c프로그래밍및 실습

지출 관리 자동화 구현

최종 보고서

제출일자: 23/12/24
제출자명: 이나경
제출자학번: 231135

1. 프로젝트 목표

1) 배경 및 필요성

제한된 예산 안에서 생활해야 하는 만큼 당월 사용할 수 있는 예산과 지출 분야 등을 확실히 기록하고 분석하여 지출을 관리하는 것은 필수적입니다. 그러나 매달마다 수동으로 예산을 계획하고 기록하는 것은 시간이 많이 들고 번거로울 뿐 아니라, 수동으로 수를 계산해야 하다 보니 오류의 가능성이 높습니다. 이 문제를 해결하기 위해 자동으로 지출을 저장 및 계산해주는 프로그램이 필요합니다.

2) 프로젝트 목표

사용자가 입력하는 지출을 자동으로 저장, 분리 및 관리하여 사용자가 보다 편하게 돈을 관리할 수 있도록 하는 것이 목표입니다.

3) 차별점

기존 프로그램들은 전기세나 가스비 등 필수 지출 내용으로 실제로 사용할 수 없는 돈들을 사용할 수 있는 돈과 분리하여 제공하지 않아 돈이 많이 있는 것으로 착각할 수도 있다는 문제가 있습니다. 따라서 필수 지출 금액의 날짜와 금액을 미리 입력받아 사용 가능한 금액과 분리하여 저장하는 것에 기존 프로그램과 차별점이 있습니다. 또한 예산 계획을 짜는 것에 관여하지 않는 기존 프로그램과 달리, 우리는 지난 달 사용하지 않고 남았거나 부족한 예산의 비율을 기준으로 이번 예산을 일정 수치만큼 늘리거나 줄이는 것이 어떻나고 제안하는 점에서 차별점을 가집니다.

2. 기능 계획

1) 기능 1 : 당월 예산 저장 기능.

- 설명 : 분류별로 사용할 예산과 총액을 입력받습니다.

(1) 세부 기능 1: 필수 지출 금액 저장.

- 설명: 지출이 필수적으로 발생할 금액을 입력받아 총액과 따로 저장합니다.

2) 기능 2 : 카테고리 편집 기능

- 설명: 지출을 따로 저장할 카테고리를 편집합니다.

(1) 세부 기능 1: 카테고리 삭제 기능

- 설명: 사용하지 않는 카테고리를 삭제합니다. 카테고리 내에 금액이 있을 경우에는 삭제를 차단합니다.

3) 기능 3 : 지출 분석 기능

- 설명: 최종적으로 입력이 끝났음을 알리면 초기 예산과 사용 예산을 비교합니다.

(1) 세부 기능 1: 소비 금액 계산

- 설명: 각 예산별 사용 금액과 총 소비액을 제시합니다.

(2) 세부 기능 2: 달성도 평가

- 설명: 예산을 아낀 정도에 의한 달성도를 각각 평가합니다.

4) 기능 4 : 예산 설정 추천 기능

- 설명: 새로운 계획을 생성할 때 지난 달 초과했거나 남았던 예산을 기반으로 예산 추천 금액을 설정합니다.

3. 기능 구현

(1) 카테고리 삭제 기능

- 입력: struct Expenditure *exp: 예산, 필수 지출 금액, 지출 발생 현황등이 저장되어있는 구조체로, 이 기능에서는 budget(예산)만을 사용합니다.

category_names: 카테고리의 이름이 저장된 2차원 배열

category_count: 저장된 카테고리의 수가 저장된 정수입니다.

출력: 선택한 카테고리가 삭제된 category_names, 삭제된만큼 적어진 category_count

- 설명: 예산을 설정할 카테고리들 중에서 하나를 선택하여 삭제하는 작업을 수행합니다. 이때 InputIntOnly함수로 정수 이외의 것이 입력되는 경우의 수를 차단합니다.

삭제 작업은 종료를 선택하기 전까지 반복되며, 삭제와 동시에 설정된 카테고리의 수도 변경됩니다. 단, 이미 예산이 할당되어 있는 카테고리는 삭제할 수 없습니다.

