Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Chapter 7: Arrays

Introduction to Computer Programming
(C language)

Nguyễn Tiến Thịnh, Ph.D.

Email: ntthinh@hcmut.edu.vn

2022 – 2023, Semester 1

Course Content

- C.1. Introduction to Computers and Programming
- C.2. C Program Structure and its Components
- C.3. Variables and Basic Data Types
- C.4. Selection Statements
- C.5. Repetition Statements
- C.6. Functions
- **C.7. Arrays**
- C.8. Pointers
- C.9. File Processing

References

- [1] "*C: How to Program*", 7th Ed. – Paul Deitel and Harvey Deitel, Prentice Hall, 2012.
- [2] "*The C Programming Language*", 2nd Ed. – Brian W. Kernighan and Dennis M. Ritchie, Prentice Hall, 1988
- and others, especially those on the Internet

Content

- Introduction
- One-dimension arrays
- Memory model of a C array
- Access to the elements of a C array
- Arrays of characters for strings
- Multidimensional arrays
- Passing arrays to functions
- Summary

Introduction – Data Types in C (Recall from Chapter 3)

- Built-in data types (primitive/fundamental)
  - char (signed char), unsigned char
  - short int, unsigned short, int, unsigned int, long int, unsigned long int, long long int, unsigned long long
  - float, double, long double
  - void
  - enum (enumerated data associated with integers)
- Derived data types
  - arrays [] of objects of a given type
  - pointers * to objects of a given type
  - structures struct containing objects of other types
  - union containing any one of several objects of various types

Introduction

- Given a set of n positive numbers, find the smallest one.

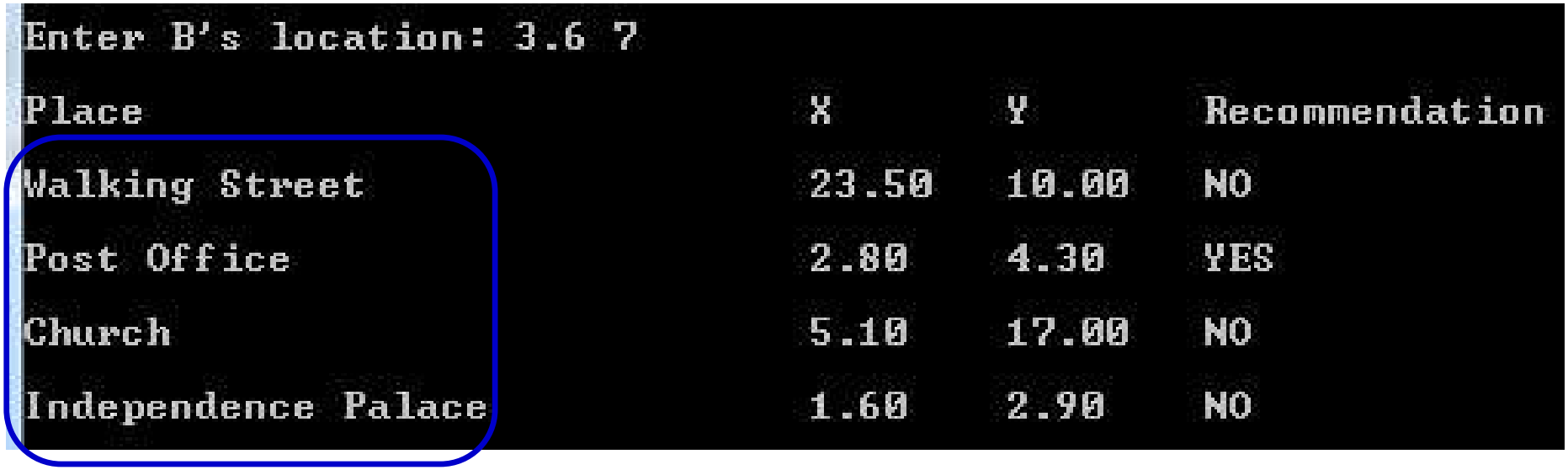
(Chapter 1 – Real code in C)

```
void main() {
    double positiveNumber[10] = {2, 1, 3, 10, 8, 3, 4, 5, 9, 12};
    int n = 10;
    double minNumber = positiveNumber[0];
    int iteration = 1;
    while (iteration < n) {
        if (minNumber <= positiveNumber[iteration])
            iteration = iteration + 1;
        else {
            minNumber = positiveNumber[iteration];
            iteration = iteration + 1;
        }
    }
}
```

Variable declaration for an array of doubles.
We haven't discussed it, have we???

Introduction

- In your exercise for practice, B.6, chapter 3, you are asked to recommend the nearest place from 4 given places to student B based on his/her current location.



How did you implement those?

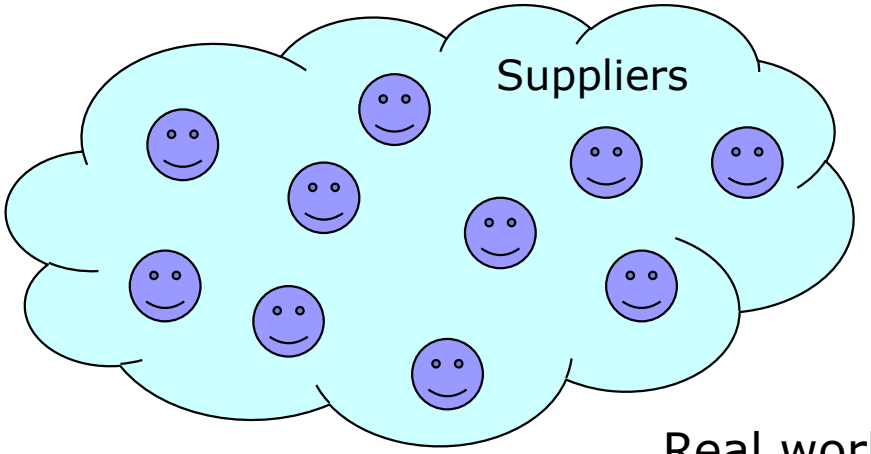
What happens if there are 20 places for recommendation?

Introduction

- In our real world, our problem is related to not only a single object of interest but also a set of objects of interest.
  - Example 1: grade a list of submissions of each of you for every chapter
  - Example 2: recommend the most relevant place to visit from a list of places in a city
  - Example 3: sort a list of suppliers based on their transaction's values
  - Example 4: find a suitable topic among a list of topics for your assignment
  - …

Introduction

- Array
  - A means to represent collections in our real world
  - A data structure consisting of related data items of the same type
  - A group of contiguous memory locations that all have the same type
    - An array remains the same size throughout its existence.



lstSupplier in memory

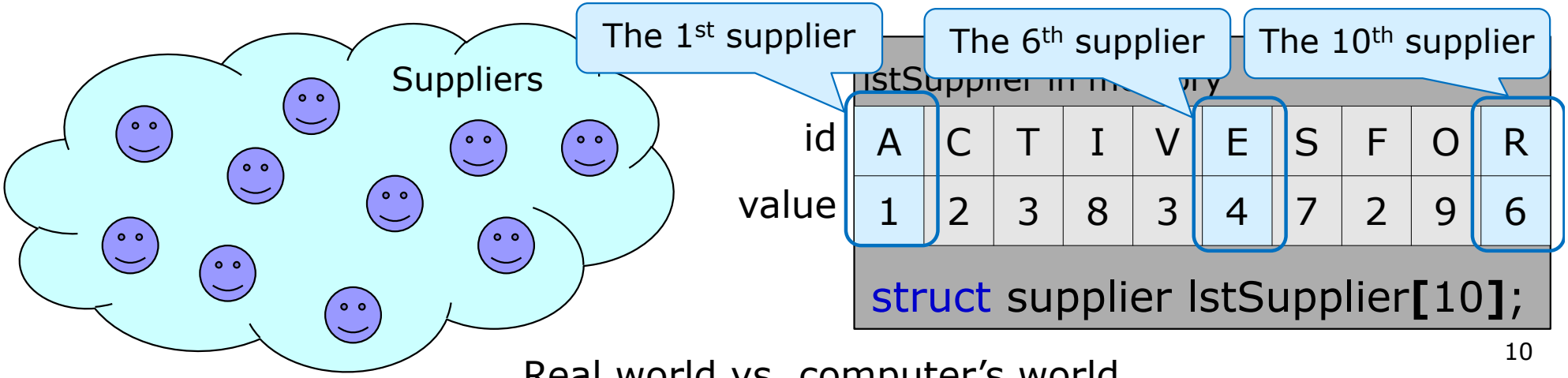
id	A	C	T	I	V	E	S	F	O	R
value	1	2	3	8	3	4	7	2	9	6

```
struct supplier lstSupplier[10];
```

Real world vs. computer's world

Introduction

- Array
  - A means to represent collections in our real world
  - A data structure consisting of related data items of the same type
  - A group of contiguous memory locations that all have the same type
    - An array remains the same size throughout its existence.



