C프로그래밍및실습

사원관리프로그램

최종 보고서

제출일자: 2023.12.24
제출자명: 정지우
제출자 학번: 181155

1. 프로젝트 목표

1) 배경 및 필요성

대기업이나 중견기업의 경우 값비싼 사원관리 프로그램을 구매하여 사용하지만 중소기업이나 스타트업의 경우 자본금이 부족하기에 이런 사원관리 프로그램까지 구매해서 사용할 여력이 없거나 빠듯한 경우가 많다. 또한 지나치게 복잡하고 많은 기능들을 포함하고 있어 이로인한 사용에 애로사항이 발생하기도 한다. 이 문제를 해결하기 위해 C언어로도 손쉽게 만들 수 있는 사원들의 목록을 관리하고 추가 및 수정이 가능하며 급여, 입사날짜, 직책 등을 저장해서 리스트화 해주는 프로그램이 필요하다.

2) 프로젝트 목표

회사 직원들의 직책, 급여, 입사날짜 등의 기본 정보와 사원들의 휴가 등을 관리하는 사원관리 프로그램을 만드는 것을 목표로 한다.

3) 차별점

기존 사원관리 프로그램의 경우 값비싼 것은 둘째치고 개인이 따라만들기에는 매우 복잡하고 어려운 경우가 많았으며 필요없는 기능도 다수 포함되어있어 오히려 필요한 기능만 골라쓰기 힘든 경우가 많았다. 하지만 이 프로젝트로 만드는 프로그램은 누구나 C언어로 따라 만들 수 있으며 실제 사무현장에서 꼭 필요한 기능들만 포함하고 있다. 또한 불필요한 것들 없이 담백하게 필요한 정보만을 출력한다.

2. 기능 계획

1) 기능 1 : 사원 정보 관리 기능

- 설명 :현재 회사에 재직중인 사원들의 정보를 입력하는 기능이다.

(1) 세부 기능 1 : 사원 추가 시스템

- 설명 : 직책,급여,입사날짜,나이,거주지역 등을 사원별로 입력하는 기능이다.

(2)세부 기능 2: 목록 수정 기능

- 설명 : 입력된 사원 중에서 원하는 사원의 원하는 카테고리를 수정하는 기능이다. 수정을 하게되면 이전 기록은 사라지고 수정한 기록만 남게 된다.

(3)세부 기능 3: 목록 출력 기능

-설명: 사원 테이블에 입력된 사원 정보를 사원별로 모두 출력해준다.

2) 기능 2 : 퇴사자 관리 기능

- 설명 : 퇴사한 직원을 따로 목록으로 만들어 관리하는 기능이다. 직원의 퇴사를 입력하게 되면 퇴사자 명단으로 넘어가서 저장되며 퇴사자 명단을 한번에 출력해서 볼 수 있다.

2) 기능 3 : 휴가(연차) 관리 기능

(1)세부 기능 1: 휴가자 명단 관리 기능

- 설명 : 휴가를 간 직원들의 휴가 간 날짜, 휴가에서 돌아오는 날짜, 사유 등을 기록한 리스트를 보여주는 기능이다.

(2)세부 기능 2: 직원들의 연차 관리 기능

-설명 : 직원들의 남은 연차를 리스트화 해서 보여주고 수정할 수 있는 기능이다.

3. 기능 구현

1) 신규 사원 기능

(1) 사원 정보 입력 기능

[AddEmployee 함수]

<입출력>

1)입력

-사원의 이름을 구조체 배열 EmployeeTable에서 num_people번째에 employ_name 문자열에 입력 받는다.

-사원의 입사날짜를 구조체 배열 EmployeeTable에서 num_people번째에 start_company 문자열에 입력 받는다.

-사원의 직급을 구조체 배열 EmployeeTable에서 num_people번째에 role 문자열에 입력 받는다.

-사원의 나이를 구조체 배열 EmployeeTable에서 num_people번째에 age변수에 입력 받는다.

-사원의 거주지를 구조체 배열 EmployeeTable에서 num_people번째에 residence 문자열에 입력 받는다.

-사원이 한명 씩 입력 될 때 마다 num_people의 수가 증가한다.

2)출력

-출력 값은 존재하지 않는다

<설명>

추가할 사원의 나이,거주지,직급,입사날짜 등의 정보를 한명씩 입력하여 EmployeeTable이라는 구조체에 저장하여 사원의 정보가 담긴 목록을 만든다. 이 과정에서 입력된 사원의 수가 할당된 용량 capacity에 도달하면 ResizeTable이라는 사용자 정의 함수를 통해 새로운 크기의 메모리를 할당하고 기존 데이터를 새 메모리에 복사한다.

<적용된 배운 내용>

-구조체, 입출력함수, 배열, 함수, 포인터, 동적메모리

<코드 스크린샷>

```
void AddEmployee(struct EmployData *table, int *num_people) {
  ResizeTable();

  printf("사원 이름을 입력하세요: ");
  scanf_s("%s", table[*num_people].employ_name,(int)sizeof(table[*num_people].employ_name));
  printf("사원의 입사날짜를 입력하세요(ex:1900/01/01): ");
  scanf_s("%s", table[*num_people].start_company,(int)sizeof(table[*num_people].start_company));
  printf("사원의 거주지역을 입력하세요: ");
  scanf_s("%s", table[*num_people].residence,(int)sizeof(table[*num_people].residence));
  printf("사원의 직급을 입력하세요: ");
  scanf_s("%s", table[*num_people].role,(int)sizeof(table[*num_people].role));
  printf("사원의 나이를 입력하세요: ");
  scanf_s("%d", &table[*num_people].age);

  (*num_people)++;
}
```

[ResizeTable 함수]

<입출력>

1)입력

-입력값은 존재하지 않는다.

2)출력

-출력값도 존재하지 않는다.

<설명>

사원 정보를 입력하는 과정에서 ResizeTable이라는 사용자 정의 함수를 통해 입력된 사원의 수가 할당된 용량 capacity에 도달하면 realloc함수를 이용해 새로운 크기의 메모리를 할당한다.

<적용된 배운 내용>

-출력함수, 함수, 동적메모리, 조건문

<코드 스크린샷>

```
void ResizeTable() {
  if (num_people == capacity) {
    capacity = (capacity == 0) ? INITIAL_CAPACITY : capacity * 2;
    EmployeeTable = realloc(EmployeeTable, capacity * sizeof(struct EmployData));
    if (EmployeeTable == NULL) {
      printf("메모리 할당 실패₩n");
      exit(EXIT_FAILURE);
    }
  }
}
```

(2) 사원 정보 수정 기능

[FindPerson 함수]

<입출력>

1)입력

-struct EmployData *table: 사원의 정보를 담고 있는 구조체 배열에 대한 포인터를 입력 받는다.

-num_people: 배열에 현재 저장된 사원 정보의 수를 입력 받는다.

-char *name: 찾고자 하는 사원의 이름이 담긴 문자열의 포인터를 입력 받는다.

2)출력

-찾고자하는 사원의 이름을 사원의 정보를 담고 있는 구조체 배열에서 찾으면 몇 번째에 위치했는지 그 i번째 값을 출력한다.

-찾고자하는 사원의 이름이 사원의 정보를 담고 있는 구조체 배열에 존재하지 않으면 무조건 -1을 출력한다.

<설명>

수정하고자 하는 사원의 이름을 입력하면 사원테이블에서 해당 사원을 찾아서 해당사원의 인덱스를 반환해준다. 해당 사원이 존재하지 않을 경우 잘못된 이름이 입력되었다는 메시지가 출력된다.

<적용된 배운 내용>

-조건문(switch), 구조체, 입출력함수, 배열, 함수, 포인터, 동적메모리

<코드 스크린샷>

```
int FindPerson(const struct EmployData *table, int num_people,
               const char *name) {
  for (int i = 0; i < num_people; ++i) {
    if (strcmp(table[i].employ_name, name) == 0) {
      return i;
    }
  }
  return -1;  // 찾지 못한 경우 -1 반환
}
```

[UpdateEmployee 함수]

<입출력>

1)입력

-struct EmployData *table: 사원의 정보를 가지고 있는 구조체 배열의 포인터를 입력 받는다. 이 배열에서 FindPerson함수를 통해 정보를 변경하고자 하는 사원을 찾는다

- num_people: 배열에 현재 저장된 사원 정보의 수를 입력 받는다.

-update_choice: 제시된 4개의 수정 가능 항목 중 수정할 항목의 번호를 입력받는다.

-수정할 항목을 고른 후 수정값을 입력하면 각각 항목에 맞게 사원 구조체 배열의 age,startcompany,role,residence에 수정값이 입력된다.

-editing: 초기 값은 1로 되어있어서 while문을 무한 반복하게 해주며 종료 값을 입력하면 0으로 바뀌고 while반복문을 종료한다.

2)출력

-출력값은 존재하지 않는다

<설명>

수정하고자 하는 사원 이름을 알맞게 입력했으면 수정할 항목을 고르는 메시지가 출력되고 수정을 원하는 항목을 골라서 수정 값을 입력하면 해당 사원의 정보가 수정이 된다. 이때 종료를 의미하는 5를 입력하기 전까지 원하는 만큼 사원의 정보를 수정할 수 있다.

<적용된 배운 내용>

-조건문(switch), 구조체, 입출력함수, 배열, 함수, 포인터, 동적메모리

<코드 스크린샷>

```
void UpdateEmployee(struct EmployData *table, int num_people) {
  char TargetName[50];
  printf("₩n수정할 사람의 이름을 입력하세요: ");
  scanf_s("%s", TargetName, (int)sizeof(TargetName));

  int peopleIndex = FindPerson(table, num_people, TargetName);

  if (peopleIndex == -1) {
    printf("잘못된 이름을 입력했습니다.₩n");
    return;  // 처음 메뉴 선택으로 돌아감
  }
  int editing = 1;
  while (editing) {
    int update_choice;
    printf("-----------------------------₩n");
    printf("수정할 항목을 선택하세요:₩n");
    printf("1. 나이₩n2. 거주지₩n3. 직급₩n4. 근무 연도₩n5. 종료₩n");
    printf("-----------------------------₩n");
    printf("메뉴: ");
    scanf_s("%d", &update_choice);

    switch (update_choice) {
      case 1:
        printf("새로운 나이 입력: ");
        scanf_s("%d", &table[peopleIndex].age);
        break;
      case 2:
        printf("새로운 거주지 입력: ");
        scanf_s("%s", table[peopleIndex].residence,(int)sizeof(table[peopleIndex].residence));
        break;
      case 3:
        printf("새로운 직급 입력: ");
        scanf_s("%s", table[peopleIndex].role,(int)sizeof(table[peopleIndex].role));
        break;
      case 4:
        printf("새로운 근무 연도 입력: ");
        scanf_s("%s", table[peopleIndex].start_company,(int)sizeof(table[peopleIndex].start_company));
        break;
      case 5:
        printf("수정된 사원 정보:₩n");
        printf("이름: %s₩n", table[peopleIndex].employ_name);
        printf("나이: %d₩n", table[peopleIndex].age);
        printf("입사 날짜: %s₩n", table[peopleIndex].start_company);
        printf("직급: %s₩n", table[peopleIndex].role);
        printf("거주지: %s₩n", table[peopleIndex].residence);
        editing = 0;
        break;
      default:
        printf("잘못된 선택입니다.₩n");
    }
  }
}
```

(3) 사원 정보 출력 기능

[PrintEmployee 함수]

<입출력>

1)입력

-struct EmployData *table: 사원의 정보를 담고 있는 구조체 배열에 대한 포인터를 입력 받는다.

- num_people: 배열에 현재 저장된 사원 정보의 수를 입력 받는다.

2)출력

-반환값은 존재하지 않으며 사원테이블에 입력된 사원별로 사원 정보를 모두 출력해준다.

<설명>

입력된 나이, 직급, 거주지, 입사날짜 등의 정보를 사원별로 모든 사원에 대해 출력해준다.

<적용된 배운 내용>

-구조체, 출력함수, 배열, 함수, 포인터

<코드 스크린샷>

```
void PrintEmployee(const struct EmployData *table, int num_people) {
  for (int i = 0; i < num_people; i++) {
    printf("------------------------------₩n");
    printf("이름: %s₩n", table[i].employ_name);
    printf("나이: %d₩n", table[i].age);
    printf("입사 날짜: %s₩n", table[i].start_company);
    printf("직급: %s₩n", table[i].role);
    printf("거주지: %s₩n", table[i].residence);
    printf("------------------------------₩n");
  }
}
```

(4) 기능1 함수 헤더파일로 만들기

[employee.h]

<입출력>

1)입력

-헤더파일에는 입력 값이 존재하지 않음

2)출력

-헤더파일에는 출력 값이 존재하지 않음

<설명>

기능1에서 구현된 EmployData 구조체,변수(num_people,capacity),FindPerson/ResizeTable/

AddEmployee/UpdateEmployee/PrintEmployee/Freememory 함수를 전부 헤더파일로 만들어서 선언해 소스파일의 관리 등을 용이하도록 만들었다. 또한 #ifndef, #endif라는 C 프리프로세서 지시문을 통해 헤더파일이 중복해서 포함되는 것을 방지하였다.