- 적용된 배운 내용: 함수, 구조체, 포인터, 반복문, 조건문

- 코드 스크린샷

```
// 카테고리 삭제 선택.
if (choice_category == 1) {

    if (category_count == 1) { // 카테고리 전부 삭제 차단
        printf("최소 한 개 이상의 카테고리가 존재해야 합니다. ");
        continue;
    }

    printf("삭제를 원하는 카테고리의 번호를 선택해주세요.(예산이 할당되지 않은 카테고리만 삭제할 수 있습니다)₩n");

    del_category = InputIntOnly();
    del_category = del_category-1; // 인덱스 벗어나는 거 방지.

    // 예산이 할당된 카테고리 제거 금지
    if (exp->budget[del_category] != 0) {
        printf("해당 카테고리에 할당된 예산이 존재합니다. ₩n");
        continue;
    }

    // 삭제 진행.
    for (int i = del_category - 1; i < category_count - 1; i++) {
        for (int j = 0; j < 10; j++) {
            category_names[i][j] = category_names[i + 1][j]; // 배열을 한 칸씩 당겨 삭제.
        }
    }
    category_count -= 1;

    printf("삭제가 완료되었습니다.₩n");
```

(2) 당월 예산 저장 기능/ 필수 지출 금액 저장 기능

- 입력: struct Expenditure *exp: 예산, 필수 지출 금액, 지출 발생 현황등 저장되어있는 구조체로, 이 기능에서는 budget(예산), essential_ex(필수 지출 금액)을 사용합니다.

category_names: 카테고리의 이름이 저장된 2차원 배열, category_count: 저장된 카테고리의 수가 저장된 정수.

출력: 예산이 할당된 budget과 essential_ex

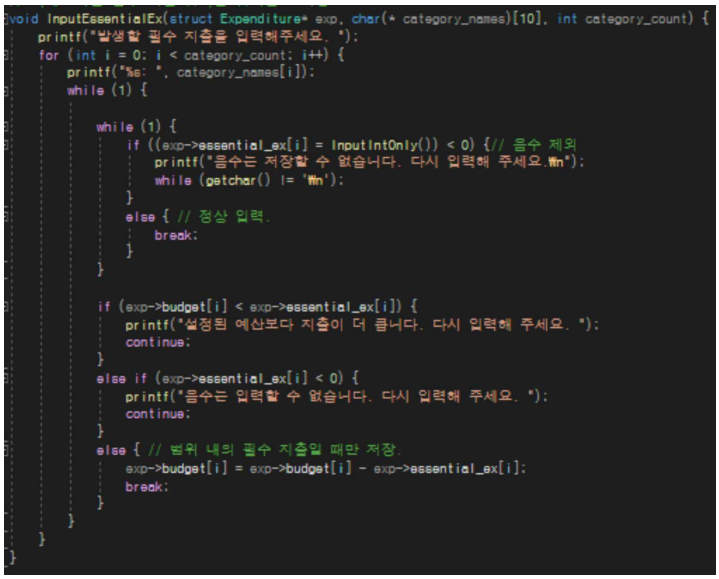
- 설명: 지난 달 예산 사용 내용을 바탕으로 저장된 파일을 열어 카테고리별로 예산 설정 추천 금액을 제공합니다. 추천 금액은 단순히 추천으로만 작용하며, 예산을 설정하는 것에 별도의 영향을 미치지는 않습니다. 예산 설정 추천 기능에서 파일 등에 대해 더 자세히 다룹니다.

1차로 1차원 배열 budget에 정수를 입력받아 할당합니다. 2차로 essential_ex에 발생하게 될 필수 지출을 입력받아 budget과 같은 순서로 저장하며(1차원 배열) 기존 budget에서 essential_ex를 각각 빼서 budget을 갱신합니다. 따라서 입력받는 essential_ex는 음수가 아니어야 하며, 같은 순서의 budget보다 작도록 합니다.

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

- 코드 스크린샷

```
void InputBudget(struct Expenditure* exp, char(*category_names)[10], int category_count) {

    FILE* file;
    fopen_s(&file, "expenditure.txt", "r");
    if (file == NULL) {
        printf("파일을 작성할 수 없습니다.\n");
        fclose(file);
    }

    int line_num = 1; // 짝수 줄만 출력하기 위한 index용.
    char text[100]; // 파일 내용 저장용


    for (int i = 0; i < category_count; i++) {
        while (fgets(text, sizeof(text), file) != NULL) {
            if (line_num % 2 == 0) { // 짝수 줄
                printf("%s", text);
                break; // 한 줄만 출력.
            }
            line_num++;
        }

        fseek(file, 0, SEEK_CUR); // 파일의 현재 위치로 이동
        line_num = 1;

        printf("%s: ", category_names[i]);


        while (1) {
            if ((exp->budget[i] = InputIntOnly()) < 0) {// 음수 제외
                printf("음수는 저장할 수 없습니다. 다시 입력해 주세요.\n");
            }
            else { // 정상 입력.
                break;
            }
        }

        printf("\n");
    }

    fclose(file);
```



```
void InputEssentialEx(struct Expenditure* exp, char(* category_names)[10], int category_count) {
    printf("발생할 필수 지출을 입력해주세요. ");
    for (int i = 0; i < category_count; i++) {
        printf("%s: ", category_names[i]);
        while (1) {

            while (1) {
                if ((exp->essential_ex[i] = InputIntOnly()) < 0) {// 음수 제외
                    printf("음수는 저장할 수 없습니다. 다시 입력해 주세요.₩n");
                    while (getchar() != '₩n');
                }
                else { // 정상 입력.
                    break;
                }
            }

            if (exp->budget[i] < exp->essential_ex[i]) {
                printf("설정된 예산보다 지출이 더 큽니다. 다시 입력해 주세요. ");
                continue;
            }
            else if (exp->essential_ex[i] < 0) {
                printf("음수는 입력할 수 없습니다. 다시 입력해 주세요. ");
                continue;
            }
            else { // 범위 내의 필수 지출일 때만 저장
                exp->budget[i] = exp->budget[i] - exp->essential_ex[i];
                break;
            }
        }
    }
}
```

