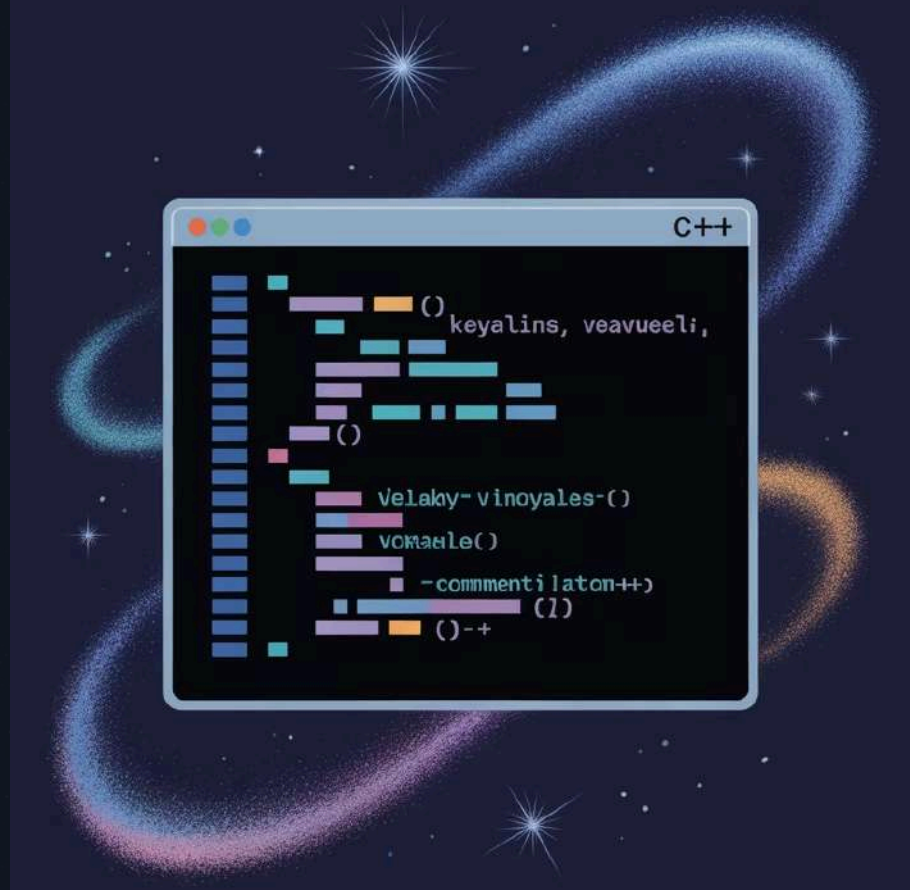
Structures and Records in C++

Exploring user-defined data types for creating complex objects



📘 Lecture: Structures and Records (C++)

What is a structure?

A user-defined data type that combines different fields into a single entity. Analogous to a "record" in algorithms.

Why are they needed?

To model real-world objects: student, book, point, bus, and other complex entities with multiple characteristics.