Real world vs. computer's world

One-dimension arrays

- A one-dimension array is the simplest type of array in C.

type_name variable_name[*constant_expression*$_{opt}$] =$_{opt}$ {*expression_list*}$_{opt}$;

  - *variable_name*: a valid identifier for a collection
  - *type_name*: a valid data type
    - Basic data type name, derived data type name, or new name of some existing data type
  - *constant_expression*$_{opt}$: optional, if specified, an integer expression used for a size of the array
    - showing the number of objects of interest in the collection
  - *expression_list*: optional, a list of expressions separated by comma for initialized values

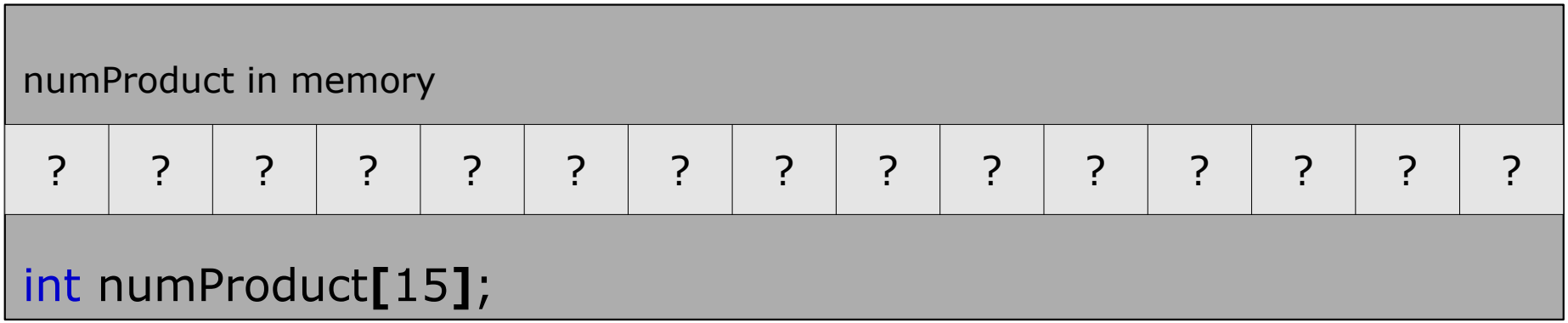
One-dimension arrays

- A one-dimension array is the simplest type of array in C.

type_name variable_name[*constant_expression*$_{opt}$] =$_{opt}$ {*expression_list*}$_{opt}$;

Example 1: an array of 15 integer numbers to represent the number of products bought by 15 customers

```
int numProduct[15];
```



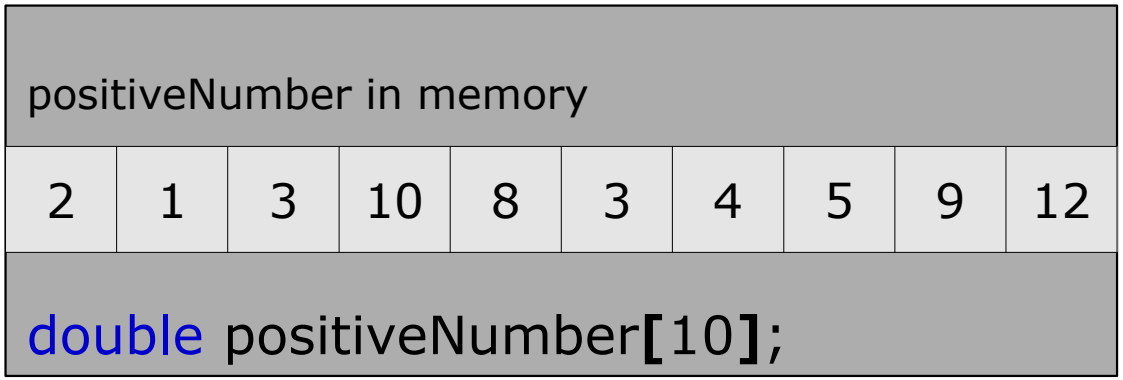
One-dimension arrays

- A one-dimension array is the simplest type of array in C.

type_name variable_name[*constant_expression*$_{opt}$] =$_{opt}$ {*expression_list*}$_{opt}$;

Example 2: an array of 10 doubles to represent 10 positive numbers

```
double positiveNumber[10] = {2, 1, 3, 10, 8, 3, 4, 5, 9, 12};
```



One-dimension arrays

- A one-dimension array is the simplest type of array in C.

type_name variable_name[*constant_expression*$_{opt}$] =$_{opt}$ {*expression_list*}$_{opt}$;

Example 3: an array of 10 elements of struct supplier to represent 10 suppliers

```
struct supplier {
    char id;
    float value;
};

struct supplier lstSupplier[10] = {{'A', 1}, {'C', 2},  \
                {'T', 3}, {'I', 8}, {'V', 3}, {'E', 4}, \
                {'S', 7}, {'F', 2}, {'O', 9}, {'R', 6}};
```

lstSupplier in memory

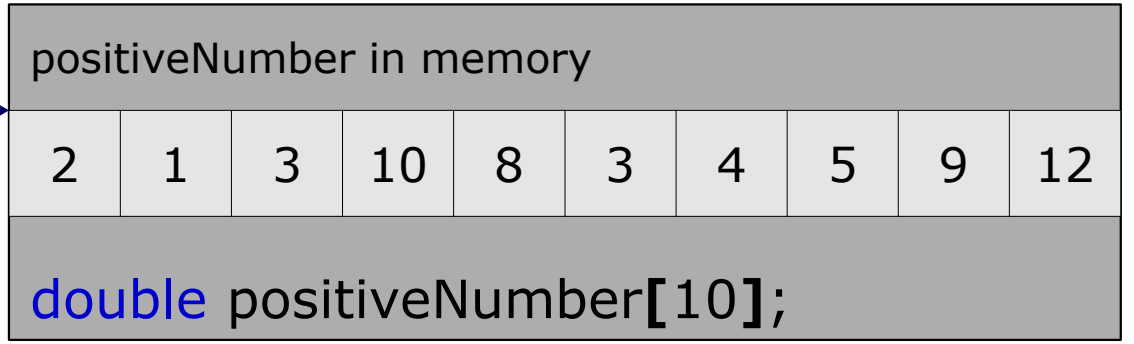
id	A	C	T	I	V	E	S	F	O	R
value	1	2	3	8	3	4	7	2	9	6

struct supplier lstSupplier[10];

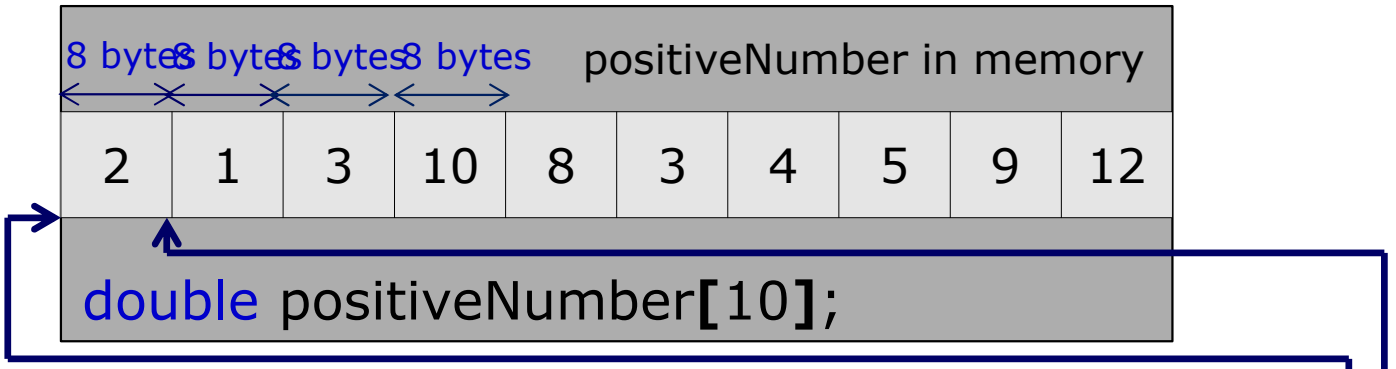
Memory model of a C array

- When defined, an array can be:
  - A global variable,
  - A local variable,
  - A static local variable,
  - A dynamic local variable, …
  - → It will be organized with a fixed size in an appropriate corresponding memory segment (.data, .bss, stack, heap).
  - → Regardless of its memory segment, an array is a group of contiguous memory locations that all have the same type.
    - → Array name is the address of the first location among these contiguous memory locations.

positiveNumber is the address of the first double memory location.



