C 프로그래밍 및 실습

가계부 자동화

진척 보고서 #2

제출일자: 2023-12-10
제출자명: 문준혁
제출자학번: 201595

1. 프로젝트 목표

1) 배경 및 필요성

자취를 시작하며 용돈 사용량이 늘어나고 소비의 종류도 다양하게 되었다. 예상에 없던 지출이 자주 생겨 계획적인 소비와 절약에 어려움을 느끼게 되었고 같은 상황이 반복되어 월 말에 경제난이 자주 발생하는 상황이다. 그래서 이러한 문제를 해결하기 위해 나만의 가계부를 체계적으로 작성해 현재 내 소비가 적당한지에 대하여 판단하게 해줄 프로그램이 필요하다고 느꼈다.

2) 프로젝트 목표

나의 과거에 대한 소비량과 현재의 소비량을 확인하여 현재 나의 소비가 적절한지를 판단하게 해주는 프로그램 코딩을 목표로 한다.

3) 차별 점

기존의 가계부들은 카테고리가 없이 소득과 지출만 보여주는 것이나 한 달을 기준으로 소득과 지출을 보여주어 해당 월 말에만 남은 잔액을 통해 나의 소비에 대한 판단을 할 수 있다. 이 프로그램은 프로그램 종료 시 마다 저번 달의 지출액 뿐만 아니라 이번 달 현재 시점의 지출을 비교하여 몇 퍼센트를 더 썼는지, 또는 덜 썼는지를 알려주어 남은 날 동안 소비를 어떻게 해야 할 지 확실히 알 수 있다. 또한 최근 3달간의 평균 지출을 알려주어 나의 최근 평균 지출에 대해서도 파악할 수 있다. 이러한 방법으로 평소에 비해 많이 쓰는 항목을 인지하고 사용자의 소비에 참고할 수 있게 할 것이다. 또한 계획과 달리 매달 일어나는 소비가 아닌 가끔 발생하는 소비, 예를 들어 여행, 병원비 등 이러한 것들로 인한 지출이 자연스럽게 반영되어 앞으로의 남은 날 동안의 소비에 참고할 수 있다.

2. 기능 계획

1) 기능 1: 메뉴 입력 받기

- 설명: '1. 지출 입력 2. 소득 입력 3. 지출 및 소득 내역 수정, 4. 특정 월 소득 및 지출 확인, 5. 종료' 목록을 보여준다

(1) 세부 기능 1: 출력된 메뉴 중 사용자가 원하는 메뉴 선택

- 설명: 출력된 메뉴 중 사용자가 원하는 메뉴를 입력 받고 실행한다.

2) 기능 2: 지출 입력 받고 저장

- 설명: 지출을 입력 받고 프로그램에 저장한다.

(1) 세부 기능 1: 지출 항목 및 내역 입력 받기

- 설명: 지출한 항목과 해당 항목의 금액을 사용자에게 입력 받는다.

3) 기능 3: 소득 입력 받고 저장

- 설명: 사용자가 얻은 소득을 입력 받고 프로그램에 저장한다.

(1) 세부 기능 1: N빵 결제 시 각 항목에 이체 받은 돈을 입력 받기

- 설명: 순전히 벌어들인 돈이 아닌 사람들과 나누어 낸 것을 돌려받는 경우 등 들어오는 모든 소득을 입력 받아 실제 사용한 금액만 반영한다.

4) 기능 4: 소득 및 지출 수정

- 설명: 사용자가 입력했던 소득 및 지출 내역을 수정한다.

(1) 세부 기능 1: 소득과 지출을 나누어 내역을 수정

- 설명: 메뉴 중 수정을 선택하면 소득과 지출을 다시 한 번 선택한 후 수정하고 싶은

내역을 수정한다.

5) 기능 5: 특정 월 소득 및 지출 확인

- 설명: 사용자가 원하는 달의 지출 내역과 소득 내역을 출력해준다

(1) 세부 기능 1: 소득과 지출의 내역의 총합을 같이 출력

- 설명: 내역 뿐만 아니라 총 금액의 합계도 같이 출력해준다.

6) 기능 6: 프로그램 종료 시 소비평가 출력

- 설명: 프로그램 종료 시 나의 소비 평가에 대해 출력해준다.

(1) 세부 기능 1: 해당 달의 지출의 총 합계와 소득의 총 합계를 출력

- 설명: 프로그램을 종료할 때 마다 내 현재 내역을 보며 이번 달 소비를 평가한다.

(2) 세부 기능 2: 저번 달과 비교해 이번 달 지출과 소득 비율, 남은 잔액을 출력

- 설명: 총액 뿐만 아니라 비율로 표시해 더 직관적으로 소비를 평가할 수 있게 해주며, 남은 잔액 역시 출력해준다.

(3) 세부 기능 3: 최근 세 달의 지출 및 소득의 총 합계 평균을 출력

- 설명: 이번 달 뿐만 아니라 최근 평균 소비와 같이 보며 내 현재 소비를 평가해준다.

6) 기능 7: 현재 날짜 인식

- 설명: 프로그램에게 현재 날짜를 알려준다.

(1) 세부 기능 1: 사용자가 최근 입력한 날짜를 현재 날짜로 변수에 저장

- 설명: 소비평가 출력 시 이번 달, 저번 달, 최근 세 달 평균 등을 계산하기 위해 현재 날짜를 프로그램에게 인식시킨다.

6) 기능 8: 파일 입출력

- 설명: 파일을 입력하고 출력한다.

(1) 세부 기능 1: 사용자가 입력한 내용을 파일에 저장

- 설명: 사용자가 프로그램에서 입력했던 내용을 파일에 모두 저장한다.

(2) 세부 기능 2: 저장된 파일을 프로그램 실행 시 불러옴

- 설명: 사용자가 지금까지 입력했던 내용들을 프로그램 실행 시 불러온다.

