동국대학교
dongguk university

실습: Week 6

Data Structures

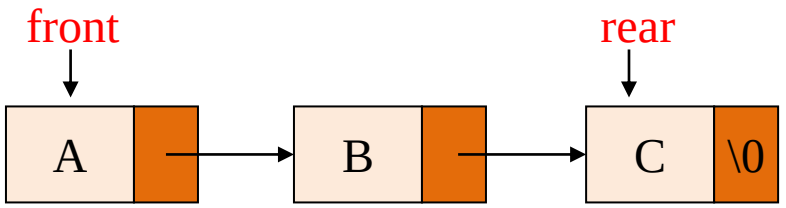
Contents

- Linked Queue
  - Linked Queue 구현
- Queue simulation
  - Linked Queue 를 이용한 프린터 시뮬레이션
- 실습
  - 실습 6-1. Queue (linked list) 구현
  - 실습 6-2. Queue simulation 수행

Linked Queues

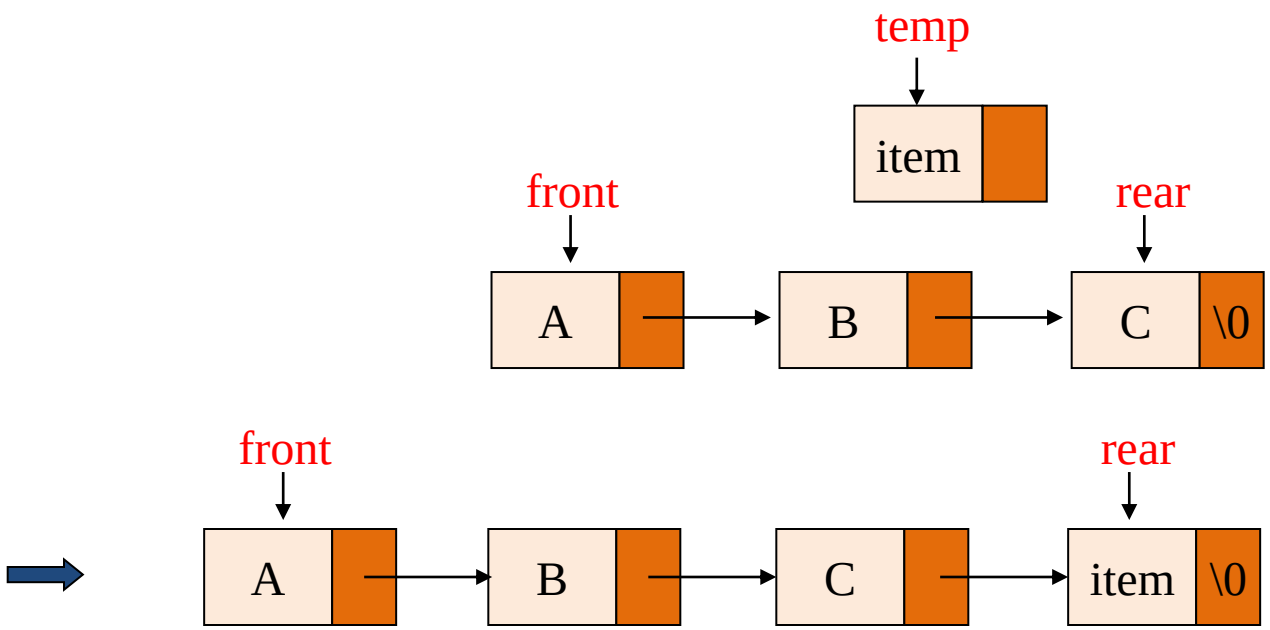
- Linked Queue

```
typedef struct queue *queue_pointer;
typedef struct queue{
        element         item;
        queue_pointer   link;
}
queue_pointer front, rear;
```



