```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NAME_LEN 50  // ì\213\235ë\213¹ ì\235´ë¦\204ì\235\230 ìµ\234ë\214\200ê¸¸ì\235´
#define INIT_CAPACITY 10 // ë\217\231ì \201 ë°°ì\227´ì\235\230 ì´\210ê¸°ì\232©ë\237\211

// ì\235\214ì\213\235 ì¢\205ë¥\230ë¥¼ ë\202\230í\203\200ë\202´ë\212\224 ì\227´ê±°í\230\225
typedef enum { Korean, Chinese, Japanese, Western, Unknown } FoodType;

// ì\213\235ë\213¹ ì \225ë³´ë¥¼ ì \200ì\236¥í\225\230ë\212\224 êµ¬ì¡°ì²´
typedef struct {
  char name[MAX_NAME_LEN]; // ì\213\235ë\213¹ ì\235´ë¦\204
  FoodType type;           // ì\235\214ì\213\235 ì¢\205ë¥\230 (Korean, Chinese, Japanese, Western)
  double x;                // ì\213\235ë\213¹ì\235\230 x ì¢\214í\221\234
  double y;                // ì\213\235ë\213¹ì\235\230 y ì¢\214í\221\234
  double score;            // í\217\211ì \220 (0.0 ~ 5.0)
} Restaurant;

// ì\213\235ë\213¹ ì \225ë³´ë\223¤ì\235\204 ê´\200ë¦¬í\225\230ë\212\224 êµ¬ì¡°ì²´
typedef struct {
  Restaurant *restaurants; // ì\213\235ë\213¹ ì \225ë³´ë¥¼ ì \200ì\236¥í\225\230ë\212\224 ë\217\231ì \201 ë°°ì\227´
  int capacity;            // ë\217\231ì \201 ë°°ì\227´ì\235\230 í\230\204ì\236¬ì\232©ë\237\211
  int count;               // ì \200ì\236¥ë\220\234 ì\213\235ë\213¹ ê°\234ì\210\230
} RestaurantManager;

// í\225¨ì\210\230 ì\204 ì\226¸
void add_restaurant(RestaurantManager *manager);
void recommend_restaurants(RestaurantManager *manager);
void search_restaurants(RestaurantManager *manager);
void print_restaurant(Restaurant restaurant);
void reallocate_restaurants(RestaurantManager *manager);
double calculate_distance(double x1, double y1, double x2, double y2);
FoodType get_food_type(const char *type_str);
void print_food_type(FoodType type);

// ì\213\235ë\213¹ ì \225ë³´ë¥¼ ì¶\224ê°\200í\225\230ë\212\224 í\225¨ì\210\230
void add_restaurant(RestaurantManager *manager) {
  if (manager->count >= manager->capacity) {
    reallocate_restaurants(manager);
  }

  printf("--- Add Restaurant ---\n");

  // ì\213\235ë\213¹ ì\235´ë¦\204 ì\236\205ë ¥
  printf("Enter restaurant name: ");
  scanf("%s", manager->restaurants[manager->count].name);

  // ì\235\214ì\213\235 ì¢\205ë¥\230 ì\236\205ë ¥
  char type_str[20];
  printf("Enter food type (Korean, Chinese, Japanese, Western): ");
  scanf("%s", type_str);

  manager->restaurants[manager->count].type = get_food_type(type_str);
  if (manager->restaurants[manager->count].type == Unknown) {
    printf("Invalid food type.\n");
    getchar(); // ë\202¨ì\225\204 ì\236\210ë\212\224 ê°\234í\226\211 ë¬¸ì\236\220ì \234ê±°
    return;
  }

  // ì\234\204ì¹\230 ì\236\205ë ¥
  printf("Enter location (x y): ");
  if (scanf("%lf %lf", &manager->restaurants[manager->count].x, &manager->restaurants[manager->count].y) != 2) {
    printf("Invalid location format.\n");
    getchar(); // ë\202¨ì\225\204 ì\236\210ë\212\224 ê°\234í\226\211 ë¬¸ì\236\220ì \234ê±°
    return;
  }

  // í\217\211ì \220 ì\236\205ë ¥
  printf("Enter score (0.0 ~ 5.0): ");
  if (scanf("%lf", &manager->restaurants[manager->count].score) != 1 ||
