[SWE2015-42] Introduction to Data Structures (자료구조개론)

Basics of C Programming

Department of Computer Science and Engineering

Instructor: Hankook Lee (이한국)

Codedang Registration

Invitation Code: 789797

- Many students are not registered yet
  - Sign up for Codedang with your official email (@skku.edu)
  - After sign-up, submit this form: https://forms.gle/4rHBkTwD3Tb2uAse6

- **Codedang** is not a perfect platform
  - **Your participation** would be very helpful for improving its quality
  - Please share your any feedback anytime when you feel uncomfortable with this
  - Anonymous Survey: https://forms.gle/vQvjkKymcynrytF36

Variables and Constants

- You can define variables and constants

main.c

```c
#include <stdio.h>
#define N 100                   // Constant

int main() {
    int n = 10;                 // Variable (integer)
    printf("%d\n", n+N);
    n = 5;
    printf("%d\n", n+N);
    return 0;
}
```

Variables and Constants

- Basic data types in C

Data Type	Size in Bytes	Range	Use
char	1	-128 to 127	To store characters
int	4	-2147483648 to 2147483647	To store integer numbers
float	4	3.4E-38 to 3.4E+38	To store floating-point numbers
double	8	1.7E-308 to 1.7E+308	To store big floating-point numbers

- You can check the byte size by `sizeof()` function

main.c

```c
#include <stdio.h>

int main() {
    int a;
    printf("%lu\n" sizeof(a));
    return 0;
}
```

Input and Output Functions

- You can read user inputs via `scanf` and print messages via `printf`

main.c

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n); // Read an integer from user
    printf("Integer: %d\n", n); // Print the integer
    return 0;
}
```

- `&n` : the address of the variable `n`
- `%d` : the type specifier for integer values
- `\n` : the new line character

Input and Output Functions

- Type specifiers

Data Type	Specifier	Use
char	%c	For single characters
string	%s	For a sequence of characters
int	%d	For integer values
float / double	%f / %lf	For floating point numbers
pointer	%p	For pointers & addresses

- Escape characters

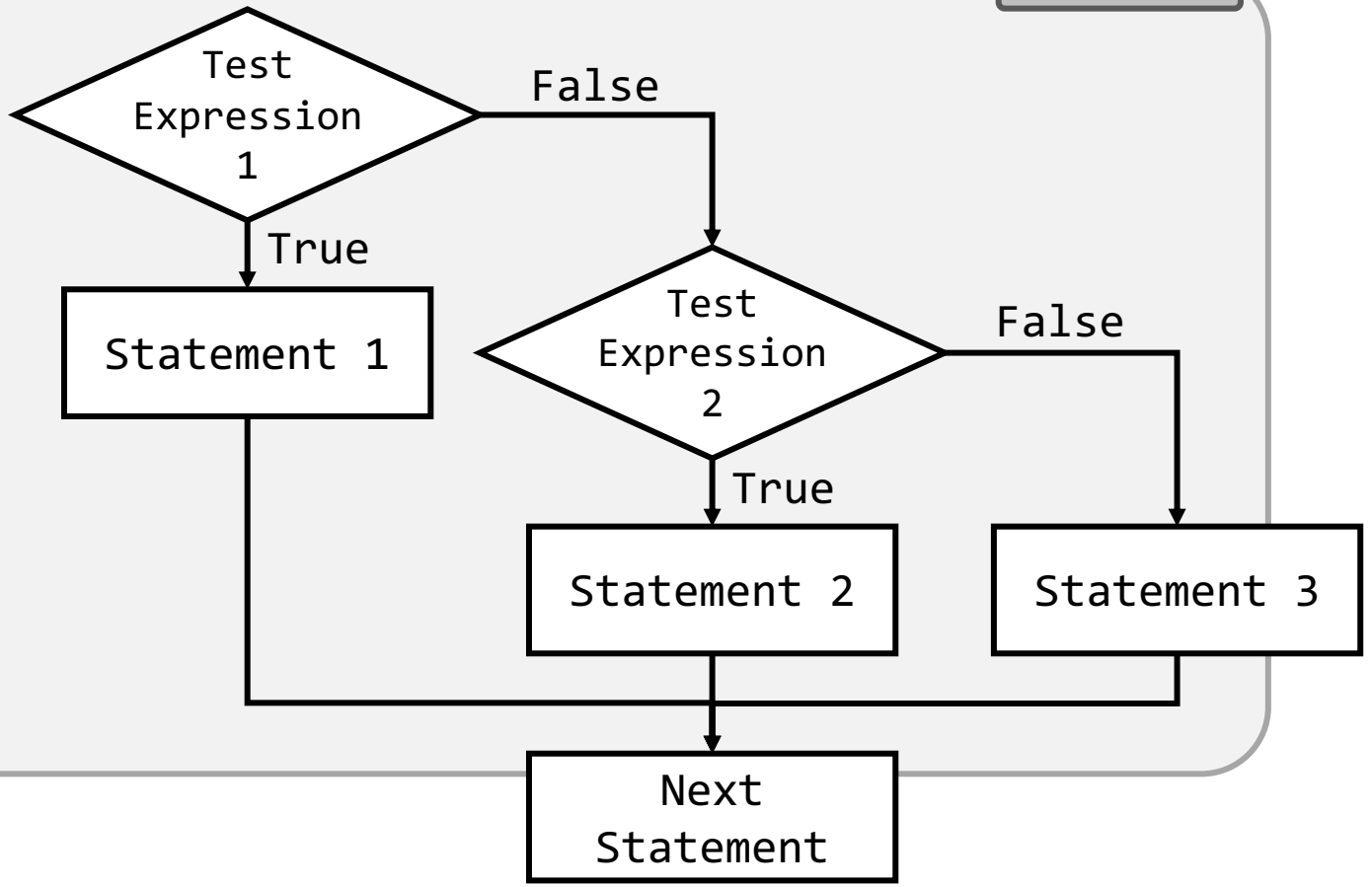
Escape Character	Use
\n	New line
\t	Tab

IF Statements

- You can control the flow of a sequence of instructions based on conditions

main.c

