C프로그래밍및실습

사원관리프로그램

진척 보고서 #2

제출일자: 2023.12.10
제출자명: 정지우
제출자학번: 181155

1. 프로젝트 목표

1) 배경 및 필요성

대기업이나 중견기업의 경우 값비싼 사원관리 프로그램을 구매하여 사용하지만 중소기업이나 스타트 업의 경우 자본금이 부족하기에 이런 사원관리 프로그램까지 구매해서 사용할 여력이 없거나 빠듯한 경우가 많다. 또한 지나치게 복잡하고 많은 기능들을 포함하고 있어 이로 인한 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) 차별 점

기존 사원관리 프로그램의 경우 값비싼 것은 둘째 치고 개인이 따라 만들기에는 매우 복잡하고 어려운 경우가 많았으며 필요 없는 기능도 다수 포함되어 있어 오히려 필요한 기능만 골라 쓰기 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. + 퇴사날짜 역시 입력하여 저장하도록 한다.

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. 진척사항

1) 퇴사자 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) 종료 시 동적 메모리 해제 기능

[FreeMemory_Employee/ FreeMemory_ResignedEmployee 함수]

<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_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
```

(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

<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 진척보고서1과 동일한 내용이다.

2)출력

-기능1의 함수들을 그대로 사용하여서 진척보고서1과 동일한 내용이다.

<설명>

Employee.h에서 선언된 EmployData 구조체,변수(num_people,capacity),FindPerson/Resize Table/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) 테스트 결과

(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

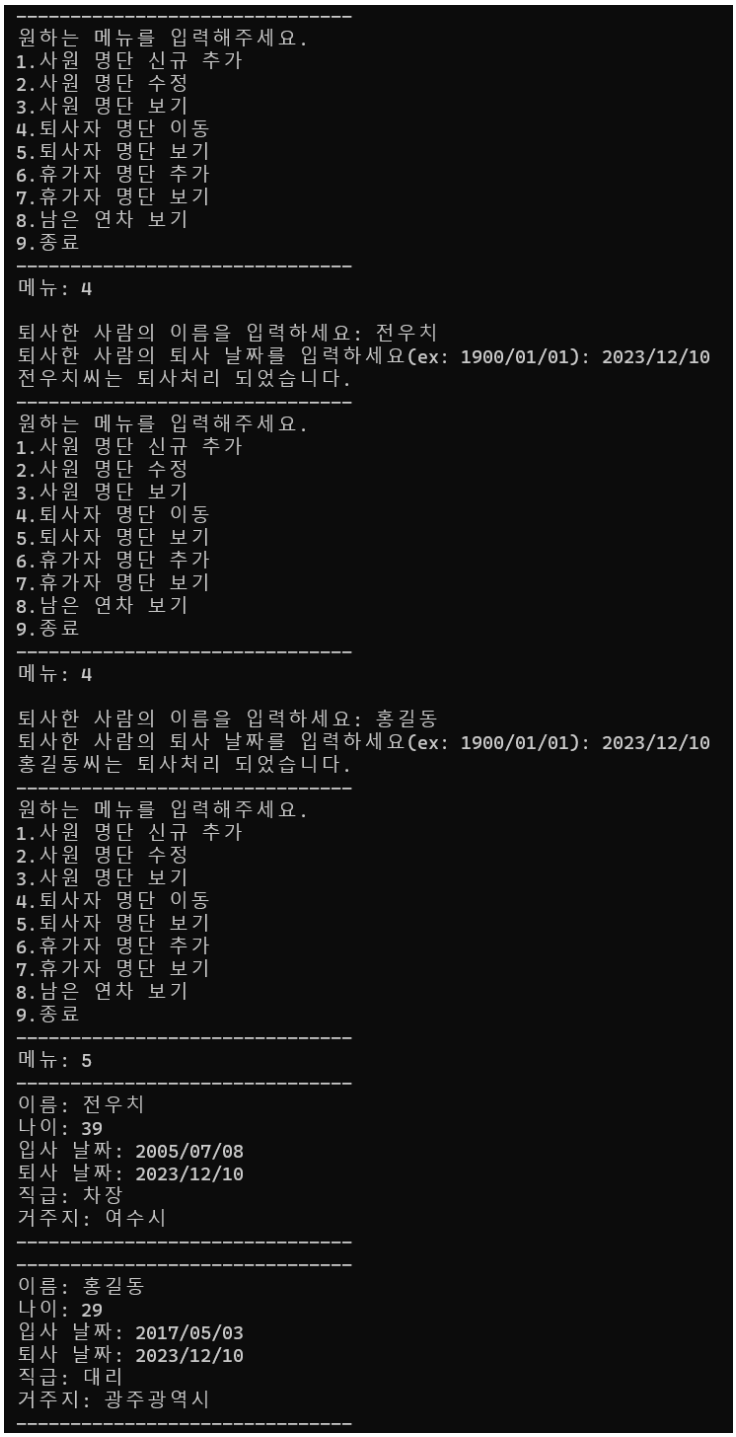
```
------------------------------
원하는 메뉴를 입력해주세요.
1.사원 명단 신규 추가
2.사원 명단 수정
3.사원 명단 보기
4.퇴사자 명단 이동
5.퇴사자 명단 보기
6.휴가자 명단 추가
7.휴가자 명단 보기
8.남은 연차 보기
9.종료
------------------------------
메뉴: 1
사원 이름을 입력하세요: 전우치
사원의 입사날짜를 입력하세요(ex:1900/01/01): 2005/07/08
사원의 거주지역을 입력하세요: 여수시
사원의 직급을 입력하세요: 차장
사원의 나이를 입력하세요: 39
------------------------------
원하는 메뉴를 입력해주세요.
1.사원 명단 신규 추가
2.사원 명단 수정
3.사원 명단 보기
4.퇴사자 명단 이동
5.퇴사자 명단 보기
6.휴가자 명단 추가
7.휴가자 명단 보기
8.남은 연차 보기
9.종료
------------------------------
메뉴: 4

