

C Programming and Practice Project Practice REPORT



수강 과 목 | C프로그래밍및실습

담당 교 수 | 김미수

학 과 | 인공지능학부

학 번 | 214249

이 름 | 김건형

제 출 일 | 2023.10.26.

목차

목차.....	1
I. 서론.....	3
1. 프로젝트 목적 및 배경.....	3
2. 목표.....	3
II. 요구사항.....	3
1. 사용자 요구사항.....	3
2. 기능 요구사항.....	3
III. 설계 및 구현.....	4
1. 기능별 구현 사항.....	4
i. 메뉴 출력 및 프로그램 시작.....	4
ii. 할 일 추가.....	5
iii. 할 일 삭제.....	7
iv. 목록 확인.....	8
v. 프로그램 종료.....	9
vi. 할 일 수정.....	10
IV. 테스트.....	11
1. 기능별 테스트 결과.....	11
i. 할 일 추가하기.....	11
ii. 할 일 삭제하기.....	12
iii. 목록 확인하기.....	12
iv. 할 일 수정하기.....	12
v. 종료하기.....	13
2. 최종 테스트 스크린샷.....	14

V. 결과 및 결론	15
1. 프로젝트 결과	15
2. 느낀 점	15

I. 서론

1. 프로젝트 목적 및 배경

대부분의 1학년 학부생들은 프로젝트를 진행해본 경험이 없는 경우가 많다. 이러한 상황에서 프로젝트를 갑자기 진행하게 된다면 소스코드가 중구난방으로 작성될 가능성이 높다. 이러한 문제를 해결하기 위해 할 일 목록을 관리하는 프로그램을 만들어보며 기능 단위 함수화를 몸에 익히는 경험을 하는 것을 목적으로 한다.

2. 목표

할 일 목록 관리 프로그램(이하 TO DO Manager)를 만들어 보며 문자열 데이터를 처리해보고, 기능들을 함수화하며 재사용할 수 있도록 구현하는 것이 목표이다

II. 요구사항

1. 사용자 요구사항

TO DO Manager에서는 사용자가 할 일 목록을 생성, 삭제 수정할 수 있어야하고, 현재 할 일을 확인할 수 있는 기능을 제공해야 한다. 또한 사용의 편의를 위하여 프로그램을 종료할 때에는 특정한 키를 입력하여 프로그램을 종료할 수 있도록 한다. 사용자는 이 모든 기능을 사용할 수 있다는 사실을 알 수 있어야 한다.

2. 기능 요구사항

최대 길이 100의 문자열을 최대 10개 입력 받아 저장할 수 있어야하고, 해당 문자열마다 수정, 삭제할 수 있는 기능, 현재 저장되어 있는 문자열을 확인할 수 있는 기능이 있어야 한다. 또한 특정한 키를 입력하면 프로그램이 종료되는 기능 또한 필요하다. 이러한 기능은 메뉴로 정리되어 사용자에게 출력된다.

III. 설계 및 구현

1. 기능별 구현 사항

i. 메뉴 출력 및 프로그램 시작

```
#include <stdio.h>
// 최대 할 일의 개수
#define MAX_TASKS 10
// 최대 할 일의 길이
#define CHAR_NUM 100
#include <string.h>
```

그림 1. 문자열을 저장하기 위해 최대 크기를 상수로 선언하는 코드블록

최대 길이 100의 문자열을 10개 저장할 수 있도록 하는 코드를 작성하기 위해 소스코드 가장 위에서 이 수들을 상수로 선언해준다. 이렇게 작성하는 이유는 후에 유지보수에 있어 최대 길이가 변하게 된다면 이 값만 바꿔주면 되는 편리함이 있기 때문이다.

```
// 할 일을 저장할 배열
char tasks[MAX_TASKS][CHAR_NUM] = {" "};
// 할 일의 개수 변수
int taskCount = 0;
printf("TODO 리스트 시작! \n");

while (1) {
    // 사용자의 선택을 저장할 변수
    int choice = -1;

    // 메뉴 출력
    printf("-----\n");
    printf("메뉴를 입력해주세요.\n");
    printf("1. 할 일 추가\n2. 할 일 삭제\n3. 목록 보기\n4. 종료\n5. 할 일 수정\n");
    printf("현재 할 일 수 = %d\n", taskCount);
    printf("-----\n");
    // 사용자의 선택을 입력받음
    scanf_s("%d", &choice);
```

그림 2. 프로그램 시작 및 메뉴 출력을 위한 코드블록

tasks 배열을 선언하여 문자열을 저장할 수 있도록하고, taskCount 변수를 선언하여 현재 몇 개의 task가 저장되어 있는지 알 수 있도록 변수로 선언해준다.