```
int main() {
    if (test expression 1) {
        statement 1;
    }
    else if (test expression 2) {
        statement 2;
    }
    else {
        statement 3;
    }
    next statement;
    return 0;
}
```



Practice

- Read `name` (string) and `score` (integer) of a student
- If the `score` ≥ 80, give A grade
- If the `score` ≥ 60, give B grade
- Otherwise, give C grade

- Print a message with the following format:

```
[name]'s grade is [grade].
```

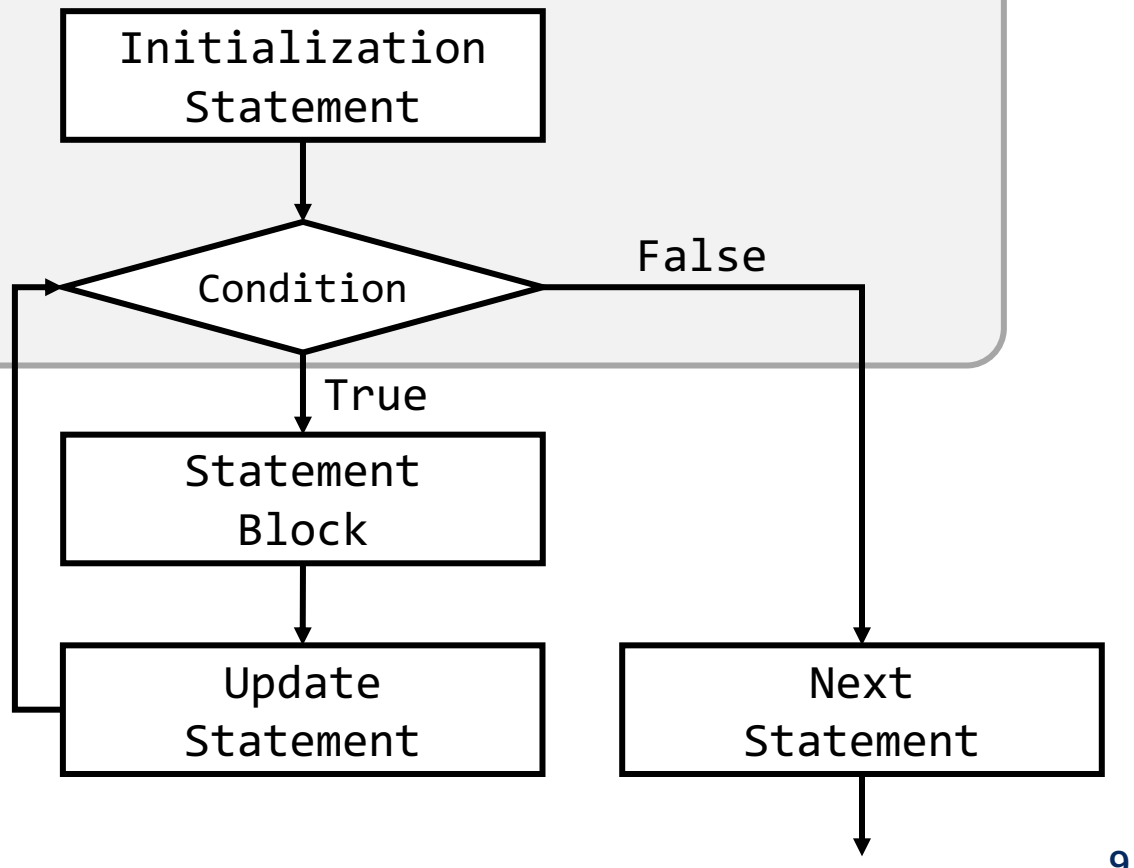
  - Example: `John's grade is A.`

Loop Statements - for

- You can repeat the execution of a sequence of statements

main.c

```
int main() {
    for (initialization; condition; update) {
        statement block;
    }
    next statement;
    return 0;
}
```

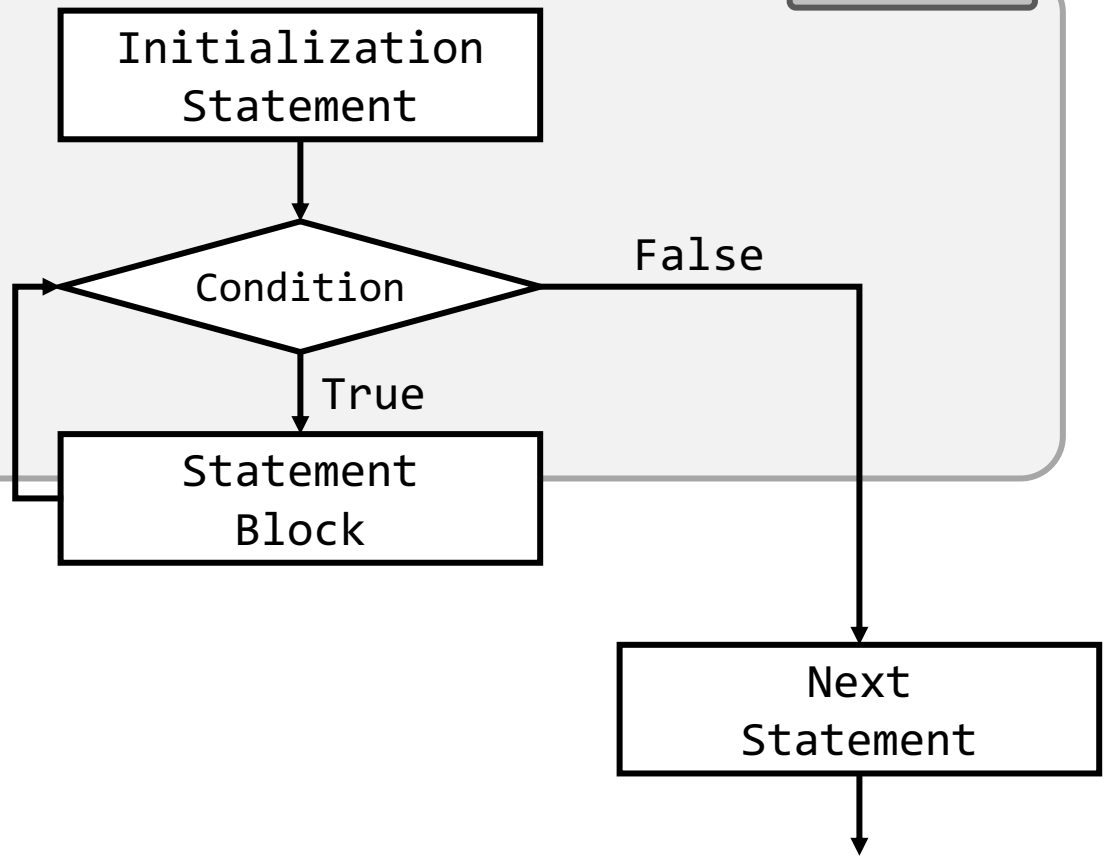


Loop Statements - while

- You can repeat the execution of a sequence of statements

main.c

