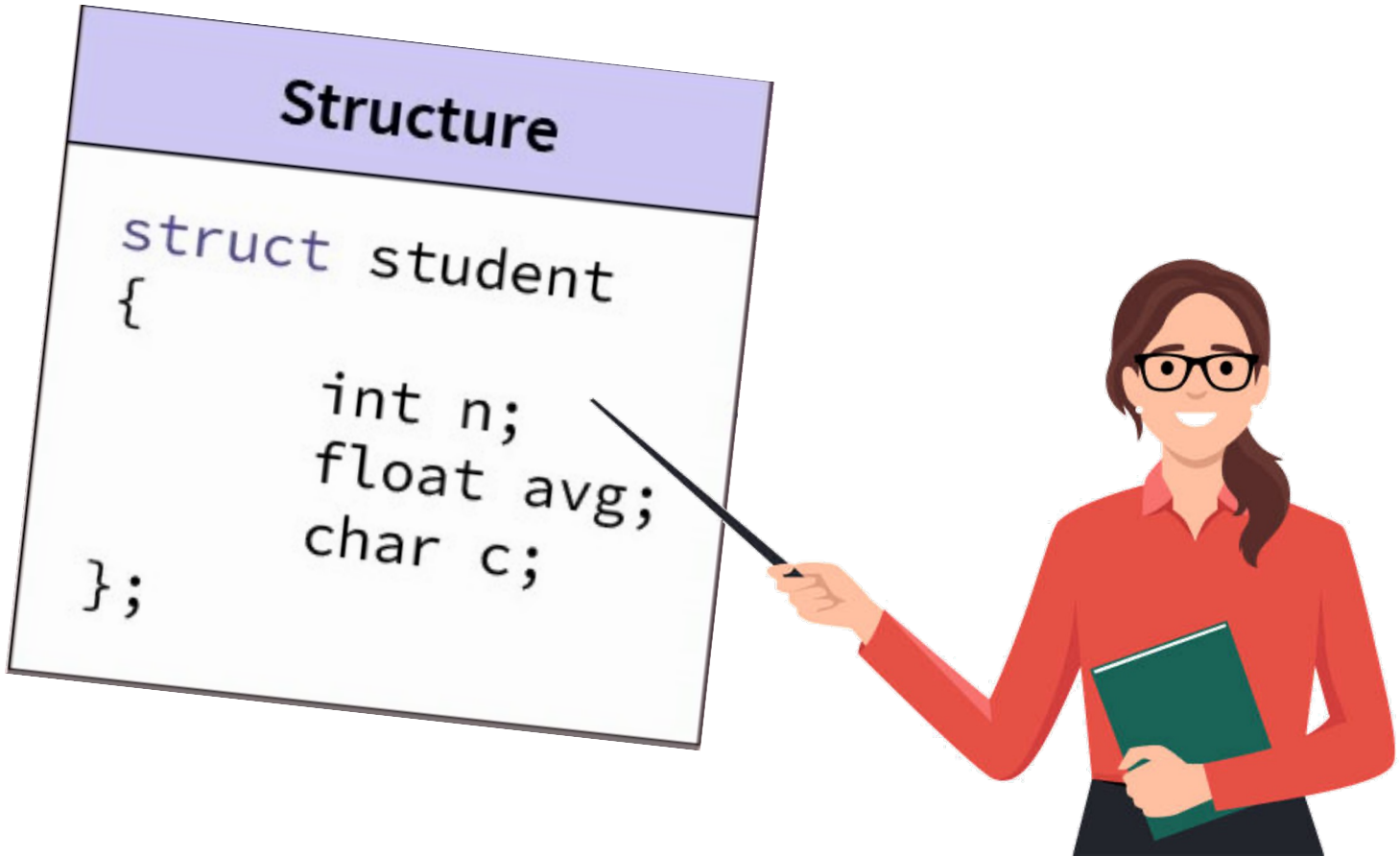
ALGORITHM AND COMPUTATIONAL THINKING 2

WEEK 7 – Structures



Session **objectives**

- ✓ Differentiate **initialization** vs **assignment**

  Understand the behavior on primitives, arrays, pointers, and structs.

- ✓ Manipulate **structs** to group related data

  Define, initialize, and access struct members.

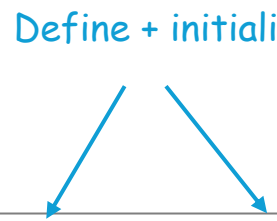
  use typedef for cleaner syntax.

