Examples
Example

- Read 10 integers from the user and print them backwards:

```c
//Declare array
int a[10];
int i;
//Read values
for (i=0; i<10; i++)
{
    scanf("%d", &a[i]);
}
//Print from last to first
for (i=9; i>=0; i--)
{
    printf("%d", a[i]);
}
```
Example: Sum of an array’s elements

```c
int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
```
Example: Sum of an array’s elements

```c
#include <stdio.h>

int main()
{
    int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};

    float sum = 0.0;

    for (int i = 0; i < 10; ++i)
        sum = sum + numbers[i];

    printf("The sum of numbers is %f\n", sum);
    return 0;
}
```
Example: Sum of array’s element

```c
int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
```

```
[C++] $gcc sum.cc -o sum.exe
[C++] $./sum.exe
The sum of numbers is 59.000000
```

Example(III): Searching the index and value of smallest number

```
int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
```

Output:

smallest number: -2
index: 3
Example: Searching the index and value of smallest number

```
int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
```

```c
#include <stdio.h>

int main()
{
    int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
    float min = 0.0;
    int index = -1, i;

    for (int i = 0; i < 10; ++i)
        if(numbers[i] < min)
            {
                min = numbers[i];
                index = i;
            }

    printf("The index and value of smallest number is \%d \t \%f\n", index, min);
    return 0;
}```
Example: Searching the index and value of smallest number

3  4  9  -2  4  8  12  1  15  5
Exercise: Searching the location for the given value in an array

(numbers)

(3 4 9 -2 4 8 12 1 15 5)

(write down your name, PID, and sec#, hand in your paper)
Exercise: Searching the location for the given value in an array

```c
#include <stdio.h>

int main()
{
    int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};

    int key = 0.0;
    int index = -1;

    printf("Please enter the number to be found:\n");
    scanf("%d", &key);

    for (int i = 0; i < 10; ++i)
        if(numbers[i] == key)
            index = i;

    if(index == -1)
        printf("The number does not exist!\n");
    else
        printf("The location of %d in the array is: %d \n", key, index);

    return 0;
}
```
Example(IV): number of elements greater than average

| 3 | 4 | 9 | -2 | 4 | 8 | 12 | 1 | 15 | 5 |

```
[C++] $gcc belowaverage.cc -o belowaverage.exe
[C++] ./belowaverage.exe
The average is: 5.900000
The numbers below average counts to: 6
C++ $`
```
Example: number of elements greater than average

```c
#include <stdio.h>

int main()
{
    int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
    double sum = 0.0, average = 0.0;
    int belowAverage = 0;

    /* Step 1: find the average of numbers */
    for (int i = 0; i < 10; ++i)
        sum = sum + numbers[i];
    average = sum/10;
    printf("The average is: %f \n", average);

    /* Step 2: find the numbers below the average */
    for (int j = 0; j < 10; ++j)
        if(numbers[j] < average)
            belowAverage++;
    printf("The numbers below average counts to: %d \n", belowAverage);
    return 0;
}```
Variable Length Array (VLA)

• In C99: possible to use an expression that is not a constant for the array size

```c
int size;
printf("How many elements?\n");
scanf("%d", &size);
int a[size];
```

• Advantage: use right size instead of guessing, avoid problems with array too short or too long
Variable Length Array (VLA)

• Don't be confused!
  – Once an array is declared, its size can NOT change
  • "Dynamic array" is supported by advanced data-structures, which you may learn in an advanced programming class

```c
int size;
printf("How many elements?\n");
scanf("%d", &size);
int a[size];
```

```c
int a[1000];
```
Variable Length Array (VLA)

– The only difference between a VLA and a constant length array: the length of a VLA is determined in running time, e.g.,

```c
int size;
printf("How many elements?\n");
scanf("%d", &size);
int a[size];
```

vs.

```c
int a[1000];
```
Copy an Array

• Direct assignment is not applicable
  ```c
  int a[5] = {10, 20, 30, 40, 50};
  int b[5];
  b = a;  //Generates a compilation error!
  ```

• Use a loop, copy elements one by one
  ```c
  for (i=0;i<5;i++)
    b[i] = a[i];
  ```

• Use `memcpy` (memory copy) function in `<string.h>`:
  ```c
  memcpy(dest, src, size);
  ```

  ```c
  include <string.h>
  ....
  memcpy(b, a, sizeof(a));  //faster than a loop
  ```
Size of arrays

• `sizeof` operator:

```c
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int size_a = sizeof(a); //10*4 (4 bytes for an int)
int size_int = sizeof(a[0]); //4
int num_elements = sizeof(a) / sizeof(a[0]); //10
```

• Byte
  – 1 byte = 8 bits, can store integer value 0 ~ 255, or one English character
  – E.g., "my iPhone has 4GB memory and 32GB hard drive", "this image is 1.5 MB"

• 1GB = $2^{30}$ bytes, 1MB = $2^{20}$ bytes, 1KB = $2^{10}$ bytes
Example: copy array1 to array2 in reverse order

```c
#include <stdio.h>

int main()
{
    int numbers[10] = {3, 4, 9, -2, 4, 8, 12, 1, 15, 5};
    int compyOfNumbers[10];

    for (int i = 0; i < 10; ++i)
        compyOfNumbers[i] = numbers[i];

    printf("The copied array is:\n");

    for (int j = 0; j < 10; ++j)
        printf("%d", compyOfNumbers[j]);
    printf("\n");
    return 0;
}
```

What's wrong here?
Example: copy array1 to array2 in reverse order

```
3  4  9  -2  4  8  12  1  15  5
```

```
[C++] $gcc copyarray.cc -o copyarray.exe
[C++] $./copyarray.exe
The copied array is:
3  4  9  -2  4  8  12  1  15  5
C++ $
```
Multidimensional Arrays
Would you define a separate array for each student?

```c
float student1[10];
float student2[10];
float student3[10];
...
2D- Arrays

A two-dimensional array consists of both **rows** and **columns** of elements. It is essentially a **matrix**.

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 1</th>
<th>Row 2</th>
<th>Row 3</th>
<th>Row 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="matrix.png" alt="Matrix" /></td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>-4</td>
<td>8</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Declaration of 2D- Arrays in C

To declare a two-dimensional array, we merely use two sets of square brackets.
- The first contains the **number of rows**
- The second contains the **number of columns**

```
array_Type arrayName [ numberOfRows ] [numberOfColumns ];
```

- **The type of each entry in the 2D-array**
- **The name of 2D-array**
- **The number of rows in the 2D-array**
- **The number of columns in the 2D-array**
Example of 2D- Arrays in C

```
array_Type arrayName [ numberOfRows ] [numberOfColumns ];

int SudokuTable [9][9];
```

- The name of this array is “SudokuTable”.
- The type of individual elements is \texttt{int}
- This declaration sets aside a chunk of memory that is big enough to hold 81 integers.
- It does NOT initialize those memory locations to 0 or any other value.
How are 2D-arrays formatted in memory?

- In memory:

  A static two-dimensional array looks like an array of arrays - it's just laid out contiguously in memory.