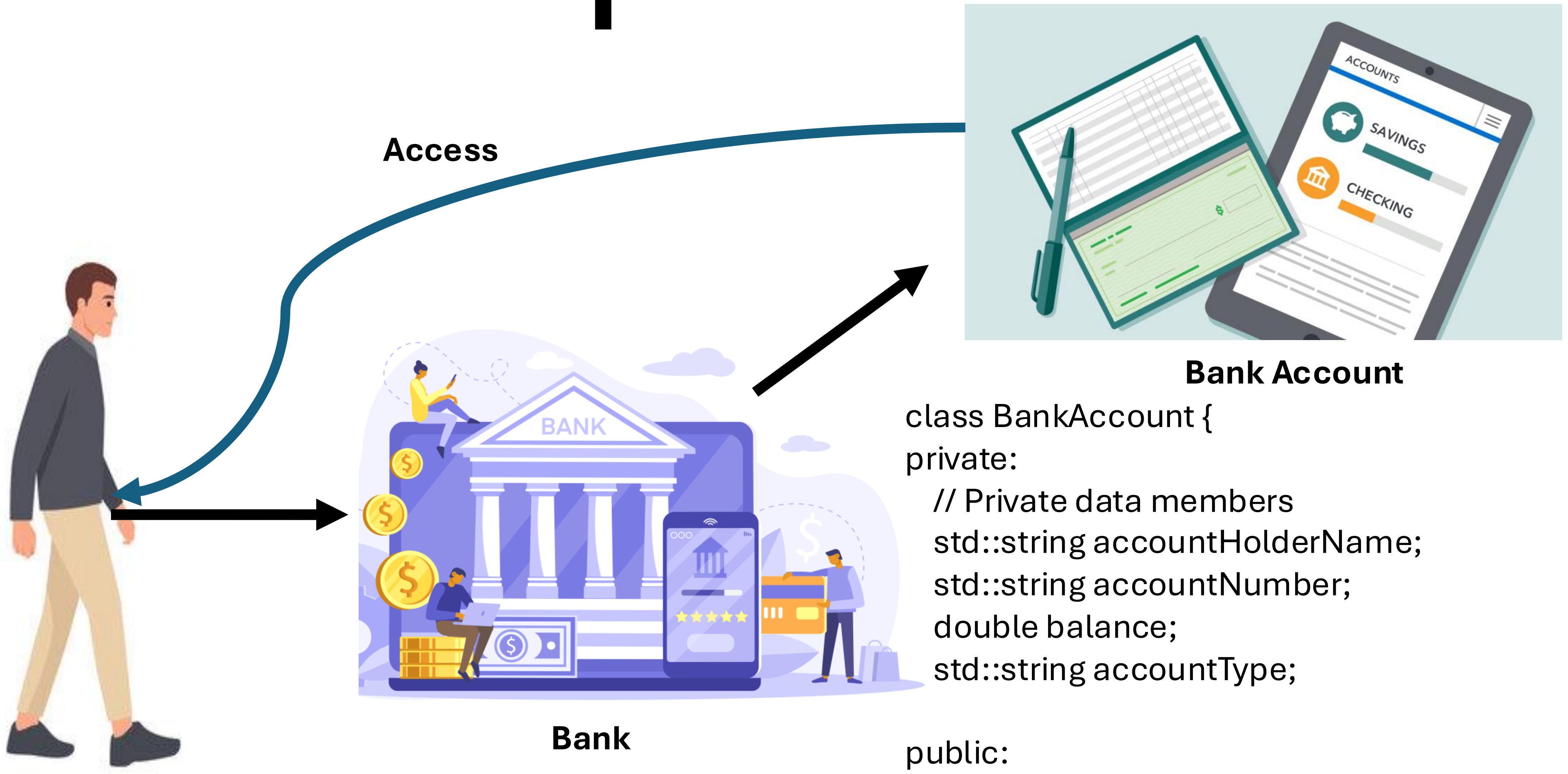
Encapsulation



```
class BankAccount {
private:
   // Private data members
   std::string accountHolderName;
   std::string accountNumber;
   double balance;
   std::string accountType;

public:
   // Constructor
   BankAccount(const std::string& holderName)
   ~BankAccount();
```

ATM

Cash

Groceries

```
class BankAccount {
private:
   // Private data members
   std::string accountHolderName;
   std::string accountNumber;
   double balance;
   std::string accountType;
}
```

```
#include <iostream>
using namespace std;

class BankAccount {
private:
   double balance;

public:
   BankAccount() : balance(0.0) {}

   // Only the ATM can access private balance details
   friend class ATM;

   void deposit(double amount) {
      balance += amount;
   }
};
```

```
class ATM {
public:
   void withdraw(BankAccount& account, double amount) {
      if (account.balance >= amount) {
         account.balance -= amount;
         cout << "Withdrawal successful. New balance: " << account.balance << endl;
      } else {
         cout << "Insufficient funds. Current balance: " << account.balance << endl;
      }
   }

   void checkBalance(BankAccount& account) {
      cout << "Your balance is: " << account.balance << endl;
   }
};
```

