C프로그래밍 및 실습

24시간이 충분한 스케줄 플래너

최종 보고서

제출일자:2023.12.24
제출자명:최세인
제출자학번:231344

1. 프로젝트 목표

1) 배경 및 필요성

많은 사람들이 하루를 효율적으로 보내기를 희망하지만 계획 세우는 일에 능하지 않거나 무리한 계획을 세우는 경우가 다반사이다. 또한 아예 계획을 세우지 않고 즉흥적으로 일을 처리하는 사람들도 존재한다. 시간을 효율적으로 쓰지 못하는 현대인들은 24시간이 부족하다고 느끼며 이러한 악습관은 바쁜 일상 속에서 쉽게 피로와 무기력함을 느끼게 한다. 이 문제를 해결하기 위해 자동으로 일정을 짜주는 사용자 맞춤형 스케줄 플래너가 필요하다.

2) 프로젝트 목표

주어진 24시간을 적절하게 자동 분배해 사용자가 입력한 할 일을 가지고 합리적인 플래너를 작성해주는 프로그램을 제작한다. 이 프로그램은 해야 할 일을 효율적으로 수행 한 후, 남는 시간을 활용할 수 있도록 도와 삶의 균형을 맞추고 여가 시간 확보, 심리적 불안 감소, 삶의 효율 증진을 목표로 한다.

3) 차별점

기존의 스케줄 프로그램은 단순히 사용자가 입력한 할 일을 저장하여 잊어버리지 않게 끔 환기해 주는 역할에 그쳤으며 시간을 일일이 계산해서 스케줄표를 작성하는 번거로움이 있다. 이번 프로젝트에서 시행될 프로그램은 예상 소요시간, 일의 난이도 등을 고려하여 최적의 시간표와 일정을 제공하는 차별화된 서비스가 특징이다.

2. 기능 계획

1) 기능 1 : 모드 설정

- 직장, 공부, 휴일, 여행의 4가지 모드를 제공하여 스케줄의 강도를 결정한다.

(1) 세부 기능 1 : 집중 모드

- 직장, 공부 모드는 집중 모드로 분류하여 정해진 시간 내에 집중하여 일의 효율을 높일 수 있도록 예상 소요시간에 가중치를 부여한다.

(2) 세부 기능 2 : 여유 모드

휴일, 여행 모드는 여유 모드로 분류하여 시간에 쫓기지 않고 자유로움을 최대한 즐길 수 있도록 예상 소요시간에 가중치를 부여한다.

2) 기능 2 : 스케줄 시작 시간 설정 및 할 일 입력 받기

- 하루 중 스케줄을 시작하고자 하는 시간을 입력 받아 그 시간을 기준으로 삼는다.

- 할 일을 하나씩 입력 받는다.

3) 기능 3 : 세부 고려사항

- 일을 처리하는 데 걸리는 예상 소요시간과 일의 난이도를 입력 받는다.

(1) 세부 기능 1 : 난이도

- 1~5점으로 평가한다.

(2) 세부 기능 2 : 시간 분배

- 예상 소요시간과 난이도의 차를 구하고 범위에 따라 집중 모드의 경우 +-0.5~1의 가중치를, 여유 모드의 경우 +0.5~1의 가중치를 부여한다. 가중치를 부여할 때, 집중 모드의 경우 예상소요시간과 난이도를 고려해 소요시간을 줄이거나 늘리지만 여유모드는 줄이지 않고 늘리기만 할 수 있도록 설정했다. 가중치가 부여된 예상 소요시간을 가지고 플래너를 작성한다.

3. 기능 구현

(1) 모드 설정

- 입력 값: 모드 번호/ 출력 값: 선택한 모드

- 1~4번까지 4가지 모드 중에 하나를 선택한다.

  잘못 입력 시 경고 메시지 출력 후 올바르게 입력 할 때까지 코드를 반복한다.

