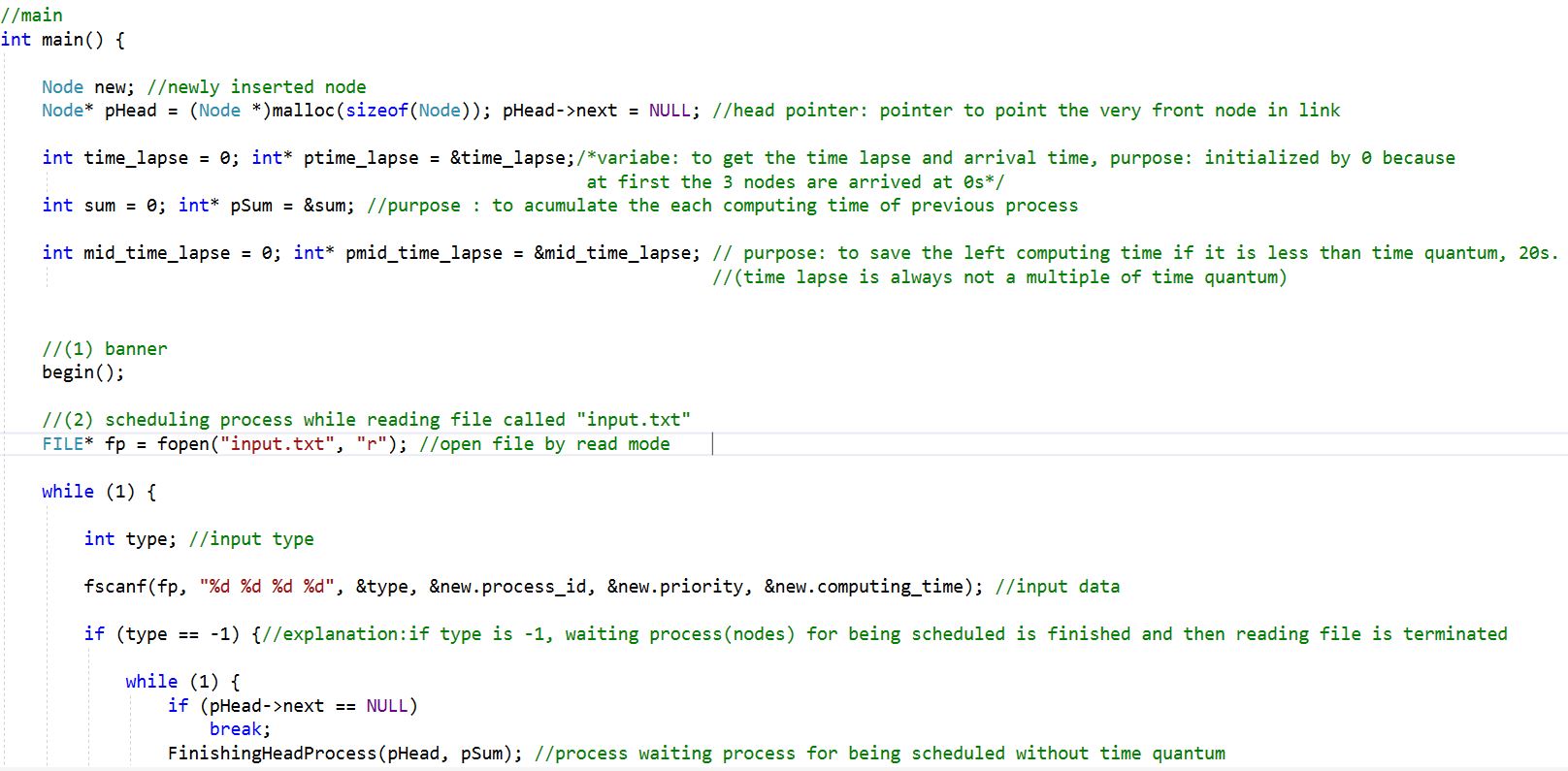
pHead: 맨 앞에 있는 프로세스(노드)를 가리키고 있는 포인터

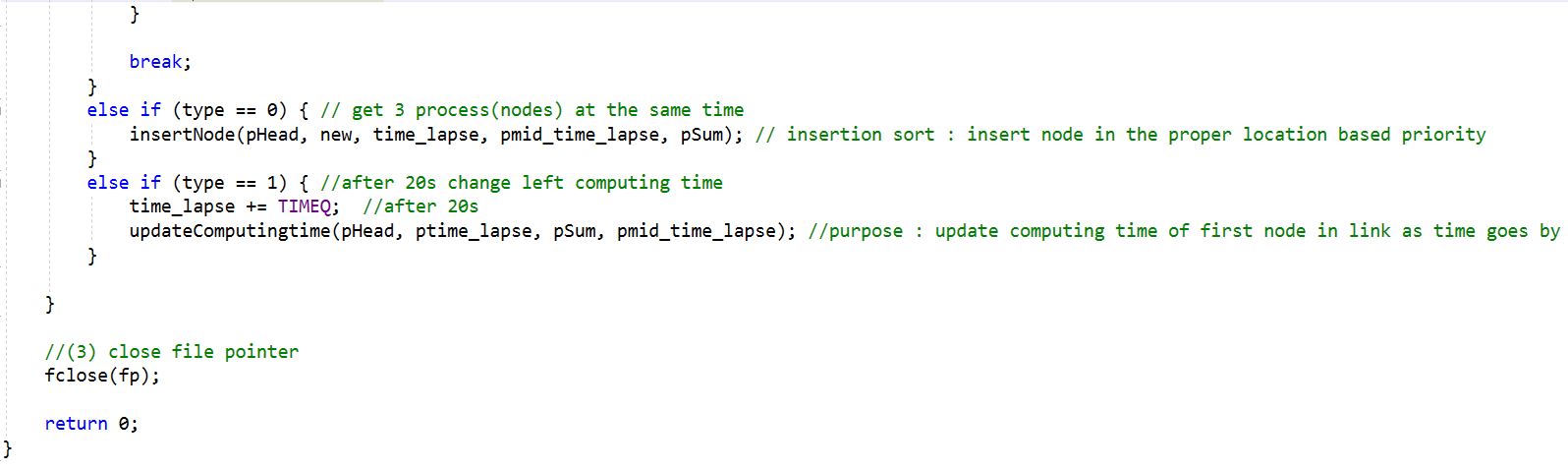
pSum: 스케줄링 완료시킬 프로세스(노드)를 동적할당 해제하기 전에 프로세스 정보를 출력해야 하는데 그 중 TAT를 계산하기 위해서는 누적 수행시간 sum을 가리키고 잇는 pSum 포인터가 필요

**③ 핵심 알고리즘 설명**

**(1) main문**

**(코드)**



****

* **main 안의 큰 실행 흐름 3가지**

**1단계:** 시작 banner를 출력

**2단계:** 파일을 읽으면서 스케줄링

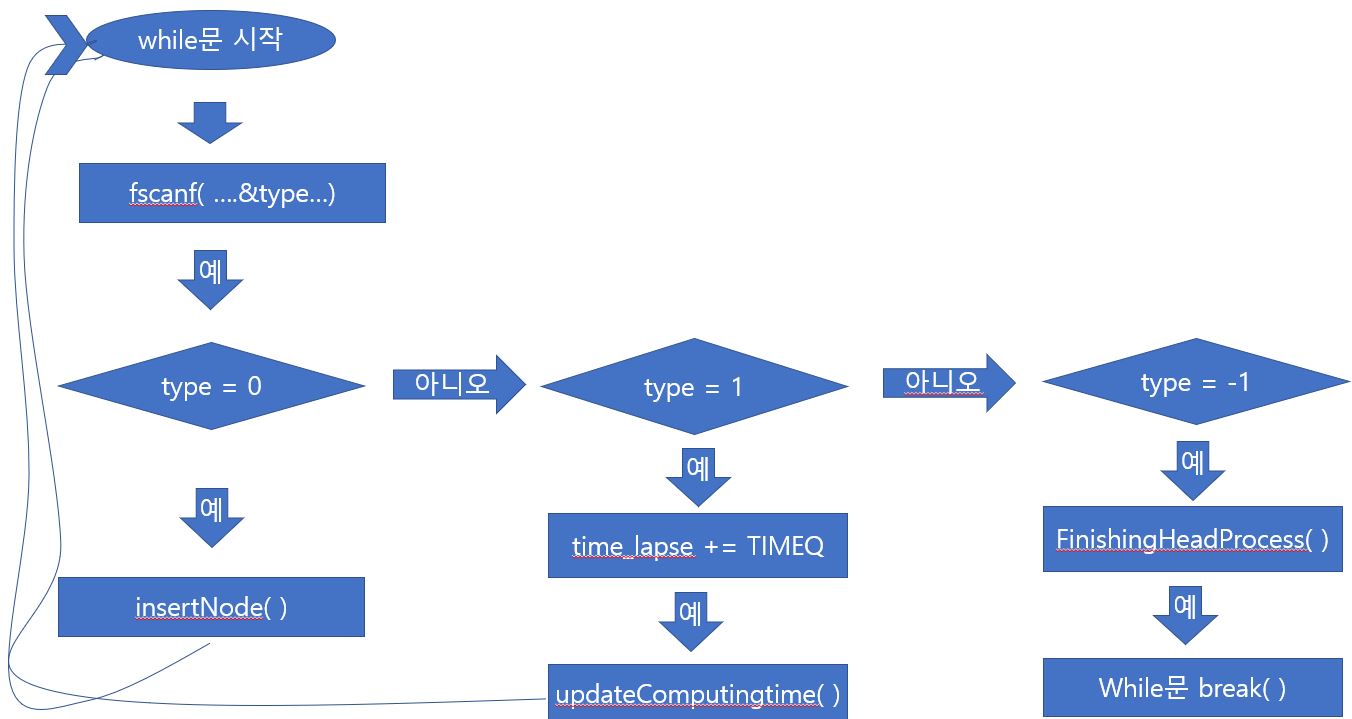
- type이 -1: 남아있는 대기중인 프로세스 모두 스케줄링 완료시키고 파일 읽기 종료

- type이 0: 프로세스들을 동시간에 우선순위를 기준으로 정렬하면서 받아들이기

- type이 1: 20초 시간 경과와 데이터 삽입 및 스케줄링

**3단계:** 파일 포인터 닫기

**(while문 알고리즘)**

****

* 우선 while문이 시작한다.
* fscanf( )함수를 통해서 inpu.txt에 있는 데이터를 노드로 읽어온다.
* 이 때 type이 0일 경우 insertNode( )함수를 호출해서 3개의 노드를 연결리스트에 삽입한다. 동시간대에 프로세스 3개가 들어오는 것이므로 arrival\_time은 동일 시간대로 설정하고 3개의 프로세스를 우선순위가 높은 순서대로 삽입 정렬한다. (처음 0초에 프로세스가 들어오는 것이라면 당연히 3개의 프로세스의 arrival\_time은 0초 초기화한다.)

-> insertNode( )작업 부분을 마치면 다시 while문을 시작한다.

* fscanf( )함수를 통해서 받아들인 type이 1일 경우는 20초 단위로 변화하는 시간 경과를 저장하는 time\_lapse 변수에 타임퀀텀만큼 매크로 상수를 더하여 증감시켜준다.

-> 시간을 경과시켰으면 updateComputing( )함수를 호출해서 시간이 경과한 만큼 프로세스의 남은 실행시간을 계산한다. 0초가 되면 스케줄링을 완료시킨다. 프로세스의 실행시간이 아직 남았으면 스케줄링을 완료시키지 않고 함수를 updateComputing( )함수를 빠져나온다.

