CSED101. Programming & Problem solving
Fall, 2020

Programming Assignment #4
(75 points)

강다현(dahyun.kang@postech.ac.kr)

- *Due*: 2020.12.18 23:59
- *Development Environment*: **Windows Visual Studio 2019**

- *제출물*
  - **C언어 코드 압축 파일 (assn4.zip)**
    - 소스 코드 (*.c와 *.h)를 압축하여 assn4.zip으로 제출한다.
    - 프로그램의 소스 코드를 이해하기 쉽도록 반드시 **주석**을 붙일 것.
  - **보고서 파일** (**assn4.docx, assn4.hwp 또는 assn4.pdf**)
    - AssnReadMe.pdf 를 참조하여 작성할 것.
    - **명예서약(Honor code)**: 표지에 다음의 내용을 포함한다. "나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다." 보고서 표지에 명예 서약이 없는 경우는 과제를 제출하지 않은 것으로 처리한다.
    - 소스코드와 보고서 파일을 LMS를 이용하여 제출한다.

- *주의사항*
  - 각 문제에 해당하는 요구사항을 반드시 지킬 것.
  - 모든 문제의 출력 형식은 아래의 예시들과 동일해야 하며, 같지 않을 시는 감점이 된다.
  - 각 문제에 제시되어 있는 파일이름으로 제출 할 것. 그 외의 다른 이름으로 제출하면 감점 또는 0점 처리된다.
  - 컴파일 & 실행이 안되면 무조건 0점 처리된다.
  - **하루 late시 20%가 감점되며, 3일 이상 지나면 받지 않는다. (0점 처리)**
  - 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 'POSTECH 전자컴퓨터공학부 부정행위 정의'를 따른다. (LMS의 과목 공지사항의 제목 [document about cheating]의 첨부파일인 disciplinary.pdf를 참조할 것.)
  - 이번 과제에서는 추가 기능 구현에 대한 추가 점수는 없습니다.

Problem: 가상의 치킨집 운영 프로그램

요약

본 과제에서는 가상의 치킨집에 손님들이 출입할 때 출입 명부를 파일로 기록하고, 각 테이블에서 음식을 주문하는 정보를 연결 리스트로 관리하는 것을 목적으로 한다.

개요

주요 기능은 다음과 같다.

1. 손님 입장
2. 테이블 현황 출력
3. 손님 퇴장
4. 손님 입/퇴장 시 매번 출입 명부에 기록
5. 음식 주문
6. 프로그램 종료

목적

이번 과제의 목적은 다음과 같다.

- 연결 리스트(Linked List)의 사용법을 익힌다.
- 사용자 정의 구조체(Structure)의 사용법을 익힌다.
- 다중 소스 파일의 사용법을 익힌다.

주의사항

1. 이번 과제는 총 5개의 명령어(enter, leave, order, show, quit)를 입력 받아 기능을 수행한다.
2. 최소한 각 명령어 별로 함수를 정의하여 사용한다. 명령어 외에 필요한 함수는 정의하여 사용할 수 있다. 기능적으로 독립됐거나 반복적으로 사용되는 기능은 사용자 함수를 정의해서 구현하도록 한다.
3. 이번 과제는 여러 개의 파일로 분할해서 작성한다.
   헤더 파일 작성은 마지막페이지의 "헤더 파일 작성" 부분을 참고한다. (점수 있음)
   - **`list.h, list.c`**
     - 테이블을 구조체로 정의하고 테이블 리스트를 연결 리스트로 구현한다.
     - 손님을 구조체로 정의하고 손님 리스트를 연결 리스트로 구현한다.
     - 테이블 및 손님과 관련된 함수의 선언 및 정의를 한다.
   - **`utils.h, utils.c`**
     - 테이블 및 손님과 관련 없지만 프로그램에 필요한 함수의 선언 및 정의를 한다.
   - **`assn4.c`**
     - `main()` 함수 정의, `main()` 함수 내에서는 명령어를 입력 받아 적절한 함수를 호출해 처리하는 기능을 구현한다.
   - **`macro.h`**
     - 각종 매크로 정의(제공 됨)
4. 이번 과제는 구조체와 연결 리스트를 활용하는 것이 목표이므로 문제에 구조체와 연결 리스트를 언급한 부분을 배열을 이용하여 해결 시 감점 처리 된다.
5. 명시한 에러 처리 외에는 고려할 필요가 없다.
6. string.h 에서 제공하는 라이브러리 함수를 사용할 수 있다.
7. 전역 변수, goto 문을 사용할 수 없다.

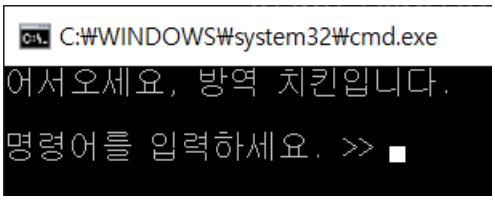
데이터 구조

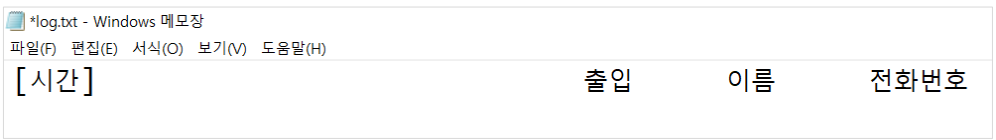
```
typedef struct Menu {
        int chicken;
        int beer;
} MENU;

typedef struct Person {
        char* name;
        char* phone;
        struct Person* next;
} PERSON;

typedef struct Table {
        int table_id;
        struct Menu menu;
        struct Person* people;
        struct Table* next;
} TABLE;
```

