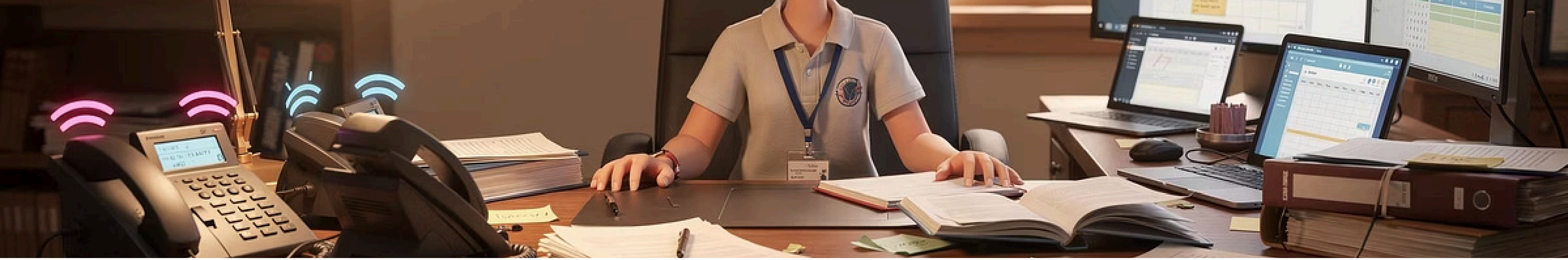
# Introduction to Classes in C++

Learn object-oriented programming through the lens of managing campus organizations at an HBCU. Discover how classes help organize complex information and create powerful, maintainable code.



SCENARIO

# Managing Your Campus Organization

Imagine you're president of your university's Student Government Association (SGA). This semester, you're coordinating Homecoming with 12 organizations, planning MLK Day of Service with 200+ volunteers, organizing Spring Career Fair with 40+ companies, and managing an executive board of 15 officers. You're also tracking a $50,000 annual budget across multiple initiatives.

You've been tracking everything in different places: member contact info in your phone, committee assignments in Google Docs, event budgets in notebooks, and RSVPs in group chats. With Homecoming just two months away, the chaos is overwhelming. There has to be a better way to organize all this information.

## The Solution: Classes in Programming

This organizational challenge is exactly what **classes** in programming help solve. A class is like a blueprint or template for organizing related information and actions together.

Just as your SGA has certain characteristics (name, number of members, budget) and can perform certain actions (hold meetings, plan events, recruit members), a class in C++ lets you define both the **data** (characteristics) and **functions** (actions) that belong together.

### Why Classes Matter

Classes bring order to complexity by grouping related information and behaviors into a single, manageable structure. They're the foundation of object-oriented programming.

LEARNING GOALS

## What You'll Learn

01

### Understand Classes

Learn what a class is and why it's useful for organizing code

02

### Identify Components

Recognize member variables and member functions within classes

03

### Write Class Definitions

Create basic class definitions in C++ with proper syntax

04

### Create Objects

Instantiate and use objects from your class definitions

05

### Apply Access Control

Understand and use public and private access specifiers

## What is a Class?

A **class** is a user-defined data type that groups together related data and functions. Think of it as a blueprint or template—just as an architect creates a blueprint for a house, you create a class as a blueprint for organizing information.

Blueprint (Class)
Actual House (Object)

An **object** is a specific instance created from that class. If the class is the blueprint, then each actual house built from that blueprint is an object. You can create many houses (objects) from one blueprint (class), and each can have different characteristics even though they follow the same basic design.

## Classes and Objects in Action

### The Class (Blueprint)

StudentOrganization is our class—the template that defines what information and actions all student organizations share.

### Object: SGA

Student Government Association is one specific object created from the StudentOrganization class.

### Object: CAB

Campus Activities Board is another object—same blueprint, different characteristics and data.

### Object: Pre-Law Society

Pre-Law Society is yet another instance, demonstrating how one class creates many unique objects.

