[DASF004-42]

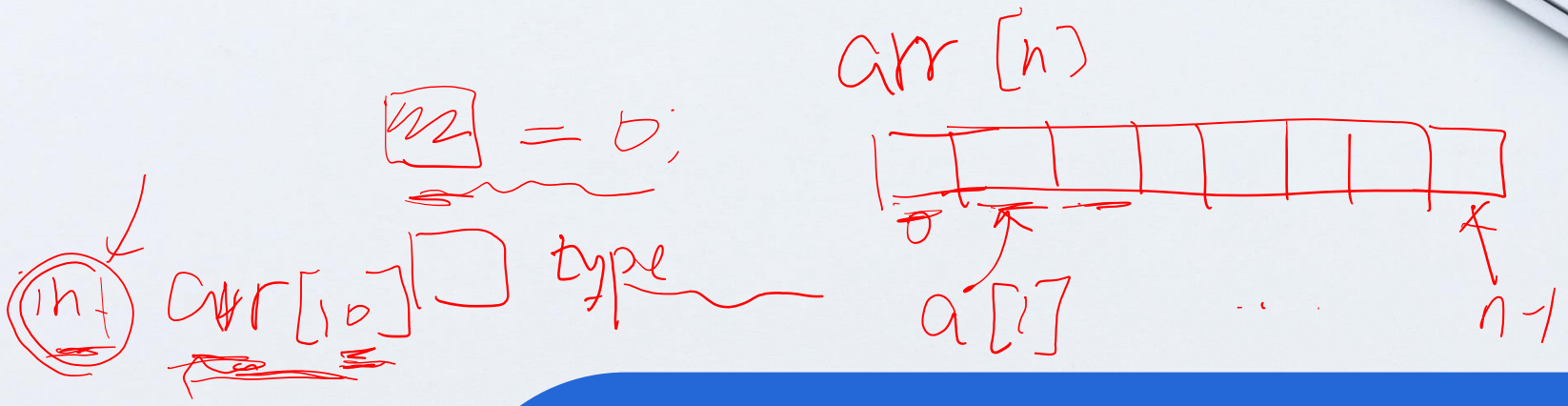
Basis and Practice in Programming

(프로그래밍 기초와 실습)

Spring 2022

HYUNGJOON KOO

SUNG KYUN KWAN UNIVERSITY(SKKU)



Passing an Array to a Function

Passing Arrays to Functions

- **Arrays can be used as parameters.**
  - Usage of a one-dimensional array as a parameter

```
int f(int a[])  // no length specified
{
        …
}
void main()
{
        int a1[100] = {1,2,3};
        f(a1);
}
```

address

- **In function f, how do we know the proper length of a[]?**

Passing Arrays to a Function with a Size

- Since `sum_array` needs to know the actual length of a, we must supply it as a second argument.

```
int sum_array(int a[], int n) {          // n: size of a
   int i, sum = 0;
   for (i = 0; i < n; i++)
      sum += a[i];
   return sum;
}

void main() {
   int a1[100] = {1,2,3};
   int sum = sum_array(a1, 100);
}
```

Passing Arrays to a Function with a Symbolic Constant

- When the array size is globally known in advance and fixed during the execution, we can use a symbolic constant for it.

```
#define SIZE 100
int sum_array(int a[]) {
    int i, sum = 0;
    for (i = 0; i < SIZE; i++)
        sum += a[i];
    return sum;
}
void main() {
    int a1[SIZE] = {1,2,3};
    int sum = sum_array(a1);
}
```

Passing Arrays to a Function with the *sizeof* Operator

- When the array size is globally known in advance and fixed during the execution, we can use a symbolic constant for it.

```
int sum_array(int a[], int n) {
    int i, sum = 0;
    for (i = 0; i < n; i++)
        sum += a[i];
    return sum;
}
void main() {
    int a1[] = {1,2,3};
    size_t size = sizeof(arr) / sizeof(arr[0]);
    int sum = sum_array(a1, size);
}
```

Printing an Array with a Function Call

SUNG KYUN KWAN UNIVERSITY(SKKU)

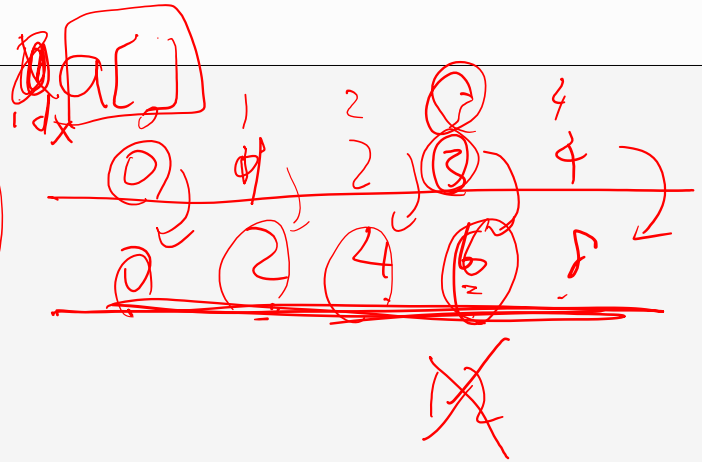
- **Printing an array by passing it**
  - With printArray(), we can check array elements multiple times by calling it

```
void printArray(int a[], size_t size)
{
  for(size_t i = 0; i < size; i++) {
    char c;
    c = (i==size-1)? '\n':' ';      // for the pretty output
    printf("%d%c", a[i], c);
  }
}

int main()
{
  int nums[SIZE];
  for( size_t i = 0; i < SIZE; i++ )
    nums[i] = i;                    // initializing the array
  printArray(nums, SIZE);
  return 0;
}
```