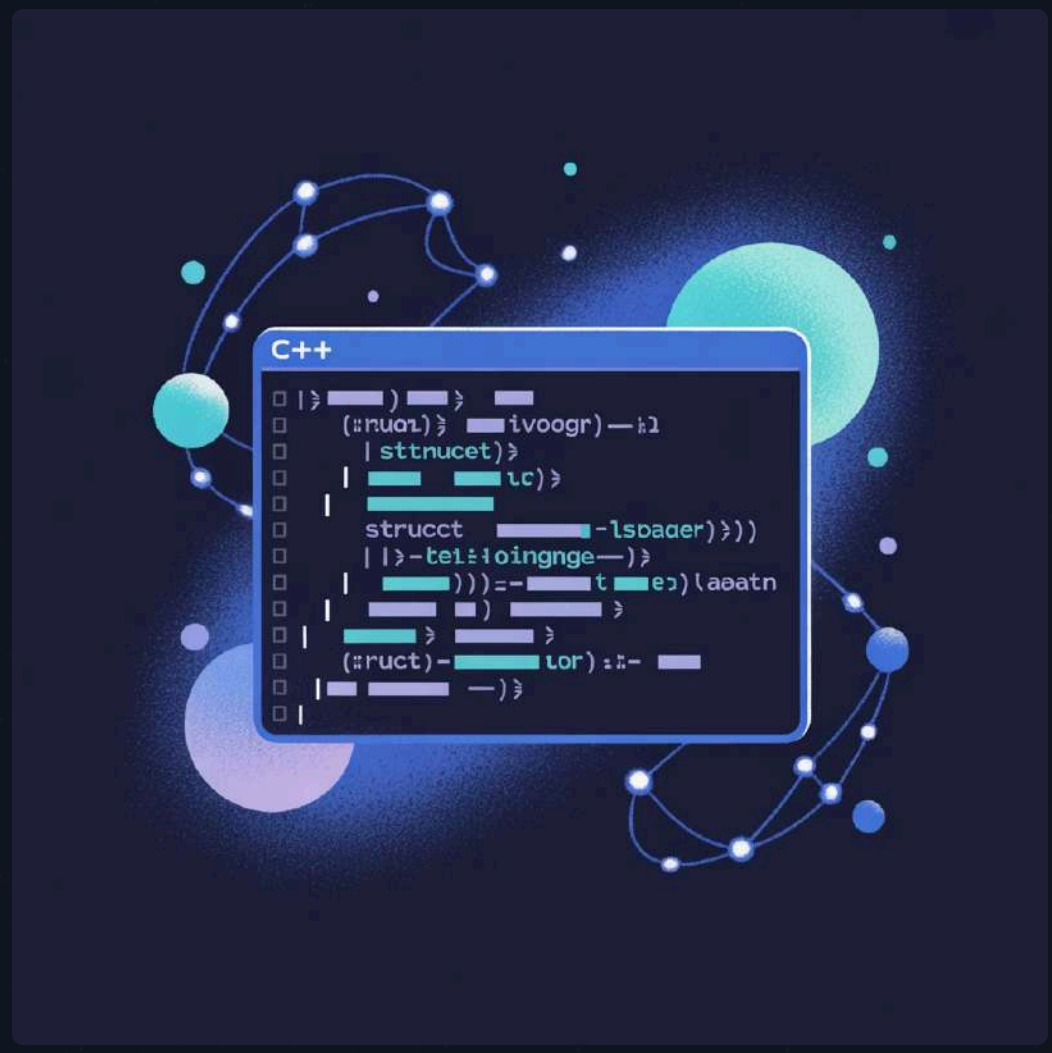
Structures allow you to create logically related groups of data, which makes code more organized and understandable. This is a fundamental tool for building complex programs.

Declaring and Using Structures

Struct Syntax

Structures are declared using the keyword `struct`. Unlike classes, all members of a structure have a `public` access modifier by default.