-> 함수를 빠져나오면 다시 while문을 시작한다.

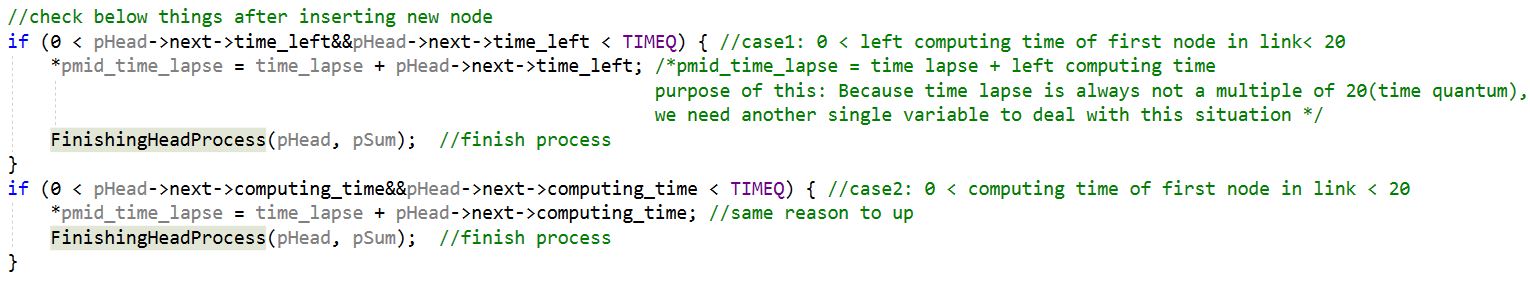
* fscanf( )함수를 통해서 받아들인 type이 -1일 경우 input.txt 파일에 있는 데이터를 다 읽고 마지막 -1을 만났다는 뜻이다. 하지만 파일 읽기를 종료시키기 전에 아직 한 번도 처리되지 않고 처리되기를 기다리는 프로세스들 즉, 시간이 경감되지 않아서 스케줄링 완료되기를 기다리고 있는 많은 프로세들이 있다. 따라서 이제 타임제한 없이 FinishingHeadProcess( )함수를 호출해서 프로세스들을 마저 스케줄링 완료시킨다.

-> pHead->next == NULL이라면 연결리스트에 프로세스가 아무것도 남아있지 않다는 것으로 더 이상 완료시킬 프로세스들이 없다는 것이다. 따라서 pHead->next == NULL일 경우 while문을 탈출한다.

* while문 종료

**(2) updateComputing()함수 부분**

**첫번째. insertNode( )함수 마지막 부분**



예

\*pmid\_time\_lapse = 0

아니오

0<time\_left<TIMEQQ

예

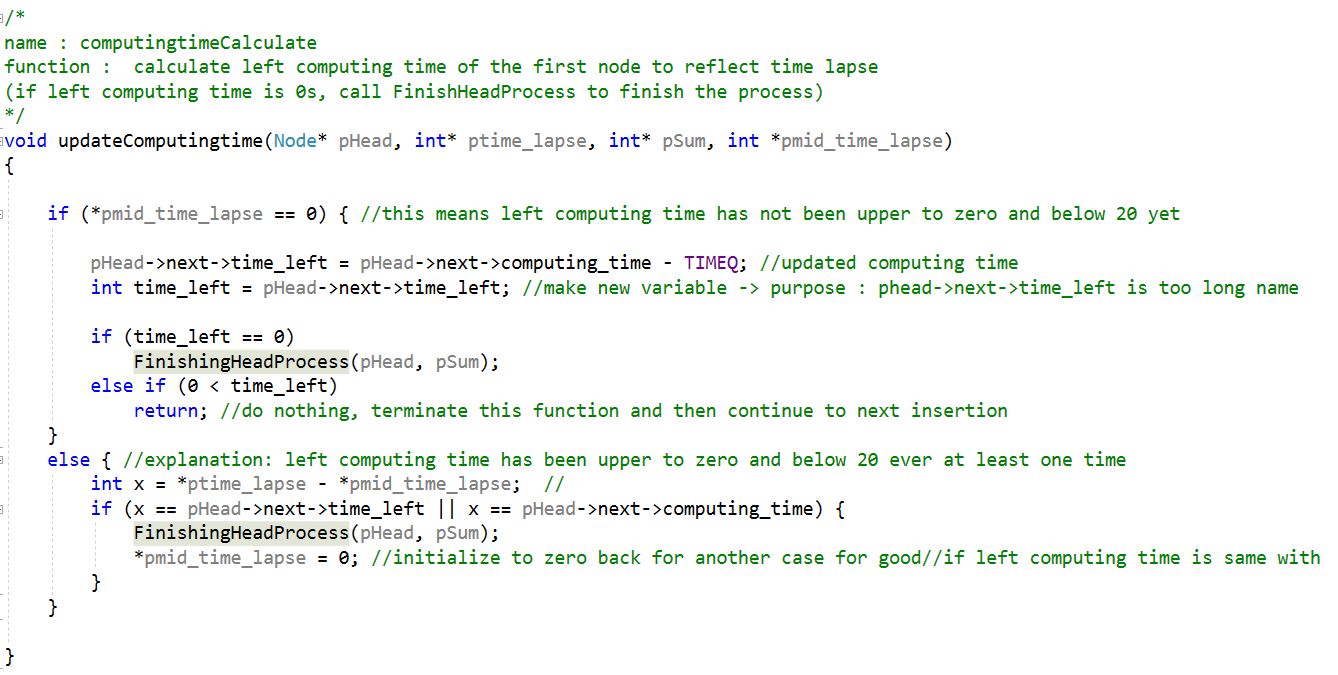
\*pmid\_time\_lapse = time\_lapse + pHead->next->time\_left

* main의 while문에서의 진행 흐름을 보면 insertNode( )함수가 먼저 진행되고 updateComputingtime( )함수가 진행되는 것을 알 수 있다.
* 코드를 짜면서 스케줄 하는 부분을 생각해보니 시간이 20초 단위로 경과는 경우 외의 상황을 생각해 봐야 했다.

예를 들어 현재 20초가 경과한 상태이고 time\_left가 10이여서 10초만 경과하면 맨 앞에 있는 프로세스를 완료시킬 수 있는 상황이다. 그럴 경우 시간 경과는 30초가 된다. 하지만 time\_lapse는 20초 단위로 시간이 경과하게끔 설정해 놔서 사용할 수 없다.

* 따라서 20초 단위로 시간이 경과하지 않고 스케줄링이 완료되는 특이 케이스에는 mid\_time\_lapse 변수에 30초와 같은 경과된 시간을 따로 저장하고 스케줄링을 완료시킨다.

**두번째. updateComputing( )함수 부분**



예

\*pmid\_time\_lapse = 0

\*pmid\_time\_lpase = 0

FinishingHeadProcess()

아니오

예

return;

아니오

time\_left = 0

예

FinishingHeadProcess()

* updateComputingtmie( )함수가 시작된다.
* 이전에 insertNode( )함수에서 20초 단위로 시간이 경과되지 않고 스케줄링이 완료되는 특이 케이스가 있어서 \*pmid\_time\_lapse가 다른 값으로 초기화되었으면 스케줄링 완료를 시키고 \*pmid\_time\_lapse 값은 다음 상황을 위해 0으로 초기화 시켜준다.

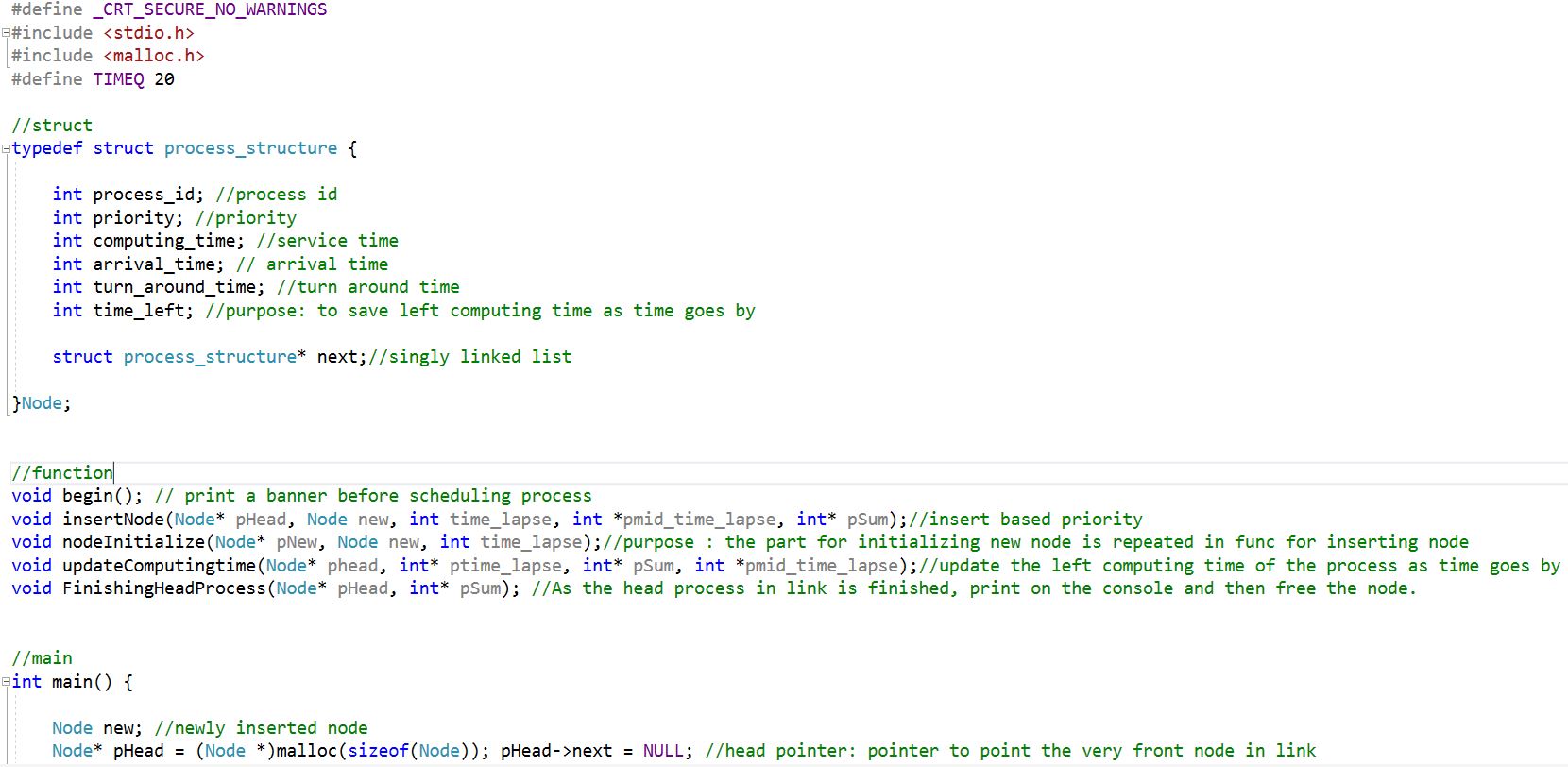
-> updateComputingtime( )함수를 종료시킨다.

