**Report 3 – 설명**

**20182931 신석경**

Report 3-1. 기본적인 포커 게임을 할 수 있는 프로그램

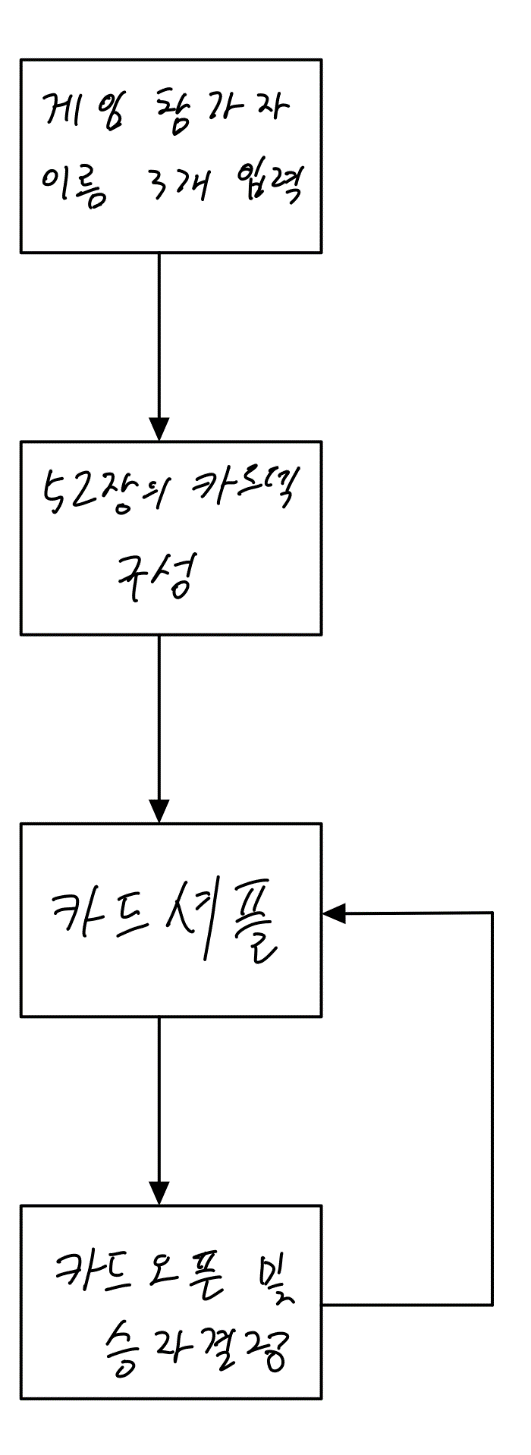
실행파일: pocker.c

이 프로그램은 반복적으로 카드를 shuffle하고 3명의 사람에게 분배하여 각자 가지고 있는 족보에 따라 게임의 승패를 결정하는 프로그램이다. 이 프로그램의 메인 흐름은

1번. 게임 참가자 이름 3개를 입력 받는다.

2번. 덱을 구성한다.

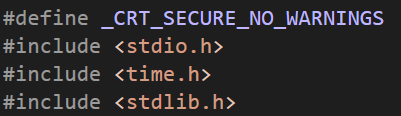
3번. 셔플하고 카드오픈하여 승자를 결정한다. 이후 입력에 따라 3번의 과정을 반복한다.

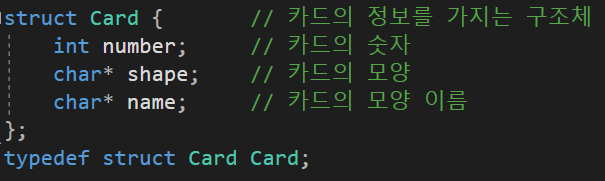
프로그램의 동작 흐름을 살펴보면 옆의 그림과 같다. 게임 참가자 이름을 3개 입력하고, 52장의 카드덱을 구성한다. 이후 카드를 섞고 오픈하고 승자를 결정한다. 이후 이 과정을 반복하는 과정이다.

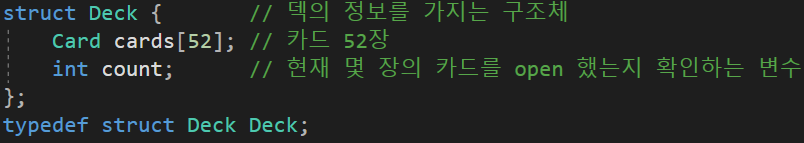
이 프로그램은 반복적 루프이기 때문에 while문을 사용하였다. while() 문 속에서 shuffle -> deal -> cardOpen 순으로 실행되고, 그 내부에서 다시 계속하는 옵션을 while문 속에서 입력 받는다. 이때 입력 오류에 대한 예외처리도 한다. 자세한 것은 코드를 보며 살펴보자.

또한, 3가지의 구조체를 사용한다. 먼저 Card를 나타내는 Card 구조체와 Deck을 나타내는 Deck 구조체를 선언하였다. 그리고 참가자에 대한 구조체인 Gamer 또한 정의하였다.

또한 1개의 열거형을 사용하였다. 덱을 구성할 때 편하도록 트럼프 카드의 모양인 SPADES, HEARTS, DIAMONDS, CLUBS에 해당하는 열거형을 선언하였다. 자세한 것은 코드를 살펴보자.

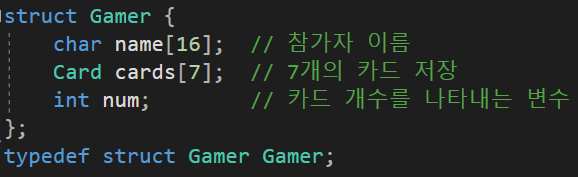
먼저, 필요한 헤더 파일을 include 하였다. time, stdlib은 난수 생성인 rand에서 사용하기 위하여 include 하였다.



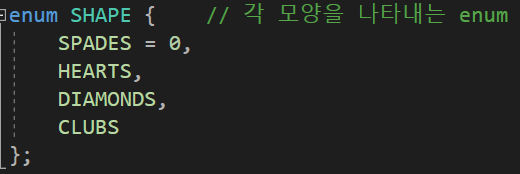


먼저, Card를 나타내는 구조체이다. 각각의 카드는 자신만의 숫자와 shape을 가지고, 그 shape의 이름을 가진다. 예를 들어 Spades A는 number = 1, shape = “♠”, name = “Spades”를 가진다.

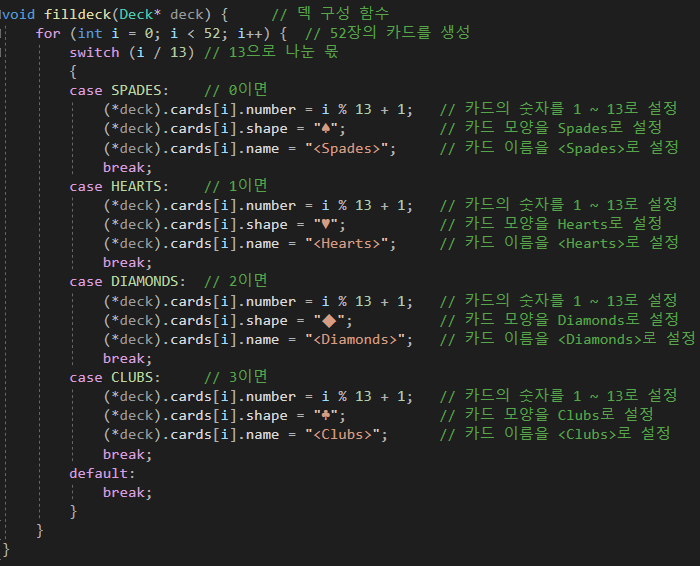
그 다음 Deck을 나타내는 구조체이다. Deck은 총 52의 길이를 가지는 Card배열을 가지고, 몇 개의 카드를 open 했는지 알기 위한 변수 count를 가지고 있다.



이제 Gamer를 나타내는 구조체이다. 각자의 이름을 가지고 char배열 name을 가지고 있고, 분배 받는 cards를 저장하는 7 길이의 Card 배열이 있다. 그리고 현재 가지고 있는 카드 개수를 저장하는 num int 변수를 가지고 있다.

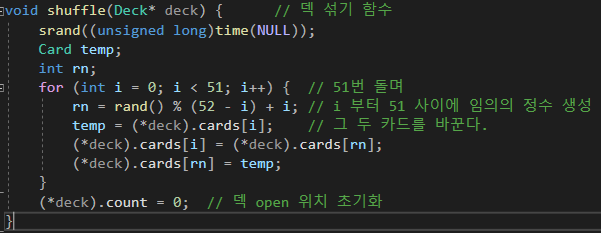
각각의 카드 모양에 대한 열거형이다.