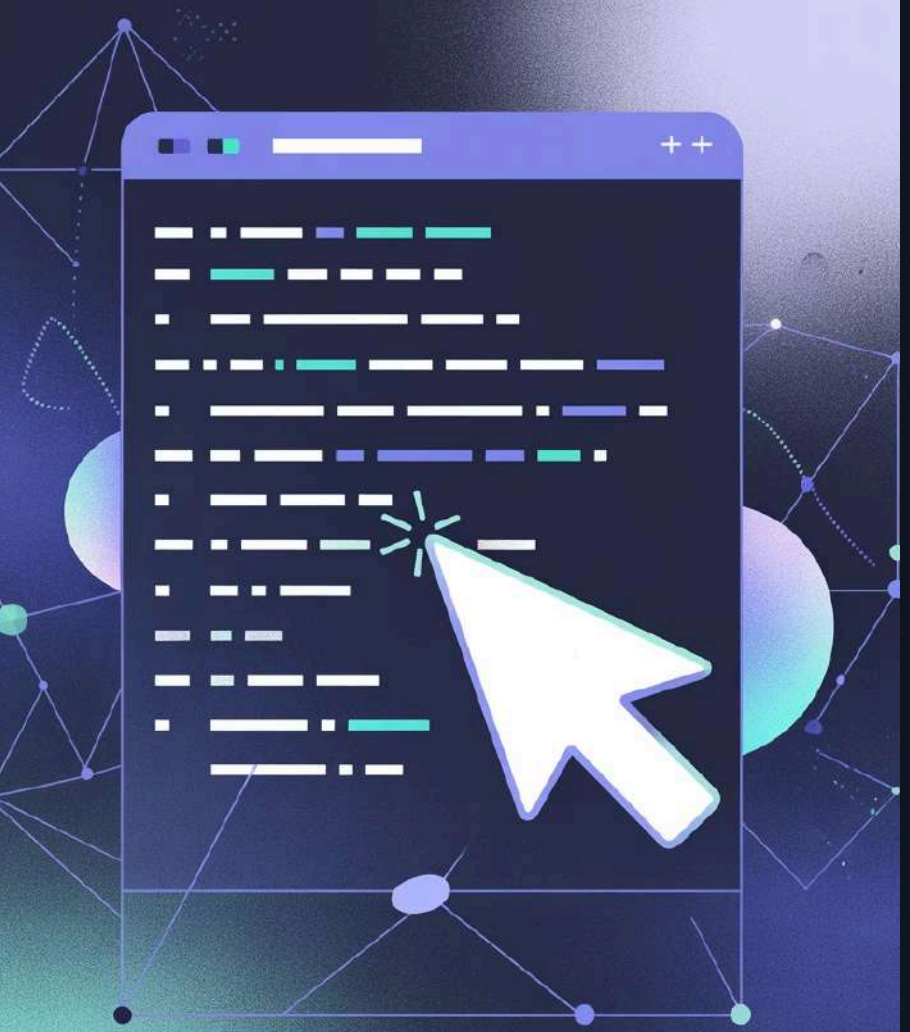
This makes structures ideal for simple data containers where there is no need to hide internal implementation.



Example 1: Simple Structure

```
#include
using namespace std;

struct Student {
    string name;
    int age;
    double gpa;
};

int main() {
    Student s1 = {"Aigerim", 18, 3.9};
    cout << s1.name << " (" << s1.age << ") GPA = " << s1.gpa << endl;
    return 0;
}
```

This example demonstrates creating a Student structure with three fields and initializing it with a list of values.



Accessing Structure Members

Dot Operator (.)

Used for direct access to the members of a structure object. The most common way to work with members.

Arrow Operator (->)

Applied when working with a pointer to a structure. A convenient alternative to dereferencing the pointer.

Example 2: Working with a pointer to a structure

```
Student s2 = {"Dana", 19, 3.5};
Student *p = &s2;

cout << p->name << endl;      // access via ->
cout << (*p).gpa << endl;     // alternative
```

The `->` operator is syntactic sugar and is equivalent to `(*p).member`, but is significantly more convenient to use.

Structures as Function Parameters

Pass by Value

When passing by value, an entire copy of the structure is created. This is safe, but can be inefficient for large structures due to copying overhead.