Modifying an Array Element

- Consider the following two functions and function calls for them.

```
void modifyArray(int a[], size_t size) {
  for(size_t i = 0; i < size; i++)
    a[i] *= 2;
}

void modifyElement(int e) {
  e *= 2;
}

(Somewhere in the code)
modifyArray(a, SIZE);          // a is an array
modifyElement(a[3]);           // calling with an element
```



Modifying an Array Element

SUNG KYUN KWAN UNIVERSITY(SKKU)

- Consider the following two functions and function calls for them.

```
void printArray(int a[], size_t size)
{
  for(size_t i = 0; i < size; i++) {
    char c;
    c = (i==size-1)? '\n':' ';   // for the pretty output
    printf("%d%c", a[i], c);
  }
}

int main()
{
  int nums[SIZE];
  for( size_t i = 0; i < SIZE; i++ )
    nums[i] = i;                  // initializing the array
  printArray(nums, SIZE);
  modifyArray(nums, SIZE);
  printArray(nums, SIZE);
  modifyElement(nums[3]);
  printArray(nums, SIZE);
  return 0;
}
```

Call-by-reference VS Call-by-value

```
(Somewhere in the code)
modifyArray(a, SIZE);          // a is an array
modifyElement(a[3]);           // calling with an element
```

- **We can see that the array is modified with the first function call, but no modification occurs with the second one.**
- **Call-by-reference or pass-by-reference**
  - When we pass an array to a function, we can modify elements of the array, and this mechanism is called as 'call-by-reference'.
  - It is about the addresses of variables, which is related to pointers. *(will be discussed later.)*
- **Call-by-value or pass-by-value**
  - Other data types rather than arrays or pointers, only values are copied into the function via parameters, which are local variables for functions.
  - Values in the callers' variables cannot be modified with this mechanism.

Hands-on Lab

- sum_array1.c
- sum_array2.c
- sum_array3.c
- arr_print_fn.c
- call_by_ref_vs_call_by_val.c

[DASF004-42]

Basis and Practice in Programming

(프로그래밍 기초와 실습)

Spring 2022

HYUNGJOON KOO

SUNG KYUN KWAN UNIVERSITY(SKKU)

다차원 배열 : 2차원

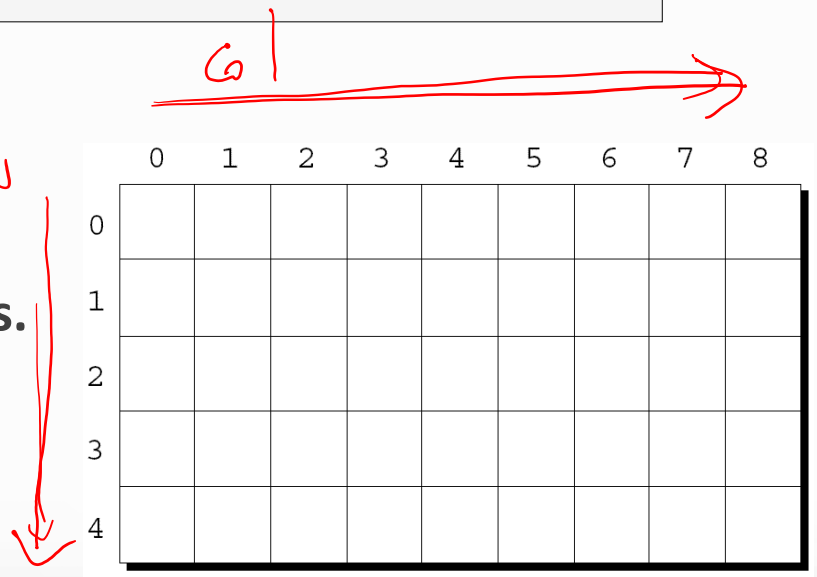
Multidimensional Arrays

Multidimensional Arrays

- **Arrays in C can have multiple indices.**
- **The following declaration creates a two-dimensional array**
  - a table or a matrix, in mathematical terminology

```
int m[5][9];
```

  - m has 5 **rows** and 9 **columns**. (called 5-by-9 array)
  - Both rows and columns are indexed from 0.



- **Multidimensional arrays can have more than two indices.**

Initializing a Multidimensional Array

- **Review: Initialization of an array with declaration**

```
int m[5] = {1, 1, 1, 1, 1};
```