Global Function as a Friend

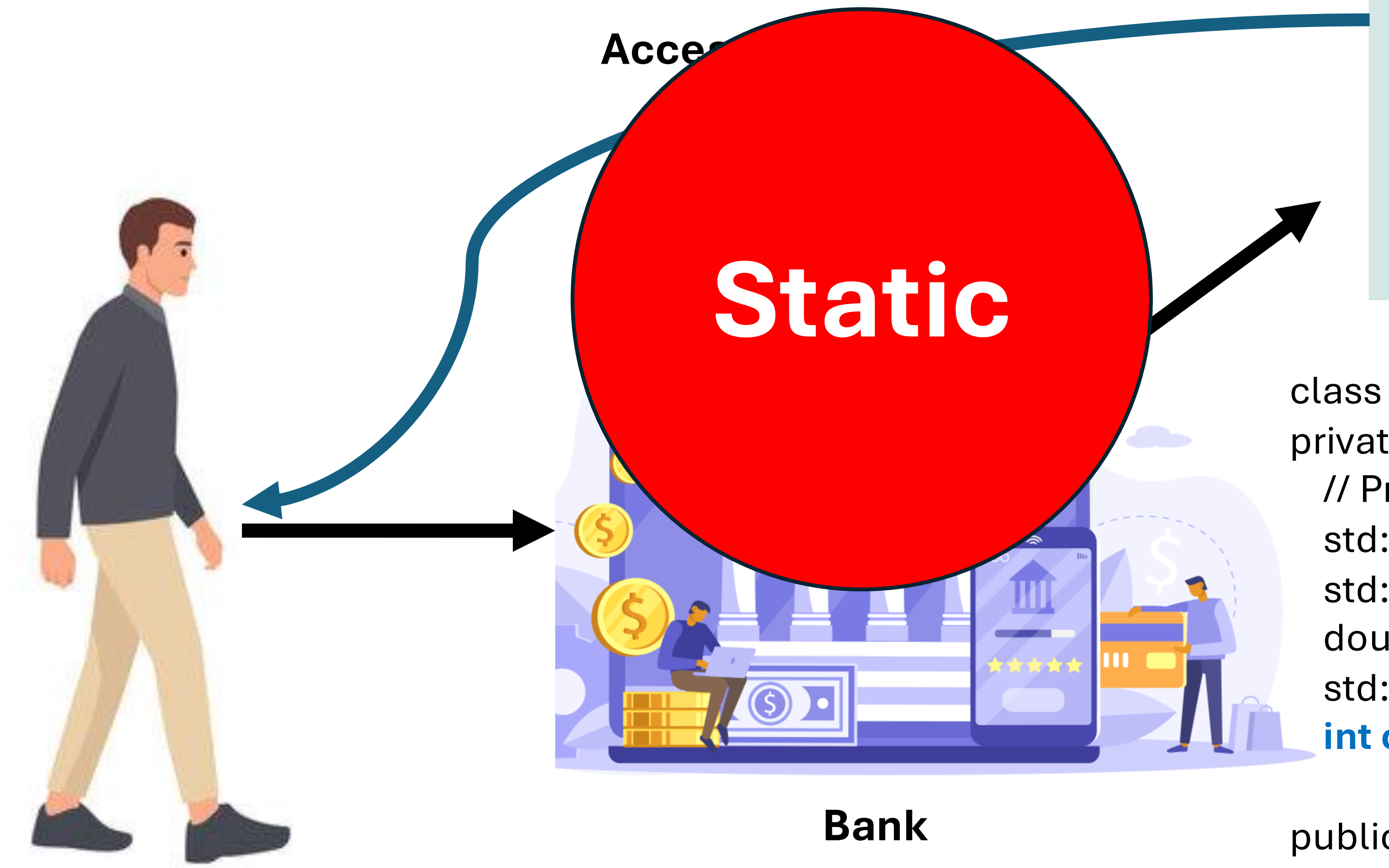
```
#include <iostream>
using namespace std;

class Box {
private:
   double width;

public:
   Box() : width(0.0) {}

   // Friend function declaration
   friend void printWidth(Box box);

   void setWidth(double w) {
      width = w;
   }
};
```