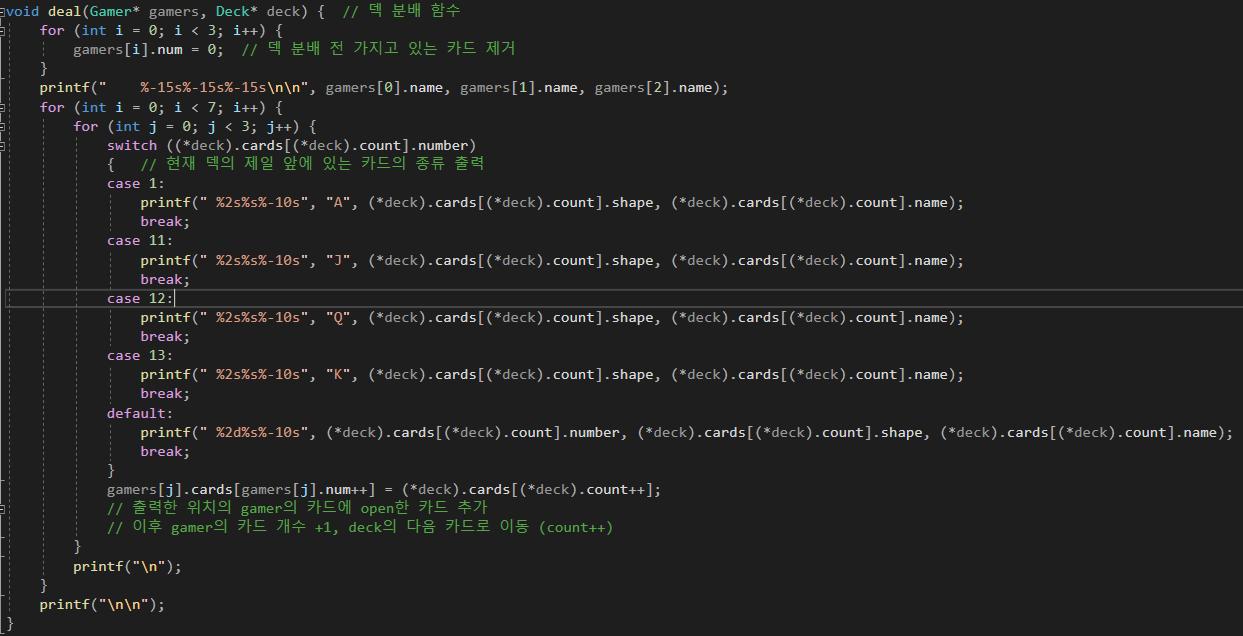
filldeck() 함수 정의 및 구현이다. 총 52장의 카드이기 때문에 for 문에서 52번 반복한다. 이후 switch 문으로 들어가는데, i를 13으로 나눈 몫에 따라 shape가 결정된다. 이때, i가 0 ~ 12 일 때는 i/13은 0이기 때문에 열거형인 SPADES에 걸리게 된다. 그러면 deck.cards배열의 i번째 요소의 Card의 number = i % 13 + 1을 하여 1 ~ 13에 해당하는 숫자가 배당되게 한다. 그리고 shape과 name을 설정해준다. 즉, 0 ~ 12 는 Spades 카드 1 ~ 13을 할당하고, 13 ~ 25 은 Hearts 카드 1 ~ 13 을 할당, 26 ~ 38는 Diamonds 카드 1 ~ 13, 39 ~ 51은 Clubs 카드 1 ~ 13을 할당한다.





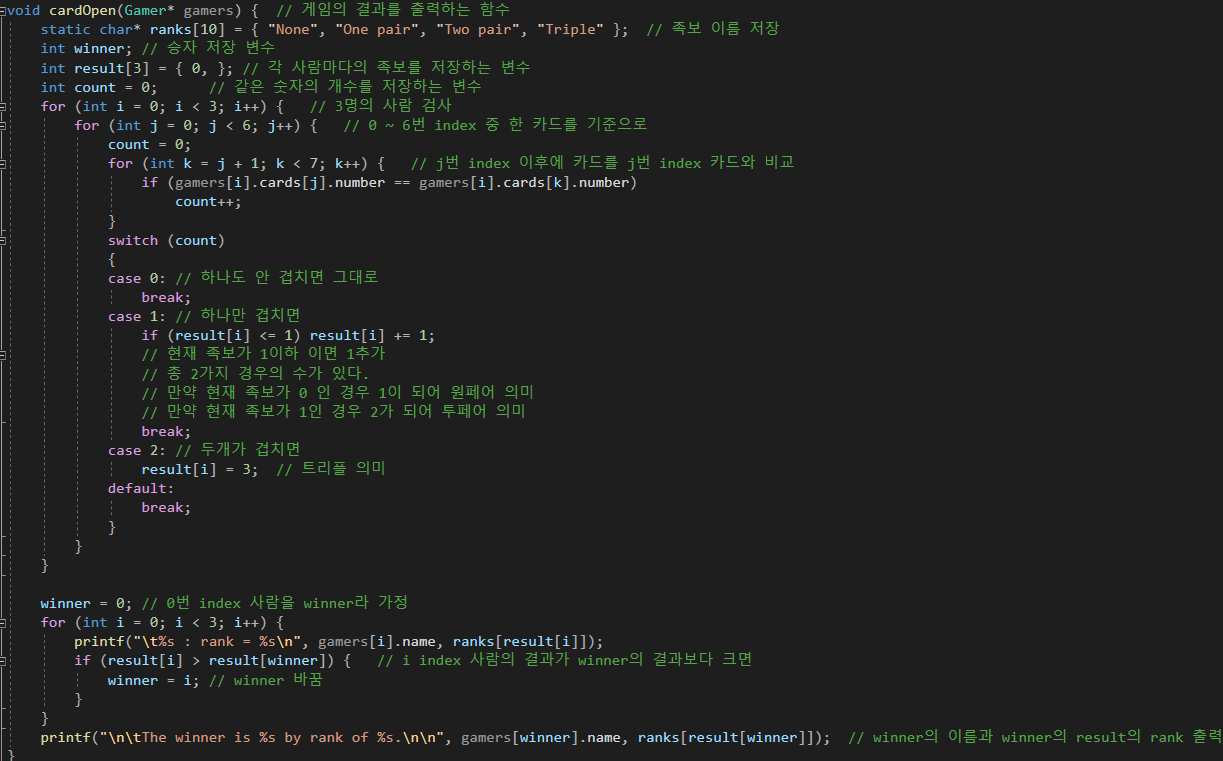
덱을 shuffle하기 위한 함수 shuffle()이다. 먼저 Card 구조체 변수 temp를 설정하고, 임의의 index를 저장하기 위한 변수 rn을 설정한다. 그리고 for 문을 통해 총 51번의 swap을 한다. 현재 index부터 51 사이의 임의의 정수를 생성한다. 그리고 그 index와 i index의 Card를 바꾼다. 모든 Card를 바꾸고, deck의 count를 0으로 초기화하여 한 장도 open되지 않은 상황을 가정한다.





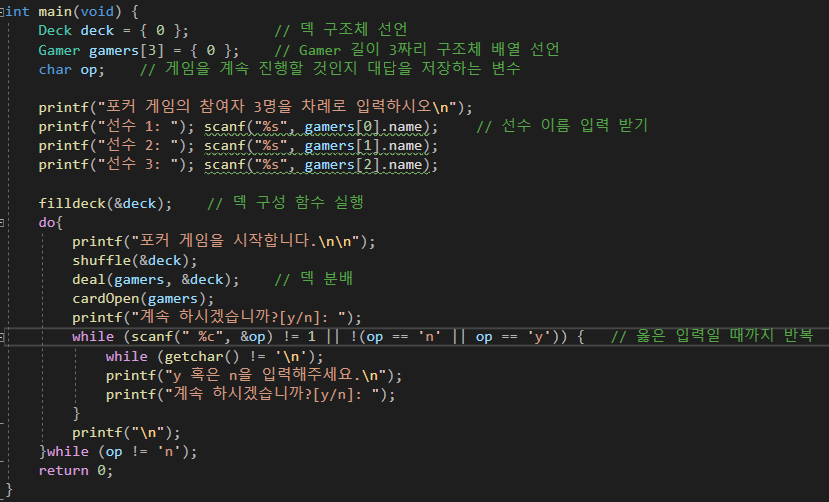
덱 분배 함수인 deal() 함수이다. 먼저 카드를 분배하기 전에 각 gamer의 카드 소유 개수를 0으로 초기화한다. 그리고 1명 당 7장 씩 총 21개의 카드를 분배한다. 이 함수에서는 분배하는 동시에 카드를 오픈한다. switch문은 현재 deck.count 위치의 cards 종류에 따라 자신의 숫자와 카드 모양, 카드 모양의 이름을 출력한다. 1인 경우는 “A”로 출력하고, 11, 12, 13은 각각 “J”, “Q” “K” 순서로 출력한다. 나머지의 경우는 그냥 자신의 숫자를 출력한다. 그리고 나눠줄 차례인 gamer에게 open한 cards를 할당한다. 이때, gamer의 num에 따라 gamers.cards 변수에 하나씩 deck.cards[deck.count]에 해당하는 카드가 들어가는 것이다. 그리고 num과 count를 1씩 증가시켜, 다음 번에 저장할 자리(num)과 다음 오픈할 카드 위치(count)를 update한다. 이것을 총 21번 반복하면 각각 3명의 사람에게 7장씩 카드가 분배되게 된다.