Memory model of a C array




```
positiveNumber[0] = 2.0 at address 000000000022FDF0
positiveNumber[1] = 1.0 at address 000000000022FDF8
positiveNumber[2] = 3.0 at address 000000000022FE00
positiveNumber[3] = 10.0 at address 000000000022FE08
positiveNumber[4] = 8.0 at address 000000000022FE10
positiveNumber[5] = 3.0 at address 000000000022FE18
positiveNumber[6] = 4.0 at address 000000000022FE20
positiveNumber[7] = 5.0 at address 000000000022FE28
positiveNumber[8] = 9.0 at address 000000000022FE30
positiveNumber[9] = 12.0 at address 000000000022FE38
```

Memory model of a C array

- Array name is the address of the first location among these contiguous memory locations.

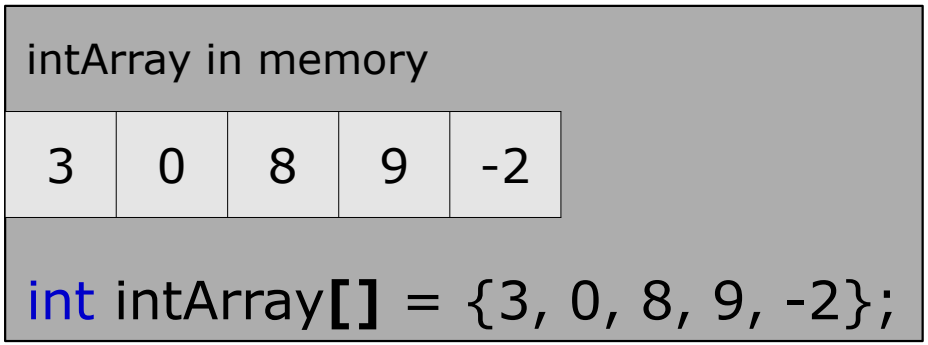
```c
#include <stdio.h>

void main() {

    int a[5] = {1, 5, -2, 3, 4};

    printf("\na = %p vs. address of a[0] = %p\n", a, &a[0]);

}
```

```
a = 000000000022FE30 vs. address of a[0] = 000000000022FE30
```

Therefore, assignment for an entire array is not valid!!!

a = {4, 2, 9, -8, 0}; // INVALID!!!!!!!!!!!!!!

Memory model of a C array



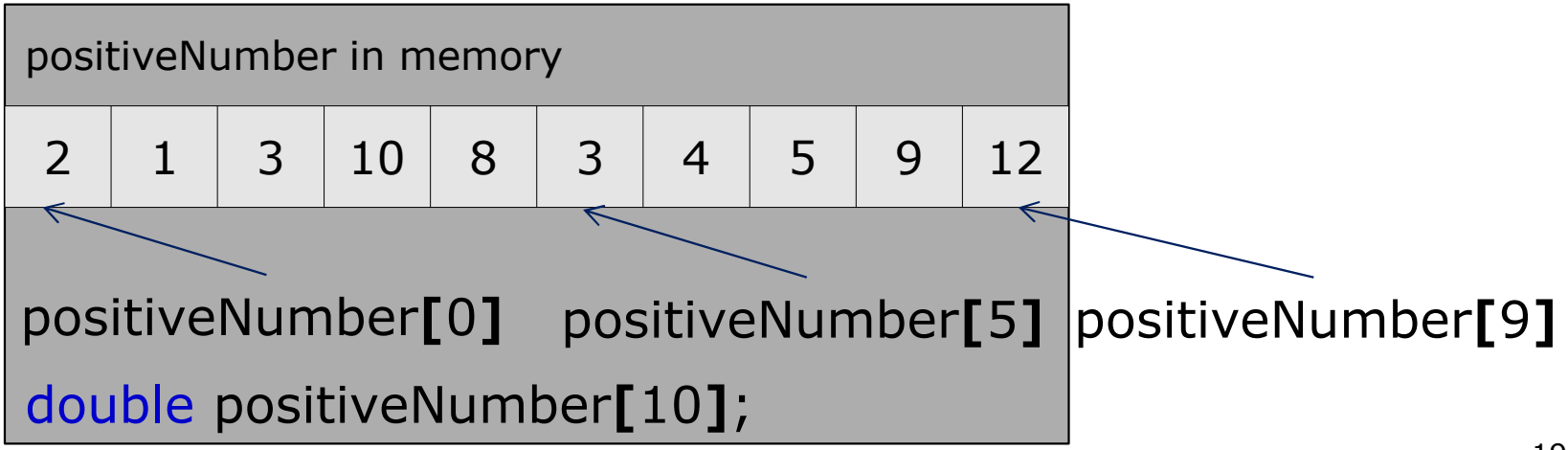

```
int intArray[] = {3, 0, 8, 9, -2};

for (i=0; i<5; i++)
    printf("\nintArray[%d] = %d at address %p\n", i, intArray[i], &intArray[i]);
```

The compiler determined the size of this array based on the number of initial values

```
intArray[0] = 3 at address 000000000022FDD0
intArray[1] = 0 at address 000000000022FDD4
intArray[2] = 8 at address 000000000022FDD8
intArray[3] = 9 at address 000000000022FDDC
intArray[4] = -2 at address 000000000022FDE0
```

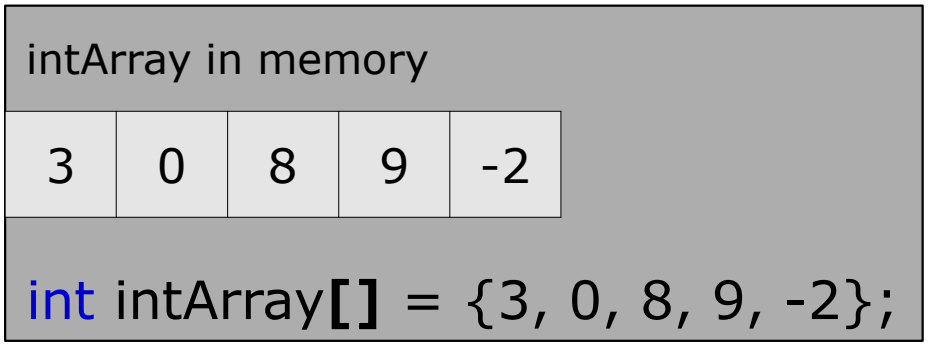
Access to the elements of a C array

- Element access is based on index starting from 0 (zero) with *name*[..].
  - Index for the first element is 0: *name*[0]
  - Index for the second element is 1: *name*[1]
  - Index for the (*i*+1)-th element is *i*: *name*[*i*]
  - Index for the *n*-th element is *n*-1: *name*[*n*-1]



Access to the elements of a C array

- Element access is based on index with *name*[..].




```
int intArray[] = {3, 0, 8, 9, -2};

for (i=0; i<5; i++)
    printf("\nintArray[%d] = %d at address %p\n", i, intArray[i], &intArray[i]);
```

```
intArray[0] = 3 at address 000000000022FDD0
intArray[1] = 0 at address 000000000022FDD4
intArray[2] = 8 at address 000000000022FDD8
intArray[3] = 9 at address 000000000022FDDC
intArray[4] = -2 at address 000000000022FDE0
```

Access to the elements of a C array

- Element access is based on index with *name*[..].
  - What happens with intArray[-1]?
  - What happens with intArray[5]?

```
int intArray[] = {3, 0, 8, 9, -2};

for (i=0; i<5; i++)
    printf("\nintArray[%d] = %d at address %p\n", i, intArray[i], &intArray[i]);

printf("\n\nintArray[-1] = %d at address %p\n", intArray[-1], &intArray[-1]);

printf("\nintArray[5] = %d at address %p\n\n", intArray[5], &intArray[5]);
```

```
intArray[0] = 3 at address 000000000022FDD0
intArray[1] = 0 at address 000000000022FDD4
intArray[2] = 8 at address 000000000022FDD8
intArray[3] = 9 at address 000000000022FDDC
intArray[4] = -2 at address 000000000022FDE0

intArray[-1] = 1076363264 at address 000000000022FDCC
intArray[5] = 0 at address 000000000022FDE4
```

Unspecified values!!!