<적용된 배운 내용>

-구조체,함수,포인터,배열과 포인터,헤더파일,상수와변수

<코드 스크린샷>

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

#ifndef MAINHEADER_H
#define MAINHEADER_H

#define INITIAL_CAPACITY 10  // 초기 구조체 배열 메모리 할당 크기

struct EmployData {
    char employ_name[50];
    char start_company[30];
    char residence[50];
    char role[20];
    int age;
};  // 사원의 정보를 저장할 구조체 정의

extern struct EmployData *EmployeeTable;
extern int num_people;  // 입력된 사원 수를 의미하는 num_people
extern int capacity ;  // 동적으로 할당된 Employee 데이터를 저장하는 배열의 용량을 의미하는 변수 선언

int FindPerson(const struct EmployData *table, int num_people,
               const char *name);
void ResizeTable();
void AddEmployee(struct EmployData *table, int *num_people);
void UpdateEmployee(struct EmployData *table, int num_people);
void PrintEmployee(const struct EmployData *table, int num_people);
void FreeMemory();

#endif
```

[employee.c]

<입출력>

1)입력

-기능1의 함수들을 그대로 사용하여서 기능1의 모든 함수들과 동일한 내용이다.

2)출력

-기능1의 함수들을 그대로 사용하여서 기능1의 모든 함수들과 동일한 내용이다.

<설명>

employee.h에서 선언된 EmployData 구조체,변수(num_people,capacity),FindPerson/ResizeTable/AddEmployee/UpdateEmployee/PrintEmployee/Freememory 함수들을 구현해주었다.

<적용된 배운 내용>

-구조체,함수,포인터,배열과 포인터,문자열,조건문,반복문,입출력함수,변수,헤더파일

<코드 스크린샷>

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "employee.h"

struct EmployData *EmployeeTable = NULL;
num_people = 0;
capacity = 0;

int FindPerson(const struct EmployData *table, int num_people,
               const char *name) {
  for (int i = 0; i < num_people; ++i) {
    if (strcmp(table[i].employ_name, name) == 0) {
      return i;  // 찾으면 해당 i값을 반환
    }
  }
  return -1;  // 찾지 못한 경우 -1 반환
}  // 사원 정보가 입력된 구조체 배열에서 입력한 이름의 사원을 찾아주는 함수
   // 해당 사원을 찾았으면 i값을 반환하며 찾지못했으면 -1을 반환한다.

void ResizeTable() {
  if (num_people == capacity) {
    capacity = (capacity == 0) ? INITIAL_CAPACITY : capacity * 2;
    EmployeeTable =
        realloc(EmployeeTable, capacity * sizeof(struct EmployData));
    if (EmployeeTable == NULL) {
      printf("메모리 할당 실패₩n");
      exit(EXIT_FAILURE);
    }
  }
}  // 구조체 배열의 크기를 동적으로 조절하는 함수
   // 배열에 새로운 데이터를 추가하기 전에 배열 크기가 충분한지 확인하고 필요한
   // 경우 배열의 크기를 늘린다.
```

```c
void AddEmployee(struct EmployData *table, int *num_people) {
  ResizeTable();  // 데이터를 입력하기전 배열 크기 충분한지 확인하고 필요한 경우
                  // 배열 크기 조정

  printf("사원 이름을 입력하세요: ");
  scanf_s("%s", table[*num_people].employ_name,
          (int)sizeof(table[*num_people].employ_name));
  printf("사원의 입사날짜를 입력하세요(ex:1900/01/01): ");
  scanf_s("%s", table[*num_people].start_company,
          (int)sizeof(table[*num_people].start_company));
  printf("사원의 거주지역을 입력하세요: ");
  scanf_s("%s", table[*num_people].residence,
          (int)sizeof(table[*num_people].residence));
  printf("사원의 직급을 입력하세요: ");
  scanf_s("%s", table[*num_people].role, (int)sizeof(table[*num_people].role));
  printf("사원의 나이를 입력하세요: ");
  scanf_s("%d", &table[*num_people].age);

  (*num_people)++;
}  // 구조체 배열에 사원 정보를 입력하는 함수
```

```c
void UpdateEmployee(struct EmployData *table, int num_people) {
  char TargetName[50];  // 정보를 수정할 사원의 이름을 입력받아 저장하는 문자열
                        // 선언
  printf("₩n수정할 사람의 이름을 입력하세요: ");
  scanf_s("%s", TargetName,
          (int)sizeof(TargetName));  // 수정하고자 하는 사원의 이름을 입력 받음

  int peopleIndex = FindPerson(table, num_people, TargetName);

  if (peopleIndex == -1) {
    printf("잘못된 이름을 입력했습니다.₩n");
    return;  // 처음 메뉴 선택으로 돌아감
  }  // 해당 사원을 못찾으면 잘못된 이름을 입력했다는 문구와 함께 처음 메뉴
     // 선택으로 돌아감
  int editing = 1;  // while문을 무한 반복할 수 있게 초기값을 1로 선언
  while (editing) {
    int update_choice;  // 수정할 항목의 번호를 입력받아 저장하는 변수 선언
    printf("-----------------------------₩n");
    printf("수정할 항목을 선택하세요:₩n");
    printf("1. 나이₩n2. 거주지₩n3. 직급₩n4. 근무 연도₩n5. 종료₩n");
    printf("-----------------------------₩n");
    printf("메뉴: ");
    scanf_s("%d",
            &update_choice);  // 수정하고자 하는 항목의 번호를 입력받아 저장함

    switch (update_choice) {
      case 1:  // 나이 수정
        printf("새로운 나이 입력: ");
        scanf_s("%d", &table[peopleIndex].age);
        break;
      case 2:  // 거주지 수정
        printf("새로운 거주지 입력: ");
        scanf_s("%s", table[peopleIndex].residence,
                (int)sizeof(table[peopleIndex].residence));
        break;
      case 3:  // 직급 수정
        printf("새로운 직급 입력: ");
        scanf_s("%s", table[peopleIndex].role,
                (int)sizeof(table[peopleIndex].role));
        break;
      case 4:  // 근무 연도 수정
        printf("새로운 근무 연도 입력: ");
        scanf_s("%s", table[peopleIndex].start_company,
                (int)sizeof(table[peopleIndex].start_company));
        break;
      case 5:  // 수정 기능을 종료하기전에 수정한 사원의 정보를 출력해주고 종료
        printf("수정된 사원 정보:₩n");
        printf("이름: %s₩n", table[peopleIndex].employ_name);
        printf("나이: %d₩n", table[peopleIndex].age);
        printf("입사 날짜: %s₩n", table[peopleIndex].start_company);
        printf("직급: %s₩n", table[peopleIndex].role);
        printf("거주지: %s₩n", table[peopleIndex].residence);
        editing = 0;
        break;
      default:
        printf("잘못된 선택입니다.₩n");
    }
  }
}
```

```c
void PrintEmployee(const struct EmployData *table, int num_people) {
  for (int i = 0; i < num_people; i++) {
    printf("-----------------------------₩n");
    printf("이름: %s₩n", table[i].employ_name);
    printf("나이: %d₩n", table[i].age);
    printf("입사 날짜: %s₩n", table[i].start_company);
    printf("직급: %s₩n", table[i].role);
    printf("거주지: %s₩n", table[i].residence);
    printf("-----------------------------₩n");
  }
}  // 입력된 사원의 정보를 모두 사원별로 출력해주는 함수

void FreeMemory() { free(EmployeeTable); }
```

2) 퇴사 관리 기능

(1) 퇴사자 명단 추가 기능

[ResizeResignedTable 함수]

<입출력>

1)입력

-입력값은 존재하지 않는다.

2)출력

-출력값도 존재하지 않는다.

<설명>

사원 정보를 입력하는 과정에서 ResizeTable이라는 사용자 정의 함수를 통해 입력된 사원의 수가 할당된 용량 capacity에 도달하면 realloc함수를 이용해 새로운 크기의 메모리를 할당한다.

<적용된 배운 내용>

-출력함수, 함수, 동적메모리, 조건문

<코드 스크린샷>

```
void ResizeResignedTable() {
    if (num_resigned == resigned_capacity) {
        resigned_capacity =
            (resigned_capacity == 0) ? INITIAL_CAPACITY : resigned_capacity * 2;
        ResignedTable =
            realloc(ResignedTable, resigned_capacity * sizeof(struct ResignedData));
        if (ResignedTable == NULL) {
            printf("메모리 할당 실패₩n");
            exit(EXIT_FAILURE);
        }
    }
}
```

[MoveToResigned함수]

<입출력>

1)입력

-struct EmployData *table: 사원의 정보를 담고 있는 구조체 배열에 대한 포인터를 입력 받는다.

-num_people: 배열에 현재 저장된 사원 정보의 수를 입력 받는다.

-TargetName: 찾고자하는 사원의 이름을 입력받아 저장하고 FIndPerson함수에 그 이름을 입력한다.

2)출력

-FindPerson함수에 찾고자하는 사원의 이름을 TargetName에 저장해 입력하면 사원의 정보를 담고 있는 구조체 배열에서 찾으면 몇 번째 에 위치했는지 그 i번째 값을 출력한다.

<설명>

사원 명단에 입력된 사원 중 퇴사자 명단으로 이동하고 싶은 사원의 이름을 입력하면 FindPerson함수를 이용하여 그 사원의 정보를 찾고 퇴사자 명단에 복사하여 저장한다. 또한 그와 동시에 기존 사원 명당에서는 해당 사원의 정보를 삭제한다. 또한 그 과정에서 퇴사자 명단에 동적 메모리를 할당해준다.

<적용된 배운 내용>

-조건문,함수,문자열,입출력함수,상수와변수,포인터,배열과포인터

<코드 스크린샷>

```
void MoveToResigned(struct EmployData *table, int *num_people) {
  char TargetName[50];
  printf("₩n퇴사한 사람의 이름을 입력하세요: ");
  scanf_s("%s", TargetName, (int)sizeof(TargetName));

  int peopleIndex = FindPerson(table, *num_people, TargetName);

  if (peopleIndex == -1) {
    printf("잘못된 이름을 입력했습니다.₩n");
    return;
  }

  // 퇴사자 명단으로 이동
  ResizeResignedTable();
  strcpy_s(ResignedTable[num_resigned].employ_name,
           sizeof(ResignedTable[num_resigned].employ_name),
           table[peopleIndex].employ_name);
  strcpy_s(ResignedTable[num_resigned].start_company,
           sizeof(ResignedTable[num_resigned].start_company),
           table[peopleIndex].start_company);
  strcpy_s(ResignedTable[num_resigned].residence,
           sizeof(ResignedTable[num_resigned].residence),
           table[peopleIndex].residence);
  strcpy_s(ResignedTable[num_resigned].role,
           sizeof(ResignedTable[num_resigned].role), table[peopleIndex].role);
  ResignedTable[num_resigned].age = table[peopleIndex].age;


  printf("퇴사한 사람의 퇴사 날짜를 입력하세요(ex: 1900/01/01): ");
  scanf_s("%s", ResignedTable[num_resigned].resignation_date,
          (int)sizeof(ResignedTable[num_resigned].resignation_date)); //퇴사날짜 입력하여 정보추가

  num_resigned++;

  // 기존 사원 명단에서 삭제
  for (int i = peopleIndex; i < *num_people - 1; i++) {
    strcpy_s(table[i].employ_name, sizeof(table[i].employ_name),
             table[i + 1].employ_name);
    strcpy_s(table[i].start_company, sizeof(table[i].start_company),
             table[i + 1].start_company);
    strcpy_s(table[i].residence, sizeof(table[i].residence),
             table[i + 1].residence);
    strcpy_s(table[i].role, sizeof(table[i].role), table[i + 1].role);
    table[i].age = table[i + 1].age;
  }
  (*num_people)--;

  printf("%s씨는 퇴사처리 되었습니다.₩n", TargetName);
}
```

(2) 퇴사자 명단 보여주기 기능

[PrintResigned함수]

<입출력>

1)입력

-struct Resigned *table: 퇴사한 사원의 정보를 담고 있는 구조체 배열에 대한 포인터를 입력 받는다.

- num_people: 배열에 현재 저장된 사원 정보의 수를 입력 받는다.

2)출력

- 반환값은 존재하지 않으며 사원테이블에 입력된 사원별로 사원 정보를 모두 출력해준다.

<설명>

입력된 나이, 직급, 거주지, 입사날짜 등의 정보를 퇴사한 사원별로 모든 사원에 대해 출력해준다.

<적용된 배운 내용>

-구조체, 출력함수, 배열, 함수, 포인터

<코드 스크린샷>

```
void PrintResigned(const struct ResignedData *table, int num_resigned) {
  for (int i = 0; i < num_resigned; i++) {
    printf("------------------------------₩n");
    printf("이름: %s₩n", table[i].employ_name);
    printf("나이: %d₩n", table[i].age);
    printf("입사 날짜: %s₩n", table[i].start_company);
    printf("퇴사 날짜: %s₩n", table[i].resignation_date);
    printf("직급: %s₩n", table[i].role);
    printf("거주지: %s₩n", table[i].residence);
    printf("------------------------------₩n");
  }
} // 입력된 퇴사한 사원의 정보를 사원별로 모두 출력해주는 함수
```

(3) 퇴사시 연차데이터 삭제기능

[RemoveLeaveEntry함수]

<입출력>

1)입력

-char *employeeName: 퇴사하는 사원의 이름이 함수에 입력값으로 들어간다.

2)출력

- 반환값은 존재하지 않는다.

<설명>

사원이 퇴사를 하면 사원 명단에서 정보를 삭제함과 동시에 연차 테이블에서도 해당사원의 정보를 모두 삭제해준다.

<적용된 배운 내용>

-함수, 포인터와 배열, 반복문, 변수

<코드 스크린샷>

```
void RemoveLeaveEntry(const char *employeeName) {
  for (int i = 0; i < num_leave; ++i) {
    if (strcmp(LeaveTable[i].employee_name, employeeName) == 0) {
      // 해당 사원의 정보를 찾으면 삭제
      for (int j = i; j < num_leave - 1; ++j) {
        strcpy_s(LeaveTable[j].employee_name,
                 sizeof(LeaveTable[j].employee_name),
                 LeaveTable[j + 1].employee_name);
        LeaveTable[j].total_leave_days = LeaveTable[j + 1].total_leave_days;
        LeaveTable[j].remaining_leave_days =
            LeaveTable[j + 1].remaining_leave_days;
      }
      --num_leave;  // 연차 구조체의 총 사원 수를 감소
      break;
    }
  }
}
```

(4) 종료 시 동적 메모리 해제 기능

[FreeMemory_Employee/ FreeMemory_ResignedEmployee 함수]

<입출력>

1)입력

-동적 메모리 할당을 해제할 구조체 입력

2)출력

- 반환값은 존재하지 않는다.

<설명>

동적으로 할당된 메모리를 해제하고 프로그램을 종료시켜서 메모리 누수를 방지하고 자원을 효율적으로 관리할 수 있게 해준다.

<적용된 배운 내용>

-함수

<코드 스크린샷>

```
void FreeMemory_ResignedEmployee() { free(ResignedTable); }; //동적할당 메모리 해제하여 메모리 누수 방지
```

```
void FreeMemory_Employee() { free(EmployeeTable); }//동적할당 메모리 해제하여 메모리 누수 방지
```

```
case 9:
  FreeMemory_Employee();
  FreeMemory_ResignedEmployee();
  printf("프로그램을 종료합니다.");
  return 0;  // 종료
```

(5) 기능2 함수 헤더파일로 만들기

[resigned_employee.h]

<입출력>

1)입력

-헤더파일에는 입력 값이 존재하지 않음

2)출력

-헤더파일에는 출력 값이 존재하지 않음

<설명>

기능2에서 구현된 ResignedData 구조체,변수(num_resigned,capacity),MoetoResigned/ResizeResignedTaBle/Freememory_ResignedEmployee 함수를 전부 헤더파일로 만들어서 선언해 소스파일의 관리 등을 용이하도록 만들었다. 또한 #ifndef, #endif라는 C 프리 프로세서 지시문을 통해 헤더파일이 중복해서 포함되는 것을 방지하였다.