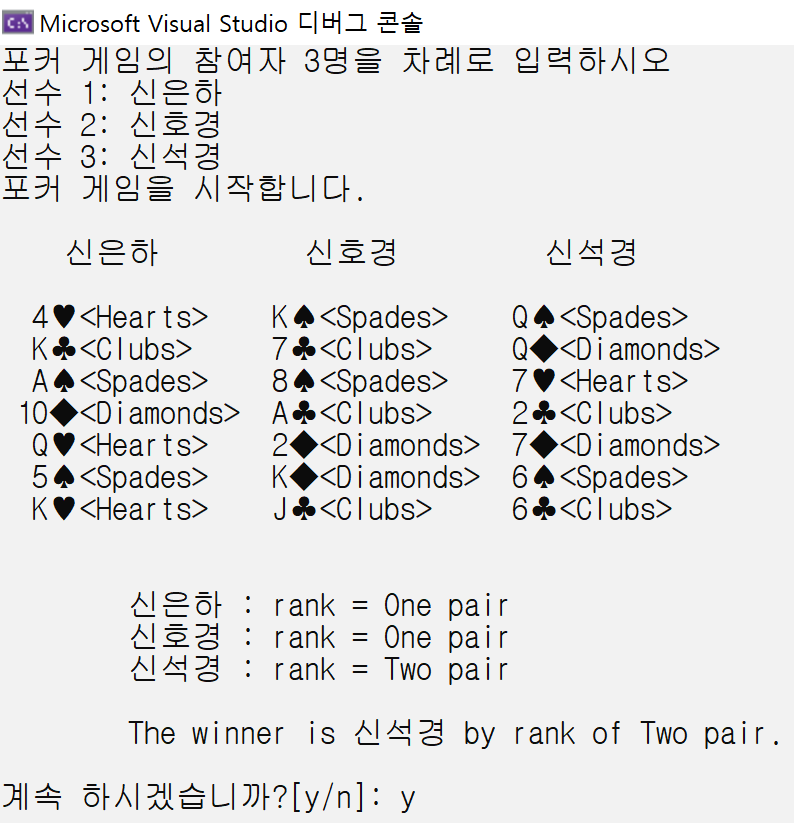
카드를 오픈하고 각 gamer가 가진 카드 종류에 따라 승자를 결정하는 cardOpen() 함수이다. 먼저 족보를 저장하는 변수 char\* ranks[10]을 선언하여 족보의 순서를 저장한다. 그리고 승자를 저장하는 변수 winner, 각 사람의 족보 rank를 저장하는 result 배열, 같은 숫자가 몇 개 나왔는지 판단하는 count 변수를 선언한다. 3명에 사람에 대해 0 ~ 6번의 index의 카드를 기준으로 그 index 뒤의 카드를 검사한다. 예를 들어 a의 카드가 {1, 5, 6, 8, 2, 6, 13} 인 경우, 1을 기준으로 {5, 6, 8, 2, 6, 13}을 검사하고 그 다음 5를 기준으로 {6, 8, 2, 6, 13}을 검사하는 것이다. 이것을 검사하며 현재 기준인 숫자와 몇 개의 숫자가 같은 숫자인지 검사하게 된다. 이것은 if 문을 통해 검사하였다. 하나를 기준으로 한 계산이 끝나면 그때의 count 값을 살펴본다. count가 0이라는 것은 하나도 겹치지 않았다는 것으로 족보에 아무 변화 없다. count가 1인 경우 기존의 족보가 1이하이면 족보의 rank를 1 증가 시킨다. 만약 기존 rank가 0인 경우 1되어 One pair 가 되는 것이고, 기존 rank가 1인 경우는 2가 되어 Two pair가 되는 것이다. 만약 기존 족보가 2인 경우는 똑 같은 Two pair이기 때문에 아무 변화 없다. 만약 count가 2이면, 자기 자신까지 총 3장의 카드가 똑같은 경우 이므로 족보의 rank를 3으로 바꿔 Triple을 의미한다. 이것을 총 3명에 대해 반복하면, result 변수에는 각 사람에 대한 족보 rank가 저장되어 있는다.

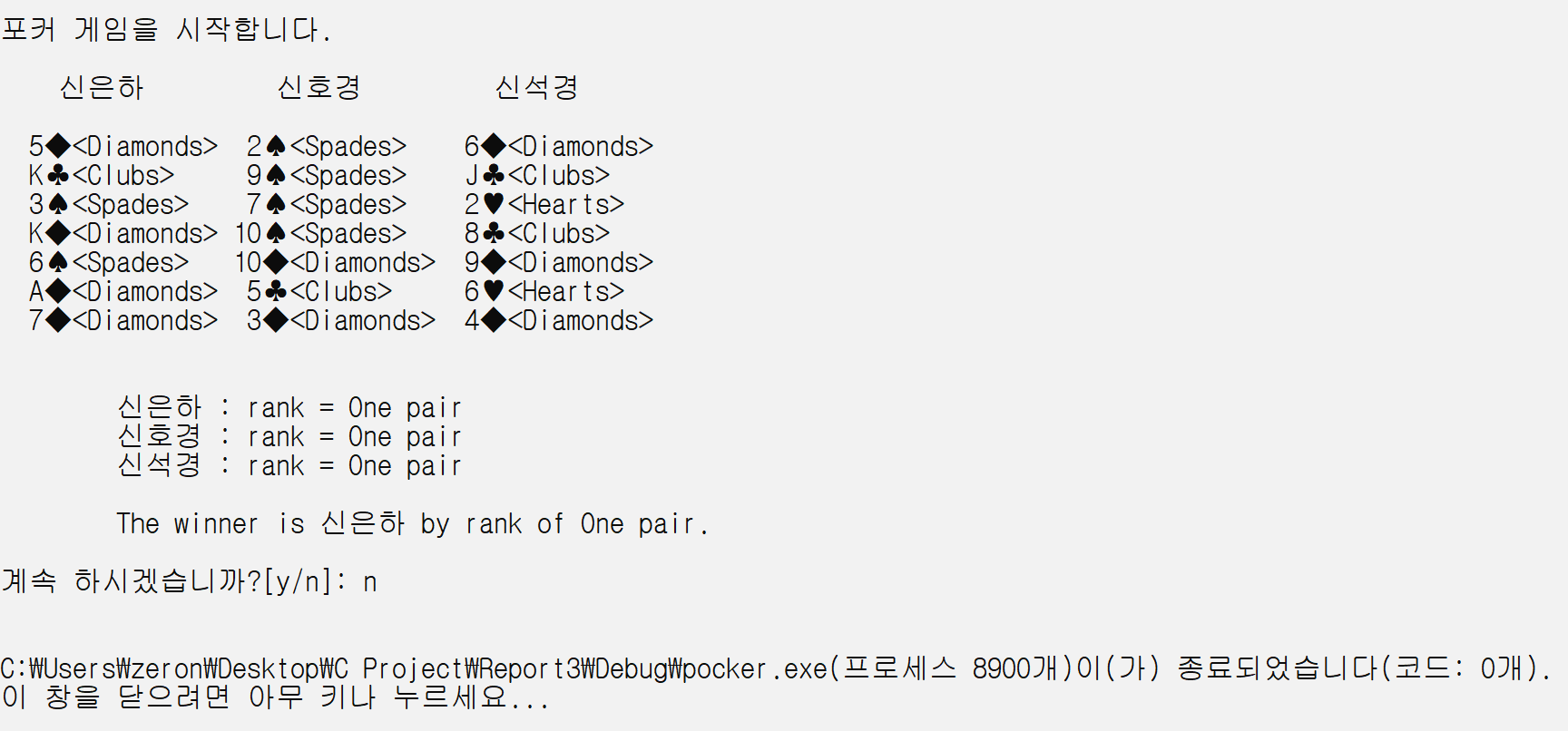
이후, 승자를 결정 짓기 위한 알고리즘을 시작한다. 먼저 winner를 0번 index 사람이라 가정하고각 사람의 rank를 출력하면서 winner의 rank보다 i번 index 사람의 rank가 더 크면 winner = i로 바꾼다. 이것을 모두 반복하면 최종 winner에 할당된 index가 최종 승자 사람이 되는 것이다. 이 것을 출력하면 한 게임이 끝난 것이다.



