Array of Strings
Dynamic Memory Allocation

For copyright and license information,
http://class.icc.skku.ac.kr/~min/program/license.html

Dynamic Memory

- Use "**void *malloc(size_t size)**;" to get memory from operating system.
- Use "**void free(void *ptr)**;" to return the memory to operating system.
- "malloc()" and "free()" should be a pair, i.e**, all the memory obtained from operating system should be returned** to operating system after use.
- Typical usage

```
int  iarray[100];
int  *parray = (int *)malloc(100*sizeof(int));
free(parray);
```

자주 저지르는 실수 - 1

- malloc을 하고 free는 하지 않는다.

```
int k, *pt;
for ( k = 0 ; k < 1000 ; k++) {
        pt = (int *)malloc(100000*sizeof(int));
        ...
        /* free(pt); */
}
```

자주 저지르는 실수 - 2

- malloc 1번에 free를 2번 이상 수행

```
int *p, *q;
p = (int *)malloc(10*sizeof(int));
q = p;
free(p);
free(q);
p = (int *)malloc(10*sizeof(int));
q = (int *)malloc(10*sizeof(int));
p[0] = 12;
/* Since “p” and “q” may be the same,
 the following code may change the value of p[0]
*/
q[0] = 20;
```

자주 저지르는 실수 - 3

- malloc하지 않은 pointer를 free한다.

```
int *pt;            /* nobody knows value of “pt” */
free(pt);           /* value of pt is registered as a memory */
pt = (int *)malloc(10*sizeof(int);
                    /* memory address may be the value of pt */
pt[0] = 7;          /* may result in segmentation error */
```

자주 저지르는 실수 - 4

- Pointer를 malloc없이 사용한다.

(예1)
```
int *pt = 0;
pt[0] = 7;
```

(예2)
```
int *pt;
pt[0] = 7;
```

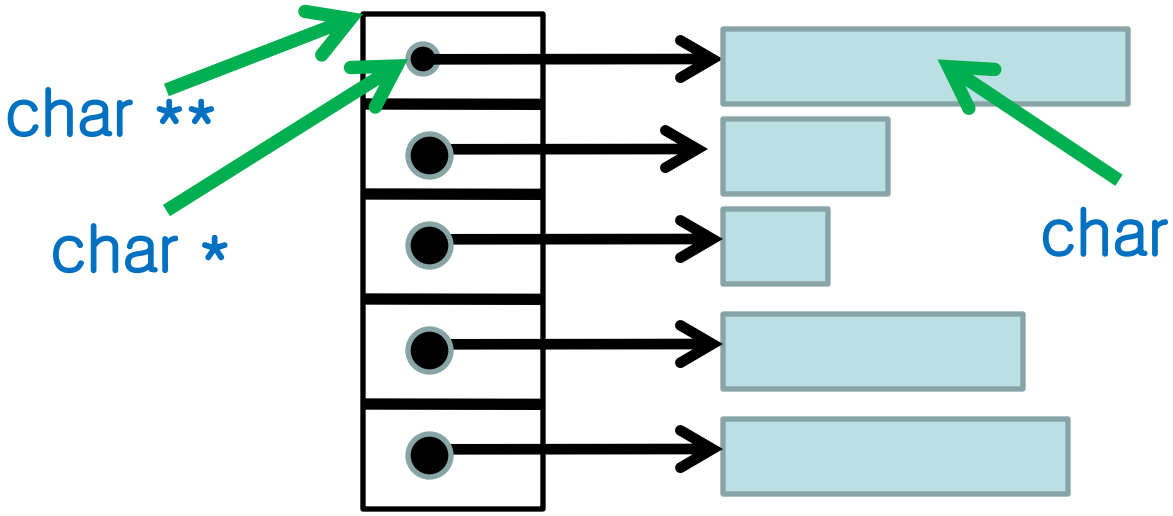
Array of Names

- 5사람의 이름을 다루려고 한다. 방법?
- Two dimensional array

```
char names[5][30];
strcpy(names[0], "H.B. Min");
strcpy(names[1], "Johann S. Bach");
……
```

- What's the Problem?
  - 사람의 숫자가 변한다면?
  - 이름 길이가 다르므로 memory 낭비

Table of names
Another Two-dim. Array



메모리는 더도 말고 덜도 말고 꼭 필요한 만큼만 사용해야 합니다. 고정 크기를 사용하지 마세요.