(3) 소비 금액 계산 기능

- 입력: struct Expenditure *exp: 예산, 필수 지출 금액, 지출 발생 현황등 저장되어있는 구조체로, 이 기능에서는 total_expenditure(총지출)과 cost(변경용 예산), budget(초기 예산)을 사용합니다.

category_names: 카테고리의 이름이 저장된 2차원 배열, category_count: 저장된 카테고리의 수가 저장된 정수.

출력: 새로운 지출 내역이 반영된 cost, 카테고리별 지출 총액이 저장된 total_expenditure

- 설명: 사용자에게 지출이 발생한 카테고리를 입력받습니다. 이때 존재하지 않는 카테고리를 입력받고자 하면 다시 입력받고, 입력받은 카테고리의 위치를 동시에 저장해 예산 관련 연산을 위해 사용합니다.

지출 금액을 입력받아 total_expenditure(총지출)의 입력받은 카테고리의 위치에 더해 해당 카테고리의 누적 총지출을 저장하고 , cost의 카테고리 위치에서 지출을 마이너스하여 해당 카테고리에서 남은 예산을 저장합니다.

마지막으로 해당 카테고리 금액/ 총 예산을 시각적으로 제시합니다.

- 적용된 배운 내용: 함수, 구조체, 포인터, 조건문, 반복문

- 코드 스크린샷

```c
void SaveExpenditure(struct Expenditure* exp, char(*category_names)[10], int category_count) {

    char input_cate[10]; // 카테고리 최대 글자수 10
    int valid_cate = 0; // 카테고리 존재 확인 용도
    int input_index = 0; // 입력받은 카테고리 존재 위치 저장
    int input_expenditure = 0; //임시적 지출 저장.


    printf("지출이 발생한 카테고리를 입력해 주세요. ");
    while (valid_cate != 1) {
        scanf_s("%s", &input_cate, (int)sizeof(input_cate));
        for (int i = 0; i < category_count; i++) {
            if (strcmp(input_cate, category_names[i]) == 0) { // strcmp -> 0 일치
                input_index = i; // 카테고리 위치 저장.
                valid_cate = 1;
                break;
            }
        }
        if (valid_cate == 0) { // 없는 카테고리 입력받는 경우 제외
            printf("존재하지 않는 카테고리입니다. 다시 입력해 주세요.");
            continue;
        }
    }



    printf("발생한 지출 금액을 입력해주세요.");
    while (1) {
        if ((input_expenditure = InputIntOnly()) < 0) {// 음수 제외
            printf("음수는 저장할 수 없습니다. 다시 입력해 주세요.₩n");
        }
        else { // 정상 입력.
            break;
        }
    }

    exp->total_expenditure[input_index] += input_expenditure; // 카테고리별 지출 총액 저장
    exp->cost[input_index] -= input_expenditure; // 예산에서 마이너스

    // 예산 초과 경고만, 음수 연산 허용.
    if (exp->total_expenditure[input_index] > exp->budget[input_index]) {
        printf("설정 예산을 초과했습니다. ₩n");
    }


    // 각 예산별 사용 금액과 총 소비액 제시.
    // 해당 카테고리 사용 금액 / 총 예산
    printf("해당 카테고리에서의 총 사용 금액:%d / 예산: %d₩n", exp->total_expenditure[input_index], exp->budget[input_index]);
    printf("%d원 남았습니다. ", exp->cost[input_index]);

}
```

(4) 달성도 평가 기능

- 입력: struct Expenditure *exp: 예산, 필수 지출 금액, 지출 발생 현황 등 저장되어있는 구조체로, 이 기능에서는 save_budget(예산 - 총지출), budget(예산), total_expenditure(총지출), essential_ex(필수 지출 금액), save_percentage(save_budget/budget * 100, 달성도 퍼센트) 를 사용합니다.