1) MENU
   - chicken: 테이블에서 시킨 치킨 수
   - beer: 테이블에서 시킨 생맥주 잔 수
2) PERSON
   - name: 손님의 이름을 저장하는 동적 할당 메모리 주소
   - phone: 손님의 전화번호를 저장하는 동적 할당 메모리 주소
   - next: 같은 테이블에 앉은 다음 손님 주소 (연결 리스트에서 사용)
3) TABLE
   - table_id: 테이블 고유 번호
   - menu: 테이블에서 시킨 메뉴
   - people: 테이블에 있는 손님들 연결 리스트 시작 노드
   - next: 다음 테이블 주소 (연결 리스트에서 사용)

프로그램 기능

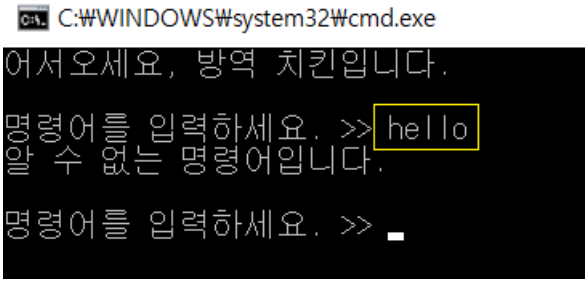
0. 프로그램 시작 화면





a. 프로그램을 시작하면 먼저 인사를 한다. 그리고 `log.txt`을 쓰기용으로 열어서 첫 행을 `"[시간]\t\t\t\t출입\t이름\t전화번호\n"` 으로 초기화한다. 이 파일은 프로그램이 시작할 때 1회 위와 같이 초기화 된다. 이 명부에 손님이 들어오고 나갈 때마다 정보를 기록하게 된다.

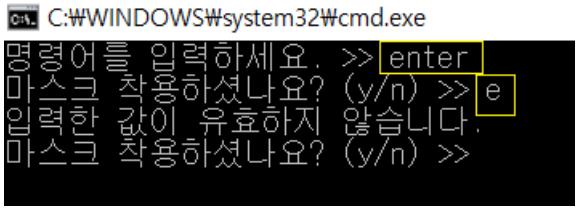
b. 명령어를 입력 받는다. 명령어는 `enter`, `leave`, `order`, `show`, `quit` 중 하나만 가능하다. 명령어 코드 문자열의 최대 길이는 `macro.h`에 정의되어 있는 `COMMAND_MAX_LEN`를 넘지 않는다.



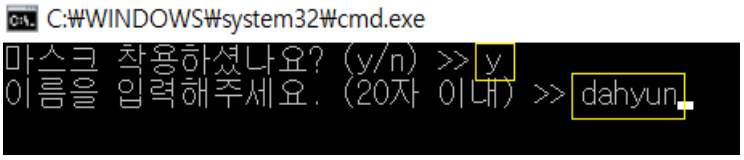
c. 유효하지 않은 명령어가 들어왔을 때: 위와 같이 에러 메시지 출력 후, 명령어를 재입력 받는다.

1. **손님 입장(명령어: `enter`)**
   a. [0. 프로그램 시작 b.] 에서 `enter`를 입력하면 손님이 입장한다.
   b. **방역 치킨집**은 방역과 손님의 건강에 신경을 쓰는 치킨집이다. 따라서, 손님이 입장하면 마스크 착용을 했는지 묻고 예/아니오 (`y`/`n`) 대답을 받는다.

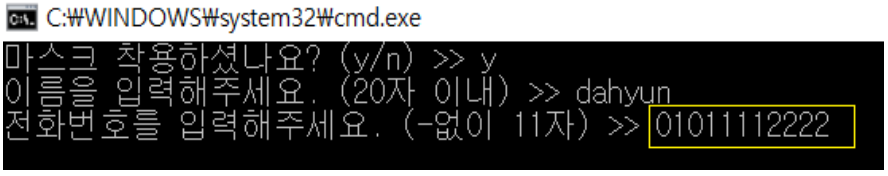
i. `n`을 입력 받았을 때: 마스크를 쓰지 않았다는 대답을 받으면 위의 그림과 같이 마스크를 쓰라는 안내와 함께 다시 [0. 프로그램 시작 b.]으로 돌아간다.
ii. `y`을 입력 받았을 때: 마스크를 썼다는 대답을 받으면 계속 진행한다.



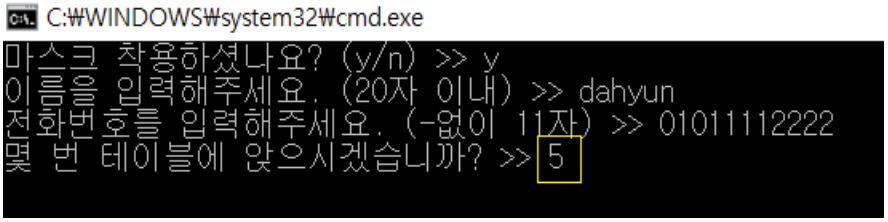
iii. y나 n가 아닌 문자를 입력 받았을 때: 위와 같이 예외처리 하고 [1. 손님 입장 b.]로 다시 돌아가서 마스크 착용 여부를 묻는다.



c. 위와 같이 이름(문자열)을 입력 받는다.
   i. 이름의 길이만큼 메모리를 동적 할당 받아서 저장하도록 한다.
   ii. 이름을 입력 받을 때의 최대길이는 NULL문자 포함하여 NAME_MAX_LEN 을 넘지 않는 것으로 가정한다. NAME_MAX_LEN은 macro.h에 정의되어 있다.
   iii. 한 테이블에 동명이인이 앉는 일은 없다고 가정한다.
   iv. 문자열 저장을 하기 위해 [유용한 팁 1.b]를 참고할 것.