```
int main() {
    initialization;
    while (condition) {
        statement block;
    }
    next statement;
    return 0;
}
```



Loop Statements

- `break` - terminate the execution of the loop statement
- `continue` - skip the rest of the statements in the block

main.c

```c
#include <stdio.h>

int main() {
    int i;
    for (i = 0; i < 10; i ++) {
        if (i % 2 == 0) continue;
        if (i > 5) break;
        printf("%d\n", i);
    }
    return 0;
}
```

Practice

- Read n (positive integer, ≥2)
- Check n is a prime number

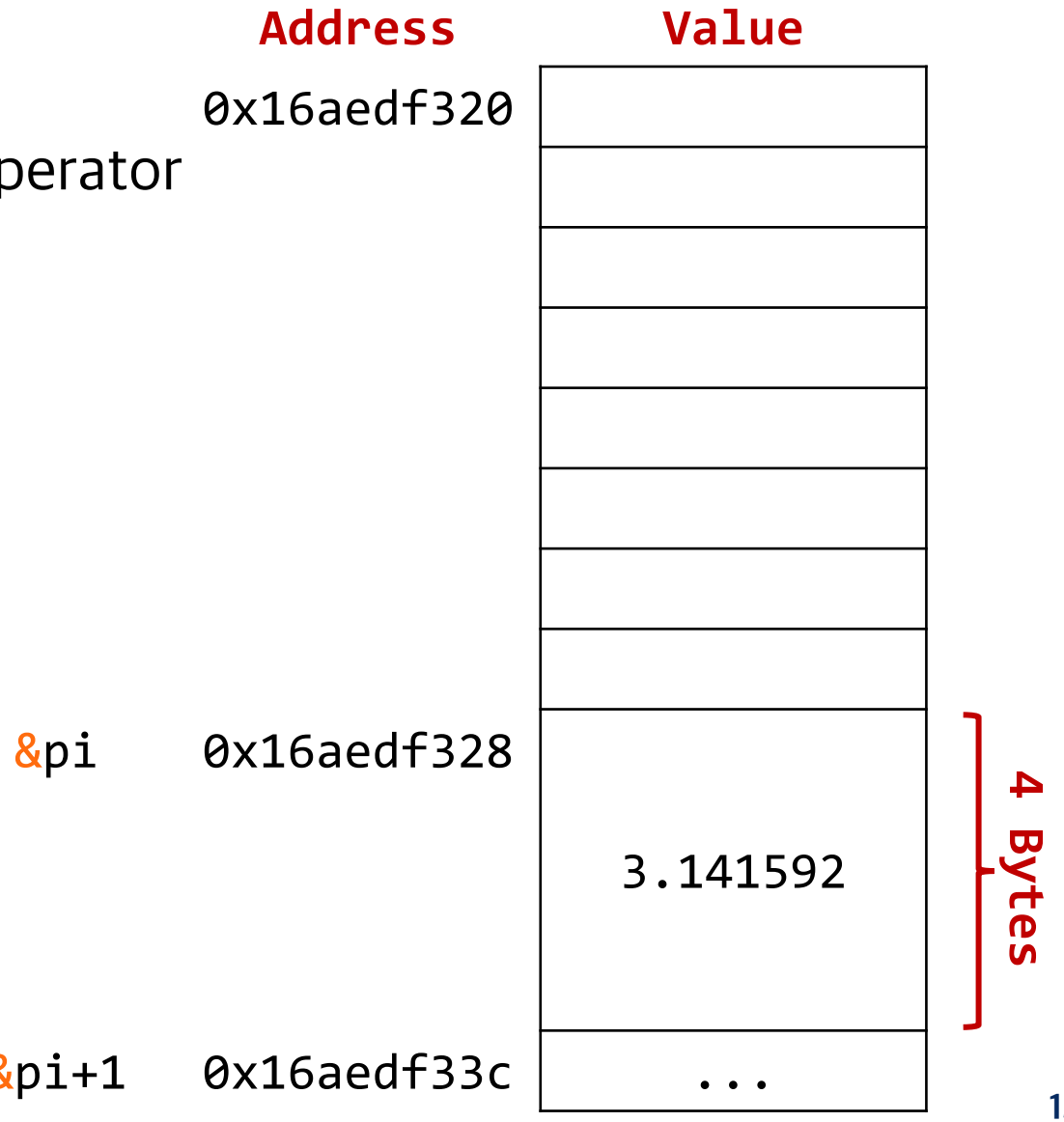
Pointers

- Where is data stored in memory?
  - Access **address** of a variable by **&** (ampersand) operator

main.c