- 적용된 배운 내용: 반복문(while/ do-wile), 구조체, 배열

-코드 스크린샷

```
int main(){
    while (1) {
        printf("24시간이 충분한 스케줄 플래너₩n");
        printf("----------------------------₩n");

        /*모드 설정 코드 블록*/

        Modeset set = { 0, 0 };
        char mode[4][10] = { "직장", "공부", "휴일", "여행" };  //4가지 모드 배열에 저장
        printf("모드를 설정합니다.₩n");
        printf("<집중 모드>₩n");
        printf("1. 직장₩n");
        printf("2. 공부₩n");
        printf("<여유 모드>₩n");
        printf("3. 휴일₩n");
        printf("4. 여행₩n");
        printf("----------------------------₩n");
        printf("원하는 모드의 번호를 입력하세요: ");
        do {                                          //범위에 벗어난 모드 번호 입력시 경고 메시지 출력
            scanf_s("%d", &(set.modenum));
            if (set.modenum < 1 || set.modenum>4) {
                printf("1에서 4 사이의 번호를 입력하세요.");
            }
        } while (set.modenum < 1 || set.modenum>4);
        printf("%s모드", mode[set.modenum - 1]);
        printf("₩n----------------------------₩n");
```

(2) 스케줄 시작 시간 설정

- 입출력 값: 스케줄 시작 시간

- 스케줄 시작을 원하는 시간을 입력받는다.

-코드 스크린샷

```
/*스케줄 시작 시간 설정 코드 블록*/
double start;
printf("스케줄 시작 시간을 입력하세요(00시~24시): ");
scanf_s("%lf", &start);
printf("%.1lf시", start);
printf("₩n----------------------------₩n");
```

(3) 할 일 입력 받기

- 입력 값: 할 일

- 헤더 파일을 만들어 함수를 정의하였다.

-문자열을 입력 받은 후 할당한 동적 메모리에 값을 복사하여 저장한다. end를 입력하면 입력 받기를 멈춘다.

- 적용된 배운 내용: 입력 함수(getchar)로 버퍼 비우기, 구조체, 헤더파일, 포인터, 반복문, 입력 함수(fgets), 문자열 비교/길이 함수, 조건문, 동적 메모리, 문자열 복사함수

-코드 스크린샷

```
int ch;
while ((ch = getchar()) != '₩n');  //버퍼 비우기

/*할 일 입력 받는 함수 호출*/

Todo s1 = { "","",0 };  //구조체 초기화
todo(&s1);
printf("------------------------------₩n");
```

```
#ifndef _TODO_H_  // todo.h 헤더파일 중복 방지
#define _TODO_H_

#define MAX_TASKS 100

typedef struct {
    char tasks[MAX_TASKS];
    char* str[MAX_TASKS];
    int count;
}Todo;            //할 일 입력 받는 구조체 선언

/*할 일 입력받는 함수*/

void todo(Todo* ps);

#endif
```

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

/*할 일 입력받는 함수*/

void todo(Todo* ps) {
    int i;
    for (i = 0; i < 30; i++) {

        printf("할 일을 하나씩 입력하세요(end를 입력하면 종료): ");
        fgets(ps->tasks, sizeof(ps->tasks), stdin);
        ps->tasks[strlen(ps->tasks) - 1] = '\0';      //개행문자 제거
        if (strcmp(ps->tasks, "end") == 0) {      //end를 입력하면 할 일 입력 받기 중지
            break;
        }
        ps->str[i] = (char*)malloc(strlen(ps->tasks) + 1);  //할 일을 저장할 메모리 동적할당
        if (ps->str[i] == NULL) {                    //동적 할당에 실패할 경우
            printf("메모리가 부족합니다.\n");
            exit(1);                              // 프로그램 종료
        }
        strcpy_s(ps->str[i], strlen(ps->tasks) + 1, ps->tasks);     //동적 메모리에 할 일 복사
        ps->count++;   //할 일 개수 세기

    }
}
```

(4) 세부 고려사항- 예상 소요시간과 난이도 입력 받기

- 입력 값: 사용자가 예상하는 소요시간과 난이도

- 헤더 파일을 만들어 함수를 정의하였다.

- 집중모드와 여유모드로 나눠 시간 분배에 필요한 예상 소요시간과 난이도를 입력 받는다.

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

-코드 스크린샷

