

Report 5

Ch. 10 - 함수 기초 (2)

학번	2018212236
학부	전자정보통신공학
이름	김동주

제출일자	2018-10-14
담당교수	반상우

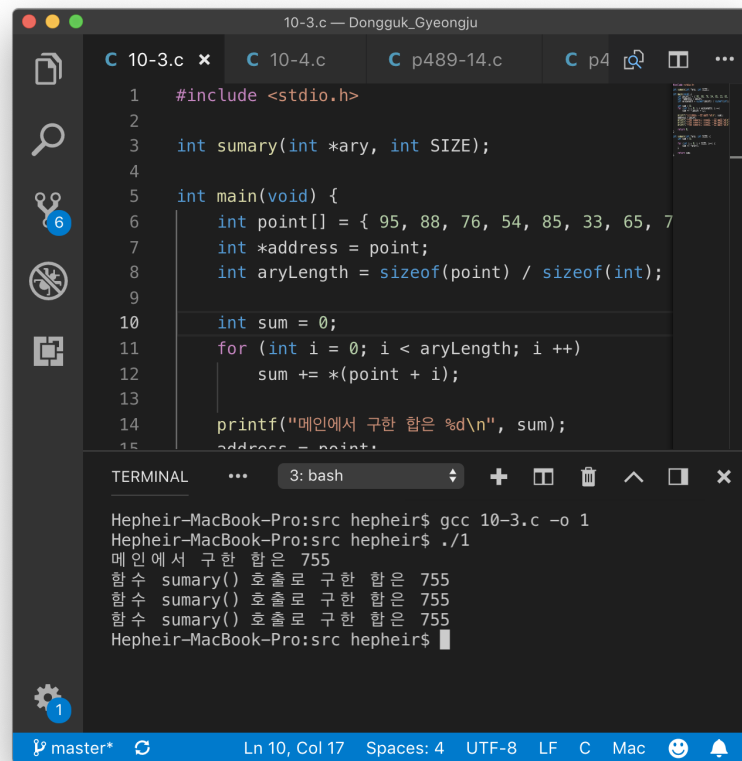
작업환경	Visual Studio Code
컴파일러	MinGW GCC

필수 문제

1. 예제 10-3

내용	교과서 예제 10-3번을 구현
코드	<pre>#include <stdio.h> int sumary(int *ary, int SIZE); int main(void) { int point[] = { 95, 88, 76, 54, 85, 33, 65, 78, 99, 82 }; int *address = point; int aryLength = sizeof(point) / sizeof(int); int sum = 0; for (int i = 0; i < aryLength; i++) sum += *(point + i); printf("메인에서 구한 합은 %d\n", sum); address = point; printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(point, aryLength)); printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(&point[0], aryLength)); printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(address, aryLength)); return 0; } int sumary(int *ary, int SIZE) { int sum = 0; for (int i = 0; i < SIZE; i++) { sum += *ary++; } return sum; }</pre>

실행결과



The screenshot shows a code editor with a C program named 10-3.c. The program defines a function `sumary` and a `main` function. In `main`, an array `point` is initialized with values {95, 88, 76, 54, 85, 33, 65, 7}, and `aryLength` is calculated using `sizeof(point) / sizeof(int)`. The `sumary` function is called three times, and the results are printed. The terminal output shows the compilation and execution of the program, resulting in the value 755 being printed three times.

```
#include <stdio.h>

int sumary(int *ary, int SIZE);

int main(void) {
    int point[] = { 95, 88, 76, 54, 85, 33, 65, 7 };
    int *address = point;
    int aryLength = sizeof(point) / sizeof(int);

    int sum = 0;
    for (int i = 0; i < aryLength; i++)
        sum += *(point + i);

    printf("메인에서 구한 합은 %d\n", sum);
    address = point;
}
```

```
Hepheir-MacBook-Pro:src hepheir$ gcc 10-3.c -o 1
Hepheir-MacBook-Pro:src hepheir$ ./1
메인에서 구한 합은 755
함수 sumary() 호출로 구한 합은 755
함수 sumary() 호출로 구한 합은 755
함수 sumary() 호출로 구한 합은 755
Hepheir-MacBook-Pro:src hepheir$
```

고찰

배열의 길이를 구할 때,

```
int aryLength = sizeof(point) / sizeof(int);
```

위와 같이 `sizeof` 함수를 이용하는 방법이 있다는 사실을 10-3 예제를 통해 알게되었다. 기존에는 전처리를 이용하여 배열의 크기를 미리 정해놓은 후에, 매크로를 사용하였는데, 위 방식은 굉장히 유용할 것 같다고 생각했다.