```c
#include <stdio.h>

int main() {
    float pi = 3.141592;
    printf("%p\n", &pi);
    return 0;
}
```



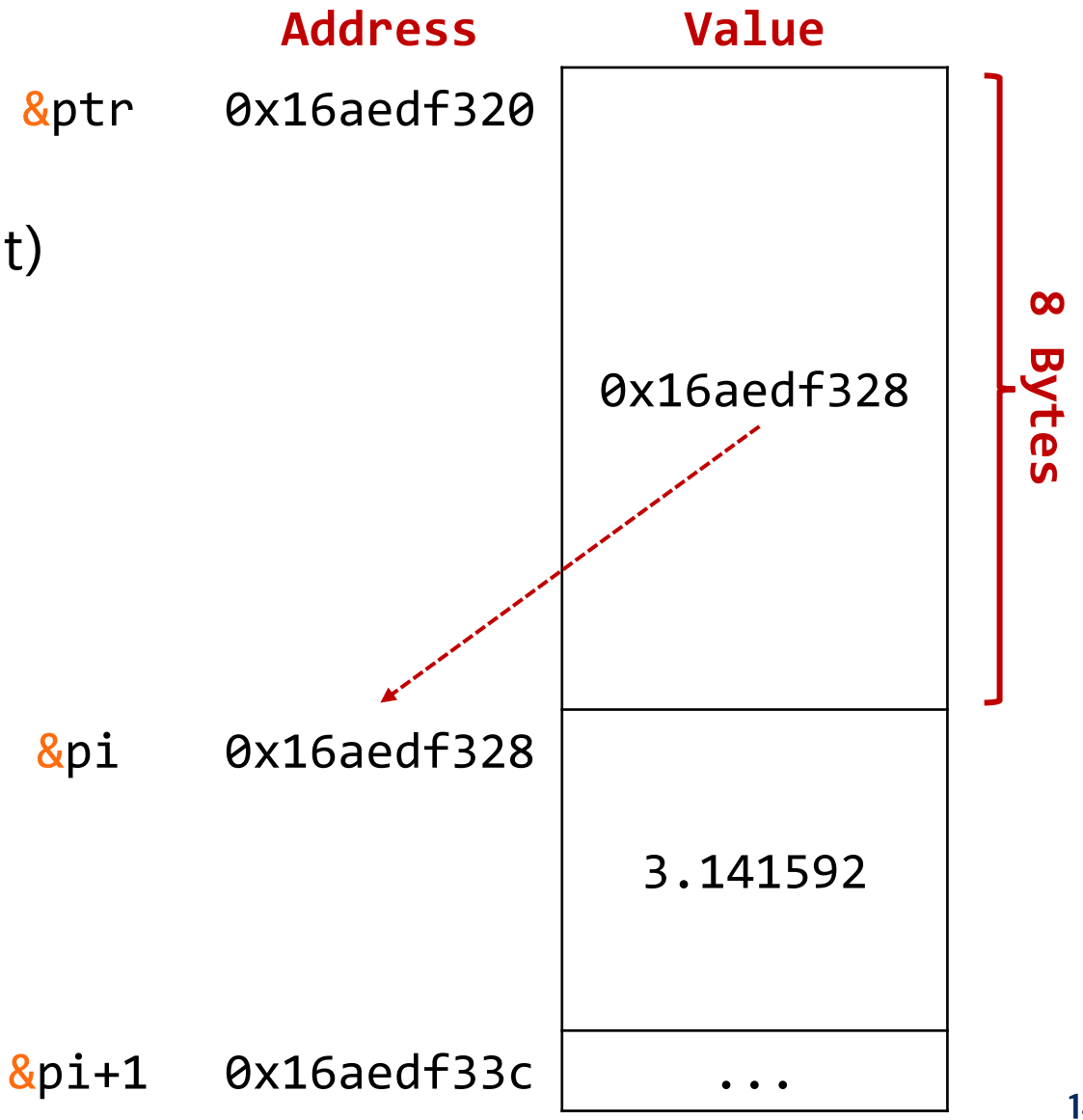
Pointers

- **Pointer** is the data type for address
  - Declare a pointer by * (asterisk) operator
  - Check the size of a pointer → 8 bytes (64-bit)

main.c