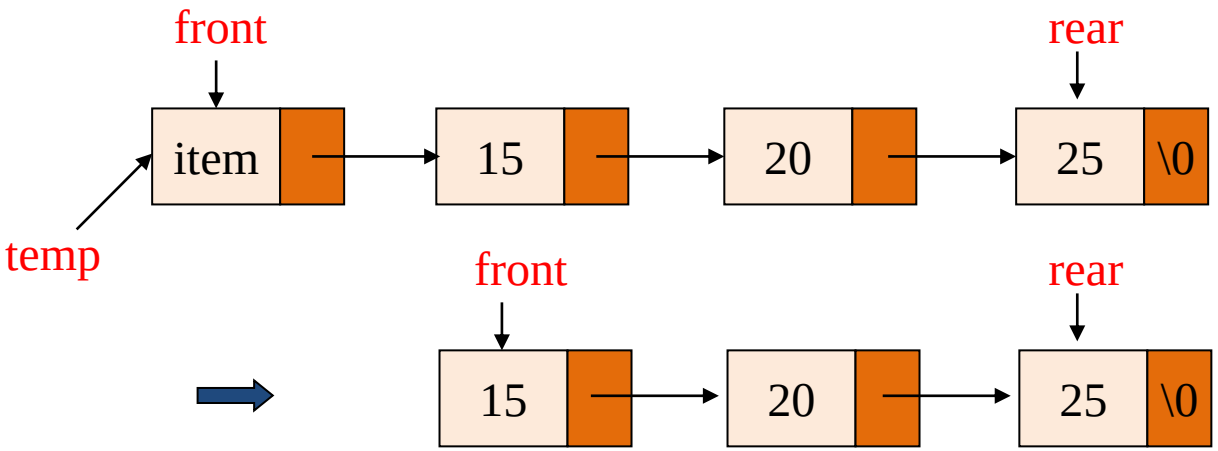
Linked Queues

- 삽입



Linked Queues

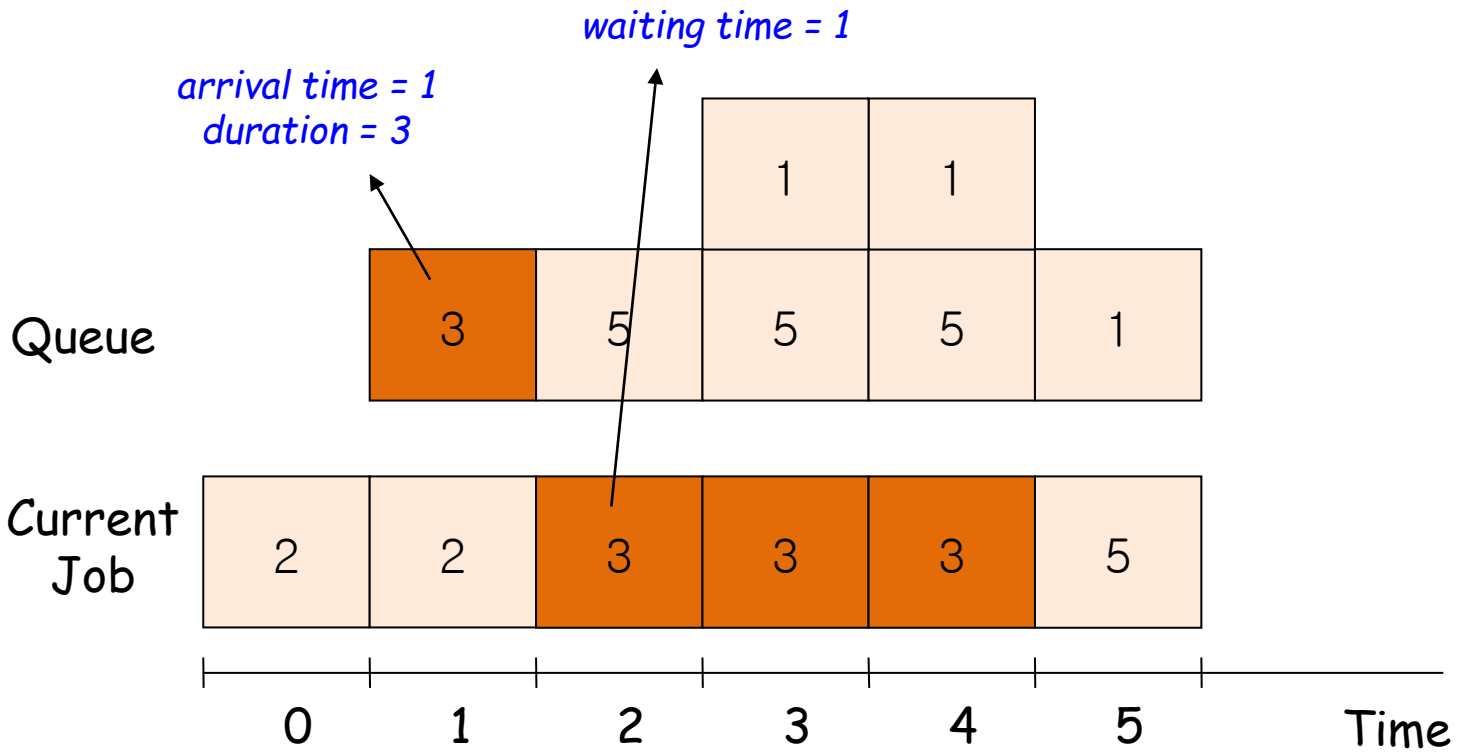
- 삭제



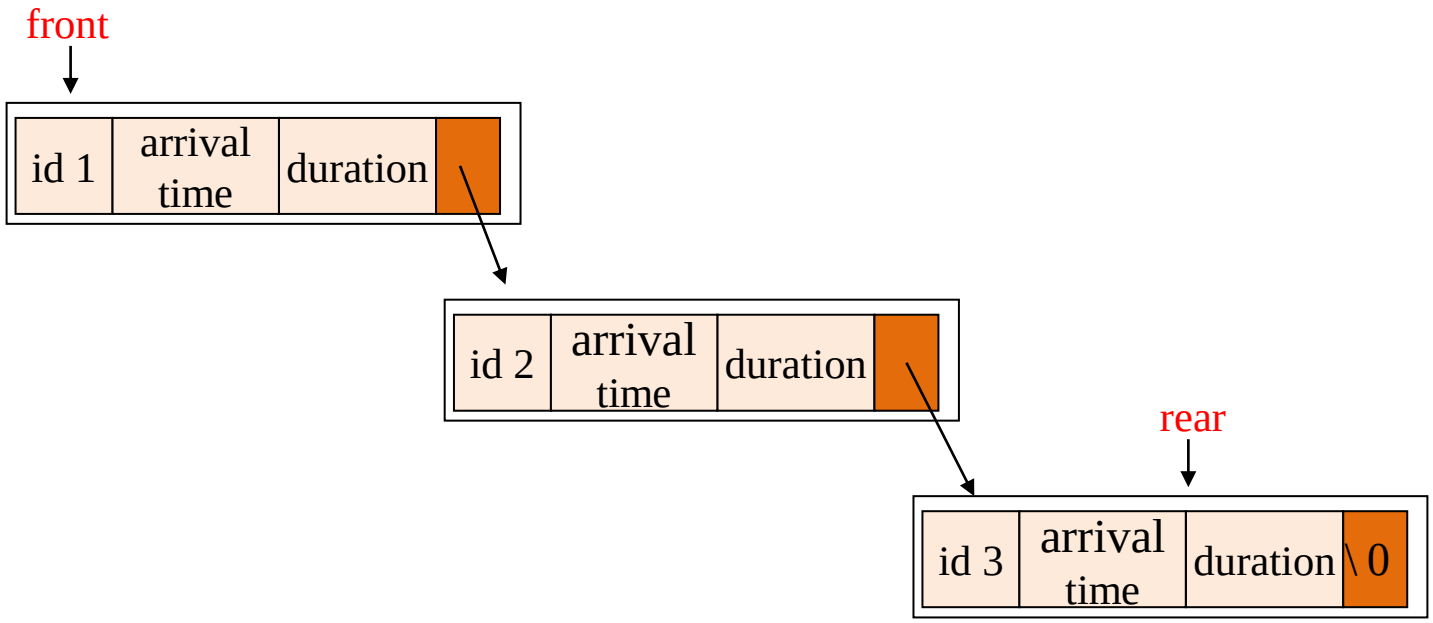
Queue Simulation

- 프린터 큐 시뮬레이션
  - 프린터에 도착하는 프린트 job 들을 처리하는 과정을 시뮬레이션
    - job 은 랜덤하게 프린터에 도착 (프린트 시간도 랜덤)
    - job 은 현재 다른 job이 프린트 중이면 큐에 대기
    - 프린터는 현재 job의 프린트가 끝나면 다음 job을 큐에서 가져와 그 job의 프린트 시간만큼 수행
  - 정해진 시간만큼 시뮬레이션 수행 후, 프린트된 모든 job 들의 평균 대기시간 출력 (누적 대기시간 / job 수)

Queue Simulation



Queue Simulation



실습 6-1. Linked Queue 구현

- 문자들의 Linked Queue를 테스트하는 프로그램 구현
- 명령어
  - +<c> : AddQ
  - - : DeleteQ
  - S : Show
  - Q : Quit

실습 6-1. 실행 예

```
**************Command************
+<c> : AddQ c , - : DeleteQ,
S: show, Q : Quit
*********************************

Command> +1
Command> +2
Command> +3
Command> +4
Command> +5
Command> s
1       2       3       4       5
Command> -
 1
Command> -
 2
Command> -
 3
Command> -
 4
Command> -
 5
Command> -
 Queue is empty !!!

Command> s
```

자료구조 및 함수

- void addq(Element e)
  - Results : Queue에 e를 삽입
- Element deleteq()