이 후 "TODO 리스트 시작!" 이라는 안내 문구를 출력하여 프로그램이 시작되었음을 사용자에게 알려준다.

그 후 while문을 통해 사용자가 종료하기 전까지 계속 프로그램이 돌아갈 수 있도록 무한 반복문을 사용하고 사용자의 메뉴 선택을 저장해 놓을 choice라는 변수를 선언한다.

사용자가 어떤 메뉴가 있는지 알 수 있도록 할 일 추가, 할 일 삭제, 목록 보기, 종료, 할 일 수정이라는 메뉴를 줄마다 출력해준다.

각각의 기능은 1부터 5까지의 번호로 사용자가 선택할 수 있도록 한다. scanf_s 함수를 통해 사용자의 선택을 choice에 저장해준다.

ii. 할 일 추가

```
// 할 일 추가
case 1:
    printf("할 일을 입력하세요 (공백 없이 입력하세요): ");
    // 할 일을 tasks 배열에 저장
    scanf_s("%s", tasks[taskCount], (int)sizeof(tasks[taskCount]));
    printf("할 일 \"%s\"가 저장되었습니다\n\n", tasks[taskCount]);
    // 할 일의 개수를 1 증가
    taskCount++;
    break;
```

그림 3. 사용자가 할 일을 추가하는 기능이 있는 코드블록

사용자가 할 일의 내용을 입력할 수 있도록 할 일을 입력하라는 안내 문구를 출력하고, 입력을 받아서 tasks배열에 저장한다. 저장한 후에는 taskCount 변수에 1을 더해 현재 할 일의 개수가 하나 늘어났다는 것을 프로그램에서 알 수 있도록 한다.

```

void addTask(char tasks[][CHAR_NUM]) {
    printf("할 일을 입력하세요 (공백 없이 입력하세요): ");
    // 할 일을 tasks 배열에 저장
    scanf_s("%s", tasks[taskCount], (int)sizeof(tasks[taskCount]));
    printf("할 일 \"%s\"가 저장되었습니다\n\n", tasks[taskCount]);
    // 할 일의 개수를 1 증가
    taskCount++;
}

```

그림 4. 사용자가 할 일을 추가하는 기능이 있는 함수

같은 기능이 함수화 되어 구현되었을 때에는 매개변수로 tasks를 받는다. 문자열을 입력받은 다음 다시 반환하여 할당하는 과정 대신에 문자열을 매개변수로 받아 입력과 할당을 한번에 해결한 것이다. 배열은 고정형 포인터와 같이 동작하기 때문에 매개변수로 넘겨 바로 입력을 할당해도 값이 변한다. 또한 코드 블록과 다른 점은 taskCount를 함수 내부에서 증가할 수 있도록 하기 위해 taskCount를 전역 변수로 선언해주었다. (모든 함수에서 사용 가능 하도록)

iii. 할 일 삭제

```
// 할 일 삭제
case 2:
    printf("삭제할 할 일의 번호를 입력해주세요. (1부터 시작):");
    // 삭제할 할 일의 인덱스를 입력받음
    scanf_s("%d", &delIndex);
    // 삭제할 할 일의 인덱스가 유효한지 확인
    if (delIndex > taskCount || delIndex <= 0 )
    {
        printf("삭제 범위가 벗어났습니다.\n");
    }
    else
    {
        printf("%d. %s : 할 일을 삭제합니다.\n", delIndex, tasks[delIndex - 1]);

        // 지을 할 일 인덱스부터 뒤의 할 일을 한 칸씩 앞으로 당김
        for (int i = delIndex; i < taskCount + 1; i++) {
            strcpy_s(tasks[i - 1], sizeof(tasks[i]), tasks[i]);
        }
        // 할 일의 개수를 1 감소
        taskCount -= 1;
    }
    break;
```

그림 5. 사용자가 할 일을 삭제할 수 있는 기능이 있는 코드블록

사용자가 어떤 할 일을 삭제할지 선택할 수 있도록 삭제할 할 일의 번호를 입력받아 delIndex에 저장한다. (이전에 int 변수로서 선언되었다.) 만약 삭제할 할 일의 번호가 현재 저장되어 있는 할 일의 개수보다 많거나 1보다 작다면 삭제 범위에서 벗어난 것이므로 삭제 범위에서 벗어났다는 안내 문구만 출력한다.