      manager->restaurants[manager->count].score < 0.0 ||
      manager->restaurants[manager->count].score > 5.0) {
    printf("Invalid score.\n");
    getchar(); // ë\202¨ì\225\204 ì\236\210ë\212\224 ê°\234í\226\211 ë¬¸ì\236\220ì \234ê±°
    return;
  }

  manager->count++;
  printf("Restaurant added successfully!\n");
  getchar(); // ë\202¨ì\225\204 ì\236\210ë\212\224 ê°\234í\226\211 ë¬¸ì\236\220ì \234ê±°
}

// ì¶\224ì²\234 ì\213\235ë\213¹ ì¶\234ë ¥ í\225¨ì\210\230
void recommend_restaurants(RestaurantManager *manager) {
  char type_str[20];
  double x, y, min_score;

  printf("--- Recommend Restaurants ---\n");
  printf("Enter food type (Korean, Chinese, Japanese, Western): ");
  scanf("%s", type_str);

  FoodType type = get_food_type(type_str);
  if (type == Unknown) {
    printf("Invalid food type.\n");
    return;
  }

  printf("Enter your location (x y): ");
  scanf("%lf %lf", &x, &y);
  printf("Enter minimum score: ");
  scanf("%lf", &min_score);

  printf("\n--- Recommended Restaurants ---\n");
  int found = 0;

  for (int i = 0; i < manager->count; i++) {
    if (manager->restaurants[i].type == type &&
        calculate_distance(x, y, manager->restaurants[i].x, manager->restaurants[i].y) <= 500 &&
        manager->restaurants[i].score >= min_score) {
      print_restaurant(manager->restaurants[i]);
      found = 1;
    }
  }

  if (!found) {
    printf("No result\n");
  }
}
```

```c
// ê²\200ì\203\211 ì\213\235ë\213¹ ì¶\234ë ¥ í\225¨ì\210\230
void search_restaurants(RestaurantManager *manager) {
  char query[MAX_NAME_LEN];
  double x, y, distance;

  printf("--- Search for Restaurants ---\n");
  printf("Enter search query (type, name, or location): ");
  fgets(query, MAX_NAME_LEN, stdin);
  query[strcspn(query, "\n")] = '\0';

  int is_numeric_input = sscanf(query, "%lf %lf %lf", &x, &y, &distance);

  printf("\n--- Search Results ---\n");
  int found = 0;

  if (is_numeric_input == 3) {
    // ì\210«ì\236\220 3ê°\234 ì\236\205ë ¥ ì²\230ë¦¬ (x, y, ê±°ë¦¬ d)
    for (int i = 0; i < manager->count; i++) {
      if (calculate_distance(x, y, manager->restaurants[i].x,
manager->restaurants[i].y) <= distance) {
        print_restaurant(manager->restaurants[i]);
        found = 1;
      }
    }
  } else if (get_food_type(query) != Unknown) {
    // ì\235\214ì\213\235 ì¢\205ë¥\230 ì\236\205ë ¥ ì²\230ë¦¬
    FoodType type = get_food_type(query);
    for (int i = 0; i < manager->count; i++) {
      if (manager->restaurants[i].type == type) {
        print_restaurant(manager->restaurants[i]);
        found = 1;
      }
    }
  } else {
    // ë¶\200ë¶\204 ë¬¸ì\236\220ì\227´ ê²\200ì\203\211 ì²\230ë¦¬
    for (int i = 0; i < manager->count; i++) {
      if (strstr(manager->restaurants[i].name, query) != NULL) {
        print_restaurant(manager->restaurants[i]);
        found = 1;
      }
    }
  }

  if (!found) {
    printf("No result\n");
  }
}

// ë\217\231ì \201 ë°°ì\227´ì\235\230 í\201¬ê¸°ë¥¼
ì\236¬í\225 ë\213²í\225\230ë\212\224 í\225¨ì\210\230