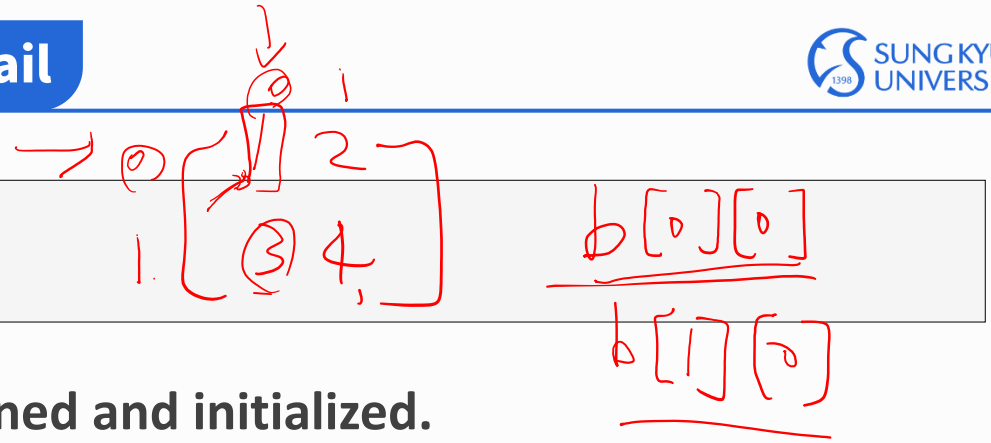
type name # init.val

- **Initializing multidimensional arrays**
  - Similar to one dimensional arrays
  - Nesting one-dimensional initializers
  - The values are grouped by row in braces.

```
// "5 rows, 9 columns" means that each row has 9 elements.
// We have 5 one-dimensional initializers with 9 elements.
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

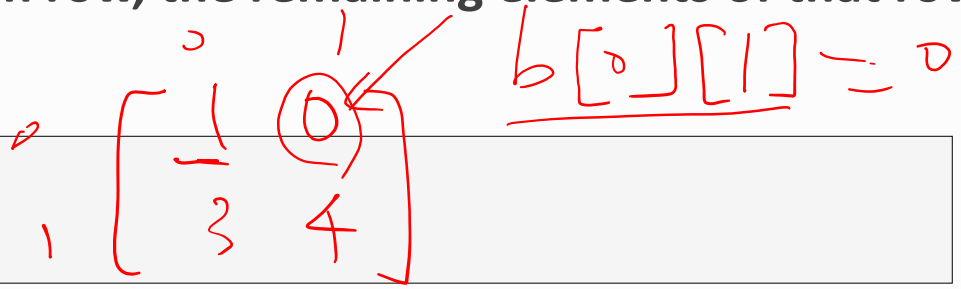
Initializing a Multidimensional Array in Detail


```
int b[2][2] = {{1, 2}, {3, 4}};
```



- **A two-dimensional array int b[2][2] is defined and initialized.**
  - The values in the first braces initialize row 0 and the values in the second braces initialize row 1.
  - 1 and 2 are for b[0][0] and b[0][1], respectively.
  - 3 and 4 are for b[1][0] and b[1][1], respectively.
- **If there are not enough initializers for a given row, the remaining elements of that row are initialized to 0.**

```
int b[2][2] = {{1}, {3, 4}};
```

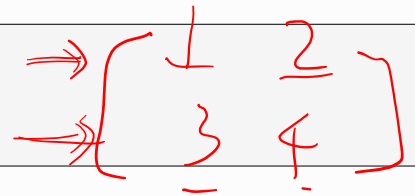


- **This code would initialize b[0][0] to 1, b[0][1] to 0, b[1][0] to 3 and b[1][1] to 4.**

Initializing a Multidimensional Array in a different way

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
int b[2][2] = {1, 2, 3, 4};
```

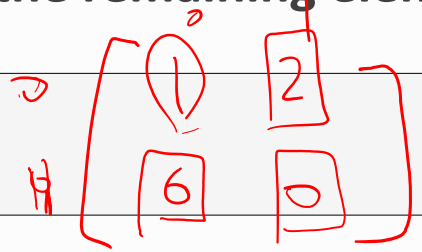


- **We can initialize a multidimensional array without inner braces.**
  - Arrays occupy contiguous memory space.
  - The values in the initializer will be placed as in order.
  - First two values are for the first row and the next two are for the second row.

a [row][col]

- **If there are not enough initializers, the remaining elements are initialized to 0.**

```
int b[2][2] = {1, 2};
```



- **Above code initializes b[0][0] to 1, b[0][1] to 2, b[1][0] to 0 and b[1][1] to 0.**

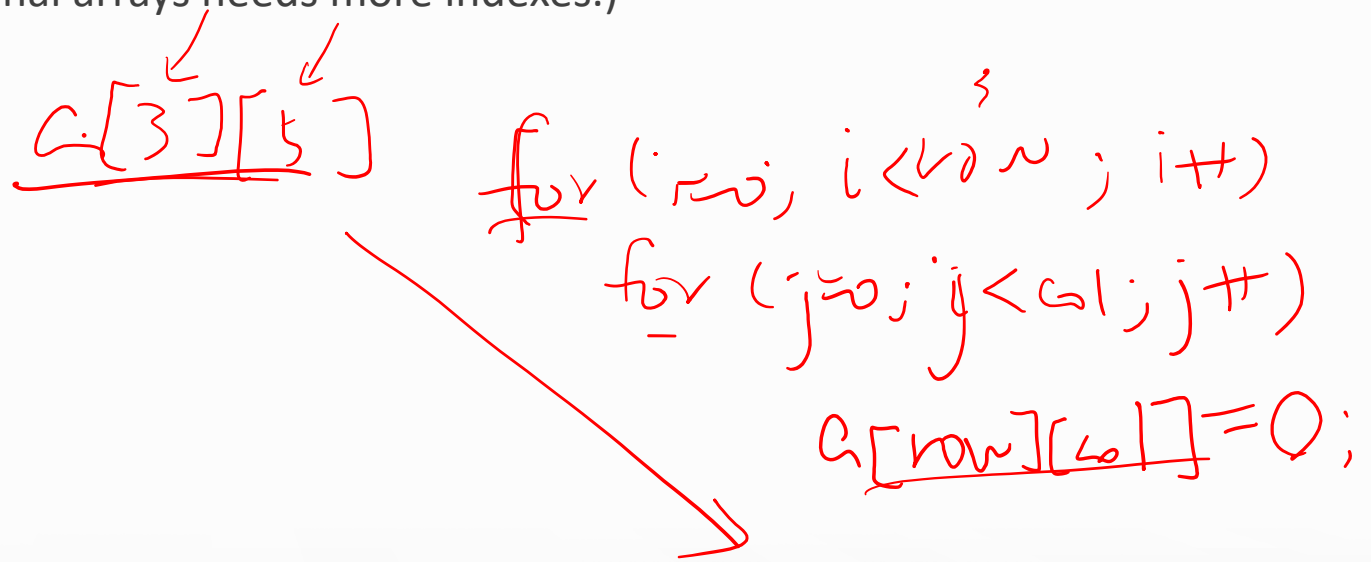
Traversing Multidimensional Arrays

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **For loops are used for traversing arrays.**
  - For loops match to array operations.
  - It naturally gives us indexes for traversing.

- **Multiples indexes for multidimensional arrays**
  - For rows and columns (Higher dimensional arrays needs more indexes.)