만약 입력이 범위 안에 잘 들어와 있다면 delIndex부터 하나씩 위의 할 일 목록을 돌며 저장되어 있는 할 일을 덮어써준다. 이때 주의할 점은 tasks에서 인덱스를 사용할 때 delIndex-1을 (코드에서는 for문을 사용하고 있으므로 i-1) 사용해야 한다는 것이다. 사용자에게 할 일 목록의 번호는 1부터 시작하지만 C언어에서 배열의 인덱스는 0부터 시작하는 차이를 해결하기 위함이다.

배열에서 해당 할 일을 삭제한 후에는 taskCount를 하나 줄여 현재 할 일이 하나 줄어들었다는 것을 프로그램에서 사용할 수 있도록 한다.


```

void delTask(char tasks[][CHAR_NUM]) {
    int delIndex = -1;
    printf("삭제할 할 일의 번호를 입력해주세요. (1부터 시작):");
    // 삭제할 할 일의 인덱스를 입력받음
    scanf_s("%d", &delIndex);
    // 삭제할 할 일의 인덱스가 유효한지 확인
    if (delIndex > taskCount || delIndex <= 0)
    {
        printf("삭제 범위가 벗어났습니다.\n");
    }
    printf("%d. %s : 할 일을 삭제합니다.\n", delIndex, tasks[delIndex - 1]);

    // 지을 할 일 인덱스부터 뒤의 할 일을 한 칸씩 앞으로 당김
    for (int i = delIndex; i < taskCount + 1; i++) {
        strcpy_s(tasks[i - 1], sizeof(tasks[i]), tasks[i]);
    }
    // 할 일의 개수를 1 감소
    taskCount--;
}

```

그림 6. 사용자가 할 일을 삭제할 수 있는 기능이 있는 함수

같은 기능을 함수화하면 입력을 받는 함수와 마찬가지로 매개변수로 tasks 배열을 받는다. 보통의 매개변수라면 매개변수의 값을 변경한다고 하여서 함수를 호출한 과정에 있던 변수의 값이 변경되지는 않지만 배열은 포인터와 같이 동작하기 때문에 값을 변경하면 실제 변수(배열)의 값이 변경된다.

다른 내용은 코드블록에 있는 설명과 동일하다.

iv. 목록 확인

```

// 할 일 목록 보기
case 3:
    printf("할 일 목록\n");
    // 할 일의 개수만큼 반복하며 할 일을 출력
    for (int i = 0; i < taskCount; i++) {
        printf("%d. %s \n", i + 1, tasks[i]);
    }
    printf("\n");
    break;

```

그림 7. 저장된 할 일을 확인하는 기능이 있는 소스코드

사용자가 할 일 목록이 출력된다는 사실을 알 수 있도록 할 일 목록이라는 문구를 출력하고, 현재 할 일의 개수만큼 for문을 실행한다. 이 때 주의할 점은 for문 내의 i는 0부터 taskCount-1만큼 증가하는데, tasks내의 인덱스는 맞지만 사용자가 인지하는 할 일의 번호와는 다르다는 것이다. 그렇기 때문에 출력할 때 %d에 할당되는 변수가 i가 아니라 i+1이다. (tasks의 인덱스는 i인 점을 주의).

개행문자 (\n)을 매 할 일 마다 출력하여 사용자가 할 일을 더 잘 확인 할 수 있도록 한다.

```
void printTask(char tasks[][CHAR_NUM]) {
    printf("할 일 목록\n");
    // 할 일의 개수만큼 반복하며 할 일을 출력
    for (int i = 0; i < taskCount; i++) {
        printf("%d. %s \n", i + 1, tasks[i]);
    }
    printf("\n");
}
```

그림 8. 저장된 할 일을 확인하는 기능이 있는 함수

매개변수로 tasks를 받아오는 것을 제외하면 소스코드와 동일하게 동작한다.

v. 프로그램 종료

```
// 종료
case 4:
    // 종료 상태 변수를 1로 변경
    terminate = 1;
    break;
```

그림 9. 프로그램의 종료 상태를 변경하는 소스코드

```
// 종료 상태 변수가 1로 변경되면 반복문 종료
if (terminate == 1) {
    printf("종료를 선택하셨습니다. 프로그램을 종료합니다.\n");
    break;
}
```

그림 10. 종료 상태 변수를 확인하고 프로그램을 종료하는 소스코드

사용자가 프로그램의 종료를 원한다면 맨 처음 선언한 while문을 종료할 수 있

도록 terminate 변수를 (위에서 int로 선언됨) 1로 변경하고 만약 1이라면 while문에서 break를 실행한다.