The previous memory of intArray[0]

The next memory of intArray[4]

Access to the elements of a C array

- What happens if an array defined with a given size is initialized with less initial values?

```
int a1[5] = {2, -4, 6};

for (i=-1; i<6; i++)
    printf("\na1[%d] = %d at address %p\n", i, a1[i], &a1[i]);

char a2[5] = {'a', 'e', 'c', 'b'};

for (i=-1; i<6; i++)
    printf("\na2[%d] = \'%c\' with ASCII = %d at address %p\n", i, a2[i], a2[i], &a2[i]);
```

```
a1[-1] = 2046 at address 000000000022FDAC
a1[0] = 2 at address 000000000022FDB0
a1[1] = -4 at address 000000000022FDB4
a1[2] = 6 at address 000000000022FDB8
a1[3] = 0 at address 000000000022FDBC
a1[4] = 0 at address 000000000022FDC0
a1[5] = 0 at address 000000000022FDC4
```

```
a2[-1] = ' ' with ASCII = 0 at address 000000000022FD9F
a2[0] = 'a' with ASCII = 97 at address 000000000022FDA0
a2[1] = 'e' with ASCII = 101 at address 000000000022FDA1
a2[2] = 'c' with ASCII = 99 at address 000000000022FDA2
a2[3] = 'b' with ASCII = 98 at address 000000000022FDA3
a2[4] = ' ' with ASCII = 0 at address 000000000022FDA4
a2[5] = ' ' with ASCII = 0 at address 000000000022FDA5
```

Zero-filled bytes: 0, '\0'

Access to the elements of a C array

- What happens if an array defined with a given size is initialized with more initial values?

```
int b[5] = {3, 5, 9, -1, 8, 7};

for (i=-1; i<6; i++)
    printf("\nb[%d] = %d at address %p\n", i, b[i], &b[i]);
```

```
b[-1] = 1076363264 at address 000000000022FD7C
b[0] = 3 at address 000000000022FD80
b[1] = 5 at address 000000000022FD84
b[2] = 9 at address 000000000022FD88
b[3] = -1 at address 000000000022FD8C
b[4] = 8 at address 000000000022FD90
b[5] = 0 at address 000000000022FD94
```

Unspecified value!!!

Not 7 in initialization!!!

Access to the elements of a C array

- Element access is based on index starting from 0 (zero) with *name*[..].
- Each element access with *name*[..] plays a role of a single variable of the same type.
  - Used in any relevant expressions
    - a1[0] == 2
    - a1[i] > 1
  - Used in any relevant statements
    - while (a1[2] <= 10) {…}
    - a1[i] = a1[i-1] * 5;
    - …

Access to the elements of a C array

- How many tens have you got for your midterm exams?
- What is your averaged grade?

```
You have 2 ten(s) for the midterm exams.

Your averaged grade is 8.70.
```

```c
#include <stdio.h>

void main() {

    float yrGrades[5] = {7, 10, 8, 10, 8.5};

    float avgGrade = 0;

    int cntTen = 0, i;

    for (i=0; i<5; i++) {
        if (yrGrades[i] == 10) cntTen++;
        avgGrade += yrGrades[i];
    }

    avgGrade /= 5;

    printf("\nYou have %d ten(s) for the midterm exams.\n", cntTen);
    printf("\nYour averaged grade is %.2f.\n", avgGrade);
}
```

- List the frequency of each character in your collection.

```c
#include <stdio.h>

void main() {

    char lstChars[10] = {'a', 'A', 'f', 'A', 'y', 'y', 'o', 'A', 'z', 'r'};

    char disChars[10];
    int cnt[10], i;

    printf("\nA given collection
    for (i=0; i<10; i++) {
        printf("\'%c\'\t", lstCh
        disChars[i] = '\0';
        cnt[i] = 0;
    }

    for (i=0; i<10; i++) {
        int j;
        for (j=0; j<10 && disChars[j] != '\0'; j++)
            if (lstChars[i] == disChars[j]) {
                cnt[j]++;
                break;
            }
        if (j<10 && disChars[j] == '\0') {
            disChars[j] = lstChars[i];
            cnt[j]++;
        }
    }

    printf("\nFrequency is listed as follows:\n");
    for (i=0; i<10; i++)
        if (disChars[i] != '\0')
            printf("\n\'%c\': %d\n", disChars[i], cnt[i]);
        else break;
}
```

```
A given collection  of characters:
'a'     'A'     'f'     'A'     'y'     'y'     'o'     'A'     'z'     'r'
Frequency is listed as follows:
'a': 1
'A': 3
'f': 1
'y': 2
'o': 1
'z': 1
'r': 1
```

Arrays of characters for strings

- The C language has no specific data type for strings.
  - Each string is considered as a one-dimension array of characters ended by '\0'.
  - A standard library for strings: <string.h>
    - size_t strlen(const char *str)
    - char *strcpy(char *dest, const char *src)
    - int strcmp(const char *str1, const char *str2)
    - char *strcat(char *dest, const char *src)
    - char *strtok(char *str, const char *delim)
    - …

Strings and Characters

- <ctype.h>
  - int toupper(int c)
  - int tolower(int c)
  - int isupper(int c)
  - int islower(int c)
  - int ispunct(int c)
  - int isspace(int c)
  - int isalnum(int c)
  - int isalpha(int c)
  - int isdigit(int c)
  - …

Strings and Numbers

- <stdlib.h>
  - double atof(const char *str)
  - int atoi(const char *str)
  - long int atol(const char *str)
  - double strtod(const char *str, char **endptr)
  - long int strtol(const char *str, char **endptr, int base)
  - unsigned long int strtoul(const char *str, char **endptr, int base)

…

Arrays of characters for strings

- strCName is a string
  - defined as an array of 20 characters
  - ended with an additional character '\0'
  - located in memory with 21 bytes

```
Course Name = "Computer Programming"

size of strCName = 21

strCName[0]='C' with ASCII = 67
strCName[1]='o' with ASCII = 111
strCName[2]='m' with ASCII = 109
strCName[3]='p' with ASCII = 112
strCName[4]='u' with ASCII = 117
strCName[5]='t' with ASCII = 116
strCName[6]='e' with ASCII = 101
strCName[7]='r' with ASCII = 114
strCName[8]=' ' with ASCII = 32
strCName[9]='P' with ASCII = 80
strCName[10]='r' with ASCII = 114
strCName[11]='o' with ASCII = 111
strCName[12]='g' with ASCII = 103
strCName[13]='r' with ASCII = 114
strCName[14]='a' with ASCII = 97
strCName[15]='m' with ASCII = 109
strCName[16]='m' with ASCII = 109
strCName[17]='i' with ASCII = 105
strCName[18]='n' with ASCII = 110
strCName[19]='g' with ASCII = 103
strCName[20]='' with ASCII = 0
```

```c
#include <stdio.h>

void main() {

    char strCName[] = "Computer Programming";

    printf("\nCourse Name = \"%s\"\n", strCName);
    printf("\nsize of strCName = %d\n\n", sizeof(strCName));

    int i;
    for (i=0; i<sizeof(strCName); i++)
        printf("strCName[%d]=\'%c\' with ASCII = %d\n", i, strCName[i], strCName[i]);
}
```

	C	o	m	p	u	t	e	r		P	r	o	g	r	a	m	m	i	n	g	\0
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Number of bytes in memory

size >= length + 1

Number of characters in use, not counting '\0'

```
Course Name = "Computer Programming"

size of strCName = 21

strCName[0]='C' with ASCII = 67
strCName[1]='o' with ASCII = 111
strCName[2]='m' with ASCII = 109
strCName[3]='p' with ASCII = 112
strCName[4]='u' with ASCII = 117
strCName[5]='t' with ASCII = 116
strCName[6]='e' with ASCII = 101
strCName[7]='r' with ASCII = 114
strCName[8]=' ' with ASCII = 32
strCName[9]='P' with ASCII = 80
strCName[10]='r' with ASCII = 114
strCName[11]='o' with ASCII = 111
strCName[12]='g' with ASCII = 103
strCName[13]='r' with ASCII = 114
strCName[14]='a' with ASCII = 97
strCName[15]='m' with ASCII = 109
strCName[16]='m' with ASCII = 109
strCName[17]='i' with ASCII = 105
strCName[18]='n' with ASCII = 110
strCName[19]='g' with ASCII = 103
strCName[20]='' with ASCII = 0
```

```
length of strCName = 20

strCName[0]='C' with ASCII = 67
strCName[1]='o' with ASCII = 111
strCName[2]='m' with ASCII = 109
strCName[3]='p' with ASCII = 112
strCName[4]='u' with ASCII = 117
strCName[5]='t' with ASCII = 116
strCName[6]='e' with ASCII = 101
strCName[7]='r' with ASCII = 114
strCName[8]=' ' with ASCII = 32
strCName[9]='P' with ASCII = 80
strCName[10]='r' with ASCII = 114
strCName[11]='o' with ASCII = 111
strCName[12]='g' with ASCII = 103
strCName[13]='r' with ASCII = 114
strCName[14]='a' with ASCII = 97
strCName[15]='m' with ASCII = 109
strCName[16]='m' with ASCII = 109
strCName[17]='i' with ASCII = 105
strCName[18]='n' with ASCII = 110
strCName[19]='g' with ASCII = 103
```