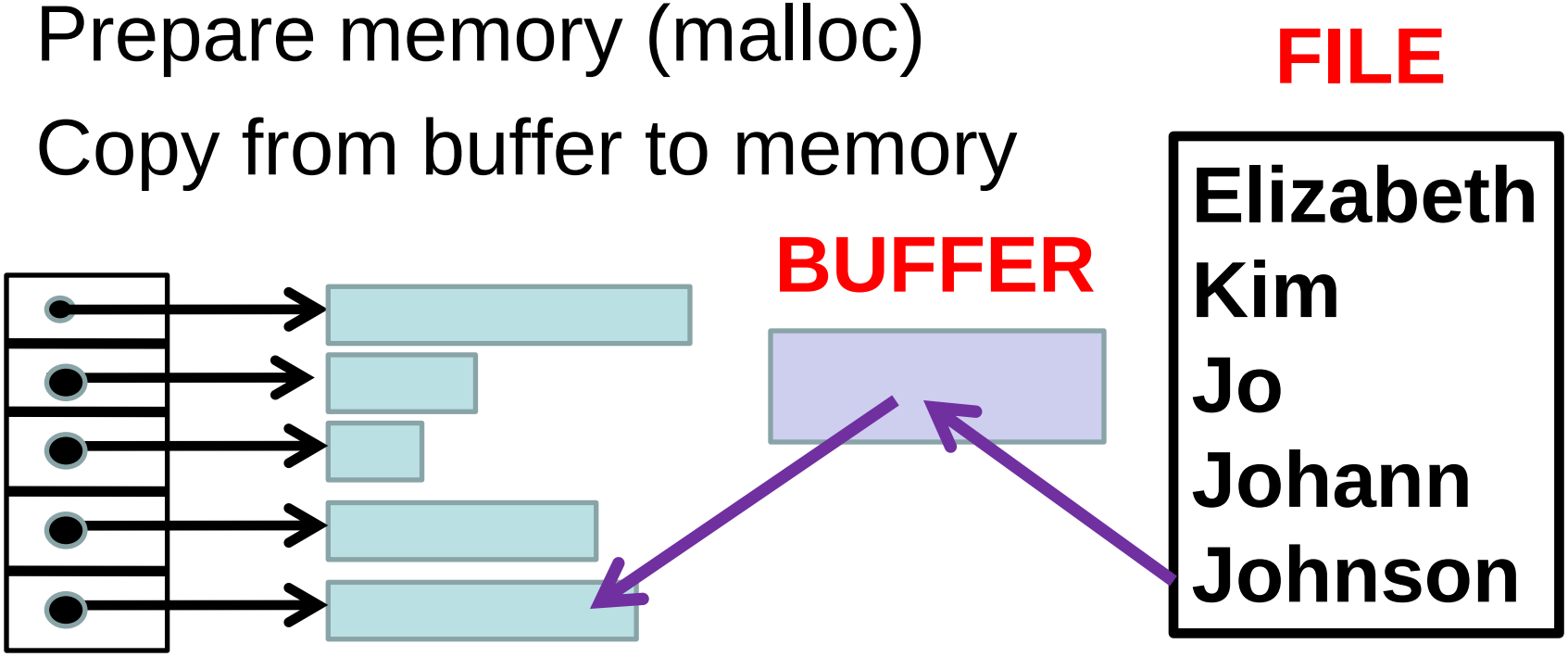
```
char **names = (char **)malloc(N*sizeof(char *));
names[0] = (char *)malloc((len1+1)*sizeof(char));
strcpy(names[0], "Name 0");
names[1] = (char *)malloc((len2+1)*sizeof(char));
strcpy(names[1], "Name 1");
.....
.....
```