메인 함수이다. 먼저 Deck 구조체 변수인 deck과, Gamer 구조체 배열인 gamers[3]를 선언한다. 이후 3명의 이름을 입력 받아 각각 gamers 배열 속 구조체에 할당한다. 이후 filldeck(&deck)을 통해 deck의 덱을 구성하고, do while()문을 통해 셔플 -> 분배 -> 결과 출력의 과정을 반복한다. 이때, 중간 입력이 n이면 전체 게임을 종료하고 프로그램을 종료한다.

이제 각각의 상황에 대한 결과를 보겠다.





총 3번의 게임을 진행하였다. One pair, Two pair, Triple을 모두 잘 인식하는 것을 확인할 수 있다.

Report 3-2. 마방진을 만드는 프로그램

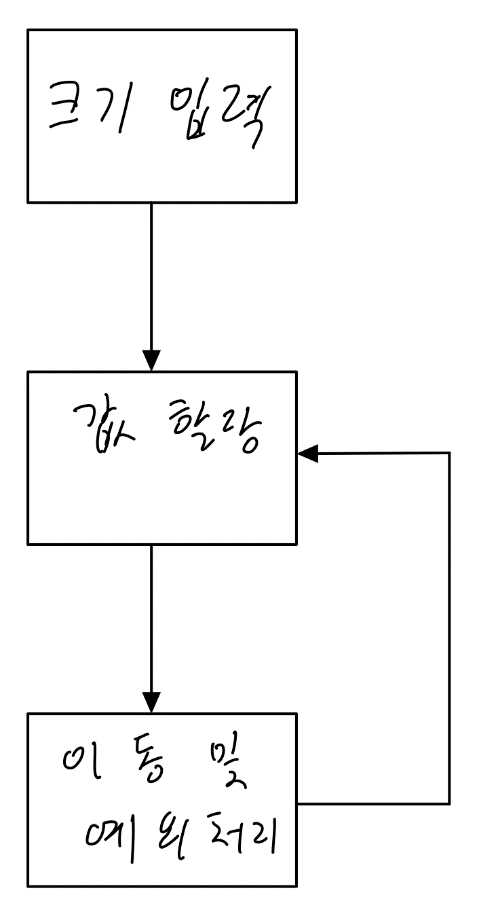
실행파일: magicsquare.c

이 프로그램은 주어진 크기의 정방 행렬에서 가로, 세로, 대각선 원소들의 합이 같도록 만드는 것이다. 이것은 반복적으로 이동 후 할당이라는 과정을 수행한다. 이 프로그램의 흐름은

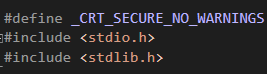
1번. 정방 행렬의 크기를 입력 받는다.

2번. 첫 행, 중앙 열에서 1을 시작으로 대각선으로 이동해가며 값을 할당한다.

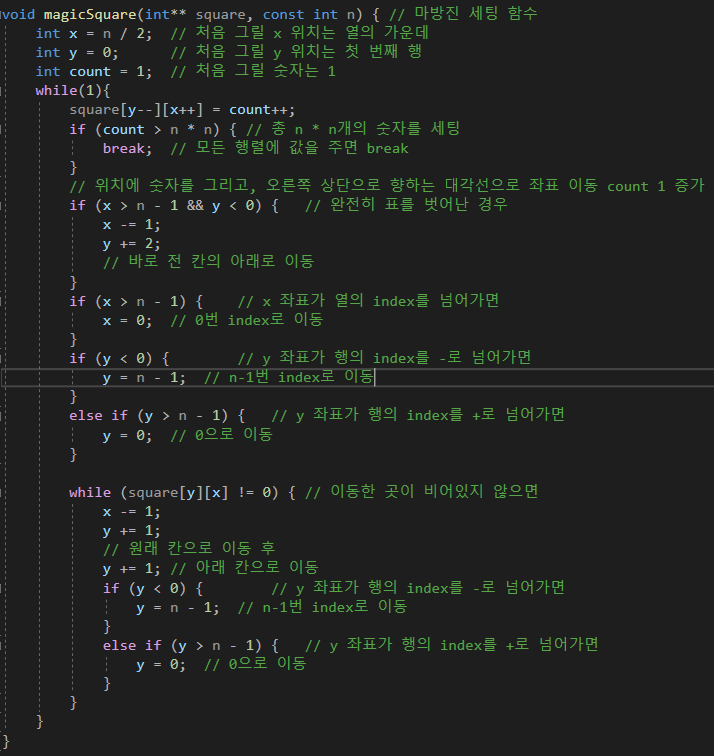
3번. 행 또는 열을 초과한 경우 반대편 행 또는 열로 이동하여 값을 할당한다. 만약 이미 값이 있는 경우 마지막으로 할당한 값 아래에 할당한다. 예외적으로 행과 열 모두 넘어간 경우 마지막으로 할당한 위치 바로 아래에 값을 할당한다.

 프로그램의 흐름 그림이다. 정방 행렬의 크기를 입력 받고, 먼저 첫 행, 중앙 열에 1을 할당한다. 그리고 그 다음 곳으로 이동하는데 이때, 예외가 되는 부분들에 대한 처리까지 하여 값을 할당할 수 있는 부분까지 이동을 한다. 그리고 다시 값을 할당하는 과정을 반복하므로 마방진을 완성할 수 있다.

이 프로그램은 일정 패턴의 반복 실행이기 때문에 while문을 사용한다. 그리고 예외처리를 위해 다양한 조건을 검사하여야 해서 if문을 사용한다. 이때, 모든 조건들이 독립적이기 때문에 if문도 분리하여 사용한다. 자세한 것은 코드를 보며 알아보자.

stdlib을 include하였다. 그 이유는 입력된 크기에 따라 동적으로 배열을 할당해야 하는데, 그때 stdlib에 정의된 calloc을 사용하기 때문이다.





마방진을 세팅하는 magicSqaure()함수이다. 그리는 위치를 정하기 위한 x, y변수를 설정한다. 이때, 첫 x 위치는 열의 중앙인 n/2, 첫 y의 위치는 첫 행의 위치인 0이다. 그리고 그릴 숫자를 정하기 위해 count = 1 를 선언하여 1부터 값을 할당하기 시작한다. 무한 루프를 돌며 값을 할당한다. square[y][x]에 count를 할당하고, y는 -1, x는 +1 하여 오른쪽 위 대각선 방향으로 좌표를 이동한다. 그리고 count는 1 증가시켜 다음에 할당할 값을 update한다. 이제 예외처리를 시작한다.

1. 이동한 x, y 좌표가 둘 다 값을 벗어난 경우

: x는 -1하여 그 전 x 값 위치로, y는 +2 하여 그 전 값 위치의 바로 아래로 이동하도록 한다.

1. 이동한 x, y 좌표에서 x가 열의 index를 넘어간 경우

: x = 0으로 하여 반대편 열로 이동한다.

1. 이동한 y 좌표가 행의 index를 -로 넘어가면, 즉 -1이 되면

: y = n-1로 하여 반대편 행으로 이동한다.