d. 위와 같이 전화번호(문자열)를 입력 받는다.
   i. 전화번호의 길이만큼 메모리를 동적 할당 받아서 저장하도록 한다.
   ii. 전화번호를 입력 받을 때 최대 길이는 NULL문자 포함하여 PHONE_MAX_LEN을 넘지 않는 것으로 가정한다. PHONE_MAX_LEN은 macro.h에 정의되어 있다
   iii. 문자열 저장을 하기 위해 [유용한 팁 1.b]를 참고할 것.



e. 위와 같이 앉고 싶은 테이블의 번호(정수형)를 입력 받는다. 테이블 번호는 양의 정수만 고려한다.

f. [1. 손님 입장 0.] 에서 입력 받은 번호가 테이블의 고유한 번호 (table_id)가 된다. 입력 받은 테이블 번호에 해당하는 테이블 노드에 입장한 손님의 정보 노드를 저장한다.
   i. 테이블과 손님들은 모두 연결 리스트로 관리된다.
   ii. 없는 테이블 번호를 입력 받았을 때: 테이블 리스트에 새로운 테이블 노드를 만들어 추가한다. 새로운 노드를 테이블 리스트에 삽입할 때, **테이블 번호에 따라 오름차순으로 정렬되도록 삽입한다.** 그리고, 새로운 테이블에 새로 들어온 손님 노드를 추가한다.
   iii. 있는 테이블 번호를 입력 받았을 때: 해당하는 테이블을 찾은 뒤, 그 테이블의 손님

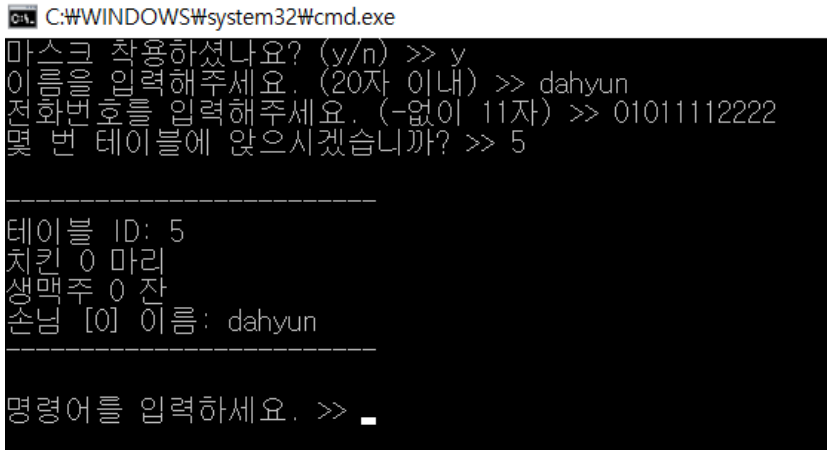
리스트의 마지막에 새로 들어온 손님 노드를 추가한다.

log.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

[시간]	출입	이름	전화번호
[Wed Dec 2 21:51:32 2020]	In	dahyun	01011112222

g. 위와 같이 현재 시각, 출입 형태 (In), 이름, 전화번호를 log.txt에 기록한다. 이 부분은 [3. 출입 명부 기록]에서 보충설명한다.



h. 전체 테이블 현황을 출력한다. 이 부분은 [2. 테이블 현황 출력]에서 보충설명한다.

i. [0. 프로그램 시작 b.]로 돌아가서 다시 명령어를 받는다.

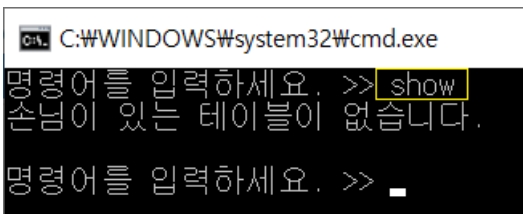
2. 테이블 현황 출력(명령어: show)

a. [0. 프로그램 시작 b.] 에서 show를 입력하면 방역 치킨집 테이블의 현황을 출력한다. 각 테이블에 대하여 각 참석자들의 이름을 순서대로 출력해준다. 테이블 출력 순서는 테이블 번호를 기준으로 오름차순으로 정렬되도록 한다.

5번에 dahyun 입장	3번에 wonho 입장	5번에 ys 입장

b. 위는 테이블 5번에 dahyun, 테이블 3번에 wonho, 테이블 5번에 ys이 차례로 들어왔을 때 show 명령어의 결과를 보여준다.

c. 손님 번호는 0부터 시작한다. 손님 번호는 각 테이블마다 0부터 시작하여, 중간에 있던 손님이 나가도 0, 1, 2, … 와 같은 연속적 번호를 가진다.



d. 테이블이 하나도 없는 경우, show 명령 입력시, 위와 같이 출력한다.

3. **손님 입/퇴장 시 매번 출입 명부에 기록**

a. 방역 치킨집은 방역 지침에 따라 현재 시각, 손님의 출입, 이름, 전화번호를 출입 명부에 적는다. [1. 손님 입장] 과 [5. 손님 퇴장] 기능을 수행할 때마다 자동으로 출입 명부 `log.txt`에 정보를 기록하도록 한다. 각 항목은 탭문자 (`\t`) 로 분리한다.

