4190.103A-001: Programming Practice Lab. 10

Euidong Lee
Junseok Lee
Minhyo Jung
(snucsl.ta@gmail.com)

Systems Software & Architecture Lab.

Seoul National University

Spring 2022

실습내용

- 문자열 & 문자열 라이브러리
- 포인터 실습
- 게임 만들기 Pico 2048
- 과제 설명

참고

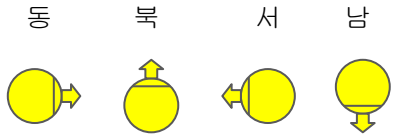
- Programming Sites
  - Backjoon Online Judge: https://www.acmicpc.net/
  - AlgoSpot: https://www.algospot.com/judge/problem/list/
  - LeetCode Online Judge: https://leetcode.com/
  - Project Euler: https://projecteuler.net/
  - CodeCombat: http://codecombat.com/
  - Code.Org: https://code.org/

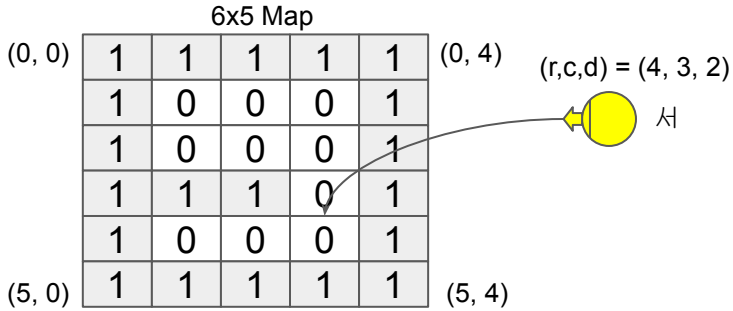
과제 풀이

Lab 9 과제 1

과제 1 – 설명

- 로봇 청소기가 움직인 거리 및 청소한 칸의 개수를 구하세요
  - N x M 크기의 Matrix형태의 Map을 가짐
  - 청소기는 한칸씩 이동을 하며 청소진행. 벽은 이동할 수 없음
  - 청소기가 바라보는 방향은 동,서,남,북 중 하나임
  - 지도의 북쪽으로부터 r번째, 서쪽으로부터 c번째 위치한 칸을 (r, c)라고 부름
  - 입력
    - 세로크기 : N, 가로크기 : M (3 <= N, M <= 10)
    - 청소기의 초기위치 (r, c), 바라보는 방향 d (동 : 0, 북 : 1, 서 : 2, 남 : 3),
    - 빈칸은 0, 벽은 1 로 이루어진 NxM 행렬 Map.
    - 지도의 첫 행, 마지막 행, 첫 열, 마지막 열은 모두 벽임
  - 출력
    - 움직인 거리, 청소한 칸의 개수
    - 청소 후 전체 맵

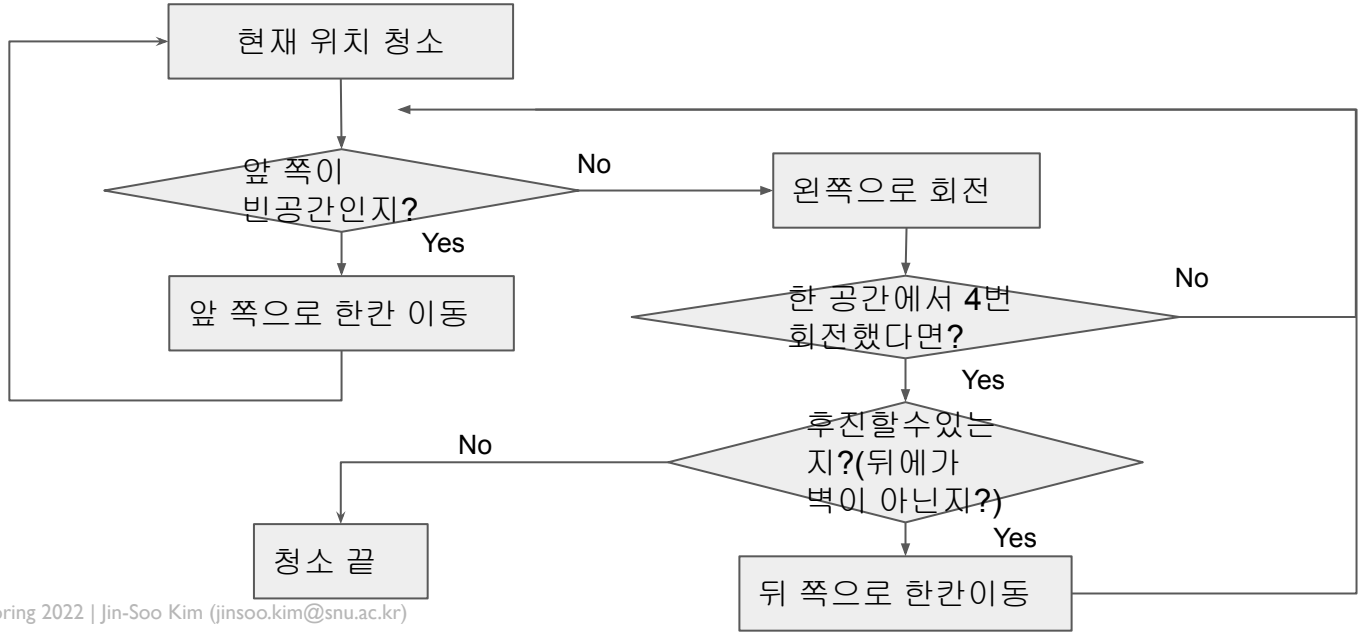




과제 1 – 설명

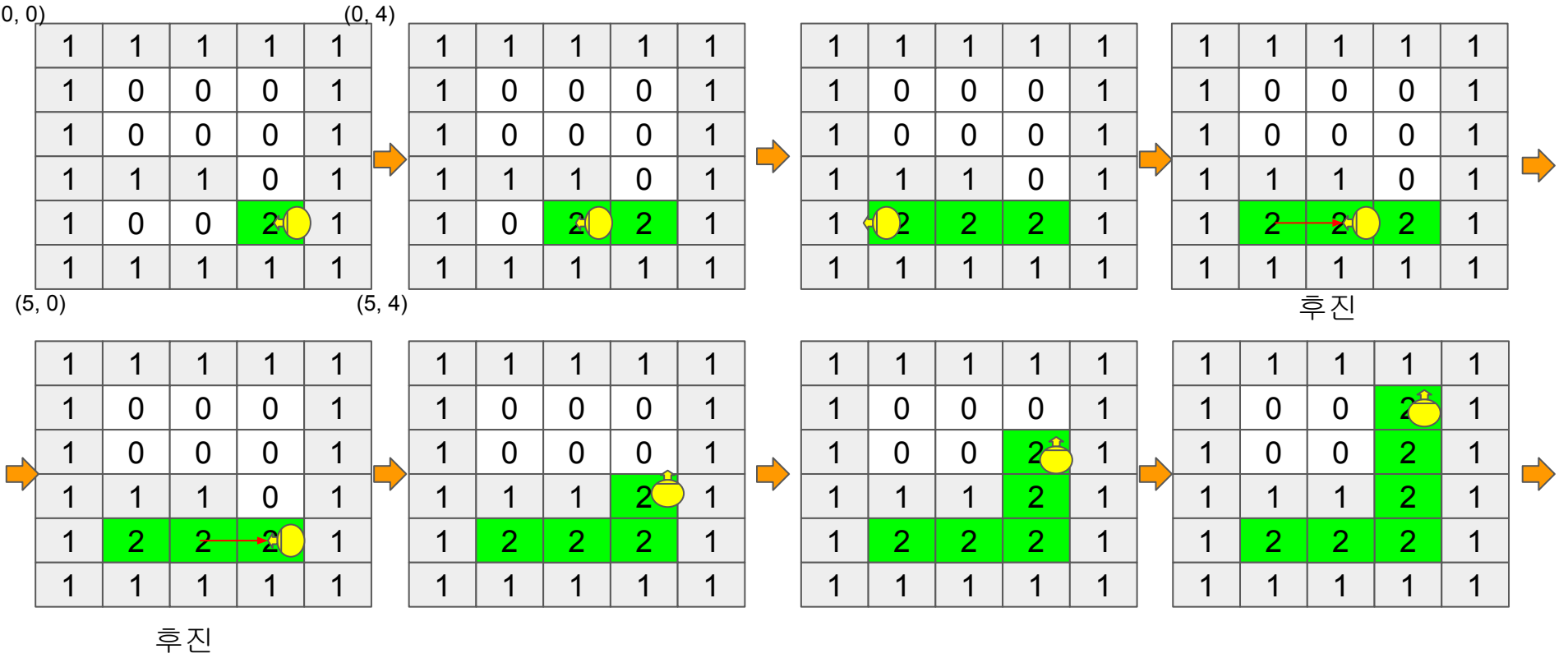
청소 순서

1. 현재 위치 청소함
2. 현재 방향의 앞쪽에 청소하지 않은 빈 공간이 존재한다면, 앞쪽으로 한칸 전진하고 1번으로 돌아감. 그렇지 않을 경우 왼쪽으로 회전하여 다시 2번 진행함
3. 2번 단계가 연속으로 4번 실행되었을 경우(현재 위치를 기준으로 동,서,남,북 모두가 벽 혹은 이미 청소한 곳이라면) 한 칸 후진하고 2번 다시 진행. 만약 바로 뒤쪽이 벽이라면 청소 끝