```c
*예상 소요시간과 난이도를 입력 받는 함수 + 시간계산 함수 호출*

rint_time time;


/집중모드
f ((set.modenum == 1) || (set.modenum == 2)) {
   calculate1(&s1, &set, &time);


/여유모드
f ((set.modenum == 3) || (set.modenum == 4)) {
   calculate2(&s1, &set, &time);
```

```c
#include "todo.h"

typedef struct {
    int modenum;
    double start;
}Modeset;          //모드와 시작 시간 구조체 선언

typedef struct {
    double* end[MAX_TASKS];      //할 일 끝나는 시각 변수
    double *setting_time[MAX_TASKS];    //할 일 시작하는 시각 변수
}Print_time;     //스케줄별 시간 구조체 선언

void calculate1(Todo* ps, Modeset* pset, Print_time* ptime);
void calculate2(Todo* ps, Modeset* pset, Print_time* ptime);
```

```c
 /*집중 모드*/
void calculate1(Todo* ps, Modeset* pset, Print_time* ptime) {
    int i;
    double req_time=-1;  //예상 소요시간 변수
    int level;       //난이도 변수
    double gap;

    for (i = 0; i < ps->count; i++) {
        printf("%s의 예상 소요시간을 입력하세요: ", ps->str[i]);
        scanf_s("%lf", &req_time);

        printf("%s의 난이도를 입력하세요(1~5): ", ps->str[i]);
        scanf_s("%d", &level);
```

(5) 세부 고려사항- 플래너에 부여할 스케줄 소요시간 계산하기

- 출력 값: 일의 시작 시각 및 종료 시각

- 헤더 파일에 함수를 정의하였다.

- 집중 모드는 차이 값의 범위에 따라 예상 소요시간에 +- 0.5~1의 가중치를 부여해주어 플래너에 작성할 시간을 최종 결정한다. 여유 모드는 차이 값의 범위에 따라 최대 1시간을 추가적으로 부여하여 자유를 즐길 수 있도록 시간을 최종 결정한다.

- 할 일이 끝나는 시간을 다른 할 일의 시작 시간으로 지정한다.

- 스케줄이 하루 (24시)를 넘어갈 경우 여유시간 확보를 권고하는 메시지 출력

- 적용된 배운 내용: 구조체, 헤더 파일, 포인터, 조건문, 반복문, 동적 메모리

-코드 스크린샷

```c
/*예상 소요시간과 난이도를 입력 받는 함수 + 시간계산 함수 호출*/

Print_time time;


//집중모드
if ((set.modenum == 1) || (set.modenum == 2)) {
    calculate1(&s1, &set, &time);

}
//여유모드
if ((set.modenum == 3) || (set.modenum == 4)) {
    calculate2(&s1, &set, &time);

}
```

```c
#include "todo.h"

typedef struct {
    int modenum;
    double start;
}Modeset;          //모드와 시작 시간 구조체 선언

typedef struct {
    double* end[100];  //할 일 끝나는 시각 변수
    double* setting_time[100];    //할 일 시작하는 시각 변수
}Print_time;     //스케줄별 시간 구조체 선언

void calculate1(Todo* ps, Modeset* pset, Print_time* ptime);
void calculate2(Todo* ps, Modeset* pset, Print_time* ptime);
```

```
    /*예상 소요시간에 비해 난이도가 어려운 경우*/
    if (gap < -2) {
        req_time += 1;                //예상 소요시간에 +1 가중치 부여
    }
    else if (gap < -1) {
        req_time += 0.5;              //예상 소요시간에 +0.5 가중치 부여
    }
    else if (-1 <= gap && gap <= 1) {
        req_time += 0;                // 예상 소요시간 그대로 유지
    }
    /*예상 소요시간에 비해 난이도가 쉬운 경우*/
    else if (1 < gap && gap <= 2) {
        req_time -= 0.5;              //예상 소요시간에 -0.5 가중치 부여
    }
    else if (gap > 2) {
        req_time -= 1;                //예상 소요시간에 -1 가중치 부여
    }
    ptime->setting_time[i] = (double*)malloc(sizeof(double));  //할 일 시작하는 시간 저장하는 동적 메모리 할당
    if (ptime->setting_time[i] == NULL) {
        exit(1);
    }
    ptime->end[i] = (double*)malloc(sizeof(double));          //할 일 끝나는 시간 저장하는 동적 메모리 할당
    if (ptime->end[i] == NULL) {
        exit(1);
    }
    if (i == 0) {
        *(ptime->setting_time[i]) = pset->start;
    }
    else {
        *(ptime->setting_time[i]) = *(ptime->end[i - 1]);
    }
    *(ptime->end[i]) = *(ptime->setting_time[i]) + req_time;       //할 일이 끝나는 시간 = 다음 할 일 시작 시간