```c
#include <stdio.h>

int main() {
    float pi = 3.141592, *ptr = &pi;
    printf("%p\n", ptr);
    return 0;
}
```



Pointers

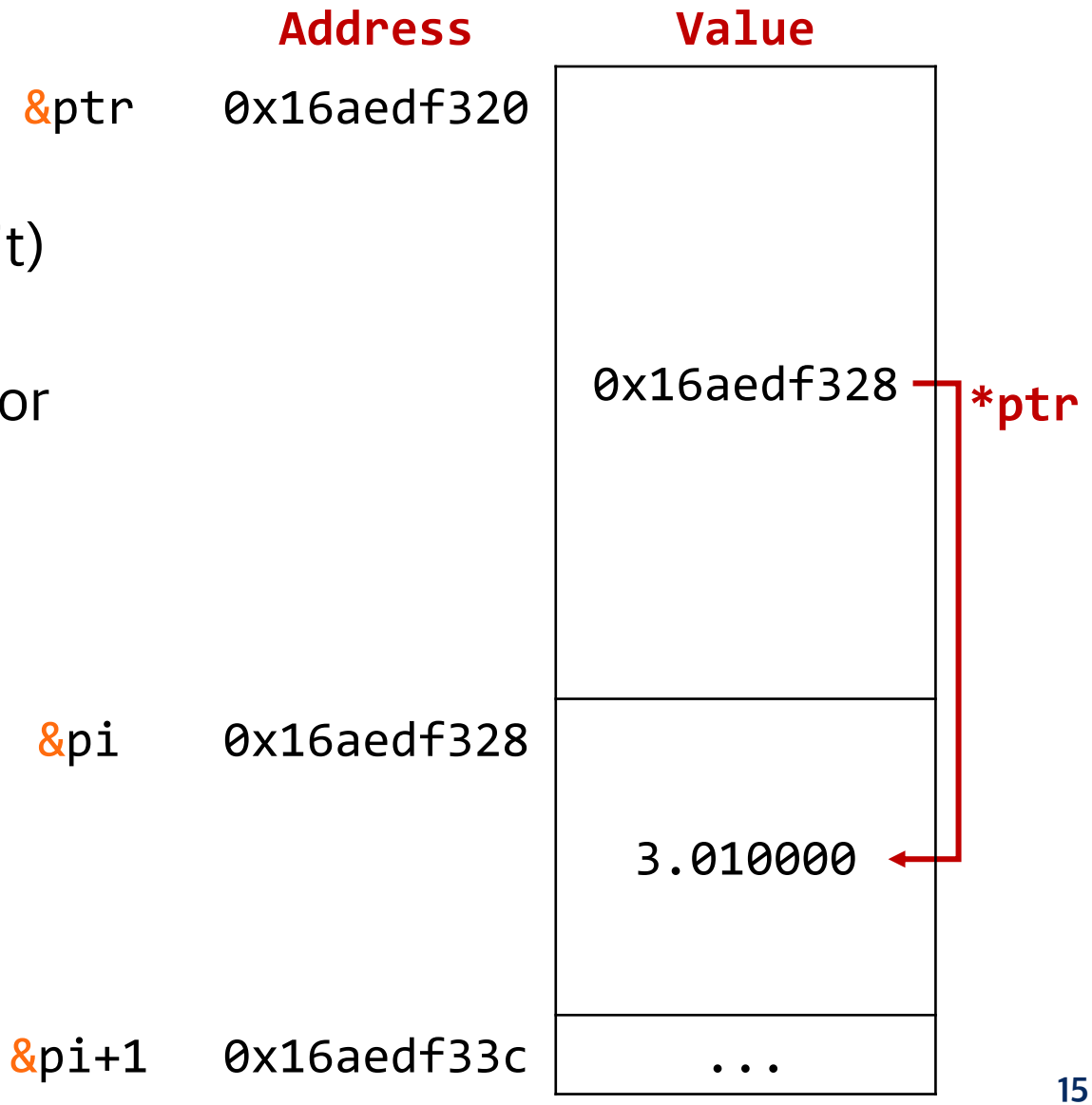
- **Pointer** is the data type for address
  - Declare a pointer by * (asterisk) operator
  - Check the size of a pointer → 8 bytes (64-bit)

  - Access the value at the address by * operator

main.c