과제 1 – 설명

0 : 청소해야할 곳
1 : 벽
2 : 청소한 곳
초기 위치 : (4, 3, 2(서))



과제 1 – 설명

후진

주변에 청소안한 칸은 없고,
뒷쪽은 벽이라 후진할 수
없으므로 청소 끝

청소한 칸의 개수 : 10
움직인 거리 : 12

과제 1 – 출력 예시

입력 1 : N, M
입력 2 : 초기 위치
(Row,Column, Direction)
(동: 0, 북:1, 서:2, 남:3)

입력 3: 벽을 포함한 Map
(1:벽, 0:청소해야할곳)

출력 1 :
청소한 칸의 개수,
이동한 횟수
출력 2 : Map 전체
(2: 청소를 한 곳)

```
> ./main
6 5
4 3 2
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 1 1 0 1
1 0 0 0 1
1 1 1 1 1

10 12
1 1 1 1 1
1 2 2 2 1
1 2 2 2 1
1 1 1 2 1
1 2 2 2 1
1 1 1 1 1
```

```
> ./main
6 6
1 1 1
1 1 1 1 1 1
1 0 0 0 0 1
1 0 0 0 0 1
1 0 0 0 0 1
1 0 0 0 0 1
1 1 1 1 1 1

16 17
1 1 1 1 1 1
1 2 2 2 2 1
1 2 2 2 2 1
1 2 2 2 2 1
1 2 2 2 2 1
1 1 1 1 1 1
```

```
> ./main
5 5
3 3 3
1 1 1 1 1
1 0 0 0 1
1 1 1 0 1
1 1 0 0 1
1 1 1 1 1

5 6
1 1 1 1 1
1 2 2 2 1
1 1 1 2 1
1 1 0 2 1
1 1 1 1 1
```

청소를 못하는 부분도
발생함

과제 1 – 해답

```c
#include <stdio.h>

#define MAX_LENGTH 20
#define NUM_DIRECTION 4

#define EAST 0
#define NORTH 1
#define WEST 2
#define SOUTH 3

#define EMPTY   0
#define WALL    1
#define CLEANED 2
```

```c
void print_map(int map[][MAX_LENGTH], int n, int m)
{
  for (int i = 0; i < n; i++){
    for (int j = 0; j < m; j++)
      printf("%d ", map[i][j]);

    printf("\n");
  }
  printf("\n");
}
```

※ define 활용으로 코드 가독성을 높일 수 있습니다.

과제 1 – 해답

```c
int move(int map[][MAX_LENGTH], int
*row, int *column, int direction) {
  int r = *row;
  int c = *column;

  switch(direction)
  {
    case EAST : c++; break;
    case WEST : c--; break;
    case NORTH : r--; break;
    case SOUTH : r++; break;
  }
  if (map[r][c] == EMPTY){
    *row = r;
    *column = c;
    return 1;
  }
  return 0;
}
```

```c
int back(int map[][MAX_LENGTH], int
*row, int *column,
 int direction){
  int r = *row;
  int c = *column;
  switch(direction)
  {
    case EAST : c--; break;
    case WEST : c++; break;
    case NORTH : r++; break;
    case SOUTH : r--; break;
  }
  if (map[r][c] != WALL)
  {
    *row = r;
    *column = c;
    return 1;
  }
  return 0;
```

※ 방향에 따라 **row/column** 값을 **update** 합니다.
move/back은 반대로 수행합니다. 다른 코드는 동일.

```c
void rotate(int * d)
{
  *d = (*d + 1) % NUM_DIRECTION;
}
```

※ **modular** 연산을 이용해 순서대로 회전할 수 있게 했습니다.

과제 1 – 해답

```c
int main(void) {
  int n, m, r, c, d;
  int map[MAX_LENGTH][MAX_LENGTH] = {0,};
  int clean_cnt = 0, move_cnt = 0;
  int is_moved;
  scanf("%d %d", &n, &m);
  scanf("%d %d %d",&r, &c, &d);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      scanf("%d", &map[i][j]);

  while(1) {
    if (map[r][c] == EMPTY) {
      clean_cnt++;
      map[r][c] = CLEANED;
    }

    is_moved = 0;
    for (int i = 0; i < NUM_DIRECTION; i++) {
      if (move(map, &r, &c, d)) {
        is_moved = 1;
        move_cnt++;
        break;
      }
      rotate(&d); // 왼쪽으로 회전
    }
    if (is_moved == 0) {// 동서남북 모두 청소안한곳이 없어
앞으로 이동못했다면?
      if (back(map, &r, &c, d)) move_cnt++;
      else break; // 청소 끝
    }
  }
  printf("\n%d %d\n", clean_cnt, move_cnt);
  print_map(map, n, m);
  return 0;
}//main
```

과제 1 – 학생답안1

```c
#include<stdio.h>

int main() {
   int N, M, row, column, direction;
   int clean = 1, move = 0, turn_count = 0,
backing = 0;
   int Matrix[10][10];
   scanf("%d %d", &N, &M);
   scanf("%d %d %d", &row, &column,
&direction);
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
         scanf("%d", &Matrix[i][j]);
      }
   }
```

```c
Matrix[row][column] = 2;
   while (1) {
      if (direction == 0) {
         if (Matrix[row][column+1] != 0) {
            direction++;
            turn_count++;
         }
         else {
            column++;
            Matrix[row][column] = 2;
            turn_count = 0;
            move++;
            clean++;
            backing = 0;
         }
      }
```

과제 1 – 학생답안1

```
    else if (direction == 1) {
        if (Matrix[row+1][column] != 0) {
            direction++;
            turn_count++;
        }
        else {
            row++;
            Matrix[row][column] = 2;
            turn_count = 0;
            move++;
            clean++;
            backing = 0;
        }
    }
```

```
/* (skip) */

    printf("\n%d %d\n", clean, move);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            printf("%d ", Matrix[i][j]);
        }
        printf("\n");
    }
}
```

과제 1 – 학생답안2

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int *Map;
int total=0;

void sweep(int *pos,int M){
  int row=pos[0];
  int col=pos[1];
  if(*(Map+M*row+col)==0){
    *(Map+M*row+col)=2;
    total++;
    }
}

int check(int row,int col,int M){
  if(*(Map+M*row+col)==0)
    return 1;
  else if(*(Map+M*row+col)==2)
    return -1;
  return 0;
}
```

```c
void move(int *pos,int *d){
  switch (*d){
    case 0:
      (*(pos+1))++;
      break;
    case 1:
      (*pos)--;
      break;
    case 2:
      (*(pos+1))--;
      break;
    case 3:
      (*pos)++;
      break;
  }
}

void reverse(int *pos,int *d){
  int reversed=(*d+2)%4;
  move(pos,&reversed);
}
```

과제 1 – 학생답안2

```c
int search(int *pos,int *d,int M){
  int row=pos[0];
  int col=pos[1];
  int cnt=0;
  while(cnt<4){
    switch (*d){
      case 0:
        if(check(row,col+1,M)==1)
          return 1;
          break;
      case 1:
        if(check(row-1,col,M)==1)
          return 1;
          break;
      case 2:
        if(check(row,col-1,M)==1)
          return 1;
          break;
      case 3:
        if(check(row+1,col,M)==1)
          return 1;
          break;
      }
    *d=(*d+1)%4;
    cnt++;
  }
  return 0;
}

void print(int *Map,int N,int M){
  for(int i=0;i<N;i++){
    for(int j=0;j<M;j++){
      printf("%d ",*(Map+i*M+j));
    }
    printf("\n");
  }
}
```

과제 1 – 학생답안2

```c
int main(void) {
  int N,M;
  Map=calloc(100,sizeof(int));
  assert (Map!=NULL);
  int pos[2];
  int d;
  int path=0;

  scanf("%d %d",&N,&M);
  scanf("%d %d %d",&pos[0],&pos[1],&d);
  for(int i=0;i<N;i++){
    for(int j=0;j<M;j++){
      scanf("%d",Map+M*i+j);
    }
  }
```

```c
  while(1){
    sweep(pos,M);
    //printf("%d %d %d\n",pos[0],pos[1],d);
    if(search(pos,&d,M)){
      move(pos,&d);
      path++;
      }
    else{
      //printf("reversed!\n");
      reverse(pos,&d);
      if(check(pos[0],pos[1],M)==-1)
        path++;
      else
        break;
    }
  }
  printf("\n");
  printf("%d %d\n",total,path);
  print(Map,N,M);
  free(Map);
  return 0;
}
```

Lab 9 과제 2

과제 2 – 설명

- 주어진 영단어의 알파벳을 모두 입력할 때까지, 알파벳을 한 글자씩 입력하며 주어진 횟수안에 맞추는 게임을 만들어봅시다
  - 정답이 되는 영단어 하나를 입력하고 엔터키를 누름 (10글자 이하)
    - getchar()사용시 '\n'이 나올 때까지 입력
    - scanf("%s") 사용해도 됨
  - 정답을 맞추기 위해 한 글자씩 알파벳을 입력하고 엔터키를 누름
  - 알파벳을 하나씩 입력할 때마다 현재 상태를 화면에 출력
  - 영단어 길이의 2배까지 시도 가능
  - 단어를 맞추거나, 최대 시도 횟수를 경과하면 프로그램 종료
  - 대소문자는 구분하지 않음