void reallocate_restaurants(RestaurantManager *manager) {
  manager->capacity *= 2;
  manager->restaurants = (Restaurant *)realloc(manager->restaurants,
                                               sizeof(Restaurant) *
manager->capacity);

  if (manager->restaurants == NULL) {
    printf("Memory reallocation error!\n");
    exit(1);
  }
}

// ë\221\220 ì \220 ì\202¬ì\235´ì\235\230 ê±°ë¦¬ë¥¼
ê³\204ì\202°í\225\230ë\212\224 í\225¨ì\210\230
double calculate_distance(double x1, double y1, double x2, double y2) {
  return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}

// ì\235\214ì\213\235 ì¢\205ë¥\230ë¥¼ ë¬¸ì\236\220ì\227´ë¡\234
ë³\200í\231\230í\225\230ë\212\224 í\225¨ì\210\230
FoodType get_food_type(const char *type_str) {
  if (strcmp(type_str, "Korean") == 0) {
    return Korean;
  } else if (strcmp(type_str, "Chinese") == 0) {
    return Chinese;
  } else if (strcmp(type_str, "Japanese") == 0) {
    return Japanese;
  } else if (strcmp(type_str, "Western") == 0) {
    return Western;
  } else {
    return Unknown;
  }
}

// ì\235\214ì\213\235 ì¢\205ë¥\230ë¥¼ ì¶\234ë ¥í\225\230ë\212\224 í\225¨ì\210\230
void print_food_type(FoodType type) {
  switch (type) {
  case Korean:
    printf("Korean");
    break;
  case Chinese:
    printf("Chinese");
    break;
  case Japanese:
    printf("Japanese");
    break;
  case Western:
    printf("Western");
    break;
  default:
    printf("Unknown");
  }
}

// ì\213\235ë\213¹ ì \225ë³´ë¥¼ ì¶\234ë ¥í\225\230ë\212\224 í\225¨ì\210\230
void print_restaurant(Restaurant restaurant) {
  printf("Name: %s, Type: ", restaurant.name);
  print_food_type(restaurant.type);
  printf(", Location: (%.2lf, %.2lf), Score: %.1lf\n",
         restaurant.x, restaurant.y, restaurant.score);
}

// ë©\224ì\235¸ í\225¨ì\210\230
int main() {
  RestaurantManager manager;
  manager.capacity = INIT_CAPACITY;
  manager.count = 0;

  manager.restaurants = (Restaurant *)malloc(sizeof(Restaurant) *
manager.capacity);

  if (manager.restaurants == NULL) {
    printf("Memory allocation error!\n");
    exit(1);
  }

  int choice;
  while (1) {
    printf("\n--- Restaurant Management ---\n");
    printf("1. Add restaurant information\n");
    printf("2. Recommend restaurants\n");
    printf("3. Search for restaurants\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
```

```
250:
251:     if (scanf("%d", &choice) != 1) {
252:       printf("Invalid choice.\n");
253:       fflush(stdin);
254:       continue;
255:     }
256:
257:     getchar();
258:
259:     switch (choice) {
260:     case 1:
261:       add_restaurant(&manager);
262:       break;
263:     case 2:
264:       recommend_restaurants(&manager);
265:       break;
266:     case 3:
267:       search_restaurants(&manager);
268:       break;
269:     case 4:
270:       printf("Exiting program. Memory cleared.\n");
271:       free(manager.restaurants);
272:       return 0;
273:     default:
274:       printf("Invalid choice.\n");
275:     }
276:   }
277:
278:   return 0;
279: }
280:
```