}
if (**ptime->end + ps->count-1 > 24) {
    printf("₩n*할 일을 줄이고 여유시간을 확보해 보세요*₩n");  //스케줄 시간이 하루(24시)를 넘어갈 시 삶의 밸런스를 위해 권고 메시지 출력
}
```

```
    /*예상 소요시간에 비해 난이도가 어려운 경우*/
    if (gap <= -1) {
        req_time += 1;
    }
    else if (-1 < gap < 0) {
        req_time += 0;
    }
    /*예상 소요시간에 비해 난이도가 쉬운 경우*/
    else if (gap >= 1) {
        req_time += 0.5;       // 여유모드이므로 추가시간 있음
    }
    ptime->setting_time[i] = (double*)malloc(sizeof(double));
    if (ptime->setting_time[i] == NULL) {
        exit(1);
    }
    ptime->end[i] = (double*)malloc(sizeof(double));
    if (ptime->end[i] == NULL) {
        exit(1);
    }
    if (i == 0) {
        *(ptime->setting_time[i]) = pset->start;
    }
    else {
        *(ptime->setting_time[i]) = *(ptime->end[i - 1]);
    }
    *(ptime->end[i]) = *(ptime->setting_time[i]) + req_time;
}
if (**ptime->end + ps->count - 1 > 24) {
    printf("₩n*할 일을 줄이고 여유시간을 확보해 보세요*₩n");  //스케줄 시간이 하루(24시)를 넘어갈 시 삶의 밸런스를 위해 권고 메시지 출력
}
```

(6) 플래너 출력

출력 값: 완성된 플래너

- 계산한 시간을 가지고 플래너를 출력한다.

- 00시~24시로 시간 표현 ex) 26시->2시

- 적용된 배운 내용: 구조체, 포인터, 조건문, 반복문, 동적 메모리

-코드 스크린샷

```
/*스케줄 플래너 작성*/

printf("----------------------------₩n");
printf("플래너 작성 완료!₩n");


for (int i = 0; i < s1.count; i++) {
    /*00시~24시로 시간 표현*/
    if (*time.setting_time[i] > 24) {
        *time.setting_time[i] = *time.setting_time[i] - 24;
    }
    if (*time.end[i] > 24) {
        *time.end[i] = *time.end[i] - 24;

    }
    printf("%.1f시-%.1f시: %s₩n", *time.setting_time[i], *time.end[i], s1.str[i]);
}
```

(7) 추가 메뉴

- 입력 값: 메뉴 번호 / 출력 값: 코드 반복 또는 프로그램종료

- 새로운 플래너를 만드는 추가하기 항목과 프로그램을 종료하는 항목 중에 선택한다.

- 적용된 배운 내용: 조건문

-코드 스크린샷

```
/*스케줄 플래너 작성 후 메뉴 선택 코드 블록*/

int menu_num;
printf("₩n----------------------------₩n");
printf("1. 플래너 추가하기 ");
printf("2. 종료하기 ");
scanf_s("%d", &menu_num);
if (menu_num == 2) {
    break;      //스케줄 작성 종료
}
printf("₩n");
```

(8) 동적 메모리 반환

- 동적으로 할당한 저장 공간을 free함수를 사용하여 반환

- 적용된 배운 내용: 반복문, 동적 메모리

-코드 스크린샷

