

# 12. 빙고게임 만들기

2022 Fall

# Objectives

- 지금까지 배운 문법 내용을 활용하여 Bingo 프로그램을 작성
  - 2차원 배열 문법 활용
  - 임의로 숫자를 배치하는 알고리즘 구현
  - Bingo를 자동으로 인식하는 알고리즘 구현

# BINGO 게임

- 우선 한 게임 해보죠.
- bingo.exe 실행

```
C:\Users\WSM-PC\Documents\code\bingo\Project1.exe
=====
*****
***** BINGO GAME *****
*****
=====

=====
6      24      5      2      3
25     21      7      12     16
20     11      18     15     17
19      8      22     13     14
10     23      1      9      4
=====

No. of completed lines : 0
select a number :
```

```
C:\Users\WSM-PC\Documents\code\bingo\Project1.exe
=====
*****
***** CONGRATULATION!!!! *****
***** *#&$*#^&*$&^@($!@(* BINGO!!!! #)!@*(#)*$(*#)*@*( *****
***** YOU WIN!!!! *****
*****
=====

=====
X      X      X      X      X
13     20     18     X      X
24      5      X     25     7
1       X     10     14     15
X      11     12      8     19
=====

-----
Process exited after 14.6 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

# BINGO 게임

- 게임 시작과 함께 bingo 보드 생성
- 숫자를 입력하여 완성된 줄 수를 자동으로 계산
- 지정된 줄 수만큼 완성되면 게임 종료
- 매크로 사용
  - N\_SIZE : 가로 혹은 세로 크기
  - N\_BINGO : 이기기 위해 완성해야 할 줄 수

# 실습 1 : 구현 시작

- project 및 main.c 생성
- rand() 및 srand() 함수 사용을 위한 header include
  - #include <stdlib.h>
  - #include <time.h>
- main() 함수 윗부분에 rand 초기화 함수 호출
  - srand( unsigned(time(NULL) );

# 실습 1 : 구현 시작

- 게임 시작 및 종료를 알리는 출력 꾸미기
- 각자 개성에 맞게 (똑같이 안해도 됩니다.)

```
122 int main(int argc, char *argv[]) {  
123  
124     srand((unsigned)time(NULL));  
125  
126     printf("=====\n");  
127     printf("*****\n");  
128     printf("                BINGO GAME                \n");  
129     printf("*****\n");  
130     printf("=====\n\n\n");  
131  
132  
133     printf("\n\n\n\n\n\n\n\n\n\n");  
134     printf("=====\n");  
135     printf("*****\n");  
136     printf("                CONGRATULATION!!!!                \n");  
137     printf("#&$*#^&*$&^@($!@(* BINGO!!!! #)!@*(#)*$($*)@*(\n");  
138     printf("                YOU WIN!!!!                \n");  
139     printf("*****\n");  
140     printf("=====\n\n\n");  
141  
142     return 0;  
143 }  
144
```

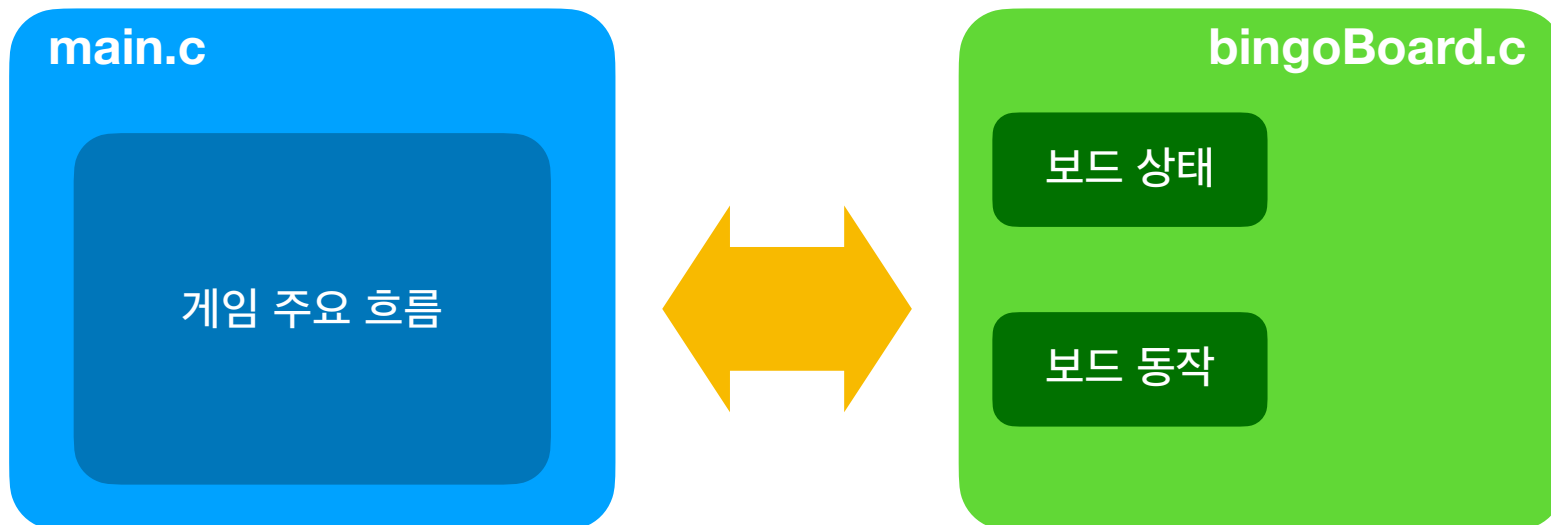
# 실습 2 : 전체 코드 설계

- 전체 동작 과정에 대해 생각해 보기
- main() 함수 수준에서의 큰 흐름 구상
- 코드에 주석으로 흐름을 표기해보기