COMPONENTS

## Key Components of a Class

### Member Variables

The characteristics or attributes of the class

- Organization name
- Number of members
- Annual budget
- President name

### Member Functions

The actions or behaviors the class can perform

- Add members
- Plan events
- Display information
- Update budget

Together, these components create a complete, self-contained unit that models a real-world entity with both state (data) and behavior (functions).

STEP 1

## Class Declaration: The Foundation

Let's start building our StudentOrganization class. In C++, we use the keyword `class` followed by the class name to begin our definition:

```
class StudentOrganization {
    // Class contents will go here
};
```

### Class names start with capital letters

This is a C++ convention that makes classes easy to identify

### Definition ends with semicolon

Don't forget the semicolon after the closing brace—it's required!

### Contents go between braces

Everything between { } belongs to the class definition

STEP 2

## Adding Member Variables

Now let's add member variables to store information about our organization. We'll include the organization name, number of members, and annual budget:

```
class StudentOrganization {
    private:
        string organizationName;
        int numberOfMembers;
        double annualBudget;
};
```

### Understanding Access Specifiers

The `private:` keyword is an *access specifier*. It means these variables can only be accessed from within the class itself—not from outside code. This is a key concept in **encapsulation**, which helps protect data from accidental misuse.

Think of it like your organization's financial records: you wouldn't want just anyone to change your budget numbers. By making these private, we ensure they can only be modified through specific functions we create.

STEP 3

## Adding a Constructor

A **constructor** is a special function that runs automatically when you create an object. It sets up the initial values of your member variables. The constructor has the same name as the class and no return type:

```
class StudentOrganization {
    private:
        string organizationName;
        int numberOfMembers;
        double annualBudget;

    public:
        // Constructor
        StudentOrganization(string name, int members, double budget) {
            organizationName = name;
            numberOfMembers = members;
            annualBudget = budget;
        }
};
```

Notice we added `public:` before the constructor. This means the constructor can be accessed from outside the class—which is necessary because we need to call it when creating objects!

STEP 4

## Adding Member Functions

Let's add functions to make our class useful. We'll create functions to display information, add members, and update the budget:

```
// Display organization information
void displayInfo() {
    cout << "Organization: " << organizationName << endl;
    cout << "Members: " << numberOfMembers << endl;
    cout << "Annual Budget: $" << annualBudget << endl;
}

// Add new members
void addMembers(int newMembers) {
    numberOfMembers += newMembers;
    cout << "Added " << newMembers << " new members!" << endl;
}

// Update budget
void updateBudget(double newBudget) {
    annualBudget = newBudget;
    cout << "Budget updated to $" << annualBudget << endl;
}
```

These public member functions provide controlled access to our private data, allowing us to modify and display information safely.

## The Complete Class Definition

Here's our complete StudentOrganization class with all components working together:

```
class StudentOrganization {
 private:
 string organizationName;
 int numberOfMembers;
 double annualBudget;

 public:
 StudentOrganization(string name, int members, double budget) {
 organizationName = name;
 numberOfMembers = members;
 annualBudget = budget;
 }

 void displayInfo() {
 cout << "\nOrganization: " << organizationName << endl;
 cout << "Members: " << numberOfMembers << endl;
 cout << "Annual Budget: $" << annualBudget << endl;
 }

 void addMembers(int newMembers) {
 numberOfMembers += newMembers;
 }

 void updateBudget(double newBudget) {
 annualBudget = newBudget;
 }
};
```

IN ACTION

## Creating Objects: Bringing Classes to Life

Now that we've defined our class, let's see how to actually use it! Creating an object is called **instantiation**:

```
// Creating objects (instances) of StudentOrganization
StudentOrganization sga("Student Government Association", 15, 50000.00);
StudentOrganization cab("Campus Activities Board", 25, 15000.00);
```

1

### Declare Type

StudentOrganization specifies the class

2

### Name Object

sga and cab are our object names

3

### Pass Values

Values in parentheses initialize the object

Each object is independent—changing sga doesn't affect cab, even though they're both StudentOrganization objects.