```
/*동적 메모리 반환*/

for (int i = 0; i < s1.count; i++) {
    free(s1.str[i]);
    free(time.setting_time[i]);
    free(time.end[i]);
}
```

4. 테스트 결과

(1) 모드 설정

- 원하는 모드의 번호를 입력 받고 모드 이름을 출력한다.

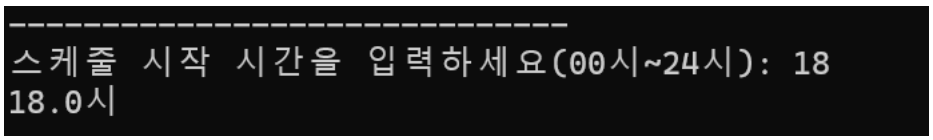
- 테스트 결과 스크린샷

```
24시간이 충분한 스케줄 플래너
------------------------------
모드를 설정합니다.
<집중 모드>
1. 직장
2. 공부
<여유 모드>
3. 휴일
4. 여행
------------------------------
원하는 모드의 번호를 입력하세요: 1
직장모드
```

(2) 스케줄 시작 시간 설정

- 스케줄 시작 시간을 입력 받고 출력한다.

- 테스트 결과 스크린샷



(3) 할 일 입력 받기

- 할 일을 하나씩 입력 받고 더 이상 없으면 end를 입력한다.

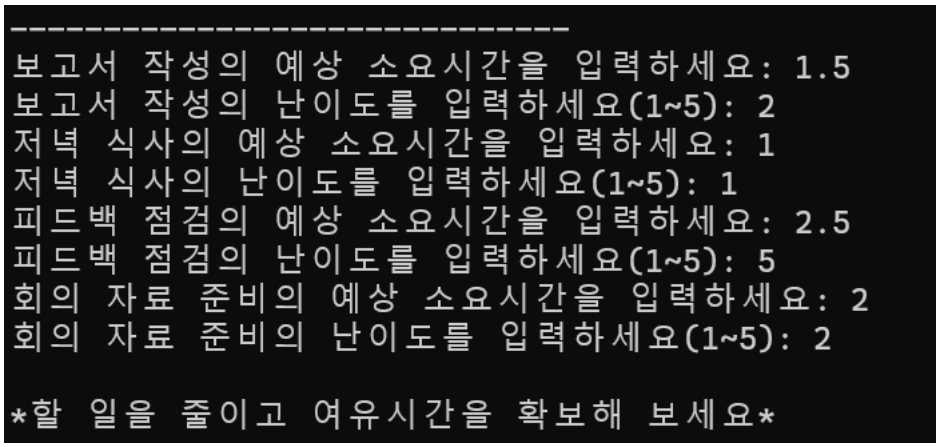
- 테스트 결과 스크린샷

(4) 세부 고려사항- 예상 소요시간과 난이도 입력 받기

- 예상되는 소요시간과 그 일의 난이도를 입력 받는다.

- 플래너 출력 전, 일이 끝나는 시간이 하루(24시)를 넘긴다면 여유 시간 확보를 권고하는 메시지를 출력한다.

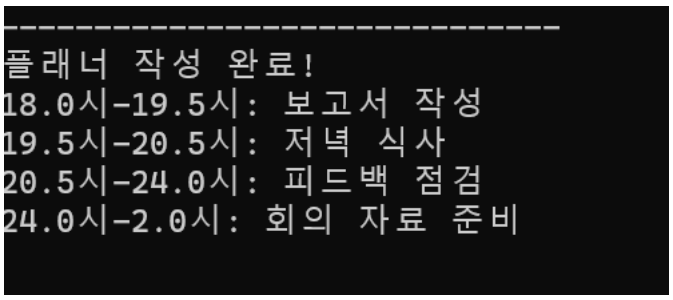
- 테스트 결과 스크린샷



(5) 플래너 출력

- 계산된 소요시간을 가지고 플래너를 출력한다..

- 테스트 결과 스크린샷

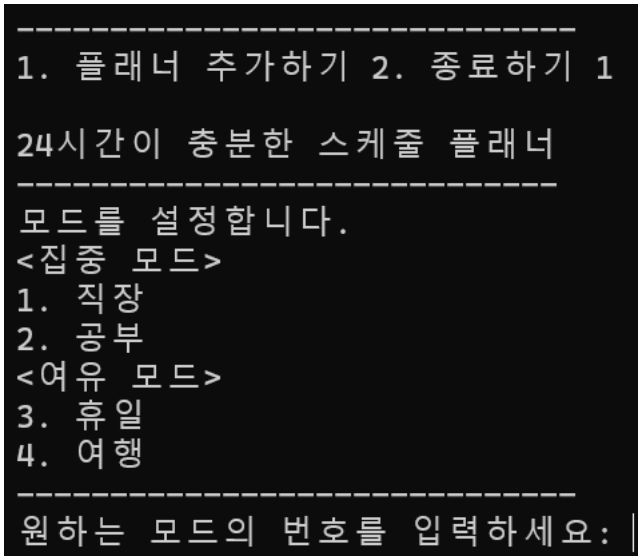


(6) 추가 메뉴

- 플래너 추가하기와 종료하기 중 선택한다.

- 플래너 추가 시 처음부터 코드를 반복한다.

- 테스트 결과 스크린샷



5. 계획 대비 변경 사항

1) 24시간 중 제외할 시간 설정하기

- 이전: 기상 시간, 취침 시간, 약속 시간 등 스케줄 조정에서 제외하고자 하는 시간을 따로 설정하려 하였다.