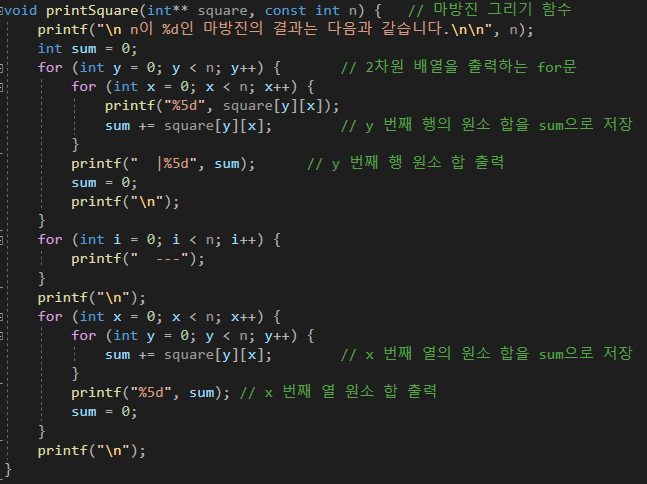
1. 만약 이동한 위치의 값이 비어 있지 않으면 (square[y][x] != 0)

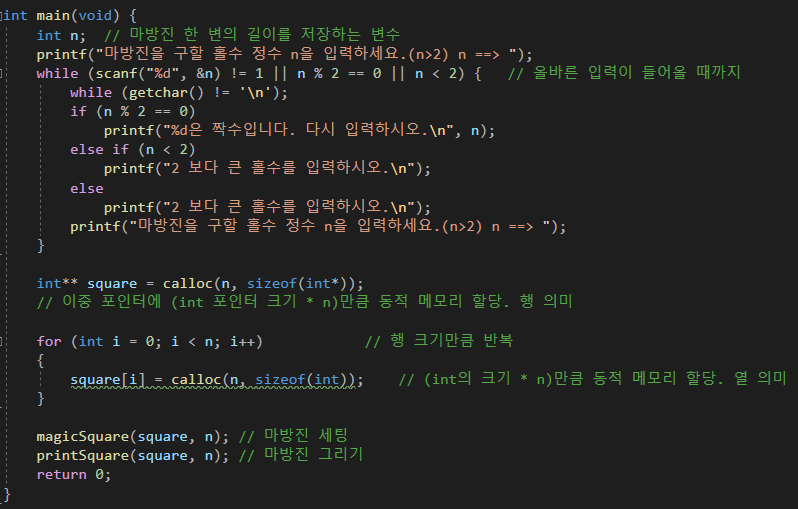
: 이 경우 추가 설명이 필요하다. 처음에 calloc으로 동적 할당하였기 때문에 값이 할당되기 전 배열의 값은 항상 0이다. 즉 0이 아니면 이전에 값이 할당된 적이 있다는 것이다. 이런 경우, 그 전에 할당된 곳으로 가서 (x -= 1, y += 1) 아래 칸으로 이동 후 (y+=1) 그 다음 y 값의 index 값에 따라 예외 처리를 다시 해준다. 즉, 이것을 while로 실행하므로 계속 할당된 곳을 하면 계속 비어 있는 곳을 찾아 내려가도록 하였다. 물론 올바르게 작동하면 이 과정은 한번이면 충분하지만 혹시를 대비하여 while로 하였다.

위의 과정을 반복하여 할당한 이후 count > n\*n이 되면 break하여 while을 빠져 나온다.

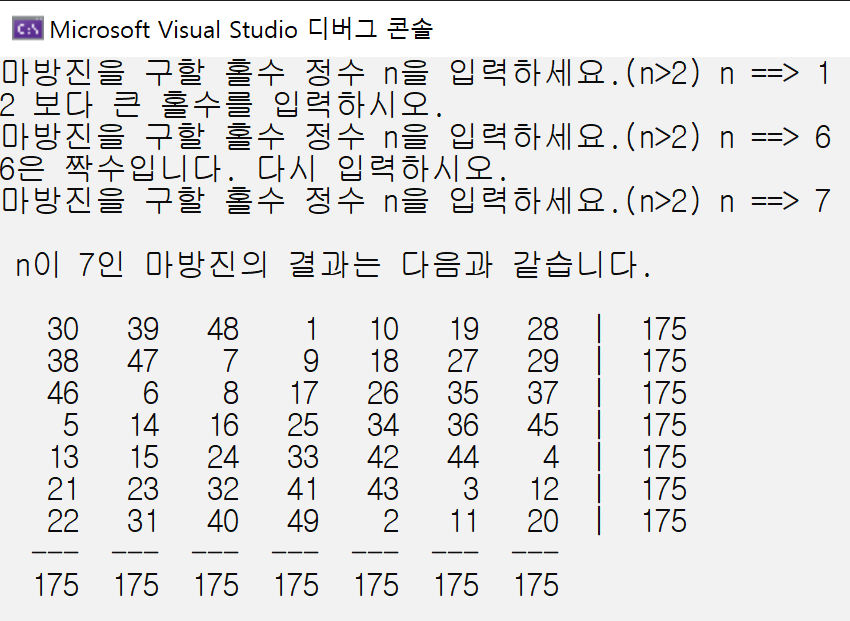
그러면 모든 2차원 배열에 값이 할당된다.

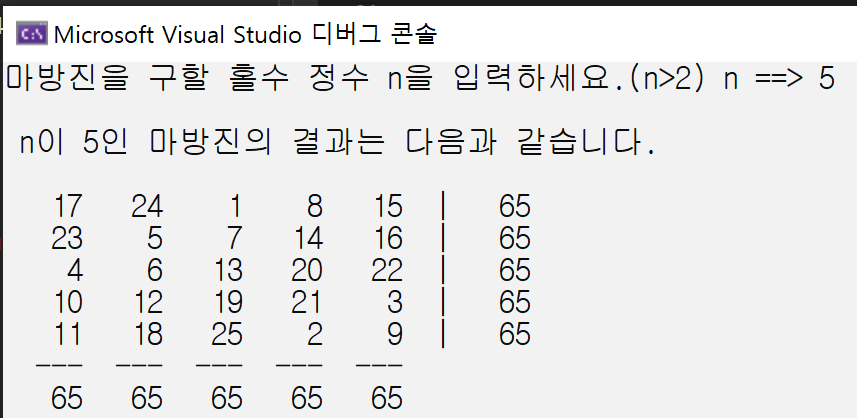


마방진을 그리고 각 열과 행의 원소의 합을 출력하는 printSquare() 함수이다. 먼저 2차원 배열을 출력하며 각 행의 원소 합을 sum으로 저장하고 한 행을 다 그리면 sum 값을 출력 후 다시 그 다음 행을 계산한다. 이렇게 모든 행을 다 그리면, 이제 열을 기준으로 sum만 계산하여 마지막 행 다음 줄에 각 열의 원소 합을 출력한다.



메인 함수이다. 먼저 마방진 한 변의 길이를 저장하는 변수 n을 설정한다. n이 정수이고, 2 초과의 홀수인 경우에만 입력으로 인식하고 아닌 경우에는 계속 다시 입력을 받는다. 이후 calloc을 통해 int\*\* square에 n개의 int 포인터를 할당하고, 이후 각 행마다 int 변수를 n개 할당하여 n \* n 2차원 배열을 할당한다. 이후 마방진을 세팅하고 마방진을 그려 결과를 확인한다.





프로그램 실행 결과이다. 2보다 작은 수 혹은 짝수가 들어오면 다시 입력 받는 것을 확인할 수 있다. 그리고 올바른 홀수가 입력되면 모든 가로, 세로의 원소 합이 같도록 배열이 할당된 것도 확인할 수 있다.

Report 3-3. 구조체로 복소수를 정의하고 복소수의 더하기, 빼기, 곱하기, 절댓값을 구하는 프로그램

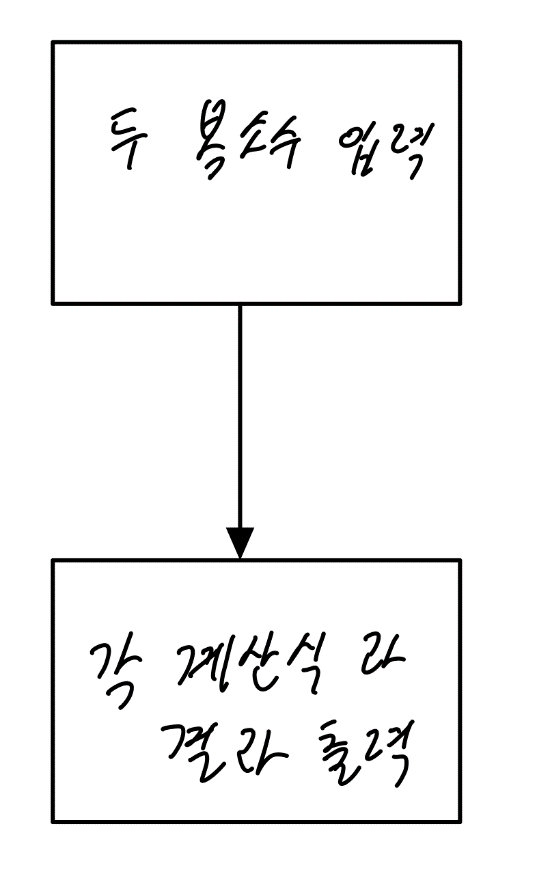
실행파일: complexnum.c

이 프로그램은 두 개의 복소수를 입력 받아 더하기, 빼기, 곱하기, 절댓값을 출력하는 프로그램이다. 단순 계산과 입력이기 때문에 입력 받은 값으로 구조체를 초기화하고 연산을 진행하면 된다. 이 프로그램의 흐름은

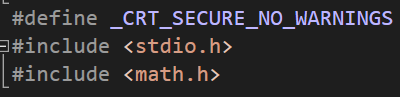
1번. 첫 번째 복소수에 해당하는 실수 2개를 입력 받는다.