*log.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

[시간]	출입	이름	전화번호
[Wed Dec 2 22:21:23 2020]	In	dahyun	01011112222
[Wed Dec 2 22:21:32 2020]	In	wonho	01022223333
[Wed Dec 2 22:23:30 2020]	In	ys	01033334444
[Wed Dec 2 22:23:55 2020]	In	hunho	01044445555

b. 위는 네 명의 손님이 입장한 상황에서 출입 명부 예시이다.

*log.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

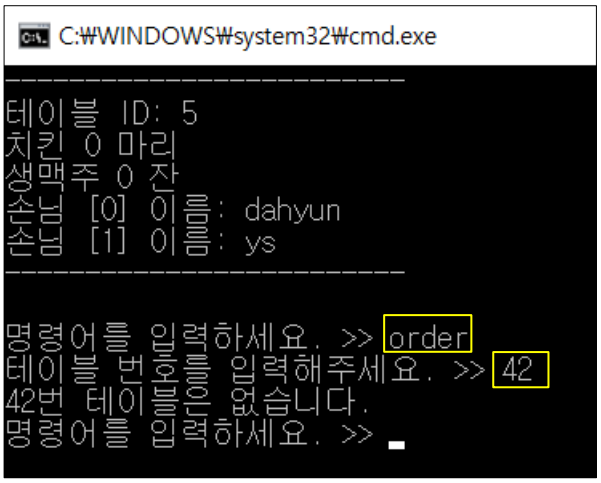
[시간]	출입	이름	전화번호
[Wed Dec 2 22:21:23 2020]	In	dahyun	01011112222
[Wed Dec 2 22:21:32 2020]	In	wonho	01022223333
[Wed Dec 2 22:23:30 2020]	In	ys	01033334444
[Wed Dec 2 22:23:55 2020]	In	hunho	01044445555
[Wed Dec 2 22:41:01 2020]	Out	wonho	01022223333

c. 위는 네 명의 손님이 입장했다가, 그 중에 한 명이 나갔을 때의 출입 명부 예시이다.

d. 시간은 명부를 적기 직전에 연, 월, 일, 시를 기록하도록 한다. 시간을 재고 기록하는 방법은 [유용한 팁 5.]에서 보충설명한다.

4. **음식 주문(명령어: order)**

a. [0. 프로그램 시작 b.] 에서 order를 입력하면 음식을 주문할 수 있다.



b. 음식을 주문 할 테이블 번호를 입력 받는다. 이 때, 유효하지 않은 테이블 번호를 입력했을 때, 위와 같이 예외 처리하고 다시 [0. 프로그램 시작 b.]으로 돌아간다.

c. 음식을 주문하기 위해서 예시 주문 코드를 출력한 뒤, 주문코드를 입력한다.

A. 주문 코드는 음식 코드 (c: 치킨 혹은 b: 생맥주)와 숫자 코드 (정수형)가 이어져있는 문자열이다.

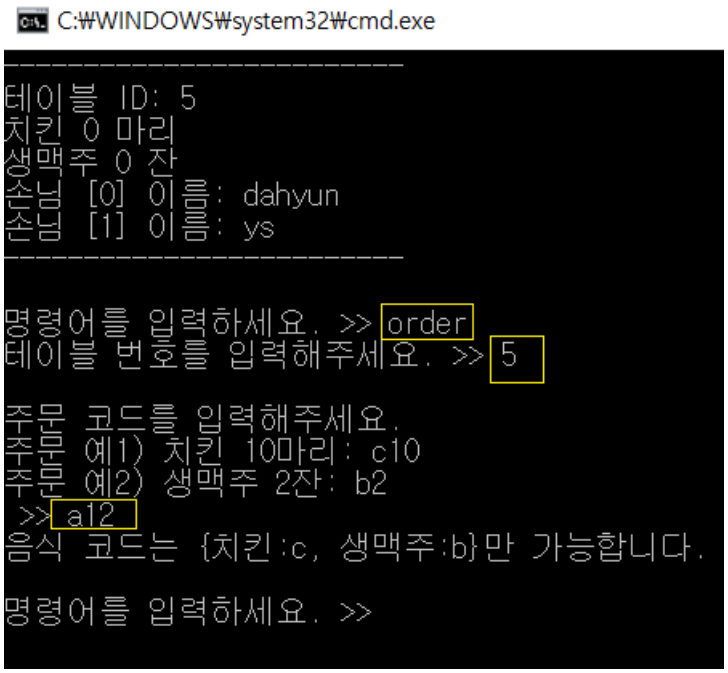
i. 주문 코드 예1) 치킨 10마리: "c10"

ii. 주문 코드 예2) 생맥주 2잔: "b2"

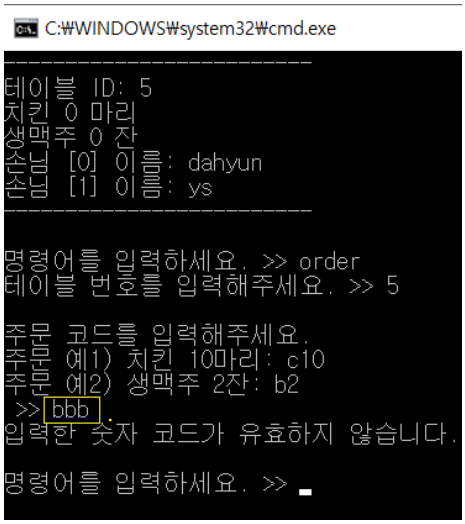
iii. 치킨과 맥주를 동시에 주문하지 않는 것으로 한다.

iv. 주문 코드 문자열의 최대 길이는 macro.h에 정의되어 있는 ORDER_CODE_MAX_LEN를 넘지 않는다.