```
int main(int argc, char *argv[]) {  
  
    srand((unsigned)time(NULL));  
  
    printf("=====\n");  
    printf("*****\n");  
    printf("                BINGO GAME                \n");  
    printf("*****\n");  
    printf("=====\n\n");  
  
    //generate numbers  
    //initialize bingo boards  
    //while (game is not end)  
    /*  
    {  
        //bingo board printing  
        //print no. of completed lines  
        //select a proper number  
        //put the number on the board  
    }  
    */  
  
    printf("\n\n\n\n\n\n\n\n\n\n");  
    printf("=====\n");  
    printf("*****\n");  
    printf("                CONGRATULATION!!!!                \n");  
    printf("##&$*#^&$*^@($!@(* BINGO!!!! #)!@*(#)%$*(#*)@*(\n");  
    printf("                YOU WIN!!!!                \n");  
    printf("*****\n");  
    printf("=====\n\n");  
  
    bingo_printBoard();  
  
    return 0;  
}
```

# 실습 2 : 전체 코드 설계

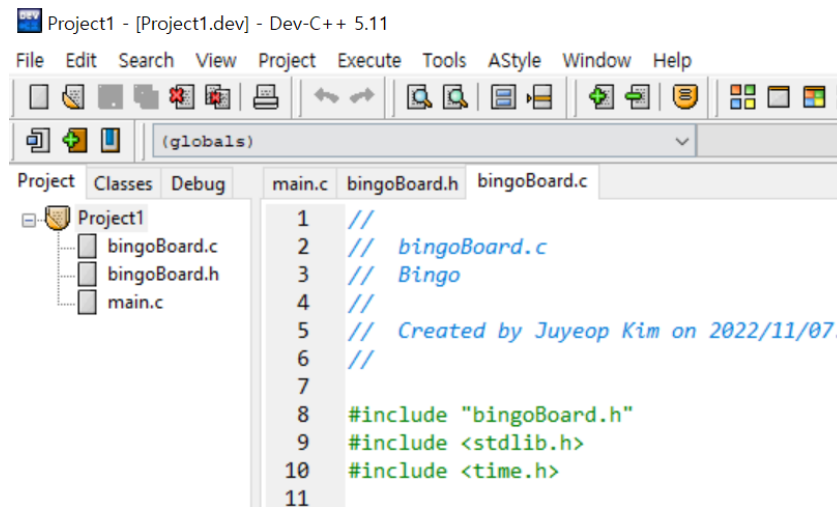
- 소스코드 구조 잡기
  - main.c : 게임의 주요 흐름
  - bingoBoard.c : 보드와 관련된 단편적인 동작 수행
    - 보드의 상태 및 함수들의 모음





# 실습 2 : 전체 코드 설계

- bingoBoard.c 및 bingoBoard.h 두개 파일 추가
- project 우클릭 및 New File 선택
- 새로 추가된 파일들이 project에 포함되어 있어야 함



# 실습 2 : 전체 코드 설계

- 각 동작 별로 적절한 함수의 prototype 만들기
  - 보드 초기화 -> bingo\_init()
  - 보드 출력 -> bingo\_printBoard()
  - 완성된 줄 수 계산 -> bingo\_countCompletedLine()
  - 번호 선택 -> get\_number()
  - 보드에 번호 입력 -> bingo\_inputNum()

## main.c