<적용된 배운 내용>

-구조체,함수,포인터,배열과 포인터, 헤더파일,상수와변수

<코드 스크린샷>

```
#pragma once
#ifndef RESIGNED_MANAGEMENT_H
#define RESIGNED_MANAGEMENT_H

#include "employee.h"  // Assuming your employee management functions are in "employee.h"

struct ResignedData {
    char employ_name[50];
    char start_company[30];
    char resignation_date[30];
    char residence[50];
    char role[20];
    int age;
};

extern struct ResignedData *ResignedTable;
extern int num_resigned;
extern int resigned_capacity;

void ResizeResignedTable();
void MoveToResigned(struct EmployData *table, int *num_people);
void PrintResigned(const struct ResignedData *table, int num_resigned);
void FreeMemory_ResignedEmployee();

#endif
```

[resigned_employee.c]

<입출력>

1)입력

-기능2의 함수들을 그대로 사용하여서 기능2의 모든 함수들과 동일한 내용이다.

2)출력

-기능2의 함수들을 그대로 사용하여서 기능2의 모든 함수들과 동일한 내용이다.

<설명>

Resigned_employee.h에서 선언한ResignedData 구조체,변수(num_resigned,capacity),Move toResigned/ResizeResignedTaBle/Freememory_ResignedEmployee 함수등을 구현해주었다.

<적용된 배운 내용>

-구조체,함수,포인터,배열과 포인터,문자열,조건문,반복문,입출력함수,변수,헤더파일

<코드 스크린샷>

```c
#include "resigned_employee.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "employee.h"

struct ResignedData *ResignedTable = NULL;
int num_resigned = 0;
int resigned_capacity = 0;

void ResizeResignedTable() {
    if (num_resigned == resigned_capacity) {
        resigned_capacity =
            (resigned_capacity == 0) ? INITIAL_CAPACITY : resigned_capacity * 2;
        ResignedTable =
            realloc(ResignedTable, resigned_capacity * sizeof(struct ResignedData));
        if (ResignedTable == NULL) {
            printf("메모리 할당 실패₩n");
            exit(EXIT_FAILURE);
        }
    }
}  // 구조체 배열의 크기를 동적으로 조절하는 함수
```

```c
void PrintResigned(const struct ResignedData *table, int num_resigned) {
    for (int i = 0; i < num_resigned; i++) {
        printf("-------------------------------₩n");
        printf("이름: %s₩n", table[i].employ_name);
        printf("나이: %d₩n", table[i].age);
        printf("입사 날짜: %s₩n", table[i].start_company);
        printf("퇴사 날짜: %s₩n", table[i].resignation_date);
        printf("직급: %s₩n", table[i].role);
        printf("거주지: %s₩n", table[i].residence);
        printf("-------------------------------₩n");
    }
}  // 입력된 퇴사한 사원의 정보를 사원별로 모두 출력해주는 함수

void FreeMemory_ResignedEmployee() {
    free(ResignedTable);
};  // 동적할당 메모리 해제하여 메모리 누수 방지
```

```c
void MoveToResigned(struct EmployData *table, int *num_people) {
  char TargetName[50];
  printf("₩n퇴사한 사람의 이름을 입력하세요: ");
  scanf_s("%s", TargetName, (int)sizeof(TargetName));

  int peopleIndex = FindPerson(table, *num_people,
                               TargetName);  // 사원 테이블에서 해당 사원 찾기

  if (peopleIndex == -1) {
    printf("잘못된 이름을 입력했습니다.₩n");
    return;
  }  // 못찾을시 못찾았다는 문구 출력

  // 퇴사자의 사원 명단 데이터를 퇴사자 명단으로 이동
  ResizeResignedTable();
  strcpy_s(ResignedTable[num_resigned].employ_name,
           sizeof(ResignedTable[num_resigned].employ_name),
           table[peopleIndex].employ_name);
  strcpy_s(ResignedTable[num_resigned].start_company,
           sizeof(ResignedTable[num_resigned].start_company),
           table[peopleIndex].start_company);
  strcpy_s(ResignedTable[num_resigned].residence,
           sizeof(ResignedTable[num_resigned].residence),
           table[peopleIndex].residence);
  strcpy_s(ResignedTable[num_resigned].role,
           sizeof(ResignedTable[num_resigned].role), table[peopleIndex].role);
  ResignedTable[num_resigned].age = table[peopleIndex].age;

  printf("퇴사한 사람의 퇴사 날짜를 입력하세요(ex: 1900/01/01): ");
  scanf_s("%s", ResignedTable[num_resigned].resignation_date,
          (int)sizeof(ResignedTable[num_resigned]
                          .resignation_date));  // 퇴사날짜 입력하여 정보추가

  num_resigned++;

  // 기존 사원 명단에서 삭제
  for (int i = peopleIndex; i < *num_people - 1; i++) {
    strcpy_s(table[i].employ_name, sizeof(table[i].employ_name),
             table[i + 1].employ_name);
    strcpy_s(table[i].start_company, sizeof(table[i].start_company),
             table[i + 1].start_company);
    strcpy_s(table[i].residence, sizeof(table[i].residence),
             table[i + 1].residence);
    strcpy_s(table[i].role, sizeof(table[i].role), table[i + 1].role);
    table[i].age = table[i + 1].age;
  }
  (*num_people)--;

  printf("%s씨는 퇴사처리 되었습니다.₩n", TargetName);
}
```

3) 휴가 관리 기능

(1)휴가자 명단 관리 기능

[ResizeVacationTable함수]

<입출력>

1)입력

-입력값은 존재하지 않는다.

2)출력

-출력값도 존재하지 않는다.

<설명>

휴가자 정보를 입력하는 과정에서 ResizeVacationTable이라는 사용자 정의 함수를 통해 입력된 사원의 수가 할당된 용량에 도달하면 realloc함수를 이용해 새로운 크기의 메모리를 할당한다.

<적용된 배운 내용>

-출력함수, 함수, 동적메모리, 조건문

<코드 스크린샷>

```
void ResizeVacationTable() {
    if (num_vacation == vacation_capacity) {
        vacation_capacity =
            (vacation_capacity == 0) ? VACATION_CAPACITY : vacation_capacity * 2;
        VacationTable =
            realloc(VacationTable, vacation_capacity * sizeof(struct VacationData));
        if (VacationTable == NULL) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
    }
}
```

[ManageVacation함수]

<입출력>

1)입력

-employee_name:사원 명단에 존재하는 사람 중에서 휴가자 이름을 입력받는다

-start_date: 휴가 출발날짜를 사용자로부터 입력받음

-return_date: 휴가 복귀날짜를 사용자로부터 입력받음

-duration: 휴가기간 일수를 사용자로부터 입력받음

-remainingLeaveDays: 연차 테이블의 총 연차일수에서 휴가기간 일수 뺀 값이 자동으로 입력됨

2)출력

-remainingLeaveDays가 0보다 작을시 남은 연차일수가 부족하다는 문구가 출력됨

-ManageVacation함수에서 요구하는 모든 정보가 저장되면 휴가정보가 저장이 완료되었다는 문구가 출력됨

<설명>

사용자로부터 휴가 출발일, 휴가 복귀일, 휴가기간 일수를 입력받아 휴가 구조체에 저장하며 휴가 구조체에서 이름 부분은 기능1의 신규사원 추가기능이 실행되면 자동으로 이름이 같이 입력된다. 또한 휴가기간 일수가 입력되면 자동으로 연차 구조체의 남은 연차일수를 총 연차일수-휴가기간 일수로 계산해서 입력되게 해준다.

<적용된 배운 내용>

-함수, 조건문, 입출력함수, 문자열, 자료형, 상수와 변수

<코드 스크린샷>

```
void ManageVacation() {
  char employee_name[50];
  printf("휴가자의 이름을 입력하세요: ");
  scanf_s("%s", employee_name, (int)sizeof(employee_name));

  int employeeIndex = FindPerson(EmployeeTable, num_people, employee_name);

  if (employeeIndex == -1) {
    printf("해당 사원은 존재하지 않습니다.₩n");
    return;
  }

  ResizeVacationTable();

  printf("휴가 출발일을 입력하세요 (ex: 1900/01/01): ");
  scanf_s("%s", VacationTable[num_vacation].start_date,
          (int)sizeof(VacationTable[num_vacation].start_date));

  printf("휴가 복귀일을 입력하세요 (ex: 1900/01/01): ");
  scanf_s("%s", VacationTable[num_vacation].return_date,
          (int)sizeof(VacationTable[num_vacation].return_date));

  printf("휴가 기간을 입력하세요 (일수): ");
  scanf_s("%d", &VacationTable[num_vacation].duration);

  strcpy_s(VacationTable[num_vacation].employee_name,
           sizeof(VacationTable[num_vacation].employee_name), employee_name);

   int remainingLeaveDays =
      LeaveTable[num_vacation].total_leave_days - VacationTable[num_vacation].duration;

  if (remainingLeaveDays < 0) {
    printf("남은 연차일수가 부족합니다.₩n");
    return;
  }

  // 연차 테이블 업데이트
  LeaveTable[num_vacation].remaining_leave_days = remainingLeaveDays;

  num_vacation++;

  printf("%s님의 휴가 정보가 저장되었습니다.₩n", employee_name);
}
```

[PrintVacation함수]

<입출력>

1)입력

-struct VacationData *table: 휴가자의 정보를 담고 있는 구조체 배열에 대한 포인터를 입력 받는다.

- num_vacation: 배열에 현재 저장된 휴가자 정보의 수를 입력 받는다.

2)출력

-반환값은 존재하지 않으며 사원테이블에 입력된 사원별로 사원 정보를 모두 출력해준다.

<설명>

휴가 구조체에 입력된 휴가를 간 사원들의 이름, 휴가 출발일, 휴가 복귀일, 휴가기간 일수를 사원별로 전부 출력해준다.

<적용된 배운 내용>

-구조체, 출력함수, 배열, 함수, 포인터

<코드 스크린샷>

```
void PrintVacation(const struct VacationData *table, int num_vacation) {
  for (int i = 0; i < num_vacation; i++) {
    printf("------------------------------₩n");
    printf("이름: %s₩n", table[i].employee_name);
    printf("휴가 출발일: %s₩n", table[i].start_date);
    printf("휴가 복귀일: %s₩n", table[i].return_date);
    printf("휴가 기간: %d 일₩n", table[i].duration);
    printf("------------------------------₩n");
  }
}
```

[ReturnFromVacation함수]

<입출력>

1)입력

-TargetName_Vacation: 휴가자 목록에 저장된 사원중에 찾을 사원의 이름을 입력받아 저장하는 변수이다.

-index: FindPerson함수로 찾은 사원의 인덱스 번호를 저장한다.

2)출력

-휴가 복귀처리 할 사원의 이름을 입력하면 휴가 복귀처리가 되었다는 문구가 출력되고 해당 사원이 휴가 명단에 없으면 없는 사원이라는 문구가 출력된다.

<설명>

휴가자 명단에 존재하는 휴가간 사원에 대해서 복귀했을 경우 해당 사원의 이름을 입력하면 휴가자 명단에서 해당사원의 이름, 휴가출발일, 휴가복귀일, 휴가기간 일수 등의 모든 데이터를 삭제하고 휴가 복귀처리가 되었다는 메시지가 출력된다.

<적용된 배운 내용>

-포인터와 배열, 함수, 입출력함수, 조건문

<코드 스크린샷>

```
void ReturnFromVacation(const char *name) {
  int TargetName_Vacation[50];
  printf("₩n휴가 복귀한 사원의 이름을 입력하세요: ");
  scanf_s("%s", TargetName_Vacation, (int)sizeof(TargetName_Vacation));
  int index = FindPerson(VacationTable, num_vacation, TargetName_Vacation);
  if (index != -1) {
    // 휴가 명단에서 해당 사원의 정보를 삭제
    for (int i = index; i < num_vacation - 1; ++i) {
      strcpy_s(VacationTable[i].employee_name,
               sizeof(VacationTable[i].employee_name),
               VacationTable[i + 1].employee_name);
      strcpy_s(VacationTable[i].start_date, sizeof(VacationTable[i].start_date),
               VacationTable[i + 1].start_date);
      strcpy_s(VacationTable[i].return_date, sizeof(VacationTable[i].return_date),
               VacationTable[i + 1].return_date);
      VacationTable[i].duration = VacationTable[i + 1].duration;
    }
    --num_vacation;
    printf("%s님의 휴가 복귀 처리가 완료되었습니다.₩n", name);
  } else {
    printf("%s님은 휴가 명단에 없는 사원입니다. 복귀 처리를 할 수 없습니다.₩n",
           name);
  }
}
```

(2)연차 관리 기능

[initializeAnnualLeave함수]

<입출력>

1)입력

-char *name:앞서 소개했던 신규 사원 기능에서 신규사원 추가할 때 입력되는 이름이 InitializeAnnualLeave함수에도 같이 입력된다.

2)출력

-출력값은 없다.

<설명>

신규사원을 추가해서 해당 사원의 정보를 사원 명단에 입력하게 되면 그와 동시에 연차 테이블에도 해당 사원의 이름이 자동으로 입력되고 해당사원의 총 연차일수와 남은 연차일수가 자동으로 0으로 초기화된다.

<적용된 배운 내용>

-함수,구조체

<코드 스크린샷>

```
void InitializeAnnualLeave(const char *name) {
    ResizeLeaveTable();
    strcpy_s(LeaveTable[num_leave].employee_name,
             sizeof(LeaveTable[num_leave].employee_name), name);
    LeaveTable[num_leave].total_leave_days = 0;       // 총 연차일수 초기화
    LeaveTable[num_leave].remaining_leave_days = 0;   // 남은 연차일수 초기화
    num_leave++;
}
```

[PrintAnnualLeave함수]

<입출력>

1)입력

-입력값은 존재하지 않는다

2)출력

-반환값은 존재하지 않으며 출력값으로 연차테이블의 모든 항목별 데이터를 사원별로 출력해준다.

<설명>

연차 테이블에 있는 이름, 총 연차일수, 남은 연차일수의 모든 데이터를 사원별로 출력해준다.

<적용된 배운 내용>

-구조체, 출력함수, 배열, 함수, 포인터, 반복문

<코드 스크린샷>

```
void PrintAnnualLeave() {
  printf("사원들의 연차 명단:₩n");
  for (int i = 0; i < num_leave; ++i) {
    printf("-----------------------------₩n");
    printf("이름: %s₩n", LeaveTable[i].employee_name);
    printf("총 연차일수: %d₩n", LeaveTable[i].total_leave_days);
    printf("남은 연차일수: %d₩n", LeaveTable[i].remaining_leave_days);
    printf("-----------------------------₩n");
  }
}
```

[SetTotalLeaveDays함수]

<입출력>

1)입력

-total_days: 총 연차일수를 사용자로부터 입력받아 구조체에 저장한다.