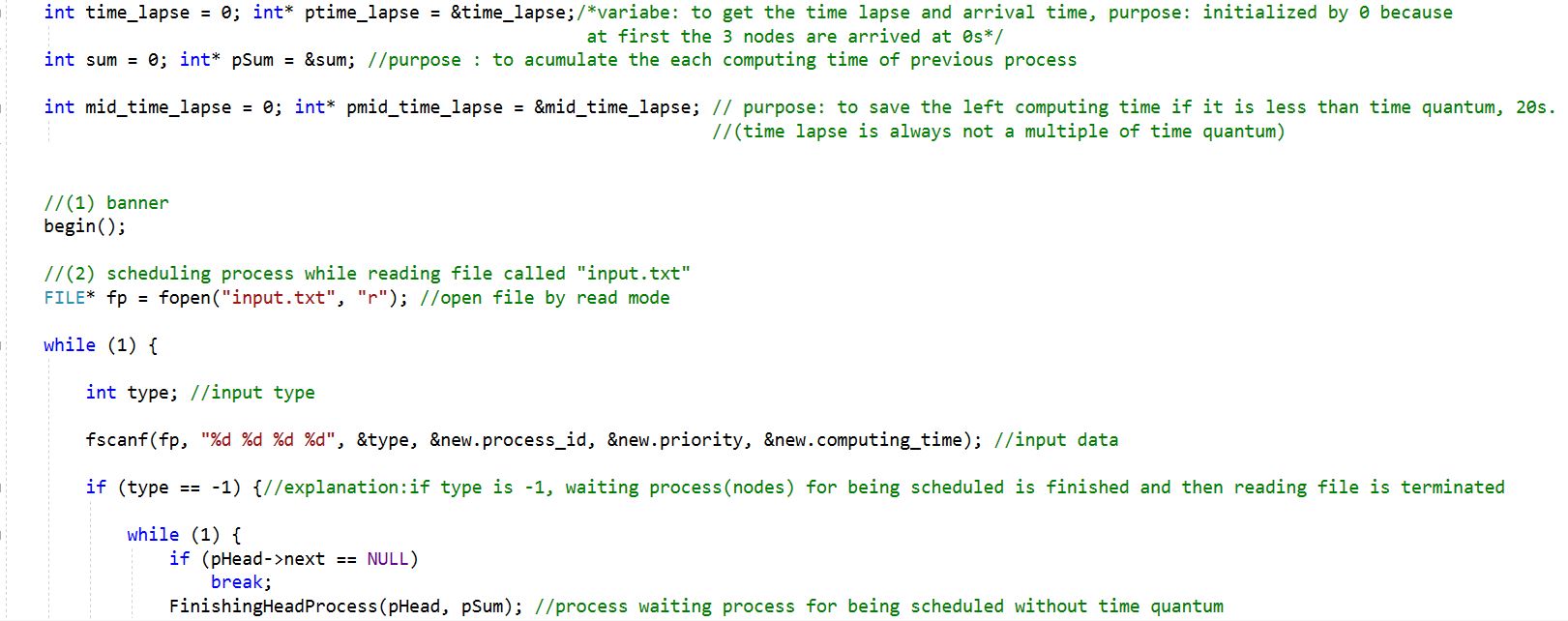
* insertNode( ) 함수에서 위와 같은 특이 케이스가 없었을 경우 원래대로 \*pmid\_time\_lapse 값은 0으로 초기화되어 있을 것이다. 이 때 프로세스의 남은 수행시간인 time\_left 값이 0인지 확인한다.

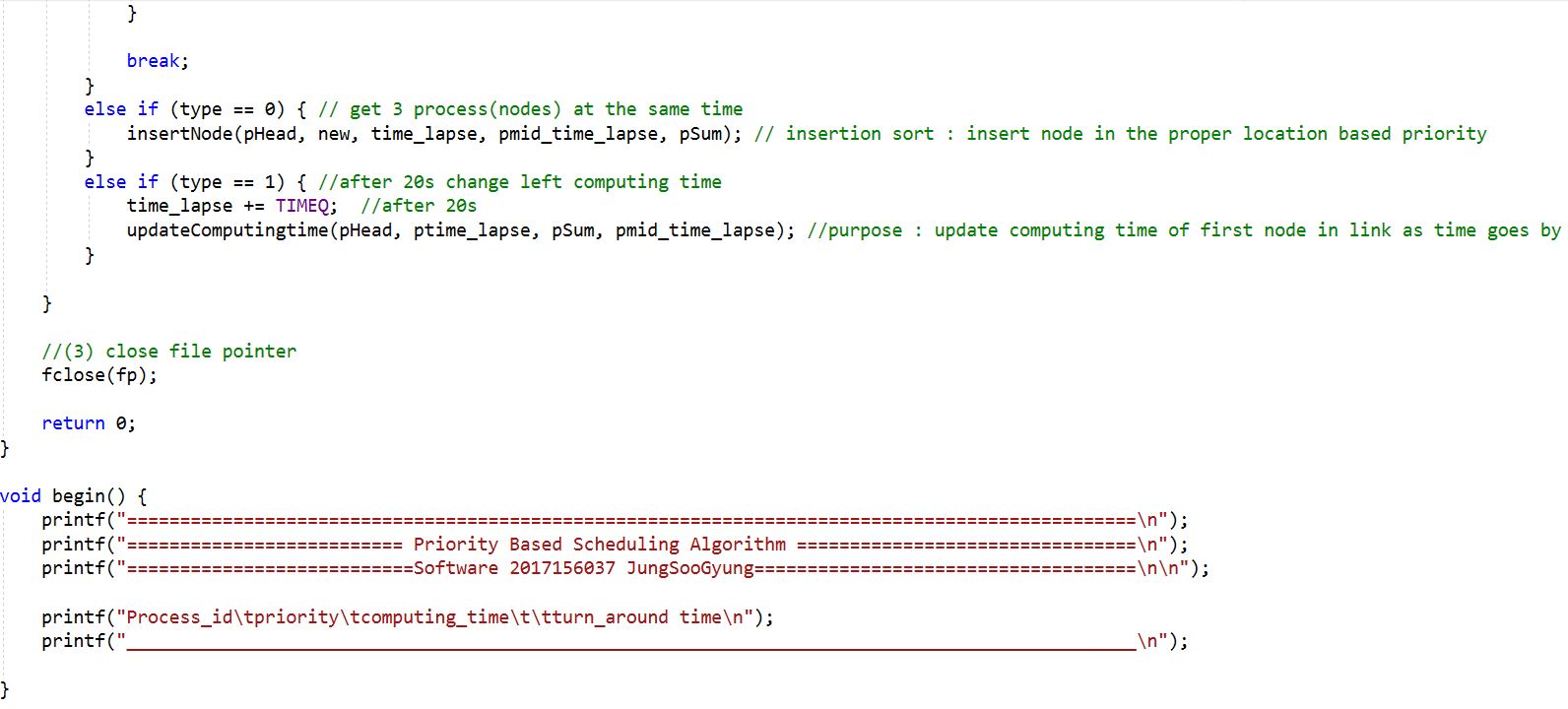
-> 0이라면 FinishingHeadProcess( ) 함수를 호출해서 스케줄링을 완료시킨다.

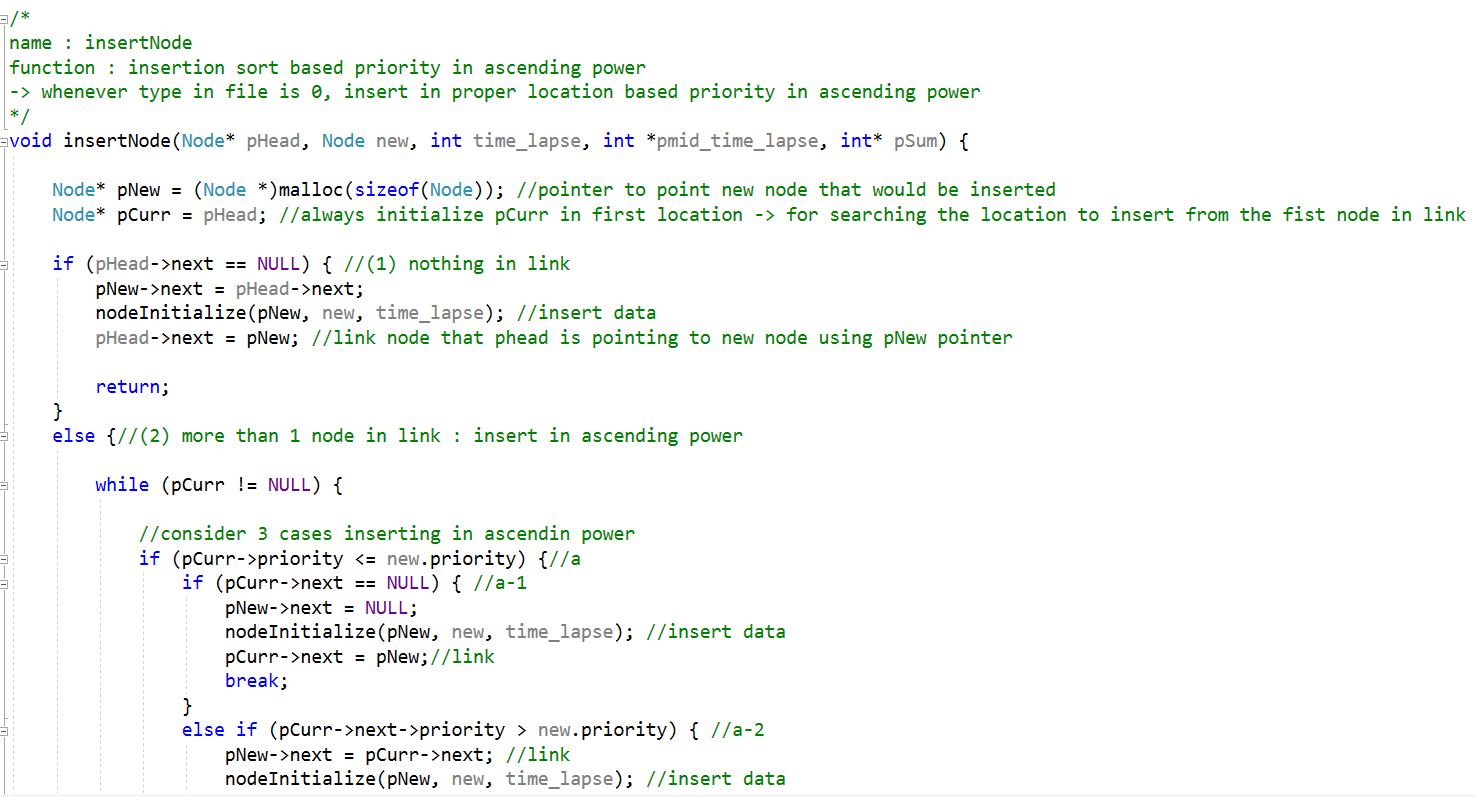
-> 0이 아닐 경우에는 아직 처리시간이 남아 있으므로 스케줄링 완료시키면 안 된다는 뜻이므로 return;시켜서 updateComputingtime( ) 함수를 종료시킨다..