```
48  
49 int get_number(void)  
50 {
```

## bingoBoard.h

```
void bingo_init(void);  
void bingo_printBoard(void);  
void bingo_inputNum(int sel);  
int bingo_countCompletedLine(void);
```

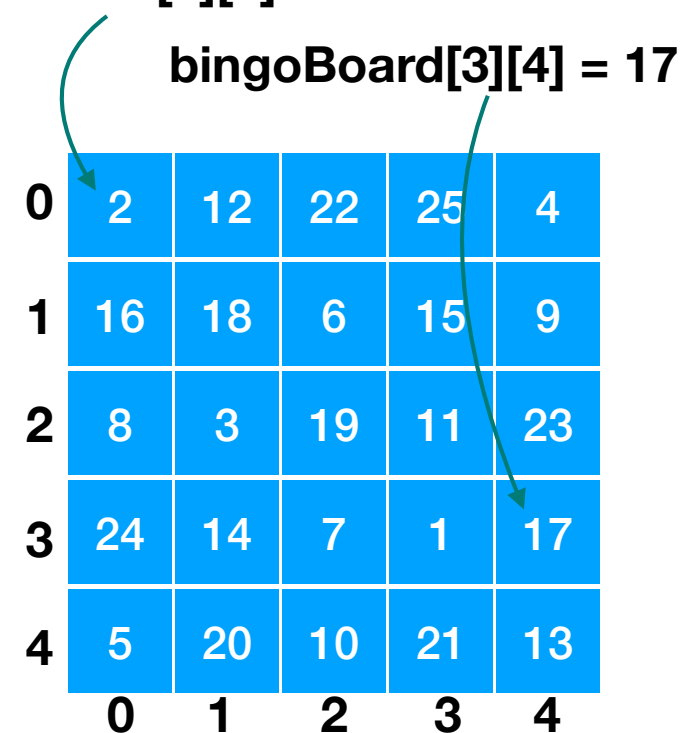
## bingoBoard.c

# 실습 3 : BINGO 보드 모델링

- bingoBoard (2차원 배열)
  - BINGO 보드의 모델링
    - N\_SIZE x N\_SIZE 크기
    - 각 변수는 해당 칸의 숫자를 의미

bingoBoard[0][0] = 2

bingoBoard[3][4] = 17

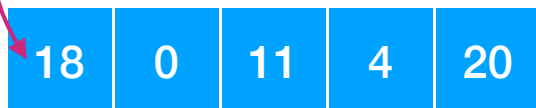


0	2	12	22	25	4
1	16	18	6	15	9
2	8	3	19	11	23
3	24	14	7	1	17
4	5	20	10	21	13
	0	1	2	3	4

# 실습 3 : BINGO 보드 모델링

- numberStatus : 1차원 배열
  - 특정 번호가 있는 위치
  - 번호로부터 배열의 index 파악시 활용

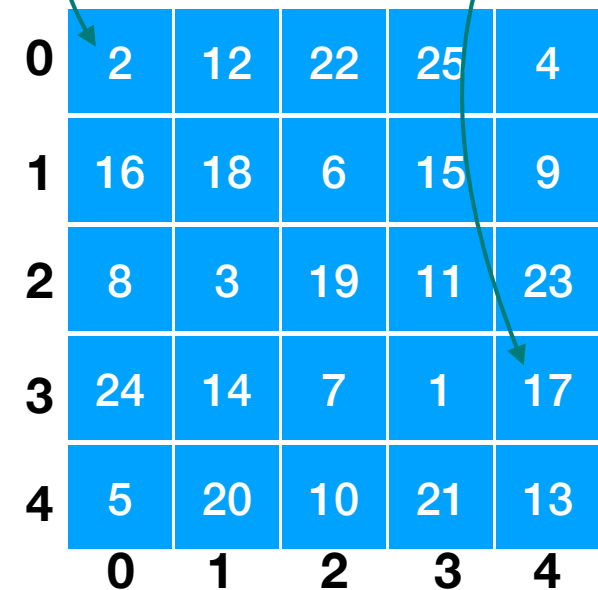
numberStatus[0] = 18 (3x5 + 3)  
(1 숫자의 위치 : bingoBoard[3][3])



18	0	11	4	20
----	---	----	---	----

bingoBoard[0][0] = 2

bingoBoard[3][4] = 17



0	2	12	22	25	4
1	16	18	6	15	9
2	8	3	19	11	23
3	24	14	7	1	17
4	5	20	10	21	13
	0	1	2	3	4

# 실습 3 : BINGO 보드 모델링

- bingoBoard.c 내의 전역변수로 지정
- bingo\_printBoard 함수 구현
  - 2차원 배열의 숫자를 격자 형태로 출력
  - 2중 반복문 활용