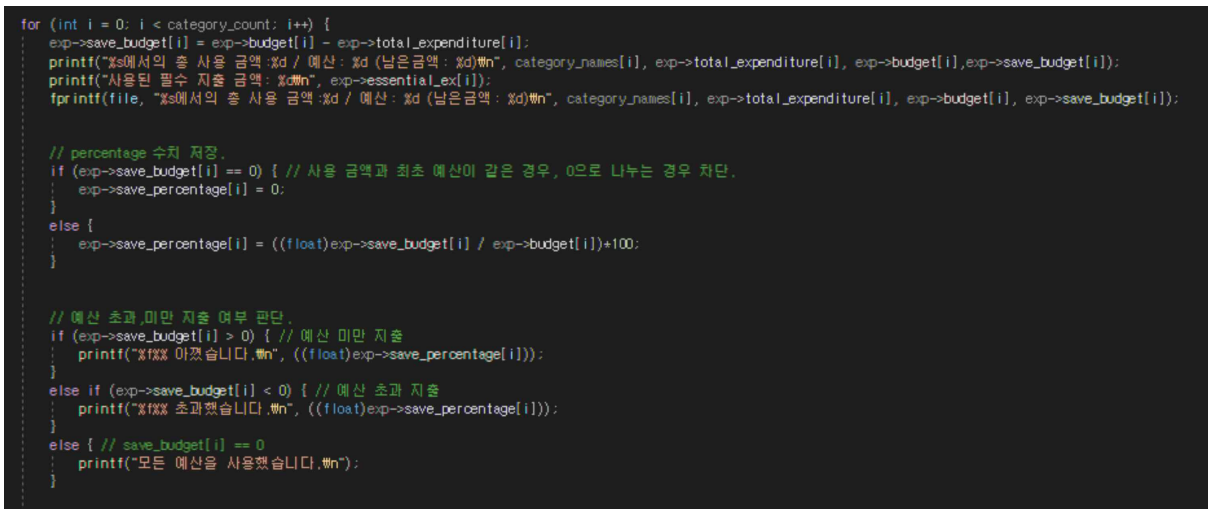
category_names: 카테고리의 이름이 저장된 2차원 배열, category_count: 저장된 카테고리의 수가 저장된 정수.

출력: 각 카테고리별 예산 - 총지출 연산이 수행된 save_budget과 카테고리별 달성도를 저장한 exp-> save_percentage.

- 설명: 카테고리의 개수만큼 순회하며, 각 카테고리별로 예산에서 총지출을 제한 금액을 각 save_budget에 저장하며, 이를 기반으로 초기 budget 대비 아낀 퍼센트 수치를 save_percentage에 저장하고, 초기 예산 대비 아끼거나 초과한 수치를 사용자에게 제공합니다.

- 적용된 배운 내용: 구조체, 포인터, 조건문, 반복문

- 코드 스크린샷



```
for (int i = 0; i < category_count; i++) {
    exp->save_budget[i] = exp->budget[i] - exp->total_expenditure[i];
    printf("%s에서의 총 사용 금액 :%d / 예산 : %d (남은금액 : %d)₩n", category_names[i], exp->total_expenditure[i], exp->budget[i],exp->save_budget[i]);
    printf("사용된 필수 지출 금액 : %d₩n", exp->essential_ex[i]);
    fprintf(file, "%s에서의 총 사용 금액 :%d / 예산 : %d (남은금액 : %d)₩n", category_names[i], exp->total_expenditure[i], exp->budget[i], exp->save_budget[i]);


    // percentage 수치 저장.
    if (exp->save_budget[i] == 0) { // 사용 금액과 최초 예산이 같은 경우, 0으로 나누는 경우 차단.
        exp->save_percentage[i] = 0;
    }
    else {
        exp->save_percentage[i] = ((float)exp->save_budget[i] / exp->budget[i])*100;
    }


    // 예산 초과,미만 지출 여부 판단.
    if (exp->save_budget[i] > 0) { // 예산 미만 지출
        printf("%f%% 아꼈습니다.₩n", ((float)exp->save_percentage[i]));
    }
    else if (exp->save_budget[i] < 0) { // 예산 초과 지출
        printf("%f%% 초과했습니다.₩n", ((float)exp->save_percentage[i]));
    }
    else { // save_budget[i] == 0
        printf("모든 예산을 사용했습니다.₩n");
    }
```

(5) 예산 설정 추천 기능.