2. 예제 10-4

내용	교과서 예제 10-4번을 구현
코드	<pre>#include <stdio.h> double sum(double data[][3], int, int); void printarray(double data[][3], int, int); int main(void) { double x[][3] = { { 1, 2, 3 }, { 7, 8, 9 }, { 4, 5, 6 }, { 10, 11, 12 } }; int rowsize = sizeof(x) / sizeof(x[0]); int colsize = sizeof(x[0]) / sizeof(x[0][0]); printf("2차원 배열의 자료값은 다음과 같습니다.\n"); printarray(x, rowsize, colsize); printf("2차원 배열 원소합은 %.31f 입니다.\n", sum(x, rowsize, colsize)); return 0; } double sum(double (*data)[3], int rowsize, int colsize) { double total = 0; for (int i = 0; i < rowsize; i++) for (int j = 0; j < colsize; j++) total += data[i][j]; return total; } void printarray(double (*data)[3], int rowsize, int colsize) { for (int i = 0; i < rowsize; i++) { printf("%d행원소: ", i + 1);</pre>

	<pre> for (int j = 0; j < colsize; j++) printf("x[%d][%d] = %5.2lf ", i, j, data[i][j]); printf("\n"); } printf("\n"); }</pre>
실행결과	 <p>10-4.c — Dongguk_Gyeongju</p> <pre>19 20 return 0; 21 22 23 double sum(double (*data)[3], int rowsize, int colsize) 24 { 25 double total = 0; 26 for (int i = 0; i < rowsize; i++) 27 for (int j = 0; j < colsize; j++) 28 total += data[i][j]; 29 } 30 return total; 31 32</pre> <p>3: bash</p> <p>2차원 배열의 자료값은 다음과 같습니다. 1행 원소: x[0][0] = 1.00 x[0][1] = 2.00 x[0][2] = 3.00 2행 원소: x[1][0] = 7.00 x[1][1] = 8.00 x[1][2] = 9.00 3행 원소: x[2][0] = 4.00 x[2][1] = 5.00 x[2][2] = 6.00 4행 원소: x[3][0] = 10.00 x[3][1] = 11.00 x[3][2] = 12.00</p> <p>2차원 배열 원소합은 78.000 입니다. Hepheir-MacBook-Pro:src hepheir\$</p> <p>Ln 29, Col 5 Spaces: 4 UTF-8 LF C Mac</p>
고찰	<p>1차 배열(가장 바깥 배열?)의 크기가 상수로 정의되지 않은 배열 x를 인자로 받아야 하는 함수 sum에서 이를 어떻게 처리하였는지를 보았다.</p> <pre>... double x[][3] = { { 1, 2, 3 }, ... { 10, 11, 12 } }; ... double sum(double (*data)[3], int rowsize, int colsize)</pre>

...
'double (*data)[3]'.
크기가 다양한 배열을 같은 함수에서 처리할 때, 위와 같은 표현을 사용한 것이 매우 신기하였다.

위 표현방법이 정확히 어떤 원리로 동작하는지가 궁금하여 인터넷에 검색해 보았다.

찾아본 자료들 중에서 아래 링크에서 알 수 있었던 바로는

https://stackoverflow.com/questions/13910749/difference-between_ptr10-and_ptr10

For the following code:

```
int (*ptr)[10];  
int a[10]={99,1,2,3,4,5,6,7,8,9};  
ptr=&a;  
printf("%d", (*ptr)[1]);
```

What should it print? I'm expecting the garbage value here but the output is 1.

(for which I'm concluding that initializing this way pointer array i.e `ptr[10]` would start pointing to elements of `a[10]` in order).