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

void main() {

    char strCName[] = "Computer Programming";

    printf("\nCourse Name = \"%s\"\n", strCName);
    printf("\nsize of strCName = %d\n\n", sizeof(strCName));

    int i;
    for (i=0; i<sizeof(strCName); i++)
        printf("strCName[%d]=\'%c\' with ASCII = %d\n", i, strCName[i], strCName[i]);

    printf("\nlength of strCName = %d\n\n", strlen(strCName));

    for (i=0; i<strlen(strCName); i++)
        printf("strCName[%d]=\'%c\' with ASCII = %d\n", i, strCName[i], strCName[i]);
}
```

The compiler determined the size of this string based on the number of initial characters

C	o	m	p	u	t	e	r		P	r	o	g	r	a	m	m	i	n	g	\0
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Number of bytes in memory

size >= length + 1

Number of characters in use, not counting '\0'

```
Course Name = "C Programming"

size of strCName = 20

strCName[0]='C' with ASCII = 67
strCName[1]=' ' with ASCII = 32
strCName[2]='P' with ASCII = 80
strCName[3]='r' with ASCII = 114
strCName[4]='o' with ASCII = 111
strCName[5]='g' with ASCII = 103
strCName[6]='r' with ASCII = 114
strCName[7]='a' with ASCII = 97
strCName[8]='m' with ASCII = 109
strCName[9]='m' with ASCII = 109
strCName[10]='i' with ASCII = 105
strCName[11]='n' with ASCII = 110
strCName[12]='g' with ASCII = 103
strCName[13]=' ' with ASCII = 0
strCName[14]=' ' with ASCII = 0
strCName[15]=' ' with ASCII = 0
strCName[16]=' ' with ASCII = 0
strCName[17]=' ' with ASCII = 0
strCName[18]=' ' with ASCII = 0
strCName[19]=' ' with ASCII = 0
```

```
length of strCName = 13

strCName[0]='C' with ASCII = 67
strCName[1]=' ' with ASCII = 32
strCName[2]='P' with ASCII = 80
strCName[3]='r' with ASCII = 114
strCName[4]='o' with ASCII = 111
strCName[5]='g' with ASCII = 103
strCName[6]='r' with ASCII = 114
strCName[7]='a' with ASCII = 97
strCName[8]='m' with ASCII = 109
strCName[9]='m' with ASCII = 109
strCName[10]='i' with ASCII = 105
strCName[11]='n' with ASCII = 110
strCName[12]='g' with ASCII = 103
```

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

void main() {

    char strCName[20] = "C Programming";

    printf("\nCourse Name = \"%s\"\n", strCName);
    printf("\nsize of strCName = %d\n\n", sizeof(strCName));

    int i;
    for (i=0; i<sizeof(strCName); i++)
        printf("strCName[%d]=\'%c\' with ASCII = %d\n", i, strCName[i], strCName[i]);

    printf("\nlength of strCName = %d\n\n", strlen(strCName));

    for (i=0; i<strlen(strCName); i++)
        printf("strCName[%d]=\'%c\' with ASCII = %d\n", i, strCName[i], strCName[i]);
}
```

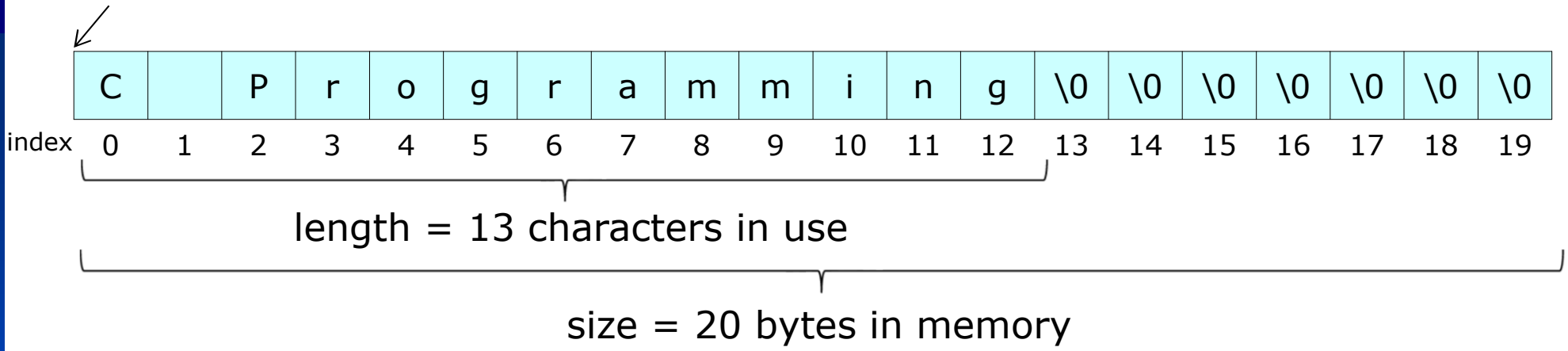
The compiler determined the size of this string based on the given number of elements in array

	C		P	r	o	g	r	a	m	m	i	n	g	\0	\0	\0	\0	\0	\0	\0
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Arrays of characters for strings

char strCName[20] = “C Programming”;

strCName is the address of the first char memory location.



strCName == &strCName[0] => TRUE

strCName[10] == ‘i’ => TRUE

strCName[10] == “i” => FALSE

type value memory

char ‘i’ | i |

char [] “i” | i | \0 |
0 1

strCName = “Programming”; => INVALID for assignment => *strcpy*(..)

strCName == “B Programming” => INVALID for comparison => *strcmp*(..)

Arrays of characters for strings

- Enter your full name and then rewrite your name as “first_name last_name” in English

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

void main() {

    char yrName[20];
    char lastName[10], firstName[10];

    printf("Enter a full name: ");
    gets(yrName);

    printf("\nThe given full name is %s.\n", yrName);

    char * tokens = strtok(yrName, " ");

    strcpy(lastName, tokens);

    printf("\nEach component in the name is listed below:\n");
    while (tokens != NULL) {

        printf("\n%s\n", tokens);

        strcpy(firstName, tokens);

        tokens = strtok(NULL, " ");
    }

    printf("\nA rewritten name in English is %s %s.\n", firstName, lastName);
}
```

```
Enter a full name: Vo Thi Ngoc Chau

The given full name is Vo Thi Ngoc Chau.

Each component in the name is listed below:

Vo

Thi

Ngoc

Chau

A rewritten name in English is Chau Vo.
```

Check if an input name is valid!!

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

char isValidName(char aName[]);

void main() {
    char yrName[20];
    char lastName[10], firstName[10];

    while(1) {
        printf("Enter a full name: ");
        gets(yrName);

        if (isValidName(yrName)) break;
    }
    printf("\nThe given full name is %s.\n", yrName);

    char * tokens = strtok(yrName, " ");

    strcpy(lastName, tokens);

    printf("\nEach component in the name is listed below:\n");
    while (tokens != NULL) {
        printf("\n%s\n", tokens);

        strcpy(firstName, tokens);

        tokens = strtok(NULL, " ");
    }

    printf("\nA rewritten name in English is %s %s.\n", firstName, lastName);
}

//return: 0 for invalid; 1 for valid
char isValidName(char aName[]) {
    int i;
    for (i=0; i<strlen(aName); i++)
        if (aName[i] != ' ' && aName[i] != '\t'
            && (aName[i] < 'a' || aName[i] > 'z')
            && (aName[i] < 'A' || aName[i] > 'Z')) return 0;
    return 1;
}
```

```
Enter a full name: Vo Thi Ngoc-Chau
Enter a full name: Vo Thi_Ngoc Chau
Enter a full name: Vo T&N Chau
Enter a full name: Vo T2N Chau
Enter a full name: Vo .. Chau
Enter a full name: Vo Thi Ngoc Chau

The given full name is Vo Thi Ngoc Chau.

Each component in the name is listed below:

Vo

Thi

Ngoc

Chau

A rewritten name in English is Chau Vo.
```

Full name => First_name Last_name

Checked for a valid full name

Multidimensional arrays

- In our previous example, a string is tokenized into many substrings based on delimiters.
  - Input:

    char aString**[**50**]** = “Introduction to Computer Programming”;
  - Output:
    - **char** Token_1**[**50**]** = “Introduction”;
    - **char** Token_2**[**50**]** = “to”;
    - **char** Token_3**[**50**]** = “Computer”;
    - **char** Token_4**[**50**]** = “Programming”;
  - → Can we have a list of a list of … of a list?
  - → Multidimensional arrays!!!