- 입력: struct Expenditure *exp: 예산, 필수 지출 금액, 지출 발생 현황 등 저장되어있는 구조체로, 이 기능에서는 save_percentage(초기 budget 대비 아낀 퍼센트 수치), budget(예산)을 사용합니다.

category_names: 카테고리의 이름이 저장된 2차원 배열, category_count: 저장된 카테고리의 수가 저장된 정수.

- 설명: 최종적 지출과 관련 내용을 저장할 문서 expenditure.txt를 작성합니다. 아낀 수치를 -150,-100,0,30,60,100등 구간으로 분리하여 save_percentage가 해당되는 퍼센트에 따라 차후에 추천할 예산 금액을 조정합니다.

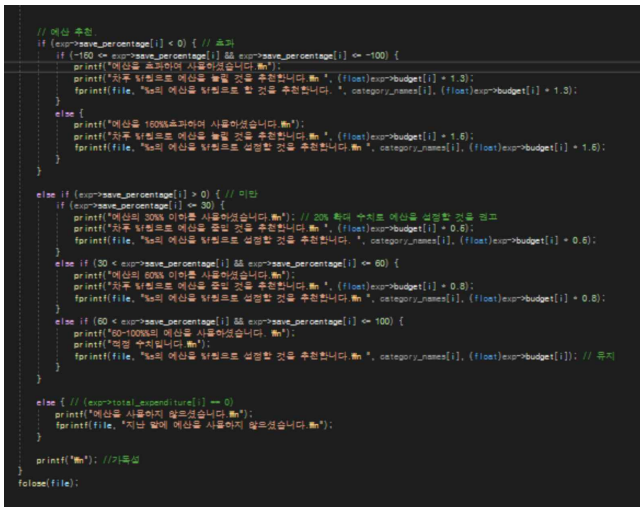
1차적으로 종료할 때 차후에 설정하기에 적절한 예산 수치를 제공하고, 동시에 이를 파

일에 작성함으로 다음 예산 설정 시 카테고리별 추천 금액 관련 내용이 출력될 수 있도록 합니다. 이때 홀수 줄은 총 사용 금액과 예산 등 단순 정보 줄이므로, 추천 내용이 들어있는 짝수 줄만 출력될 수 있도록 합니다.

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

- 코드 스크린샷

```
// 달성도 평가(지출 분석 기능)
void Evaludation(struct Expenditure* exp, char(*category_names)[10], int category_count) {

    FILE* file;
    fopen_s(&file,"expenditure.txt", "w"); // 1회차 내용만 저장.
    if (file == NULL) {
        printf("파일을 작성할 수 없습니다.₩n");
        fclose(file);
    }
```



```
    // 예산 추천
    if (exp->save_percentage[i] < 0) { // 초과
        if (-160 <= exp->save_percentage[i] && exp->save_percentage[i] <= -100) {
            printf("예산을 초과하여 사용하셨습니다.₩n");
            printf("차후 5퍼센트로 예산을 늘릴 것을 추천합니다.₩n ", (float)exp->budget[i] * 1.3);
            fprintf(file, "%s의 예산을 5퍼센트로 할 것을 추천합니다. ", category_names[i], (float)exp->budget[i] * 1.3);
        }
        else {
            printf("예산을 160%초과하여 사용하셨습니다.₩n");
            printf("차후 5퍼센트로 예산을 늘릴 것을 추천합니다.₩n ", (float)exp->budget[i] * 1.6);
            fprintf(file, "%s의 예산을 5퍼센트로 설정할 것을 추천합니다.₩n ", category_names[i], (float)exp->budget[i] * 1.6);
        }
    }

    else if (exp->save_percentage[i] > 0) { // 이하
        if (exp->save_percentage[i] <= 30) {
            printf("예산의 30% 이하를 사용하셨습니다.₩n"); // 20% 확대 수치로 예산을 설정할 것을 권고
            printf("차후 5퍼센트로 예산을 줄일 것을 추천합니다.₩n ", (float)exp->budget[i] * 0.6);
            fprintf(file, "%s의 예산을 5퍼센트로 설정할 것을 추천합니다. ", category_names[i], (float)exp->budget[i] * 0.6);
        }
        else if (30 < exp->save_percentage[i] && exp->save_percentage[i] <= 60) {
            printf("예산의 60% 이하를 사용하셨습니다.₩n");
            printf("차후 5퍼센트로 예산을 줄일 것을 추천합니다.₩n ", (float)exp->budget[i] * 0.8);
            fprintf(file, "%s의 예산을 5퍼센트로 설정할 것을 추천합니다.₩n ", category_names[i], (float)exp->budget[i] * 0.8);
        }
        else if (60 < exp->save_percentage[i] && exp->save_percentage[i] <= 100) {
            printf("60-100%의 예산을 사용하셨습니다. ₩n");
            printf("적절 수치입니다.₩n");
            fprintf(file, "%s의 예산을 5퍼센트로 설정할 것을 추천합니다.₩n ", category_names[i], (float)exp->budget[i]); // 유지
        }
    }

    else { // (exp->total_expenditure[i] == 0)
        printf("예산을 사용하지 않으셨습니다.₩n");
        fprintf(file, "지난 달에 예산을 사용하지 않으셨습니다.₩n");
    }

    printf("₩n"); //가독성
}
fclose(file);
```