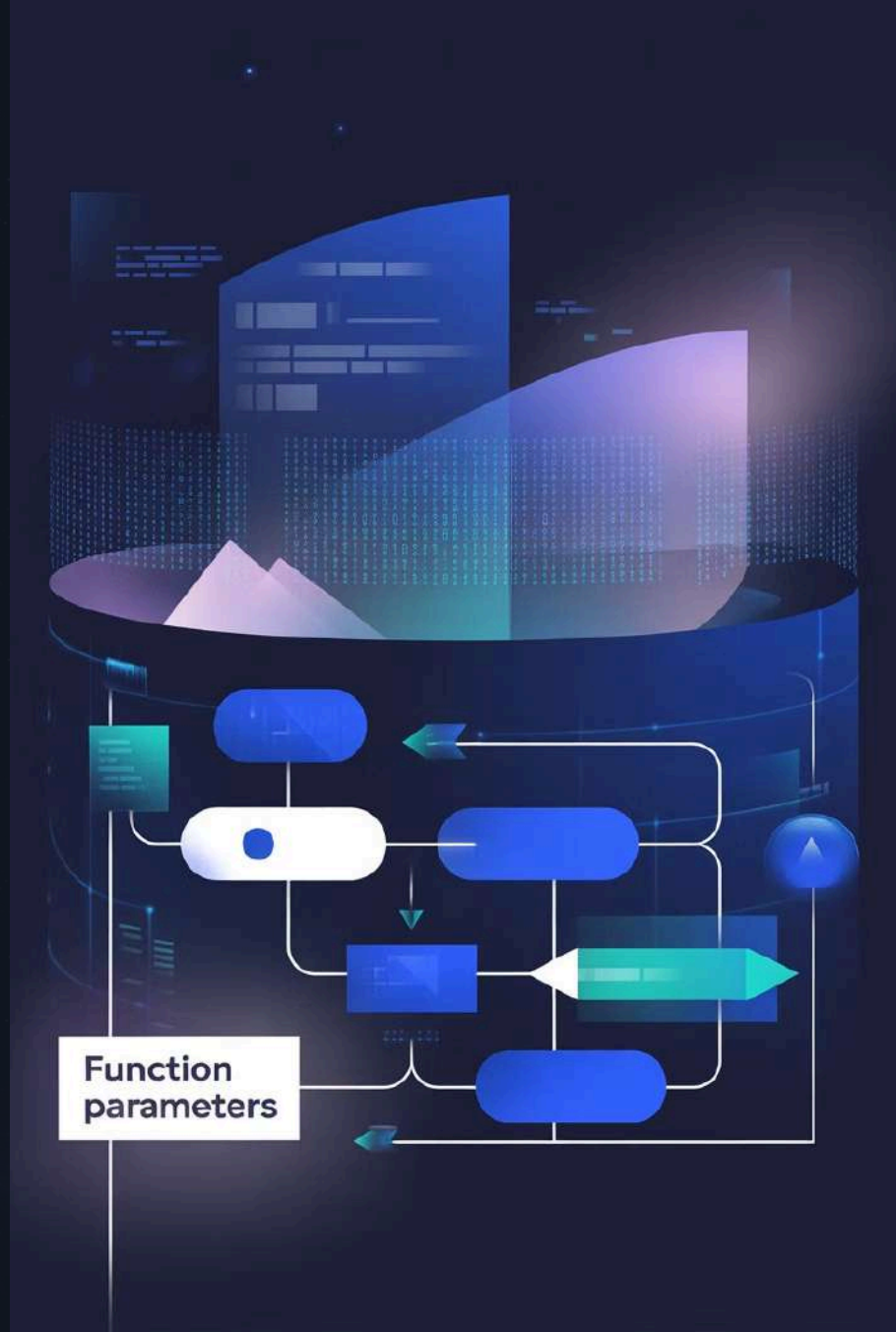
Pass by Reference

A more efficient approach is to pass by reference. Use `const` to protect against accidental changes if the function only reads the data.

Example 3: Function with a Structure Parameter

```
void printStudent(const Student &s) {
    cout << s.name << " (" << s.age << ") GPA = " << s.gpa << endl;
}
```

Using `const Student &s` ensures efficiency (no copying) and safety (no accidental changes).



Arrays of Structures

Arrays of structures allow storing collections of homogeneous objects, which is especially useful for working with lists of students, products, database records, and other data sets.