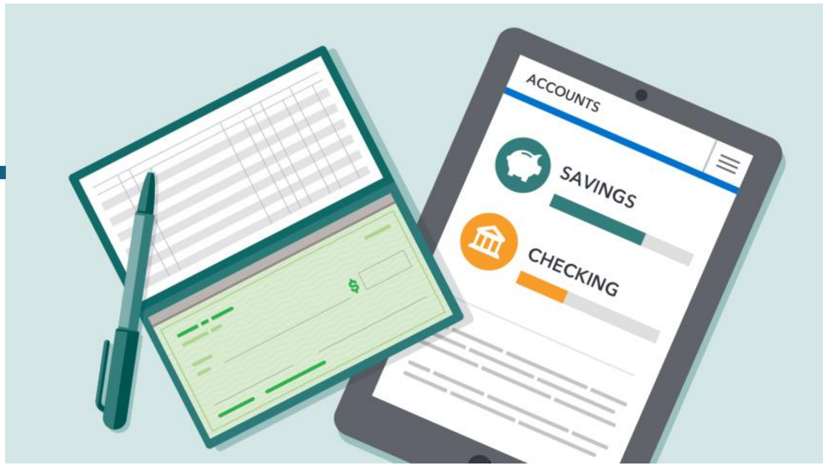
```
// Friend function definition
void printWidth(Box box) {
   cout << "Width of box: " << box.width << endl;
}

int main() {
   Box box;
   box.setWidth(10.5);
   printWidth(box); return 0;
}
```

```cpp
class B;  // Forward declaration of class B

class A {
private:
   int privateData;
   friend class B;  // Class B is declared a friend of class A

public:
   A() : privateData(42) {}
};
```

```cpp
class B {
public:
   void accessA(A& a) {
      // Class B can access the private data of class A
      std::cout << "Accessing A's private data: " << a.privateData << std::endl;
   }
};

int main() {
   A objA;
   B objB;
   objB.accessA(objA);
   return 0;
}
```

Encapsulation





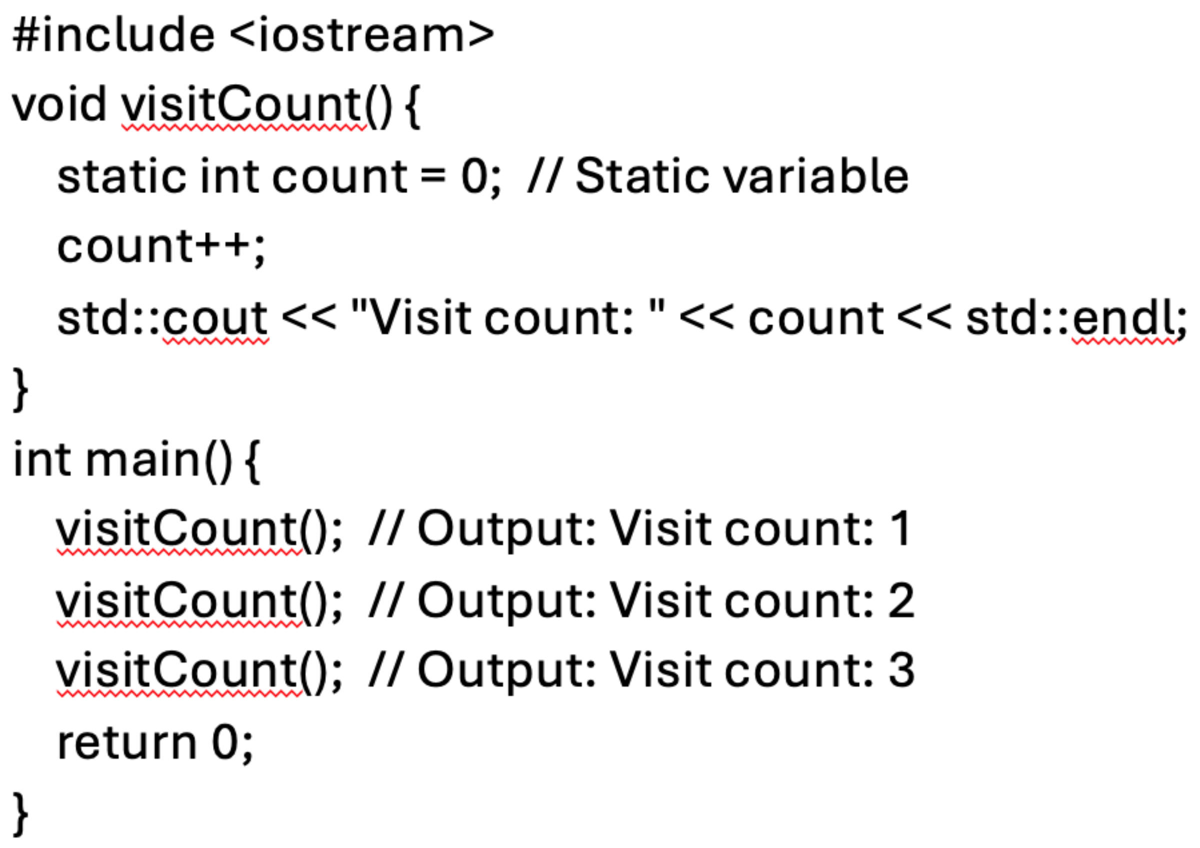
Bank Account

```
class BankAccount {
private:
   // Private data members
   std::string accountHolderName;
   std::string accountNumber;
   double balance;
   std::string accountType;
   int count;

public:
   // Constructor
   BankAccount(const std::string& holderName)
   ~BankAccount();
```

