Structures

Terms

Enumerations
Enumerators
Methods
Nested structures
Operator overloading
Strongly-typed enums
Structures
Structure members

Summary

- We use *structures* to define custom data types.
- Members of a structure can be variables or functions (also called *methods*).
- Structures can be nested to represent more complex types.
- To compare two structures, we have to compare their individual members.
- We can provide operators for our structures using a technique called *operator overloading*.
- Just like the built-in data types, structures can be used as function parameters or their return type.
- Using an *enumeration*, we can group related constants into a single unit. Members of this unit are called *enumerators*.
- *Strongly-typed enumerations* define a scope for their members. This allows two enums having members with the same name.


```cpp
// Defining a structure
struct Movie {
    string title;
    int releaseYear = 1900;
};


// Creating an instance of a structure
Movie movie = { "Terminator 1", 1984 };

// Unpacking a structure
auto [title : string, releaseYear : int] = movie;


// Operator overloading
bool operator==(const Movie& first, const Movie& second) {
    return (
        first.title == second.title &&
        first.releaseYear == second.releaseYear
    );
}

// Pointer to a structure
void showMovie(Movie* movie) {
    // Structure pointer operator
    cout << movie->title;
}
```



```cpp
// Classic (unscoped) enumeration
enum Action {
    list,
    add,
    update
};


// Strongly-typed enumeration
enum class Operation {
    list,
    add,
    update
};


// Using enumerations
void doSomething(Operation operation) {
    if (operation == Operation::add) {
        // ...
    }
    else if (operation == Operation::list) {
        // ...
    }
}
```