```
> ./main
Apple
[0]: a
A _ _ _ _
[1]: b
A _ _ _ _
[2]: l
A _ _ l _
[3]: p
A p _ l _
[4]: p
A p p l _
[5]: e
A p p l e
Success
```

Tip. 한 글자씩 입력 받기 위해 getchar() 함수를 사용하고 나서 엔터 문자를 버퍼에서 제거해줘야 정상 동작함

과제 2 – 출력 예시

입력 1 : 정답 영단어
입력 2 : 알파벳

출력 1 : 현재상황.
맞춘 알파벳은 출력하고,
그렇지 않은 알파벳은 '_'
으로 출력
(알파벳 사이에 공백)

출력 2 : 제한된 횟수안에
정답을 맞추면 Success,
아니면 Fail 출력

```
> ./main
apple
[0]: p
_ p _ _ _
[1]: l
_ p _ l _
[2]: p
_ p p l _
[3]: p
_ p p l _
[4]: e
_ p p l e
[5]: a
a p p l e
Success
```

대소문자 구분안함
상태 출력시 정답의 것으로 출력

```
> ./main
ProGram
[0]: p
P _ _ _ _ _ _
[1]: R
P r _ _ _ _ _
[2]: O
P r o _ _ _ _
[3]: g
P r o G _ _ _
[4]: R
P r o G r _ _
[5]: A
P r o G r a _
[6]: M
P r o G r a m
Success
```

최대 횟수 : 단어길이 *2
BUS -> 6번(3*2)

```
> ./main
BUS
[0]: a
_ _ _
[1]: b
B _ _
[2]: c
B _ _
[3]: d
B _ _
[4]: e
B _ _
[5]: f
B _ _
Fail
```

과제 2 – 해답

```c
#include <stdio.h>
#define MAX 10

void print_current_state(char arr[], int
done[], int length)
{
  for (int i = 0; i < length; i++)
    if (done[i])
      printf("%c ", arr[i]);
    else
      printf("_ ");
  printf("\n");
}
```

```c
char to_lowercase(char c)
{
  if (c >= 'A' && c <= 'Z')
    c = c - ('A' - 'a');
  return c;
}

char to_uppercase(char c)
{
  if (c >= 'a' && c <= 'z')
    c = c + ('A' - 'a');
  return c;
}
```

과제 2 – 해답

```c
int main(void) {
  char answer[MAX];
  int done[MAX] = {0,};
  int count = 0;
  int length = 0;
  int num_try;
  char guess;

  while ((guess = getchar()) != '\n')
  {
    answer[length++] = guess;
  }
  for (int num_try = 0; num_try < length*2 && count < length;num_try++)
  {
    printf("[%d]: ", num_try);
    guess = getchar();
    for (int i = 0; i < length; i++) {
      if (done[i] != 0)
        continue;

      if (answer[i] == to_lowercase(guess) ||
          answer[i] == to_uppercase(guess)) {
        done[i] = 1;
        count++;
        break;
      }
    }
    print_current_state(answer, done, length);
    getchar(); // '\n' 버우기
  }
  if (count == length) printf("Success\n");
  else printf("Fail\n");

  return 0;
}
```

과제 2 – 학생답안1

```c
#include <stdio.h>
char ch[11] = {0};
char result[11] = {0};
int search(char c, int length) {
  for (int j = 0; j < length; j++) {
    if (c <= 'Z' && c >= 'A') {
      if (ch[j] == c) {
        result[j] = c;
        ch[j] = -1;
        return 1;
      } else if (ch[j] == c + 'a' - 'A') {
        result[j] = c + 'a' - 'A';
        ch[j] = -1;
        return 1;
      }
    } else if (c <= 'z' && c >= 'a') {
      if (ch[j] == c) {
        result[j] = c;
        ch[j] = -1;
        return 1;
      } else if (ch[j] == c + 'A' - 'a') {
        result[j] = c + 'A' - 'a';
        ch[j] = -1;
        return 1;
      }
    }
  }
  return 0;
}
```

과제 2 – 학생답안1

```c
int main(void) {
  char c;
  scanf("%s", ch);
  getchar(); //버퍼 정리

  int length = 0;
  while (ch[length] != '\0') {
    length++;
  }

  int cnt = 0;

  for (int i = 0; i < 2 * length; i++) {
    printf("[%d]: ", i);
    c = getchar();

    if (search(c, length) == 1)
      cnt++;

    for (int k = 0; k < length; k++) {
      if (result[k] == 0)
        printf("_ ");
      else
        printf("%c ", result[k]);
    }

    getchar(); //버퍼 정리
    printf("\n");

    if (cnt == length) {
      printf("Success\n");
      return 0;
    }
  }
  printf("Fail\n");
  return 0;
}
```

과제 2 – 학생답안2

```c
#include <stdio.h>

#define MAX_LENGTH 10

int main(void) {
  int word[MAX_LENGTH];  // 사용자가 입력하는 단어
  int c;  // 한개씩 입력되는 알파벳
  int index = 0;  // 단어 길이
  int success[MAX_LENGTH] = {0,};  // 단어의 알파벳 중 맞춘 인덱스는 1

  while((c = getchar()) != '\n') {
    word[index++] = c;
  }

  int matched = 0;  // 맞춘 알파벳 개수

  for (int i = 0; i < index * 2; i++) {
    printf("[%d]: ", i);
    c = getchar();

    if (!(getchar() < 'A' || getchar() > 'z' && getchar() < 'A' ||
      getchar() > 'Z'))
      continue;

    for (int j = 0; j < index; j++) {
      if (success[j])
        continue;

      if (c >= 'a' && c <= 'z' && (c == word[j] ||
          c - 'a' + 'A' == word[j]) || c >= 'A' && c<= 'Z' &&
        (c == word[j] || c - 'A' + 'a' == word[j])) {
        success[j] = 1;
        matched++;
        break;
      }
    }
```

과제 2 – 학생답안2

```
    for (int j = 0; j < index; j++) {
      if (success[j] == 1)
        printf("%c ", word[j]);
      else
        printf("_ ");
    }

    printf("\n");

    if (matched == index) {
      printf("Success\n");
      return 0;
    }
  }

  printf("Fail\n");

  return 0;
}
```

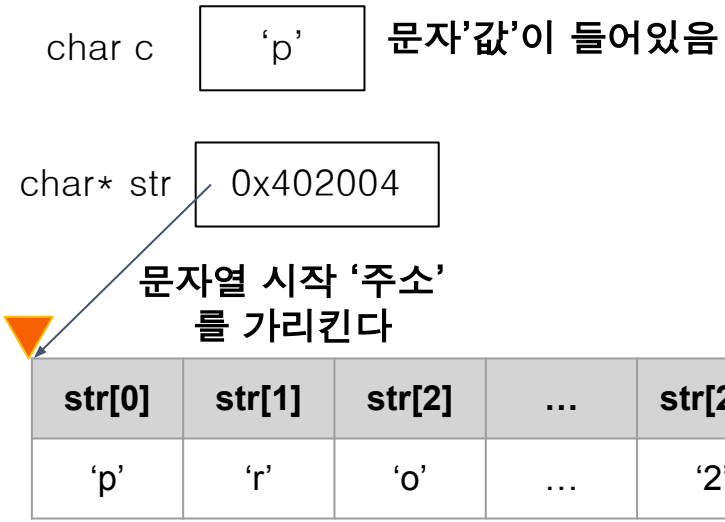
실습 1

문자열

- 문자열 개념을 다시 살펴보면
  - 1차원의 char 타입의 배열이고, 즉 문자 여러개가 계속 이어진 상태입니다.
  - 문자 하나는 1 byte 크기의 char 형에 저장이 가능하지만,
  - 이보다 크면 그게 어렵겠죠?
  - 그리고 중요한 부분이 널문자 (0x00 또는 '\0')가 마지막에 오는 점 입니다.

문자열

- 문자와 문자열(char *)의 차이
- 문자열의 메모리 주소를 출력 해봅시다.