- **Nested for loops**
  - for loops inside of a for loop



Traversing Multidimensional Arrays

SUNG KYUN KWAN UNIVERSITY(SKKU)

```
total = 0;
for (size_t row = 0; row <= 2; ++row) {
   for (size_t column = 0; column <= 3; ++column) {
      total += a[row][column];
   }
}
```

- The for statement totals the elements of the array one row at a time.
- The variable 'row' and 'column' are used for fixing rows and columns to visit.
- When the nested for statement terminates, `total` contains the sum of all the elements in the array a.

Count 1s in the Array

DEMO

- **With a given two-dimensional array, make a program that count the occurrences of 1.**
  - A nested loop can be the best choice for this problem.
  - If you want to count the occurrences on each line, how can we modify the program?

```c
#include <stdio.h>

int main()
{
  int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1}, …};

  int noo = 0;
  for(int i=0; i<5; i++)
    for(int j=0; j<9; j++)
      if(m[i][j] == 1)
        noo++;

  printf("%d\n", noo);
  return 0;
}
```

```c
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

Multidimensional Arrays as Arguments

- **REVIEW: Passing a single dimensional array to a function**
  - The size of the array should be specified.

- **Similar thing happens here, too.**
  - First dimensional size can be omitted, but with passing the number of rows.
  - Subsequent sizes are **mandatory**.

```
void PrintArray (int m[][9], int rows){
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < 9; j++) {
            printf("%d ",  m[i][j]);
        }
        printf("\n");
    }
}
```

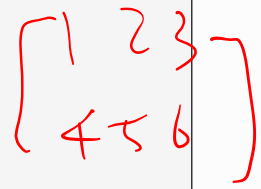
Multidimensional Array Example

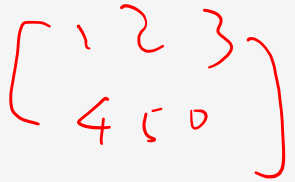
DEMO

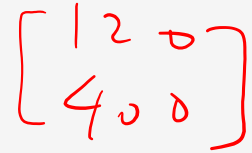
- **Example code for 'passing arrays to functions' and 'nested initializers'.**
  - Variation will be done on the initializers.
  - printArray function is working but not properly written. What is wrong with it?

```c
void printArray(int a[][3])
{
  for (int i=0; i<=1; i++) {
    for(int j=0; j<=2; j++) {
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  return;
}
```

```c
int main() {
  int array1[2][3] = {{1,2,3}, {4,5,6}};
  printArray(array1);

  int array2[2][3] = {1,2,3,4,5};
  printArray(array2);

  int array3[2][3] = {{1,2}, {4}};
  printArray(array3);

  return 0;
}
```







- `sizeof` **determines the size in bytes of an array (or any other data type) .**
  - A special unary operator for querying size of the object or type.
  - Used only when actual size of the object is known.

- **Use case:**

```
int a[10] = {};
// assuming 4 bytes for integers
// sizeof(a) : 40
// sizeof(a[0]) : 4

// To get the number of elements
int num_of_elements = sizeof(a)/sizeof(a[0]);
//int num_of_elements = sizeof(a)/sizeof(int);
```

- **Assuming a function receives an array of 20 elements as an argument …**
  - The function's parameter is simply a pointer to the array's first element.
    - » Pointer is a data type for storing addresses, which will be dealt later.
  - When you use sizeof with a pointer, it returns the size of the pointer.
    - » Not the size of the item to which it points.

```
void PrintArray (int m[][9], int rows){
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < 9; j++) {
            printf("%d ",  m[i][j]);
        }
        printf("\n");
    }
    printf("size: %lu\n", m); // will print 4(32bit) or 8(64bit), size of pointer
}
```

Exercise: Identity Matrix

DEMO

- **Fill in the code to make an *identity matrix*.**
  - Initializing a two-dimensional array with nested for loops
  - `sizeof` will be tested in this demo.

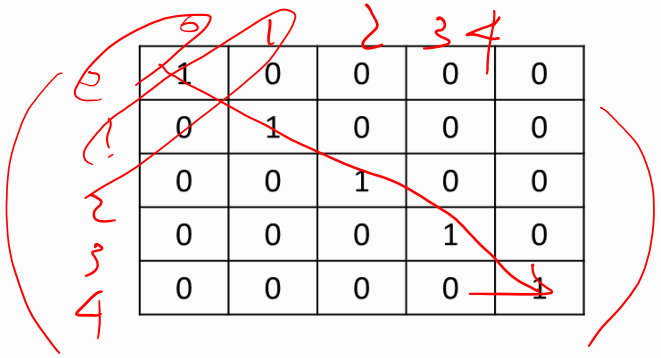
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

```
#define N 5
double e_mat[N][N];
for (int row = 0; row < N; row++) {
    for (int col = 0; col < N; col++) {