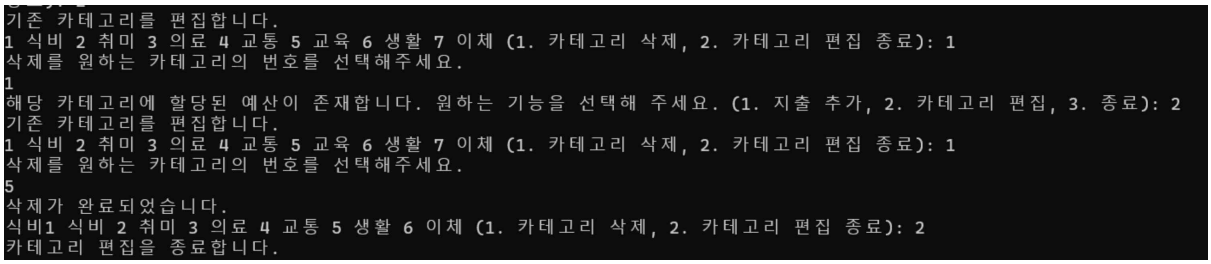
```
void InputBudget(struct Expenditure* exp, char(*category_names)[10], int category_count) {

    FILE* file;
    fopen_s(&file, "expenditure.txt", "r");
    if (file == NULL) {
        printf("파일을 작성할 수 없습니다.₩n");
        fclose(file);
    }

    int line_num = 1; // 짝수 줄만 출력하기 위한 index용.
    char text[100]; // 파일 내용 저장용


    for (int i = 0; i < category_count; i++) {
        while (fgets(text, sizeof(text), file) != NULL) {
            if (line_num % 2 == 0) { // 짝수 줄
                printf("%s", text);
                break; // 한 줄만 출력.
            }
            line_num++;
        }

        fseek(file, 0, SEEK_CUR); // 파일의 현재 위치로 이동
        line_num = 1;
```

4. 테스트 결과

(1) 카테고리 편집 기능

- 설명: 저장된 예산이 없는 카테고리만 삭제되는지, 효과적으로 삭제되는지 확인하기 위해 테스트합니다.

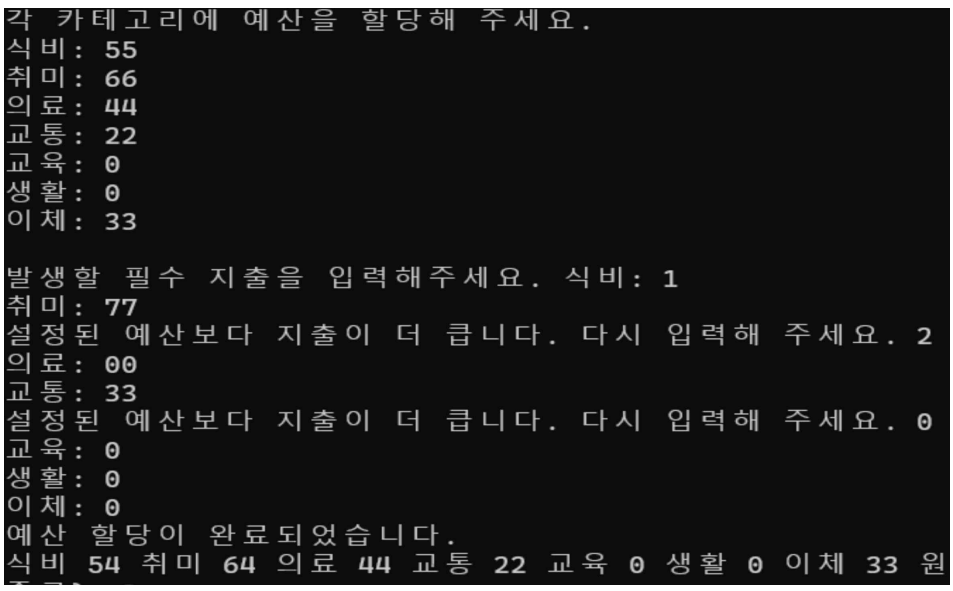
- 테스트 결과 스크린샷



(2) 당월 예산 저장 기능

- 설명: budget과 essential_ex가 잘 저장되었는지, budget에 essential_ex를 제외한 금액이 저장되었는지 확인하기 위해 테스트합니다.