```c
#include <stdio.h>
#define MAX (64)

int main(int argc, char* argv[])
{
  char c = 'p';
  char* str = "ProgrammingPractice2022";

  printf("%p\n", str);
}
```

```
make -s
./main
0x402004
```

문자열

- 다음 예제를 완성하며 마지막 문자의 index를 추측해봅시다.
- strlen()은 문자열의 길이를 구해주는 함수입니다.

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[])
{
  char* str = "Any last words?";
  int index = strlen(str) + ①____;
  // print question mark as a character
  printf("%c\n", str[index]);
}
```

```
make -s
./main
?
```

문자열

- 정답은 -1 입니다.
- 문자열의 경우 항상 NULL문자('\0')를 조심해야합니다.

strlen(str) == 15

	str[0]	str[1]	str[2]	…	str[14]	str[15]
값	'a'	'n'	'y'	…	'?'	'\0'
index	0	1	2	…	**14**	15

문자열

- '\0'과 0과 '0'의 차이는?
- 아래 코드에서 asciiNul과 zero는 0의 값을, zeroChar는 0x30의 값을 가지고 있습니다.

```
int main() {

  char asciiNul = '\0'; // ASCII CODE 0x00
  int zero = 0;
  char zeroChar = '0'; // Numeric literal 0. ASCII 0x30 or 48

  return 0;
}
```

문자열

- 문자열은 다음처럼 %s를 이용해 입력 받을 수 있습니다.

```c
#include <stdio.h>
#define MAX (64)

int main()
{
    char str[MAX];
    scanf("%s", str);
    printf("%s\n", str);

    return 0;
}
```

```
make -s
./main
ProgrammingPractice
ProgrammingPractice
```

문자열

- 그렇다면 문자열 포인터로 처리하면 어떨까요?
- 정상적으로 처리되지 않습니다. 왜 그럴까요?

```c
#include <stdio.h>
#define MAX (64)

int main()
{
    char* str;
    scanf("%s", str);
    printf("%s\n", str);

    return 0;
}
```

```
> make -s
> ./main
ProgrammingPractice
(null)
>
```

문자열

- 입력 값을 문자열 포인터에 저장하려면 저장 공간이 따로 필요합니다.
- malloc 함수를 이용해 메모리를 할당해주고 문자열을 저장했습니다.

```c
#include <stdio.h>
#include <memory.h>
#define MAX (64)

int main() {
  char *str = malloc(sizeof(char) * MAX);
  scanf("%s", str);
  printf("%s\n", str);
  free(str);
  return 0;
}
```

```
make -s
./main
ProgrammingPractice
ProgrammingPractice
```

문자열 Library

```
make -s
./main
79
79
```

- <string.h> library 의 일부 기능들을 사용해보겠습니다.
- strlen은 문자열의 길이를 구해주는 함수 입니다. (string length)

```c
#include <stdio.h>
#include <string.h>
#define MAX (1024)

int main() {
  char *str_ptr = "Software is a great combination between artistry and engineering. -- Bill Gates";
  char str_arr[MAX] = "Software is a great combination between artistry and engineering. -- Bill Gates";
  printf("%d\n", strlen(str_ptr));
  printf("%d\n", strlen(str_arr));
  return 0;
}
```

```c
size_t strlen(const *_Str);
```

문자열 Library

- strcmp는 두 개의 문자열이 같은지 판단할 수 있습니다.
- 두 문자열이 같으면 0을 return 하고, 같지 않은 경우에는 0이 아닌 값을 return 합니다.
- str1이 긴 경우에는 1을 return 하고,
- str2가 긴 경우에는 -1을 return 합니다.

```
int strcmp(const char* str1, const char* str2);
```

문자열 Library

- 아래 예제를 통해 strcmp를 사용해봅시다.
- str1과 str2는 같으므로 0이, str1과 str3은 다르므로 1이 출력됩니다.

```c
#include <stdio.h>
#include <string.h>

int main() {
  const char *str1 = "Programming";
  const char *str2 = "Programming";
  const char *str3 = "Programing";
  printf("strcmp(%s, %s) = %d\n", str1, str2, strcmp(str1, str2));
  printf("strcmp(%s, %s) = %d\n", str1, str3, strcmp(str1, str3));
  return 0;
}
```

```
> make -s
> ./main
strcmp(Programming, Programming) = 0
strcmp(Programming, Programing) = 1
>
```

문자열 Library

- strcpy를 사용해봅시다.
-

```
char* strcpy(char* dest, const char* origin);

#include <stdio.h>
#include <string.h>
#define MAX (128)

int main(void) {
    char org[] = "Programming Practice 2022";
    char dst[MAX];
    strcpy(dst, org);
    printf("%s\n", dst);

    return 0;
}
```

문자열 Library

- strcpy는 org에서 NULL문자까지 복사합니다.
-

index	0	1	2	…	**22**	**23**	24	25	26	27	…	127
org	‘p’	‘r’	‘o’	…	‘0’	‘2’	‘2’	‘\0’	-	-	-	-
dst	‘p’	‘r’	‘o’	…	‘0’	‘2’	‘2’	‘\0’	‘\0’	‘\0’	‘\0’	‘\0’

실습 2

제출 - 회문 판별 (10 mins)

- 회문(回文, Palindrome)이란?
  - 거꾸로 읽어도 제대로 읽는 것과 같은 문장, 숫자열, 문자열 등 (e.g. level, eye, gag, level & noon)
- 입력
  - 문자열 하나 [알파벳 소문자만 고려합니다. (공백, 특수문자 없음, null 포함 최대길이 128)]
- 출력
  - 회문인 경우: [ ] is a palindrome
  - 회문이 아닌 경우: [ ] is not a palindrome

※ 필요에 따라 <string.h>를 사용해보세요.

회문 판별 - 설명

- 알고리즘 자체는 그냥 숫자 배열처럼 생각해도 되지만,
- 배열의 크기를 유심히 챙겨보세요!

a[0]	a[1]	a[2]
101	121	101

c[0]	c[1]	c[2]	c[3]
‘e’	‘y’	‘e’	‘\0’

```
> make -s
> ./main
3: eye
3: eye
>
```

```c
#include <stdio.h>
#include <string.h>

int main(void) {
  int a[3] = {101, 121, 101};
  char c[4] = {'e', 'y', 'e', '\0'};

  int a_len = sizeof(a)/sizeof(int);
  int c_len = strlen(c);

  printf("%d: ", a_len);
  for (int i = 0; i < a_len; i++)
    printf("%c", a[i]);
  printf("\n");

  printf("%d: %s\n", c_len, c);

  return 0;
}
```

회문 판별 - 알고리즘 예시

- void isPalindrome(char str[]) 함수를 짜는 경우를 생각해보겠습니다.
- 문자열 str의 길이를 찾아 n이라고 둡니다.
- 양쪽 끝을 비교할 수 있도록 left, right index를 지정하고 이를 각각 0과 n-1로 초기화합니다.
- left가 right보다 작은 동안 실행되도록 하고
  - a) str[left]이 str[right]와 같지 않으면 false를 return 합니다.
  - b) 그렇지 않으면 left을 증가 시키고, right를 감소 시킵니다.
- left, right가 위의 조건을 모두 만족하고 빠져나오면 틀린 부분이 없다는 뜻 입니다.

회문 판별 - Code Template

```c
#include <stdio.h>
#include <string.h>

void isPalindrome(char str[]) {
  /* DEBUG PURPOSE ONLY */
  printf("%lu %s\n", strlen(str), str);
  // str의 양쪽 끝에서 비교하는 index.
  // 알맞은 값을 지정해보세요.
  int left;
  int right;

  // index가 가리키는 문자들이 같다면 계속 진행함
  while (/*write your code*/) {
    /*write your code */
  }
  /*write your code*/
}
```

```c
int main()
{
  char c[128];
  scanf("%s", c);
  isPalindrome(c);
}
```

```
> make -s
> ./main
eye
eye is a palindrome
> ./main
gag
gag is a palindrome
> ./main
apple
apple is not a palindrome
> ./main
horse
horse is not a palindrome
> ./main
saippuakivikauppias
saippuakivikauppias is a palindrome
>
```

회문 판별 - 해답

```
void isPalindrome(char str[])
{
    int left = 0;
    int right = strlen(str) - 1;

    while (right > left)
    {
        if (str[left++] != str[right--])
        {
            printf("%s is not a palindrome\n", str);
            return;
        }
    }
    printf("%s is a palindrome\n", str);
}
```

실습 3

문자열을 함수로 전달하기

- C언어는 문자열을 저장하는 별도의 자료형이 없어 char의 배열이나 포인터를 사용합니다.
- 문자열을 함수로 전달하는 방법은 다음과 같습니다.
- ```
  void  doSomething(char* str);
  ```
- 또는 인자를 배열로 지정할 수 있는데, 이 두 방법은 사실 같습니다.

```
void  doSomething(char str[]);
```
```

문자열을 함수로 전달하기

- 다음 프로그램을 실행시켜 결과를 확인해봅시다.