- ✓ Work with **arrays and structs**
- ✓ Pass **structs** to **functions**

EXPLAIN

Variable **Initialization**

= Giving a variable its first value **during** the definition.



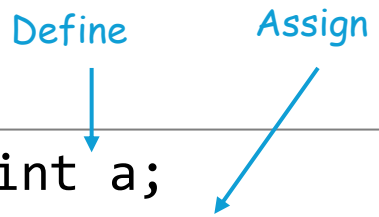

```
int a = 3;                          // ✅ Initialization

int arr[5] = {1, 2, 3, 4, 5};       // ✅ Initialization
```

- ✓ The compiler allocates memory and fills it with the given values at compile time.
- ✓ This is not assignment — it's the creation of variable with a defined initial state.

EXPLAIN

Variable **Assignment**

= Updating a variable's value **after** the definition.




```
int a;
a = 45;
int arr1[5];
arr1 =  {1, 2, 3, 4, 5};
```

✅ **int type is assignable**

❌ **array type is not assignable**

- ✓ C does not support assignment of array types once the array has been declared.
- ✓ Even though both arrays have the same size and type, **the = operator does not apply to arrays**.

EXPLAIN

Initialization

```
int a = 3;
```

Assignment

```
a = 3;
```

	INT, FLOAT, BOOL, CHAR	POINTER	ARRAY	STRING
Initialization	✅	✅	✅	✅
Assignment	✅	✅	❌	❌

*Assignment operation work when C **knows how to copy** the value from one memory location to another.*

EXPLAIN

How to **assign arrays**?

Since we can't use = to assign one array to another after declaration..

LOOP ON ELEMENTS

```
for (int i = 0; i < N; ++i)
        arr2[i] = arr1[i];
```

USE MEMCOPY


```
memcpy ( arr2, arr1, sizeof(arr1));
```

Copies the values of N bytes from a source to a destination.

EXPLAIN

What is a **Structure**?

Structs are a way to group several variables into one place.

Each variable in the structure is known as a **member of the structure**.

Structure

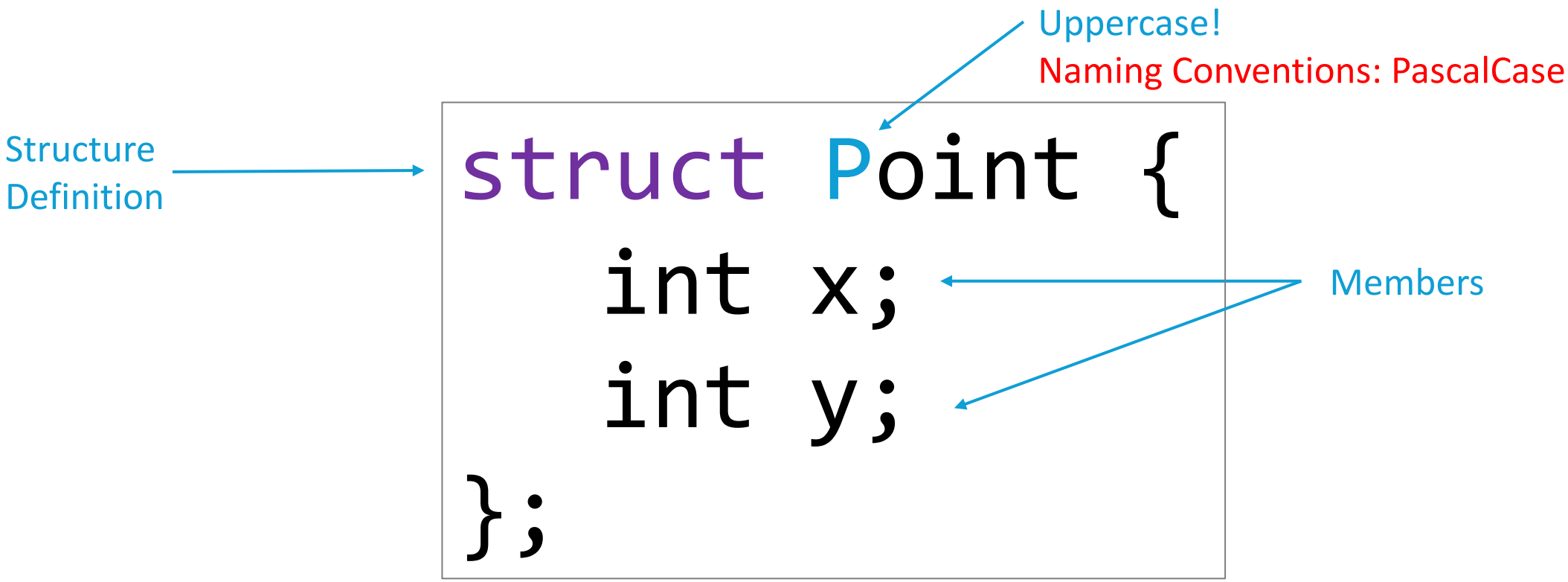
```
struct student
{
    int n;
    float avg;
    char c;
};
```