2)출력

-입력받은 총 연차일수를 연차 구조체에 총 연차일수 항목에 반환해준다.

<설명>

사원들의 총 연차일수를 일괄적으로 입력해서 연차 구조체에 저장해준다. 초기에 입력하고 끝이 아니라 다시 재입력시 가장 최근에 입력한 값으로 저장된다. 또한 남은 연차일수를 계산할 때 총 연차일수에서 입력된 휴가기간 일수를 차감해서 계산하며 휴가기간 일수가 남은 연차일수보다 커야 휴가처리가 되므로 사원이 신규 입력될 때마다 필수로 총 연차일수일수를 일괄 입력시켜줘야 한다.

<적용된 배운 내용>

-함수, 반복문, 입출력 함수, 구조체, 배열

<코드 스크린샷>

```
void SetTotalLeaveDays() {
    printf("모든 사원의 총 연차일수를 입력하세요: ");
    int total_days;
    scanf_s("%d", &total_days);

    for (int i = 0; i < num_leave; ++i) {
        LeaveTable[i].total_leave_days = total_days;
        LeaveTable[i].remaining_leave_days = total_days;
    }

    printf("총 연차일수가 일괄 입력되었습니다.₩n");
}
```

[ResizeLeaveTable함수]

<입출력>

1)입력

-입력값은 존재하지 않는다.

2)출력

-출력값도 존재하지 않는다.

<설명>

연차 정보를 입력하는 과정에서 ResizeLeaveTable이라는 사용자 정의 함수를 통해 입력된 사원의 수가 할당된 용량에 도달하면 realloc함수를 이용해 새로운 크기의 메모리를 할당한다.

<적용된 배운 내용>

-출력함수, 함수, 동적메모리, 조건문

<코드 스크린샷>

```
void ResizeLeaveTable() {
    if (num_leave == leave_capacity) {
        leave_capacity =
            (leave_capacity == 0) ? INITIAL_CAPACITY : leave_capacity * 2;
        LeaveTable = realloc(LeaveTable, leave_capacity * sizeof(struct AnnualLeave));
        if (LeaveTable == NULL) {
            printf("연차 구조체 배열의 메모리 할당 실패₩n");
            exit(EXIT_FAILURE);
        }
    }
}
```

(3)동적 메모리 할당 해제 기능

[FreeMemory_AnnualLeave/FreeMemoryVacation함수]

<입출력>

1)입력e

-동적 메모리 할당을 해제할 구조체 입력

2)출력

- 반환값은 존재하지 않는다.

<설명>

동적으로 할당된 메모리를 해제하고 프로그램을 종료시켜서 메모리 누수를 방지하고 자원을 효율적으로 관리할 수 있게 해준다.

<적용된 배운 내용>

-함수

<코드 스크린샷>

```
void FreeMemory_Vacation() { free(VacationTable); }
void FreeMemory_AnnualLeave() { free(LeaveTable); }
```

(5) 기능2 함수 헤더파일로 만들기

[vacation_manangement.h]

<입출력>

1)입력

-헤더파일에는 입력 값이 존재하지 않음

2)출력

-헤더파일에는 출력 값이 존재하지 않음

<설명>

기능3에서 구현된 VactionData/AnnualLeaave 구조체,변수(num_vacation, num_leave,capacity),initializeddAnnualLeave/PrintAnnualLeave/SetTotalLeaveDays/Freememory_AnuualLeave/ManageVacation/PrintVacation 등의 함수를 전부 헤더파일로 만들어서 선언해 소스파일의 관리 등을 용이하도록 만들었다. 또한 #ifndef, #endif라는 C 프리프로세서 지시문을 통해 헤더파일이 중복해서 포함되는 것을 방지하였다.

<적용된 배운 내용>

-구조체,함수,포인터,배열과 포인터, 헤더파일,상수와변수

<코드 스크린샷>

```h
#pragma once
#ifndef VACATION_MANAGEMENT_H
#define VACATION_MANAGEMENT_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define VACATION_CAPACITY 10  // 초기 휴가 구조체 배열 메모리 할당 크기

struct VacationData {
    char employee_name[50];
    char start_date[30];
    char return_date[30];
    int duration;
};

struct AnnualLeave {
    char employee_name[50];
    int total_leave_days;
    int remaining_leave_days;
};

extern struct AnnualLeave *LeaveTable;
extern int num_leave;
extern int leave_capacity;

extern struct VacationData *VacationTable;
extern int num_vacation;
extern int vacation_capacity;

void ResizeVacationTable(); //휴가구조체 메모리 동적할당 함수
void ManageVacation(); //휴가 명단 추가
void PrintVacation(const struct VacationData *table, int num_vacation); //휴가자 명단 출력
void FreeMemory_Vacation(); //동적메모리 할당 해제
void ReturnFromVacation(const char *name); //휴가 복귀처리 함수

void InitializeAnnualLeave(const char *name); //신규 사원추가시 연차명단에도 해당 사원이름 추가
void PrintAnnualLeave(); //연차명단 출력함수
void SetTotalLeaveDays(); //총 연차일수 입력 함수
void ResizeLeaveTable(); //연차구조체 메모리 동적할당 함수
void FreeMemory_AnnualLeave(); //동적메모리 할당 해제
#endif
```

[vacation_manangement.h]

<입출력>

1)입력

-기능3의 함수들을 그대로 사용하여서 기능3의 모든 함수들과 동일한 내용이다.

2)출력

-기능3의 함수들을 그대로 사용하여서 기능3의 모든 함수들과 동일한 내용이다.

<설명>

Vacation_management.h 에서 선언한 VacationData/AnnualLeaveData 구조체,변수(num_vacation,num_leave,capacity),PrintAnnualLeave/ResizeLeaveTable/Freememory_AnnualLeave/ManageVacation 등의 함수들을 구현해주었다.

<적용된 배운 내용>

-구조체,함수,포인터,배열과 포인터,문자열,조건문,반복문,입출력함수,변수,헤더파일

<코드 스크린샷>

```c
#include "vacation_management.h"
#include "employee.h"

struct VacationData *VacationTable = NULL;
int num_vacation = 0;
int vacation_capacity = 0;

struct AnnualLeave *LeaveTable = NULL;
int num_leave = 0;
int leave_capacity = 0;

void ResizeVacationTable() {
    if (num_vacation == vacation_capacity) {
        vacation_capacity =
            (vacation_capacity == 0) ? VACATION_CAPACITY : vacation_capacity * 2;
        VacationTable =
            realloc(VacationTable, vacation_capacity * sizeof(struct VacationData));
        if (VacationTable == NULL) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
    }
}

void ResizeLeaveTable() {
    if (num_leave == leave_capacity) {
        leave_capacity =
            (leave_capacity == 0) ? INITIAL_CAPACITY : leave_capacity * 2;
        LeaveTable = realloc(LeaveTable, leave_capacity * sizeof(struct AnnualLeave));
        if (LeaveTable == NULL) {
            printf("연차 구조체 배열의 메모리 할당 실패₩n");
            exit(EXIT_FAILURE);
        }
    }
}
```

```c
void ManageVacation() {
  char employee_name[50];
  printf("휴가자의 이름을 입력하세요: ");
  scanf_s("%s", employee_name, (int)sizeof(employee_name));

  int employeeIndex = FindPerson(EmployeeTable, num_people, employee_name);

  if (employeeIndex == -1) {
    printf("해당 사원은 존재하지 않습니다.₩n");
    return;
  }

  ResizeVacationTable();

  printf("휴가 출발일을 입력하세요 (ex: 1900/01/01): ");
  scanf_s("%s", VacationTable[num_vacation].start_date,
          (int)sizeof(VacationTable[num_vacation].start_date));

  printf("휴가 복귀일을 입력하세요 (ex: 1900/01/01): ");
  scanf_s("%s", VacationTable[num_vacation].return_date,
          (int)sizeof(VacationTable[num_vacation].return_date));

  printf("휴가 기간을 입력하세요 (일수): ");
  scanf_s("%d", &VacationTable[num_vacation].duration);

  strcpy_s(VacationTable[num_vacation].employee_name,
           sizeof(VacationTable[num_vacation].employee_name), employee_name);

   int remainingLeaveDays =
      LeaveTable[num_vacation].total_leave_days - VacationTable[num_vacation].duration;

  if (remainingLeaveDays < 0) {
    printf("남은 연차일수가 부족합니다.₩n");
    return;
  }

  // 연차 테이블 업데이트
  LeaveTable[num_vacation].remaining_leave_days = remainingLeaveDays;

  num_vacation++;

  printf("%s님의 휴가 정보가 저장되었습니다.₩n", employee_name);
}

void PrintVacation(const struct VacationData *table, int num_vacation) {
  for (int i = 0; i < num_vacation; i++) {
    printf("------------------------------₩n");
    printf("이름: %s₩n", table[i].employee_name);
    printf("휴가 출발일: %s₩n", table[i].start_date);
    printf("휴가 복귀일: %s₩n", table[i].return_date);
    printf("휴가 기간: %d 일₩n", table[i].duration);
    printf("------------------------------₩n");
  }
}
```

```c
void ReturnFromVacation(const char *name) {
  int TargetName_Vacation[50];
  printf("₩n휴가 복귀한 사원의 이름을 입력하세요: ");
  scanf_s("%s", TargetName_Vacation, (int)sizeof(TargetName_Vacation));
  int index = FindPerson(VacationTable, num_vacation, TargetName_Vacation);
  if (index != -1) {
    // 휴가 명단에서 해당 사원의 정보를 삭제
    for (int i = index; i < num_vacation - 1; ++i) {
      strcpy_s(VacationTable[i].employee_name,
               sizeof(VacationTable[i].employee_name),
               VacationTable[i + 1].employee_name);
      strcpy_s(VacationTable[i].start_date, sizeof(VacationTable[i].start_date),
               VacationTable[i + 1].start_date);
      strcpy_s(VacationTable[i].return_date, sizeof(VacationTable[i].return_date),
               VacationTable[i + 1].return_date);
      VacationTable[i].duration = VacationTable[i + 1].duration;
    }
    --num_vacation;
    printf("%s님의 휴가 복귀 처리가 완료되었습니다.₩n", name);
  } else {
    printf("%s님은 휴가 명단에 없는 사원입니다. 복귀 처리를 할 수 없습니다.₩n",
           name);
  }
}

void InitializeAnnualLeave(const char *name) {
  ResizeLeaveTable();
  strcpy_s(LeaveTable[num_leave].employee_name,
           sizeof(LeaveTable[num_leave].employee_name), name);
  LeaveTable[num_leave].total_leave_days = 0;      // 총 연차일수 초기화
  LeaveTable[num_leave].remaining_leave_days = 0;  // 남은 연차일수 초기화
  num_leave++;
}

void PrintAnnualLeave() {
  printf("사원들의 연차 명단:₩n");
  for (int i = 0; i < num_leave; ++i) {
    printf("------------------------------₩n");
    printf("이름: %s₩n", LeaveTable[i].employee_name);
    printf("총 연차일수: %d₩n", LeaveTable[i].total_leave_days);
    printf("남은 연차일수: %d₩n", LeaveTable[i].remaining_leave_days);
    printf("------------------------------₩n");
  }
}

void SetTotalLeaveDays() {
  printf("모든 사원의 총 연차일수를 입력하세요: ");
  int total_days;
  scanf_s("%d", &total_days);

  for (int i = 0; i < num_leave; ++i) {
    LeaveTable[i].total_leave_days = total_days;
    LeaveTable[i].remaining_leave_days = total_days;
  }

  printf("총 연차일수가 일괄 입력되었습니다.₩n");
}

void FreeMemory_Vacation() { free(VacationTable); }
void FreeMemory_AnnualLeave() { free(LeaveTable); }
```

4. 테스트 결과

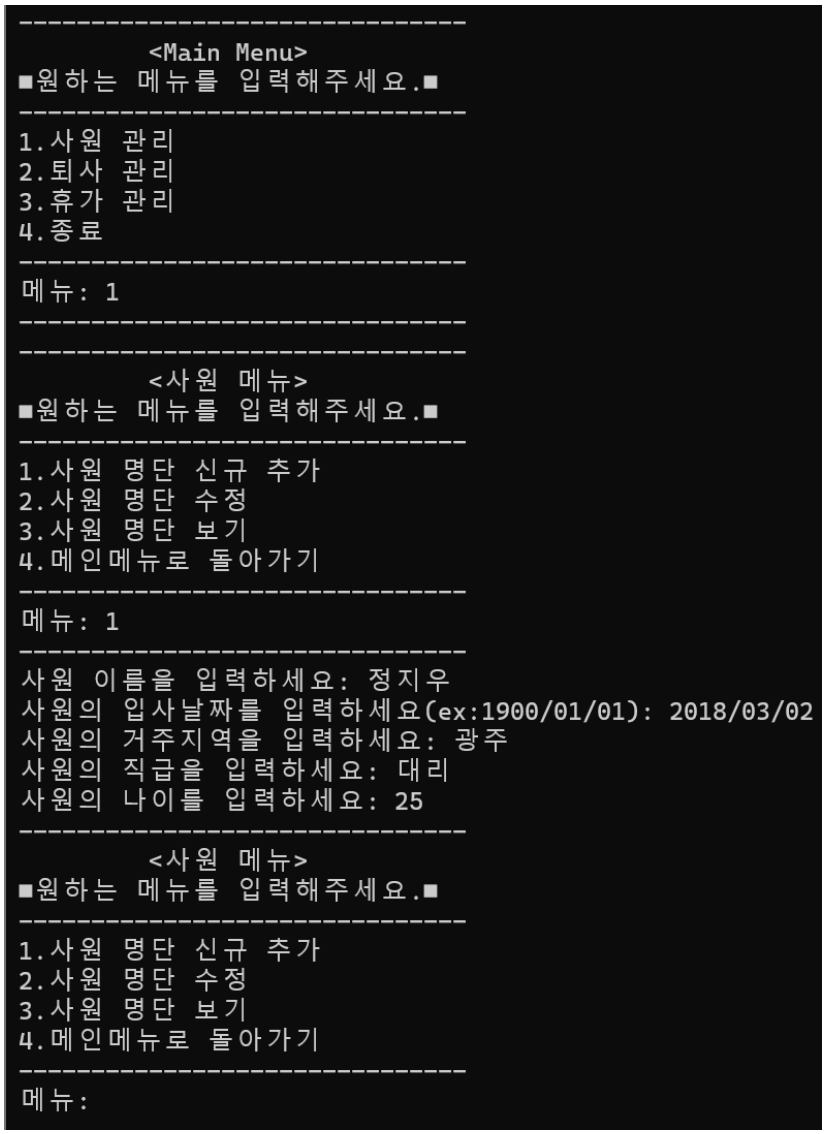
1)신규 사원 기능

(1) 사원 정보 입력 기능

- 설명

처음에 원하는 메뉴를 입력하는 문구가 출력되면 사원 명단 신규 추가인 1번을 입력하면 사원을 입력하는 문구가 출력된다. 차례차례 이름부터 입사날짜,거주지,직급,나이를 입력하는 문구가 출력되면 각각 정보를 입력하면 사원 정보 입력이 완료되고 다시 처음에 메뉴를 선택으로 돌아간다..