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

#define MAX_LEN 20      // ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ë¬¸ì\236\220ì\227´ ìµ\234ë\214\200 ê¸¸ì\235´
#define MAX_RECORDS 100 // ì \200ì\236¥ ê°\200ë\212¥í\225\234 ìµ\234ë\214\200 ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ê°\234ì\210\230
#define KEY_LEN 10     // ì\225\224í\230¸í\231\224 í\202¤ ê¸¸ì\235´

// ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ê µ¬ì¡°ì²´ ì \225ì\235\230
typedef struct {
    char ssn[MAX_LEN];    // ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ (XXXXXX-XXXXXXX í\230\225ì\213\235)
    int is_encrypted;     // ì\225\224í\230¸í\231\224 ì\203\201í\203\234 (0: í\217\211ë¬¸, 1: ì\225\224í\230¸í\231\224)
} SSNRecord;

// ì \204ì²´ ë \210ì½\224ë\223\234 ê´\200ë¦¬ êµ¬ì¡°ì²´
typedef struct {
    SSNRecord records[MAX_RECORDS];
    int count;            // ì \200ì\236¥ë\220\234 ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ê°\234ì\210\230
} SSNManager;

// ì\225\224í\230¸í\231\224 í\202¤ ì\203\235ì\204± í\225¨ì\210\230 (ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì\225\236 12ì\236\220ë¦¬ì\231\200 ê°\200ì¤\221ì¹\230ë¥¼ ê³±í\225\234 í\225©ì\235\204 11ë¡\234 ë\202\230ë\210\210 ë\202\230ë¨¸ì§\200 ì\202¬ì\232©)
void generate_encryption_key(const char* ssn, char* key) {
    int sum = 0;
    int weights[] = {2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4, 5}; // ê°\200ì¤\221ì¹\230 ë°°ì\227´

    for (int i = 0; i < 12; i++) {
        if (ssn[i] != '-') {
            sum += (ssn[i] - '0') * weights[i];
        }
    }

    sum %= 11; // 11ë¡\234 ë\202\230ë\210\210 ë\202\230ë¨¸ì§\200

    // aë¶\200í\204° jê¹\214ì§\200ì\235\230 ì\225\214í\214\214ë²³ì\234¼ë¡\234 í\202¤ ì\203\235ì\204±
    for (int i = 0; i < KEY_LEN; i++) {
        key[i] = 'a' + ((sum + i) % 10);
    }
    key[KEY_LEN] = '\0';
}

// ì\225\224í\230¸í\231\224 í\225¨ì\210\230 (ê°\201 ì\236\220ë¦¬ì\227\220 í\202¤ê°\222ì\235\204 ë\215\224í\225\234 í\233\204 ì\225\214í\214\214ë²³ì\234¼ë¡\234 ë³\200í\231\230)
void encrypt_ssn(char* ssn, const char* key) {
    int key_idx = 0;
    int key_len = strlen(key);

    for (int i = 0; ssn[i] != '\0'; i++) {
        if (ssn[i] >= '0' && ssn[i] <= '9') {
            int num = ssn[i] - '0';
            int key_val = key[key_idx % key_len] - 'a';
            ssn[i] = 'a' + ((num + key_val) % 10);
            key_idx++;
        }
        // í\225\230ì\235´í\224\210ì\235\200 ê•¸ë\214\200ë¡\234 ì\234 ì§\200
    }
}

// ë³µí\215¸ ì\210«ì\236\220 ì\203\235ì\204± ë°\217 ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì\203\235ì\204±
void decrypt_ssn(char* encrypted_ssn, const char* key) {
    int key_idx = 0;
    int key_len = strlen(key);

    for (int i = 0; encrypted_ssn[i] != '\0'; i++) {
        if (encrypted_ssn[i] >= 'a' && encrypted_ssn[i] <= 'j') {
            int val = encrypted_ssn[i] - 'a';
            int key_val = key[key_idx % key_len] - 'a';
            int num = (val - key_val + 10) % 10; // ì\235\214ì\210\230ë°©ì§\200ë¥¼ ì\234\204í\225´ +10
            encrypted_ssn[i] = '0' + num;
            key_idx++;
        }
    }
}

// ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì\203\235ì\204± í\225¨ì\210\230 (ì\203\235ì\205\204ì\233\224ì\235¼, ì\204±ë³\204 ì\236\205ë ¥ ë°\233ì\225\204ì\203\235ì\204±)
void generate_ssn(SSNRecord* record) {
    char birth[9];
    int gender;

    printf("ì\203\235ë\205\204ì\233\224ì\235¼ ì\236\205ë ¥ (ì\230\210: 19990101): ");
    scanf("%s", birth);

    // ì\203\235ë\205\204ì\233\224ì\235¼ ì\234 í\232¨ì\204± ê²\200ì\202¬ (ê°\204ë\213¨í\225\234 ê²\200ì\202¬ë§\214 ì¶\224ê°\200)
    if (strlen(birth) != 8) {
        printf("ì\236\230ëª»ë\220\234 ì\203\235ë\205\204ì\233\224ì\235¼ í\230\225ì\213\235ì\236\205ë\213\210ë\213¤.\n");
        return;
    }

    printf("ì\204±ë³\204 ì\236\205ë ¥ (ë\202¨: 1, ì\227¬: 2): ");
    scanf("%d", &gender);