# Calling Member Functions

We use the dot operator (`.`) to call member functions on an object. This tells the object to execute its function:

```
// Using the objects
sga.displayInfo();
cab.displayInfo();

// Modifying the SGA
sga.addMembers(3);
sga.updateBudget(55000.00);
sga.displayInfo();
```

## Output Example

```
Organization: Student Government Association
Members: 15
Annual Budget: $50000

Added 3 new members!
Budget updated to $55000

Organization: Student Government Association
Members: 18
Annual Budget: $55000
```

## Key Insight

The dot operator connects an object to its functions. Think of it as saying "Hey sga, display your info!"

PRACTICE

## Challenge 1: Add More Member Variables

Add at least two new private member variables to track additional information. Consider adding:

- `string presidentName` - who leads the organization
- `int yearFounded` - when it was established
- `string meetingDay` - when the organization meets
- `int numberOfEvents` - how many events held this year

Remember: You'll need to update your constructor to initialize these new variables!

## Challenges 2 & 3: Advanced Functions

### Challenge 2: Implement a Budget Management System

Create member functions to:

- Spend money from the budget (with validation to prevent overspending)
- Add funds to the budget
- Calculate the budget per member
- Check if the organization can afford a specific event

### Challenge 3: Member Management

Create member functions to:

- Add new members (updating the count)
- Remove members (with validation to prevent negative counts)
- Calculate what percentage of the target membership has been reached
- Display whether the organization is "growing," "stable," or "declining" based on changes

# Challenges 4 & 5: Build More Features

### Challenge 4: Event Planning

Add functionality to:

- Store a count of planned events
- Add a new event (increment counter)
- Calculate events per member
- Display a message about the organization's activity level

### Challenge 5: Create a Related Class

Think about other aspects of HBCU campus life that could be modeled with classes. Design and implement a new class such as:

- Course (with properties like name, professor, credits, enrollment)
- HomecomingEvent (with properties like name, date, attendees, location, committee)
- GreekOrganization (with properties like chapter name, founding year, members, colors)
- StepShowTeam (with properties like team name, members, practice schedule, competitions won)
- CareerFairCompany (with properties like company name, positions available, contact person)
- CommunityServiceProject (with properties like project name, volunteers needed, hours completed)

## Reflection Questions

### Why make member variables private?

Consider data protection, encapsulation, and controlled access through functions.

### What's the difference between a class and an object?

Think about blueprints versus actual buildings. Can you create another real-world analogy?

### What other classes might be useful for campus life?

Consider courses, homecoming events, Greek organizations, step shows, community service projects, career fairs, and more.

### How do classes improve code organization?

Reflect on maintainability, readability, and modeling real-world entities effectively.

## Key Takeaways

### Classes are Blueprints

A class is a blueprint for creating objects that groups related data and functions together into a cohesive unit.

### Member Functions Define Behavior

Member functions define the behaviors or actions an object can perform, representing what the object can do.

### Constructors Initialize Objects

Constructors automatically initialize objects when they are created, setting up initial values for member variables.

### Member Variables Store State

Member variables store the characteristics or state of an object, representing what the object knows.

### Access Control Protects Data

Access specifiers (private and public) control who can access different parts of your class, ensuring data integrity.

### Classes Model Reality

Classes help organize code, make it more maintainable, and effectively model real-world entities and relationships.

## Next Steps in Your Journey

Congratulations on creating your first C++ class! As you continue learning object-oriented programming, you'll discover more advanced concepts that build on this foundation:

### Inheritance

Creating new classes based on existing ones, establishing parent-child relationships

### Polymorphism

Objects taking multiple forms, allowing different implementations of the same interface

### Operator Overloading

Defining how operators work with your custom classes

### Templates

Creating generic classes that work with different data types

## Keep Practicing!

The more you practice, the more natural object-oriented thinking will become. Start by creating classes to model different aspects of your life and interests—your favorite hobbies, campus activities, or personal projects.

Remember: every expert programmer started exactly where you are now. Classes are fundamental to modern software development, and you've just taken your first steps into a powerful programming paradigm. Keep building, keep experimenting, and most importantly, keep coding!

> "The best way to learn programming is to write programs. Start small, think big, and never stop learning."