- 테스트 결과 스크린샷

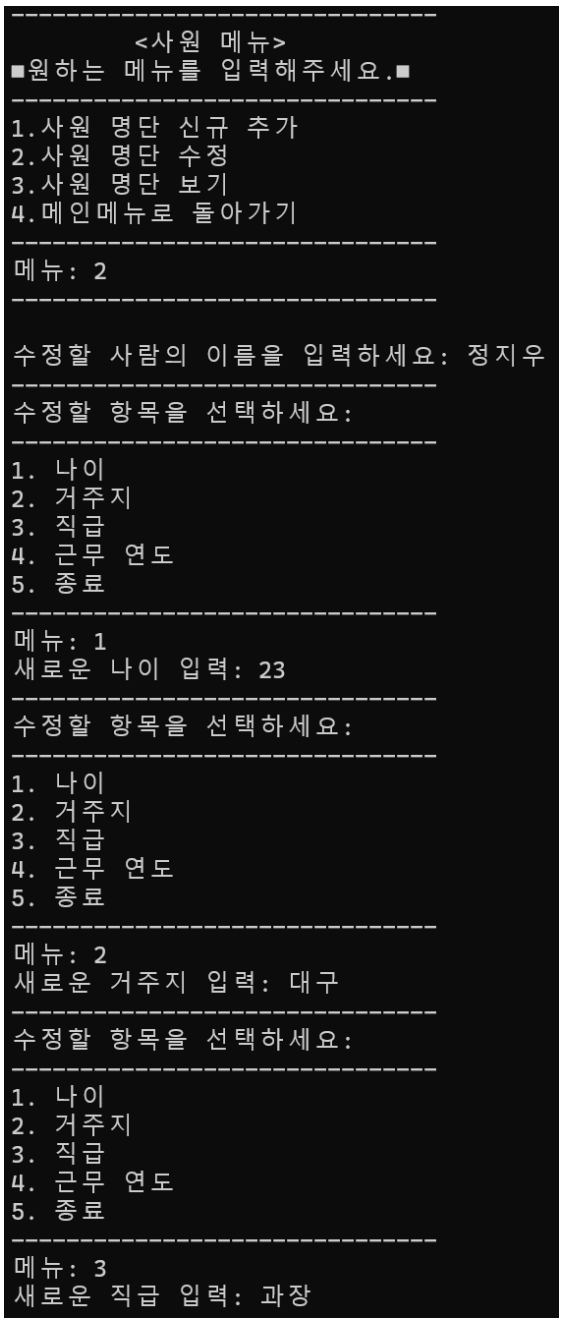


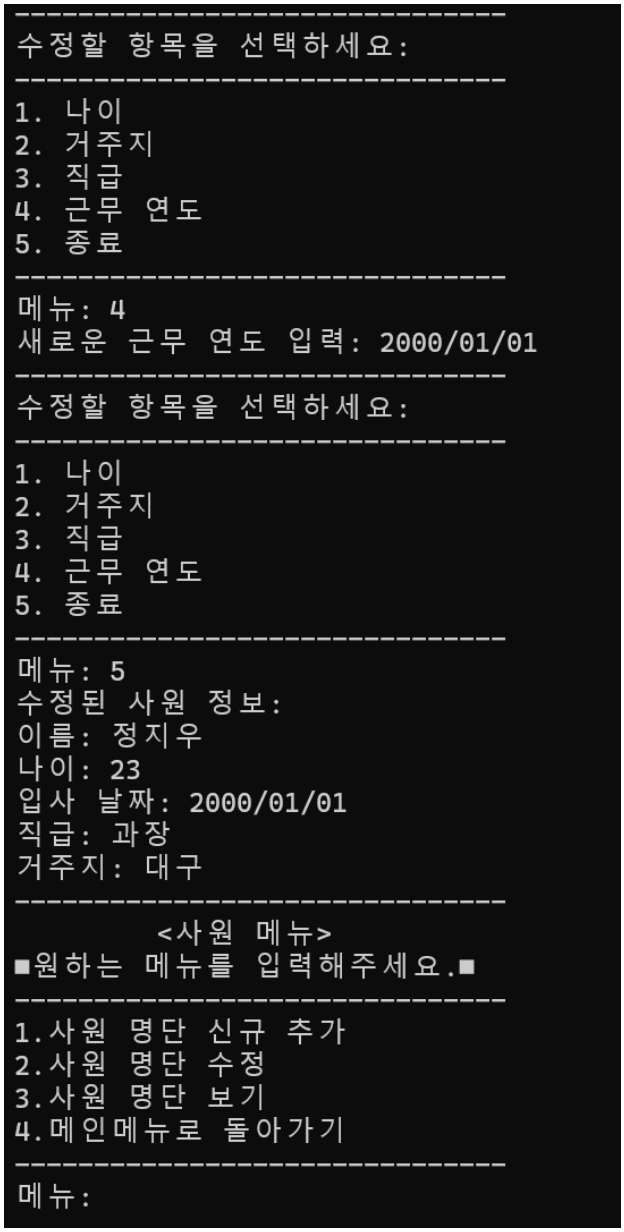
(2) 사원 정보 수정 기능

- 설명

사원 정보를 수정하고자 하는 사원의 이름을 입력하면 수정할 항목들을 고르는 문구가 출력된다. 수정할 해당 항목의 번호를 입력하면 수정할 정보를 입력할 수 있으며 수정이 다 끝나고 종료 5를 입력하면 수정이 종료되고 입력한 수정 정보가 출력되고 처음 메뉴를 고르는 화면으로 돌아간다.

- 테스트 결과 스크린샷



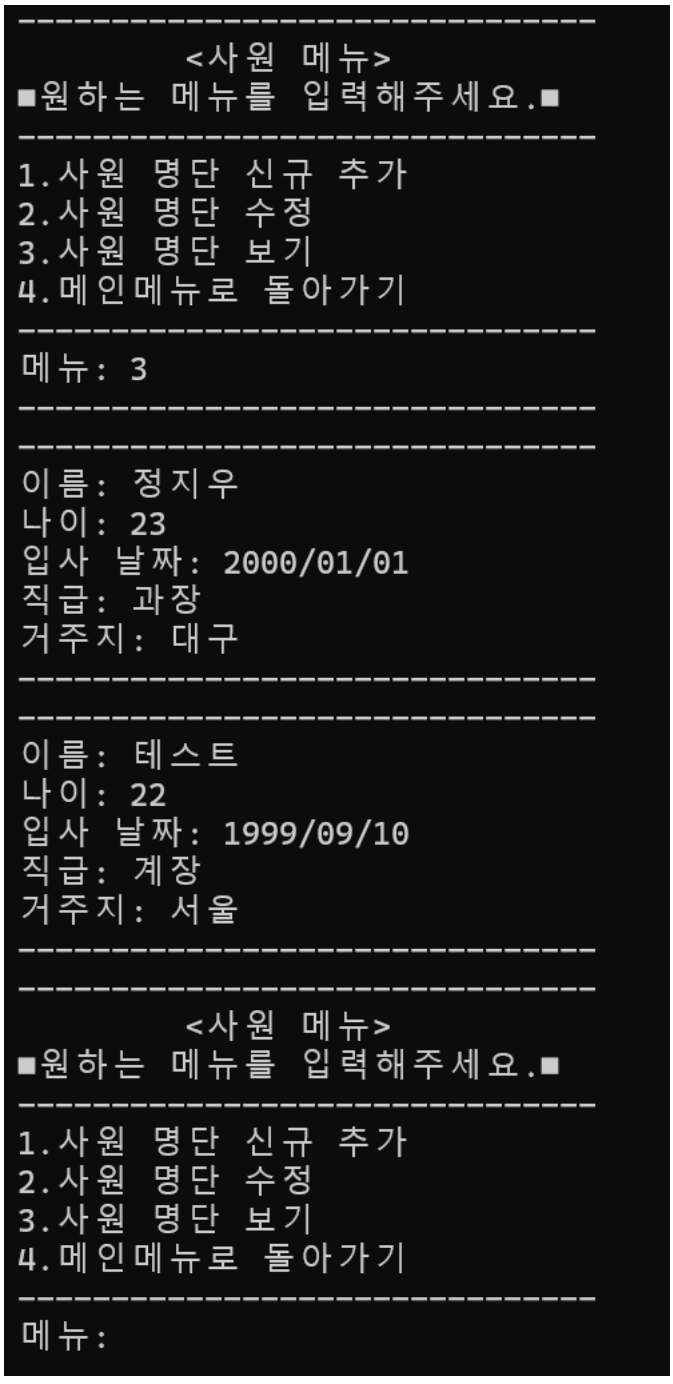


(3) 사원 정보 출력 기능

- 설명

입력된 사원 정보를 보고싶다면 초기 원하는 메뉴를 고르는 문구가 출력되면 사원 명단 보기를 의미하는 3을 입력한다. 3을 입력하면 현재 저장된 모든 사원의 정보가 사원별로 출력되어 확인할 수 있다.

- 테스트 결과 스크린샷



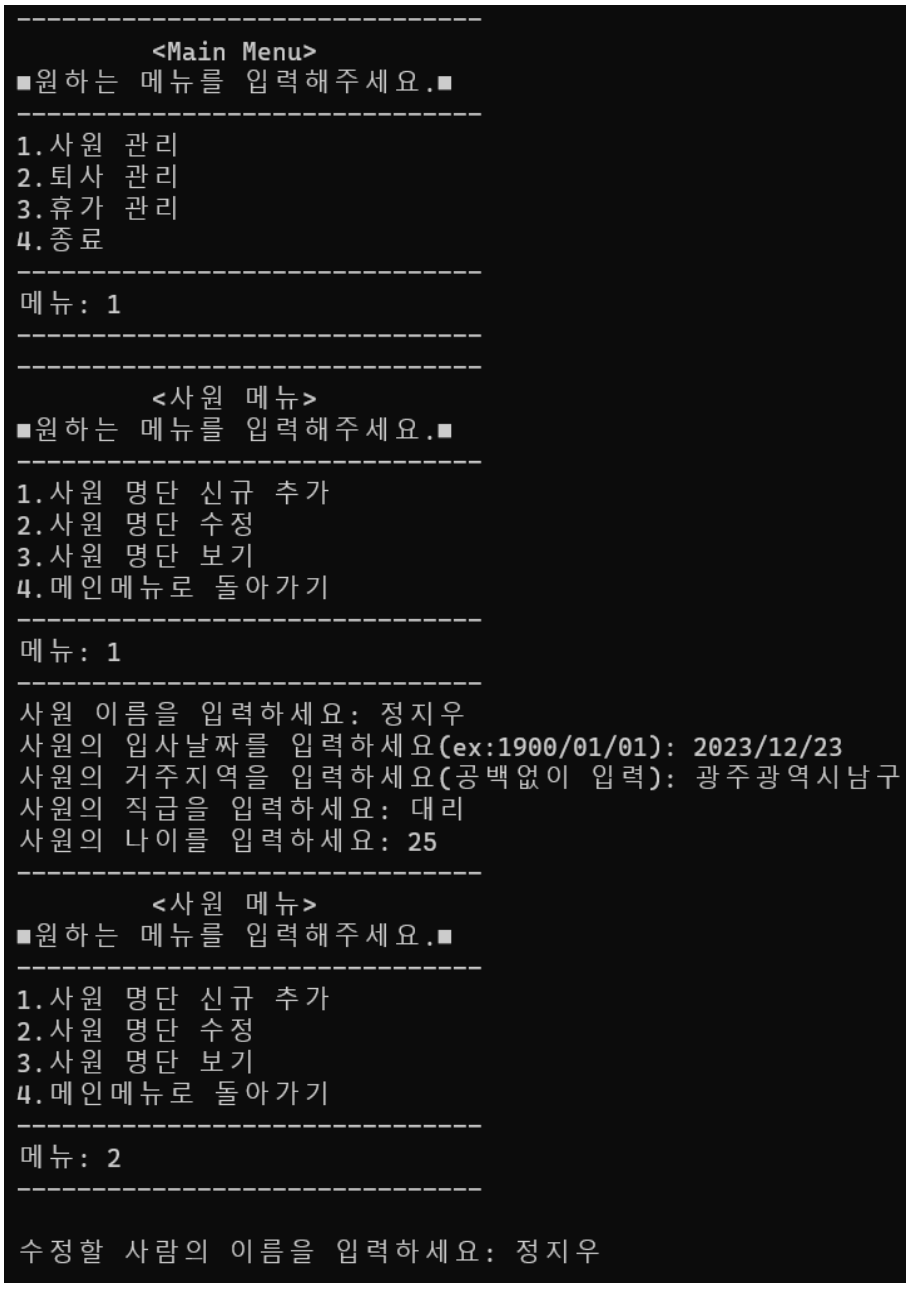
(4) 기능1 함수 헤더파일로 만들기

- 설명

기능1에서 구현하였던 함수들을 모두 employee.h에서 선언하고 employee.c.에서 정의하여 main함수에서는 헤더파일로 불러와서 사용하도록 헤더파일로 만들었으며 기능1의 사원명단 추가/수정/출력 기능들을 모두 테스트해본 결과 정상작동하였다.