            Fill in here


    }
}
```

Exercise: Identity Matrix (Solution)

DEMO

- **Fill in the code to make an *identity matrix*.**
  - Initializing a two-dimensional array with nested for loops
  - `sizeof` will be tested in this demo.

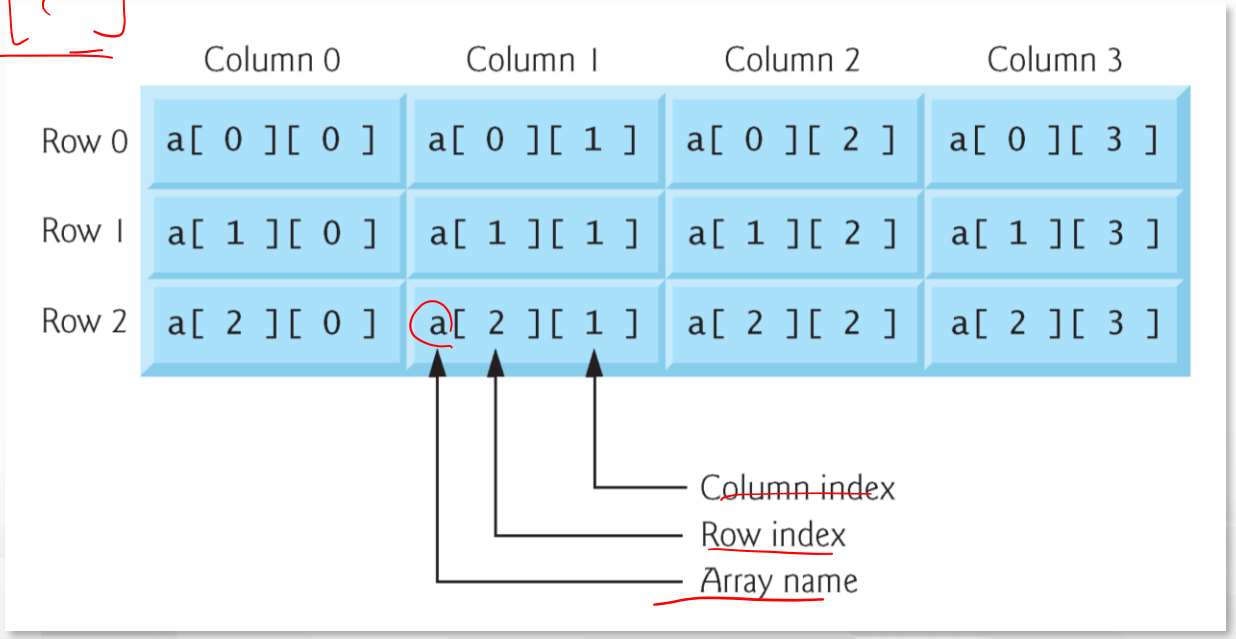


1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

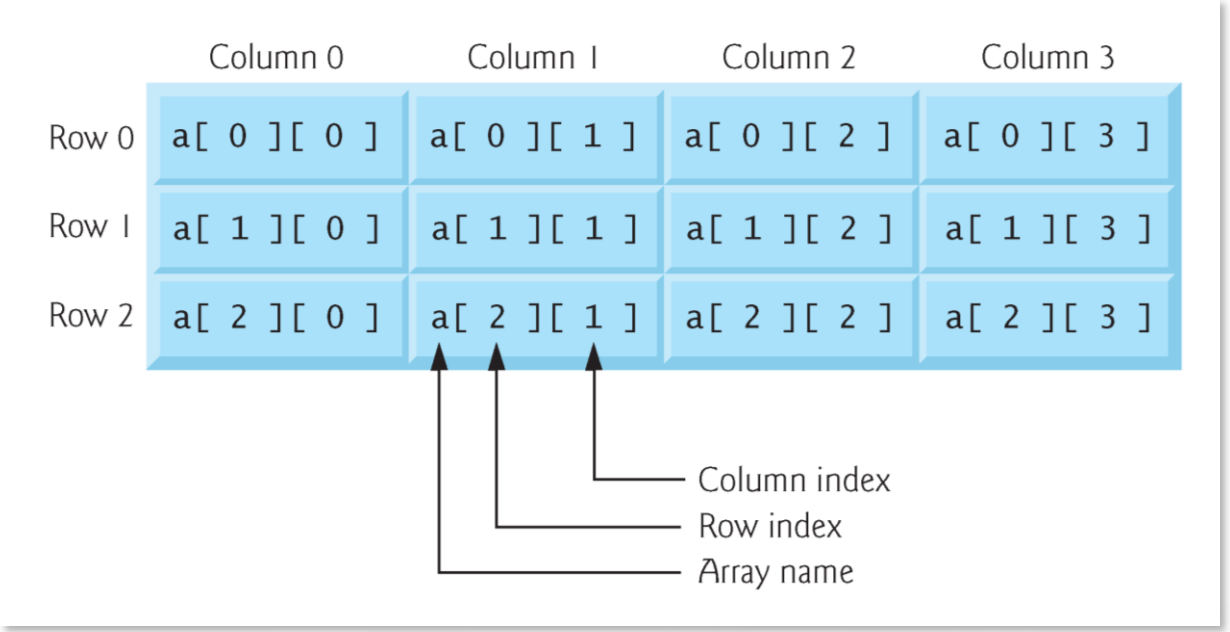
```
#define N 5
double e_mat[N][N];
for (int row = 0; row < N; row++) {
    for (int col = 0; col < N; col++) {
            if (row == col)
                e_mat[row][col] = 1.0;
            else
                e_mat[row][col] = 0.0;
    }
}
```

Reading Materials

- Arrays in C can have multiple indices.
- A common use of **multidimensional arrays** is to represent **tables** of values consisting of information arranged in *rows* and *columns*.
- To identify a particular table element, we must specify two indices:
  The first (by convention) identifies the element's *row* and the second (by convention) identifies the element's *column*.
- Tables or arrays that require two indices to identify a particular element are called **two-dimensional arrays**.
- Multidimensional arrays can have more than two indices.



- The figure illustrates a two-dimensional array, a. The array contains three rows and four columns, so it's said to be a 3-by-4 array.
- In general, an array with *m* rows and *n* columns is called an ***m*-by-*n* array**



Hands-on Lab

- `multi_array_init_traverse.c`
- `multi_array_count.c`