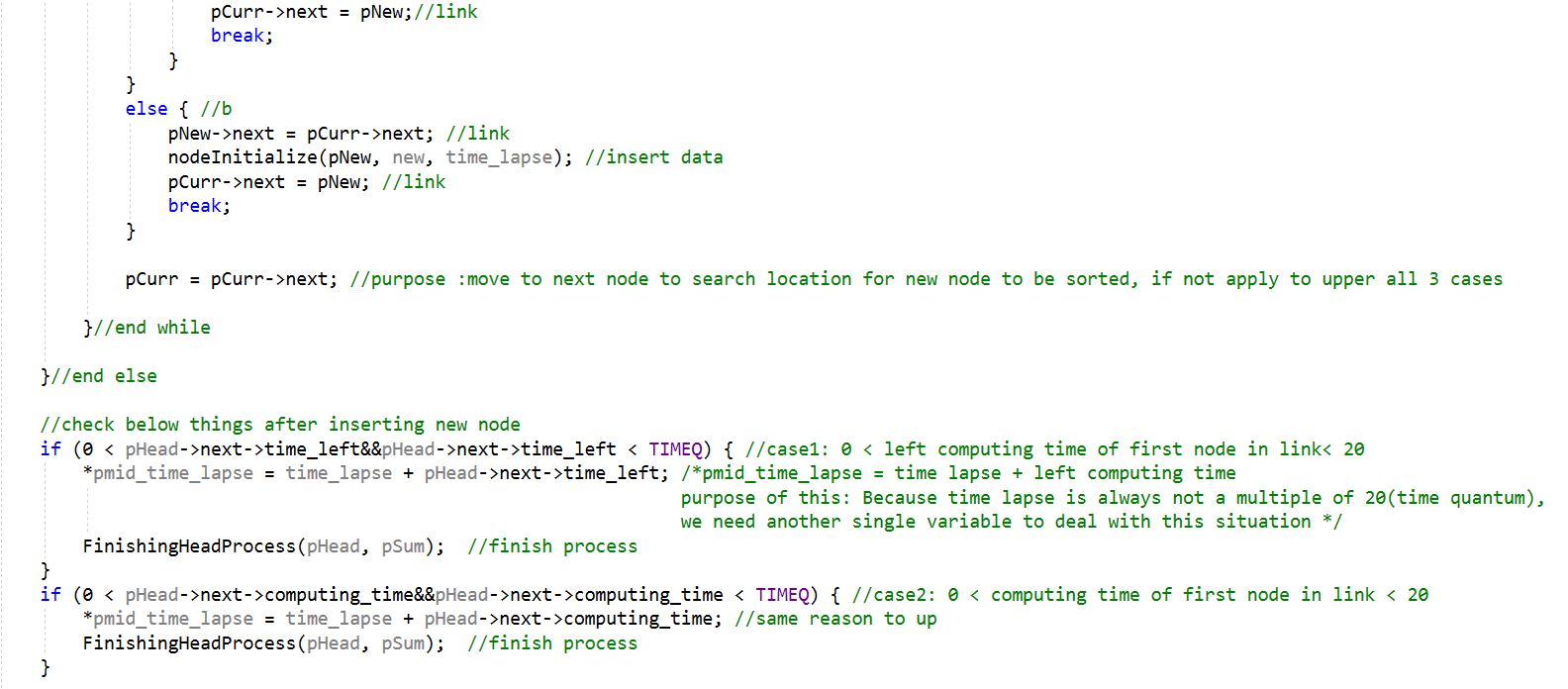
**d. 소스코드 부착**

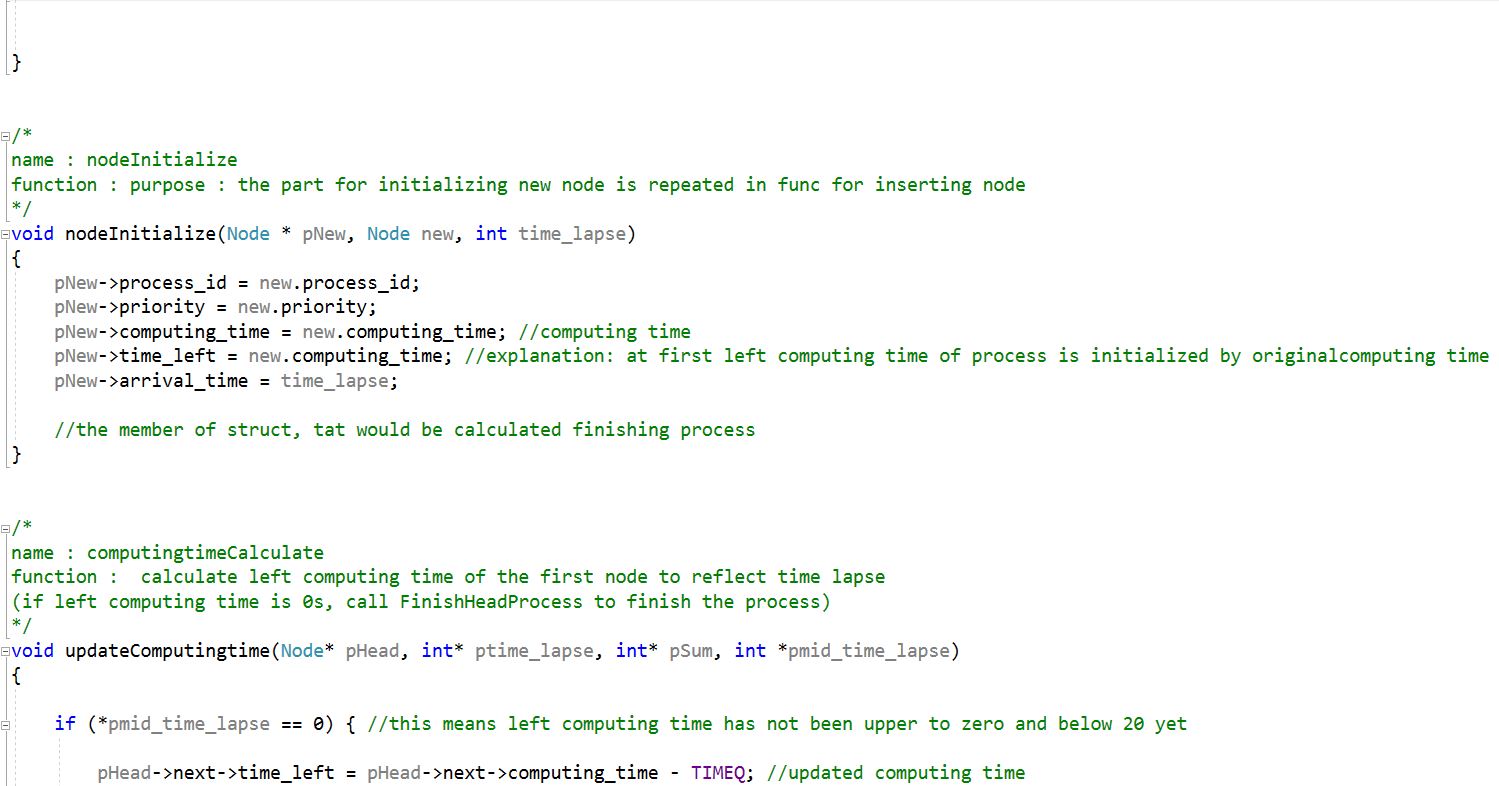
****

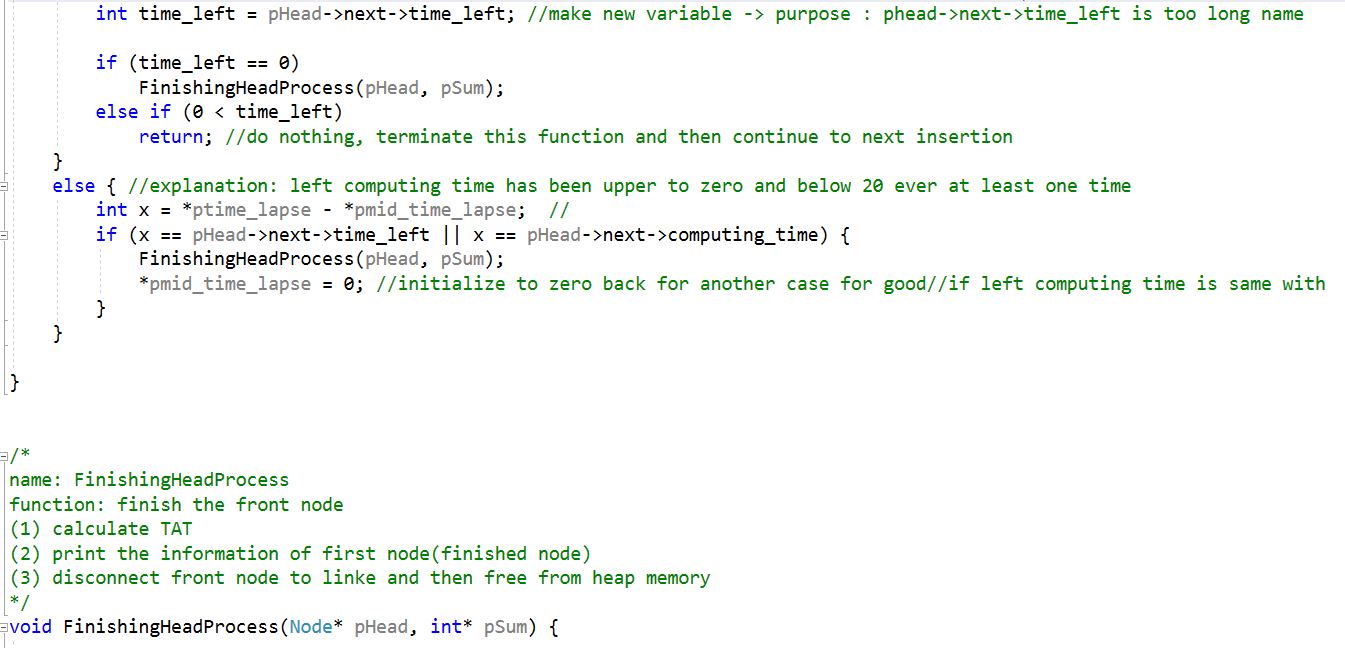
****

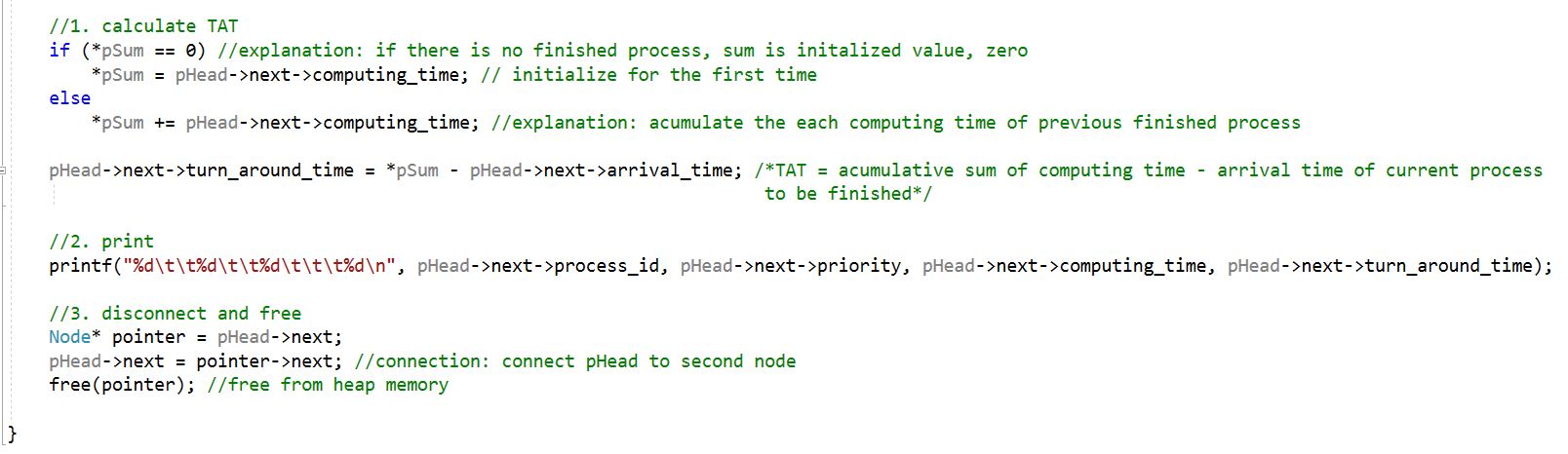
****

****

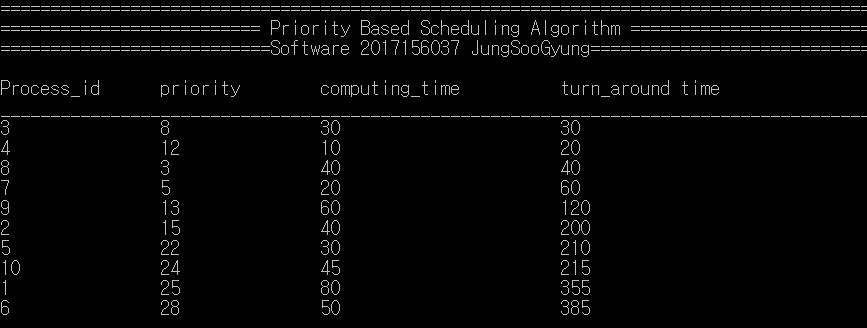
****

****

****

****

**e. 데모 결과 캡쳐 및 설명**



데모 결과 첫 번째 줄과 두 번째 줄만 살펴보면, 아래 그려 놓은 순서도와 같이 30초에 3번 프로세스(Process\_id가 3번인 프로세스)가 완료되고 40초에 4번 프로세스가 완료된다. 이때 3번 프로세스의 도착시간은 0초이고 수행시간은 30초이므로 3번 프로세스의 tat는 30 – 0 = 30으로 30초이다. 4번 프로세스의 도착시간은 20초이고 수행시간은 10초이므로 4번 프로세스의 tat는 이전에 완료시킨 프로세스의 수행시간 30초에 현재 프로세스의 수행시간 10초를 더하여 현 프로세스의 도착시간을 뺀 (30 + 10) - 20 = 20으로 20초가 된다.

3번 | 8 | 30초

2번 | 15 | 40초

3번 | 8 | 10초

t = 20초

①업데이트

t = 0초

1번 | 25 | 80초

2번 | 15 | 40초

1번 | 25 | 80초

t = 30초

삽입정렬

3번 완료

4번 | 12 | 0초

2번 | 15 | 40초

5번 | 22 | 30초

1번 | 25 | 80초

6번 | 28 | 50초

t = 40초

7번 | 5 | 20초

9번 | 13 | 60초

2번 | 15 | 40초

5번 | 22 | 30초

1번 | 25 | 80초

②삽입정렬

8번 | 3 | 40초

4번 | 12 | 10초

2번 | 15 | 40초

5번 | 22 | 30초

1번 | 25 | 80초

6번 | 28 | 50초

3번 | 8 | 0초

①업데이트

4번 완료

6번 | 28 | 50초

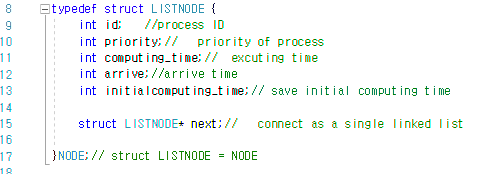
**5. Round-Robin 알고리즘(권한길)**

**a. 스케줄링 알고리즘 설명**

라운드 로빈 스케줄링은 시분할 시스템을 위해 설계된 선점형 스케줄링의 하나로서, 프로세스들 사이에 우선순위를 두지 않고, 순서대로 시간단위(Time Quantum)로 들어온 순서대로 CPU를 할당하는 방식의 CPU 스케줄링 알고리즘이다.

**b. 사용한 자료 구조 설명**

**① 구조체**

****

Process id, priority, computing time을 받을 수 있는 변수들을 정의.

* arrive, initialcomputing\_time 변수 추가.

-> arrive 변수에는 각 노드(프로세스) 마다 도착시간을 저장하는 변수.

-> initialcomputing\_time 변수에는 프로세스가 실행되면서 computing\_time 시간을 소비하는 구조이기 때문에 처음 들어왔던 실행시간을 보관하기 위해 쓰이는 변수.

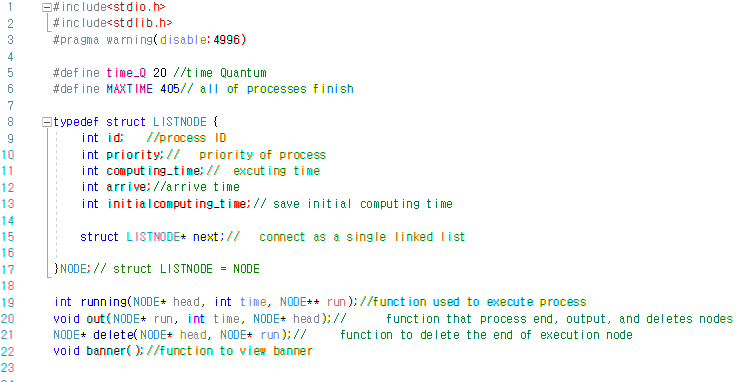