String Buffer (1)

- 다수의 string을 **파일**에서 읽기 위한 임시 저장소 1개
- String data의 특성에 따라 다르지만, 0.5 ~ 8Kbytes의 고정 크기
- String data는 필요한 memory가 제각각이므로, 사용 메모리 최적화를 위하여 buffer가 필요함.

String Buffer (2)

- Read a name from file to string buffer
- Measure length (strlen)
- Prepare memory (malloc)
- Copy from buffer to memory



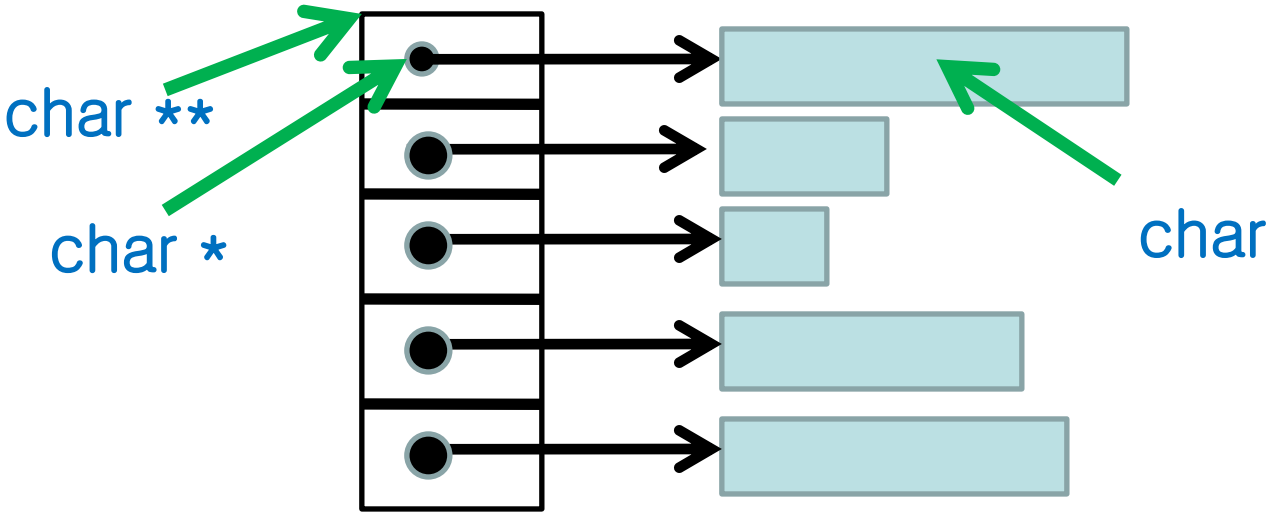
설계할 프로그램

- 학생 6명의 "이름/성별/4과목-점수"가 들어 있는 text file (**infile.txt**)가 주어진다. (6, 4는 command line argument에서 읽어 들임)
- 이 text file을 읽어서, 그 내용과 4과목 점수의 평균값(double)을 binary file에 저장한다 (**binary.dat**)
- 방금 저장한 binary file을 읽어서, 그 binary file의 크기를 측정하여 printf()를 이용하여 출력하고, binary file의 내용은 다시 text file에 저장한다. (**outfile.txt**)