2번. 두 번째 복소수에 해당하는 실수 2개를 입력 받는다.

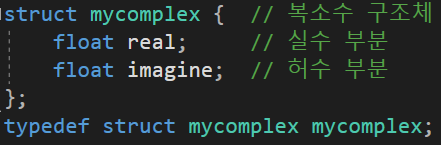
3번. 각각의 연산 결과를 식과 함께 출력한다.

 프로그램의 흐름을 그림으로 나타낸 것이다. 입력 받은 2 실수를 기본으로 하나의 복소수를 정의한다. 이 과정을 2번 반복하여 2개의 복소수를 입력 받고, 그것을 바탕으로 각 계산식과 계산 결과를 출력한다.

이 프로그램에서는 복소수를 정의하기 때문에 복소수를 정의할 수 있는 구조체를 정의한다. 그리고 각 계산을 할 수 있는 함수도 따로 구현하였다.

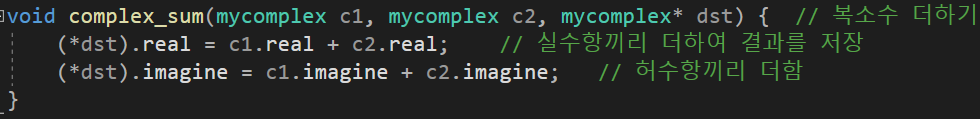


필요한 헤더를 include 한다. 이때, math.h는 sqrt를 하기 위하여 선언하였다. 이때, math에 complex라는 이름이 선언되어 있기 때문에 우리 구조체는 mycomplex로 선언하겠다.



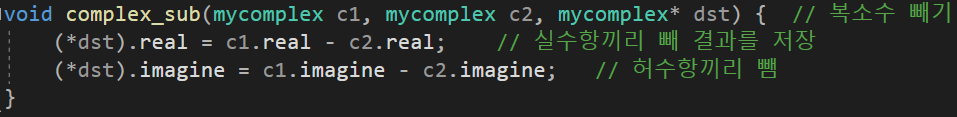
구조체 정의 부분이다. 복소수는 각각 real 파트와 imagine 파트로 구성되어 있기 때문에실수형 변수 real과 imagine 을 가지고 있다.





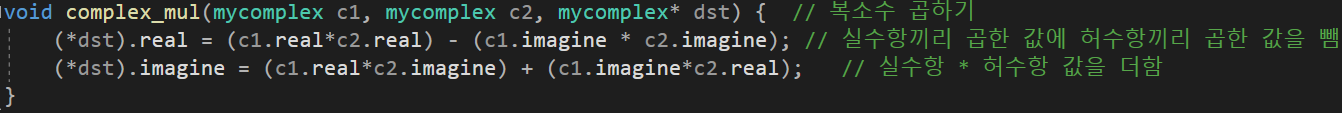
복소수의 덧셈을 수행하는 complex\_sum() 함수이다. 두 복소수를 입력 받아 dst 구조체에 값을 할당한다. 구조체는 복사하여 파라미터로 전달되기 때문에 외부 구조체의 값을 바꾸려면 포인터로 접근해야 한다. 두 복소수의 real 파트를 더해 dst의 real 파트로 할당하고, imagine 파트를 더하여 dst.imagine 파트에 할당한다.





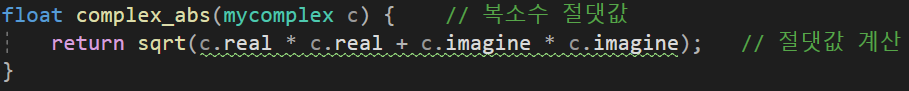
복소수의 뺄셈을 수행하는 complex\_sub() 함수이다. 이때도 위의 sum 함수와 유사하게 real 파트끼리 빼서 할당하고 imagine 파트끼리 빼서 할당한다.





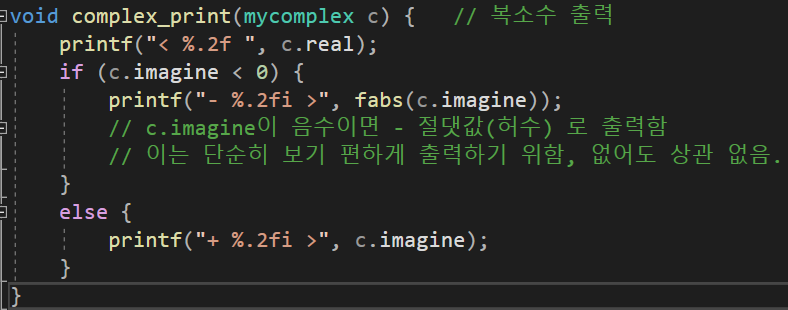
곱셈인 complex\_mul() 함수이다. 이때는 a+bi, c+di라는 두 복소수가 있을 때, real 파트는 ac – bd로 계산하고, imagine 파트는 ad + bc로 계산하기 때문에 위와 같은 계산을 수행한다.



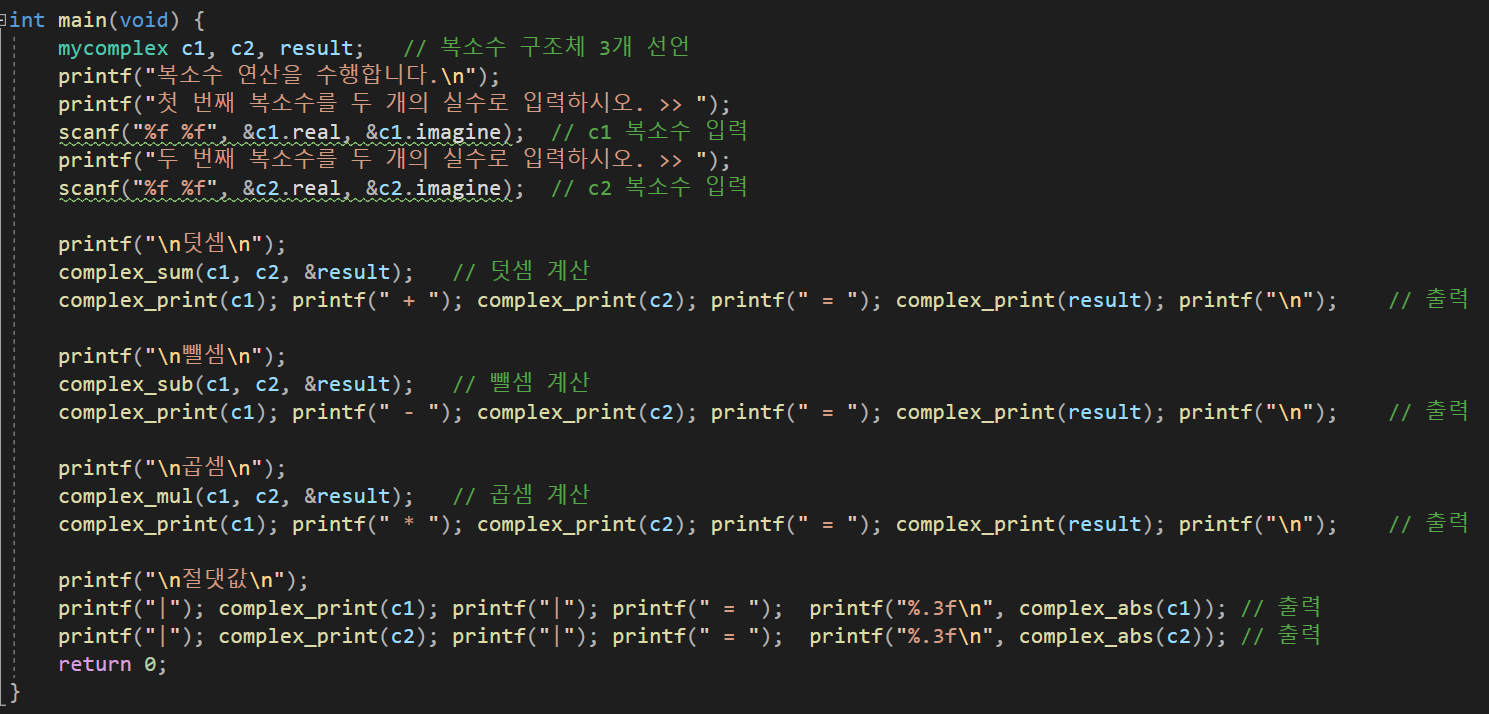


복소수의 절댓값을 구하는 함수 complex\_abs()이다. real 부분의 제곱과 imagine 부분의 제곱 합의 sqrt()취해 절댓값을 계산하여 반환한다.