* 구조체 포인터 next 추가

->연결리스트를 구현하기 위해 다음 노드를 가리키는 구조체 포인터를 사용.

**② 단일 연결리스트**

->단일 연결리스트 자료구조를 이용하여 노드의 개념을 구현하였습니다. 삽입, 삭제, 이동, 교환할 수 있고 하나로 연결된 데이터를 포인터라는 개념으로 구현할 수 있다. 또한 4바이트의 헤더 포인터 변수를 가지고 많은 양의 데이터를 간접 접근할 수 있다는 큰 장점이 있다.

**c. 구현 방법 (함수 및 변수 설명)**



**① 주요 변수**

**Time\_Q** : define 해줌으로서 Time Quantum 을 자유롭게 바꿔서 실행 할 수 있도록 구현.

**MAXTIME** : 모든 프로세스가 끝나기에 충분한 시간 정의

**NUMBER\_NODE** : 들어올 노드의 수 정의

**Head :** 헤더 포인터

**Time :** out() 출력함수에 현재까지 흐른시간(runtime)을 넘겨 받는다.

**Run :** 현재 실행시키고 있는 프로세스를 가리키는 포인터 변수

**동작원리**: 포인터 변수는 head, old, run, p 포인터를 사용하고 있고 head는 첫번째 노드를 가리키고, old는 늘어나는 노드가 있을 때마다 맨 뒤 노드를 항상 가리키고 있는 포인터. Run 포인터는 현재 실행하는 노드(프로세스) 를 가리켜 실제로는 CPU할당하는 역할을. 마지막으로 p 는 새로운 노드를 만들어 input.txt 파일 에서 입력 값을 입력 받아 type 에 따라 연결하거나 재배열 하거나 입력을 마치고 실행을 진행하도록 해주는 포인터.

**② 사용한 함수**

**(1) int running(NODE \* head, int time, NODE \*\*run)**

- **함수 명**: running()

- **함수** **기능**: 이중 포인터를 사용하였으며 이를 통해 실행 하는 시간을 기록하는 cnt 변수의 return 과 실행을 마치고 다음 노드를 가리키기 위한 포인터 run 을 동시에 지키며 반환할 수 있다.

- **매개변수 의도**

**head**: 헤더 포인터의 주소 전달

**time**: 실행시간을 저장하고 반환하는 변수

**run**: return은 time을 해주고 있기 때문에 run 포인터가 실행되고 함수가 끝날 때 다시 그 포인터 변수를 저장하기 위한 이중 포인터.

**(2) void out(NODE \* run, int time, NODE \* head)**

-**함수 명** : out()

-**함수 기능** : 실행이 종료되고 출력을 하는 함수

- **매개변수 의도**

**run**: 실행중인 노드를 가리키고 있는 run 포인터의 주소를 넘겨받는다.

**head**: head 포인터의 주소를 받아 실행이 종료된 노드를 찾는데 쓰인다.

**time**: 현재 경과한 시간(runtime)을 넘겨줘서 turnaround 타임을 계산할 때 쓰인다.

**(3) NODE \* delete(NODE \* head, NODE \* run)**

-**함수 명** : delete

-**함수 기능** : 실행 시간이 끝나서 출력까지 한 후에 연결리스트에서 삭제하는 함수. 삭제된 노드를 제외하고 연결해야 하므로 삭제된 노드의 다음 노드의 주소 값을 넘겨준다.