But what about this code fragment:

```
int *ptr[10];  
int a[10]={0,1,2,3,4,5,6,7,8,9};  
*ptr=a;  
printf("%d", *ptr[1]);
```

It is giving the segmentation fault.

```
int *ptr[10];
```

This is an array of 10 `int*` pointers, not as you would assume, a pointer to an array of 10 `ints`

```
int (*ptr)[10];
```

This is a pointer to an array of 10 `int`

It is I believe the same as `int *ptr;` in that both can point to an array, but the given form can ONLY point to an array of 10 `int`s

`double (*pointer)[10]` 을 인자로 받는다는 것은,

“a pointer to an array of 10 `int` (여기선 `double`)“ :

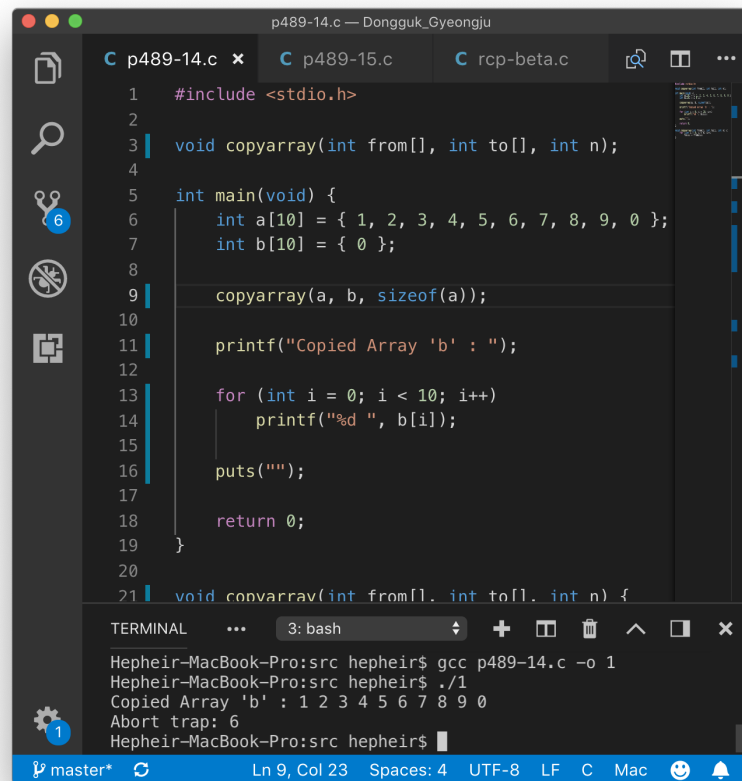
10개의 `int(double)`을 원소로 갖는 어떤 배열을 받는다는 것을 의미한다.

그렇기에 $N * 3$ 의 크기로 선언된 `int`형 배열 `x`를 `sum`함수에서 받을 수 있었던 것이다.

3. p.489-14

내용	<p>다음과 같이 일차원 배열을 복사하는 함수를 작성하여 결과를 알아보는 프로그램을 작성하시오.</p> <ul style="list-style-type: none">• void copyarray(int from[], int to[], int n /* 배열 원소 수 */)• 배열 from의 첫 번째 원소부터 (n-1)번째 원소까지 같은 순서대로 배열 to로 값을 복사하는 함수
코드	<pre>#include <stdio.h> void copyarray(int from[], int to[], int n); int main(void) { int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }; int b[10] = { 0 }; copyarray(a, b, sizeof(a)); printf("Copied Array 'b' : "); for (int i = 0; i < 10; i++) printf("%d ", b[i]); puts(""); return 0; } void copyarray(int from[], int to[], int n) { for (int i = 0; i < n; i++) to[i] = from[i]; }</pre>

실행결과



```
p489-14.c — Dongguk_Gyeongju
C p489-14.c x C p489-15.c C rcp-beta.c

1  #include <stdio.h>
2
3  void copyarray(int from[], int to[], int n);
4
5  int main(void) {
6      int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
7      int b[10] = { 0 };
8
9      copyarray(a, b, sizeof(a));
10
11     printf("Copied Array 'b' : ");
12
13     for (int i = 0; i < 10; i++)
14         printf("%d ", b[i]);
15
16     puts("");
17
18     return 0;
19 }
20
21 void copyarray(int from[], int to[], int n) {
```

```
TERMINAL  ...  3: bash
Hepheir-MacBook-Pro:src hepheir$ gcc p489-14.c -o 1
Hepheir-MacBook-Pro:src hepheir$ ./1
Copied Array 'b' : 1 2 3 4 5 6 7 8 9 0
Abort trap: 6
Hepheir-MacBook-Pro:src hepheir$
```