여기서 주의해야 할 점은 그림 10에 있는 break문은 case문에서 빠져나오는 break가 아니라 while문을 빠져나오는 break이기 때문에 case문이 모두 종료된 후에 작성되어야 한다는 것이다.

vi. 할 일 수정

```
case 5:
    printf("수정할 할 일의 번호를 입력해주세요. (1부터 시작):");
    // 수정할 할 일의 인덱스를 입력받음
    scanf_s("%d", &changeIndex);
    // 수정할 할 일의 인덱스가 유효한지 확인
    if (changeIndex > taskCount || changeIndex <= 0)
    {
        printf("수정 범위가 벗어났습니다..\n");
        break;
    }
    printf("수정할 내용을 입력해주세요 : ");
    // 내용 입력 전에 버퍼를 비움
    scanf_s("%c", &ch, 1);
    // 할 일을 tasks 배열에 저장
    scanf_s("%[^\n]s", tasks[changeIndex - 1], (int)sizeof(tasks[changeIndex - 1]));
    break;
```

그림 11. 할 일을 수정할 수 있는 기능이 있는 소스코드

수정하는 기능의 경우에 삭제하는 기능과 유사하게 동작한다. 먼저 수정할 할 일의 번호를 changeIndex(위에서 int로 선언됨)에 입력받은 후에 해당 번호가 현재 저장되어 있는 할 일의 개수에서 벗어나지 않는지 확인해준다.

이후에 입력 버퍼를 비워주는데, 이는 인덱스를 입력하며 개행문자가 버퍼에 들어있을 수 있기 때문이다.

새로운 할 일의 내용을 입력받고 tasks에 저장한다. 이 때 저장되는 위치, 즉 tasks의 인덱스는 changeIndex-1이 된다. (배열의 인덱스와 사용자가 인지하고 있는 할 일의 번호가 다르기 때문).

```

void editTask(char tasks[][CHAR_NUM]) {
    int editIndex = -1;
    char ch;
    printf("수정할 할 일의 번호를 입력해주세요. (1부터 시작):");
    // 수정할 할 일의 인덱스를 입력받음
    scanf_s("%d", &editIndex);
    // 수정할 할 일의 인덱스가 유효한지 확인
    if (editIndex > taskCount || editIndex <= 0)
    {
        printf("수정 범위가 벗어났습니다..\n");
    }
    printf("수정할 내용을 입력해주세요 : ");
    // 입력 전에 버퍼를 비움
    scanf_s("%c", &ch, 1);
    // 수정할 내용을 입력받음
    scanf_s("%[^\n]s", tasks[editIndex - 1], (int)sizeof(tasks[editIndex - 1]));
    printf("%d. %s : 할 일을 수정합니다.\n", editIndex, tasks[editIndex - 1]);
    printf("\n");
}

```

그림 12. 할 일을 수정할 수 있는 기능이 있는 함수

매개변수로 tasks를 받는다는 것을 제외하면 소스코드와 동일하게 동작한다.

IV. 테스트

1. 기능별 테스트 결과

i. 할 일 추가하기

```

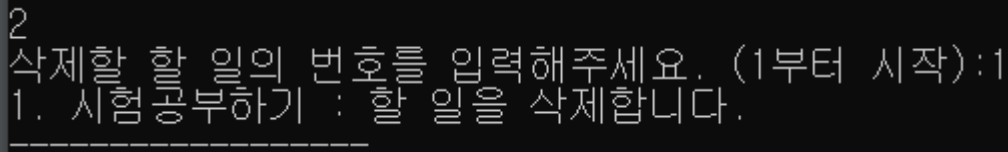
1
할 일을 입력하세요 (공백 없이 입력하세요): 시험공부하기
할 일 시험공부하기가 저장되었습니다

```

그림 13. 할 일을 추가하는 테스트를 실행한 결과

사용자가 1을 입력하면 할 일을 입력하라는 안내가 나오고 시험공부하기라는 할 일을 입력하였을 때 시험공부하기라는 할 일이 잘 저장된 모습이다.