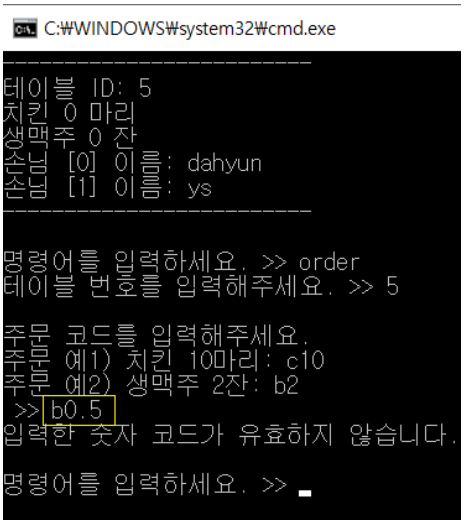
B. 주문 코드에 대한 예외 처리는 다음과 같이 진행한다.



i. 위와 같이 음식 코드가 c나 b가 아닌 경우: 위와 같이 예외 처리하고 다시 [0. 프로그램 시작 b.]으로 돌아간다.

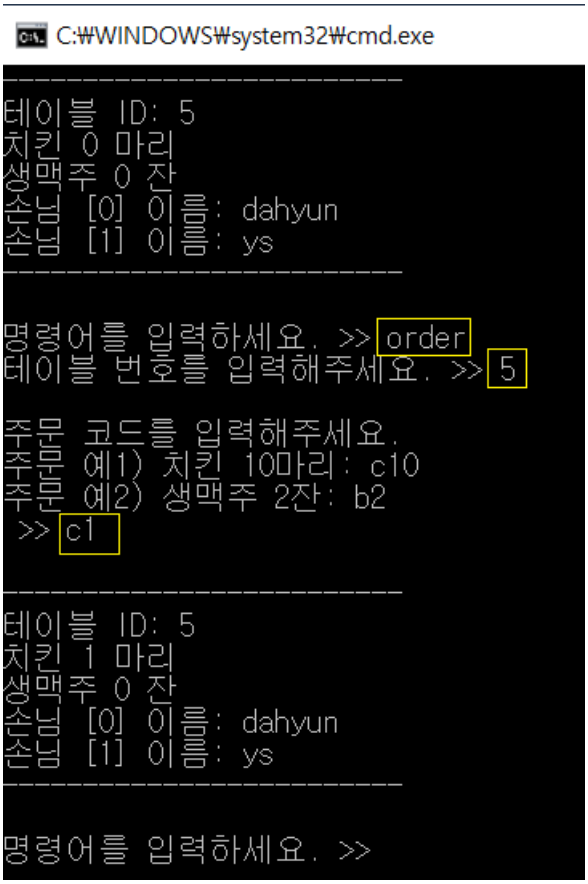


유효하지 않은 입력 예시 1



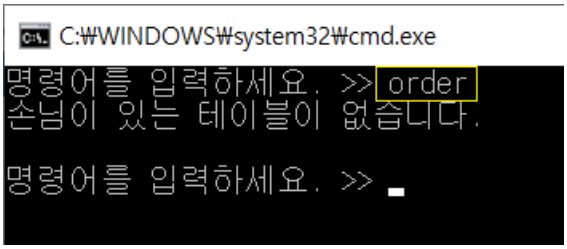
유효하지 않은 입력 예시 2

ii. 위와 같이 숫자 코드에 소수점, 문자, 0이하 숫자 등 양의 정수가 아닌 것이 들어올 때, 위와 같이 예외 처리하고 다시 [0. 프로그램 시작 b.]으로 돌아간다.



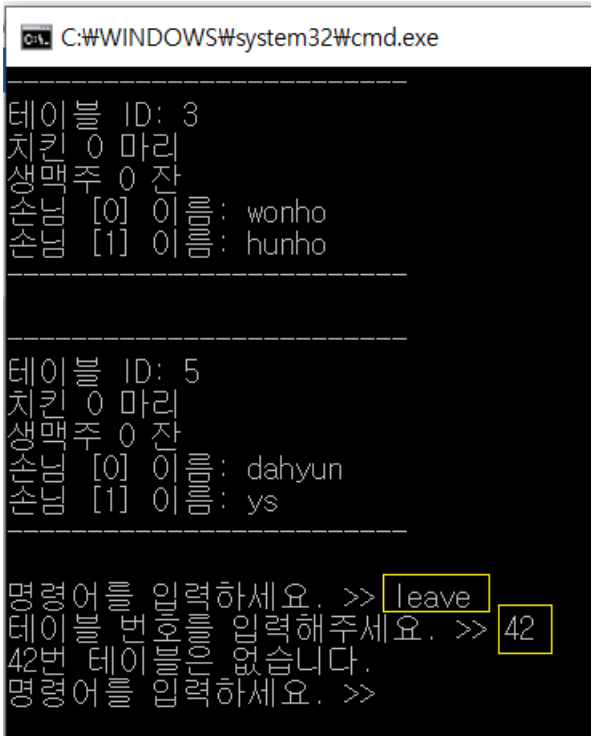
d. 위와 같이 정상적인 주문 코드에 한하여 해당 테이블에 음식 숫자를 더해준다. 주문 코드를 처리하는 요령은 [유용한 팁 2.] 참고.

e. 음식 주문이 정상적으로 마무리되면 테이블 현황을 출력하고 [0. 프로그램 시작 b.]으로 돌아가서 새로운 명령어를 받는다.