  - Requires : Queue가 비어 있지 않아야 함
  - Results : Queue에서 원소를 반환
- void queue_show()
  - Results : Queue의 내용을 보여줌
- boolean is_queue_empty()
  - Results : Queue가 비어 있으면 true 반환

linked_queue.h

```
#include <stdio.h>

#define  boolean   int
#define  true      1
#define  false     0

typedef char Element;

// Global queue
typedef struct queue* queue_pointer;
typedef struct queue {
    Element                 item;
    queue_pointer  link;
} queue;
queue_pointer front, rear;

void addq(Element e);
Element deleteq();
void queue_show();
boolean is_queue_empty();
```

linked_queue.c - main()함수

```
#include "linked_queue.h"

void main()
{
        char    c, e;
        front = rear = NULL;

        printf("**********Command**********\n");
        printf("+<c> : AddQ c , - : DeleteQ, \nS: show, Q : Quit\n");
        printf("*********************************\n");
        while (1) {
                printf("\nCommand> ");
                c = _getche();
                c = toupper(c);
                switch (c) {
                case '+':
                        e = _getche();
                        addq(e);
                        break;
```

linked_queue.c - main()함수

```
case '-':
                    if (is_queue_empty()) {
                            printf("\n Queue is empty !!! \n");
                    }
                    else {
                            e = deleteq();
                            printf("\n %c ", e);
                    }
                    break;
            case 'S':
                    queue_show();
                    break;
            case 'Q': printf("\n"); exit(1);
            default: break;
            }
    }
}
```

linked_queue.c 그 외 함수

```
boolean is_queue_empty()
{
    if(front == NULL)
        return true;
    else
        return false;
}
```

실습 6-2. Queue simulation

- 프린터 작업에 대한 simulation
  - Linked Queue로 프린터 큐 구현
- 시뮬레이션 방식
  - current_time을 증가시키면서 매 시각 가상의 프린트 job을 처리

```
while(current_time < MAX_SIMUL_TIME) {
        ...    ++current_time;
}
```

  - 시뮬레이션 종료 후 프린트 job 들의 평균 지연 시간 출력
- job
  - id – job의 ID
  - arrival time – job이 도착한 시간
  - duration – job의 프린트 시간