- 테스트 결과 스크린샷

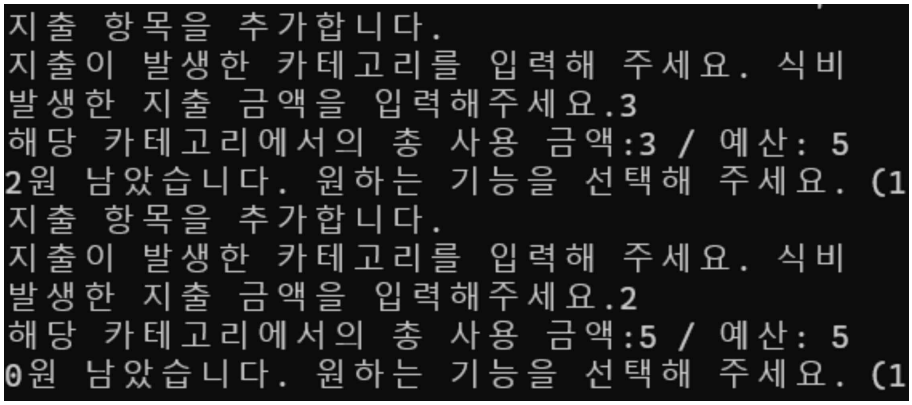


(3) 소비 금액 계산 기능

- 설명: 각 예산별 사용 금액과 총 소비액이 성공적으로 저장되었는지, 남은 예산이 성공

적으로 반영되었는지 확인하기 위해 테스트합니다.

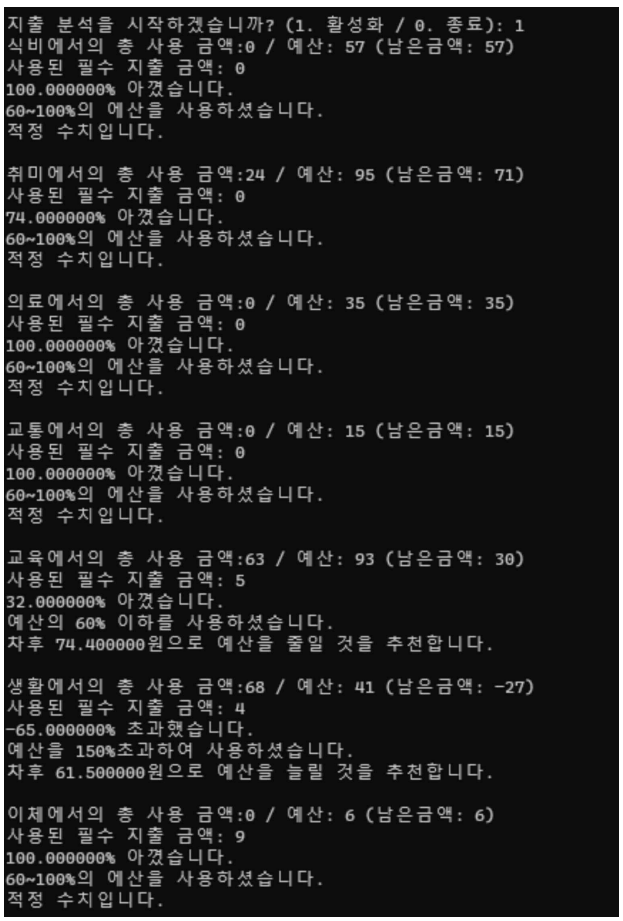
- 테스트 결과 스크린샷



(4) 달성도 평가 기능

- 설명: 초기 예산 대비 사용 금액이 잘 출력되는지, save_percentage가 잘 저장되었는지, 달성도를 평가하는데 효과적으로 적용되고 있는지 확인하기 위해 테스트합니다.