(3) 세부 기능 3: 저장된 내역에서 지출 내역과 소비 내역을 구별하여 구조체에 넣음

- 설명: 그저 텍스트만을 보여주는 것이 아니라 구조체에 넣어 모든 기능에 적용시킨다.

(4) 세부 기능 4: 구조체에 적용시킨 내역을 출력

- 설명: 적용된 내역들을 프로그램 시작 시 출력해준다.

3. 진척사항

1) 기능 구현

(1) 메뉴 입력 받기

- 입출력: 메뉴 출력 및 입력 받기

- 설명: 프로그램 시작 시 메뉴를 출력해주고 메뉴를 입력 받는다.

- 적용된 배운 내용: while문, switch문, 구조체, 함수

- 코드 스크린샷

```
while (!terminate) {
    printf("------------------₩n");
    printf("메뉴를 입력해주세요.₩n");
    printf(
        "1. 지출 입력₩n2. 소득 입력₩n3. 지출 및 소득 수정₩n4. 특정 달의 내역 "
        "확인₩n5. 종료₩n");
    printf("------------------₩n");

    scanf_s("%d", &choice);

    switch (choice) {
    case 1:  // 지출 입력 및 저장
        input_Expense(expenses);
        break;
    case 2:  // 소득 입력 및 저장
        input_Income(incomes);
        break;
    case 3:  // 지출 및 소득내역을 수정
        modify_list(expenses, incomes);
        break;
    case 4:  // 특정 월소득 및 지출 확인
        display_list(&selected_year, &selected_month, expenses, incomes,
                     expense_count, income_count);
        break;
    case 5:  // 종료 및 소비평가 출력
        printSummary(expenses, expense_count, incomes, income_count);
        compare_last_month(expenses, incomes, expense_count, income_count);
        average_last_3_months(expenses, incomes, expense_count, income_count);
        terminate = 1;
        break;

    default:
        printf("올바른 메뉴를 선택해주세요.₩n");
        break;
    }
}
```

(2) 지출 입력 받고 저장

- 입출력: 지출 입력 받기

- 설명: 지출 항목, 내역, 금액을 입력 받는 함수를 호출하여 실행하고 저장한다.

- 적용된 배운 내용: switch문, 문자열, 함수, 배열, 구조체

- 코드 스크린샷

```
void input_Expense(
    struct Account_Book expenses[]) {  // 지출 입력을 위한 함수 정의
  printf("저장할 날짜를 입력하세요(YY MM DD): ");
  scanf_s("%d %d %d", &expenses[expense_count].year,
          &expenses[expense_count].month, &expenses[expense_count].day);

  printf("지출 내역을 입력하세요: ");
  scanf_s("%19s", expenses[expense_count].description, 20);

  printf("지출 금액을 입력하세요: ");
  scanf_s("%f", &expenses[expense_count].amount);

  expense_count++;
  printf("%d년 %d월 %d일에 내역이 저장되었습니다.₩n",
         expenses[expense_count - 1].year, expenses[expense_count - 1].month,
         expenses[expense_count - 1].day);
}
```

(3) 소득 입력 받고 저장

- 입출력: 소득 입력 받기

- 설명: 소득 항목, 내역, 금액을 입력 받는 함수를 호출하여 실행하고 저장한다.

- 적용된 배운 내용: switch문, 문자열, 함수, 배열, 구조체

- 코드 스크린샷

```
void input_Income(struct Account_Book incomes[]) {  // 소득 입력을 위한 함수
                                                    // 정의
  printf("저장할 날짜를 입력하세요(YY MM DD): ");
  scanf_s("%d %d %d", &incomes[income_count].year, &incomes[income_count].month,
          &incomes[income_count].day);

  printf("소득 내역을 입력하세요: ");
  scanf_s("%19s", incomes[income_count].description, 20);

  printf("소득 금액을 입력하세요: ");
  scanf_s("%f", &incomes[income_count].amount);

  income_count++;
  printf("%d년 %d월 %d일에 내역이 저장되었습니다.₩n",
         incomes[income_count - 1].year, incomes[income_count - 1].month,
         incomes[income_count - 1].day);
}
```

(4) 소득 및 지출 수정

- 입출력: 입력 받은 소득 및 지출을 수정을 위해 재입력 받기

- 설명: 먼저 날짜를 입력 받고, 소득과 지출을 선택하여 수정할 항목, 내역, 금액을 입력

받는 함수를 호출한다. 소득 수정 함수와 지출 수정 함수를 각각 정의하여 호출한다.