고찰

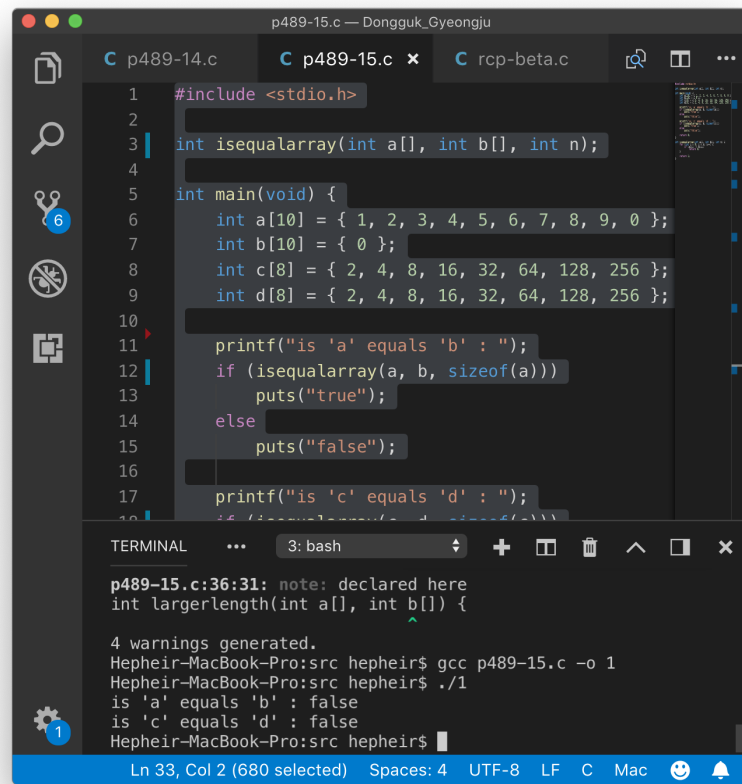
Report 4에서 선택문제로 풀었던 문제이다.

이번에 다시 풀어보면서 예제 10-3에서 보았던 `sizeof()`를 이용하여 배열의 크기를 알아내는 방법을 사용해보았다.

4. p.489-15

<p>내용</p>	<p>다음과 같이 일차원 배열의 동등함을 검사하는 함수를 작성하여 결과를 알아보는 프로그램을 작성하시오.</p> <ul style="list-style-type: none"> • int isequalarray(int a[], int b[], int n /* 배열 원소 수 */) • 배열 a와 b의 배열크기가 모두 n이며 순차적으로 원소 값이 모두 같으면 1을 반환, 아니면 0을 반환 하는 함수
<p>코드</p>	<pre>#include <stdio.h> int isequalarray(int a[], int b[], int n); int main(void) { int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }; int b[10] = { 0 }; int c[8] = { 2, 4, 8, 16, 32, 64, 128, 256 }; int d[8] = { 2, 4, 8, 16, 32, 64, 128, 256 }; printf("is 'a' equals 'b' : "); if (isequalarray(a, b, sizeof(a))) puts("true"); else puts("false"); printf("is 'c' equals 'd' : "); if (isequalarray(c, d, sizeof(c))) puts("true"); else puts("false"); return 0; } int isequalarray(int a[], int b[], int n) { for (int i = 0; i < n; i++) { if (a[i] != b[i]) return 0; } return 1; }</pre>

실행결과



The screenshot shows a code editor window titled "p489-15.c — Dongguk_Gyeongju". The code is as follows:

```
1 #include <stdio.h>
2
3 int isequalarray(int a[], int b[], int n);
4
5 int main(void) {
6     int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
7     int b[10] = { 0 };
8     int c[8] = { 2, 4, 8, 16, 32, 64, 128, 256 };
9     int d[8] = { 2, 4, 8, 16, 32, 64, 128, 256 };
10
11     printf("is 'a' equals 'b' : ");
12     if (isequalarray(a, b, sizeof(a)))
13         puts("true");
14     else
15         puts("false");
16
17     printf("is 'c' equals 'd' : ");
18     if (isequalarray(c, d, sizeof(c)))
19         puts("true");
20     else
21         puts("false");
22 }
```

The terminal output shows the following:

```
TERMINAL 3: bash
p489-15.c:36:31: note: declared here
int largerlength(int a[], int b[]) {
4 warnings generated.
Hepheir-MacBook-Pro:src hepheir$ gcc p489-15.c -o 1
Hepheir-MacBook-Pro:src hepheir$ ./1
is 'a' equals 'b' : false
is 'c' equals 'd' : false
Hepheir-MacBook-Pro:src hepheir$
```

고찰

Report 4에서 선택문제로 풀었던 문제이다.