Multidimensional arrays

- More in our real world, example 1 is:
  - A list of students
    - Each student has a list of submissions, each per chapter.
      - Each submission includes a list of exercises to be grades.
  - → For grading each exercise, we need a three-dimension array of natural numbers.
- Example 2 is:
  - A gray-scale image of size 100x100
    - Each pixel at (i, j) is a positive integer number in [0, 255].
  - → For image processing, we need a two-dimension array of positive integer numbers.
- Example 3 is related to matrices in your linear algebra which are two-dimension arrays of numbers.
- …

Multidimensional arrays

- A multidimensional array is an array. Each element of a multidimensional array is also an array.

type_name variable_name[*size*$^{1}_{opt}$][*size*$^{2}_{opt}$].. [*size*$^{d}_{opt}$] \

$=_{opt}$ {{{*expression_list*}, {…}, …}, {…}, …}$_{opt}$;

size^{1}, *size*^{2}, …, or *size*^{d}: optional, if specified, a constant integer expression used for a size of the array at the 1st, 2nd, …, or dth dimension

{*expression_list*}: optional, a list of expressions separated by comma for initialized values corresponding to dimensions

size^{1}**size*^{2}*…**size*^{d}***sizeof**(*type_name*) bytes are allocated for this array as a group of contiguous memory locations of the same type.

Multidimensional arrays

char aString**[**4**][**50**]**: a list of many different strings from tokenization

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

void main() {
    char aName[50] = "Introduction to Computer Programming";
    char aString[4][50];

    strcpy(aString[0], "Introduction");
    strcpy(aString[1], "to");
    strcpy(aString[2], "Computer");
    strcpy(aString[3], "Programming");

    int i;
    for (i=0; i<4; i++)
        printf("\n%s\n", aString[i]);
}
```

```
Introduction

to

Computer

Programming
```

Multidimensional arrays

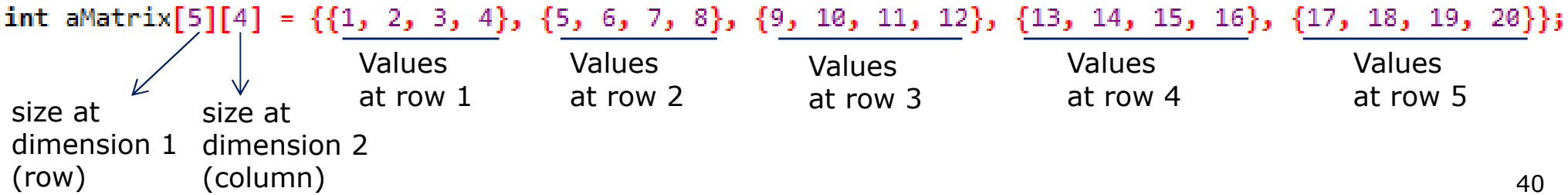
- Example 1:

```
unsigned int yrGrd[50][10][20];
```

- Example 2:

```
unsigned char gsImage[100][100];
```

- Example 3:



```c
#include <stdio.h>

void main() {

    unsigned int yrGrd[50][10][20];

    unsigned char gsImage[100][100];

    int aMatrix[5][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}, {17, 18, 19, 20}};

    printf("\naMatrix = %p\n\n", aMatrix);

    printf("Values\n\n");

    int r, c;
    for (r=0; r<5; r++) {
        for (c=0; c<4; c++)
            printf("%-5d\t", aMatrix[r][c]);
        printf("\n\n");
    }

    printf("\nAddress\n\n");
    for (r=0; r<5; r++) {
        for (c=0; c<4; c++)
            printf("%p  ", &aMatrix[r][c]);
        printf("\n\n");
    }
}
```

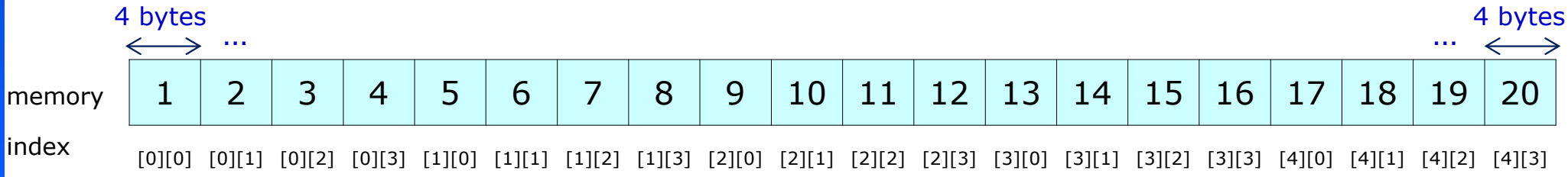
Access to an element:

$name[i^1][i^2]\ldots[i^d]$

```
aMatrix = 0000000000223AA0

Values

1       2       3       4

5       6       7       8

9       10      11      12

13      14      15      16

17      18      19      20

Address

0000000000223AA0  0000000000223AA4  0000000000223AA8  0000000000223AAC

0000000000223AB0  0000000000223AB4  0000000000223AB8  0000000000223ABC

0000000000223AC0  0000000000223AC4  0000000000223AC8  0000000000223ACC

0000000000223AD0  0000000000223AD4  0000000000223AD8  0000000000223ADC

0000000000223AE0  0000000000223AE4  0000000000223AE8  0000000000223AEC
```

A value of aMatrix[1][2]

A value of aMatrix[3][3]



Access via index at each dimension is enabled and indices start at 0 for all dimensions.
Array name is the address of the first memory location: aMatrix == &aMatrix[0][0]

Multidimensional arrays

```c
#include <stdio.h>

void main() {

    int a[][4] = {{1, 0, 3, 2}, {-2, 8, 5, 9}, {2, 7, 1, 0}};

    int b[3][4] = {{1, 4, 2, -5}, {3, 9, 1}};

    int m[][3] = {2, 3};

    int size_a = sizeof(a);
    int numEle_a = size_a/sizeof(int);

    int size_b = sizeof(b);
    int numEle_b = size_b/sizeof(int);

    int size_m = sizeof(m);
    int numEle_m = size_m/sizeof(int);

    printf("\nnumber of elements in a = %d\n", numEle_a);

    printf("\nnumber of elements in b = %d\n", numEle_b);

    printf("\nnumber of elements in m = %d\n", numEle_m);
}
```

Size of the first dimension is determined by the compiler based on the initial values.

Some elements have initial values. The rest will get zeros.

One-dimensional array

```
number of elements in a = 12

number of elements in b = 12

number of elements in m = 3
```

Multidimensional arrays

- Generate a square matrix with a unit diagonal
  - Output square matrix: nxn
- Print a transpose of a square matrix
  - Input square matrix: nxn
  - Output square matrix: nxn
- Compute a dissimilarity matrix of n objects in a 3D space input from the user
  - Input data matrix: nx3
  - Output dissimilarity matrix: nxn

Generate a square matrix with a unit diagonal

```c
#include <stdio.h>

unsigned int getNaturalNumber();

void main() {
    unsigned int n = getNaturalNumber();
    int a[n][n];
    int i, j;

    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) a[i][j] = 0;
        a[i][i] = 1;
    }

    printf("\nA square matrix with a unit diagonal:\n\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) printf("%5d", a[i][j]);
        printf("\n");
    }
}

unsigned int getNaturalNumber() {

    unsigned int N;
    do {
        fflush(stdin);
        printf("Enter a natural number N = ");
        scanf("%u", &N);
    }
    while (N<=0);

    return N;
}
```

```
Enter a natural number N = 5

A square matrix with a unit diagonal:

    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
```

a[i][i] = 1;

a[i][n-1-i] = 1;

```
Enter a natural number N = 5

A square matrix with a unit diagonal:

    0    0    0    0    1
    0    0    0    1    0
    0    0    1    0    0
    0    1    0    0    0
    1    0    0    0    0
```

Print a transpose of a square matrix

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

unsigned int getNaturalNumber();

void main() {
    unsigned int n = getNaturalNumber();
    int a[n][n];
    int i, j;

    time_t t;
    srand((unsigned) time(&t));

    for (i=0; i<n; i++)
        for (j=0; j<n; j++) a[i][j] = (int)rand()%100;

    printf("\nA random matrix:\n\n");
    for (i=0; i<n; i++){
        for (j=0; j<n; j++) printf("%5d", a[i][j]);
        printf("\n");
    }

    printf("\nThe transpose of the random matrix:\n\n");
    for (i=0; i<n; i++){
        for (j=0; j<n; j++) printf("%5d", a[j][i]);
        printf("\n");
    }
}

unsigned int getNaturalNumber() {
    unsigned int N;
    do {
        fflush(stdin);
        printf("Enter a natural number N = ");
        scanf("%u", &N);
    }
    while (N<=0);
    return N;
}
```

```
Enter a natural number N = 5

A random matrix:

   11   36   23   77   15
   28   67   83   89   91
   21   85   88   71   67
   48   40   69   89   82
   72   75   67   79   68

The transpose of the random matrix:

   11   28   21   48   72
   36   67   85   40   75
   23   83   88   69   67
   77   89   71   89   79
   15   91   67   82   68
```

In the standard library: <stdlib.h>

- void srand(): initialize the random number generator used by the function rand() based on the system time

- int rand(): generate a random number in the range of 0 to RAND_MAX (32767)

In the standard library: <time.h>

- time_t: a type for storing the calendar time

- time_t time(time_t *timer): calculate the current calendar time and encode it into time_t format

Compute a dissimilarity matrix of n objects in a 3D space input from the user

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

#define d 3

unsigned int getNaturalNumber();

float Euclidean(float v1[], float v2[]);

void main() {
    unsigned int n = getNaturalNumber();
    float a[n][d];
    float disM[n][n];
    int i, j;

    printf("\nEnter a data matrix: \n\n");
    for (i=0; i<n; i++) {
        for (j=0; j<d; j++) {
            fflush(stdin);
            printf("\na[%d][%d] = ", i+1, j+1);
            scanf("%f", &a[i][j]);
        }
        printf("\n");
    }

    printf("\nDissimilarity matrix:\n\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            disM[i][j] = Euclidean(a[i], a[j]);
            printf("%5.2f ", disM[i][j]);
        }
        printf("\n");
    }
}
```

```c
unsigned int getNaturalNumber() {
    unsigned int N;
    do {
        fflush(stdin);
        printf("Enter a natural number N = ");
        scanf("%u", &N);
    }
    while (N<=0);

    return N;
}

float Euclidean(float v1[], float v2[]) {
    float sum = 0;
    int i;
    for (i=0; i<d; i++)
        sum += (v1[i] - v2[i])*(v1[i] - v2[i]);

    return sqrt(sum);
}
```

Compute a dissimilarity matrix of n objects in a 3D space input from the user

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

#define d 3

unsigned int getNaturalNumber();

float Euclidean(float v1[], float v2[]);

void main() {
    unsigned int n = getNaturalNumber();
    float a[n][d];
    float disM[n][n];
    int i, j;

    printf("\nEnter a data matrix: \n\n");
    for (i=0; i<n; i++) {
        for (j=0; j<d; j++) {
            fflush(stdin);
            printf("\na[%d][%d] = ", i+1, j+1);
            scanf("%f", &a[i][j]);
        }
        printf("\n");
    }

    printf("\nDissimilarity matrix:\n\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            disM[i][j] = Euclidean(a[i], a[j]);
            printf("%5.2f ", disM[i][j]);
        }
        printf("\n");
    }
}
```

```c
unsigned int ge
    unsigned int
    do {
        fflush(
        printf(
        scanf("
    }
    while (N<=0)

    return N;
}

float Euclidean(
    float sum =
    int i;
    for (i=0; i<
        sum +=

    return sqrt(
}
```

```
Enter a natural number N = 4

Enter a data matrix:

a[1][1] = 2

a[1][2] = 3

a[1][3] = 1.5

a[2][1] = 0

a[2][2] = 2

a[2][3] = 1

a[3][1] = 5

a[3][2] = 2.4

a[3][3] = 0.6

a[4][1] = 7

a[4][2] = 2.1

a[4][3] = 4

Dissimilarity matrix:

 0.00  2.29  3.19  5.66
 2.29  0.00  5.03  7.62
 3.19  5.03  0.00  3.96
 5.66  7.62  3.96  0.00
```

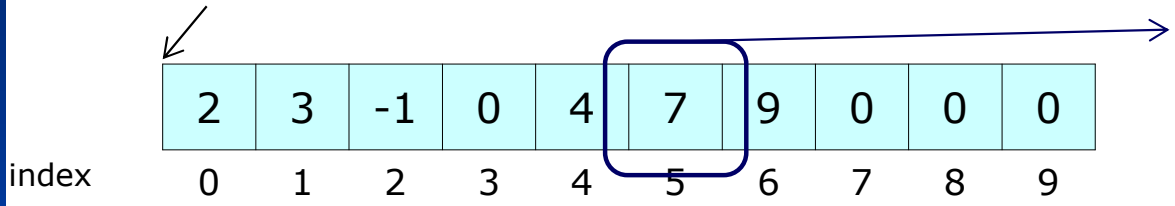
Passing arrays to functions

- Recall

```
The b matrix:

  2   3  -1   0   4
  7   9   0   0   0
  6  11  -2   5   0

b[2] = 000000000022FE18

&b[2][0] = 000000000022FE18
```

int a[10] = {2, 3, -1, 0, 4, 7, 9};

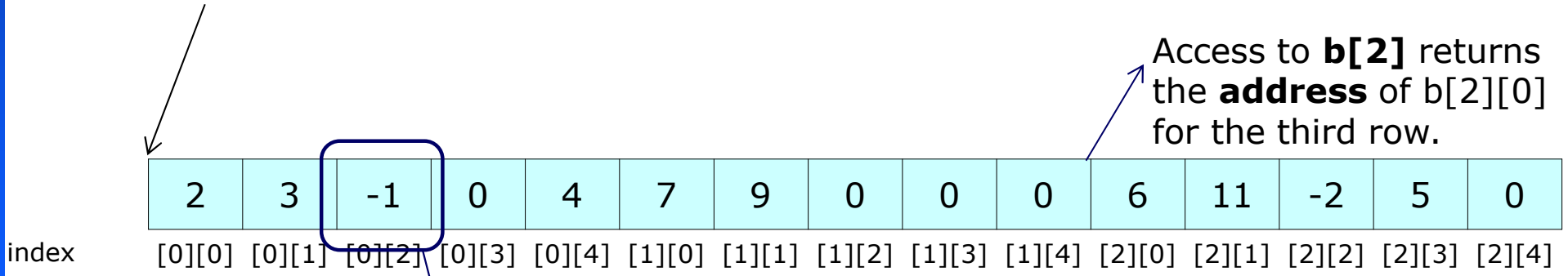
a, array name, is the address of the first **int** memory location.



Access to **a[5]** returns an **int** value: 4.

int b[3][5] = {{2, 3, -1, 0, 4}, {7, 9}, {6, 11, -2, 5}};

b, array name, is the address of the first **int** memory location.



Access to **b[2]** returns the **address** of b[2][0] for the third row.

Access to **b[0][2]** returns an **int** value: -1.