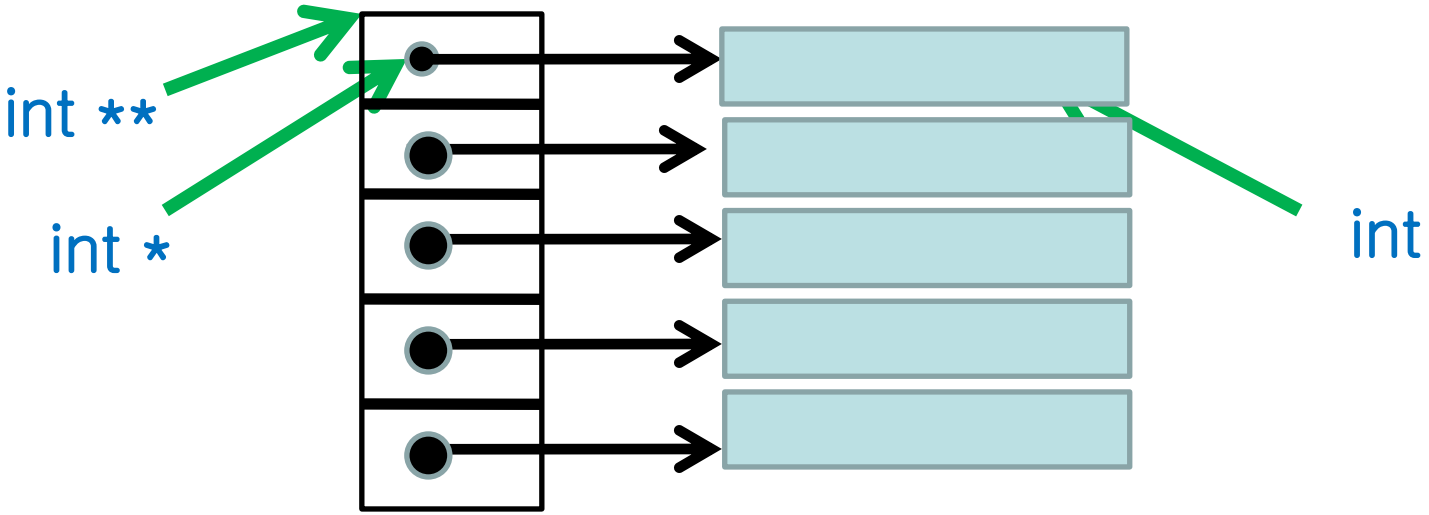
Text file => Binary file

- **readTXT( )** : text file (**infile.txt**)을 읽어서 데이터 array (name_array, gender_array, grades_array)에 저장한다.
- **writeBIN( )** : data array에 저장된 데이터와 각 학생의 점수 평균 (double)을 binary file에 저장한다 (**binary.dat**)