- 테스트 결과 스크린샷

(앞서 테스트한 기능3의 내용과 동일하며 모든 기능이 정상 작동하는지 확인)



```
수정할 항목을 선택하세요:
------------------------------
1. 나이
2. 거주지
3. 직급
4. 근무 연도
5. 종료
------------------------------
메뉴: 1
새로운 나이 입력: 24
------------------------------
수정할 항목을 선택하세요:
------------------------------
1. 나이
2. 거주지
3. 직급
4. 근무 연도
5. 종료
------------------------------
메뉴: 2
새로운 거주지 입력(공백없이 입력): 광주광역시
------------------------------
수정할 항목을 선택하세요:
------------------------------
1. 나이
2. 거주지
3. 직급
4. 근무 연도
5. 종료
------------------------------
메뉴: 3
새로운 직급 입력: 사원
------------------------------
수정할 항목을 선택하세요:
------------------------------
1. 나이
2. 거주지
3. 직급
4. 근무 연도
5. 종료
------------------------------
메뉴: 4
새로운 근무 연도 입력: 2023/12/20
```

```
------------------------------
수정할 항목을 선택하세요:
------------------------------
1. 나이
2. 거주지
3. 직급
4. 근무 연도
5. 종료
------------------------------
메뉴: 5
수정된 사원 정보:
이름: 정지우
나이: 24
입사 날짜: 2023/12/20
직급: 사원
거주지: 광주광역시
------------------------------
          <사원 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.사원 명단 신규 추가
2.사원 명단 수정
3.사원 명단 보기
4.메인메뉴로 돌아가기
------------------------------
메뉴: 3
------------------------------
------------------------------
이름: 정지우
나이: 24
입사 날짜: 2023/12/20
직급: 사원
거주지: 광주광역시
------------------------------
------------------------------
          <사원 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.사원 명단 신규 추가
2.사원 명단 수정
3.사원 명단 보기
4.메인메뉴로 돌아가기
------------------------------
메뉴:
```

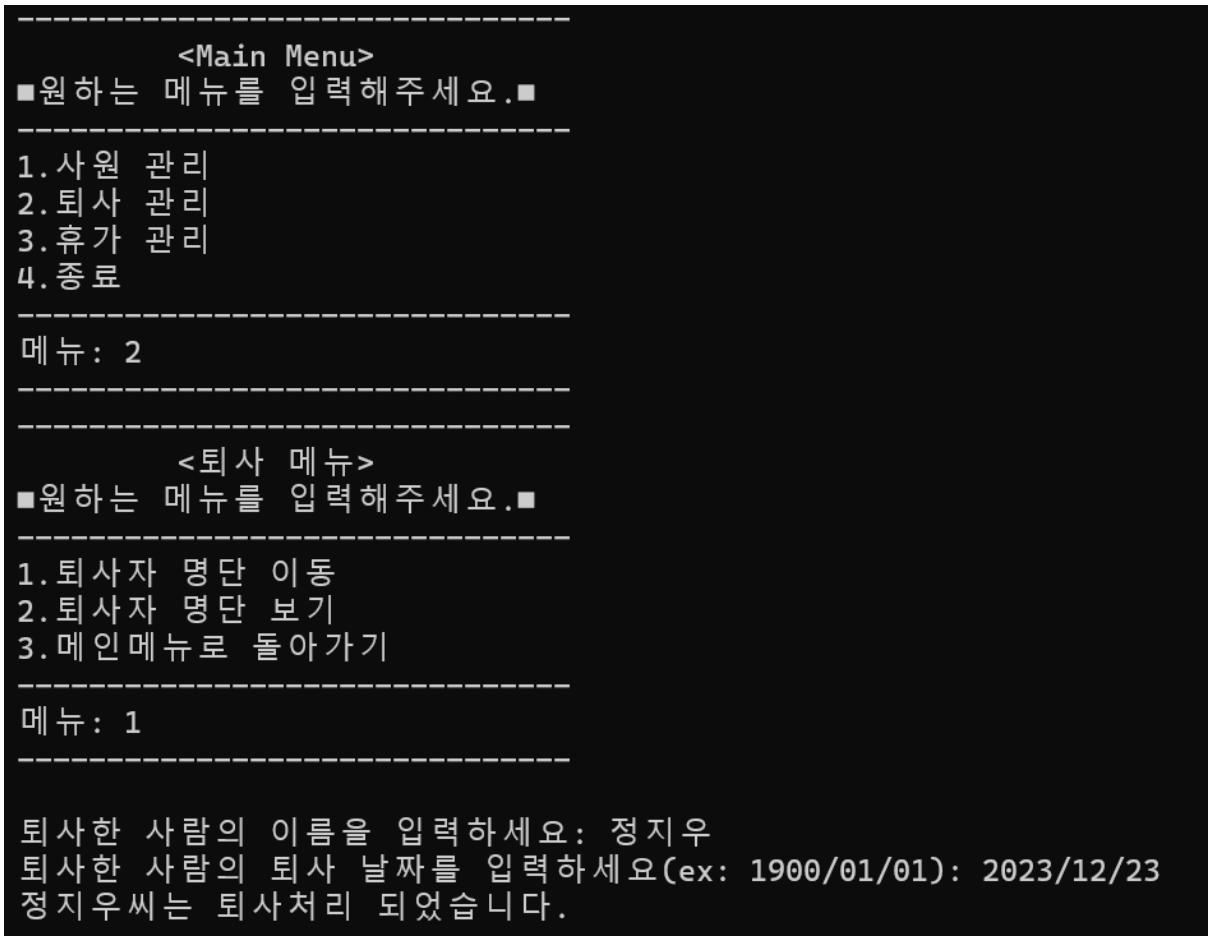
2)퇴사 관리 기능

(1) 퇴사자 명단 추가 기능

- 설명

사원 명단에 존재하는 사원 중에 퇴사한 사원의 이름을 입력하면 그 사원의 정보를 사원 명단에서 삭제하고 퇴사자 명단에 추가해주고 "***씨는 퇴사처리 되었습니다"라는 확인 문구가 출력된다. 또한 퇴사날짜를 추가로 입력하여 그 정보를 퇴사자 명단에 추가 저장해 준다.

- 테스트 결과 스크린샷

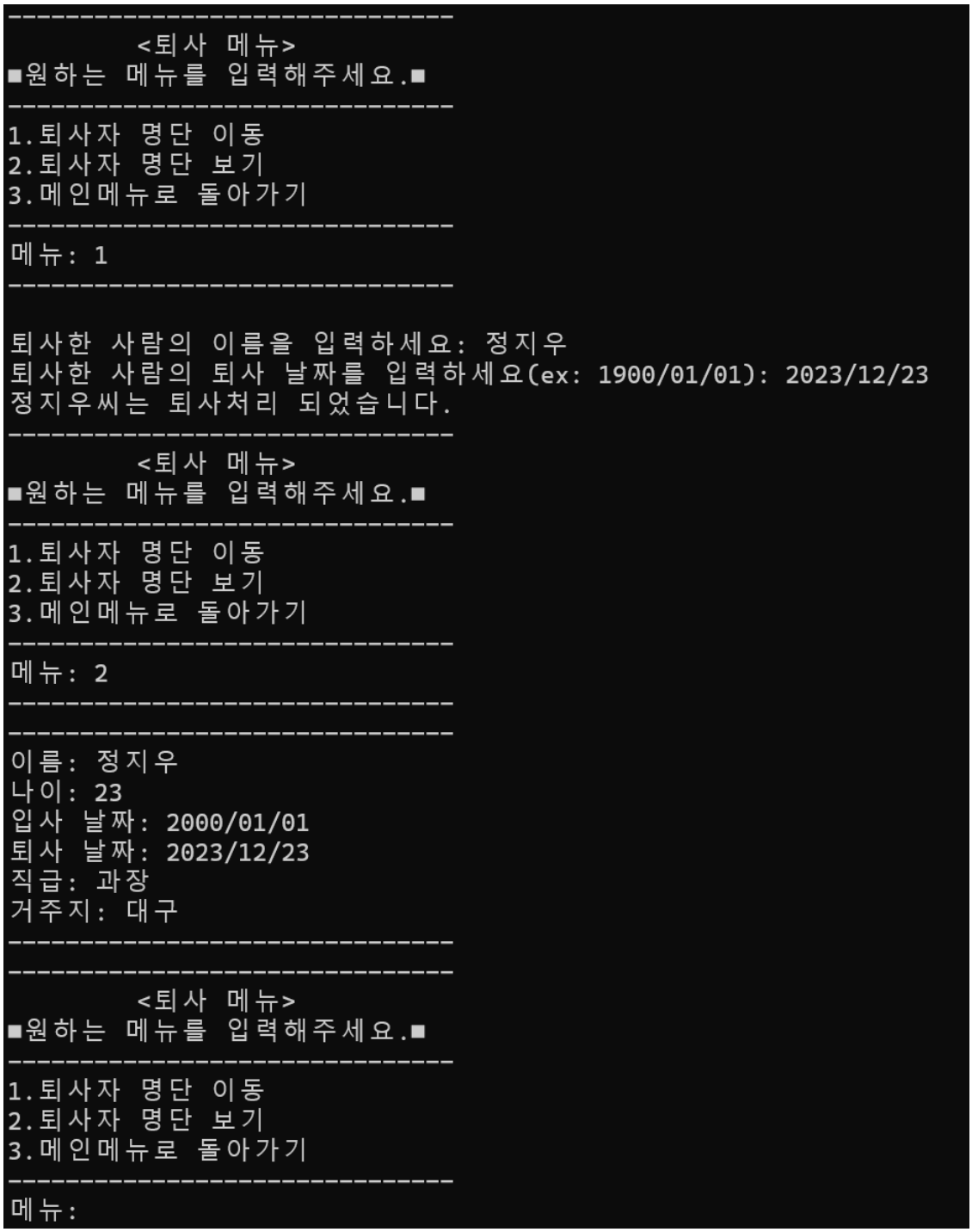


(2) 퇴사자 명단 출력 기능

- 설명

퇴사자 명단 추가 기능으로 인해 사원 명단에서 퇴사자 명단으로 이동한 퇴사한 사원의 정보들을 각 사원별로 퇴사날짜를 포함하여 모두 출력해준다.

- 테스트 결과 스크린샷



(3) 기능2 함수 헤더파일로 만들기

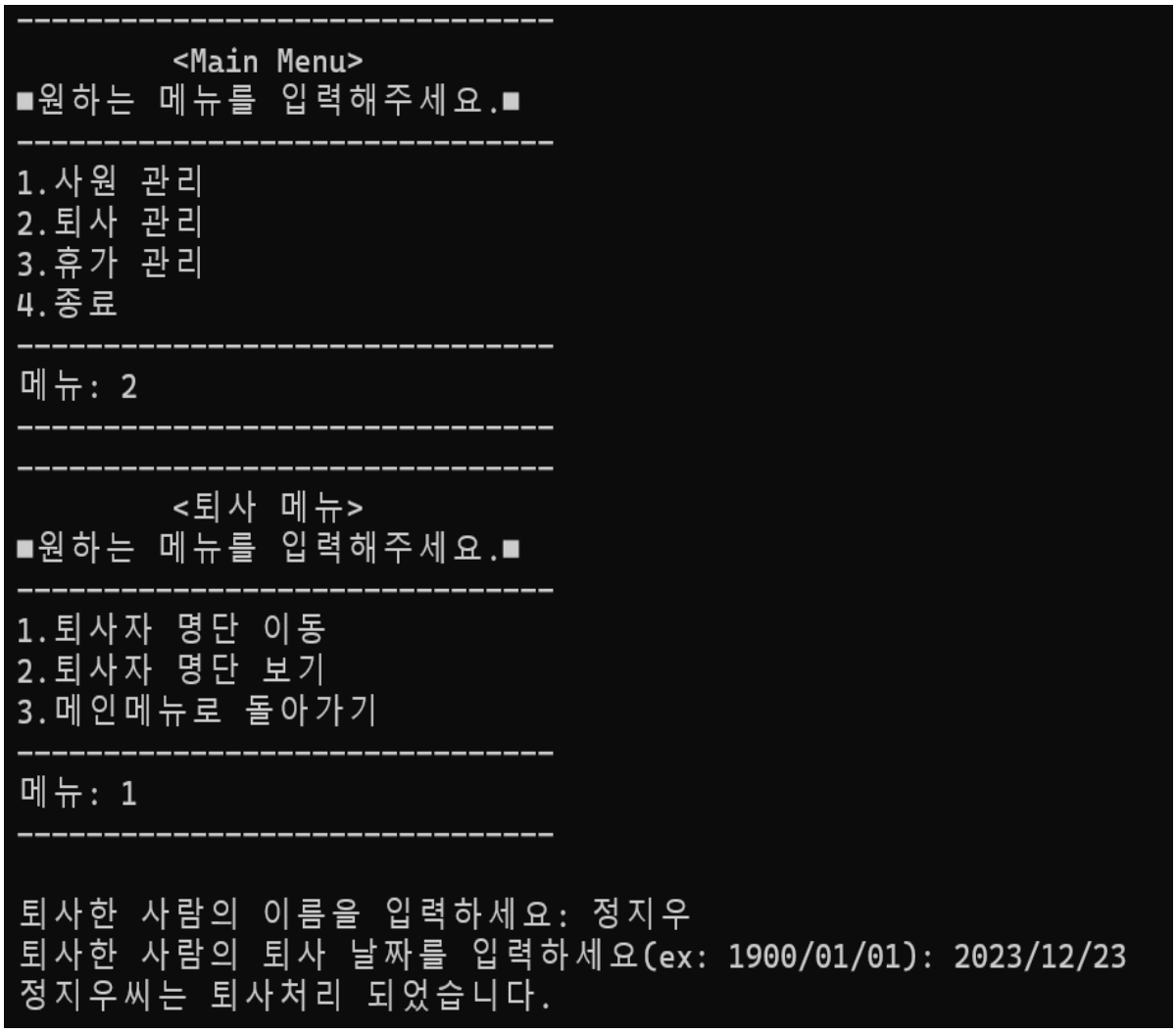
- 설명

기능2에서 구현하였던 함수들을 모두 resigned_employee.h에서 선언하고 resigned_empl

oyee.c.에서 정의하여 main함수에서는 헤더파일로 불러와서 사용하도록 헤더파일로 만들었으며 기능 2의 퇴사자 관리 기능들을 모두 테스트해본 결과 정상 작동하였다.