- 적용된 배운 내용: switch문, 조건문, 반복문, 문자열, 함수, 배열, 구조체

- 코드 스크린샷

```
void modify_list(
    struct Account_Book expenses[],  // 지출, 소득 내역 수정 함수 정의
    struct Account_Book incomes[]) {
  int choice;
  printf("수정할 내역을 선택하세요:₩n");
  printf("1. 지출 내역 수정₩n");
  printf("2. 소득 내역 수정₩n");
  printf("3. 돌아가기₩n");
  scanf_s("%d", &choice);

  switch (choice) {
    case 1:
      modify_Expense(expenses);  // 지출을 수정
      break;
    case 2:
      modify_Income(incomes);  // 소득을 수정
      break;
    case 3:
      printf("돌아갑니다.₩n");
      break;
    default:
      printf("올바른 메뉴를 선택하세요.₩n");
      break;
  }
}
```

```c
void modify_Expense(struct Account_Book expenses[]) {  // 지출 수정 함수 정의
    int input_year, input_month, input_day;
    printf("수정할 날짜를 입력하세요(YY MM DD): ");
    scanf_s("%d %d %d", &input_year, &input_month, &input_day);

    for (int i = 0; i < expense_count; i++) {
        if (expenses[i].year == input_year && expenses[i].month == input_month &&
            expenses[i].day == input_day) {
            printf("수정할 지출 내역을 선택하세요: ");
            printf("%d월 %d일 - %s, %.2f원₩n", expenses[i].month, expenses[i].day,
                expenses[i].description, expenses[i].amount);

            printf("새로운 지출 내역을 입력하세요: ");
            scanf_s("%s19", expenses[i].description, sizeof(expenses[i].description));

            printf("수정된 지출 금액을 입력하세요: ");
            scanf_s("%f", &expenses[i].amount);

            printf("%d년 %d월 %d일의 지출 내역이 수정되었습니다.₩n", input_year,
                input_month, input_day);
            return;
        }
    }
    printf("%d년 %d월 %d일에 저장된 내역이 없습니다.₩n", input_year, input_month,
        input_day);
}
```

```c
void modify_Income(struct Account_Book incomes[]) {  // 소득 수정 함수 정의
    int input_year, input_month, input_day;
    printf("수정할 날짜를 입력하세요(YY MM DD): ");
    scanf_s("%d %d %d", &input_year, &input_month, &input_day);

    for (int i = 0; i < income_count; i++) {
        if (incomes[i].year == input_year && incomes[i].month == input_month &&
            incomes[i].day == input_day) {
            printf("수정할 소득 내역을 선택하세요: ");
            printf("%d월 %d일 - %s, %.2f원₩n", incomes[i].month, incomes[i].day,
                incomes[i].description, incomes[i].amount);

            printf("새로운 소득 내역을 입력하세요: ");
            scanf_s("%s19", incomes[i].description, sizeof(incomes[i].description));

            printf("수정된 소득 금액을 입력하세요: ");
            scanf_s("%f", &incomes[i].amount);

            printf("%d년 %d월 %d일의 소득 내역이 수정되었습니다.₩n", input_year,
                input_month, input_day);
            return;
        }
    }
    printf("%d년 %d월 %d일에 저장된 내역이 없습니다.₩n", input_year, input_month,
        input_day);
}
```

(5) 특정 월 소득 및 지출 확인

- 입출력: 원하는 달을 입력 받고 소득 및 지출 내역과 총합 출력

- 설명: 사용자가 확인하고 싶은 달을 입력 받고 소득 및 지출 내역을 출력해주는 함수를 호출한다. 그리고 총합을 출력해주는 함수를 따로 정의하여 같이 호출해준다.

- 적용된 배운 내용: switch문, 반복문, 조건문, 문자열, 함수, 배열, 포인터, 구조체

- 코드 스크린샷

```
void display_list(int *year, int *month, struct Account_Book expenses[],
                  struct Account_Book incomes[], int expense_count,
                  int income_count) {
  printf("년도와 월을 입력하세요 (YY MM): ");
  scanf_s("%d %d", year, month);
  printf("%d년 %d월의 내역을 출력합니다.₩n", *year, *month);

  printf("지출 내역:₩n");
  for (int i = 0; i < expense_count; i++) {
    if (expenses[i].year == *year && expenses[i].month == *month) {
      printf("%d월 %d일: %s - %.2f원₩n", expenses[i].month, expenses[i].day,
             expenses[i].description, expenses[i].amount);
    }
  }

  printf("수입 내역:₩n");
  for (int i = 0; i < income_count; i++) {
    if (incomes[i].year == *year && incomes[i].month == *month) {
      printf("%d월 %d일: %s - %.2f원₩n", incomes[i].month, incomes[i].day,
             incomes[i].description, incomes[i].amount);
    }
  }

  calculateTotal(expenses, incomes, expense_count, income_count, *year, *month);
}
```

```c
void calculateTotal(struct Account_Book expenses[],
                    struct Account_Book incomes[], int expense_count,
                    int income_count, int year, int month) {
    float totalExpenses = 0.0;
    float totalIncomes = 0.0;

    for (int i = 0; i < expense_count; ++i) {
        if (expenses[i].year == year && expenses[i].month == month) {
            totalExpenses += expenses[i].amount;
        }
    }

    for (int i = 0; i < income_count; ++i) {
        if (incomes[i].year == year && incomes[i].month == month) {
            totalIncomes += incomes[i].amount;
        }
    }

    printf("₩n₩n%d년 %d월의 총 지출: %.2f원₩n₩n", year, month, totalExpenses);
    printf("₩n₩n%d년 %d월의 총 수입: %.2f원₩n₩n", year, month, totalIncomes);
}
```

(6) 프로그램 종료 시 이번 달 지출 및 소득 총합 출력

- 입출력: 종료 메뉴 선택 시 이번 달 지출 및 소득 총합 출력

- 설명: 사용자가 프로그램을 종료 메뉴를 선택하면 자동으로 이번 달 지출 및 소득의 총합을 출력해주는 함수와 저번 달 대비 이번 달 지출 및 소득 증감 비율을 출력해주는 함수, 그리고 최근 3달 간 지출 및 소득 평균을 출력해주는 함수를 호출 후 종료한다.