Unlike an array, a structure can contain many **different data types**.

EXPLAIN

Create a **Structure Definition**

A struct definition creates a **new data type**



- ✓ We **not create any variables** or **allocate any memory** yet.
- ✓ We just describe what a Point looks like: two integers named x and y.

EXPLAIN

Create a Structure Variable

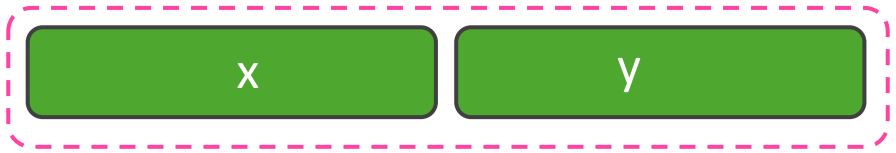
Once you have the **struct definition**, we can use it to **create a variable**.

Structure Definition

```
struct Point {
    int x;
    int y;
};
```

```
struct Point p1;
```

Structure Variable



We allocate a space for two integer

EXPLAIN

Create a Structure Variables

We can create many variables from the same struct definition.

```
struct Point {
    int x;
    int y;
};
```

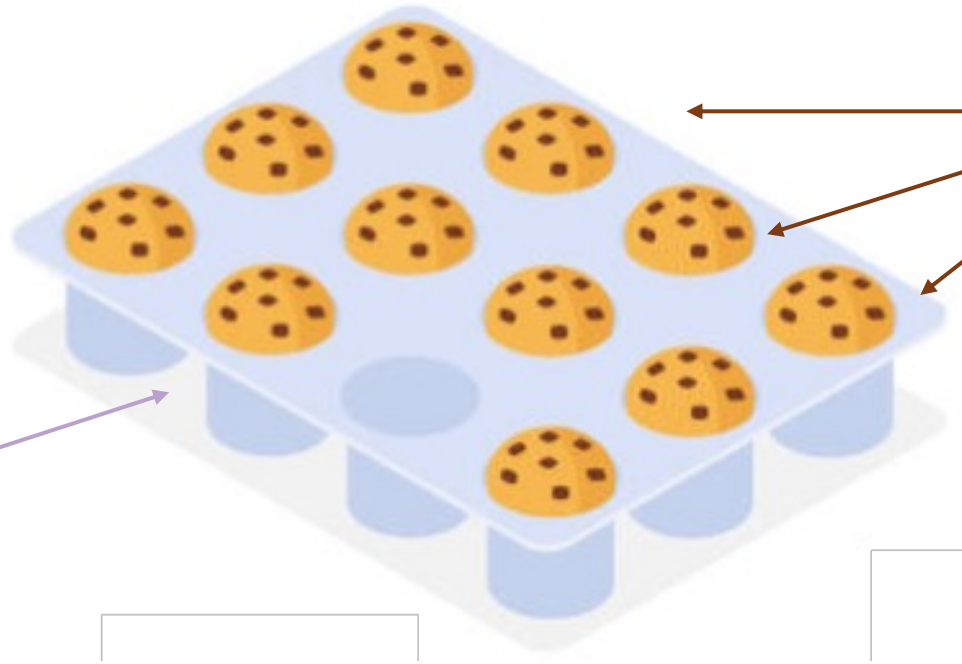
```
struct Point p1;
struct Point p2;
struct Point p3;
```

Structure Variable Definition

EXPLAIN

A structure is a like a **blueprint !**

Form 1 structure definition you can create many structure variables



Structure Variables

Structure definition

EXPLAIN

Assign **structure members**

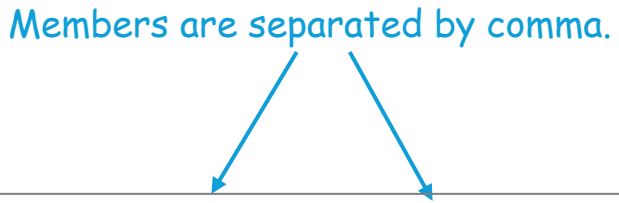
To access and set members of a structure, use the dot syntax (.)

```
struct Point p1;
p1.x = 13;                ✅ Assignment
p1.y = 4;                 ✅ Assignment

struct Point p2;
p2.x = 13;                ✅ Assignment
p2.y = 4;                 ✅ Assignment
```

