



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Functions in C++

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

www.uwtsd.ac.uk

Understanding Functions in C++: Building Blocks of Code

- Overview of the Importance of Functions

function is like creating a mini-program within your main program

Scenario: Imagine you're building a smart home system. You need functions for turning on the lights, adjusting the thermostat, and playing your favorite music. Each function has a clear purpose and helps control different aspects of your smart home.

Writing and Calling Functions

// Function to check if the player
has collected all the keys

```
bool checkKeys(int collectedKeys, int requiredKeys)
{
    return collectedKeys == requiredKeys;
}
```

```
int main() {
    if (checkKeys(collected, required))
    {
        cout << "You can open the treasure chest!";
    }
}
```

Parameters and Return Values

```
#include <iostream>
```

```
// Function to add two numbers and return the result
```

```
int add(int a, int b)  
{  
    return a + b;  
}
```

```
int main()
```

```
{
```

```
    int num1 = 5;
```

```
    int num2 = 3;
```

```
    int result = add(num1, num2);
```

```
    // Calling the 'add' function
```

```
    std::cout << "The sum is: " <<  
    result << std::endl;
```

```
    return 0;
```

```
}
```

Tutorial- (i)

- **Online Shopping Cart**

Imagine you are designing a basic online shopping cart system for an e-commerce website. In this simplified scenario, you need to create two functions to manage a shopping cart:

- 1.addProductToCart:** This function should take the product's name and price as input and add the product to the shopping cart.
- 2.viewCart:** This function should display the list of products in the shopping cart, including their names and prices.

- Your task is to design and implement these two functions to create a functional shopping cart system. Simulate interactions by adding products to the cart and viewing the cart's contents.

Function Overloading

- Function overloading allows you to use the same function name for different tasks, there are different types to name a few: different number of parameters lists, different parameter datatypes, passing parameters as pointers or references.....

- Example:** to add numbers

```
// Function to add two  
integers
```

```
int add(int a, int b)  
{  
    return a + b;  
}
```

```
// Function to add three  
integers
```

```
int add(int a, int b, int c) {  
    return a + b + c;  
}
```

```
int main()
```

```
{  
    // Calls the first function (2 parameters)  
    int result1 = add(5, 3);  
    // Calls the second function (3 parameters)  
    int result2 = add(2, 4, 6);  
}
```

Function Overloading

- **Example:** The functions have the same number of parameters, but they differ in their types

```
// Function to add two integers
int add(int a, int b)
{
    return a + b;
}
```

```
// Function to concatenate two
strings
std::string add(std::string str1,
                std::string str2)
{
    return str1 + str2;
}
```

```
int main()
{
    int intResult = add(5, 3);
    string strResult = add("Hello, ", "World!");
    cout << "Integer Result: " <<
    intResult ;
    cout << "String Result: " <<
    strResult;
    return 0;
}
```

Scope and Lifetime of Variables

- Variables declared in a function have limited scope; they exist only within that function.
- variable is visible only to the region of the code
- Once the function finishes, those variables are gone

The Main Function in C++

- In C++, the **main** function is the entry point of a C++ program. It's where program execution begins
- The 'main' function is mandatory in every C++ program. If it's missing, the program won't compile and run.
- The 'main' function typically returns an integer value. The return value is used to indicate the exit status of the program to the operating system. A return value of '0' - **'return 0'** usually indicates successful execution, while a non-zero value may indicate an error.

Access specifiers

- Access specifiers in C++ are keywords used to determine the level of access control for the members (variables and functions) of a class
- They are essential for enforcing encapsulation and controlling the visibility of class members within the class and in other parts of the code.
- There are three primary access specifiers in C++:

- **Public:**

Members declared as public are accessible from anywhere, both within and outside the class.

For example,

```
class Car {  
    public:  
        string color; // Anyone can see and change the color.  
};
```

- **Private:**

Members declared as private are only accessible from within the class where they are defined. They are not visible or accessible from outside the class.

Let's say you have a "mileage" member for your car class. You don't want just anyone to change it, so you make it private:

```
class Car {  
    private:  
        int mileage; // Only the car itself can change the mileage.  
};
```

- **Protected:**

Members declared as protected have the same access control as private members but with one key difference: they are accessible by derived classes (inheritance).

This specifier is used in the context of inheritance, where derived classes need access to certain members of the base class.

```
class Car {  
    protected:  
        float fuelLevel; // Only the car and its special sports car friends can access the fuel level.  
};  
class SportsCar : public Car {  
    // SportsCar can access the protected members.  
};
```