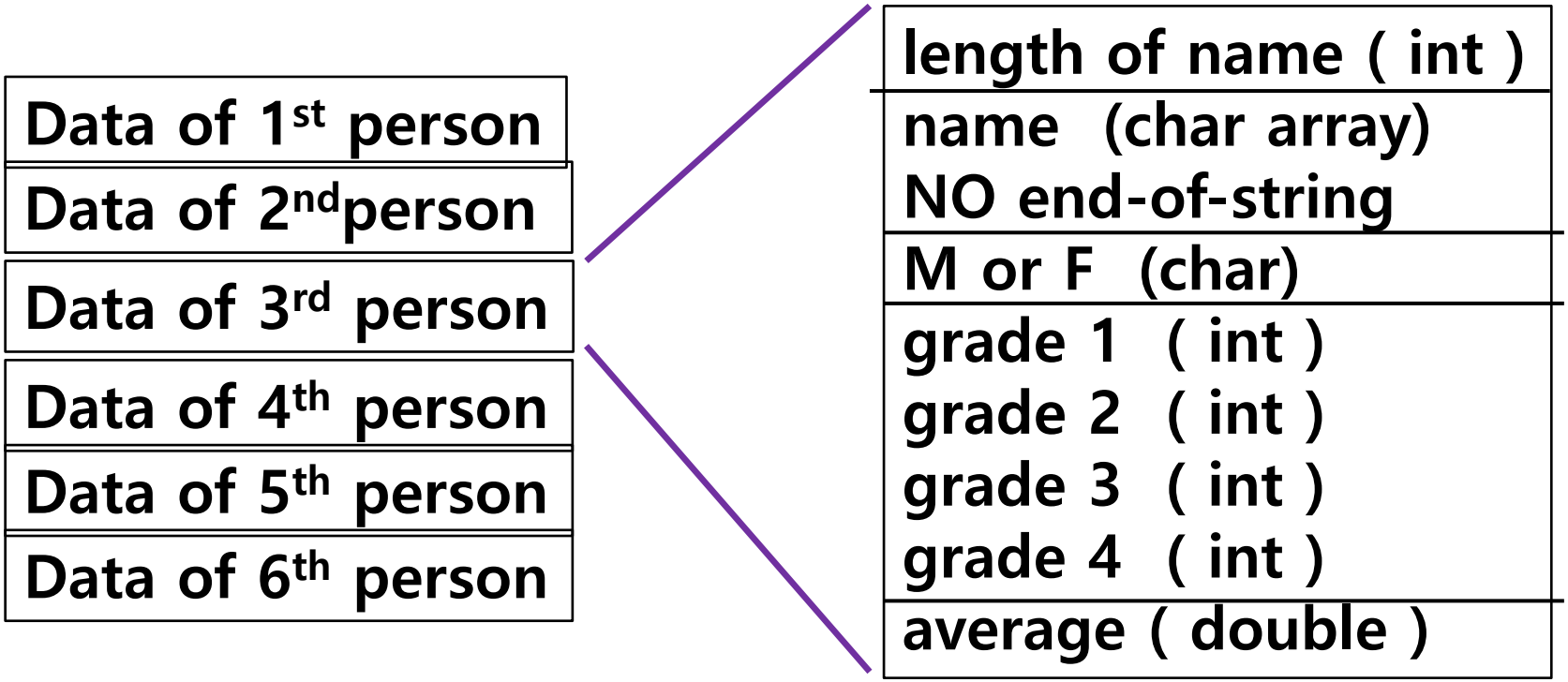
char **name_array



int **grades_array



Binary File Format



Binary file => Text fille

- readBIN( ) : binary file (**binary.dat**)을 읽어서, 그 파일의 크기를 printf()로 출력하고, 파일의 내용은 데이터 array (name_array, gender_array, grades_array, **average_array**)에 저장한다.
- writeTXT( ) : 데이터 array에 저장된 데이터 (이름, 성별, 4과목 점수, 평균점)을 새로운 text file에 저장한다 (**outfile.txt**)

Output text file (outfile.txt)

- outfile.txt는 infile.txt와 같은 형식으로 하되, 각 줄 맨 뒤에 평균값을 추가함. 즉,

- 1줄에 1명의 데이터를 출력함
- 한 사람의 모든 데이터는 공백 문자 1개로 구분하여 출력함
- 4과목 점수의 평균 값은 **소수점 이하 2자리**까지 나오도록 줄의 맨 끝에 추가함.

Program의 가정 (assumptions)

다음의 사항 들을 알고 있다고 가정한다

- 사용하는 PC는 little-endian CPU를 사용한다.
- 입력 Text file에 6명 학생의 이름, 성별 (M or F), 4과목의 성적 점수가 1줄에 1명씩 들어있다. 6과 4는 command line argument로 읽어 들입니다.

  int main(int argc, char *argv[])
- 이름의 길이는 사람마다 다르며, 100글자 미만이며, text file 1줄의 길이는 200 글자 미만이다.
- 성별은 1글자로 'M' 아니면 'F'이다.
- 4과목 성적은 정수(integer, type "int"를 사용)이다.
- 이름, 성별, 각 과목 점수 사이는 하나 이상의 blank character (' ') 로 구분되어 있다.