- `identity_matrix.c`

[DASF004-42]

Basis and Practice in Programming

(프로그래밍 기초와 실습)

Spring 2022

HYUNGJOON KOO

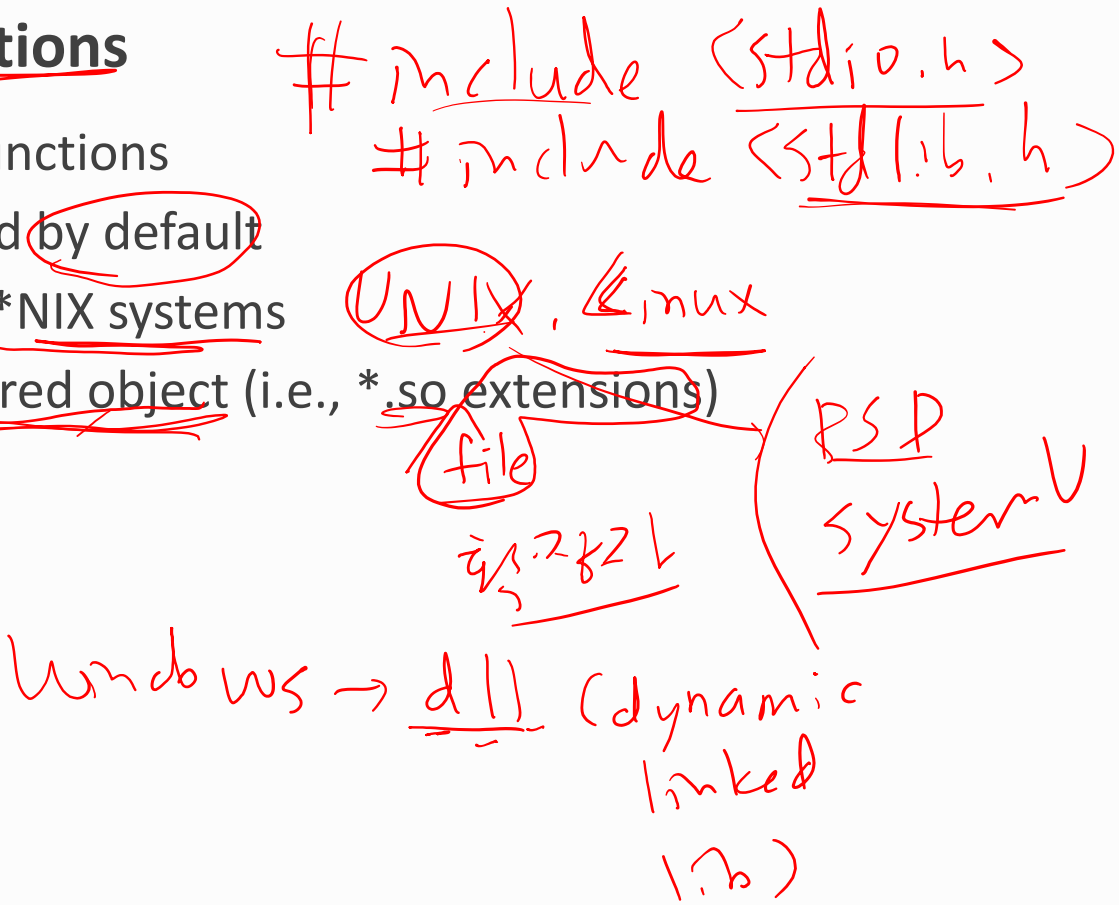
SUNG KYUN KWAN UNIVERSITY(SKKU)

C Library

C Standard Library

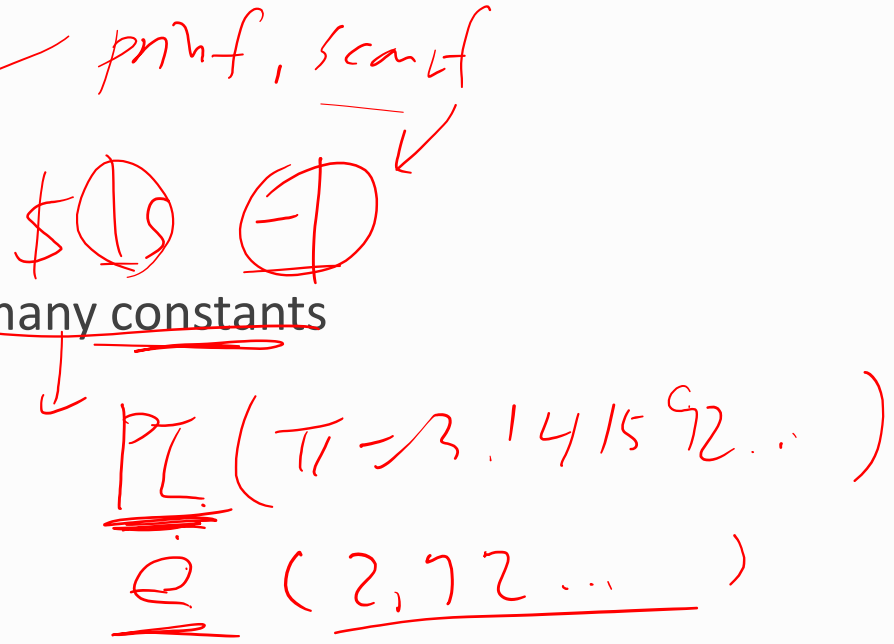
- **Library is a collection of built-in functions**
  - Include a header to use standard library functions
  - Provide many useful functionalities in need by default
  - Headers are located in `/usr/include/` in *NIX systems
  - In Linux, a library can be provided as a shared object (i.e., *.so extensions)

- **Advantages**
  - Robustness
  - Optimized functions for performance
  - Avoid reinventing the wheel



Library Functions and C Headers

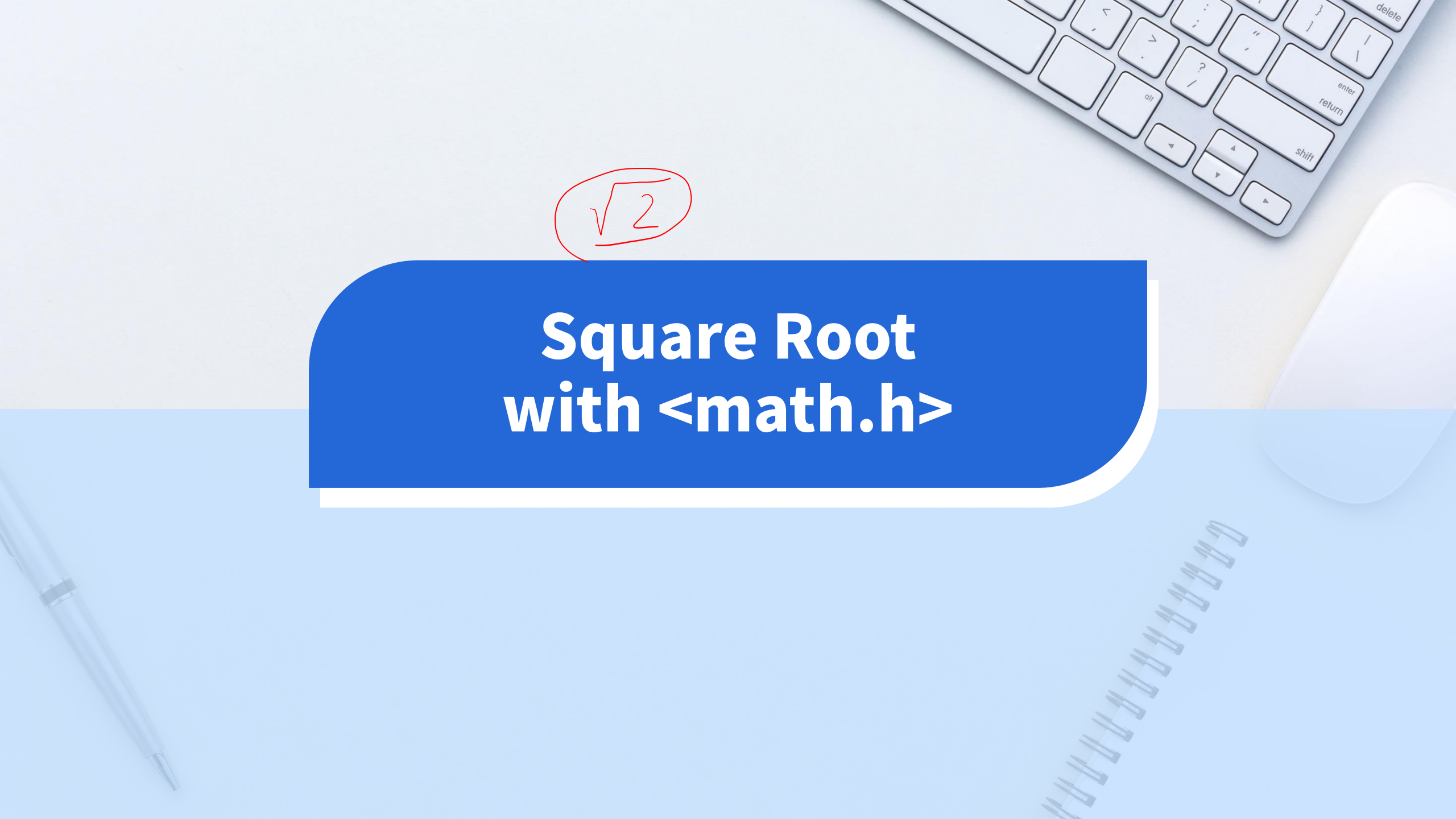
- **C Library function and its header**
  - <stdio.h> Standard input/output functions
  - <stdlib.h> Standard utility functions
  - <stdarg.h> Argument handling functions
  - <math.h> Mathematics functions; defining many constants
  - <ctype.h> Character type functions
  - …



Compilation with C Library

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **When you compile a C program, the "libc.so" library is provided by default**
  - In Linux, GNU C library would be located in `/usr/lib/x86_64-linux-gnu/*.so`
  - Your compiler (gcc) will use "libc-[ver].so" by default



Square Root with <math.h>

Square Root Function

- **The sqrt function**
  - Defined in the “math.h" header file
  - Computes a square root of a float

- **Example**