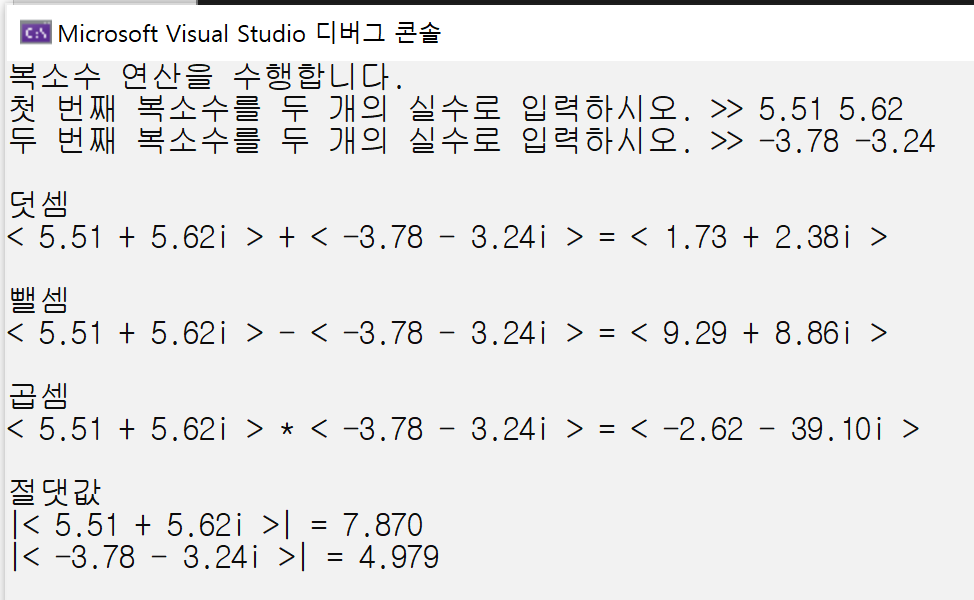


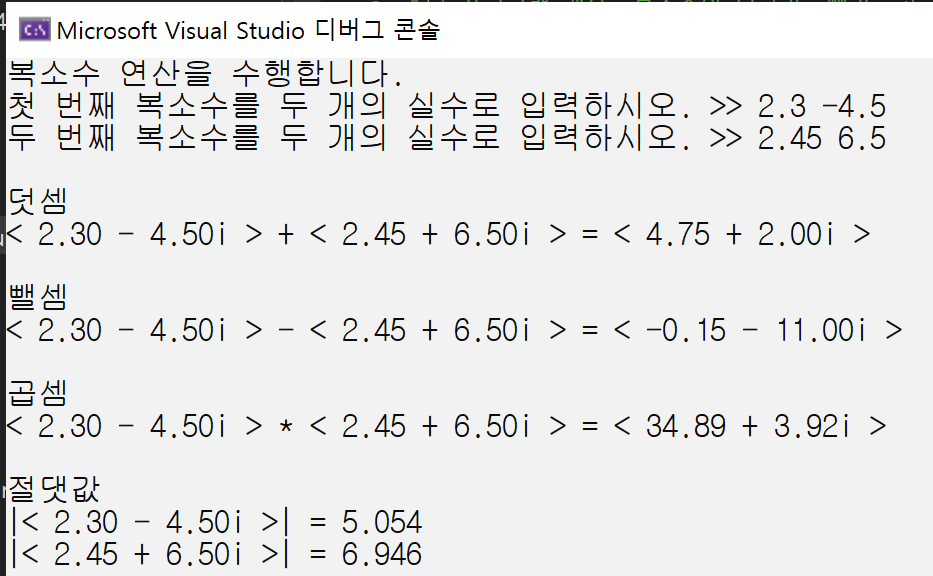
복소수를 출력하는 complex\_print() 함수이다. < (real) + (imagine)I >로 출력하기 위함이다. 이제 부호에 따라 (real) 과 (imagine) 사이의 부호를 결정하기 위해 imagine의 부호를 따져 – 혹은 +로 결정한다. 그리고 그 뒤의 출력은 이미 앞에서 부호를 처리하였기 때문에 fabs()를 사용하여 실수 절댓값을 계산한다. 이 함수는 단지 출력을 위한 함수이기 때문에 전체적인 문제 해결에는 별 상관없다.



메인 함수이다. 복소수 변수 c1, c2, result를 선언한다. 먼저 실수 2개를 입력 받아 처음에는 c1을 초기화 하고, 그 다음에는 c2를 초기화 한다. 이후 각각의 함수를 통해 result에 계산 결과를 저장하고 출력을 한다.

그 결과를 살펴보자





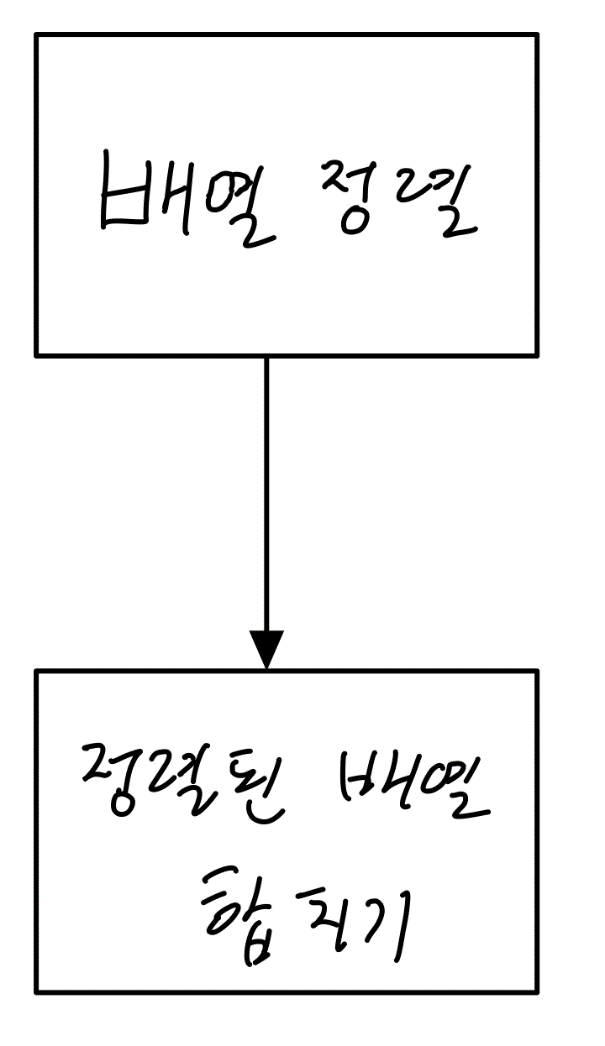
Report 3-4. 주어진 배열을 오름차순으로 각각 정렬한 이후에 정렬된 리스트를 배열 c에 통합 저장하는 프로그램

실행파일: mergesort.c

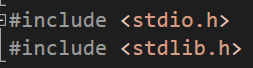
이 프로그램은 주어진 배열을 가지고 먼저 오름차순으로 정렬하고 merge하여 새로운 배열에 저장하는 프로그램이다. 이 프로그램의 흐름은

1번. 배열 a, b를 정렬한다.

2번. 정렬된 a, b를 합쳐 새롭게 정렬된 배열 c를 만든다.