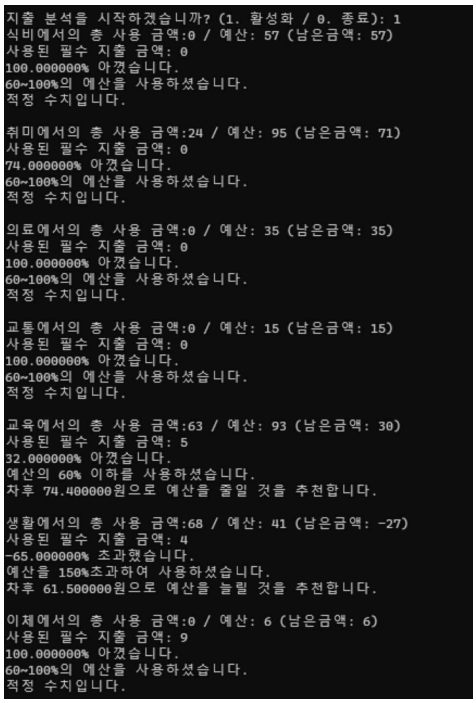
- 테스트 결과 스크린샷

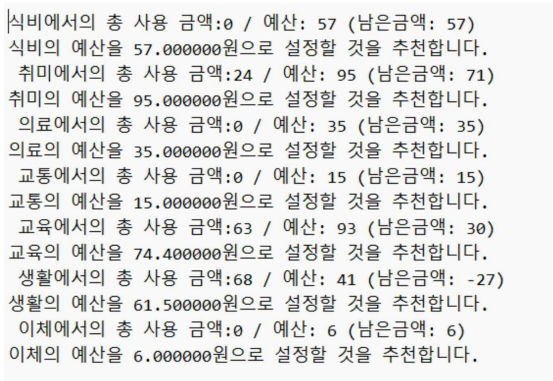


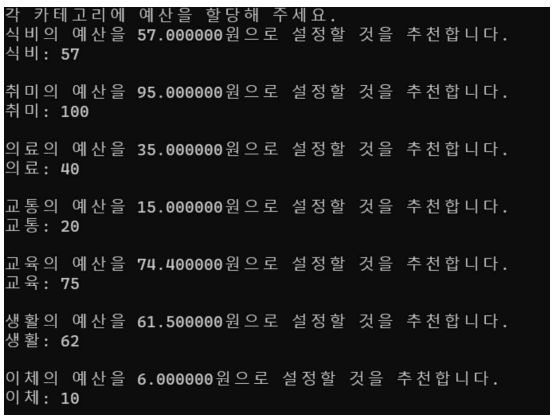
(5) 예산 설정 추천 기능.

- 설명: 최종 추천 사항이 효과적으로 출력되는지 확인하고, 이 사항이 텍스트 문서에 효과적으로 저장되고 츼 예산 입력 시에 적용되는지 확인하기 위해 테스트합니다.

- 테스트 결과 스크린샷







5. 계획 대비 변경 사항

변경사항 없음.

6. 느낀점

프로젝트를 진행하다 보니 평소에는 넘어갈 수업 내용도 사용할 수 있을지 하나하나 돌아보게 되고 수업 외 새로운 내용도 찾아보게 되어 여러모로 코딩에 대해 많이 배울 수 있었던 것 같습니다. 이번 수업이 아니었다면 코딩을 배운지 얼마 되지 않았다는 이유로 코딩으로 작은 것이라도 프로젝트를 진행할 생각을 하지 않았을 것 같은데 덕분에 다양한 경험을 할 수 있었고 앞으로도 개인적으로 구현해 보고 싶은 걸 구현해 보는 도전을 할 수 있을 것 같습니다.

교수님 수업이 개인적으로 상당히 어려웠지만 교수님께서 항상 질문을 받으려고 하시고, 다음 시간에 질문이 많이 들어왔던 내용에 대해 추가 설명도 꼼꼼히 해주시는 등 저희에게 많은 내용을 가르치고자 하는 것이 느껴져서 너무 감사했습니다. 특히 마지막 날 저희가 앞으로 가져야 할 역량을 말해주실 때 너무 인상 깊었습니다. 덕분에 코딩에 대해서 많은 것을 배울 수 있었고, 또한 코딩을 대하는 태도에 대해서도 조금 알 수 있어서 너무 좋았습니다.

교수님 수업을 두 개나 듣다 보니 다른 수업들과 프로젝트 두 개를 동시에 진행하게 되어 두 프로젝트 모두에 아쉬움이 남은 것 같습니다. 시간이 너무 부족하기도 했고, 두 프로젝트 모두의 커밋을 신경 써야 한다는 점이 상당히 압박으로 다가와 성과와 별개로 조금 많이 힘들기도 했습니다. 교수님 수업을 하나만 들었다면 하나의 프로젝트에 온전히 집중하여 그 과목에 대해 애정을 가지고 더 깊은 이해를 할 수 있었을 것 같다는 개인적인 아쉬움이 남았습니다. 그러나 프로젝트 방식은 너무 좋았고, 많은 걸 배울 수 있어서 좋았습니다.