Passing arrays to functions

- Pass a value of an element at index i of a one-dimension array a to functions
  - Call to function *func*: *func*(a[i], …)

  Value passing - unchanged
- Pass all the values of the elements of a one-dimension array a to functions
  - Call to function *func*: *func*(a, …)

  Address passing - changeable
- Pass a value of an element at indices i and j of a two-dimension array b to functions
  - Call to function *func*: *func*(b[i][j], …)

  Value passing - unchanged
- Pass a row at index i of a two-dimension array b to functions
  - Call to function *func*: *func*(b[i], …)

  Address passing - changeable
- Pass all the values of the elements of a two-dimension array b to functions
  - Call to function *func*: *func*(b, …)

  Address passing - changeable

- Find the greatest number in an array of integer numbers

getMax(a[i-1], a[i]) ?

getMaxA(a, 10) ?

```
#include <stdio.h>

#define MAX_INT 2147483647
#define MIN_INT -2147483647

int getMax(int num1, int num2);
int getMaxA(int nums[], int n);
int getMaxM(int nums[][5], int rows, int cols);

void main() {

    int a[10] = {3, -2, 5, 0, 1, 8, 0, 9, 4, 9};
    int b[4][5] = {{11, 2, -4, 1, 9}, {9, 7, 12, 6, 0}, {5, 12, 2, 12, 01}, {2, 0, -3, 8, 4}};

    int aMax, aMaxA, aMaxM, i, j;

    //find in array a[10]
    aMax = getMax(a[0], a[1]);
    for (i=2; i<10; i++) {
        int nextMax = getMax(a[i-1],a[i]);
        if (aMax<nextMax) aMax = nextMax;
    }

    aMaxA = getMaxA(a, 10);

    printf("\nArray a[10] = {3, -2, 5, 0, 1, 8, 0, 9, 4, 9}\n\n");
    printf("\nThe maximum one in array a[10] is: aMax = %d vs. aMaxA = %d.\n\n", aMax, aMaxA);

    aMaxA = getMaxA(b[0], 5);
    for (i=1; i<4; i++) {
        int nextRow = getMaxA(b[i], 5);
        if (aMaxA<nextRow) aMaxA = nextRow;
    }

    aMaxM = getMaxM(b, 4, 5);

    printf("\nArray b[4][5] = \n{{11, 2, -4, 1, 9}, {9, 7, 12, 6, 0}, {5, 12, 2, 12, 01}, {2, 0, -3, 8, 4}}\n\n");
    printf("\nThe maximum one in array b[4][5] is: aMaxA = %d vs. aMaxM = %d.\n\n", aMaxA, aMaxM);
}
```

```
Array a[10] = {3, -2, 5, 0, 1, 8, 0, 9, 4, 9}

The maximum one in array a[10] is: aMax = 9 vs. aMaxA = 9.

Array b[4][5] =
{{11, 2, -4, 1, 9}, {9, 7, 12, 6, 0}, {5, 12, 2, 12, 01}, {2, 0, -3, 8, 4}}

The maximum one in array b[4][5] is: aMaxA = 12 vs. aMaxM = 12.
```

getMaxA(b[0], 5) ?

getMaxA(b[i], 5) ?

getMaxM(b, 4, 5) ?

Find the greatest number in an array of integer numbers

```
//return the maximum one among two numbers
int getMax(int num1, int num2) {

    return num1>num2?num1:num2;
}

//return the maximum one in a one-dimension array
int getMaxA(int nums[], int n) {

    if (n<=0) return MAX_INT;

    int aMax = nums[0];
    int i;

    for (i=1; i<n; i++)
        if (aMax<nums[i]) aMax = nums[i];

    return aMax;
}

//return the maximum one in a two-dimension array
int getMaxM(int nums[][5], int rows, int cols) {

    if (rows<=0 || cols<=0) return MAX_INT;

    int aMax = MIN_INT;
    int i, j;

    for (i=0; i<rows; i++)
        for (j=0; j<cols; j++)
            if (aMax<nums[i][j]) aMax = nums[i][j];

    return aMax;
}
```

It is necessary to specify the actual number of elements present in the array

It is not necessary to specify the size of the array.

It is necessary to specify the actual number of elements present in the array at each dimension

It is required to specify the size of each non-first dimension of the array.

- Change the greatest numbers in an array of integer numbers to RAND_MAX

chgMaxA(a, 10, RAND_MAX) ?

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

#define MAX_INT 2147483647
#define MIN_INT -2147483647

int getMax(int num1, int num2);
int getMaxA(int nums[], int n);
int getMaxM(int nums[][5], int rows, int cols);
int chgMaxA(int nums[], int n, int newMax);
int chgMaxM(int nums[][5], int rows, int cols, int newMax);

void main() {

    int a[10] = {3, -2, 5, 0, 1, 8, 0, 9, 4, 9};
    int b[4][5] = {{11, 2, -4, 1, 9}, {9, 7, 12, 6, 0}, {5, 12, 2, 12, 01}, {2, 0, -3, 8, 4}};
     ...
    int cnt = chgMaxA(a, 10, RAND_MAX);

    printf("\nNew array a[10] with %d char
    for (i=0; i<10; i++) printf("%d \t", a

    aMaxA = getMaxA(b[0], 5);
    for (i=1; i<4; i++) {
        int nextRow = getMaxA(b[i], 5);
        if (aMaxA<nextRow) aMaxA = nextRow
    }

    aMaxM = getMaxM(b, 4, 5);

    printf("\n\nArray b[4][5] = \n\n");
    for (i=0; i<4; i++) {
        for (j=0; j<5; j++) printf("%d \t"
        printf("\n");
    }
    printf("\nThe maximum one in array b[4

    cnt = chgMaxM(b, 4, 5, RAND_MAX);

    printf("\nNew array b[4][5] with %d changes:\n\n", cnt);
    for (i=0; i<4; i++) {
        for (j=0; j<5; j++) printf("%d \t", b[i][j]);
        printf("\n");
    }
}
```

```
Array a[10] = {3, -2, 5, 0, 1, 8, 0, 9, 4, 9}

The maximum one in array a[10] is: aMax = 9 vs. aMaxA = 9.

New array a[10] with 2 changes:

3       -2      5       0       1       8       0       32767   4       32767

Array b[4][5] =

11      2       -4      1       9
9       7       12      6       0
5       12      2       12      1
2       0       -3      8       4

The maximum one in array b[4][5] is: aMaxA = 12 vs. aMaxM = 12.

New array b[4][5] with 3 changes:

11      2       -4      1       9
9       7       32767   6       0
5       32767   2       32767   1
2       0       -3      8       4
```

chgMaxM(b, 4, 5, RAND_MAX) ?

- Change the greatest numbers in an array of integer numbers to RAND_MAX

```
//return the number of changes
int chgMaxA(int nums[], int n, int newMax) {
    int aMax = getMaxA(nums, n);
    int i, cnt = 0;

    for (i=0; i<n; i++)
        if (nums[i]==aMax) {
            nums[i] = newMax;
            cnt++;
        }

    return cnt;
}
```

An element nums[i] is changed. This change is recorded in the memory locations of the array during the execution of the called function. Changes remain after the execution of the called function.

```
//return the number of changes
int chgMaxM(int nums[][5], int rows, int cols, int newMax) {
    int aMaxM = getMaxM(nums, rows, cols);
    int i, j, cnt = 0;

    for (i=0; i<rows; i++)
        for (j=0; j<cols; j++)
            if (nums[i][j] == aMaxM) {
                nums[i][j] = newMax;
                cnt++;
            }

    return cnt;
}
```

An element nums[i][j] is changed. This change is recorded in the memory locations of the array during the execution of the called function. Changes remain after the execution of the called function.

Put them all together

- Problem 1: String Filtering
  - Input: a string from a user obtained with gets()

"1. Today is Monday, isn't it? 234 - go 2 school."\n

  - Filtering: remove all the digits and redundant spaces (' ', '\t') by keeping and/or replacing with a single whitespace ' '
  - Output: a string with no digit and each word separated from each other by a single whitespace ' '

". Today is Monday, isn't it? - go school."

Put them all together

- Problem 2: Statistical Descriptive Info.
  - Input: a one-dimension array of n integer numbers with n is a natural number given by a user

    ```
    int a[10] = {1, -3, 9, 0, 2, 8, 9, 4, 7, 6};
    ```

  - Calculation: calculate the statistical descriptive information about the input n integer numbers: min, median, mean, mode, max
  - Output: min, median, mean, mode, max

    min = -3; median = (4+6)/2 = 5.0;

    mean = 4.3; mode = 9; max = 9

Note: sorting is needed!

Put them all together

- Problem 3: Matrix Multiplication
  - Input: two matrices *m*1 (r1xc1) and *m*2 (r2xc2) of floating-point numbers
  - Calculation: multiply *m*1 by *m*2
  - Output: a resulting matrix *m*3 (r1xc2)

$$m1 = \begin{bmatrix} 2 & 4 & 0 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & 2 & 3 & -1 \end{bmatrix}$$

$$m2 = \begin{bmatrix} 0 & 2 \\ 1 & -1 \\ -1 & 1 \\ 3 & 0 \end{bmatrix}$$

$$m3 = \begin{bmatrix} 7 & 0 \\ -3 & 5 \\ -4 & 1 \end{bmatrix}$$

Summary

- An array is a group of continuous memory locations of the same type.
  - Its size is unchanged during its existence.
  - It can be considered as a group of individual variables of the same type.
    - Used in any relevant expressions, statements, functions
- Definition can be done with initialization.
- Index-based access starts at zero.
- One-dimension vs. multidimensional arrays
- Strings are special one-dimension arrays of characters ended by '\0'.

Chapter 7: Arrays