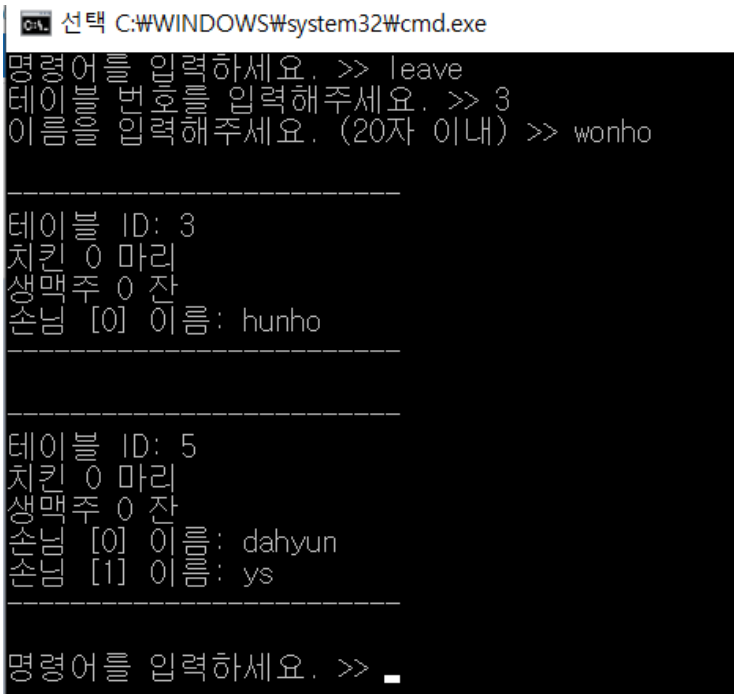
f. 테이블이 하나도 없는 경우, order 명령 입력시, 위와 같이 출력한다.

5. **손님 퇴장(명령어: leave)**
   a. [0. 프로그램 시작 b.] 에서 leave를 입력하면 손님이 퇴장할 수 있다.

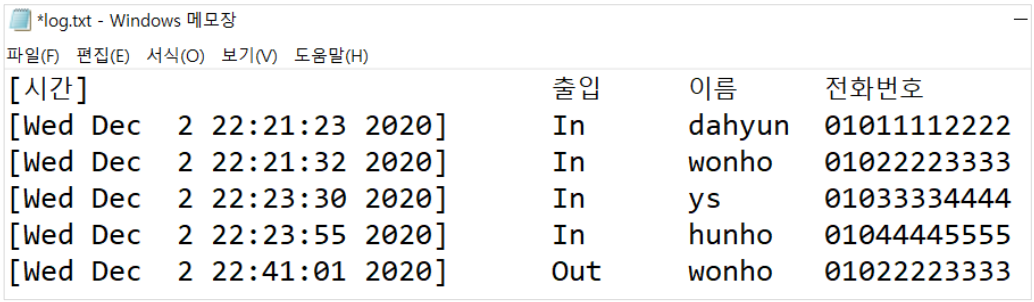


   b. 나가려고 하는 손님의 테이블 번호를 입력 받는다. 이 때, 유효하지 않은 테이블 번호를 입력했을 때, 위와 같이 예외 처리하고 다시 [0. 프로그램 시작 b.]으로 돌아간다.

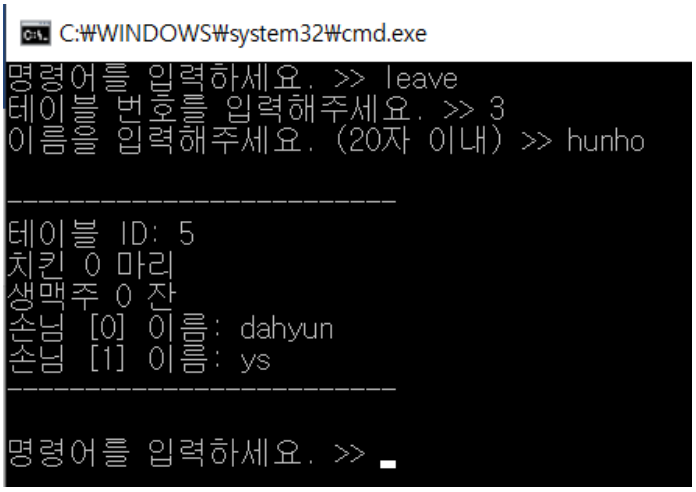
   c. [5. 손님 퇴장 b.]에서 테이블 번호가 유효할 시, 나가려고 하는 손님의 이름을 입력 받는다. 이때, [5. 손님 퇴장 0.] 에서 입력 받은 테이블에 손님 이름이 없을 시, 위와 같이 예외 처리하고 다시 [0. 프로그램 시작 b.]으로 돌아간다.



d. [5. 손님 퇴장 c.]에서 나가려는 손님이 있을 시, 해당 테이블의 손님 리스트에서 해당 손님 노드를 삭제한다. 동적 할당한 메모리가 있으면 해제해준다. 위는 3번 테이블의 wonho 손님이 퇴장한 예시이다.

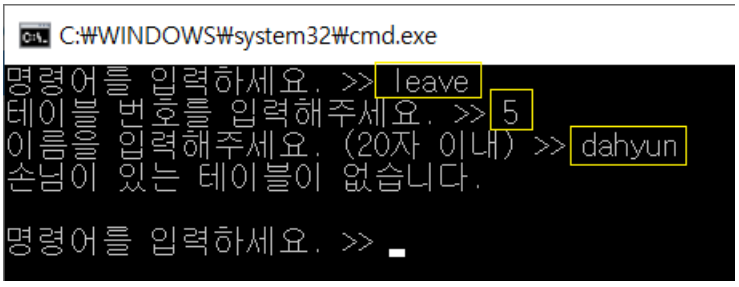


e. 위와 같이 퇴장한 손님에 대하여 시각, 출입 형태 (Out), 이름, 전화번호를 log.txt에 기록한다. 이 부분은 [3. 출입 명부 기록]에서 보충설명한다.