```c
#include <stdio.h>
void  param_as_empty_array(char str[])
{
  printf("%s %p %lu\n", str, str, sizeof(str));
}
void  param_as_array(char str[20])
{
  printf("%s %p %lu\n", str, str, sizeof(str));
}
void  param_as_pointer(char* str)
{
  printf("%s %p %lu\n", str, str, sizeof(str));
}
```

```c
int main(void)
{
  param_as_empty_array("Programming Practice");
  param_as_array("Programming Practice");
  param_as_pointer("Programming Practice");

  return 0;
}
```

```
make -s
./main
Programming Practice 0x40200f 8
Programming Practice 0x40200f 8
Programming Practice 0x40200f 8
```

my_atoi() (10 mins)

- atoi() 함수는 문자열을 받아 숫자로 변경해주는 함수입니다.
- ```
  int atoi (const char* str);
  ```
- 예를 들면, 문자열 “123”을 정수 123으로 변환해주는 함수입니다.
- 문자열 “-100”은 정수 -123 으로 변환합니다!
- 부호를 포함한 숫자로만 이루어진 문자열 2개를 입력 받아,
- my_atoi() 함수를 통해 정수 2개로 변경한 후
- 그 두 합을 출력하는 프로그램을 짜봅시다.
  ```

my_atoi() (10 mins)

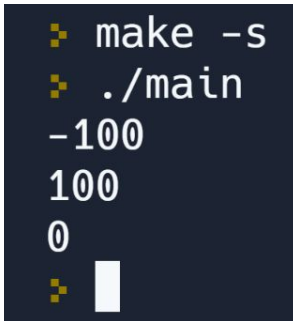
- <stdlib.h>의 atoi() 사용 예

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  char* str1 = "123";
  char* str2 = "-100";
  int num1 = atoi(str1);
  int num2 = atoi(str2);
  printf("%d\n", num1+num2);
  return 0;
}
```

```
make -s
./main
23
```

my_atoi() (10 mins)

- 예를 들어, 문자열 “123”과 “-100”을 입력으로 받으면, 각각 정수 123, -100으로 변환하여 return하는 my_atoi() 함수를 완성합니다.
- 추가로, 그 변수들의 합을 출력하는 예제를 작성해봅시다.
- 정상적인 입력만 가정하도록 합시다.



my_atoi() (10 mins)

```c
#include <stdio.h>
#include <memory.h>
#define MAX (128)

int my_atoi(/*write your code*/) {
/*write your code*/
}
```

```c
int main() {
  char *str1 = malloc(sizeof(char)*MAX);
  char *str2 = malloc(sizeof(char)*MAX);

  scanf("%s", str1);
  scanf(" %s", str2);

  int value1 = my_atoi(str1);
  int value2 = my_atoi(str2);
  printf("%d\n", value1+value2);
  return 0;
}
```

my_atoi() (10 mins)

```c
#include <stdio.h>
#include <memory.h>
#define MAX (128)

int my_atoi(char *str) {
  int res = 0;
  int sign = 1;
  int i = 0;
  if (str[0] == '-') {
    sign = -1;
    i++;
  }
  for (; str[i] != '\0'; i++)
    res = res * 10 + str[i] - '0';

  return sign * res;
}
```

```c
int main() {
  char *str1 = malloc(sizeof(char)*MAX);
  char *str2 = malloc(sizeof(char)*MAX);

  scanf("%s", str1);
  scanf(" %s", str2);

  int value1 = my_atoi(str1);
  int value2 = my_atoi(str2);
  printf("%d\n", value1+value2);
  return 0;
}
```

Pointer to an Array

- 1D Array는 single pointer로 처리 했으니까,
- 2D Array는 double pointer로 처리가 가능할까요?
- 다음 예제를 실행시켜 봅시다.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  int n[2][2] = {{1, 2}, {3, 4}};
  //incompatible pointer types initializing 'int **' with an expression of type 'int [2][2]'
  int **p = numArr;
  //signal: segmentation fault (core dumped)
  printf("%d\n", p[0][0]);
}
```

Pointer to an Array

- int **와 int[2][2]의 타입이 다르다는 경고가 발생하고
- 런타임에 seg fault 에러가 발생합니다.
- 2차원 배열을 포인터에 담으려면 다음과 같이 해야합니다.
- **자료형 (*포인터이름)[가로크기]**; `int (*p)[2];`
- 즉, 가로크기가 2인 배열 (int [2])을 가리키는 포인터입니다.
- **배열**(을 가리키는) **포인터**입니다. (not 포인터 배열)

Pointer to an Array

- 아래 코드는 C에서 불가능한 형태이지만 개념적으로 생각해서,
- `int[2] (*p);`
- 빨간 박스는 → int[2]개짜리 배열을
- 파란 박스는 → 가리키는 포인터
- 라고 생각해볼 수 있습니다.

```
int(*p)[2];
```

Pointer to an Array

- 2X2 배열 예제에 이어서 2X3 으로 실습해보겠습니다.
- 아래 예제를 완성해보세요.

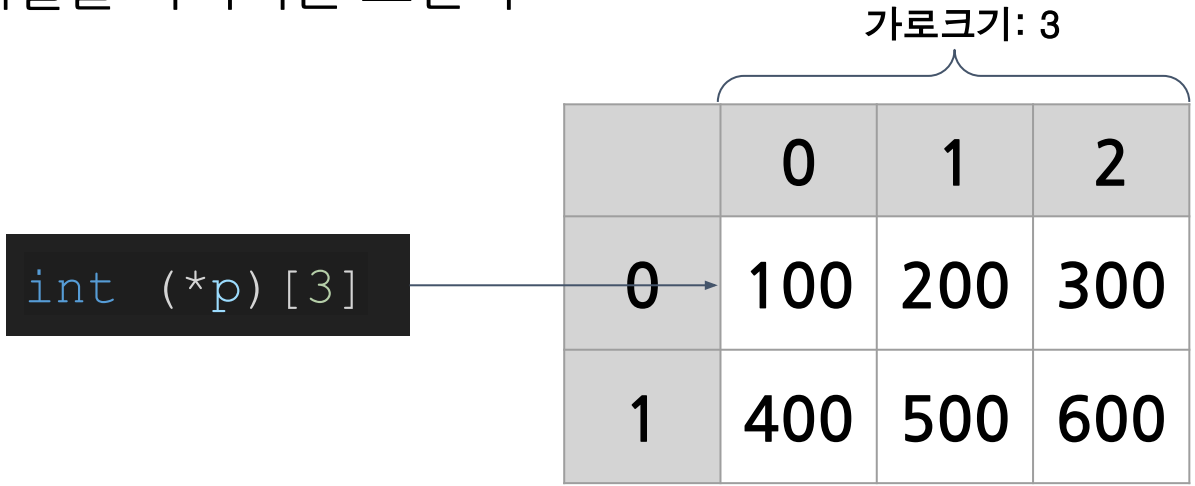
```c
#include <stdio.h>
int printArray(/*write your code*/) {
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
      printf("%d", p[i][j]);
      if (j != 2)
        printf(" ");
    }
    printf("\n");
  }
  return 0;
}
```

```c
int main(void) {
  int arr[2][3] = {{100, 200, 300}, {400,
500, 600}};
  printArray(arr);
}
```

```
make -s
./main
100 200 300
400 500 600
```

Pointer to an Array

- 다음과 같이 하면 됩니다.
- ```
  int printArray(int (*p)[3]);
  ```
- 가로크기가 3인 배열을 가리키는 포인터


  ```

Pointer to an Array

- 처음 예제도 수정해서 다시 실행해봅시다.
- 1이 잘 출력되는 것을 알 수 있습니다.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  int numArr[2][2] = {{1, 2}, {3, 4}};
  int (*numPtr)[2] = numArr;
  printf("%d\n", numPtr[0][0]); // 배열과 동일하게 배열 포인터도 [][] 으로 item 접근
}
```

Pointer to an Array

- 동일한 방식으로 3차원 배열은 다음처럼 전달할 수 있습니다.

```c
#include <stdio.h>

int printArray(int (*p)[3][4]) {
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
      for (int k = 0; k < 4; k++) {
        printf("%4d", p[i][j][k]);
        if (k != 4)
          printf(" ");
      }
      printf("\n");
    }
  }
  return 0;
}
```

```c
int main(void) {
  int arr[2][3][4] = {
      {{100, 200, 300, 400},
       {500, 600, 700, 800},
       {900, 1000, 1100, 1200}}, // 3X4
      {{1300, 1400, 1500, 1600},
       {1700, 1800, 1900, 2000},
       {2100, 2200, 2300, 2400}} // 3X4
  };                             // 2
  printArray(arr);
}
```

```
❯ make -s
❯ ./main
 100  200  300  400
 500  600  700  800
 900 1000 1100 1200
1300 1400 1500 1600
1700 1800 1900 2000
2100 2200 2300 2400
❯
```

Array of Pointers

- **Array** of Pointers, **포인터 배열**은,
- **포인터 변수**를 요소로 가지는 **배열**입니다.
- 즉 포인터 변수를 저장할 수 있는 **배열**입니다.