- 테스트 결과 스크린샷

(앞서 테스트한 기능3의 내용과 동일하며 모든 기능이 정상 작동하는지 확인)



```
------------------------------
        <퇴사 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.퇴사자 명단 이동
2.퇴사자 명단 보기
3.메인메뉴로 돌아가기
------------------------------
메뉴: 1
------------------------------

퇴사한 사람의 이름을 입력하세요: 정지우
퇴사한 사람의 퇴사 날짜를 입력하세요(ex: 1900/01/01): 2023/12/23
정지우씨는 퇴사처리 되었습니다.
------------------------------
        <퇴사 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.퇴사자 명단 이동
2.퇴사자 명단 보기
3.메인메뉴로 돌아가기
------------------------------
메뉴: 2
------------------------------
------------------------------
이름: 정지우
나이: 23
입사 날짜: 2000/01/01
퇴사 날짜: 2023/12/23
직급: 과장
거주지: 대구
------------------------------
------------------------------
        <퇴사 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.퇴사자 명단 이동
2.퇴사자 명단 보기
3.메인메뉴로 돌아가기
------------------------------
메뉴:
```

3)휴가 관리 기능

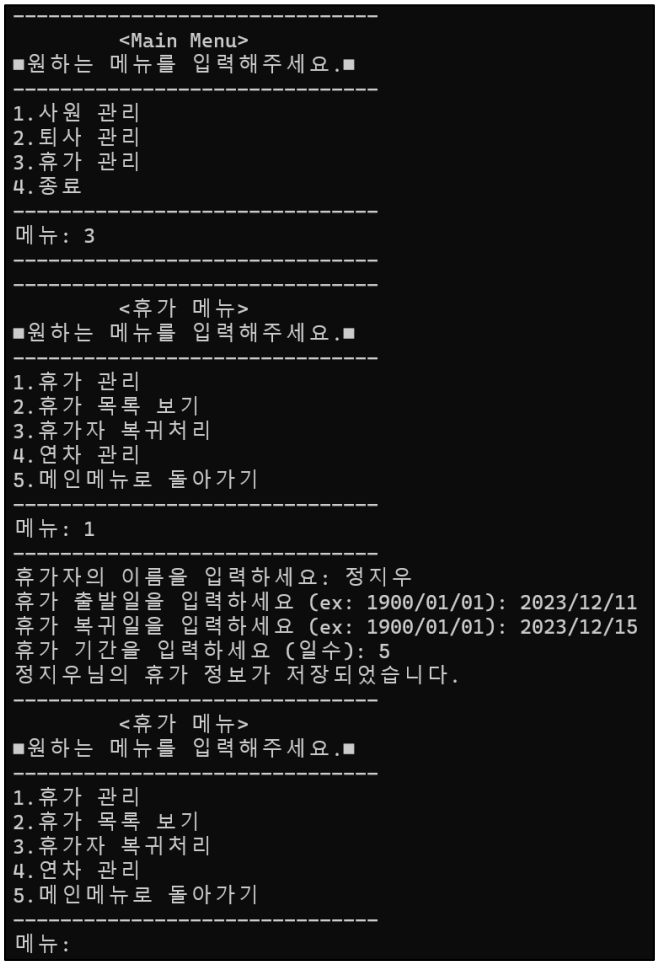
(1) 휴가자 명단 추가 기능

- 설명

최초의 Mainmenu 화면에서 3번을 입력하여 휴가 관리 메뉴로 진입하면 휴가메뉴를 고를 수 있는 문구가 출력이 된다. 그리고 1번 휴가관리 메뉴를 입력하면 휴가를 간 사원의 이름을 입력하는 문구가 출력이 되고 해당 사원의 이름을 입력하면 휴가자 명단에 해당 사원이 추가가 된다. 또한 이름을 입력하고 나면 이어서 휴가 복귀일과 휴가 출발일, 휴가기간 일수를 입력하는 문구가 추가가 되고 해당 정보들은 휴가자 명단에 함께 저장된다. 만약 휴가기간 일수가 남은 연차일수보다 클 경우 휴가 처리가 안된다. 따라서 사원을 신규 등록할시 기본연차 일괄입력을 해주어야 한다.

- 테스트 결과 스크린샷

(사원 1명을 사원관리 기능을 이용하여 입력한 상태에서 휴가관리 실행)

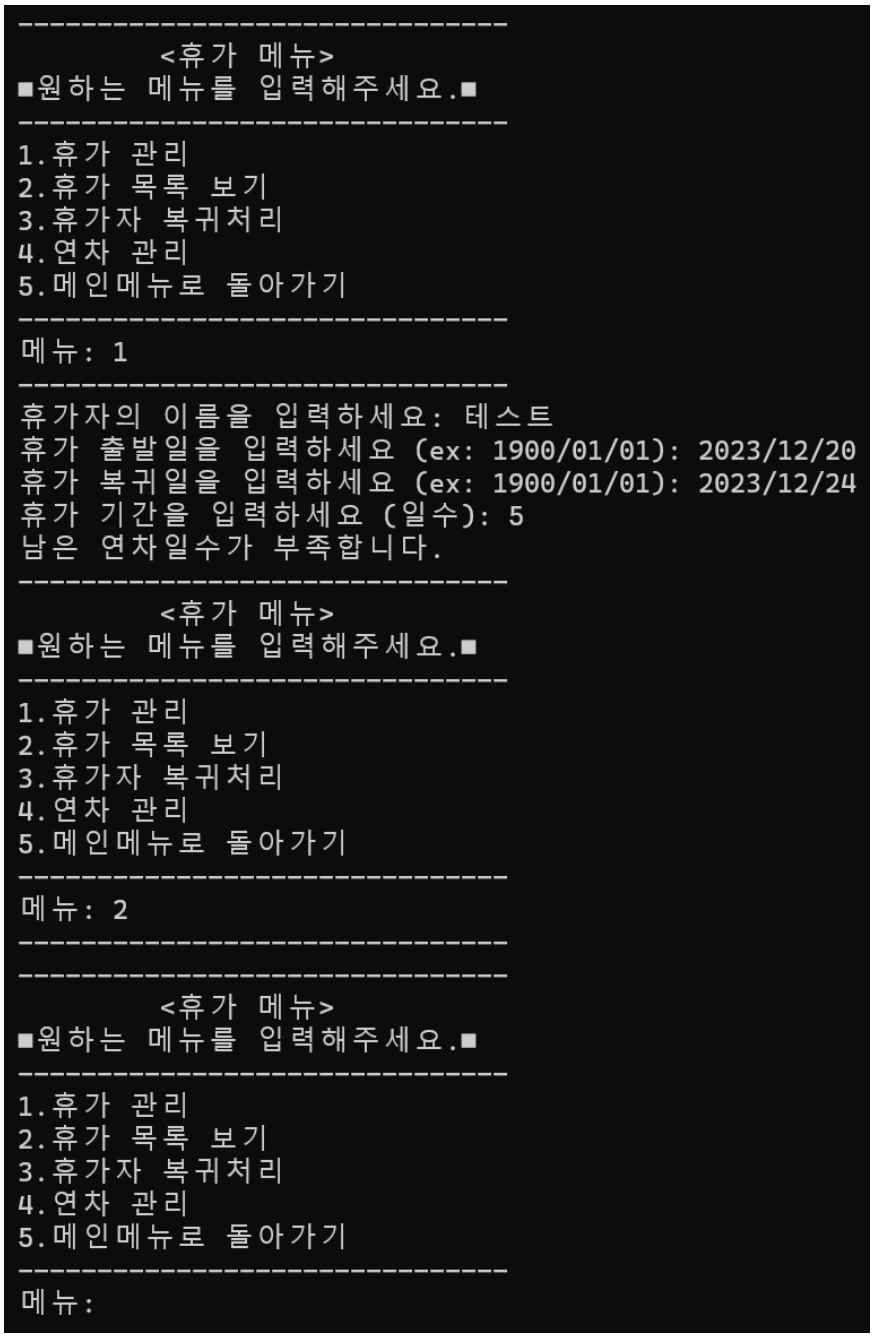


(2) 휴가자 명단 출력 기능

- 설명

휴가자 명단에 저장된 휴가자 이름, 휴가 출발일, 휴가 복귀일, 휴가 기간일수를 각 휴가자 별로 모두 출력하도록 하는 기능이다. 이때 복귀자는 복귀처리와 동시에 휴가자 명단에서 사라지기에 현재 휴가중인 사원의 정보만 표시된다.

- 테스트 결과 스크린샷(사원 1명 정보가 기능1을 사용해 선입력된 상태)

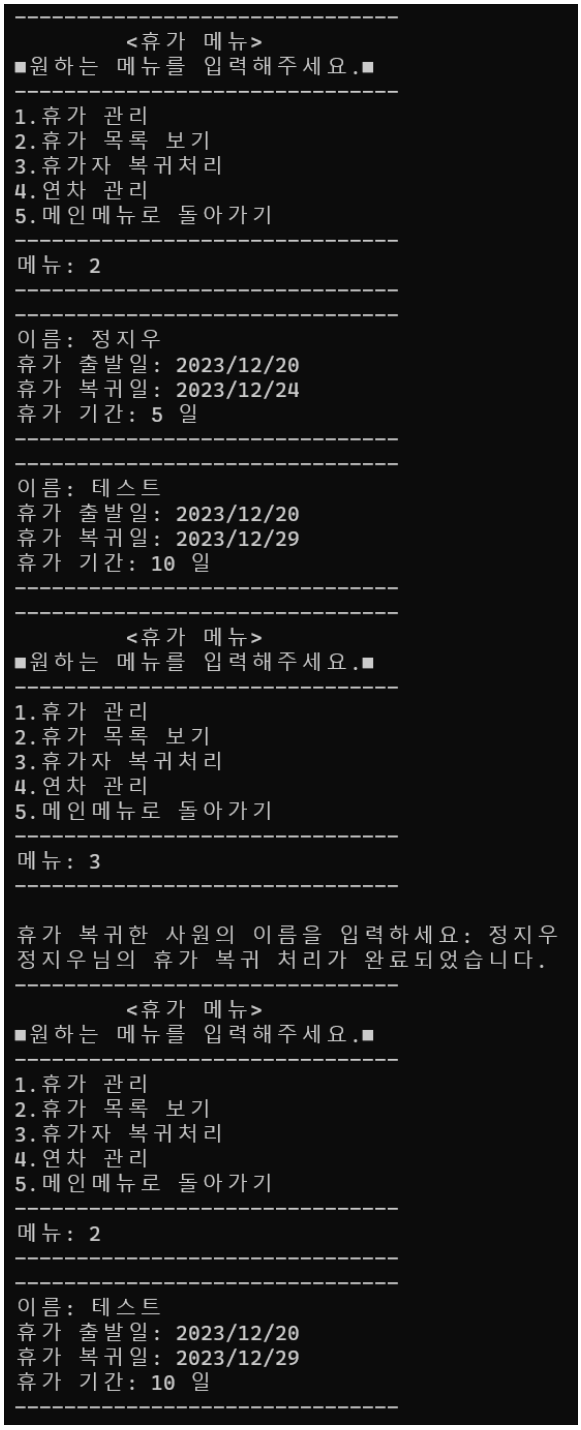


(3) 휴가자 복귀처리 기능

- 설명

휴가자 명단에 저장된 휴가자가 휴가에서 복귀한 경우 해당 사원의 데이터를 휴가자 명단에서 삭제해준다.

- 테스트 결과 스크린샷(사원 1명 정보가 기능1을 사용해 선입력된 상태)

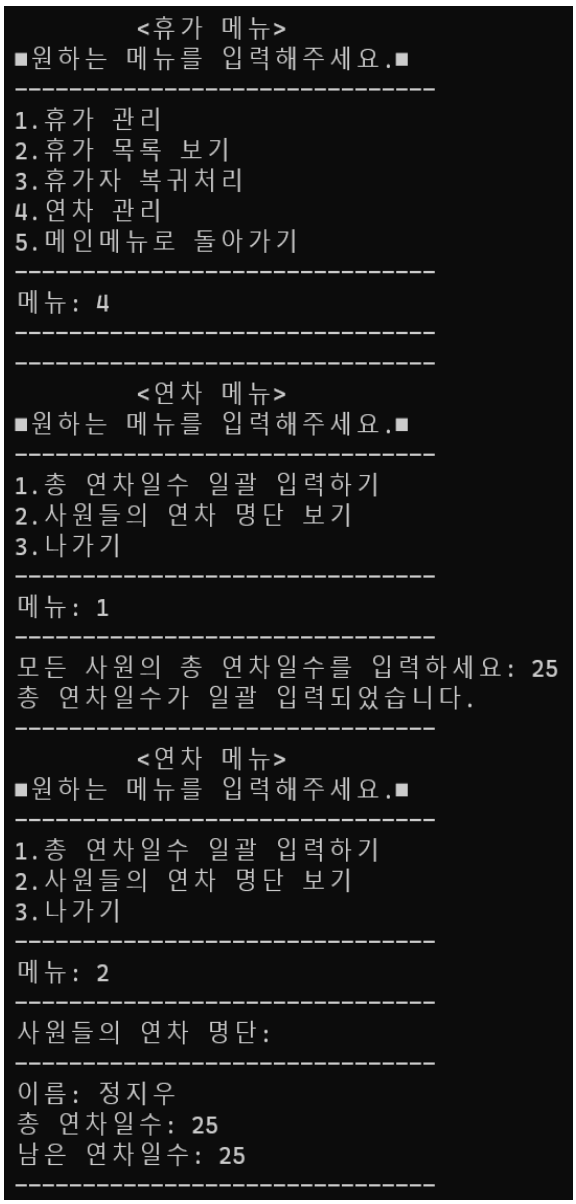


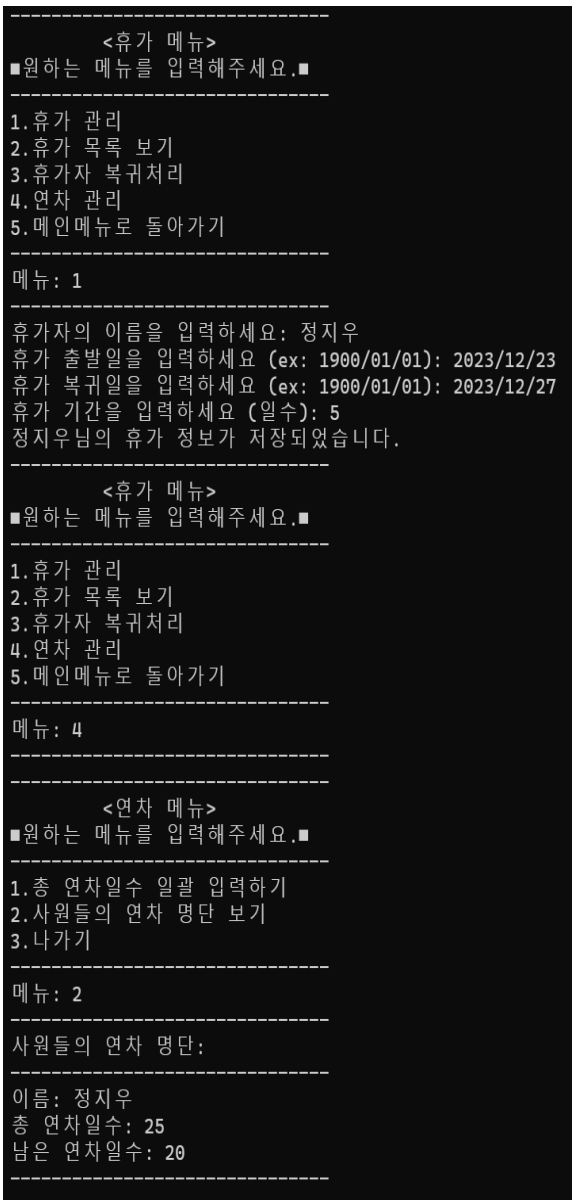
(4) 총 연차일수 일괄 입력 및 연차 명단 출력 기능

- 설명

총 연차일수 일괄 입력 기능은 회사에 입사한 사원들에게 기본적으로 주어지는 연차일수를 모든 사원들의 연차 테이블에 일괄 입력해주는 기능이다. 또한 연차 명단 출력 기능은 사원들의 총 연차일수와 휴가로 사용하고 남은 연차일수를 사원별로 출력해주는 기능이다.