```c
#include <stdio.h>

int main() {
    float pi = 3.141592, *ptr = &pi;
    *ptr = 3.01;
    printf("%f\n", pi);
    return 0;
}
```



Arrays

- An array is a collection of elements with the same data type

Address	&arr[0]	&arr[1]	&arr[2]	&arr[3]
Value	arr[0]	arr[1]	arr[2]	arr[3]

main.c

```c
#include <stdio.h>

int main() {
    int i, arr[4] = { 1, 2, 4, 8 }, sum = 0;
    for (i = 0; i < 4; i ++) sum += arr[i];
    printf("sum: %d\n", sum);
    return 0;
}
```

Arrays

- An array is a collection of elements with the same data type

Address	&arr[0]	&arr[1]	&arr[2]	&arr[3]
Value	arr[0]	arr[1]	arr[2]	arr[3]

  - The elements are stored in memory sequentially : `&arr[i+1] == &arr[i]+1`
  - The array variable is the pointer of the first element : `arr == &arr[0]`

main.c

```c
#include <stdio.h>

int main() {
    int i, arr[4] = { 1, 2, 4, 8 };
    for (i = 0; i < 4; i ++) printf("%d-th element address: %p\n", i, &arr[i]);
    printf("array address: %p\n", arr);
    return 0;
}
```

Functions

- You can define a function by `<return_type> <name>(arguments...)`
- You can call the function by `<name>(...)`

main.c

```c
#include <stdio.h>

int max(int a, int b); // Function Declaration
int main() {
    int a = 2, b = 4;
    printf("maximum of %d and %d = %d", a, b, max(a, b));
    return 0;
}

int max(int a, int b) { // Function Header
    // Function Body
    if (a > b) return a;
    else return b;
}
```

Functions - Call by Value

- Call by value - The values of the variables are passed
  - This copy values from the calling function to the called function

main.c