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int number_1 = 1, number_2 = 2, number_3 = 3;
  int *arr[3] = {&number_1, &number_2, &number_3}; // declaration of 'int type pointer array'
  int array_length = sizeof(arr) / sizeof(arr[0]);
  for (int i = 0; i < array_length; i++) {
    printf("%d\n", *arr[i]);
  }
}
```

Array of Pointers

- 아래에서 arr은 int * (pointer)를 3개 가지는 배열(array) 입니다.
- ```
  int *arr[3];
  ```
- 아래에서 arr은 int [3] (array)을 가리키는 * (pointer) 입니다.
- ```
  int (*arr)[3];
  ```
- 두가지가 헷갈린다면, 정상입니다!
- 포인터 “배열”은 주소값들을 저장하는 “배열” 입니다.
- 배열 “포인터”는 배열의 시작주소값을 아는 “포인터” 입니다.

Array of Pointers - argv

- main 함수는 parameter로 int argc와 char *argv[]를 받습니다.
- argc는 main()에 전달되는 정보의 개수입니다.
- char *argv[]는 전달되는 내용 입니다.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {

  for (int i = 0; i < argc; i++) {
    printf("%s\n", argv[i]);
  }
  return 0;
}
```

```
> make -s
> ./main
[1] items
./main
> ./main Programming Practice
[3] items
./main
Programming
Practice
>
```

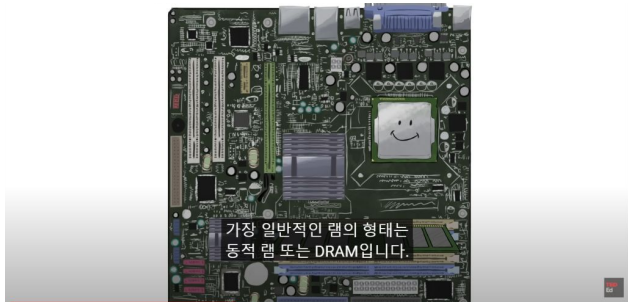
Array of Pointers - argv

- argv의 첫번째 data는 프로그램의 실행 경로로 고정되어 있습니다.
- console 창에 직접 예시를 작성해 보세요.

포인터 - 부연

- 포인터는 CPU와 메모리의 동작에 대한 이해가 필요합니다.
- 아래 영상을 이해에 도움 될 수 있습니다.
- 그리고 앞선 예시들을 변경해가면서 손에 익혀보세요!

How a CPU Works
https://youtu.be/cNN_tTXABUA



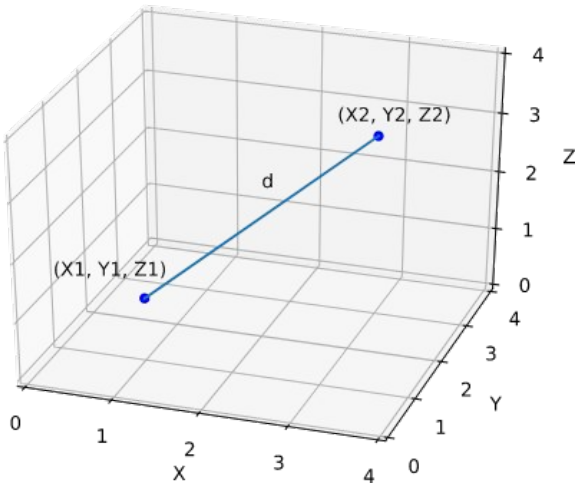
How computer memory works
https://youtu.be/p3q5zWCw8J4

실습 4

3D 두 점 사이의 거리 - 제출 (10 mins)

- 3차원 공간상의 두 점 간의 거리를 계산해봅시다.
- 두 점 P1(X1 Y1 Z1) P2(X2 Y2 Z2)가 주어졌을 때 거리 d는 다음 공식으로 구할 수 있습니다.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$



3D 두 점 사이의 거리

- **입력:** 스페이스로 구분 된 정수 6개
  - (e.g. 7 4 3 17 6 2)
  - 정수는 각각 X1 Y1 Z1 X2 Y2 Z2 에 해당한다.
- **출력:** 두 점 사이의 거리 실수 1개
  - e.g. 10.25
  - 소수점 둘째 자리까지 출력. (반올림 처리)

```
make -s
./main
7 4 3 17 6 2
10.25
```

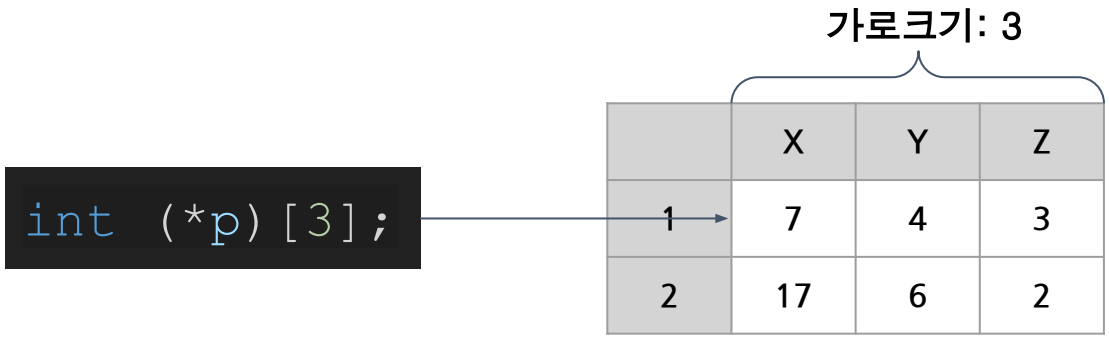
3D 두 점 사이의 거리

- 이차원 배열을 선언하고, 사용자 입력을 받습니다.
- 그리고 이를 **함수에 전달해서** 거리를 계산해보도록 합시다.
- 즉, 거리를 구하는 함수 getDistance에 이차원 배열을 담은 포인터를 넘겨 받아 거리를 계산해 값을 return 하도록 설계해보았습니다.
- root 값은 <math.h> 라이브러리의 sqrt()를 이용해 구할 수 있습니다.

```
double sqrt(double x);
```

3D 두 점 사이의 거리

- int coords[2][3]를 getDistance()의 인자로 전달하려면?



3D 두 점 사이의 거리

- 아래 코드를 참고하여 자신의 코드를 작성해보세요.
- scanf를 통한 입력과, param으로 전달하는 부분을 주의해서 보세요.

```c
#include <stdio.h>
#include <math.h>

double getDistance(int (*coords)[3])
{
  /*write your code*/
  return /*something */
}
```

```c
int main(){

  int coords[2][3]; // 3D coordinates 2개

  scanf("%d %d %d %d %d %d", &coords[0][0],
&coords[0][1], &coords[0][2], &coords[1][0],
&coords[1][1], &coords[1][2]);

  printf("%.2lf\n", getDistance(coords));
  return 0;
}
```

3D 두 점 사이의 거리 - 해답

```
double getDistance(int (*coords)[3])
{
    /*write your code*/
  int x_diff = coords[0][0] - coords[1][0];
  int y_diff = coords[0][1] - coords[1][1];
  int z_diff = coords[0][2] - coords[1][2];
  return sqrt(x_diff*x_diff+y_diff*y_diff+z_diff*z_diff);
}
```

3D 두 점 사이의 거리

- 포인터 배열로도 풀어볼까요?

```c
double getDistance(int *p[6]) {
  int x = /* write your code */;
  int y = /* write your code */;
  int z = /* write your code */;
  return sqrt(x * x + y * y + z * z);
}
```

```c
int main() {
  int coords[2][3]; // 3D coordinates 2개
  scanf("%d %d %d %d %d %d", &coords[0][0], &coords[0][1],
&coords[0][2], &coords[1][0], &coords[1][1], &coords[1][2]);
  int *p[6];
  p[0] = /*write your code */;
  p[1] = /*write your code */;
  p[2] = /*write your code */;
  p[3] = /*write your code */;;
  p[4] = /*write your code */;;
  p[5] = /*write your code */;;
  printf("%.2lf\n", getDistance(p));
  return 0;
}
```

3D 두 점 사이의 거리

- 포인터 배열로도 풀어볼까요?

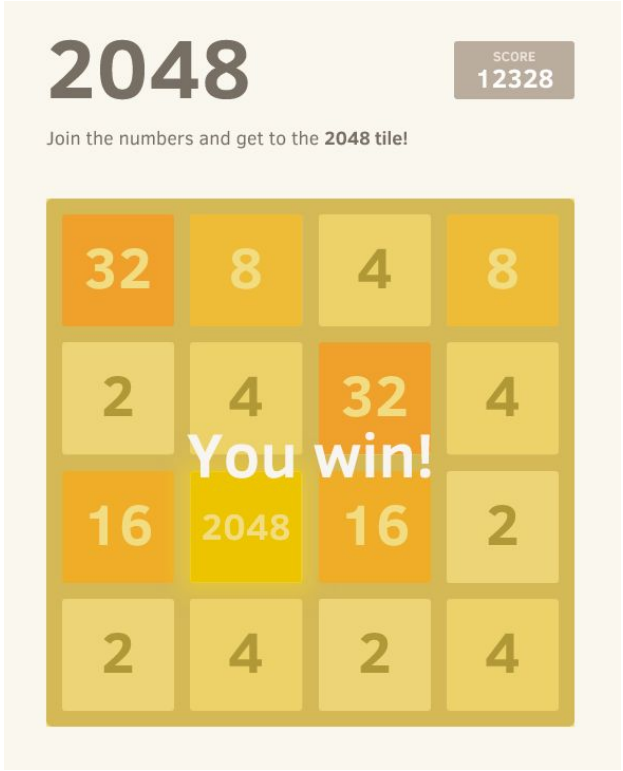
```c
double getDistance(int *p[6]) {
  int x = *p[0] - *p[3];
  int y = *p[1] - *p[4];
  int z = *p[2] - *p[5];
  return sqrt(x * x + y * y + z * z);
}
```

```c
int main() {
  int coords[2][3]; // 3D coordinates 2개
  scanf("%d %d %d %d %d %d", &coords[0][0], &coords[0][1],
&coords[0][2], &coords[1][0], &coords[1][1], &coords[1][2]);
  int *p[6];
  p[0] = &coords[0][0];
  p[1] = &coords[0][1];
  p[2] = &coords[0][2];
  p[3] = &coords[1][0];
  p[4] = &coords[1][1];
  p[5] = &coords[1][2];
  printf("%.2lf\n", getDistance(p));
  return 0;
}
```