```
root = sqrt(num);
```

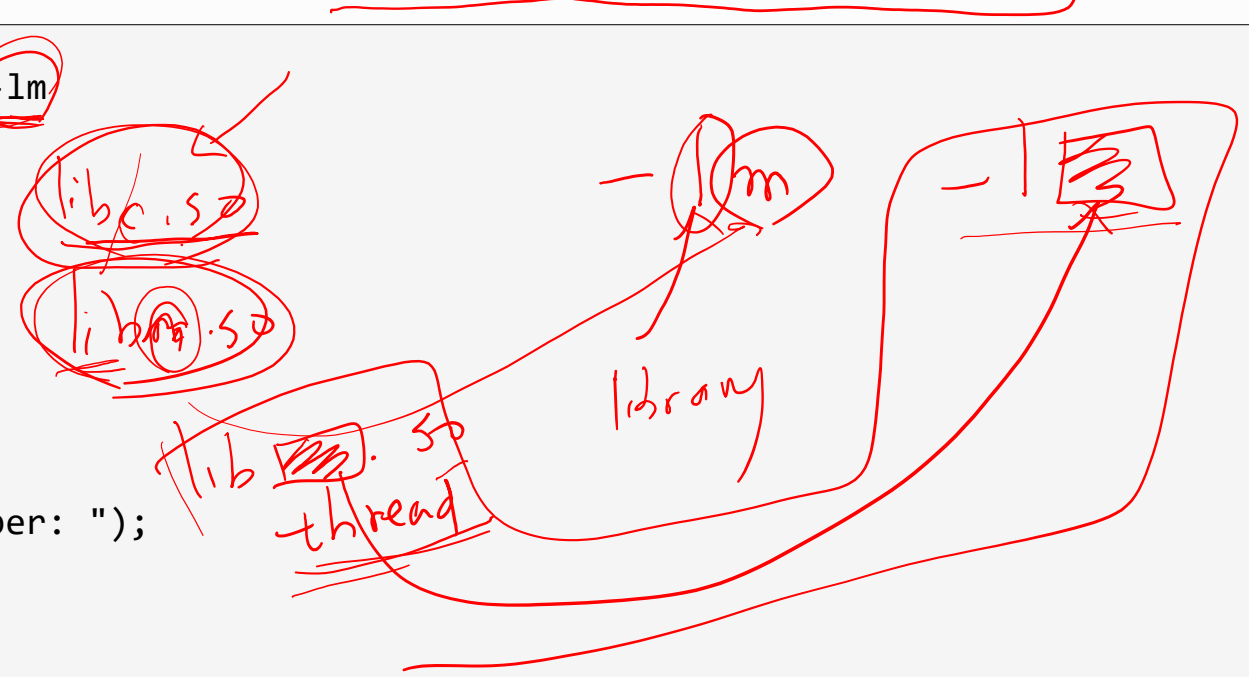
  - root will have a square root value of num
  - Probably getting an error while compilation without providing a proper shared object
    - » Hint: Include the C math library during compilation

Square Root Computation

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **A simple program that computes a square root of N**
  - We get an error message with "`undefined reference to `sqrt'`"