ii. 할 일 삭제하기

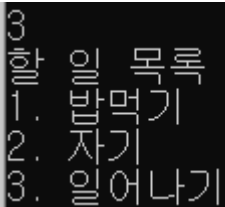


```
2
삭제할 할 일의 번호를 입력해주세요. (1부터 시작):1
1. 시험공부하기 : 할 일을 삭제합니다.
```

그림 14. 할 일을 삭제하는 테스트를 실행한 결과

사용자가 2를 입력하면 삭제할 할 일의 번호를 입력하라는 안내가 나오고 1번을 선택하자 시험공부하기라는 할 일이 삭제된 모습이다.

iii. 목록 확인하기

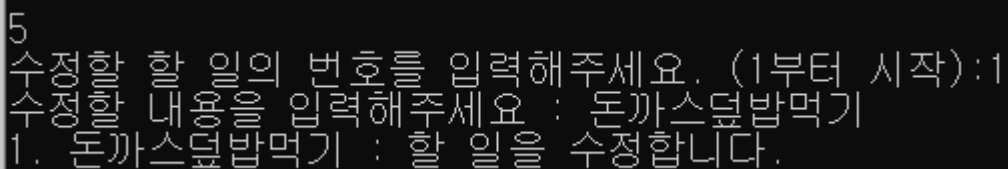


```
3
할 일 목록
1. 밥먹기
2. 자기
3. 일어나기
```

그림 15. 할 일 목록을 확인하는 테스트를 실행한 결과

사용자가 3을 입력하면 현재 저장된 할 일을 확인할 수 있다.

iv. 할 일 수정하기

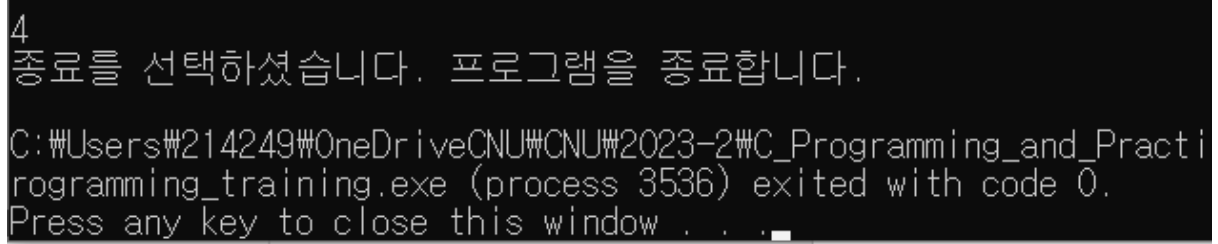


```
5
수정할 할 일의 번호를 입력해주세요. (1부터 시작):1
수정할 내용을 입력해주세요 : 돈까스덮밥먹기
1. 돈까스덮밥먹기 : 할 일을 수정합니다.
```

그림 16. 할 일을 수정하는 테스트를 실행한 결과

사용자가 5를 입력하면 수정할 할 일의 번호를 입력하라는 안내 문구가 출력되고 1번을 선택하자 수정할 내용을 입력하라는 안내가 다시 출력도니다. 돈까스덮밥먹기라는 내용으로 할 일을 수정하자 잘 수정되었다는 문구가 출력되는 것을 확인 할 수 있다.

v. 종료하기



```
4
종료를 선택하셨습니다. 프로그램을 종료합니다.
C:\Users\214249\OneDrive\CU\2023-2\C_Programming_and_Practi
rogramming_training.exe (process 3536) exited with code 0.
Press any key to close this window . . .
```

그림 17. 프로그램을 종료하는 테스트를 실행한 결과

사용자가 4를 선택하자 프로그램을 종료한다는 안내가 출력되는 모습을 확인할 수 있다.

2. 최종 테스트 스크린샷



그림 18. 모든 테스트를 실행한 결과

v. 결과 및 결론

1. 프로젝트 결과

기능 요구사항에 맞추어 프로그램을 잘 작성할 수 있었다. 문자열을 배열로 다루고 매개변수로 사용하여 함수화하는 작업 또한 성공적이었다. 이제 사용자는 이 프로그램을 통해서 간단한 할 일을 관리할 수 있을 것이다. 다만 프로그램이 종료된 후에는 할 일을 다시 불러올 수 없다는 점이 한계라고 생각한다.

2. 느낀 점

기능 단위로 함수화하며 프로그램 유지 보수에 대한 개념을 한 층 더 깊게 알 수 있게 된 기회가 된 것 같다. 다음 차시 내용인 프로젝트를 실제로 진행한다면 기능을 함수화하여 잘 사용할 수 있을 것 같다.