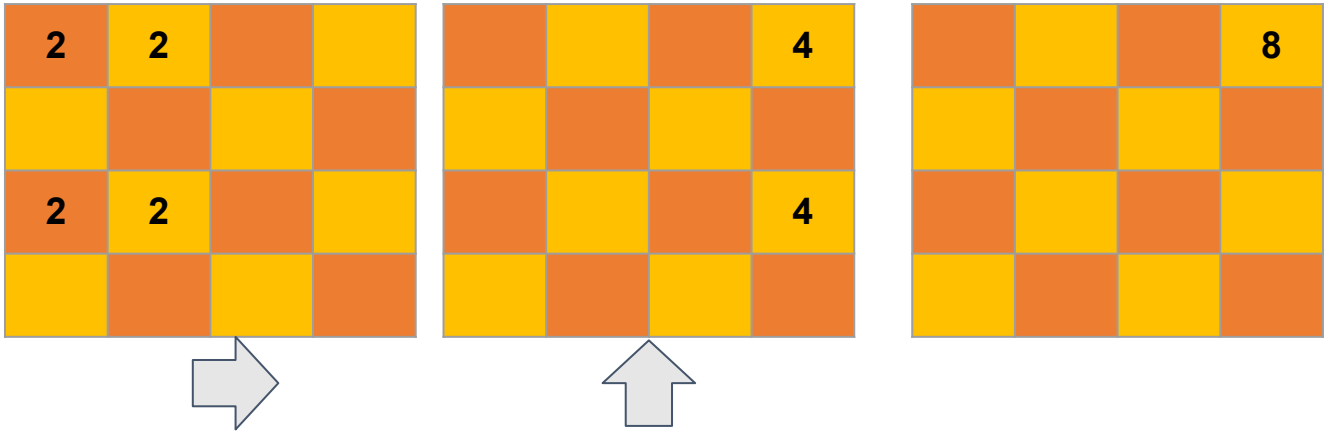
실습 5

Pico 2048 (15+a mins)

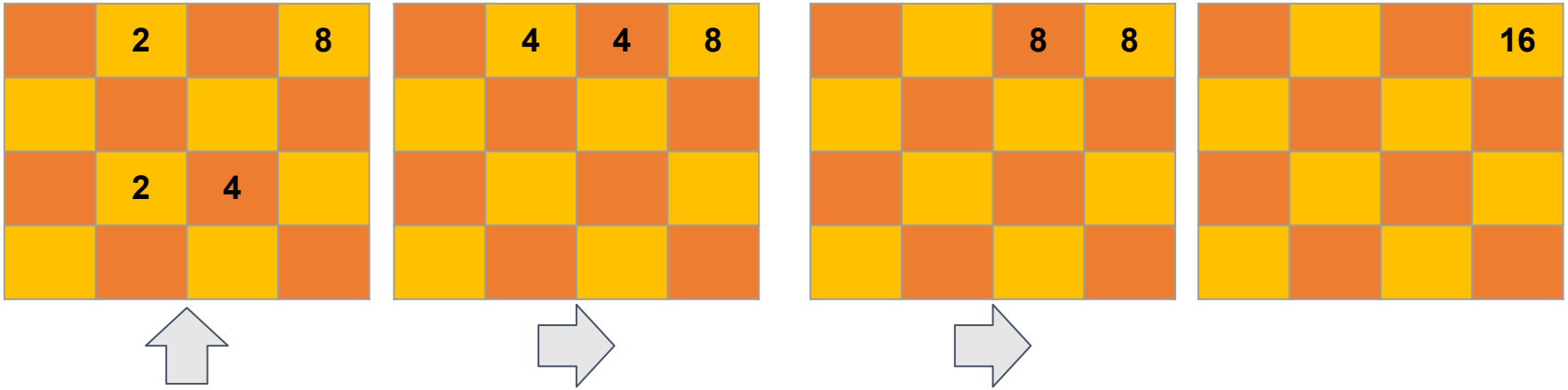
- 2048 게임을 아시나요?
- https://play2048.co/
- https://github.com/gabrielecirulli/2048



Pico 2048



Pico 2048



Pico 2048

- 2X4크기의 변형된 2048 문제를 풀어볼까 합니다.
- 상하좌우 최대 3번 움직임을 통해 해당 경우에서 나올 수 있는 최대값을 구하는 프로그램을 작성해보겠습니다.
  - 제약1: 이동 시켜도 새로운 블럭이 추가되지 않음
- 모든 경우를 찾도록 구현해보고 최적화 방법을 생각해보세요.
- 배열 index 처리를 꼼꼼하게 하지 않으면 정말 괴롭습니다!

Pico 2048

- 입력 : 2 x 4 정수 배열
  - 각 column은 space로, row는 개행으로 구분한다.
  - 빈 칸은 0으로 입력된다.
- 출력 : 3번 이동해서 얻을 수 있는 블럭의 최대값

Pico 2048

- 규칙
- 이동 한 번은 보드 위 전체 블록을 상하좌우 중 한 방향으로 이동시키는 것이다.
- 이때, 같은 값을 갖는 두 블록이 충돌하면 두 블록은 하나로 합쳐지게 된다.
- 단, 한 번의 이동에서 이미 합쳐진 블록은 또 다른 블록과 다시 합쳐질 수 없다.

Pico 2048

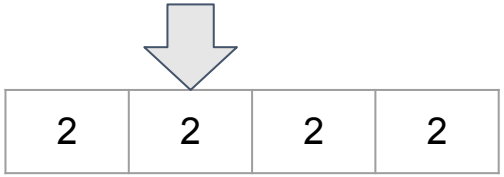
- 1X4 1D array 예제

- move_to_right()
- move_to_left()

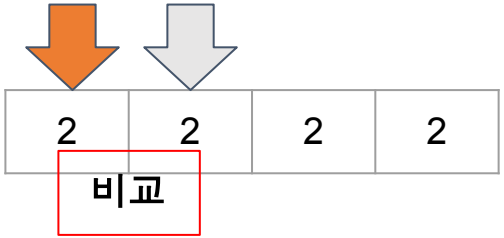
```
make -s
./main
2 2 2 2
4 4 0 0
8 0 0 0
0 0 0 8
```

Pico 2048

- 1X4 1D move_to_left 예제



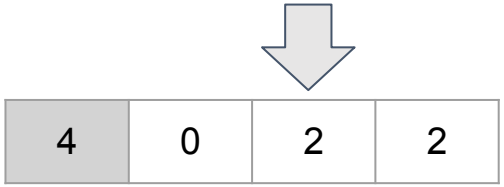
i+1 index에서 시작 (0이면 1)
. 해당 값이 0이면 건너 뛴다.
. 그렇지 않으면 이전 값을 비교한다



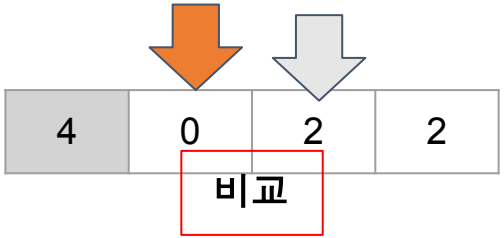
해당 값이 같고, 이번 턴에서 더해지지 않은 경우 왼쪽으로 더한다.
오른쪽 값에는 0을 넣는다.

Pico 2048

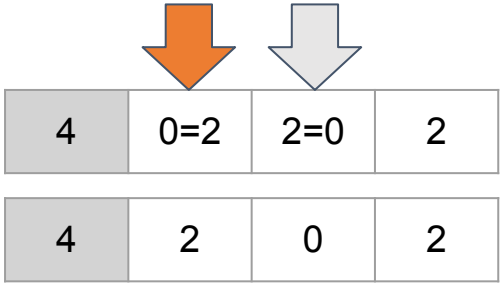
- 1X4 1D move_to_left 예제



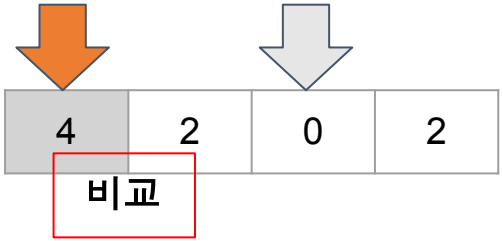
i+1 index에서 시작 (1이면 2)
. 해당 값이 0이면 건너 뛴다.
. 그렇지 않으면 이전 값을 비교한다



해당 값이 0이고 이번 턴에서 더해지지 않은 경우 왼쪽으로 더한다(이동한다).
오른쪽 값에는 0을 넣는다.