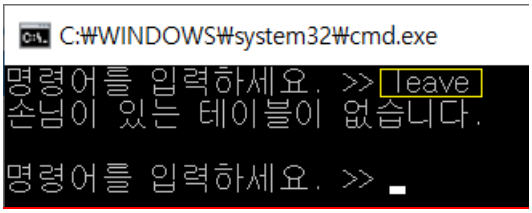


f. 만약 해당 테이블에 모든 손님이 떠났다면, 해당 테이블을 테이블 리스트에서 삭제한다. 동적 할당한 메모리가 있으면 해제해준다. 위는 3번 테이블의 모든 손님이 퇴장한 예시이다.

g. 손님 퇴장이 정상적으로 마무리되면 전체 테이블 현황을 출력하고 [0. 프로그램 시작 b.]로 돌아가서 새로운 명령어를 받는다.

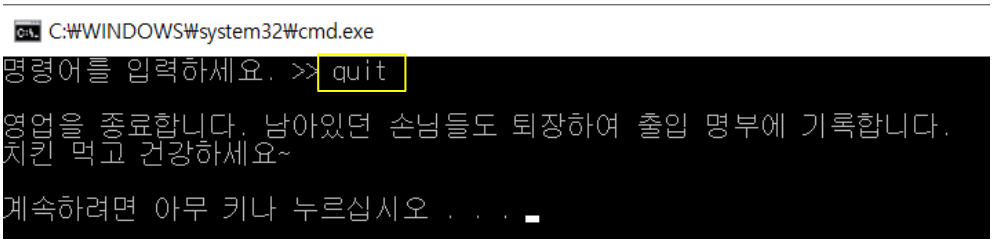


h. 마지막 테이블에 마지막 손님이 떠난 경우 전체 테이블 현황 (즉, 손님이 있는 테이블이 없는 상황)을 위와 같이 출력하고 [0. 프로그램 시작 b.]로 돌아가서 새로운 명령어를 받는다.

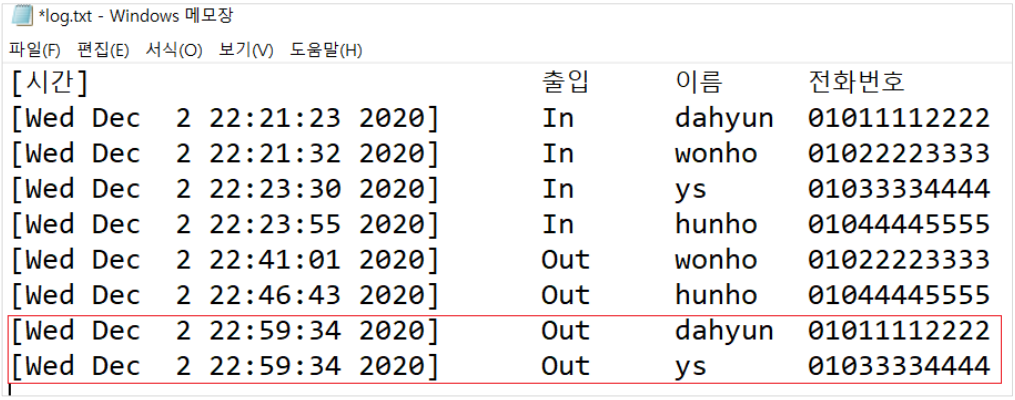


i. 테이블이 하나도 없는 경우, leave 명령어 입력 시 위와 같이 출력하고 [0. 프로그램 시작 b.]로 돌아가서 새로운 명령어를 받는다.

6. 프로그램 종료(명령어: quit)



a. [0. 프로그램 시작 b.] 에서 quit을 입력하면 인사를 한 뒤, 남아있던 모든 손님들을 퇴장시키고 퇴장 기록을 남긴다.



b. 퇴장 기록 형식은 [5. 손님 퇴장 e.]와 똑같이 하고, 순서는 테이블 번호 오름차순으로 기록한다. 위의 예제는 프로그램을 종료하면서 dahyun과 ys 손님이 퇴장 기록을 남긴 것을 보여준다.

c. **프로그램에서 할당한 모든 동적 할당 메모리를 해제하고 종료한다.**

유용한 팁

1. 문자열을 다룰 때, `string.h`의 함수를 이용해보세요.
   a. 문자열 비교: `strcmp`
   b. 문자열 복사: `strcpy` (혹은 `strcpy_s`)

2. 문자열의 시작을 조정해보세요.

```
1  char str[12] = "hello world";
2  printf("%s\n", str + 6);
3  printf("%s\n", &str[6]);
```

   위 코드에서 2와 3번째 줄에서는 같은 결과가 출력 됩니다.

3. 문자열을 정수로 바꾸고 싶을 때, `atoi` 함수를 이용해보세요. 이 함수에 정수형으로 표현될 수 없는 문자열을 넣었을 때 결과값이 어떻게 달라지는 지 확인해보세요.

4. 포인터를 다룰 때 연산자 우선순위에 주의하세요.

```
1  TABLE* ptr1;
2  TABLE** ptr2;
3  TABLE t;
4  t.menu.beer = 1;
5  ptr1 = &t;
6  ptr2 = &ptr1;
7
8  // printf("맥주 %d잔\n", *ptr2->menu.beer); // 에러 발생
9  printf("맥주 %d잔\n", (*ptr2)->menu.beer); // 정상 작동
```

   위 코드의 8번째 줄에서 에러가 발생하는 이유는 연산자 우선순위와 관련이 있습니다.