이번에 다시 풀어보면서 예제 10-3에서 보았던 `sizeof()`를 이용하여 배열의 크기를 알아내는 방법을 사용해보았다.

5. 가위바위보 게임 구현

내용	사용자의 선택에 따라 게임이 반복적으로 실행되도록 함.
코드	<pre>#include <stdio.h> #include <stdlib.h> #include <time.h> void __init__(); void Select(); void Match(); void printScr_Hr_Bold(); void printScr_Hr(); int CPU, USER; int main(void) { __init__(); while (1) { Select(); if (USER == 0) break; CPU = rand() % 3 + 1; Match(); } return 0; } void __init__() { long seconds = (long) time(NULL); srand(seconds); printScr_Hr_Bold(); puts(" 가위 바위 보 게임!!");</pre>

```

    printScr_Hr_Bold();
}

void Select() {
    printScr_Hr();

    puts(" 무엇을 내시겠습니까?");
    puts("");
    puts(" 1. 가위");
    puts(" 2. 바위");
    puts(" 3. 보");
    puts("");
    puts(" 0. 종료");
    printf(">>> ");

    scanf("%d", &USER);
}

void Match() {
    printScr_Hr();

    printf("컴퓨터는 ");

    switch (CPU)
    {
        case 1:
            printf("가위");
            break;
        case 2:
            printf("바위");
            break;
        case 3:
            printf("보");
            break;
    }

    printf("를, 당신은 ");

    switch (USER)
    {
        case 1:

```

```

        printf("가위");
        break;
    case 2:
        printf("바위");
        break;
    case 3:
        printf("보");
        break;
    }
    puts("를 냈다.\n");

    // 승부를 판가름 하는 부분.

    if (USER == CPU) {
        puts("비겼다.");
        return;
    }

    // 비기지 않았을 경우, CPU가 유저를 이겼는지 검사.

    if (++USER > 3) USER = 1;

    if (USER == CPU) {
        puts("졌다...");
        return;
    }

    // CPU가 유저를 이기지 못했으면 유저의 승리 외엔 경우의 수가
    없다.

    printf("이겼다!!!\n");
    return;
}

void printScr_Hr_Bold() {
    puts("=====");
}

void printScr_Hr() {
    puts("-----");
}

```

실행결과

```

p489-15.c — Dongguk_Gyeongju
p489-14.c  p489-15.c x  rcp-beta.c
1  #include <stdio.h>
2
TERMINAL  ...  3: bash
=====
가위 바위 보 게임!!
=====
무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

0. 종료
>>> 1

컴퓨터는 바위를, 당신은 가위를 냈다.

졌다...

무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

0. 종료
>>> 2

컴퓨터는 가위를, 당신은 바위를 냈다.

이겼다!!!

무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

0. 종료
>>> 3

컴퓨터는 바위를, 당신은 보를 냈다.
  
```

고찰

최대한 main함수에서 이 코드 전체의 흐름을 한 눈에 파악할 수 있게끔 작성해보았다.

이 게임의 특성상 한 변수를 다양한 함수에서 처리해야 하는 경우가 많다.
그렇기에

1. 변수를 선언하고, &(주소 연산자)를 통해 각 함수로 연결해 주는 방법을 사용할 지
2. main함수 외부에 전역변수를 선언하여 모든 함수로 하여금 해당 변수로의 접근을 가능하게 할 것인지

에 대하여 고민해보다가, 코드의 간결함을 우선시 하여, 후자의 방법을 택하였다.

이후에, rand() 함수를 제대로 사용하기 위해 s_rand()함수를 호출하는 부분, 변수의 초기값을 선언하는 부분 등 처음 실행시 한

번만 동작해야 하는 부분을 `__init__` 함수로 분리하여 사용하여 불필요하게 main 함수가 길어지는 것을 방지하였다.