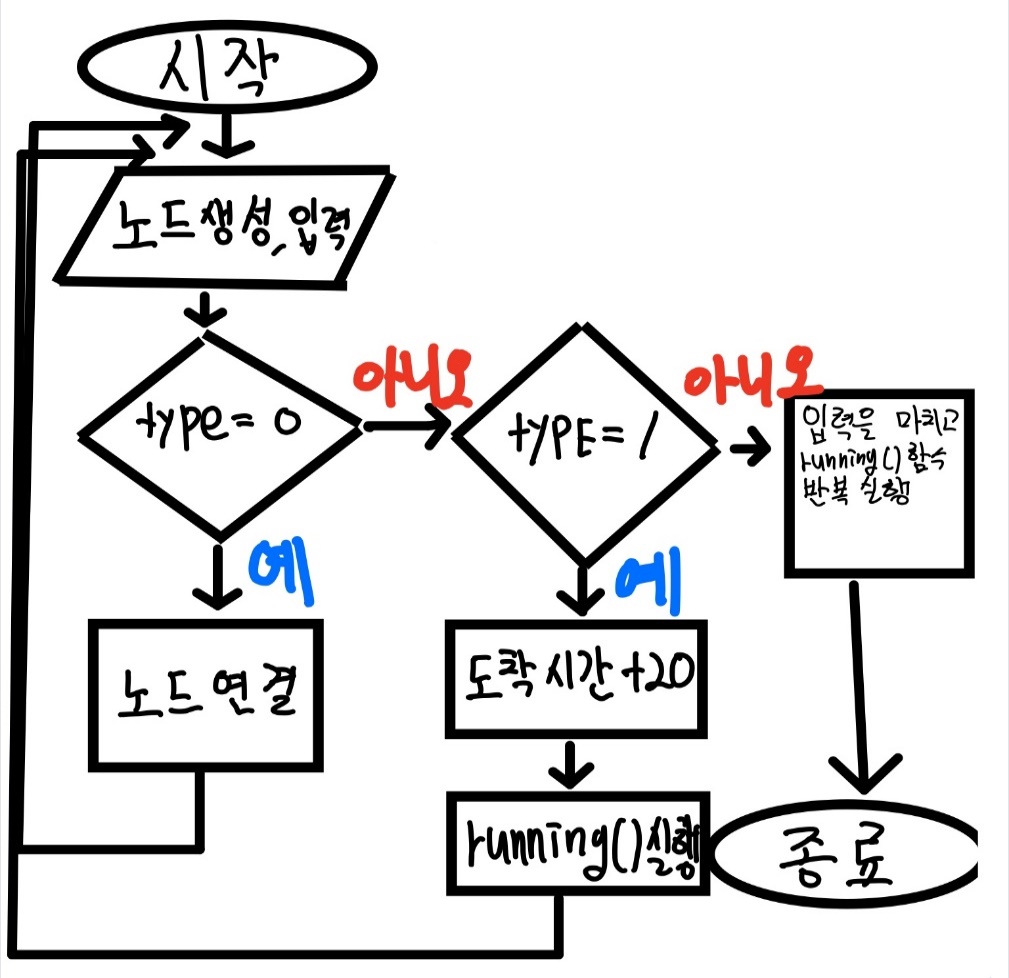
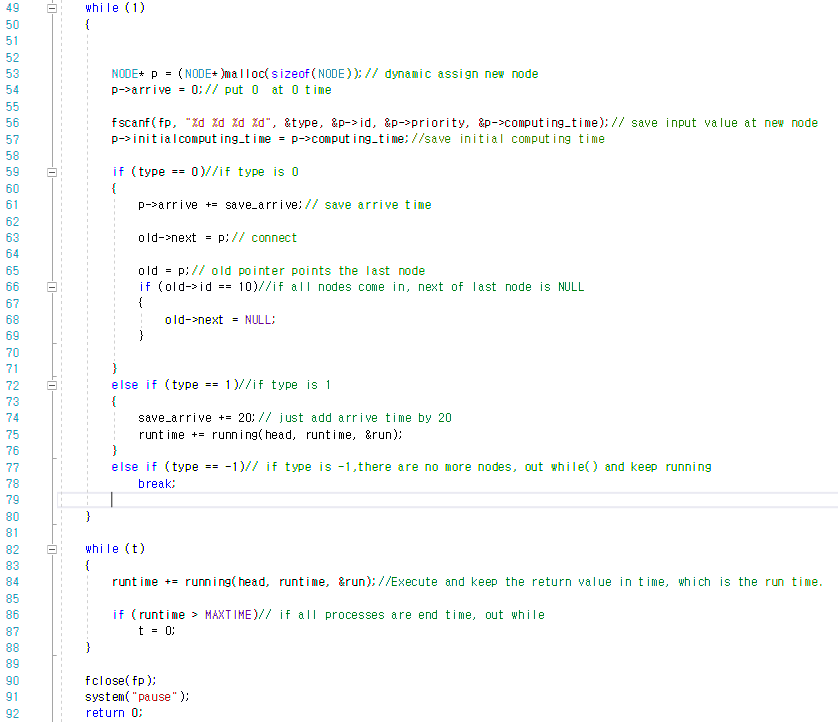
- **매개변수 의도**

**head**: 헤더 포인터의 주소 전달

**run**: 실행중에 computing\_time이 0이 된 노드를 가리킬 때 delete함수가 실행되는데 그 순간 노드를 가리키고 있는 포인터가 run이기 때문에 이 run 포인터의 함수를 넘겨주어 삭제하는데 쓰임.

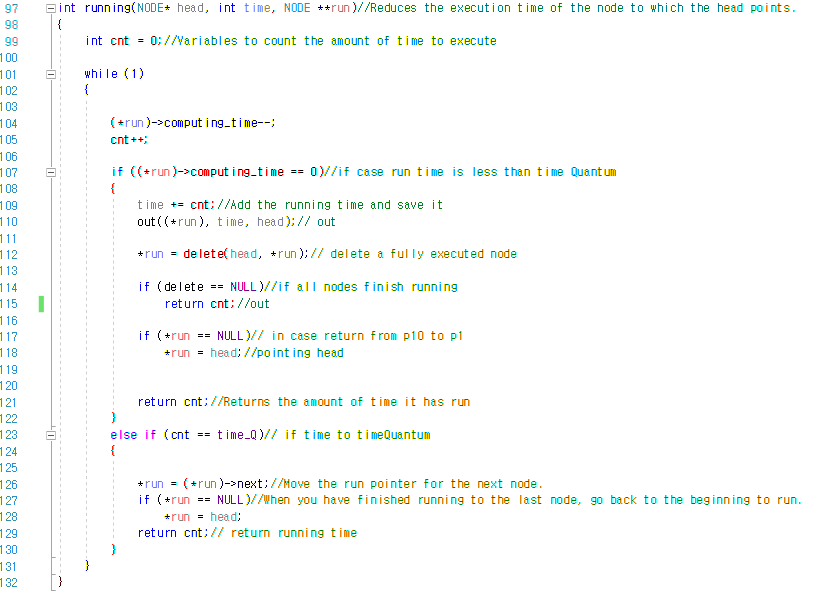
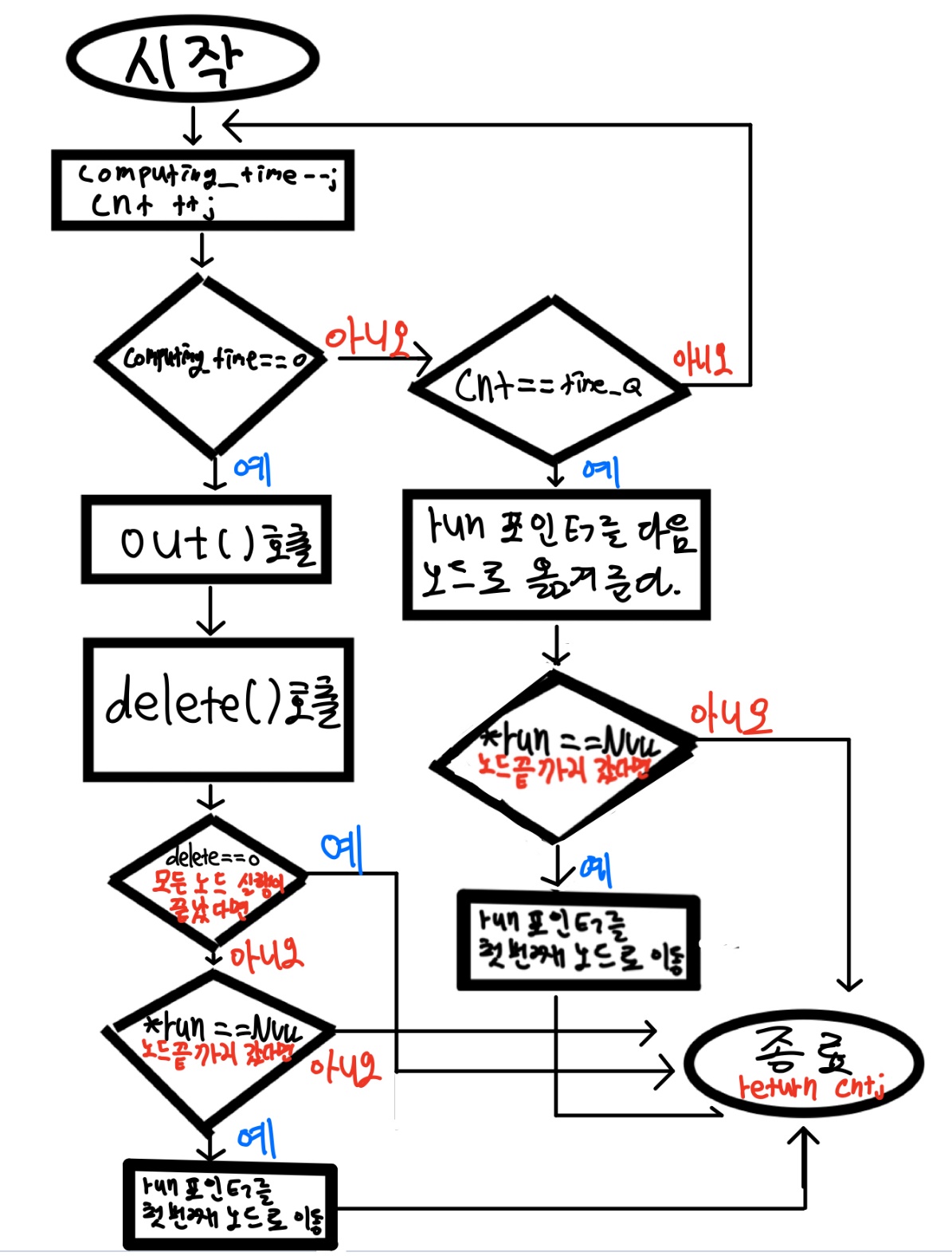
**핵심 알고리즘 설명**