    // ì\204±ë³\204 ì\234 í\232¨ì\204± ê²\200ì\202¬
    if (gender != 1 && gender != 2) {
        printf("ì\236\230ëª»ë\220\234 ì\204±ë³\204ì\236\205ë\213\210ë\213¤.\n");
        return;
    }

    // 2000ë\205\204ë\214\200ì\203\235 ì\204±ë³\204 ì¶\224ë\223\234 ë³\200í\231\230
    int year = (birth[0] - '0') * 1000 + (birth[1] - '0') * 100;
    if (year >= 2000) {
        gender += 2;
    }

    // ë\236\234ë\215¤ ì\210«ì\236\220 ì\203\235ì\204± ë°\217 ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì\203\235ì\204±
    int random_nums[6];
    for (int i = 0; i < 6; i++) {
        random_nums[i] = rand() % 10;
    }

    sprintf(record->ssn, "%c%c%c%c%c%c-%d%d%d%d%d%d",
            birth[2], birth[3], birth[4], birth[5], birth[6], birth[7],
            gender, random_nums[0], random_nums[1], random_nums[2],
            random_nums[3], random_nums[4]);

    // ë§\210ì§\200ë§\211 ì\236\220ë¦¬ ì\210«ì\236\220 ê³\204ì\202°
```

```
(generate_encryption_key í\225¨ì\210\230 í\231\234ì\232©)
  115:     char temp_key[KEY_LEN + 1];
  116:     generate_encryption_key(record->ssn, temp_key);
  117:     int last_digit = temp_key[0] - 'a'; // í\202¤ì\235\230 ì²« ë²\210ì§¸ë¬¸ì\236\220ë¥¼ ì\210«ì\236\220ë¡\234 ë³\200í\231\230
  118:     record->ssn[14] = last_digit + '0'; // ë§\210ì§\200ë§\211ì\236\220ë¦¬ì\227\220 ì\210«ì\236\220 ì¶\224ê°\200
  119:
  120:     record->is_encrypted = 0;
  121:     printf("ì\203\235ì\204±ë\220\234 ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸: %s\n", record->ssn);
  122: }
  123:
  124:
  125: int main() {
  126:     SSNManager manager = {0};  // ê¸°ì¡´ì²´ ì´\210ê¸°í\231\224
  127:     char key[KEY_LEN + 1];     // ì\225\224í\230¸í\231\224 í\202¤
  128:     int choice;
  129:     int key_generated = 0; // ì\225\224í\230¸í\231\224 í\202¤ ì\203\235ì\204±ì\227¬ë¶\200ë¥¼ ì \200ì\236¥í\225\230ë\212\224 ë³\200ì\210\230 ì¶\224ê°\200
  130:
  131:     srand(time(NULL)); // ë\236\234ë\215¤ ì\213\234ë\223\234 ì´\210ê¸°í\231\224
  132:
  133:     while (1) {
  134:         printf("===== ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ê´\200ë¦¬ í\224\204ë¡\234ê•¸ë\236¨ =====\n");
  135:         printf("1. ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì\203\235ì\204±\n");
  136:         printf("2. ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì¡°í\232\214\n");
  137:         printf("3. ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ì\225\224í\230¸í\231\224\n");
  138:         printf("4. ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ ë³µí\230¸í\231\224\n");
  139:         printf("5. ì¢\205ë£\214\n");
  140:         printf("ë©\224ë\211´ ì\204 í\203\235: ");
  141:         scanf("%d", &choice);
  142:
  143:         switch (choice) {
  144:             case 1: {
  145:                 if (manager.count < MAX_RECORDS) {
  146:                     generate_ssn(&manager.records[manager.count]);
  147:                     manager.count++;
  148:                 } else {
  149:                     printf("ë\215\224 ì\235´ì\203\201ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ê¾¼ ì \200ì\236¥í\225  ì\210\230ì\227\206ì\212µë\213\210ë\213¤.\n");
  150:                 }
  151:                 break;
  152:             }
  153:             case 2:
  154:                 if (manager.count == 0) {
  155:                     printf("ì \200ì\236¥ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ê°\200 ì\227\206ì\212µë\213\210ë\213¤.\n");
  156:                 } else {
  157:                     for (int i = 0; i < manager.count; i++) {
  158:                         printf("ë²\210í\230¸ %d: ", i + 1);
  159:                         if (manager.records[i].is_encrypted) {
  160:                             printf("ì\225\224í\230¸í\231\224ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸: %s\n", manager.records[i].ssn);
  161:                         } else {