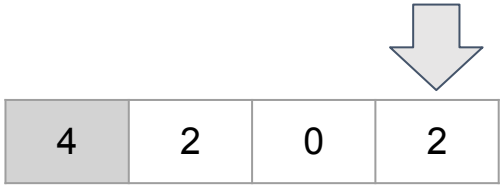


다음 값을 비교합니다.
이미 더해진 경우이기 때문에 끝납니다.

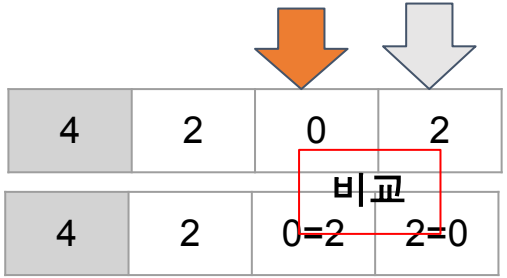


Pico 2048

- 1X4 1D move_to_left 예제

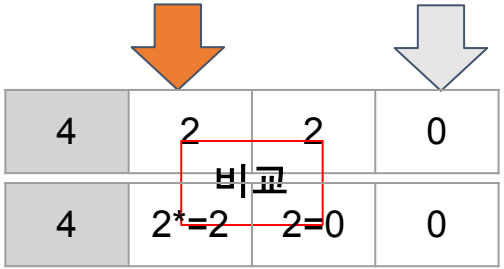


i+1 index에서 시작 (2이면 3)

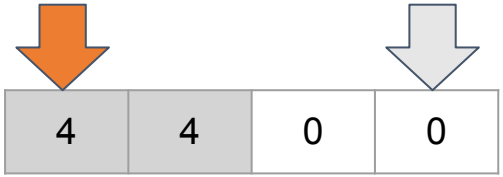


비교하는 곳이 0인 경우 이동한다.

j는 j+1과 비교해서 값이 같고, 이번 턴에서 더한 적이 없으면 계산 수행



다음 값을 비교합니다.
이미 더해진 경우이기 때문에 끝냅니다.



Pico 2048 - 1D Array

```c
#define size (4)
#include <memory.h>
#include <stdio.h>

int a[4]; // array
int v[4]; // visited

void print1DArray() {
  for (int i = 0; i < size; i++)
    printf("%d ", a[i]);
  printf("\n");
}
```

```c
void move_to_right(void) {
  memset((void *)v, 0, sizeof(int));
  for (int i = size - 2; i >= 0; i--) {
    if (a[i] == 0) // skip
      continue;
    for (int j = i + 1; j < size; j++) {
      if (a[j] == a[j - 1] && v[j] == 0) {
        a[j] *= 2;
        a[j - 1] = 0;
        v[j] = 1;
        break;
      } else if (a[j] == 0) {
        a[j] = a[j - 1];
        a[j - 1] = 0;
      }
    }
  }
  print1DArray();
}
```

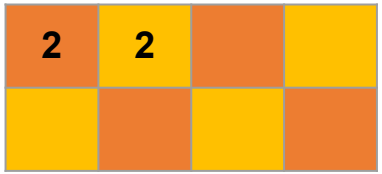
Pico 2048 - 1D Array

```c
void move_to_left(void) {
  memset((void *)v, 0, sizeof(int));
  for (int i = 1; i < size; i++) {  // 1~3
    if (a[i] == 0) {
      continue; // skip
    }
    for (int j = i - 1; j >= 0; j--) {
      if (a[j] == a[j + 1] && v[j] == 0) {
        a[j] *= 2;
        a[j + 1] = 0;
        v[j] = 1;
        break;
      } else if (a[j] == 0) {
        a[j] = a[j + 1];
        a[j + 1] = 0;
      }
    }
  }
  print1DArray();
}
```

```c
int main(void) {
  for (int i = 0; i < size; i++)
    a[i] = 2;

  print1DArray();
  move_to_left();
  move_to_left();
  move_to_right();
}
```

Pico 2048

- 위 설명을 바탕으로 2X4 2D array 예제를 풀어봅시다.

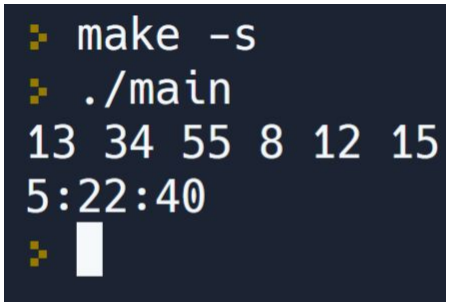


- move_to_right()
- move_to_left()
- move_to_up()
- move_to_down()

과제 1

시간 계산

- 두 개의 시간을 입력 받아서 그 차이를 출력하는 프로그램을 작성하라.
- 입력 : 정수 6개를 순서대로 받는다
  - 이는 각각 다음에 해당한다: 시간1 분1 초1 시간2 분2 초2
  - 예시: 13 34 55 8 12 15 (시간1 13:34:55, 시간2 8:12:15)
- 출력 : 시간1과 시간2의 차이 (시분초는 콜론(:)으로 구분)
  - 예시: 5:22:40
  - 조건1: 사용자 입력을 main 함수 이외의 함수로 전달해 시간을 계산할 것
  - 조건2: 24시간제로 입력을 받는다.
  - 조건3: 처음 들어오는 시간이 나중에 들어오는 시간 보다 크다고 가정한다.
  - 조건4: 분, 초의 경우 10 이하일 경우 두 자리로 출력한다. (%02d 사용)



시간 계산 - Template

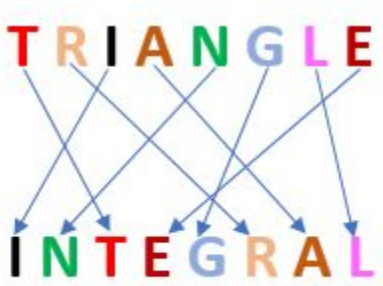
```c
void differenceBetweenTimePeriod(int *start, int *stop, int *diff) {

int main() {

  int start[3]; // int [hours][minutes][seconds]
  int stop[3];
  int diff[3];

  scanf("%d %d %d", &start[0], &start[1], &start[2]);
  scanf("%d %d %d", &stop[0], &stop[1], &stop[2]);

  differenceBetweenTimePeriod(start, stop, diff);
  printf("%d:%02d:%02d\n", diff[0], diff[1], diff[2]);
  return 0;
}
```

과제 1 - 평가 기준

- 기준 1
  - 프로그램이 동작하는가? - 1점
- 기준 2
  - 프로그램이 정상적으로 입력을 받는가? - 1점
- 기준 3
  - 기준에 부합하여 결과를 출력하는가? - 3점

과제 2

Anagram 판별



- Anagram 이란?
  - 단어나 문장을 구성하고 있는 문자의 순서를 바꾸어 다른 단어나 문장을 만드는 놀이
- 입력: 개행으로 구분된 2개의 문자열
  - 각 문자열에는 공백 및 특수문자가 포함될 수 있다.
  - a~z, A~Z 범위 외 character는 입력으로 받되, Anagram 판별에 고려하지 않는다.
  - 각 문자열의 최대 길이는 null character 포함하여 64이다.
- 출력: Anagram이면 “true” 아니면 “false” 출력
  - “false ”와 “false”, “true ”와 “true”는 다른 문자열입니다!

Anagram 판별

- anagram 판별에는 공백, 특수문자는 고려하지 않고 알파벳만 고려.
- 알파벳의 경우에는 대소문자 구분 없음. (i.e., ‘A’와 ‘a’는 같음)
- TC
  - “Dormitory”, “Dirty Room” => True
  - “I am Lord Voldemort”, “Tom Marvolo Riddle” => true
  - “O! DRACONIAN DEVIL”, “LEONARDO DA VINCI” => true
  - “OLIVE DECO”, “I LOVE CODE” => true
  - “Listen”, “Silent” => true
  - false 경우 TC는 직접 작성해서 테스트해보세요.

```
> ./main
Dormitory
Dirty Room
true
> ./main
Dormitories
Dirty Room
false
> ./main
Listen
Silent
true
> ./main
I am Lord Voldemort
Tom Marvolo Riddle
true
> ./main
O! DRACONIAN DEVIL
LEONARDO DA VINCI
true
> ./main
OLIVE DECO
I LOVE CODE
true
>
```

Anagram 판별

- 다양한 방법으로 생각하고 접근해보세요.
- 공백을 포함한 입력은 아래 코드를 참고하세요.
- 개행문자가 나오기 전까지 읽는 기능 입니다.

```
char arr1[MAX];
scanf(" %[^\n]", arr1);
```

Anagram 판별 - 공백이 있는 입력 처리

```c
#define MAX (64)
#define RETURN_ERROR (-1)
#include <stdio.h>

int isAnagram(char str1[], char str2[]);

int main() {
  // declaration of the array
  char arr1[MAX], arr2[MAX];
  int count;
  scanf(" %[^\n]", arr1);
  scanf(" %[^\n]", arr2);
  isAnagram(arr1, arr2);
  return 0;
}
```

과제 2 - 평가 기준

- 기준 1
  - 프로그램이 동작하는가? - 1점
- 기준 2
  - 프로그램이 정상적으로 입력을 받는가? - 1점
- 기준 3
  - 기준에 부합하여 결과를 출력하는가? - 3점