**(1) main 함수의 while 문 부분**

****

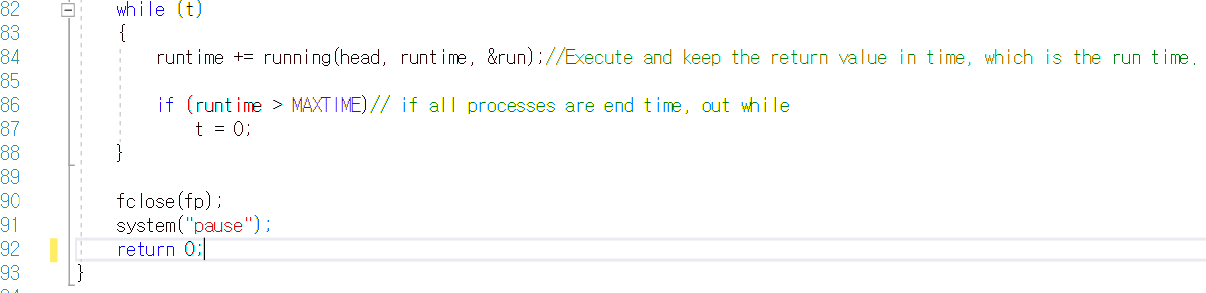
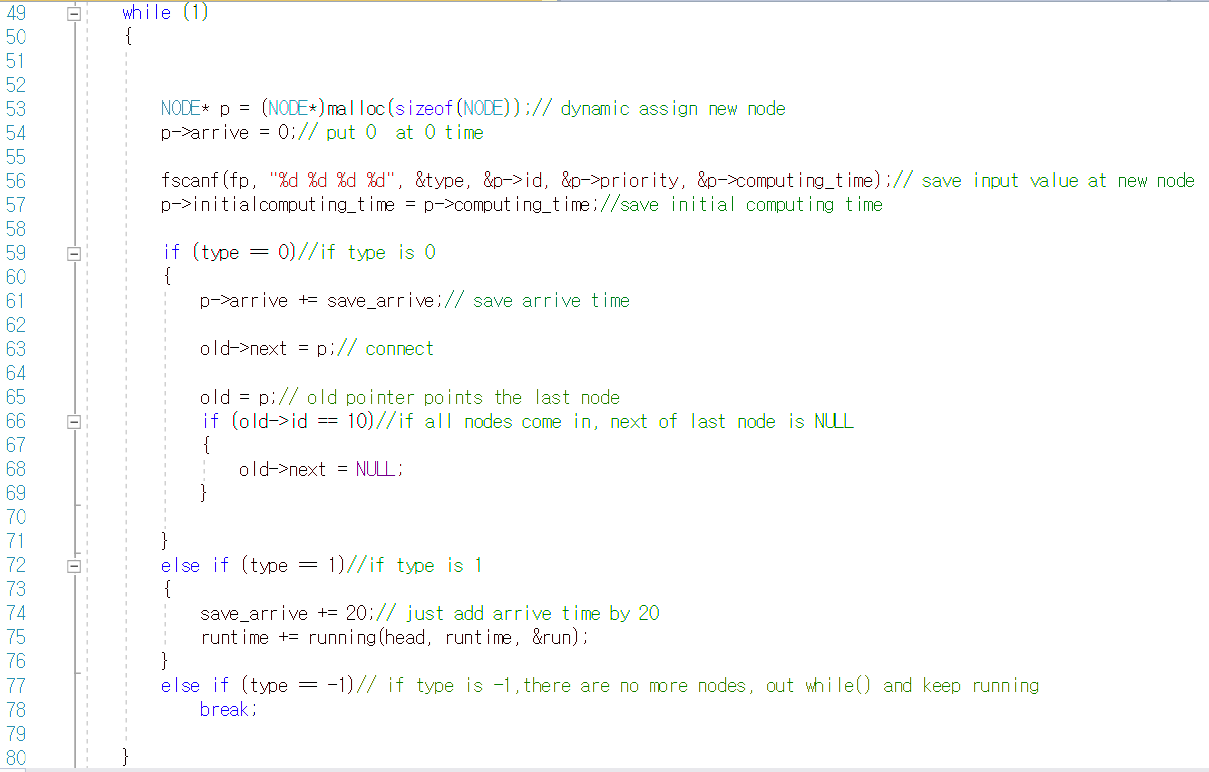
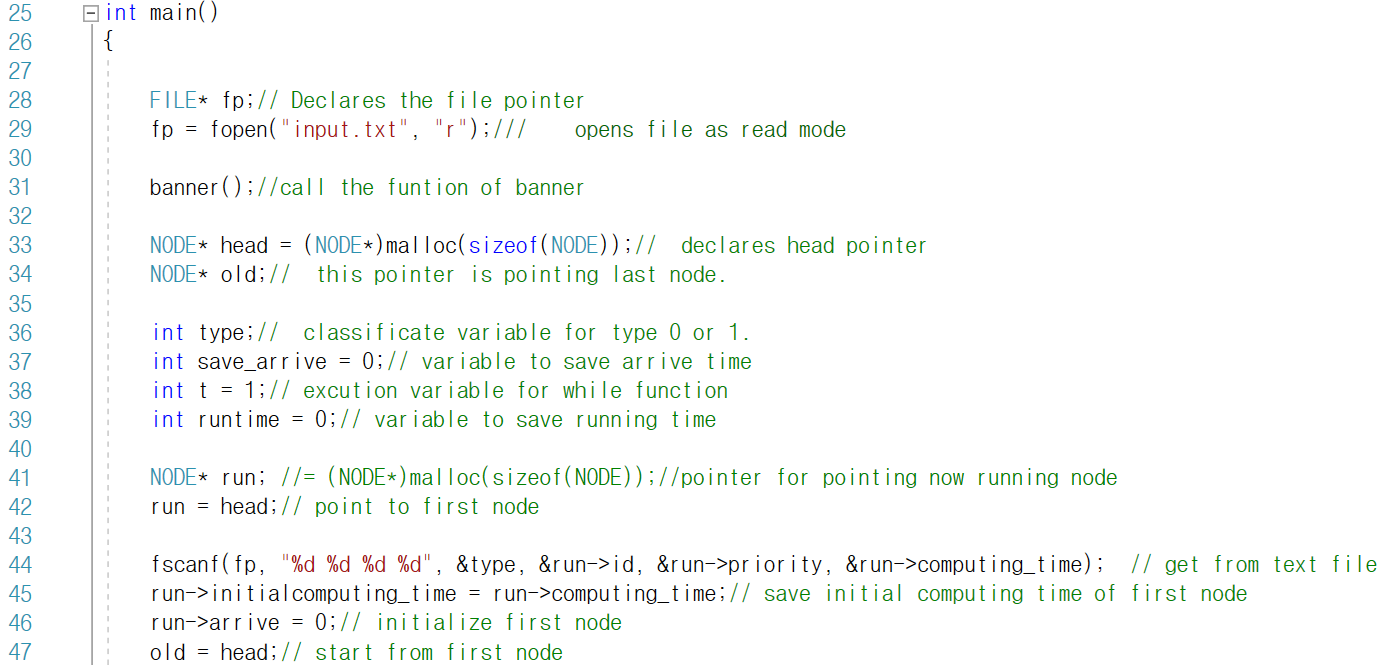
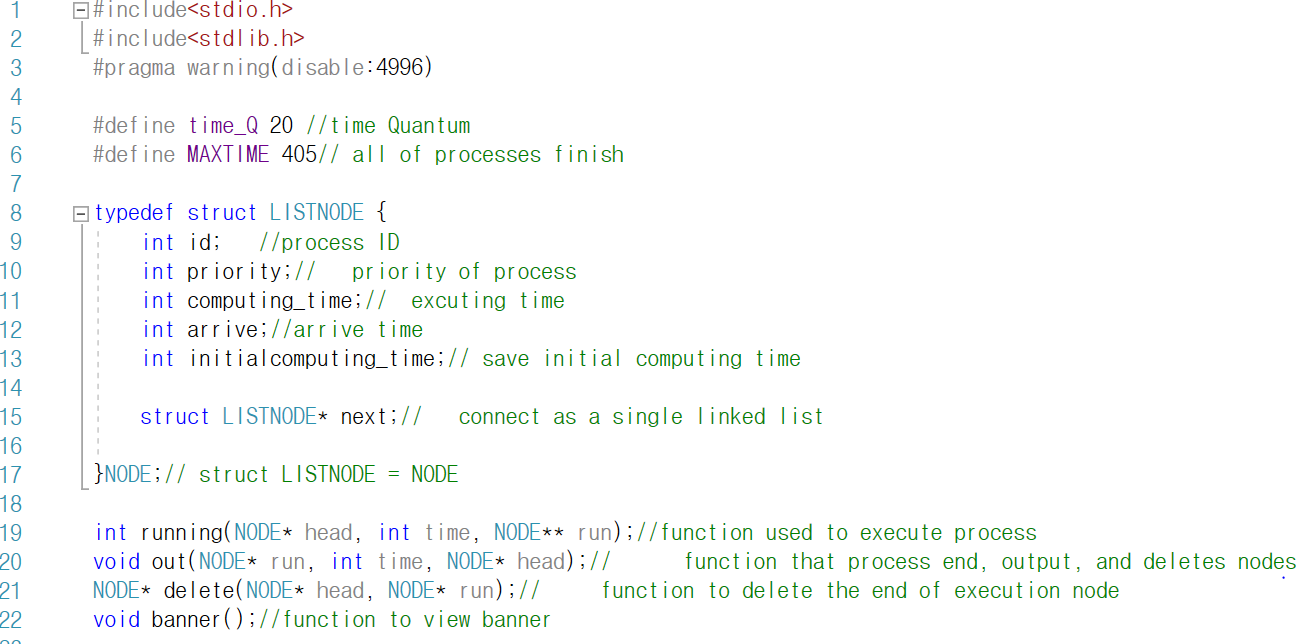
* 사진에 보이는 알고리즘은 main문 중 while 문이다. 먼저 포인터 p를 이용하여 새로운 노드를 만들고 그 노드에 값을 읽어와서 저장 후 type에 따라서 실행한다.
* Type 이 0이 들어오면 노드를 연결 시켜주고 NULL을 넣어준다
* Type 이 1이 들어올 때 마다 save\_arrive 변수에 20씩 더해서 저장하므로 각각의 노드에 알맞은 도착시간을 저장한다. 또한 20초가 지난 시점이므로 running()함수를 호출해 프로세스에게 시간을 할당한다.
* Type -1 이 들어오면 입력과 노드연결이 끝났으므로 공평하게 cpu를 timeQunantum만큼 할당해주면 되는데 run 함수를 옮겨 가면서 running()함수를 실행시켜 구현한다.

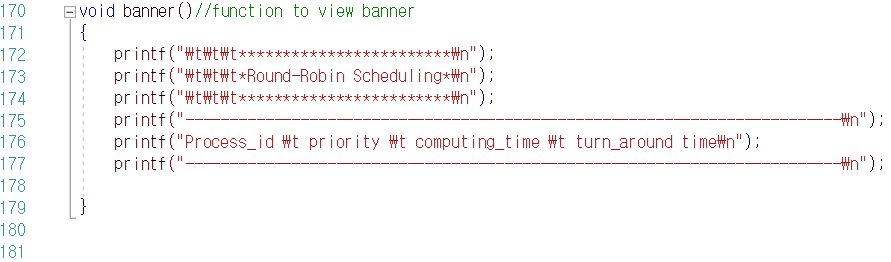
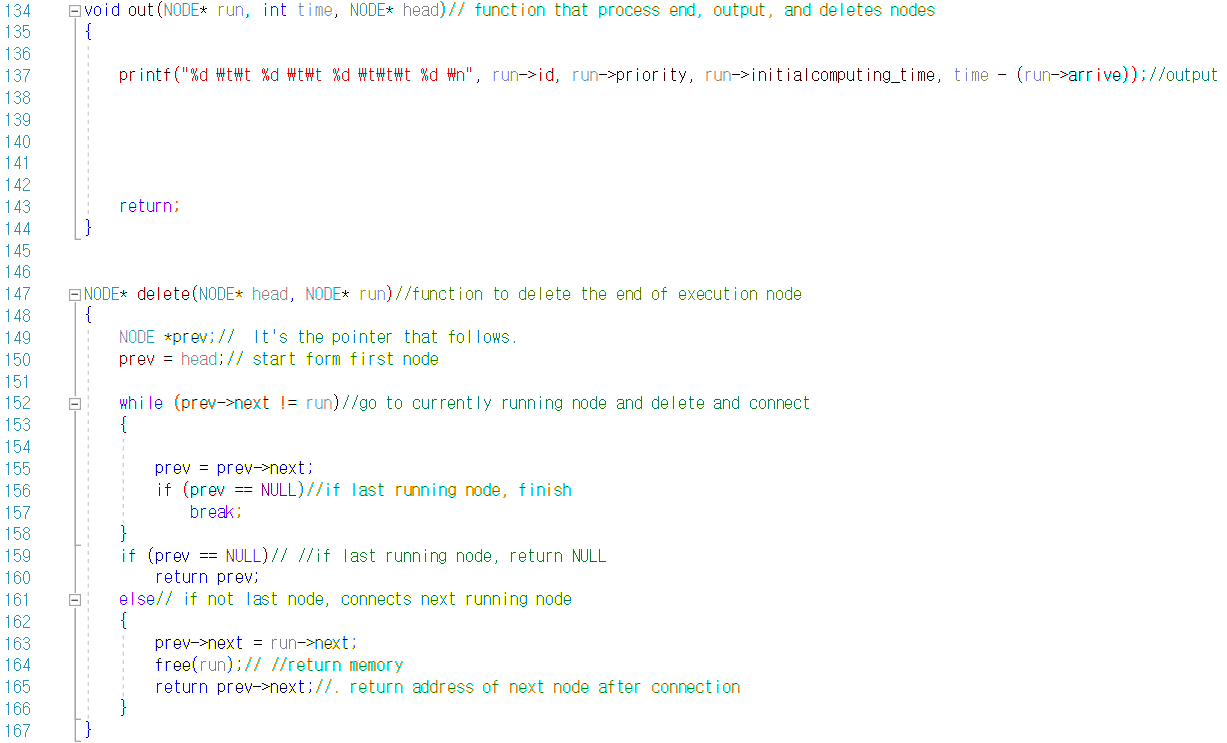
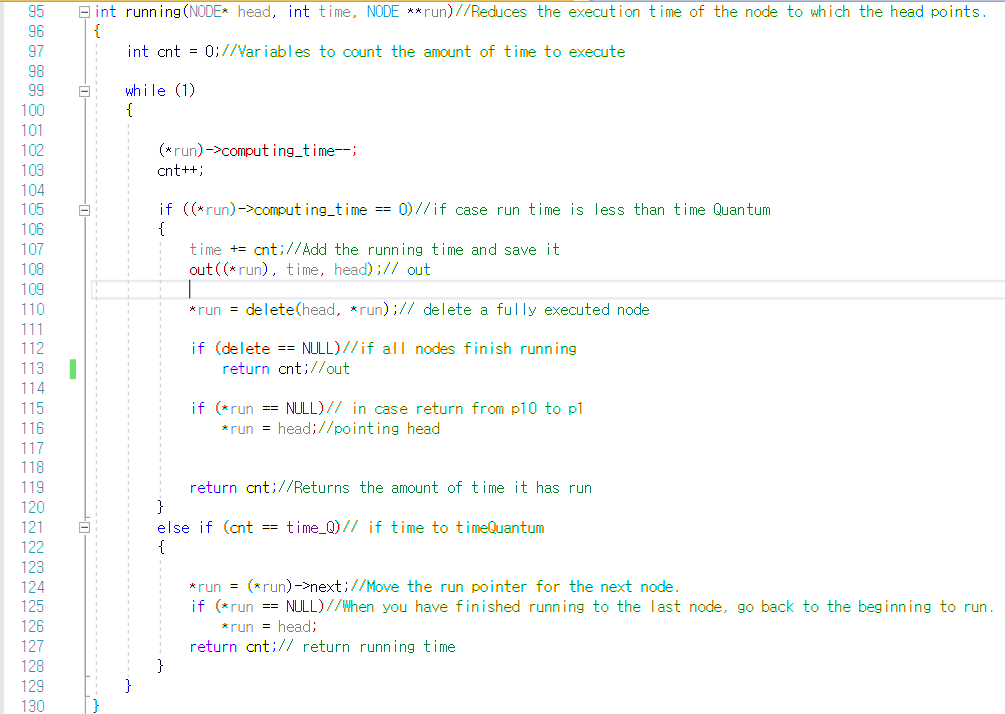
**(2) running()함수 (주요 코드 부분)**