Example 4: array of structures

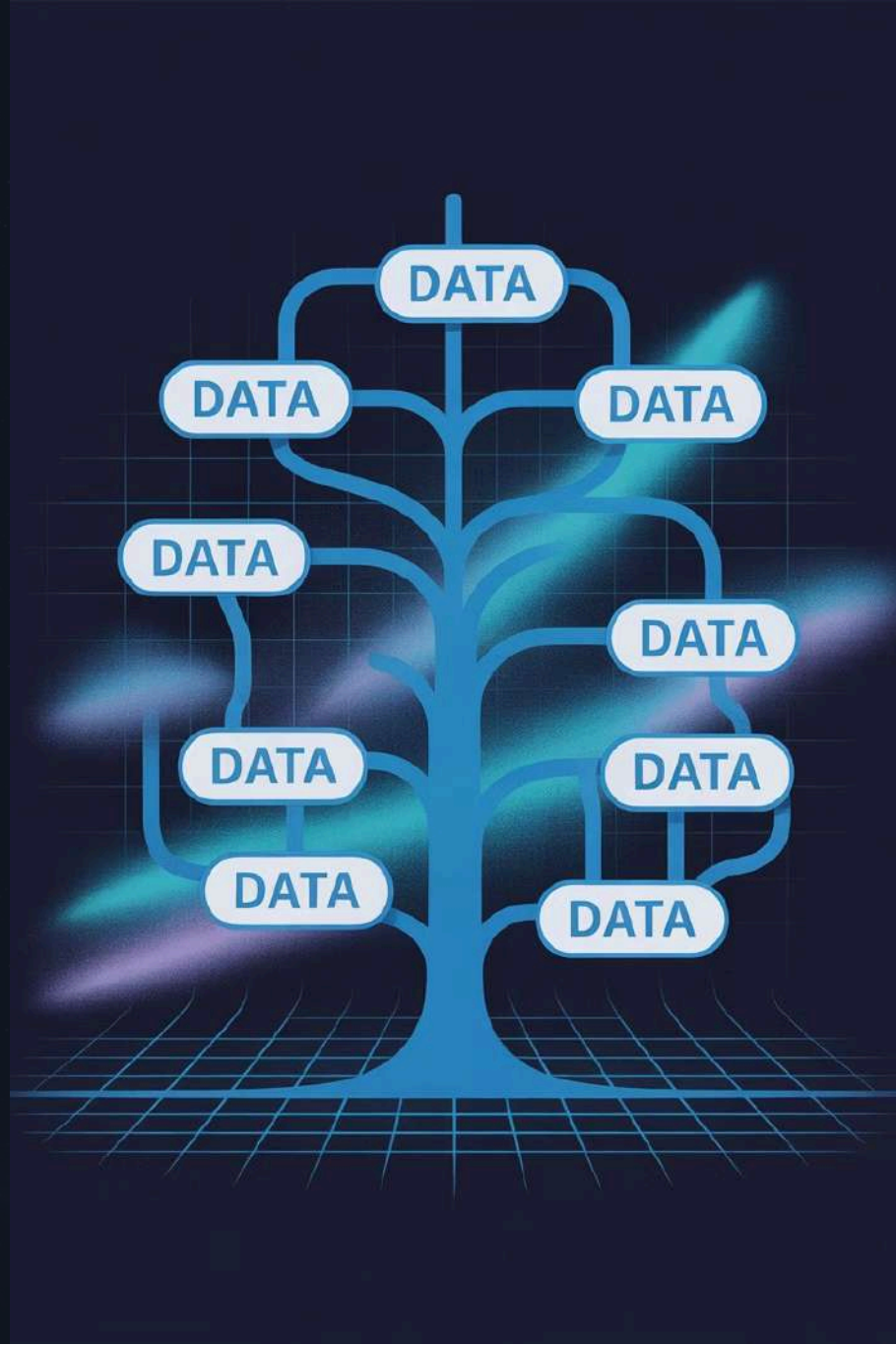
```
Student group[3] = {
    {"Ali", 18, 3.8},
    {"Mira", 20, 3.6},
    {"Omar", 19, 3.9}
};

for (int i = 0; i < 3; i++) {
    cout << group[i].name << " " << group[i].gpa << endl;
}
```

Advantages of Arrays of Structures:

- Uniform data storage
- Ease of processing in loops
- Efficient memory usage

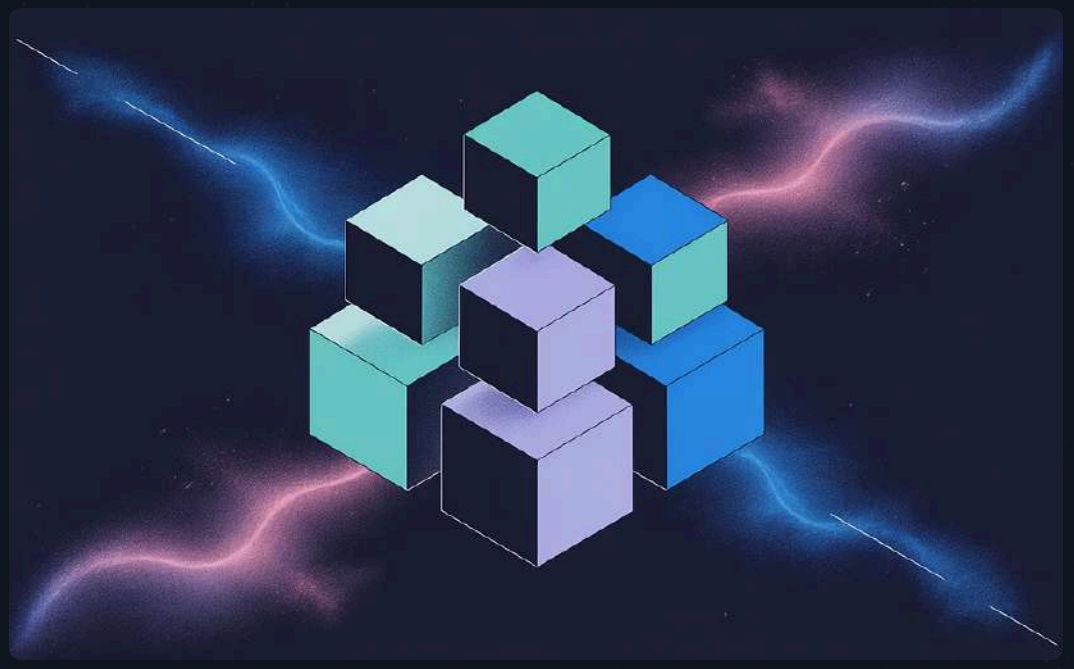
Applications:

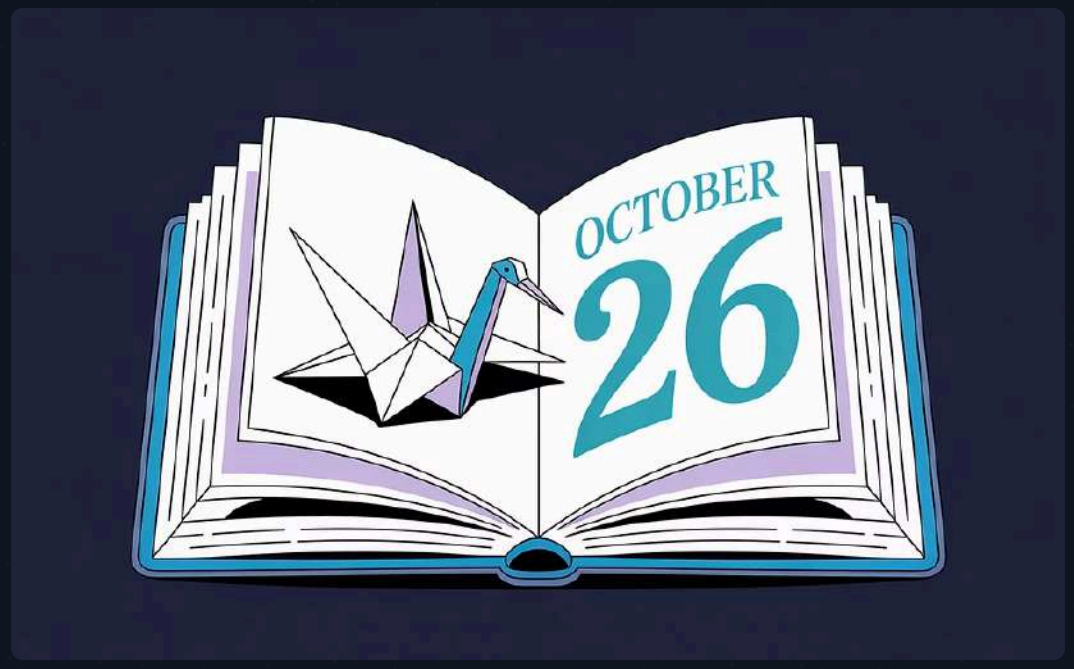
- Lists of students in a group
- Product catalogs
- Coordinates of points