- 적용된 배운 내용: switch문, 반복문, 조건문, 함수, 배열, 포인터, 구조체

- 코드 스크린샷

```c
// 종료 시 이번 달 지출 및 소득 총합 출력 함수 정의
void printSummary(struct Account_Book expenses[], int expense_count,
                  struct Account_Book incomes[], int income_count) {
    int currentYear = 0, currentMonth = 0;  // 현재 년도와 월을 저장할 변수

    decideCurrent(expenses, incomes, expense_count, income_count, &currentYear,
                  &currentMonth);

    float currentMonthExpense = 0.0, currentMonthIncome = 0.0;

    for (int i = 0; i < expense_count; ++i) {  // 지출에서 이번 달을 계산
        if (expenses[i].year == currentYear && expenses[i].month == currentMonth) {
            currentMonthExpense += expenses[i].amount;
        }
    }

    for (int i = 0; i < income_count; ++i) {  // 소득에서 이번 달을 계산
        if (incomes[i].year == currentYear && incomes[i].month == currentMonth) {
            currentMonthIncome += incomes[i].amount;
        }
    }

    printf("₩n₩n%d년 %d월의 지출 합: %.f원₩n₩n", currentYear, currentMonth,
           currentMonthExpense);
    printf("₩n₩n%d년 %d월의 소득 합: %.f원₩n₩n", currentYear, currentMonth,
           currentMonthIncome);
}
```

```c
// 종료 시 이번 달 지출, 소득과 저번 달 지출, 소득 비율 증감 출력 함수 정의
void compare_last_month(struct Account_Book expenses[],
                        struct Account_Book incomes[], int currentYear,
                        int currentMonth) {
  decideCurrent(expenses, incomes, expense_count, income_count, &currentYear,
                &currentMonth);

  float last_month_expenses = 0, last_month_incomes = 0;
  for (int i = 0; i < expense_count; ++i) {  // 지출에서 저번 달을 계산
    if (expenses[i].year == currentYear &&
        expenses[i].month == currentMonth - 1) {
      last_month_expenses += expenses[i].amount;
    }
  }

  for (int i = 0; i < income_count; ++i) {  // 소득에서 저번 달을 계산
    if (incomes[i].year == currentYear &&
        incomes[i].month == currentMonth - 1) {
      last_month_incomes += incomes[i].amount;
    }
  }

  float this_month_expenses = 0, this_month_incomes = 0;
  for (int i = 0; i < expense_count; ++i) {  // 지출에서 이번 달을 계산
    if (expenses[i].year == currentYear && expenses[i].month == currentMonth) {
      this_month_expenses += expenses[i].amount;
    }
  }

  for (int i = 0; i < income_count; ++i) {  // 소득에서 이번 달을 계산
    if (incomes[i].year == currentYear && incomes[i].month == currentMonth) {
      this_month_incomes += incomes[i].amount;
    }
  }

  float expense_ratio = this_month_expenses / last_month_expenses;
  float income_ratio = this_month_incomes / last_month_incomes;

  printf("₩n₩n저번 달 대비 지출 증감율 : %.2f₩n₩n", expense_ratio);
  printf("₩n₩n저번 달 대비 소득 증감율 : %.2f₩n₩n", income_ratio);
  printf("₩n₩n남은 잔액 : %.2f₩n₩n", this_month_incomes - this_month_expenses);
}
```

```c
// 종료 시 최근 3달 지출 및 소득 합계 평균 출력 함수 선언
void average_last_3_months(struct Account_Book expenses[],
                           struct Account_Book incomes[], int currentYear,
                           int currentMonth) {
    decideCurrent(expenses, incomes, expense_count, income_count, &currentYear,
                  &currentMonth);  // 현재 날짜 계산

    float total_expenses = 0, total_incomes = 0;
    int count_expenses = 0, count_incomes = 0;

    for (int i = 0; i < expense_count; ++i) {  // 지출에서 최근 3달을 계산
        if ((expenses[i].year == currentYear &&
             expenses[i].month == currentMonth - 1) ||
            (expenses[i].year == currentYear &&
             expenses[i].month == currentMonth - 2) ||
            (expenses[i].year == currentYear &&
             expenses[i].month == currentMonth - 3)) {
            total_expenses += expenses[i].amount;
            count_expenses++;
        }
    }

    for (int i = 0; i < income_count; ++i) {  // 소득에서 최근 3달을 계산
        if ((incomes[i].year == currentYear &&
             incomes[i].month == currentMonth - 1) ||
            (incomes[i].year == currentYear &&
             incomes[i].month == currentMonth - 2) ||
            (incomes[i].year == currentYear &&
             incomes[i].month == currentMonth - 3)) {
            total_incomes += incomes[i].amount;
            count_incomes++;
        }
    }

    float avg_expenses = total_expenses / 3;
    float avg_incomes = total_incomes / 3;

    printf("₩n₩n최근 3개월 평균 지출: %.2f₩n₩n", avg_expenses);
    printf("₩n₩n최근 3개월 평균 소득: %.2f₩n₩n", avg_incomes);
}
```

(7) 현재 날짜 인식

- 입출력: 현재 날짜를 변수에 저장

- 설명: 사용자가 마지막으로 입력한 날짜를 현재 날짜로 인식하여 변수에 저장한다.

- 적용된 배운 내용: 반복문, 조건문, 함수, 배열, 포인터, 구조체

- 코드 스크린샷

```
void decideCurrent(
    struct Account_Book expenses[],  // 마지막 지출, 소득 입력 날짜를 현재
                                     // 날짜로 인식하게 하는 함수 정의
    struct Account_Book incomes[], int expense_count, int income_count,
    int *currentYear, int *currentMonth) {
  for (int i = 0; i < expense_count; ++i) {
    if (expenses[i].year > *currentYear ||
        (expenses[i].year == *currentYear &&
         expenses[i].month > *currentMonth)) {
      *currentYear = expenses[i].year;
      *currentMonth = expenses[i].month;
    }
  }

  for (int i = 0; i < income_count; ++i) {
    if (incomes[i].year > *currentYear ||
        (incomes[i].year == *currentYear && incomes[i].month > *currentMonth)) {
      *currentYear = incomes[i].year;
      *currentMonth = incomes[i].month;
    }
  }
}
```

(8) 파일 입출력

- 입출력: 입력한 내용을 파일에 저장 및 프로그램 실행 시 저장된 파일 불러오기

- 설명: 사용자가 프로그램에서 입력한 내용을 모두 메모장에 저장하고 실행할 때마다 파일을 불러온다. 불러올 때 expense list와 income list를 문자열 함수를 통해 구별하여 불러오는 즉시 구조체에 넣고 출력해준다.