In summary,

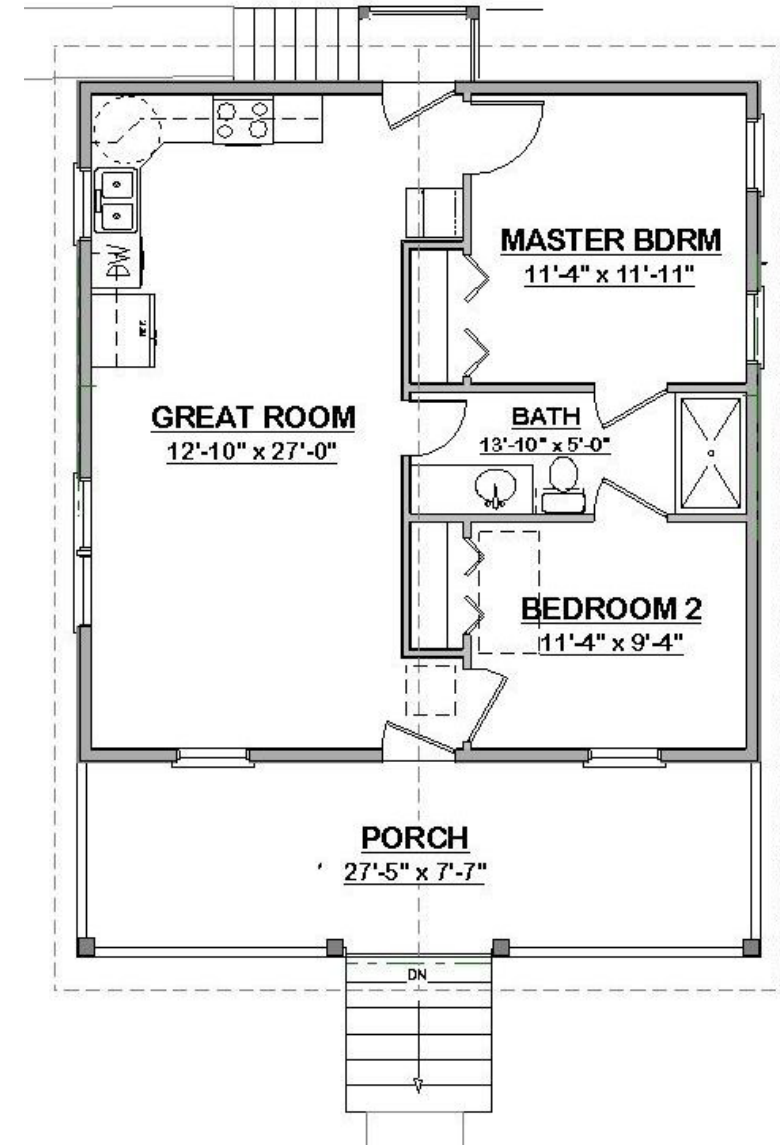
Access specifiers let you control who can see and use the members of your class.

- Public is open to all,
- private is kept secret, and
- protected is like a backstage pass for certain members.

They're essential for managing the visibility and security of your code.

Classes:

- A class is like a blueprint for creating something.
- It defines the general structure and properties of what you want to create.
- However, this blueprint doesn't represent any particular house; it's just a plan.



Classes:

- In programming,
Class defines the structure and properties of an object, including
 - what data it can hold and
 - what actions it can perform.
- It's a template for creating objects of the same type.

```
class Person {  
public:  
    // Data members  
    string name;  
    int age;  
    // Member function to introduce the person  
    void introduce()  
{  
    cout << "Hi, I'm " << name << " and I'm " << age << " years old.";  
    }  
};
```

Objects:

- Objects are like actual houses built from the blueprint (class).
- Each house can have its unique features, decorations, and characteristics, even though they all follow the same basic plan



Objects:

- In programming, Objects are specific instances created from a class.
- Each object can have its own data, attributes, and behaviors.
- While they share the same blueprint (class), they can differ in terms of the data they hold and how they behave.

```
int main() {  
    // Create two Person objects  
    Person person1;  
    Person person2;  
  
    // Set the attributes for person1  
    person1.name = "Alice";  
    person1.age = 28;  
    // Set the attributes for person2  
    person2.name = "Bob";  
    person2.age = 35;  
    // Use member functions to introduce the persons  
    person1.introduce();  
    person2.introduce();  
    return 0;  
}
```

Tutorial (ii)-Classes & objects

Student Enrollment: Imagine you are designing a system to manage student enrollments at a university.

Create a class called **Student** to represent a student's information. Your **Student** class should have the following attributes: **Student ID, Name, Age, Course** (the program or major the student is enrolled in), **Semester** (the current semester the student is in)

Design the **Student** class to allow you to create instances of students and set their attributes directly. Then, simulate the following interactions with the class:

1. Create three instances of the **Student** class to represent three different students with unique information.
2. Display the details of each student, including their Student ID, Name, Age, Course, and Semester.
3. Add a new student to the system.
4. Display the updated information for all students.

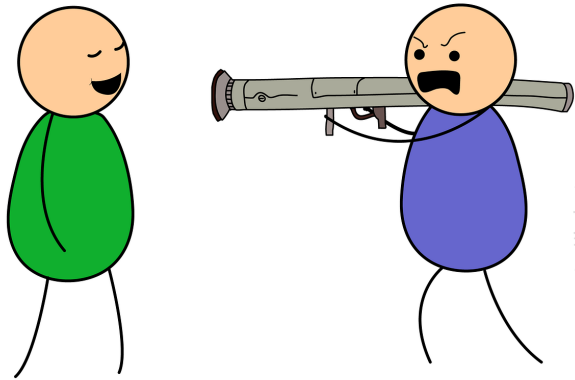
Design your class and interactions to demonstrate the management of student information within the context of the university's enrolment system.

.

Login to Net acad course and complete relevant sections on functions
(15 mins)

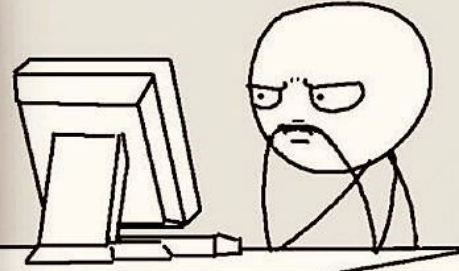
.

When someone says
programming is easy.

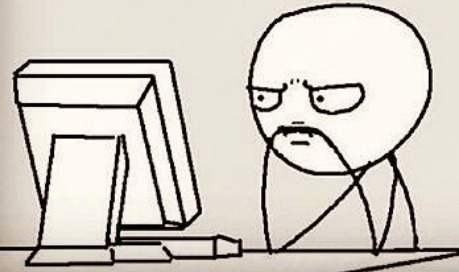


techIndustan

It doesn't work..... why?



It works..... why?



Happiness
is a day
spent
<coding/>

Happy Coding!!!

Next Session.....

Functions(pass by value, pass by reference)
Inheritance