```
#include "bingoBoard.h"
#include <stdlib.h>
#include <time.h>
```

```
static int bingoBoard[N_SIZE][N_SIZE];
static int numberStatus[N_SIZE*N_SIZE];
```

```
void bingo_printBoard(void)
{
    int i, j;

    printf("=====\n");
    for (i=0; i<N_SIZE; i++) {
        for (j=0; j<N_SIZE; j++) {
            if (bingoBoard[i][j] > 0)
                printf("%i\t", bingoBoard[i][j]);
            else
                printf("X\t");
        }
        printf("\n");
    }
    printf("=====\n\n");
}
```

# 실습 3 : BINGO 보드 모델링

- 테스트용 bingo\_init 함수 생성
  - 1 ~ 25까지 숫자를 순서대로 넣음
  - numberStatus는 아래 식을 이용해서 값을 초기화

```
void bingo_init(void)
{
    int i, j, cnt=1;
    for (i=0; i<N_SIZE; i++)
        for (j=0; j<N_SIZE; j++)
        {
            bingoBoard[i][j] = cnt;
            numberStatus[cnt-1] = N_SIZE*i+j;
            cnt++;
        }
}
```

# 실습 3 : BINGO 보드 모델링

- bingo\_inputNum 함수 구현

- 처리할 숫자를 입력 받음

```
void bingo_inputNum(int sel)
{
```

- 입력된 숫자가 보드 어디에 있는지 파악

- 행/열 index를 numStatus[sel-1] 값으로부터 계산

- bingoBoard 및 numberStatus 배열요소 값을 바꿈

```
행 index = numberStatus[sel-1]/N_SIZE;
열 index = numberStatus[sel-1]%N_SIZE;
```

# 실습 3 : BINGO 보드 모델링

- 구현에 대한 검증
  - 초기화 -> 보드 출력 -> 값 임의로 넣기 -> 보드 출력

```
//generate numbers  
bingo_init();  
bingo_printBoard();  
bingo_inputNum(5);  
bingo_printBoard();  
bingo_inputNum(12);
```

```
=====
1   2   3   4   5
6   7   8   9  10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
=====
```

```
=====
1   2   3   4   X
6   7   8   9  10
11  X  13  14  15
16  17  18  19  20
21  22  23  24  25
=====
```

```
=====
1   2   3   4   X
6   7   8   9  10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
=====
```



# 실습 4 : 입력 처리

- get\_number 함수 구현
- printf / scanf로 입력을 받고 반환
- 제대로 된 입력인지 확인

```
do {  
    안내문구 출력;  
    번호 입력 받음;  
    표준입력 스트림 비우기;  
  
    if (입력 번호가 잘못되었거나 이미 나온 번호이면)  
        잘못되었다고 문구 출력;  
  
} while (제대로된 입력이 아님);
```

```
int get_number(void)  
{  
    int selNum=0;  
  
    do {  
        printf("select a number : ");  
        scanf("%d", &selNum);  
        fflush(stdin);  
  
        if (bingo_checkNum(selNum) == BINGO_NUMSTATUS_ABSENT)  
        {  
            printf("%i is not on the board! select other one.\n", selNum);  
        }  
    } while(selNum < 1 || selNum > N_SIZE*N_SIZE || bingo_checkNum(selNum) == BINGO_NUMSTATUS_ABSENT );  
  
    return selNum;  
}
```

# 실습 4 : 입력 처리

- bingo\_checkNum() 함수 구현
  - 현재 보드 상태에서 입력 번호가 적절한지 반환
- main() 함수에서 get\_number 함수를 활용하여 정상동작하는지 확인