- 적용된 배운 내용: 반복문, 조건문, 함수, 배열, 문자열, 포인터, 구조체, 파일 입출력

- 코드 스크린샷

```c
// 파일을 account_list 메모장에 저장하는 함수 정의
void saveAccount(struct Account_Book expenses[], int expense_count,
                 struct Account_Book incomes[], int income_count) {
  FILE *file;
  fopen_s(&file, "account_list.txt", "w");  // 파일 열기
  //불러온 파일들이 종료 시 메모장에 다시 저장되기 때문에 덮어쓰기 기능으로 선택

  if (file != NULL) {
    if (expense_count > 0 ||
        income_count > 0) {  // 지출이나 소득이 있을 때만 파일에 저장
      fprintf(file, "Expense list:₩n");  // 지출 내역 저장
      for (int i = 0; i < expense_count; ++i) {
        fprintf(file, "%d년 %d월 %d일 - %s, %.f원₩n", expenses[i].year,
                expenses[i].month, expenses[i].day, expenses[i].description,
                expenses[i].amount);
      }

      fprintf(file, "₩nIncome list:₩n");  // 소득 내역 저장
      for (int i = 0; i < income_count; ++i) {
        fprintf(file, "%d년 %d월 %d일 - %s, %.f원₩n", incomes[i].year,
                incomes[i].month, incomes[i].day, incomes[i].description,
                incomes[i].amount);
      }
      printf("₩n파일에 데이터를 추가로 저장했습니다.₩n");
    } else {
      printf("₩n데이터를 입력하지 않아 파일에 내용을 저장하지 않습니다.₩n");
    }
    fclose(file);
  } else {
    printf("파일을 저장할 수 없습니다.₩n");
  }
}
```

```c
// 파일을 account_list 메모장에서 불러오는 함수 정의
void loadAccounts(struct Account_Book expenses[], struct Account_Book incomes[],
                  int *expense_count, int *income_count, const char *fileName) {
  FILE *file;
  fopen_s(&file, fileName, "r");  // 파일 열기 (읽기 모드)

  if (file != NULL) {
    char buffer[100];
    int is_expense = 0;  // 파일에서 지출과 소득을 구별하기 위한 변수

    while (fgets(buffer, sizeof(buffer), file) != NULL) {
      int year, month, day;
      char description[50];
      float amount;

      if (strstr(buffer, "Expense list:") != NULL) {
        is_expense = 1;  // 'Expense list'를 발견하면 지출로 인식
      } else if (strstr(buffer, "Income list:") != NULL) {
        is_expense = 0;  // 'Income list'를 발견하면 소득으로 인식
      }

      if (is_expense) {  // 지출인 경우
        if (sscanf(buffer, "%d년 %d월 %d일 - %49[^,], %f원₩n", &year, &month,
                   &day, description, &amount) == 5) {
          expenses[*expense_count].year = year;
          expenses[*expense_count].month = month;
          expenses[*expense_count].day = day;
          strcpy_s(expenses[*expense_count].description,
                   sizeof(expenses[*expense_count].description), description);
          expenses[*expense_count].amount = amount;
          (*expense_count)++;
        }
      } else {  // 소득인 경우
        if (sscanf(buffer, "%d년 %d월 %d일 - %49[^,], %f원₩n", &year, &month,
                   &day, description, &amount) == 5) {
          incomes[*income_count].year = year;
          incomes[*income_count].month = month;
          incomes[*income_count].day = day;
          strcpy_s(incomes[*income_count].description,
                   sizeof(incomes[*income_count].description), description);
          incomes[*income_count].amount = amount;
          (*income_count)++;
        }
      }
    }
    fclose(file);
  } else { printf("파일을 열 수 없습니다.₩n");
  }
}
```

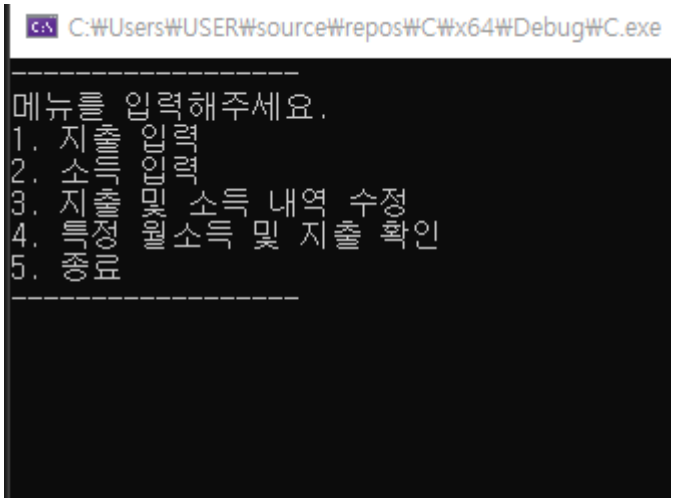
```c
//불러온 파일 내용 출력하는 함수 선언
void displayAccounts(struct Account_Book accounts[], int count) { //불러온 파일 출력
  printf("내역:₩n");
  for (int i = 0; i < count; i++) {
    printf("%d년 %d월 %d일: %s - %.f원₩n", accounts[i].year, accounts[i].month,
           accounts[i].day, accounts[i].description, accounts[i].amount);
  }
}
```

2) 테스트 결과

(1) 메뉴 입력 받기

- 설명: 프로그램 시작 시 메뉴를 출력해주고 switch문으로 메뉴를 입력 받는다.