실습 6-2. Queue simulation

- 새로운 job의 도착 (is_job_arrived())
  - random() 을 호출, 반환값이 정해진 값보다 작으면 도착한 것으로 간주

  ➡ 새 job을 큐에 삽입 (insert_job_into_queue (id, arrival_time, duration))
    - 새 job의 프린트 시간 설정 : duration = random() * MAX_PRINTING_TIME + 1

- job 을 프린트 하기
  - 프린트 시간을 매 시각 하나씩 감소시키는 것을 프린트가 되는 것으로 간주
    - 매 시각 프린트 진행 : --remaining_time

- 프린트 완료 (is_printer_idle())
  - 남은 프린트 시간이 0 이하면 완료된 것임

  ➡ 큐에서 다음 job을 가져와 실행 (process_next_job())
    - current_job_id = job.id
    - remaining_time = job.duration

실습 6-2. Queue simulation

- 난수 발생 함수
  - rand() 함수
    - 0 ~ RAND_MAX 까지의 정수를 무작위로 반환
  - srand()함수
    - 매번 새로운 난수를 발생시키기 위하여 사용
    - srand(time(NULL))

- 0.0 ~ 1.0 까지의 실수를 무작위로 반환
  - return rand()/(double)RAND_MAX;
- 1 ~ 5 까지의 정수를 무작위로 반환
  - return (int)(5*(rand()/(double)RAND_MAX)) + 1;

실습 6-2. 실행 예

```
----- time 0 -----
새 jop <1>이 들어 왔습니다. 프린트 시간은 = 3 입니다.
 프린트를 시작합니다 - jop <1>...

현재 프린터 큐 : [ ]

----- time 1 -----
새 jop <2>이 들어 왔습니다. 프린트 시간은 = 5 입니다.
아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 2

현재 프린터 큐 : [ 2<5>, ]

----- time 2 -----
아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 1

현재 프린터 큐 : [ 2<5>, ]

----- time 3 -----
새 jop <3>이 들어 왔습니다. 프린트 시간은 = 2 입니다.
 프린트를 시작합니다 - jop <2>...

현재 프린터 큐 : [ 3<2>, ]

----- time 4 -----
아직 Jop <2>을 프린트하고 있습니다 ...남은 시간 : 4

현재 프린터 큐 : [ 3<2>, ]

----- time 5 -----
아직 Jop <2>을 프린트하고 있습니다 ...남은 시간 : 3

현재 프린터 큐 : [ 3<2>, ]

----- time 6 -----
아직 Jop <2>을 프린트하고 있습니다 ...남은 시간 : 2

현재 프린터 큐 : [ 3<2>, ]

----- time 7 -----
새 jop <4>이 들어 왔습니다. 프린트 시간은 = 5 입니다.
아직 Jop <2>을 프린트하고 있습니다 ...남은 시간 : 1

현재 프린터 큐 : [ 3<2>, 4<5>, ]

----- time 8 -----
 프린트를 시작합니다 - jop <3>...

현재 프린터 큐 : [ 4<5>, ]

----- time 9 -----
아직 Jop <3>을 프린트하고 있습니다 ...남은 시간 : 1

현재 프린터 큐 : [ 4<5>, ]
```