q1.c

```
int main(int argc, char **argv) {
    char *input_file  = "infile.txt";   // input file name
    char *binary_file = "binary.dat";   // binary file name
    char *output_file = "outfile.txt";  // output file name

    char **name_array;        // array of names
    char *gender_array;       // array of genders
    int  **grades_array;      // array of arrays of grades
    double *average_array;    // array of averages

    int num_persons = get_positive_integer(argv[1]);
    int num_grades = get_positive_integer(argv[2]);
```

q1.c (txt => bin)

```
// Prepare memory for data storage
buildMemory(&name_array, &gender_array, &grades_array, 0);

// Read data from text file into arrays.
if (!readTXT(input_file, name_array, gender_array, grades_array))
    return -1;
// Write data in arrays to binary file.
if (!writeBIN(binary_file, name_array, gender_array, grades_array))
    return -2;

// Dispose memories
disposeMemory(name_array, gender_array, grades_array, 0);
```

q1.c (bin => txt)

```
buildMemory(&name_array, &gender_array, &grades_array,
    &average_array);
// Read binary file created by "writeBIN()"; put in data arrays.
if (!readBIN(binary_file, name_array, gender_array,
                          grades_array, average_array))
    return -3;
// Write data at arrays to text file.
if (!writeTXT(output_file, name_array, gender_array,
                           grades_array, average_array))
    return -4;
disposeMemory(name_array, gender_array, grades_array,
    average_array);
```

grade.h / grade.o

- 채점을 위한 function 들이 들어 있으며, program을 compile할 때, GRADE가 define 되어있어야 채점을 제대로 합니다.

  #define GRADE

  - string.c에 이미 이렇게 되어 있음
- **grade.o** : 채점용 function들이 들어 있는 source code "grade.c"를 compile한 object file입니다. 학생들이 program한 code와 link해서 사용해야 합니다. (grade.c는 비공개)

주의 사항 (1)

- Dynamic memory allocation에 memcheck.h에 정의된 **mymalloc()과 myfree()만을 사용합니다**.
- File open/close에 memcheck.h에 정의된 **myfopen()과 myfclose()만을 사용합니다**
- **main()에 들어 있는 code를 변경할 수 없습니다.** 단, code style과 comment는 얼마든지 변경할 수 있고 문제가 있다면 변경해야 합니다.
- 이 사항들은 별도로 다른 지시를 하는 경우가 아니라면, **이번 실습 뿐만 아니라 그 이후의 모든 실습에서도 같습니다.**

주의 사항 (2)

- 이 실습부터 이후 **모든 실습**에서, **크기가 큰 (예:10개 이상) 고정된 크기의 array를 사용하는 것을 금지**합니다

```
int kindOf[3];          // This is okay
#define ARR_SIZE  100
int array[ARR_SIZE];    // This is prohibited

int *array;
array = (int*)malloc(array_size*sizeof(int));
```

주의 사항 (3)

이 실습부터 이후 **모든 실습**에서

- **float를 사용하지 마세요**. float를 써야하는 경우가 있으면 **double을 사용해야 합니다**.
- **Global variable을 사용하지 마세요.**
- **주어진 program의 구조를 바꾸지 마세요. 변수를 추가하는 정도의 변경을 허용하지만, 이미 있는 function을 제거하는 등의 구조 변경을 금지합니다. 이번 실습의 경우, 이미 있는 code는 code style의 변경 외에는 절대로 변경할 수 없습니다.**

주의 사항 (4)

- Program은 **우아하게 종료되어야 합니다** (Programs should exit graciously)
- 이는 이후의 **모든 실습**에서 같습니다.
- scanf/printf, fscanf/fprintf, fread/fwrite
- file open (fopen)
- dynamic memory alloc (malloc/calloc)
- etc, …

우아한 종료 방법

- Check return value

```
int *array = (int *)malloc( … );
if(!array) {
   fprintf(stderr, "[ERROR] …");
   // return or exit
}
```

- Use "assert" macro

```
#include <assert.h>
int *array = (int *)malloc( … );
assert(array);
```