EXPLAIN

Structure & Initialization

Initialization is **allowed** for structures




```
struct Point p2 = {10, 20};   // ✅ Initialization
```

- ✓ The compiler allocates memory and fills it with the given values at compile time.
- ✓ This is not assignment — it's the creation of variable with a defined initial state.

ACTIVITY

Let's **try**

- ✓ **Create 1 structure** *(a student, a house, a car…)*
  - The structure shall have 3 members at least.

- ✓ **Create 2 variables** from this structure
  - 1st variable : **assign members value** after declaration.
  - 2nd variable : **initialize members** during declaration.

```
struct Point {
        int x;
        int y;
};
```

```
struct Point p1;
p1.x = 13;
p1.y = 4;
```

```
struct Point p1 = {13,4}
```

EXPLAIN

Structures & **Typedef**

To avoid repeating the keyword struct, we can use **typedef** syntax.

BEFORE

```
struct Point {
      int x;
      int y;
};

struct Point p1;
struct Point p2;
```

Valid but repeative

AFTER

```
typedef struct {
      int x;
      int y;
} Point;

Point p1;
Point p2;
```

Point is the struct { int x, int y;}

EXPLAIN

Structures & **Arrays**

As explained before, array (or string) assignment is not possible.

```
typedef struct {

   char group;

   char name[100];

   int scores[30];

} Student;
```

```
Student s1;

s1.group = 'S';

s1.name = "Some text";

s1.scores = {1,2,3};
```

✅ `char is assignable`

❌ `string is not assignable`

❌ `array is not assignable`

How should we assign struct member of type string or array?

EXPLAIN

Assign **array members**

Since we can't use = to assign one array to another after declaration.

Using the initialization phase

Initialization is allowed for array and string

```
Student s1= {'S', "Hello", {1,2,3}};
```

Using memcopy

```
memcopy(s1.scores, {1,2,3}, sizeof(s1.scores);
```

Using strcpy (for string only)

```
strcpy(s1.name, "Some text");
```

EXPLAIN

Struct variables can be **copied**

As long as the source and the target structures are the same.

```
typedef struct {
   int age;
   int score;
} Person;

Person p1 = {22, 100};

Person p2 = p1;

p2.age = 23;
```

We can copy a structure Person into another Structure Person

p1 struct members will be copied byte by byte

P2 changes, P1 does not change

EXPLAIN

Structures Copy & Arrays

Structure member of **type array** are also copied

```
typedef struct {
        char name[10];

} Person;

Person p1 = {"Ronan"};

Person p2 = p1;

p2.name[0] = 'M';
```

A copy of the string (array of char) is performed

P2 changes, P1 does not change

EXPLAIN

Structures Copy & Pointers

Structure member of **type pointer** still point to the same address

```
typedef struct {
        int* ptrScore;

} Result;

int score = 95;
Result result1 = {&score};

Result result2 = result1;

*(p2.ptrScore) = 96;
```

The new pointer still points to the address of score

The score changes

QUIZ

What will this code print?

```
typedef struct {
        char name[10];
} Person;

typedef struct {
        char name[10];
} Student;

Person p1 = {"Ronan"};
Student s2 = p1;
printf("%s", s2.name);
```

A. Ronan
B. (empty string)
C. Compile-time error
D. Garbage value

ANSWER

What will this code print?

```
typedef struct {
        char name[10];
} Person;

typedef struct {
        char name[10];
} Student;

Person p1 = {"Ronan"};
Student s2 = p1;
printf("%s", s2.name);
```

A. Ronan
B. (empty string)
C. Compile-time error
D. Garbage value

C does not allow direct assignment between different struct types.