현재 큐를 출력
형태 : id<duration>

```
현재 프린터 큐 : [ 3<2>, 4<5>,
```

실습 6-2. 실행 예

```
----- time 11 -----
새 jop <6>이 들어 왔습니다. 프린트 시간은 = 2 입니다.
아직 Jop <4>을 프린트하고 있습니다 ...남은 시간 : 4

현재 프린터 큐 : [ 5<1>, 6<2>, ]

----- time 12 -----
새 jop <7>이 들어 왔습니다. 프린트 시간은 = 1 입니다.
아직 Jop <4>을 프린트하고 있습니다 ...남은 시간 : 3

현재 프린터 큐 : [ 5<1>, 6<2>, 7<1>, ]

----- time 13 -----
아직 Jop <4>을 프린트하고 있습니다 ...남은 시간 : 2

현재 프린터 큐 : [ 5<1>, 6<2>, 7<1>, ]

----- time 14 -----
새 jop <8>이 들어 왔습니다. 프린트 시간은 = 1 입니다.
아직 Jop <4>을 프린트하고 있습니다 ...남은 시간 : 1

현재 프린터 큐 : [ 5<1>, 6<2>, 7<1>, 8<1>, ]

----- time 15 -----
새 jop <9>이 들어 왔습니다. 프린트 시간은 = 1 입니다.
 프린트를 시작합니다 - jop <5>...

현재 프린터 큐 : [ 6<2>, 7<1>, 8<1>, 9<1>, ]

----- time 16 -----
새 jop <10>이 들어 왔습니다. 프린트 시간은 = 3 입니다.
 프린트를 시작합니다 - jop <6>...

현재 프린터 큐 : [ 7<1>, 8<1>, 9<1>, 10<3>, ]

----- time 17 -----
아직 Jop <6>을 프린트하고 있습니다 ...남은 시간 : 1

현재 프린터 큐 : [ 7<1>, 8<1>, 9<1>, 10<3>, ]

----- time 18 -----
 프린트를 시작합니다 - jop <7>...

현재 프린터 큐 : [ 8<1>, 9<1>, 10<3>, ]

----- time 19 -----
 프린트를 시작합니다 - jop <8>...

현재 프린터 큐 : [ 9<1>, 10<3>, ]
완료된 프린트 job = 8개
평균 지연 시간 = 3.875000 단위 시간
```

통계 자료 출력

자료구조 및 함수

- Job

```
typedef struct {
        int         id;                 // Job의 ID
        int         arrival_time;       // Job이 요청된(도착한) 시간
        int         duration;           // Job의 프린트 시간
} Job;
```

- void main( )
  - 0 ~ MAX_SIMUL_TIME 까지 current_time을 증가시키면서 매 시각 다음을 수행
    - 랜덤하게 새 job을 생성하여 큐에 삽입
    - 프린터가 놀고 있으면 다음 job을 수행
    - 아직 프린트 중이면 남은 프린트 시간을 하나 줄임
    - 최종적으로 수행된 job들의 평균 대기 시간을 출력

자료구조 및 함수

- void insert_job_into_queue (int id, int arrival_time, int duration)
  - 새 Job을 큐에 삽입
    (Job의 id, arrival_time, duration 등 설정)
- void process_next_job()
  - 다음 job을 큐에서 꺼내 수행
    (current_job_id, remaining time, total_wait_time, num_printed_job 등 설정)
- boolean is_job_arrived()
  - 새로운 job이 도착했는지를 랜덤하게 결정하여 true 혹은 false 반환
  - True일 확률은 JOB_ARRIVAL_PROB (if(random() < JOB_ARRIVAL_PROB) return true)
- boolean is_printer_idle()
  - 프린터가 놀고 있으면 (remaining time <= 0 이면) true 반환

queue_simulation.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// 시뮬레이션 설정 상수
#define MAX_SIMUL_TIME          20      // 시뮬레이션 진행 시간
#define MAX_PRINTING_TIME       5       // 각 Job의 가능한 최대 프린트 시간
#define JOB_ARRIVAL_PROB        0.5     // 매 시각 새로운 Job의 도착 확률
#define boolean unsigned char
#define true        1
#define false       0

// 시뮬레이션을 위한 global variables
int current_time = 0;           // 현재 시각
int new_job_id = 0;             // 새로운 Job의 ID
int current_job_id;             // 현재 프린트하고 있는 Job의 ID
int remaining_time;             // 현재 프린트하고 있는 Job의 남은 프린트 시간. 매 시각 1씩 감소
int total_wait_time;            // 프린트를 시작한 모든 Job의 대기시간(start time - arrival time)
의 합
int num_printed_jobs;           // 시뮬레이션이 끝날 때까지 프린트가 시작된 Job의 총 수
```

queue_simulation.h