그렇게 완성된 main 함수는 다음과 같다.

```
int main(void) {  
    __init__();  
    while (1) {  
        Select();  
        if (USER == 0) break;  
        CPU = rand() % 3 + 1;  
        Match();  
    }  
    return 0;  
}
```


선택 문제

1. 가위바위보 Beta

내용	사용자 정의 함수 이용. 게임 머니 개념 도입 또는 반복적 게임 실행 후 게임 승률 표현. 기타 각자 원하는 기능 추가 구현
코드	<pre>#include <stdio.h> #include <stdlib.h> #include <time.h> // 처음 입력한 크레딧 금액을 LEVEL배 만큼 올리면 승리! #define LEVEL 20 // 구분선 그리는 함수 void printScr_Hr_Bold(); void printScr_Hr(); // 게임 흐름 제어 void __init__(); void Player_bet(); void Player_select(); void Player_match(); void Player_status(); void GameClear(); void GameOver(); // 기타 함수 int intpow(int, int); // 글로벌 변수 int GOAL; int CREDIT; int BET; int CPU, USER;</pre>

```

int WIN = 0, DRAW = 0, LOSE = 0;
int COMBO = 0, MAX_COMBO = 0;

int main(void) {
    __init__();

    while (1) {
        Player_bet();
        if (BET == 0) break;

        CPU = rand() % 3 + 1;

        Player_select();
        Player_match();

        Player_status();

        if (CREDIT <= 0) break;
        if (CREDIT >= GOAL) GameClear();
    }

    GameOver();
    return 0;
}

// 함수 원형

void __init__() {
    long seconds = (long) time(NULL);
    srand(seconds);

    printScr_Hr_Bold();

    puts(" 가위 바위 보 게임!!");

    printScr_Hr_Bold();

    puts(" 크레딧을 넣어주세요...");
    puts("");
    printf(">>> $");
}

```

```

scanf("%d", &CREDIT);

printScr_Hr();

GOAL = CREDIT * LEVEL;
printf(" $%d 크레딧을 모으면 승리!\n", GOAL);
}

void Player_bet() {
    printScr_Hr_Bold();

    puts(" 얼마를 베팅하시겠습니까?");
    puts(" ($0을 베팅하면 종료)");
    printf(" 현재 보유 크레딧 : %d/%d\n", CREDIT, GOAL);
    puts("");

    if (COMBO) printf("(연승 보너스! 승리시 받는 크레딧 x%d)\n",
intpow(2 , COMBO + 1));

    printf(">>> $");

    scanf("%d", &BET);
    CREDIT -= BET;

    if (CREDIT < 0) {
        printScr_Hr();

        CREDIT += BET;
        puts(" 감당 못하실 금액을 입력하셨습니다.");

        Player_bet();
    }

    if (BET < 0) {
        printScr_Hr();

        puts(" [경고] 리스크가 큰 모드입니다.");
        puts(" 이런걸 우리는 사서 고생한다고 하지요.");
    }
}

```

```

void Player_select() {
    printScr_Hr();

    puts(" 무엇을 내시겠습니까?");
    puts("");
    puts(" 1. 가위");
    puts(" 2. 바위");
    puts(" 3. 보");
    puts("");
    printf(">>> ");

    scanf("%d", &USER);

    // Exception Handling.
    if (USER != 1 && USER != 2 && USER != 3) {
        printScr_Hr();

        puts("올바르지 않은 입력입니다!");

        Player_select();
    }
}

void Player_match() {
    printScr_Hr();

    printf("컴퓨터는 ");

    switch (CPU)
    {
        case 1:
            printf("가위");
            break;
        case 2:
            printf("바위");
            break;
        case 3:
            printf("보");
            break;
    }
}

```

```

printf("를, 당신은 ");

switch (USER)
{
    case 1:
        printf("가위");
        break;
    case 2:
        printf("바위");
        break;
    case 3:
        printf("보");
        break;
}

puts("를 냈다.\n");

// 승부를 판가름 하는 부분.

if (USER == CPU) {
    DRAW++;
    COMBO = 0;

    puts("비겼다.");
    printf("베팅한 크레딧 %d을 돌려받았다.\n", BET);

    CREDIT += BET;
    return;
}

// 비기지 않았을 경우, CPU가 유저를 이겼는지 검사.

if (++USER > 3) USER = 1;

if (USER == CPU) {
    LOSE++;
    COMBO = 0;

    puts("졌다...");
    printf("베팅한 크레딧 %d을 잃었다.\n", BET);

    return;
}