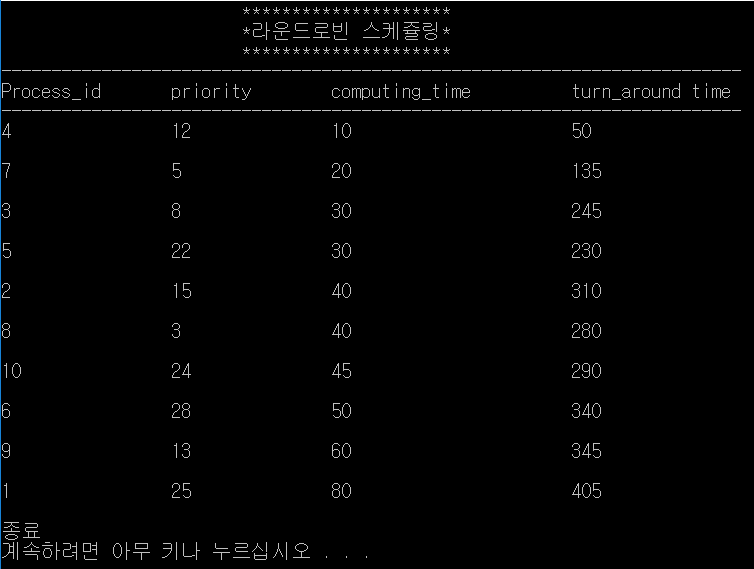


* 이 함수는 cpu가 할당되어 실제로 실행시킬 때 호출하는 함수.
* 핵심 알고리즘 코드 부분이다. cpu가 할당되어 실제로 실행시킬 때 호출하는 함수.
* 파라미터는 head포인터, runtime 값을 받는 time 변수, 실제 실행시킬 때 사용하는 run포인터의 주소를 저장하는 이중 포인터이다.
* 먼저 count 변수를 선언해주고 while 문에 들어 가면 현재 run이 가리키고 있는 노드의 computing\_time 을 1씩 깎고, 동시에 count변수의 값을 1씩 증가시킵니다.
* 그럼 실행 중에 2가지 경우의 수를 예측할 수 있는데, 한가지 경우의 수는 노드의 실행시간이 끝나는 경우와, 타임퀀텀 단위 시간을 모두 사용하였을 경우이다.
* 전자에서는 실행된 시간 즉, 프로세스가 cpu를 할당한 시간을 time 변수에 더하면서 저장. 그리고 프로세스실행이 끝났으므로 out 함수를 이용하여 출력을 해주고 delete 함수를 사용하여 노드를 삭제와 연결을 끝낸다.
* if 안의 첫번째 if 문은 만약 모든 노드를 삭제했을 경우, 즉 모든 프로세스가 끝났을 경우에 delete 함수는 NULL을 반환하게 구현하였다. 이 경우 충분한 시간인 MAXTIME 을 반환하여 프로그램을 종료하는 장치이고 두번째 if 문은 만약 run이 마지막 노드까지 갔을 때 다시 첫번째 노드로 돌아가서 반복해 실행을 시키도록 run포인터를 옮겨주는 장치이다.
* 두번째 경우에서는 타임퀀텀이 다 소비한 경우의 수인데 이 경우는 다음 노드를 가리키도록 run포인터를 이동해 줌.
* 마지막으로 이중 포인터를 사용한 이유는 return 값으로 cnt 값을 반환 하여서 메인에서 누적 시키고 있기 때문에 삭제하고 연결하는 과정에서 run 포인터를 잃지 않기 위해서 이중 포인터를 사용함.

**d. 소스코드 부착**

****

****

**e. 데모 결과 캡쳐 및 설명**

현재 TimeQuantum이 20으로 정의되었기 때문에 컴퓨팅 타임이 10이었던 4번째 프로세스가 가장 먼저 실행을 종료했음을 알 수 있다. TimeQuantum의 크기에 따라 먼저 종료되어 출력되는 프로세스의 순서는 변할 수 있고 turn\_around time 또한 변할 수 있다.

**5. 분석 및 결론**

**a. 사용한 스케줄링 알고리즘의 정규화 된 반환시간 비교 분석**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **FCFS scheduling** | **Round-Robin scheduling** | **Priority-Based scheduling** |
| **Normalized Turnaround Time** | **61.0** | **58.6** | **37.9** |
| **Normalized average Turnaround Time** | **6.1** | **5.86** | **3.79** |

**b. 결론 및 총평**

**① 3개의 CPU 스케줄링 알고리즘 비교 분석 및 결론**

* Normalized average turn\_around time 값 비교

Priority-Based scheduling(3.79) < Round-Robin scheduling(5.86) < FCFS scheduling(6.1)

* 결론 : Normalized average turn\_around time 값을 비교해 본 결과 우선순위 기반 스케줄링 방식이 가장 값이 작다. 따라서 위의 3개의 스케줄링 방식 중 가장 효율적인 스케줄링 알고리즘이라고 할 수 있다.
* 프로세스에 미치는 영향 비교 분석

FCFS: 짧은 프로세스에 불리하며 입출력 중심 프로세스에 불리함

Round Robin: 모든 프로세스에 공정함

우선순위 기반: 우선순위가 가장 높은 프로세스에게 상대적으로 유리함

* 기아상태 비교 분석

FCFS: 가능성 없음 //비선점

Round Robin: 가능성 없음 //선점

우선순위 기반: 가능성 있음 //선점

**② 프로젝트를 마친 소감 및 총평**

**김지은(FCFS 알고리즘)**

. CPU 스케줄링 방식을 잘 이해하고 있다고 생각했기에 C언어, 자료구조에서 내가 부족했던 부분만 다시 공부하면 수월하게 짤 수 있을 것이라고 생각했지만 막상 과제를 시작하니 그 과정들이 쉽지만은 않았다. 일단 인터넷과 책을 통해 알고리즘 구현에 필요한 자료구조와 C언어 문법들을 다시 공부한 후 그림을 그려가며 프로세스의 처리과정들을 차근차근 정리했는데 특히 그림을 그려가며 했던 것이 구현하는데에 아주 큰 도움이 된 것 같다. 2주동안 열심히 노력한 끝에 구현을 마무리할 수 있었고 굉장한 성취감을 느낄 수 있었다. 그리고 팀원들 모두가 틈틈이 이룸에 모여 한 달여간의 시간동안 프로젝트에 대한 구체적인 계획을 세우고, 그 계획에 맞추어 코드 구현이나 보고서 작성, PPT 제작 등을 하여 여유 있게 프로젝트를 마무리할 수 있었는데 길다면 길다고 할 수 있는 이 시간동안 각자의 역할에 대해 성실히 한 팀원들에게 너무 고맙다.

**정수경((Priority-Based 알고리즘)**

수업시간에 교수님이 설명하셨던 개념을 이번 팀 프로젝트를 하면서 몇일동안 고민하고 알고리즘을 직접 구현해보면서 이론적으로 배운 개념을 더 내 것으로 만들 수 있었던 것 같다. 또한 원래 이 프로젝트를 시작하기 전에 C언어와 자료구조가 매우 약하다는 생각이 들어 항상 C언어로 어떤 프로젝트를 맡는 게 두려웠었다. 하지만 이번에 알고리즘을 구현하면서 잘 안 되더라도 끝까지 포기하지 않고 코딩에 필요한 문법을 직접 찾아보고 머리 속에 채워 나가면서 내게 너무 힘들어 보였던 프로젝트를 마침내 완수할 수 있었다. 마지막으로 우리 팀원들 모두 누구 한 명 빠짐없이 성실하게 각자 맡은 일을 해오고 열심히 참여한 점이 이번 팀프로젝트를 수월하게 진행하는데 한 몫 한 것 같다. 팀과제가 나오자 마자 이룸에서 만나 각자 언제부터 언제까지 구현하고 언제까지 보고서를 작성할지 미리 계획을 세우고 모두다 그 날짜를 지켜내서 우리 팀은 다른 팀들과 다르게 여유롭게 팀프로젝트를 마칠 수 있었다.

**권한길(Round-Robin 알고리즘)**

이 과제가 주어졌을 때 처음에는 배운 방식대로 짜면 충분히 가능하다고 생각하고 시작했지만 직접 짜보면서 안되는 부분이 너무 많아 애먹었던 것 같습니다. 그래서 이중 포인터, 연결리스트, 함수, fscanf() 등 정확한 문법과 활용법을 알기 위해 유투브 구글 등에서 정보를 얻고 공부해서 많은 고민과 연구 끝에 완성시켰습니다. 스스로 하나의 스케줄링 알고리즘을 완성시켰을 때의 성취감은 기억에 오래 남을 것 같습니다. 완성한 결과로 팀프로젝트를 하면서 여유가 없음에도 꾸준히 모여서 고민하고 조언하고 했던 과정들 속에서도 많이 배웠습니다. 서로 존중하며 누구 하나 게을리 하지 않으며 프로젝트를 진행한 팀원들에게 정말 고맙습니다. 종합해보면 개인적으로나 팀적으로나 많은 어려움이 있었지만 그만큼 더 많은 것을 얻었다고 느꼈습니다. 결과적으로 교수님께서 원하셨던 개인의 실력향상이 된 것 같아 뿌듯합니다.