```
// Job
typedef struct Job{
        int     id;             // Job ID
        int     arrival_time;   // Job이 요청된(도착한) 시간
        int     duration;       // Job의 프린트 시간
} Job;

// Global queue
typedef struct queue* queue_pointer;
typedef struct queue {
        Job             item;
        queue_pointer   link;
} queue;
queue_pointer front, rear;
```

queue_simulation.h

```
// 새 Job을 큐에 삽입
void insert_job_into_queue(int id, int arrival_time, int duration);
// 다음 job을 큐에서 꺼내 수행(현재 job id, remaining time 등 설정)
void process_next_job();

boolean is_job_arrived();
boolean is_printer_idle();

double random();                // 0.0 - 1.0 사이의 랜덤 값을 반환
int get_random_duration();      // 1 - MAX_PRINTING_TIME+1 사이의 랜덤 값을 반환

void addq(Job e);
Job deleteq();
boolean is_queue_empty();
void queue_show();              // 현재 큐에 있는 job의 id 들을 프린트
```

queue_simulation.c - main()함수

```
#include "queue_simulation.h"

void main()
{
    int duration;

    while (current_time < MAX_SIMUL_TIME) {
        printf("\n----- time %d ----- \n", current_time);

        // 새 job이 들어오면 큐에 삽입
        if (is_job_arrived()) {
            ++new_job_id;
            duration = get_random_duration();
            insert_job_into_queue(new_job_id, current_time, duration);
        }
```

queue_simulation.c - main()함수

```
        // 프린터가 놀고 있으면 다음 job을 수행
        if (is_printer_idle()) {
                if (!is_queue_empty()) process_next_job();
        }
        // 아직 프린트 중
        else {
                printf("아직 Jop <%d>을 프린트하고 있습니다 ...남은 시간 : %d \n",
    current_job_id, remaining_time);
                --remaining_time;
        }
        // 현재 큐의 상태를 보여줌
        queue_show();

        ++current_time;
    }

    // 통계 자료 출력  - 완료된 프린트 job 수, 평균 지연 시간
    (total_wait_time/num_printed_jobs)
}
```

queue_simulation.c 그 외 함수

```
// 새 Job을 큐에 삽입
void insert_job_into_queue(int id, int arrival_time, int duration)
{
    Job p;

    // id, arrival_time, duration 등 설정 후 job p를 큐에 삽입
    // addq() 사용

    printf("새 jop <%d>이 들어 왔습니다. 프린트 시간은 = %d 입니다. \n", id, duration);
}
```

queue_simulation.c 그 외 함수

```
// 다음 job을 큐에서 꺼내 수행(현재 job id, remaining time 등 설정)
void process_next_job()
{
    Job p;

    // deleteq() 사용 -  다음 job을 큐에서 꺼내 와서
    //     current_job_id,
    //     remaining_time (duration – 1),
    //     total_wait_time (total_wait_time + (current_time – arrival_time)) 등 설정

    ++num_printed_jobs;
    printf(" 프린트를 시작합니다 - jop <%d>... \n", current_job_id);
}
```

queue_simulation.c 그 외 함수

```
// 새로운 job이 도착했는지를 랜덤하게 결정. True일 확률은 ARRIVAL_PROB
boolean is_job_arrived()
{
    if(random() < JOB_ARRIVAL_PROB)
        return true;
    else
        return false;
}

// 프린터가 놀고 있으면(현재 job의 remaining time <= 0) true
boolean is_printer_idle()
{
    if(remaining_time <= 0)
        return true;
    else
        return false;
}
```

queue_simulation.c 그 외 함수

```
// 0.0 ~ 1.0 사이의 랜덤 값을 반환
double random()
{
    return rand()/(double)RAND_MAX;
}

// 1 ~ MAX_PRINTING_TIME+1 사이의 랜덤 값을 반환
int get_random_duration()
{
    return (int)(MAX_PRINTING_TIME * random()) + 1;
}
```