```

```

    }

    // CPU가 유저를 이기지 못했으면 유저의 승리 외엔 경우의 수가
    없다.

    printf("이겼다!!!\n");
    WIN++;
    COMBO++;

    if (MAX_COMBO < COMBO)
        MAX_COMBO = COMBO;

    int bonus = intpow(2, (COMBO));

    BET *= bonus;
    printf("베팅한 크레딧의 %d배인 $%d을 벌었다.\n", bonus, BET);

    CREDIT += BET;
}

void Player_status() {
    printScr_Hr();

    printf("현재 전적 : %d 회", (WIN + LOSE + DRAW));

    if (COMBO) printf(" (%d 연승 중)", COMBO);

    puts("");
    printf(" %d 승, %d 패, %d 무, 최대 %d 연승\n", WIN, LOSE,
DRAW, MAX_COMBO);

    printf("현재 승률 : %.1f%% (무승부 제외)\n", (float) WIN /
(WIN + LOSE) * 100);
}

void GameClear() {
    printScr_Hr_Bold();
    puts("당신은 목표를 이루었습니다.");
    puts("당신은 부유합니다.");
    puts("하지만 당신의 승부는 계속됩니다...");
}

```

	<pre> void GameOver() { printScr_Hr_Bold(); puts("파산하셨습니다."); } void printScr_Hr_Bold() { puts("====="); } void printScr_Hr() { puts("-----"); } int intpow(int m, int n) { int sum = 1; for (int i = 0; i < n; i++) sum *= m; return sum; } </pre>
실행결과	<pre> Hepheir-MacBook-Pro:181008 hepheir\$ gcc "가위바위보.c" -o 1 Hepheir-MacBook-Pro:181008 hepheir\$./1 ===== 가위 바위 보 게임!! ===== 크레딧을 넣어주세요... >>> \$100 ----- \$2000 크레딧을 모으면 승리! ===== 얼마를 베팅하시겠습니까? (\$0을 베팅하면 종료) 현재 보유 크레딧 : \$100/\$2000 >>> \$1 ----- 무엇을 내시겠습니까? 1. 가위 2. 바위 3. 보 </pre>

>>> 1

컴퓨터는 보를, 당신은 가위를 냈다.

이겼다!!!
베팅한 크레딧의 2배인 \$2을 벌었다.

현재 전적 : 1 회 (1 연승 중)
1 승, 0 패, 0 무, 최대 1 연승
현재 승률 : 100.0% (무승부 제외)

=====

얼마를 베팅하시겠습니까?
(\$0을 베팅하면 종료)
현재 보유 크레딧 : \$101/\$2000

(연승 보너스! 승리시 받는 크레딧 x4)
>>> \$100

무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

>>> 2

컴퓨터는 가위를, 당신은 바위를 냈다.

이겼다!!!
베팅한 크레딧의 4배인 \$400을 벌었다.

현재 전적 : 2 회 (2 연승 중)
2 승, 0 패, 0 무, 최대 2 연승
현재 승률 : 100.0% (무승부 제외)

=====

얼마를 베팅하시겠습니까?
(\$0을 베팅하면 종료)
현재 보유 크레딧 : \$401/\$2000

(연승 보너스! 승리시 받는 크레딧 x8)
>>> \$201

무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

>>> 3

컴퓨터는 가위를, 당신은 보를 냈다.

졌다...

베팅한 크레딧 \$201을 잃었다.

현재 전적 : 3 회

2 승, 1 패, 0 무, 최대 2 연승

현재 승률 : 66.7% (무승부 제외)

=====

얼마를 베팅하시겠습니까?

(\$0을 베팅하면 종료)

현재 보유 크레딧 : \$200/\$2000

>>> \$100

무엇을 내시겠습니까?

1. 가위

2. 바위

3. 보

>>> 2

컴퓨터는 보를, 당신은 바위를 냈다.

졌다...

베팅한 크레딧 \$100을 잃었다.