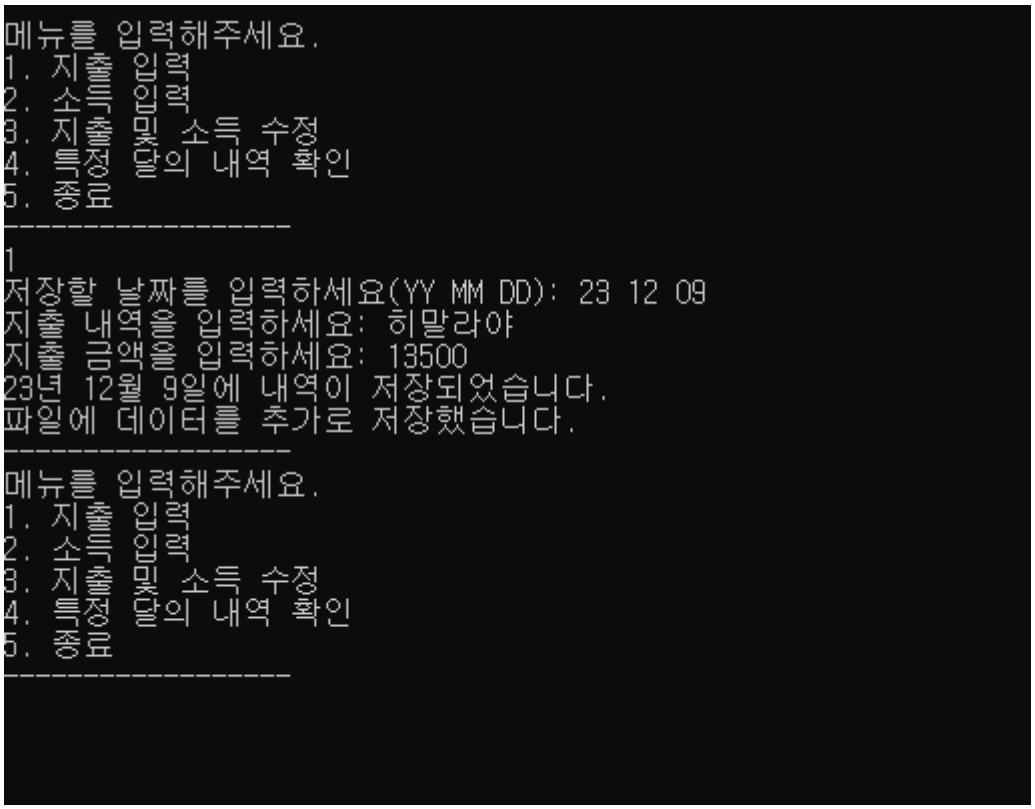
- 테스트 결과 스크린샷



(2) 지출 입력 받고 저장

- 설명: 지출 항목, 내역, 금액을 입력 받는 함수를 호출하여 실행하고 저장한다.

- 테스트 결과 스크린샷



(3) 소득 입력 받고 저장

- 설명: 소득 항목, 내역, 금액을 입력 받는 함수를 호출하여 실행하고 저장한다.

- 테스트 결과 스크린샷

```
------------------
메뉴를 입력해주세요.
1. 지출 입력
2. 소득 입력
3. 지출 및 소득 수정
4. 특정 달의 내역 확인
5. 종료
------------------
2
저장할 날짜를 입력하세요(YY MM DD): 23 12 09
소득 내역을 입력하세요: 용돈
소득 금액을 입력하세요: 50000
23년 12월 9일에 내역이 저장되었습니다.
파일에 데이터를 추가로 저장했습니다.
------------------
메뉴를 입력해주세요.
1. 지출 입력
2. 소득 입력
3. 지출 및 소득 수정
4. 특정 달의 내역 확인
5. 종료
------------------
```

(4) 소득 및 지출 수정

- 설명: 먼저 날짜를 입력 받고, 소득과 지출을 선택하여 수정할 항목과 금액을 입력 받는 함수를 호출한다. 소득 수정 함수와 지출 수정 함수를 각각 정의하여 호출한다.

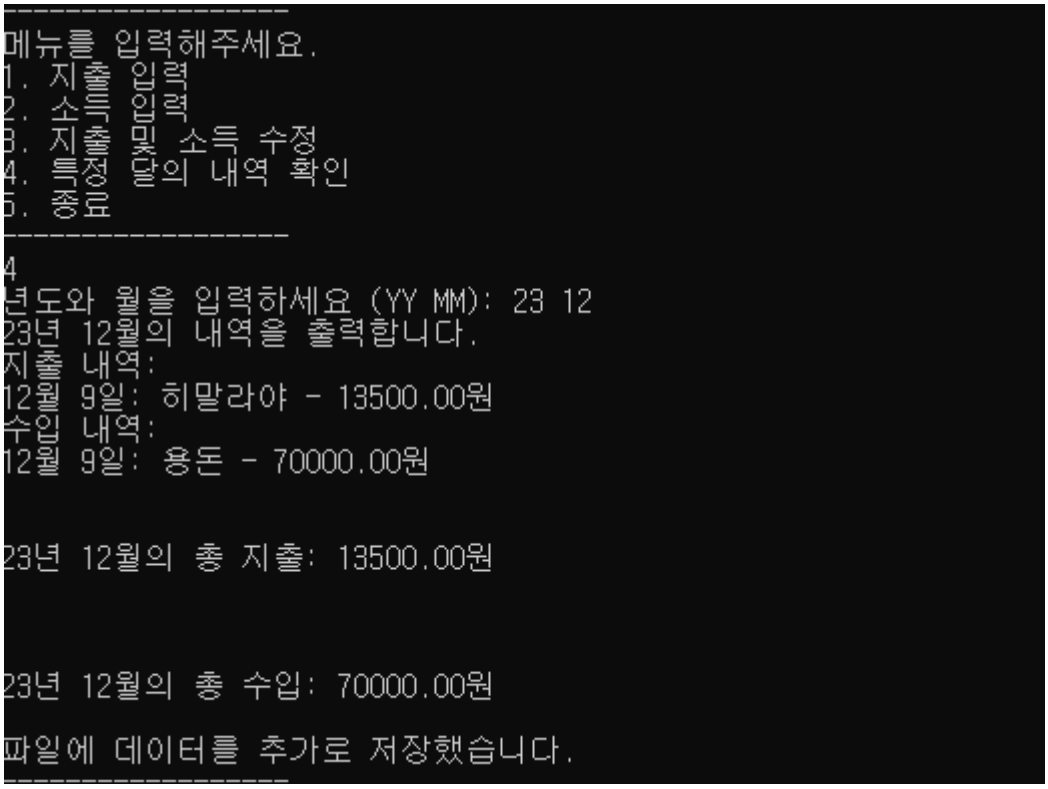
- 테스트 결과 스크린샷

```
------------------
메뉴를 입력해주세요.
1. 지출 입력
2. 소득 입력
3. 지출 및 소득 수정
4. 특정 달의 내역 확인
5. 종료
------------------
3
수정할 내역을 선택하세요:
1. 지출 내역 수정
2. 소득 내역 수정
3. 돌아가기
2
수정할 날짜를 입력하세요(YY MM DD): 23 12 09
수정할 소득 내역을 선택하세요: 12월 9일 - 용돈, 50000.00원
새로운 소득 내역을 입력하세요: 용돈
수정된 소득 금액을 입력하세요: 70000
23년 12월 9일의 소득 내역이 수정되었습니다.
파일에 데이터를 추가로 저장했습니다.
------------------
```