QUIZ

What is the **size** (in bytes) of the structure S1?

```
struct S1 {
    int x;
    char y;
};
```

A. 5

B. 6

C. 8

D. 9

ANSWER

What is the **size** (in bytes) of the structure S1?

```
struct S1 {
    int x;
    char y;
};
```

A. 5

B. 6

C. 8

D. 9

- Int x take 4 bytes
- Char 4 takes 1 byte

QUIZ

What is the **size** (in bytes) of the structure S2?

```
typedef struct {
    int x;
    char y;
} S1;

typedef struct {
    S1 a;
    char z;
} S2;
```

A. 5

B. 6

C. 8

D. 9

ANSWER

What is the **size** (in bytes) of the structure S2?

```
typedef struct {
    int x;
    char y;
} S1;

typedef struct {
    S1 a;
    char z;
} S2;
```

A. 5

B. 6

C. 8

D. 9

- S1 a — take 5 bytes
- Char z — `take another 1 byte

QUIZ

Given the following structure and array declaration, what is the size of arr?

```
struct Point {
    int x;
    int y;
};

struct Point arr[5];
```

A. 20

B. 40

C. 10

D. Depends on pointer size

ANSWER

Given the following structure and array declaration, what is the size of arr?

```
struct Point {
    int x;
    int y;
};

struct Point arr[5];
```

A. 20

B. 40

C. 10

D. Depends on pointer size

- Each Point has two ints → 8 bytes
- arr[5] → 5 × 8 = 40 bytes

QUIZ

What is the size of the following structure?

```
typedef struct {
    char type;
    int id;
} Tile;

typedef struct {
       Tile tiles[3];
} Map;
```

A. 15

B. 18

C. 24

D. 20

ANSWER

What is the size of the following structure?

```
typedef struct {
    char type;
    int id;
} Tile;

typedef struct {
      Tile tiles[3];
} Map;
```

A. 15

B. 18

C. 24

D. 20

- Tile = 1 (char) + 4 (int) = 5 bytes
- 3 × 5 = 15 bytes

EXPLAIN

Passing **structures** to **functions**

Structures are **passed by values** to functions.

```
typedef struct {
    int result;
    char feedback[100];
} Score;


int main() {
       Score score = {50, "ronan"};
       process(score);
}
```

```
void process(Score s) {
    s.result = 99;
}
```

A COPY of score is sent to the function

Score variable does NOT change

Any modifications for the structure inside the function will not affect the original struct.

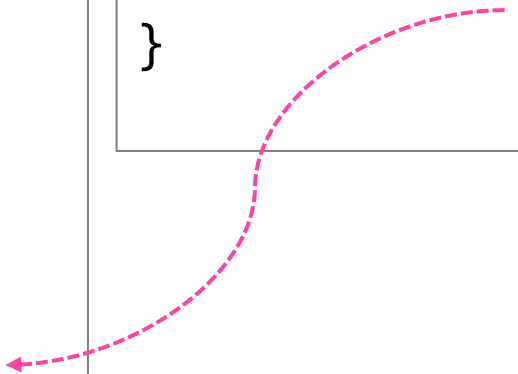
EXPLAIN

Returning **structures** from **functions**

A function can return a structure (by value)

```
typedef struct {
    int x;
    char y
} Point2D;



int main() {
   Point2D position = getRobotPosition();
   printf("%d", position.x);
}
```

```
Point2D getRobotPosition(){
    Point2D result = {50, 45};
    return result;
}
```



Returning Structure can allow to get more information from a function call.

EXPLAIN

Passing **structures** as a pointer

Structures can also be passed as pointer

```
typedef struct {
    int result;
    char feedback[100];
} Score;


int main() {
       Score score = {50, "ronan"};
       process(&score);
}
```

```
void process(Score* s) {
    (*s).result = 99;
}
```

A reference of score is sent to the function

Score variable Will change as we use a pointer

REFLECT

The root cause of why **arrays are not assignable** in C.

Unlike other types, arrays can be assigned or copied directly… Why ?

It's a question of type size prediction!

PRIMITIVES & STRUCTS

Primitive types specifies their size:

```
int a;          // 4 bytes
```

Structure types also specifies their size:

```
typedef struct {
   int age;
   int scores[3];
} Student;  // 16 bytes
```

The compiler knows exactly how many bytes to copy

ARRAY & STRING

Arrays types do not always specify their size

```
void copy(int dest[], int src[]) {}
```

The compiler cannot guarantee that the size is known in every context

What you **know now**

- ✓ Differentiate **initialization** vs **assignment**

  Understand the behavior on primitives, arrays, pointers, and structs.

- ✓ Manipulate **structs** to group related data

  Define, initialize, and access struct members

  use typedef for cleaner syntax

- ✓ Work with **arrays and structs**
- ✓ Pass **structs** to **functions**