```c
#include <stdio.h>

int sum(int a, int b) {
    return a+b;
}

int main() {
    int a = 2, b = 3;
    printf("%d\n", sum(a, b));
    return 0;
}
```

Functions - Call by Reference

- Call by reference - The addresses of the variables are passed
  - This can access the variables in the calling function using the addresses

main.c

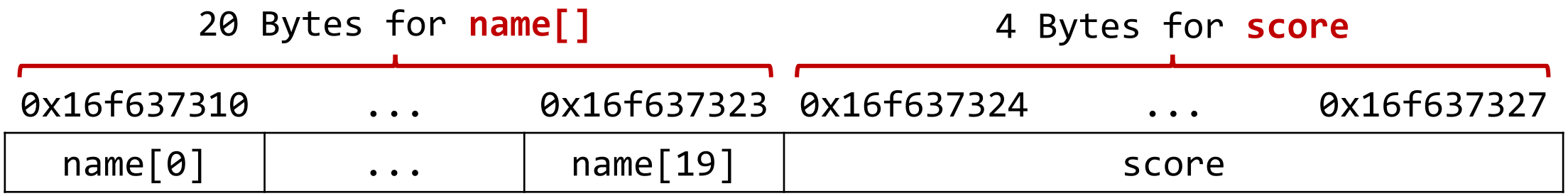
```c
#include <stdio.h>

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main() {
    int a = 2, b = 3;
    swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
```

Structures

- You can define a new data type that can store related information together



main.c

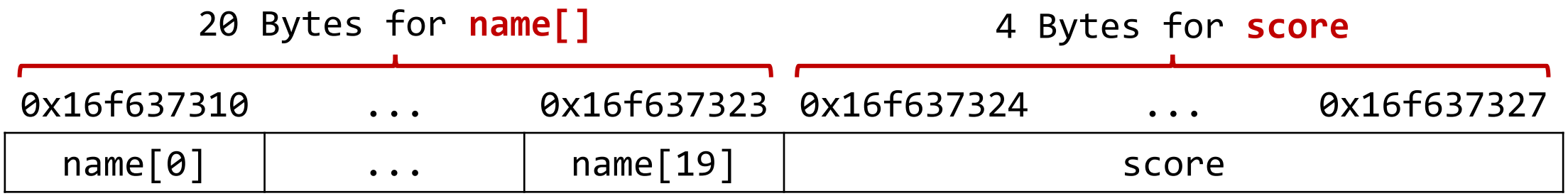
```c
struct Student {
    char name[20];
    int score;
};

int main() {
    struct Student a = {"John", 85 };
    printf("name: %s / score: %d\n", a.name, a.score);
    return 0;
}
```

You can access the fields of the structure by `.[field_name]`

Structures

- You can define a new data type that can store related information together



main.c

```c
struct Student {
    char name[20];
    int score;
};

int main() {
    struct Student a = { "John", 85 }, *ptr = &a;
    printf("name: %s / score: %d\n", ptr->name, ptr->score);
    return 0;
}
```

You can access the fields of the structure pointer by `->[field_name]`

Dynamic Memory Allocation

- `malloc(size)` - allocate `size` bytes consecutively & return its address
- `free(address)` - release the allocated memory
  - You must free the dynamically allocated memory after using it!

main.c

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

int main() {
    int n = 3, *ptr;
    ptr = (int *)malloc(sizeof(int)*n);
    ptr[0] = 50; ptr[1] = 100; ptr[2] = 150;
    for (int i = 0; i < 3; i++) printf("%d\n", ptr[i]);
    free(ptr);
    return 0;
}
```

Practice

- Declare a Fibonacci array `arr` of `n` integers …
  - `arr[i] = arr[i-1] + arr[i-2]` if `i > 1`
  - `arr[0] = 1, arr[1] = 1`

- Procedure
  - Read `n` from user inputs using `scanf()`
  - Allocate memory for a `n`-integer array using `malloc()`
  - Set values for the array elements using `for` loop
  - Print each element in a single line using `printf()`
  - Free the array memory using `free()`

Coding Tips

- **Write clean code** for better readability
  - Follow coding/naming conventions
  - Use comments
  - …
- **Test your own code** on your machine
  - Reading documents is not enough to learn programming
  - Copying a code from the internet is not helpful
  - …

Any Questions?