- 이후: 스케줄 시작 시간을 설정하는 것으로 변경하였다.
- 사유: 스케줄 전인 기상 시간과 스케줄 후인 취침 시간은 스케줄 조정 대상이 아니라고 판단했으며 약속 시간은 스케줄에 포함해야 한다고 생각해 제외대상으로 분류하지 않기로 결정하였다.

2) 이동거리 계산

- 이전: 스케줄 사이에 이동이 필요하다면 장소 간 이동거리를 고려해 스케줄을 작성하려 하였다.
- 이후: 기능 삭제-> 이동에 걸리는 시간을 포함한 예상 소요시간 입력 받기로 변경하였다.
- 사유: 이동 거리가 예상 시간과 난이도에 따라 계산되는 것이 적절하지 않다고 판단하였다.

6. 느낀점

수업시간에 todo리스트를 작성했을 때 추가적으로 구현해보고 싶었던 기능들이 있었다. 나에겐 이 프로젝트가 나만의 차별화된 기능을 추가하여 또 다른 새 프로그램을 만들어 볼 수 있는 기회였다. 간단한 모드 입력 받기부터 시작해서 복잡한 시간 계산까지 시행착오를 많이 겪었다. 실행이 잘 되는 코드도 어떻게 하면 더 가독성을 높일 수 있을지 많은 시간을 투자해 고민했다. 어렵고 막막했던 코드들이 완성되고 제 기능을 할 때 뿌듯하기도 하고 고민했던 시간들이 헛되지 않았다는 생각이 들었다. 또한 성장한 모습이 결과물로 보여서 또 다른 코드를 작성할 수 있는 원동력이 되었다. 나중에는 더 많은 기능을 추가하고 싶은 욕심이 생기기기도 했다. 간단하다고 생각했던 기능들이 몇일에 걸쳐 완성되기도 했지만 오랜 시간을 투자한 만큼 더 탄탄한 코드가 완성되는 것을 깨달았다. 이번

프로젝트에서 가장 기억에 남는 부분은 같은 기능이여도 코드를 작성하는 방법이 다양하다는 점이다. 이미 작성했던 코드들을 헤더파일로 분할 한 것과, 동적 메모리를 할당해 데이터를 저장하는 것 처럼 하나의 코드가 구현되는 방법에 여러 가지가 있다는 점을 배웠다. 프로젝트를 통해 지금까지 배운 C언어를 점검하기도 했지만 프로젝트를 수행할 때 갖춰야 할 자세나 보고서를 작성하는 방법도 같이 배울 수 있어서 좋았다. 업무 자동화를 구현한 스케줄 플래너를 위해 기획하고 실행하고 발전시키기까지 각 단계를 잘 마무리 한 것 같다.