```
#define BINGO_NUMSTATUS_ABSENT    -1
#define BINGO_NUMSTATUS_PRESENT   0

int bingo_checkNum(int selNum)
{
```

# 실습 5 : 상태 처리

- bingo\_countCompletedLine() 함수 구현

- 각 행, 열, 대각선 줄이 완성되었는지 확인

- 완성된 줄 수를 세고 반환

- flag 변수 (checkBingo) 활용

- 특정 줄의 배열 요소 중 하나라도 X가 아니면 flag 값 변경

```
int bingo_countCompletedLine(void)
{
    int i, j;
    int cnt=0;
    int checkBingo;
```

```
    //check row
    for (i=0;i<N_SIZE;i++){
        checkBingo = 1;
        for (j=0;j<N_SIZE;j++){
            if (bingoBoard[i][j] > 0){
                checkBingo=0;
                break;
            }
        }
        if (checkBingo == 1){
            cnt++;
        }
    }
}
```

# 실습 5 : 상태 처리

- check\_gameEnd() 함수 구현
- bingo\_countCompletedLine 함수 활용
- 완성된 줄수가 N\_BINGO 이상이면 게임 끝을 알리는 값을 반환

```
int check_gameEnd(void){  
    int res = BINGO_RES_UNFINISHED;
```

# 실습 5 : 상태 처리

- main()함수를 완성하고 게임 실행해보기
- 한줄 완성이 될때마다 completed lines 수가 1 증가
- N\_BINGO 줄 수만큼 완성되면 게임이 끝나야 함

```
=====
X  X  X  X  5
6  7  8  9  10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
=====

No. of completed lines : 0
select a number : 5
=====
X  X  X  X  X
6  7  8  9  10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
=====
```

```
=====
*****
CONGRATULATION!!!!
**&$*#^&*$&^@($!@(* BINGO!!!! #)!@*(#)*$(*#)*@*(
YOU WIN!!!!
*****
=====

=====
X  X  X  X  X
6  7  8  X  10
11 12 X  14 15
16 X  18 19 20
X  22 23 24 25
=====
```

# 실습 6 : 랜덤 배치

- bingo\_init() 함수 수정
  - 2중 반복문을 돌리면서 임의의 값을 생성하여 배치 (randNum)
  - 남아있는 숫자 중 randNum 만큼 큰 숫자를 선택

인덱스	0	1	2	3	4	5	6	7	
	-1	X	X	-1	-1	X	-1	-1	...
숫자	1	2	3	4	5	6	7	8	

randNum = 5      bingoBoard[i][j] = 8

```
void bingo_init(void)
{
    int i, j, k;
    int randNum;
    int maxNum = N_SIZE*N_SIZE;

    //numberStatus 값 초기화
    //for (i=0;i<N_SIZE*N_SIZE;i++)
        //numberStatus[i] 값 초기화

    /*
    for (i=0;i<N_SIZE;i++) {
        for (j=0;j<N_SIZE;j++) {
            randNum = 0 ~ maxNum-1 중 임의의 값 생성

            for (k=0;k<N_SIZE*N_SIZE;k++) {
                if (numberStatus[k]가 아직 할당이 안됨)
                {
                    if (randNum == 0) {
                        break;
                    }
                    else
                    {
                        randNum--;
                    }
                }
            }
            bingoBoard의 i번째 행, j번째 열에 k+1 값 할당;
            numberStatus[k] = N_SIZE*i+j;
            maxNum 값 하나 줄임;
        }
    }
    */
}
```

# 실습 6 : 랜덤 배치

- 전체 동작 검증

# 추가 문제 : N\_PLAYER

- N명이 같이 play 할 수 있는 게임으로 확장
  - bingoBoard 배열을 3차원으로 확장
  - numberStatus 배열을 2차원으로 확장
  - 각 함수들의 입력으로 player 번호를 입력 받아서 처리할 수 있도록 함
  - main 함수에서 while문을 돌 때 각 player 별로 번호 선택
  - 먼저 bingo를 완성한 player가 이기도록 판단 알고리즘 추가