- 테스트 결과 스크린샷(사원 1명 정보가 기능1을 사용해 선입력된 상태)





<휴가로 연차를 소진하지 않은 경우> **--→** **<휴가로 연차를 소진한 경우>**

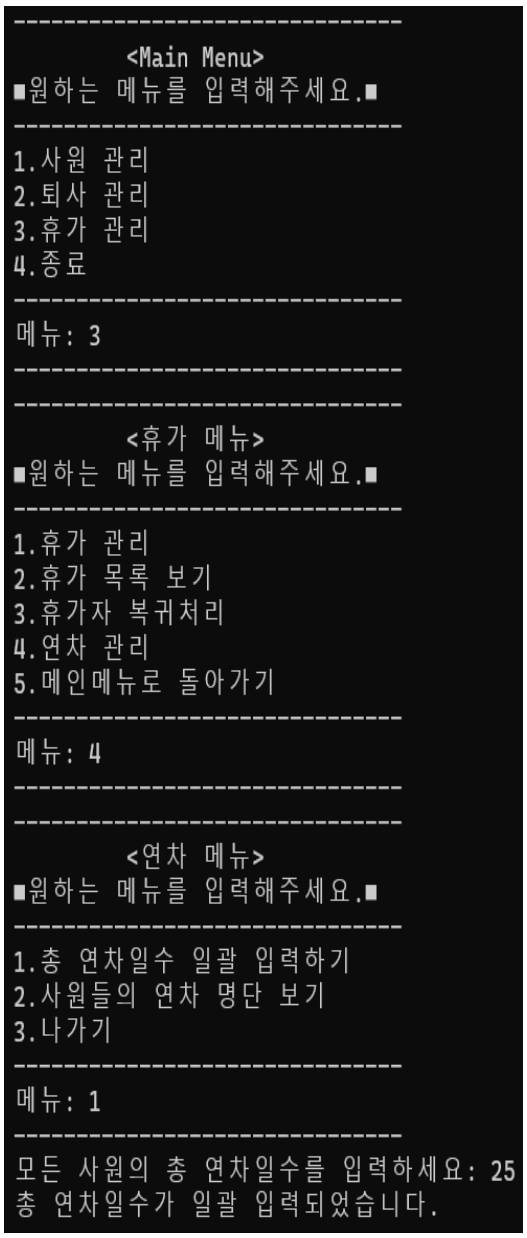
(5) 기능3 함수 헤더파일로 만들기

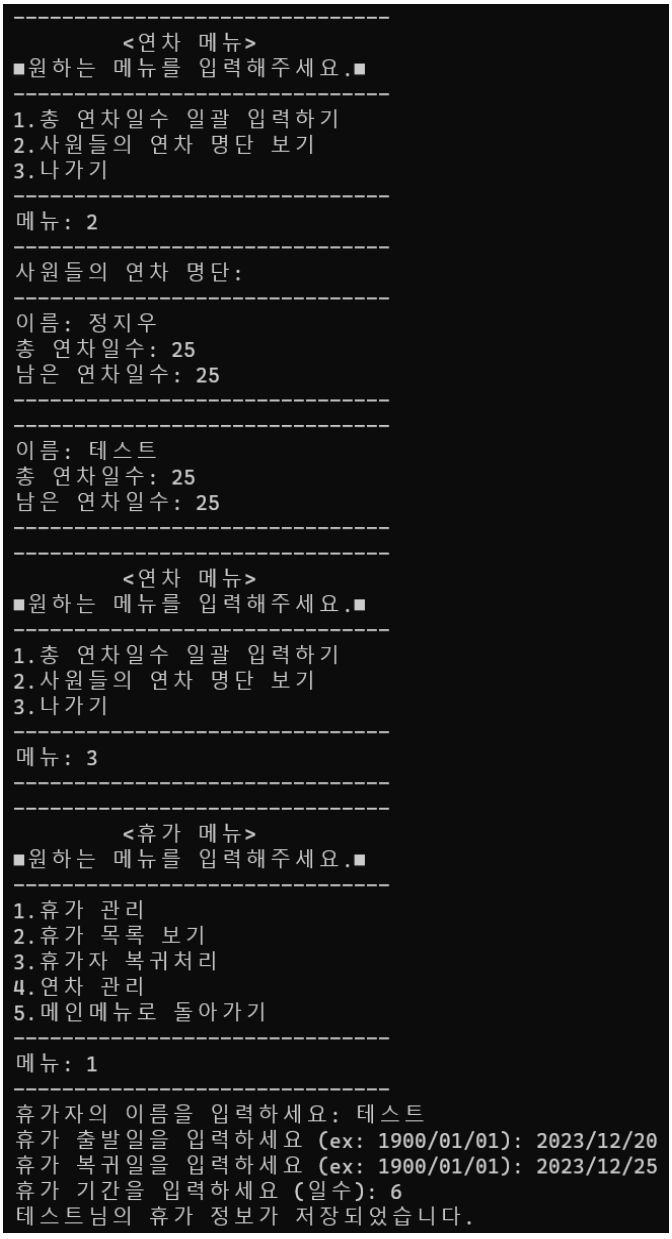
- 설명

기능3에서 구현하였던 함수들을 모두 vacation_management.h에서 선언하고 vacation_management.c.에서 정의하여 main함수에서는 헤더파일로 불러와서 사용하도록 헤더파일로 만들었으며 기능 3의 휴가 및 연차 관리 기능들을 모두 테스트해본 결과 정상 작동하였다.

- 테스트 결과 스크린샷

(앞서 테스트한 기능3의 내용과 동일하며 모든 기능이 정상 작동하는지 확인)





```
------------------------------
        <휴가 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.휴가 관리
2.휴가 목록 보기
3.휴가자 복귀처리
4.연차 관리
5.메인메뉴로 돌아가기
------------------------------
메뉴: 2
------------------------------
------------------------------
이름: 테스트
휴가 출발일: 2023/12/20
휴가 복귀일: 2023/12/25
휴가 기간: 6 일
------------------------------
------------------------------
        <휴가 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.휴가 관리
2.휴가 목록 보기
3.휴가자 복귀처리
4.연차 관리
5.메인메뉴로 돌아가기
------------------------------
메뉴: 3
------------------------------

휴가 복귀한 사원의 이름을 입력하세요: 테스트
정지우님의 휴가 복귀 처리가 완료되었습니다.
------------------------------
        <휴가 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.휴가 관리
2.휴가 목록 보기
3.휴가자 복귀처리
4.연차 관리
5.메인메뉴로 돌아가기
------------------------------
메뉴: 2
------------------------------
------------------------------
        <휴가 메뉴>
■원하는 메뉴를 입력해주세요.■
------------------------------
1.휴가 관리
2.휴가 목록 보기
3.휴가자 복귀처리
4.연차 관리
5.메인메뉴로 돌아가기
------------------------------
메뉴:
```

5. 계획 대비 변경 사항

1) 사원 정보 입력 기능 일부 수정

- 이전

초기에는 사원정보 구조체에 최대 용량을 100으로 지정해기에 이미 정해진 크기의 배열의 메모리에 입력하는 사원의 정보를 저장해서 채워 나가는 방식이었다.

- 이후

사원정보 구조체에 메모리를 지정하지 않고 동적할당을 통해 새로운 사원의 정보가 입력될 때 마다 메모리를 증가시켜 저장하도록 변경하였다.

- 사유

미리 메모리를 할당하는 것보다 상황에 따라 원하는 크기만큼 메모리를 할당하는 동적할당을 이용하는 것이 훨씬 메모리 낭비가 없고 효율적이고 유연하다고 판단하였다.

2) 사원 정보 출력 기능 추가

- 이전

사원 구조체 배열에 사원 정보를 입력하고 수정하는 기능만 있었지 출력하는 기능이 존재하지 않아 입력한 사원 정보를 볼 수 없었다.

- 이후

사원 정보를 모두 출력하는 기능을 통해 내가 입력한 사원 정보를 사원별로 모두 출력하여 확인할 수 있다.

- 사유

제안서를 쓸 때 프로그램 기능을 구상하는 과정에서 사원 명단을 출력하는 기능을 누락되었다. 따라서 이번 진척보고서를 쓰는 과정에서 기능 1의 세부기능에 출력하는 기능도 구현하여 추가하였다.

3) 퇴사자 명단 추가 기능

- 이전

이전에는 단순히 퇴사한 사원의 정보를 사원 명단에 있는 정보들만 퇴사자 명단에 옮겨서 추가하여서 퇴사한 사원이 퇴사날짜가 어떻게 되는지 알 수 없었다.

- 이후

퇴사자 명단으로 이동할 때 추가로 퇴사날짜를 입력하여 해당 퇴사 사원이 언제 퇴사하였는지 퇴사날짜를 알 수 있도록 추가하였다.

- 사유

퇴사자 명단이라는 이름에 걸맞게 퇴사날짜까지 입력하여 저장할 수 있도록 함으로써 퇴사자 관리를 보다 원할히 할 수 있고 추후 퇴사 사원이 재입사를 할 시 퇴사날짜가 참고자료가 될 수 있으므로 퇴사날짜라는 데이터를 추가로 입력하여 저장하도록 세부 기능을 추가하였다.

4) 기능 1함수 헤더파일로 만들기

- 이전

main함수가 있는 코드파일 에 모든 사용자 정의 함수들이 선언 및 정의되어 main함수와 공존하고 있었다.

- 이후

사용자 정의 함수들을 모두 employee.h라는 헤더파일에서 선언하고 employee.c라는 곳에서 정의하도록 하였다

- 사유

사용자 정의 함수들을 모두 employee.h라는 헤더파일에서 선언하고 employee.c라는 곳에서 정의하도록 함으로써 main함수가 있는 코드파일의 가독성을 높이고 코드 중복 오류 방지를 하고 컴파일 시간을 효율적으로 만들었다.

5) FreeMemory_Employee/ FreeMemory_ResigendEmployee 추가

- 이전

프로그램 종료시에 동적 메모리가 할당된 구조체에 대해 동적 메모리를 해제하는 기능이 부족하였다.

- 이후

FreeMemory_Employee/FreeMemory_ResignedEmployee 함수를 만들어 free함수를 통해 동적 메모리가 할당된 구조체에 대해 동적 메모리를 해제하도록 하였으며 9.종료 기능에 추가하여 프로그램 종료를 의미하는 9를 입력하면 프로그램 종료와 동시에 동적 메모리 할당이 모두 해제되도록 하였다.

- 사유

동적메모리가 할당된 것을 해제함으로써 프로그램이 종료되었는데 동적으로 할당된 메모리가 해제되지 않고 남게 되어 발생할 수 있는 메모리 누수를 방지하고 자원을 효율적으로 관리할 수 있도록 하였다.

6) 메뉴 선택 화면 개편

- 이전

이전에는 원래 모든 메뉴가 처음 메뉴 화면에 모두 출력되어 9가지의 메뉴를 고르도록 하였었다.

- 이후

메뉴를 기능별로 분류하여 사원 관리, 퇴사 관리, 휴가 관리로 묶어서 각각 관리별로 메뉴를 선택해서 들어가면 그 기능 카테고리에 맞는 사용가능한 메뉴가 나오도록 개편하였다..

- 사유

단순히 모든 메뉴를 한 번에 나열하니 원하는 메뉴를 한 번에 찾기 어려웠고 시각적으로 보나 미적으로 보나 난잡하고 가독성이 떨어진다고 판단하여 좀더 간단하고 정갈하게 메뉴 화면을 기능 카테고리별로 나누고 각 세부기능은 기능 카테고리를 선택해서 들어가면 고를 수 있도록 하였다.

7) 연차 테이블과 휴가 테이블의 분리&총 연차 일수 입력 기능 추가

- 이전

초기 제안서에는 휴가 테이블안에 연차 항목까지 모두 포함하여 저장하도록 설정하였으며 휴가 명단을 추출하면 휴가 데이터 뿐만 아니라 연차 데이터까지도 동시에 출력되도록 하였었다. 또한 직원들의 기본 연차일수를 입력하는 기능이 없었다.

- 이후

휴가 구조체 안에서 연차 항목을 가져와서 새로운 연차 구조체를 선언하여 연차 데이터를 저장하도록 하였고 직원들에게 똑같이 부여되는 기본 연차일수를 일괄적으로 입력해주는 기능을 추가하였다. 또한 직원들의 기본 연차일수데이터를 입력하도록 함으로써 휴가를 사용하였을 경우 휴가기간 일수에 따라 자동으로 기본 연차 일수에서 차감되도록 하여 편의성은 높였다

- 사유

이전에는 코드상에서 기본 연차일수를 고정해 놓아서 회사마다 기본 연차가 다르면 코드를 수정하여야 하였지만 기본 연차일수를 입력할 수 있게 함으로써 회사마다 다른 기본 연차일수에 따라 각자 알아서 기본 연차일수를 입력하여 사용할 수 있도록 하여 범용성을 높였다

6. 느낀 점

이전에도 파이썬이나 R, SAS 등의 다른 프로그램 언어는 배워보았지만 C언어를 본격적으로 배운 건 이번 강의가 처음이어서 기말 프로젝트인 사원관리프로그램 만들기를 진행하면서 많은 어려움이 있었다. 하지만 프로그램을 만들어 나가는 과정에서 강의시간에 배운 이론들을 확실하게 자신의 것으로 만들 수 있었고 C 언어를 조금 더 능숙하게 다를 수 있는 계기가 되었다. 물론 이번 프로젝트를 하면서 조금 더 다양한 기능들을 넣지 못하고 코드를 더 간결하게 짜지 못한 것에 대한 아쉬움이 많이 남지만 혼자서 프로그램 코드를 처음부터 끝까지 다 짜본 경험은 빅데이터 관련 직무의 취업을 준비할 때 큰 도움이 될 것 같다.