```
// gcc -o sqrt sqrt.c -lm
#include <stdio.h>
#include <math.h>

int main() {
    float num, root;
    printf("Enter a number: ");
    scanf("%f", &num);
    root = sqrt(num);
    printf("Square root of %.2f = %.2f\n", num, root);
    return 0;
}
```



Random Number Generation with <stdlib.h>

Rand function

SUNG KYUN KWAN UNIVERSITY(SKKU)

The rand function

- is defined in the "stdlib.h" header file
- generates an integer between 0 and RAND_MAX
  - RAND_MAX: a symbolic constant defined in the <stdlib.h> header

Example

```
i = rand();
```

- i will have a random value between 0 and RAND_MAX
- CHECK: How can we make i to have a random value between 1 and 6?
  - Hint: Use the remainder operator.

Rolling a Six-Sided Die


```
i = rand() % 100;
```

- **Generating a random number in the range of [0, 100).**
  - One typical example with the rand function
  - This technique with the remainder operator is called scaling.
    - The number 100 is called the scaling factor.

- **Simulating a dice**
  - Use the scaling factor of 6.
  - Shift the range of numbers produced by adding 1.

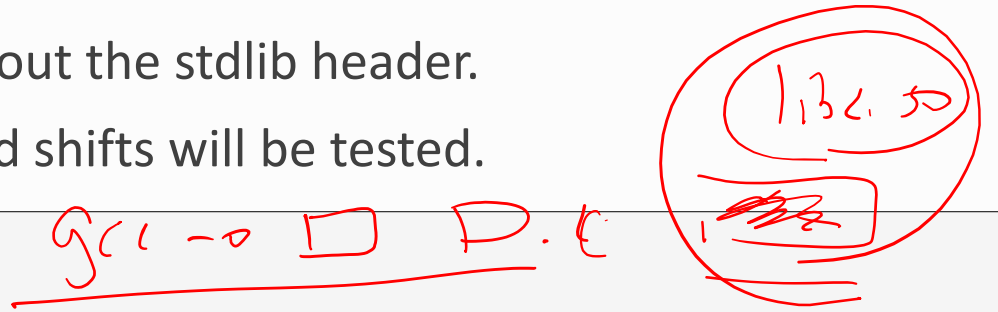
```
face = rand() % 6 + 1;
```

Rolling a Six-Sided Die

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **A simple program that prints out 20 numbers in the range of [1, 6].**
  - We can see a warning without the stdlib header.
  - Different scaling factors and shifts will be tested.

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

int main() {
    for (int i = 0; i < 20; i++)
        printf("%d ", 1 + rand() % 6);
    printf("\n");

    return 0;
}
```



Randomizing the Random Number Generator

PRNG

Repeatability of rand function

- You can observe that the execution results are always the same in the previous demo.
  - The same sequence of random numbers
  - Calling rand repeatedly produces a sequence of numbers that appears to be random. (Pseudorandom numbers)

- When debugging a program, this repeatability is essential for proving that corrections to a program work properly.
  - With different sequences of random numbers, every execution will have a different result which make it very hard to correct errors.

Randomizing the sequences

- Can be accomplished with the standard library function `srand`.

srand in stdlib.h

- **srand takes a seed.**
  - Seed: an unsigned integer argument
  - A sequence of random numbers are generated based on the value of the seed.
  - With the same seed values, the random sequences will be the same.
    - » Without calling `srand`, a default value will be used.

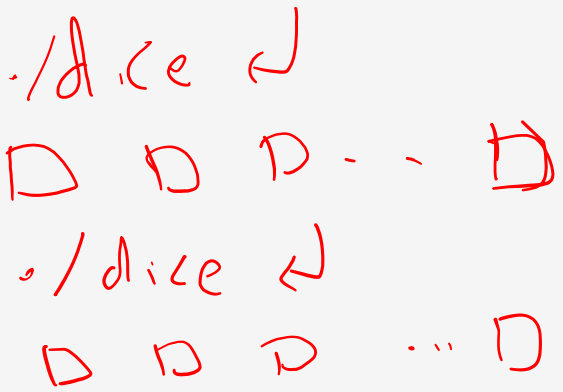
- **`srand(time(NULL));`**
  - To randomize without entering a seed each time
  - This make the program to use the current time, which will be differed on every execution.
  - The function `time` is defined in `time.h`.

Rolling a Six-Sided Die 2

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **A simple program that prints out 20 numbers in the range of [1, 6].**
  - Random sequence on each execution.
  - We can set a specific seed for srand function for debugging purposes.

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


int main() {
    srand(time(NULL));
    for (int i = 0; i < 20; i++)
        printf("%d ", 1 + rand() % 6);
    printf("\n");
    return 0;
}
```



Library

- sqrt.c
- rand.c
- srand.c
- arr_print_rand.c