  162:                             printf("í\217\211ë¬¸ ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸: %s\n", manager.records[i].ssn);
  163:                         }
  164:                     }
  165:                 }
  166:                 break;
  167:             case 3: {
  168:                 if (manager.count == 0) {
  169:                     printf("ì \200ì\236¥ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ê°\200 ì\227\206ì\212µë\213\210ë\213¤.\n");
  170:                 } else {
  171:                     if (!key_generated) { // ì\225\224í\230¸í\231\224 í\202¤ê°\200 ì\203\235ì\204±ë\220\230ì§\200 ì\225\212ì\235\200 ê²½ì\232°ì\227\220ë§\214 í\202¤ ì\203\235ì\204± ë°\217 ì¶\234ë ¥
  172:                         generate_encryption_key(manager.records[0].ssn, key); // í\202¤ ì\203\235ì\204±
  173:                         printf("ì\203\235ì\204±ë\220\234 ì\225\224í\230¸í\231\224 í\202¤: %s\n", key); // í\202¤ ì¶\234ë ¥
  174:                         key_generated = 1; // í\202¤ ì\203\235ì\204± ì\227¬ë¶\200ì\227\205ë\215°ì\235´í\212¸
  175:                     }
  176:
  177:                     for (int i = 0; i < manager.count; i++) {
  178:                         if (manager.records[i].is_encrypted) {
  179:                             printf("ì\235´ë¯¸ ì\225\224í\230¸í\231\224ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ì\236\205ë\213\210ë\213¤. \n"); // ì\235´ë¯¸ ì\225\224í\230¸í\231\224ë\220\234 ê²½ì\232° ë©\224ì\213\234ì§\200 ì¶\234ë ¥
  180:                         } else {
  181:                             encrypt_ssn(manager.records[i].ssn, key);
  182:                             manager.records[i].is_encrypted = 1;
  183:                             printf("ì\225\224í\230¸í\231\224ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸: %s\n", manager.records[i].ssn);
  184:                         }
  185:                     }
  186:                 }
  187:                 break;
  188:             }
  189:             case 4: {
  190:                 if (manager.count == 0) {
  191:                     printf("ì \200ì\236¥ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ê°\200 ì\227\206ì\212µë\213\210ë\213¤.\n");
  192:                 } else {
  193:                     printf("ë³µí\230¸í\231\224í\225ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ì\235\230 ë²\210í\230¸ë¥¼ ì\236\205ë ¥í\225\230ì\204¸ì\232\224: ");
  194:                     int index;
  195:                     scanf("%d", &index);
  196:
  197:                     if (index > 0 && index <= manager.count) {
  198:                         if (manager.records[index - 1].is_encrypted) {
  199:                             decrypt_ssn(manager.records[index - 1].ssn, key);
  200:                             manager.records[index - 1].is_encrypted = 0;
  201:                             printf("ë³µí\230¸í\231\224ë\220\234ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸: %s\n", manager.records[index - 1].ssn);
  202:                         } else {
  203:                             printf("ì\225\224í\230¸í\231\224ë\220\230ì§\200ì\225\212ì\235\200 ì£¼ë¯¼ë\223±ë¡\235ë²\210í\230¸ì\236\205ë\213\210ë\213¤.\n");
  204:                         }
  205:                     } else {
  206:                         printf("ì\236\230ëª»ë\220\234ë²\210í\230¸ì\236\205ë\213\210ë\213¤.\n");
  207:                     }
  208:                 }
  209:                 break;
  210:             }
  211:             case 5:
  212:                 printf("í\224\204ë¡\234ê•¸ë\236¨ì\235\204ì¢\205ë£\214í\225©ë\213\210ë\213¤.\n");
  213:                 return 0;
  214:             default:
  215:                 printf("ì\236\230ëª»ë\220\234ì\204 í\203\235ì\236\205ë\213\210ë\213¤.\n");
  216:         }
  217:         printf("\n");
  218:     }
  219:     return 0;
  220: }
```