(5) 특정 월 소득 및 지출 확인

- 설명: 사용자가 확인하고 싶은 달을 입력 받고 소득 및 지출 내역을 출력해주는 함수를 호출한다. 그리고 총합을 출력해주는 함수를 따로 정의하여 같이 호출해준다.

- 테스트 결과 스크린샷



(6) 프로그램 종료 시 이번 달 지출 및 소득 총합 출력

- 설명: 사용자가 프로그램을 종료 메뉴를 선택하면 자동으로 이번 달 지출 및 소득의 총합을 출력해주는 함수, 저번 달 대비 이번 달 지출 및 소득 증감 비율을 출력해주는 함수, 그리고 최근 3달 간의 지출 및 소득 평균을 출력해주는 함수를 호출 후 종료한다.

- 테스트 결과 스크린샷

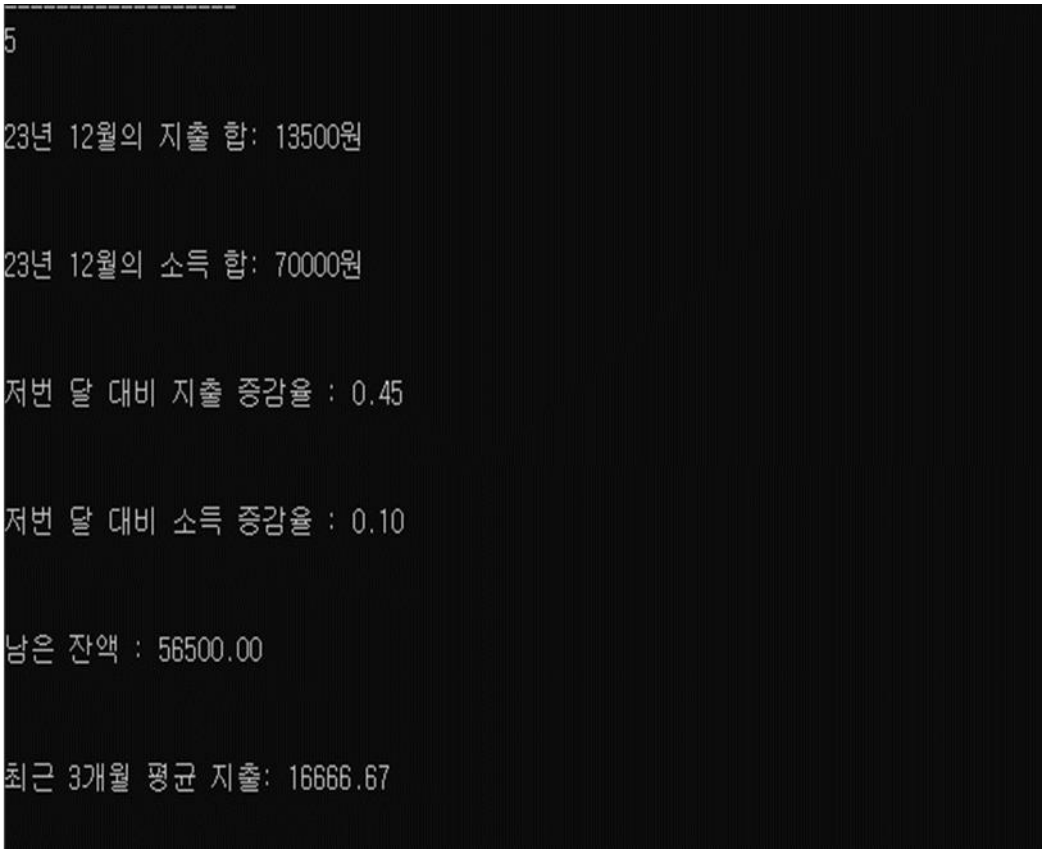
```
메뉴를 입력해주세요.
1. 지출 입력
2. 소득 입력
3. 지출 및 소득 수정
4. 특정 달의 내역 확인
5. 종료
------------------
5

23년 12월의 지출 합: 13500원


23년 12월의 소득 합: 70000원


저번 달 대비 지출 증감율 : 0.45


저번 달 대비 소득 증감율 : 0.10


남은 잔액 : 56500.00


최근 3개월 평균 지출: 16666.67


최근 3개월 평균 소득: 493333.34
파일에 데이터를 추가로 저장했습니다.
```

(7) 현재 날짜 인식

- 설명: 사용자가 입력한 최근 날짜를 현재 날짜로 인식하여 변수에 저장한다.