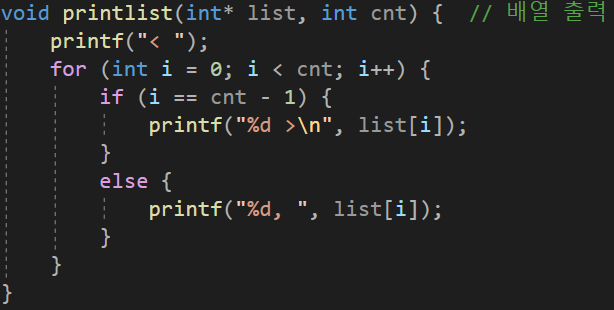
프로그램의 흐름을 그림으로 나타낸 것이다. 매우 간단하게 배열을 정렬한 이후, 그것들을 합쳐서 새로운 배열을 만들면 된다.

이것을 구현하기 위해 먼저 오름차순 정렬을 구현해야 한다. 이때는 버블 정렬을 구현하도록 하였다. 그리고 합병 정렬에서 merge 부분을 구현하였다. 요구사항으로 총 3개의 함수를 구현해야 하는데, 배열을 출력, 배열을 정렬, 두 배열을 합쳐서 정렬 이 3가지 역할을 하여야 한다. 이 과정은 아래 코드를 보며 설명하겠다.



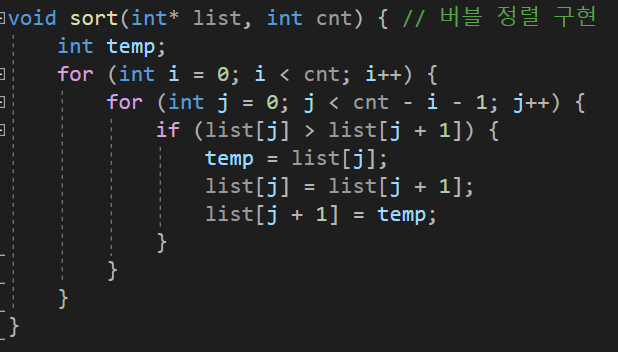
먼저, 필요한 헤더를 include 하였다. 이때 stdlib은 배열 c를 선언할 때, 기존에 선언되어 있는 a와 b의 길이를 구하여 동적으로 할당하기 위해 calloc을 사용하기 때문이다.





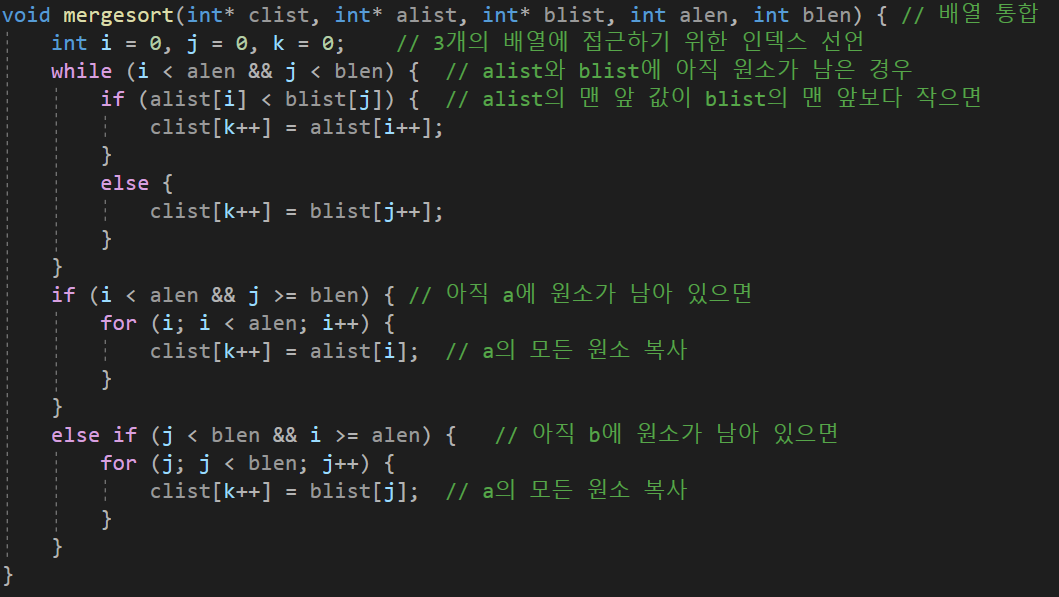
배열을 출력하는 함수이다. 배열의 포인터와 배열의 길이를 가져와 출력을 실행한다. 배열은 < > 이 사이에 출력하고 i가 cnt – 1, 즉 마지막 index 원소를 출력할 때만 반점을 없애고 출력하고 줄바꿈을 실행한다.



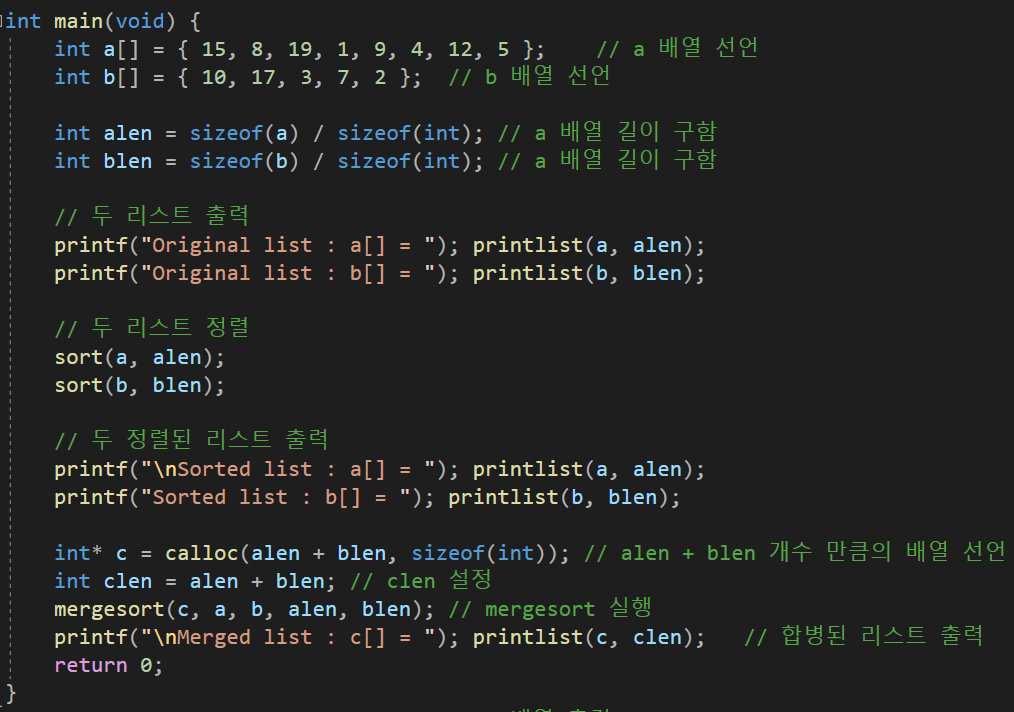


버블 정렬을 구현한 코드이다. 버블 정렬은 인접한 두 원소를 검사하여 큰 원소를 뒤로 보내는 정렬이다. 한번 회전을 수행할 때마다 가장 큰 원소가 맨 뒤로 가게 된다. 즉 i = 0인 경우 가장 큰 원소의 위치가 cnt -1 위치가 된다. j가 cnt – i – 1까지 검사하는 이유는 그 다음 코드에서 다음 원소가 지금 내 원소보다 작으면 지금 내 원소를 +1 하여서 저장하기 때문이다. 총 우리는 cnt 개수 만큼의 원소를 정렬해야 하기 때문에 cnt번 회전 하고 한번 수행마다 맨 뒤에 제일 큰 수가 결정되기 때문에 1번 회전할 때마다 마지막 원소의 index가 감소하게 된다.





두 정렬된 리스트를 합병하는 함수이다. 먼저 alist와 blist에 하나씩 접근하기 위한 변수 i, j를 선언하고 clist에 저장할 index를 가지는 변수 k를 선언한다. 이제 i 혹은 j 중에 하나가 정렬된 리스트의 index를 넘어갈 때까지 비교하여 더 작은 수를 clist에 저장한다. 이후 남아있는 alist 혹은 blist를 모두 clist에 넣으므로 합병 정렬을 완료한다.



메인 함수이다. 먼저 a, b를 선언한다. 그리고 a와 b의 길이를 저장하는 alen, blen을 계산하여 저장한다. 그리고 먼저 기본 두 리스트를 출력한다. 그리고 두 리스트를 정렬한 이후 출력한다. 그리고 c를 alen + blen의 길이만큼 동적 할당하고 clen도 선언한다. 이후 합병 정렬을 완료한 이후 그 결과를 출력한다.