현재 전적 : 4 회

2 승, 2 패, 0 무, 최대 2 연승

현재 승률 : 50.0% (무승부 제외)

=====

얼마를 베팅하시겠습니까?

(\$0을 베팅하면 종료)

현재 보유 크레딧 : \$100/\$2000

>>> \$80

무엇을 내시겠습니까?

1. 가위

2. 바위

3. 보

>>> 2

컴퓨터는 가위를, 당신은 바위를 냈다.

이겼다!!!

베팅한 크레딧의 2배인 \$160을 벌었다.

현재 전적 : 5 회 (1 연승 중)
3 승, 2 패, 0 무, 최대 2 연승
현재 승률 : 60.0% (무승부 제외)
=====

얼마를 베팅하시겠습니까?
(\$0을 베팅하면 종료)
현재 보유 크레딧 : \$180/\$2000

(연승 보너스! 승리시 받는 크레딧 x4)
>>> \$60

무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

>>> 1

컴퓨터는 바위를, 당신은 가위를 냈다.

졌다...
베팅한 크레딧 \$60을 잃었다.

현재 전적 : 6 회
3 승, 3 패, 0 무, 최대 2 연승
현재 승률 : 50.0% (무승부 제외)
=====

얼마를 베팅하시겠습니까?
(\$0을 베팅하면 종료)
현재 보유 크레딧 : \$120/\$2000

>>> \$20

무엇을 내시겠습니까?

1. 가위
2. 바위
3. 보

>>> 3

컴퓨터는 바위를, 당신은 보를 냈다.

이겼다!!!
베팅한 크레딧의 2배인 \$40을 벌었다.

현재 전적 : 7 회 (1 연승 중)
4 승, 3 패, 0 무, 최대 2 연승

현재 승률 : 57.1% (무승부 제외)

=====
얼마를 베팅하시겠습니까?

(\$0을 베팅하면 종료)

현재 보유 크레딧 : \$140/\$2000

(연승 보너스! 승리시 받는 크레딧 x4)

>>> \$80

무엇을 내시겠습니까?

1. 가위

2. 바위

3. 보

>>> 3

컴퓨터는 바위를, 당신은 보를 냈다.

이겼다!!!

베팅한 크레딧의 4배인 \$320을 벌었다.

현재 전적 : 8 회 (2 연승 중)

5 승, 3 패, 0 무, 최대 2 연승

현재 승률 : 62.5% (무승부 제외)

=====
얼마를 베팅하시겠습니까?

(\$0을 베팅하면 종료)

현재 보유 크레딧 : \$380/\$2000

(연승 보너스! 승리시 받는 크레딧 x8)

>>> \$320

무엇을 내시겠습니까?

1. 가위

2. 바위

3. 보

>>> 2

컴퓨터는 바위를, 당신은 바위를 냈다.

비겼다.

베팅한 크레딧 \$320을 돌려받았다.

현재 전적 : 9 회

5 승, 3 패, 1 무, 최대 2 연승

현재 승률 : 62.5% (무승부 제외)

=====

	<p>얼마를 베팅하시겠습니까? (\$0을 베팅하면 종료) 현재 보유 크레딧 : \$380/\$2000</p> <p>>>> \$0 =====</p> <p>파산하셨습니다.</p>
고찰	<p>기존의 가위바위보 게임에서</p> <ul style="list-style-type: none"> - 게임 머니 (크레딧) - 전적 표시 (승, 무, 패 / 연승 횟수 / 승률) <p>와 같이 문제에서 요구하는 기능과 함께 굉장히 다양한 기능을 추가하여 실제로 충분히 즐길 수 있는 게임을 만들었다.</p> <ul style="list-style-type: none"> - 동기부여 : 목표금액을 설정해주어, 플레이어로 하여금 게임을 지속하기 위한 목적을 만들어 줌. - 보상 : 승리할 시에, 베팅한 금액의 배로 돌려주게 되는데, 연승을 할 경우 받는 크레딧이 x2, x4, x8, x16, ... 로 증가하여 유저로 하여금 카타르시스를 느낄 수 있게 함. - 레벨 제도 (미완성) : 레벨 계수가 존재하여, 플레이어의 실력에 맞게 난이도를 조절 할 수 있게 함. <p>같은 수업을 듣는 동기가 직접 플레이 해보더니 재미있다고 말해주어 굉장히 기분이 좋았다.</p>