- 테스트 결과 스크린샷

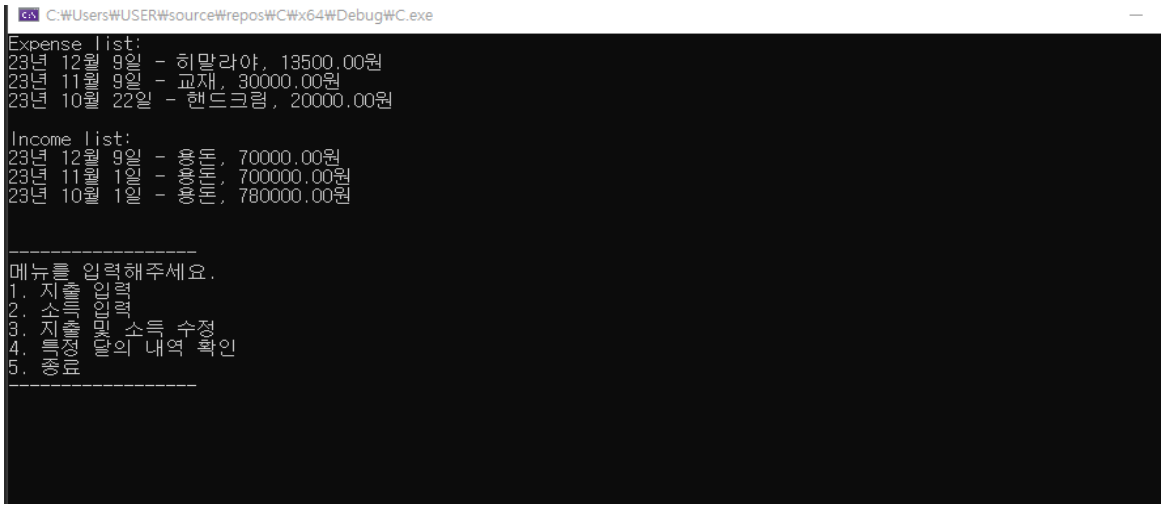
=> 위의 소비평가와 같은 스크린 샷인데 10월부터 12월까지 날짜를 무작위 순으로 지출과 소득을 각각 입력했지만 현재 날짜가 12월로 잘 적용된 코드인 것을 확인할 수 있다.



(8) 파일 입출력

- 설명: 사용자가 프로그램에서 입력한 내용을 메모장에 저장하고 실행할 때마다 파일을 불러온다. 그저 텍스트만을 불러오는 것이 아니라 지출과 소득 내역을 구별해 구조체에 넣은 후 출력해준다.

- 테스트 결과 스크린샷



4. 계획 대비 변경 사항

1) 현재 날짜 인식

- 이전-

해당사항 없음

- 이후-

[기능 7]: 현재 날짜 인식

- 설명: 프로그램에게 현재 날짜를 알려준다.

(1) 세부 기능 1: 사용자가 최근 입력한 날짜를 현재 날짜로 변수에 저장

- 설명: 소비평가 출력 시 이번 달, 저번 달, 최근 세 달 평균 등을 계산하기 위해 현재 날짜를 인식시킨다.

- 사유-

현재 날짜를 기준으로 프로그램에 저번 달과 저, 저번 달 등의 입력을 적용시키기 위해 프로그램에게 프로그램 사용 당일 날짜 인식의 필요성을 느꼈다.

2) 파일 입출력

- 이전-

해당사항 없음

- 이후-

[기능 8]: 현재 날짜 인식

- 설명: 파일을 입력하고 출력한다.

(1) 세부 기능 1: 사용자가 입력한 내용을 파일에 저장

- 설명: 사용자가 프로그램에서 입력했던 내용을 파일에 모두 저장한다.

(2) 세부 기능 2: 저장된 파일을 프로그램 실행 시 불러옴

- 설명: 사용자가 지금까지 입력했던 내용들을 프로그램 실행 시 불러온다.

(3) 세부 기능 3: 저장된 내역에서 지출 내역과 소비 내역을 구별하여 구조체에 넣음

- 설명: 그저 텍스트만을 보여주는 것이 아니라 구조체에 넣어 모든 기능에 적용시킨다.

(4) 세부 기능 4: 구조체에 적용시킨 내역을 출력

- 설명: 적용된 내역들을 프로그램 시작 시 출력해준다.

- 사유-

가계부 프로그램 특성을 고려했을 때 기존의 지출 및 소득의 과거 자료의 필요성을 느꼈고, 또한 내가 입력한 것들이 종료할 때마다 사라지는 것은 프로그램의 존재 의미가 사라지기 때문에 추가했다. 또한 파일을 여러 번 실행해보며 내역이 텍스트만을 불러오는 것을 확인한 후 구별하여 구조체에 넣는 기능을 추가하게 되었고, 같은 내역을 똑같이 저장하는 것을 발견하고 덮어서 저장하는 기능을 추가했다

5. 프로젝트 일정

업무		11/3	11/10	11/17	11/25
제안서 작성		완료			
기능1	세부기능1	---------->			
기능2	세부기능1	---------->			
기능3	세부기능1	---------->			
기능4	세부기능1		---------->		

기능5	세부기능1		---------->

업무		11/17	11/25	12/01	12/08
기능6	세부기능1	---------->			
프로그램 코드 오류 수정		---------->			
진척 보고서1 작성		---------->			
구조체로 프로그램 수정			---------->		
기능6	세부기능2		---------->		
	세부기능3		---------->		
기능7	세부기능1		---------->		

업무		12/01	12/10	12/15	12/22
기능8	세부기능1	---------->			
	세부기능2	---------->			
	세부기능3	---------->			
	세부기능4	---------->			
진척 보고서2 작성		---------->			
파일 내용 입력			---------->		
프로그램 검토 및 수정				---------->	
최종 보고서 작성				---------->	