퇴사한 사람의 이름을 입력하세요: 전우치
퇴사한 사람의 퇴사 날짜를 입력하세요(ex: 1900/01/01): 2023/12/10
전우치씨는 퇴사처리 되었습니다.
```

(2) 퇴사장 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

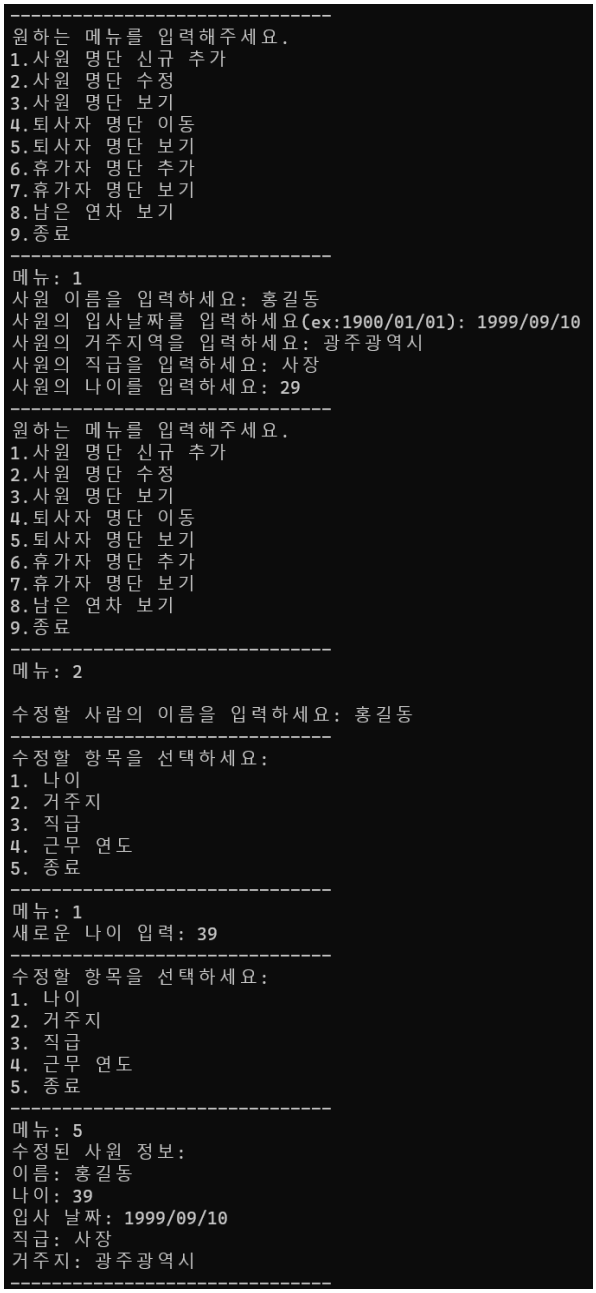


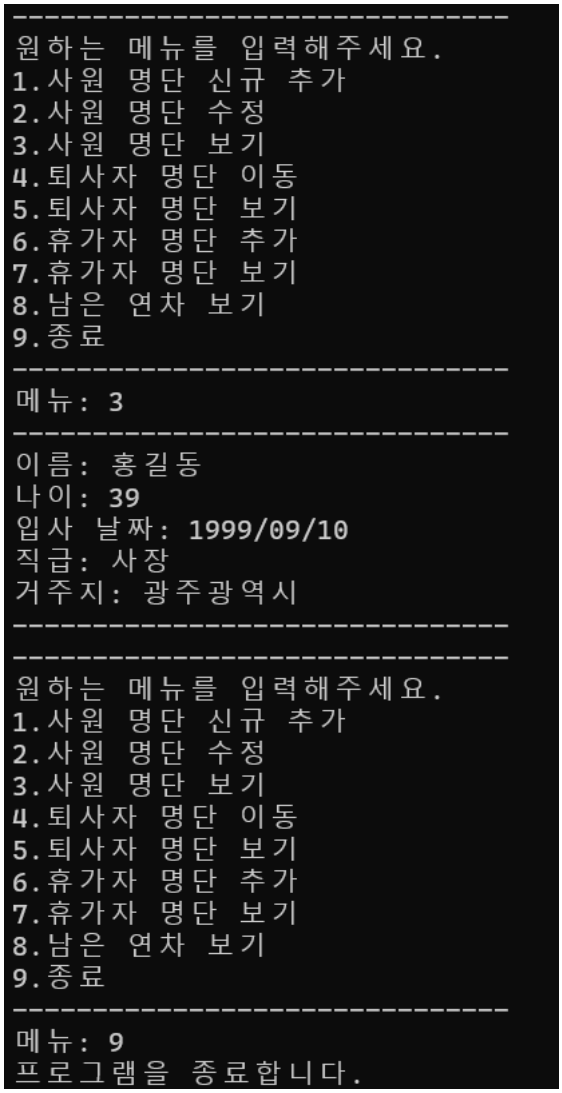
(3) 기능1 함수 헤더파일로 만들기

- 설명

이전 진척보고서1에서 구현하였던 기능1의 함수들을 모두 employee.h에서 선언하고 employee.c.에서 정의하여 main함수에서는 헤더파일로 불러와서 사용하도록 헤더파일로 만들었으며 기능1의 사원명단 추가/수정/출력 기능들을 모두 테스트해본 결과 정상작동 하였다.

- 테스트 결과 스크린샷





4. 계획 대비 변경 사항

1) 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 원할하게 할 수 있고 추후 퇴사 사원이 재입사를 할 시 퇴사날짜가 참고 자료가 될 수 있으므로 퇴사날짜라는 데이터를 추가로 입력하여 저장하도록 세부 기능을 추가하였다.

2) 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.

3) 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.

5. 프로젝트 일정

업무		11/3	11/6	11/24	12/10	12/23
제안서 작성		완료				
기능1	세부기능1		완료			
	세부기능2		완료			
	세부기능3		완료			
기능2				완료		
기능3	세부기능1				---------진행중---------->	
	세부기능2				---------진행중---------->	

기능1:사원 정보 관리 기능

->세부기능1:사원정보 입력

->세부기능1:사원정보 수정

->세부기능1:사원정보 출력

기능2:퇴사자 관리 기능

기능3:휴가(연차)관리 기능

->휴가자 명단 관리

->사원 연차 관리