Nested Structures

A structure can contain another structure as its field. This is a powerful mechanism for modeling complex real-world objects with a hierarchical data structure.





Object Composition

Nested structures allow you to create objects that consist of other objects, reflecting natural relationships in the problem domain.

Practical Application

A book contains a publication date, a student has an address, an order includes customer information – all these cases are resolved by nested structures.

Example 5: Nested Structures

```
struct Date {
 int day, month, year;
};

struct Book {
 string title;
 string author;
 Date published;
};

Book b = {"C++ Basics", "Bjarne Stroustrup", {12, 5, 1985}};
cout << b.title << " published on " << b.published.day
 << "." << b.published.month << "." << b.published.year << endl;
```

Structures and Dynamic Memory

Dynamic memory allocation for structures is necessary when the size of data is unknown at compile time or when objects need to be created during program execution.

Memory Allocation

Use the `new` operator to create a structure in dynamic memory

Working with Data

Access fields using the `->` operator or pointer dereferencing

Memory Deallocation

Always call `delete[]` for arrays or `delete` for individual objects

Example 6: Dynamic Array of Structures

```
struct Student {
    string name;
    int age;
    double gpa;
};

Student *students = new Student[2];
students[0] = {"Aset", 18, 3.4};
students[1] = {"Karina", 19, 3.7};

for (int i = 0; i < 2; i++) {
    cout << students[i].name << endl;
}

delete[] students; // Important: memory deallocation!
```

Lecture Summary

Data Unification

Structures are a means of combining different data types into one logical object, which simplifies working with complex entities.

Access Methods

Fields are accessed via the `.` (dot) and `->` (arrow) operators, depending on whether you are working with an object or a pointer.

Versatility

Structures are convenient for creating arrays, dynamic collections, nested objects, and passing complex data between functions.

Structures are a fundamental C++ tool for creating custom data types. They provide the basis for object-oriented programming and allow you to write more organized and readable code.

Review Questions

Arrays and Structures

What is the fundamental difference between a regular array and an array of structures? What advantages does using an array of structures provide for storing related data?

Parameter Passing

When is it better to pass a structure by value, and when by reference? What factors influence the choice of parameter passing method?

Structures and Classes

How are structures similar to classes, and how do they fundamentally differ? In what cases is it preferable to use structures?

Additional Questions:

- How does list initialization of structures work?
- Can one structure be copied to another?
- What happens to memory when passing by value?

Practical Tasks:

- Create a "Point" structure with x, y coordinates.
- Write a function to calculate the distance between points.
- Implement sorting of an array of structures.

Practical Assignment

Ready to apply your knowledge in practice? I propose creating a more complex example to consolidate the material!

Assignment: "Bus Fleet" System

Create a `Bus` structure with fields: route number, passenger capacity, current number of passengers, status (en route/at stop).

Array and Search

Create an array of several buses, implement functions for searching by route number and counting available seats in the entire bus fleet.

Extended Functionality

Add functions for boarding/alighting passengers with capacity checks and displaying statistics for all routes.

What you will learn:

- Designing data structures
- Working with arrays of structures
- Creating functions for data processing
- Input data validation

Additionally:

- Add a "Stop" structure
- Create a schedule
- Implement a seat reservation system
- Add data saving to a file

Good luck in learning C++ structures! This is a powerful tool that will open the way for you to create complex and efficient programs.