

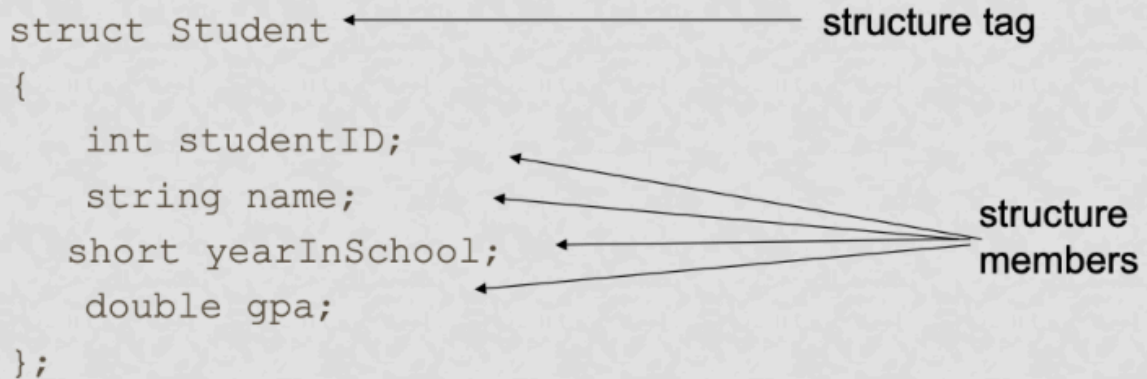
What is Structure?

- **Structure** is a C++ construct that allows multiple variables to be grouped together
 - `struct` must end with a `;` unlike functions and classes

General Format:

```
struct <structName>
{
    type1 field1;
    type2 field2;
    . . .
};
```

EXAMPLE STRUCT DECLARATION



The diagram shows a C++ struct declaration for a 'Student' structure. The code is as follows:

```
struct Student
{
    int studentID;
    string name;
    short yearInSchool;
    double gpa;
};
```

Annotations with arrows point to specific parts of the code:

- An arrow points from the text "structure tag" to the word "Student" in the struct declaration.
- Four arrows point from the text "structure members" to the four member declarations: "int studentID;", "string name;", "short yearInSchool;", and "double gpa;".

- Multiple fields with the same type can be created in one line
 - ex. `string name, address` creates 2 string objects.
- Struct does not actually allocate memory or variables, to define them you must call the structure.
 - Seems like a class in nature, but there is no initialization.
- You can not directly print the `struct`, but the values inside yes
- `struct` variables can be initialized when defined:

- `Student s = {11465, "Joan", 2, 3.75};`

- Also through individual means (like `bill.name = "blah;"`) works too

- `Structs` can be defined in arrays and then be accessed like an array

```
const int NUM_STUDENTS = 20;
Student stuList[NUM_STUDENTS];
```

- `stuList[4].name` would work

Deeper into a Structure

- **Nested Structures** work as a structure can contain another
- Array variables can be accessed as easily too
 - ex. `strcpy(me.name, "Lebron");`
 - `struct` variables and `the entire structure` itself could be used as inputs
- Structures has a memory address and inside structures could hold pointers.
- A dereferenced struct pointer must be used in order to be referred to
 - ex. `cout << (*pointerVar).studentID;`
 - This itself can be simplified into `cout << pointerVar->student.id`

UNIONS???

- **Unions**, similar to structures, share a single memory location and only one member of the union can be used at a time
- In formatting and declaration, a `union` is the same as `struct`
- Memory is allocated at declaration time, members can be called directly without dot operator (`yo.gurt`)
 - ex.

```
enum Day { MONDAY, TUESDAY,
           WEDNESDAY, THURSDAY,
           FRIDAY };
```

- In this situation when we call Day (like `Day today;`) could either have the values of MONDAY, TUESDAY... and so on.
- This works as each potential variable acts as if it was a `const int`

In memory...

```
MONDAY = 0
TUESDAY = 1
WEDNESDAY = 2
THURSDAY = 3
FRIDAY = 4
```

- You can not directly assign a variable with this `enum` a value,

```
workDay = 3; // Error!
```

Instead, you must cast the integer:

```
workDay = static_cast<Day>(3);
```

- You CAN assign enumerator to an int variable.
 - ex. `x = THURSDAY` -> `x= 3`
- `enum` can also be used in a forloop
- An anonymous enum can be declared by just calling `enum {...}`
- `enum` variables can only be declared once, however the `enum` itself can be declared multiple times