5. 주어진 `void get_current_time(char* time_string)` 함수를 이용하세요. 다음은 실행 예시입니다.

```
void get_current_time(char* time_string) {
        time_t current_time;
        char *new_line_pos;

        // 숫자 형식으로 시간을 저장
        current_time = time(NULL);

        // 숫자 형식 시간을 문자열 형태로 바꾸어 char* time_string에 저장
        // TIME_MAX_LEN은 macro.h에서 제공됨
        ctime_s(time_string, TIME_MAX_LEN, &current_time);

        // 줄바꿈 문자를 NULL로 바꿈
        if ((new_line_pos = strchr(time_string, '\n')) != NULL)
                *new_line_pos = '\0';
}
```

```
char current_time[TIME_MAX_LEN];
get_current_time(current_time);
printf("지금 이 순간: %s\n", current_time);
```

6. 마스크 모양 아스키 아트

```
printf("                        `-,:~<*r)¦vvvvv¦)r*<=:,-`                        \n");
printf("                    .,=*?vvvvvvvvvvvvvvvvvvvvvv*=,.                      \n");
printf("                `_~rvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvr~_`                 \n");
printf("             `_~?vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv?~_`             \n");
printf("           '!\vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv¦)`            \n");
printf("         `=)vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv?=`          \n");
printf("       `:)vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv):`        \n");
printf("      .*vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv*-      \n");
printf("    `!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv!`     \n");
printf("   `~vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv~`    \n");
printf("  `~vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvxvvvvvvvvvvvvvvvvvvvvvv>`   \n");
printf("  !vvvvvvvvvvvvvvvY@@@@@B##vvvvvvvvvvvvvvvvvvv@@@@@@vvvvvvvvvvvvvvv!   \n");
printf(" :vvvvvvvvvv}K@@@@@@@@@@@#vvvvvvvvvvvvvvVQ@@@@@@@@@@@B9m}vvvvvvvvvv: \n");
printf("`rvvvvvvvvvK@@@@@BBgdPB@@@@@vvvvvvvvvvvvvv@@@@VXPdgB@@@@@@Evvvvvvvvvvr`\n");
printf(":vvvvvvvvvv@@@@dIYvvvvv@@@@@vvvvvvvvvvv#@@@vvvvvvv}U@@@Rvvvvvvvvvv:\n");
printf("<vvvvvvvvvvv]ivvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvi]vvvvvvvvvv<\n");
printf("._!^)vvvvvvvvvvvvvv?)))))))))))))))))))))))))))))¦vvvvvvvvvvvvvv)^!_.\n");
printf("`````-:~rvvvvvvvv¦=-````````````````````````````````-~¦vvvvvvvr~:-`````\n");
printf("*~:-`````.,=*vvvv:`````````````````````````````````:vvvvr=,.`````-:~*\n");
printf(")vvvr<:_'````.,=*)_`````````````````````````````,)^!,.`````_:<rvvv)\n");
printf("^vvvvvvv)^!_`````````````````````````````````````-!^)vvvvvvv^\n");
printf(":vvvvvvvvvv)^!,'````````````````````````````.,!^)vvvvvvvvvv:\n");
printf("`rvvvvvvvvvvvvv\\_`````````````````````````_¦vvvvvvvvvvvvvr`\n");
printf(" \"vvvvvvvvvvvvv_`````````````````````````_vvvvvvvvvvvvv\" \n");
printf("  !vvvvvvvvvvvvv_`````````````````````````_vvvvvvvvvvvvv!  \n");
printf("  `_\":=~^*r¦vvvvv_`````````````````````````_vvvvv¦r*^~!:\"_`  \n");
printf("    `````.-,:!=.`````````````````````````.=!:,-.`````    \n");
printf("     `,_-'`````````````````````````````````````'-_,`     \n");
printf("     .*vvv\\)*^~=:' `````````````````````````.:=~^*)\vvv*.     \n");
printf("      `:)vvvvvvv_`````````````````````````_vvvvvvv):`      \n");
printf("        `=?vvvvv:`````````````````````````:vvvvv?=`        \n");
printf("          `!¦vvvv?!-`````````````````````-!?vvvv¦)`        \n");
printf("            `_~)vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv?~_`          \n");
printf("              `_~rvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvr~_`            \n");
printf("                 .,=*?vvvvvvvvvvvvvvvvvvvvvv\\*=,.                 \n");
printf("                     `-,:=<*r)?vvvvv?)r*<=:,-`                     \n");
printf("**********************마스크 착용 후 입장 부탁드립니다.**********************\n");
```

헤더 파일 작성

- 헤더 파일 작성 예시 (1) (list.h)

```
#ifndef LIST_H
#define LIST_H

// 구조체 정의
…
// 함수 선언
…

#endif
```

- 헤더 파일 작성 예시 (2) (list.h)

```
#pragma once
// 구조체 정의
…
// 함수 선언
…
```

위의 사용자 정의 헤더 파일 list.h를 필요한 곳에서 include 하여 사용합니다. list.h에는 함수 원형만 선언하고, 함수 정의는 list.c에서 작성합니다. utils.h와 utils.c도 마찬가지로 분할하여 작성합니다.

헤더 파일 작성시 위와 같이 작성을 하는 이유에 대해서 간략하게 조사하여 보고서에 서술합니다.(과제 점수에 포함됩니다.)