Std_1
Std_2
Std_3

Static Variables in Functions

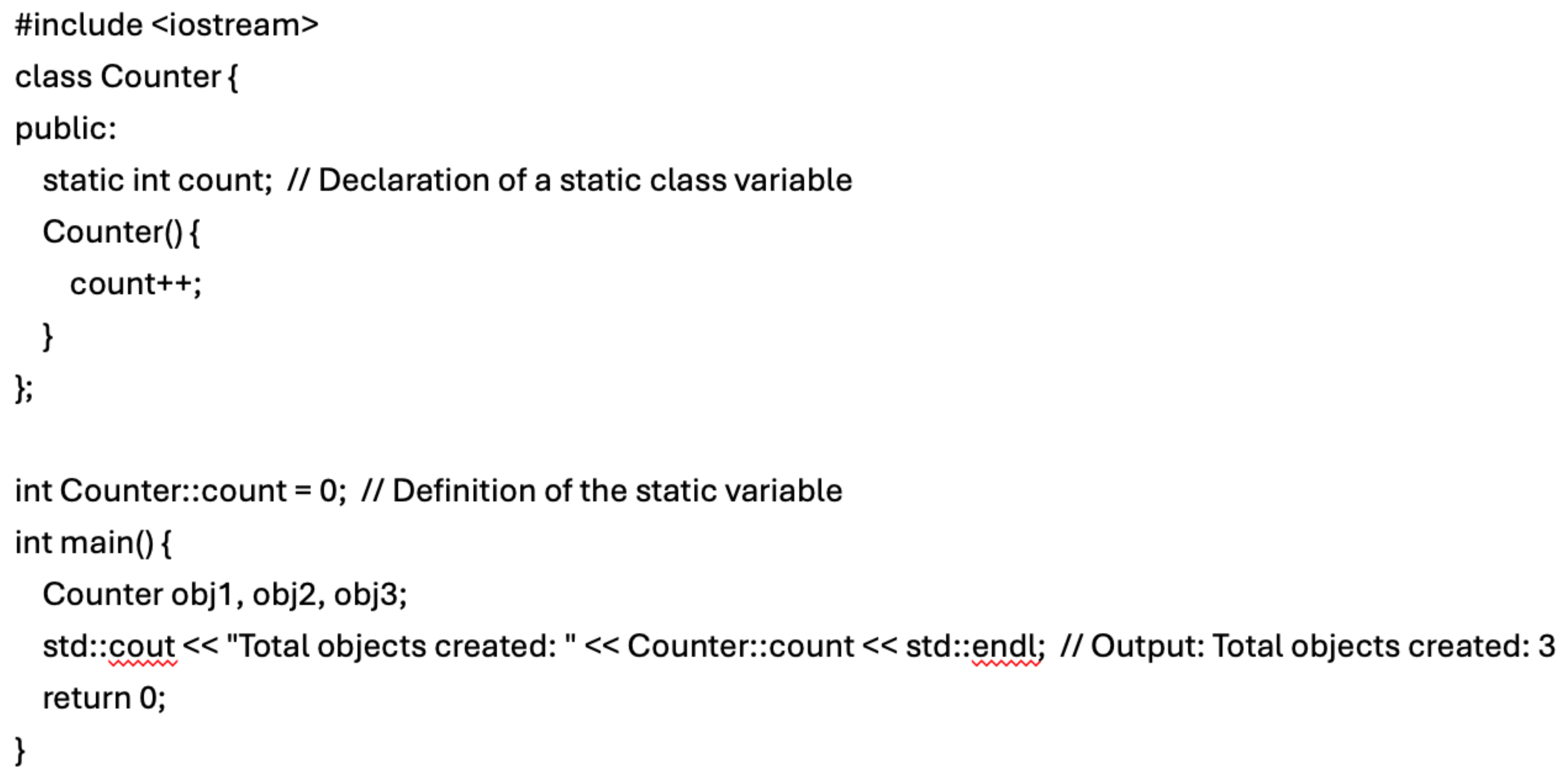
- A static variable is initialized only once and retains its value across function calls.



```
#include <iostream>
void visitCount() {
   static int count = 0;  // Static variable
   count++;
   std::cout << "Visit count: " << count << std::endl;
}
int main() {
   visitCount();  // Output: Visit count: 1
   visitCount();  // Output: Visit count: 2
   visitCount();  // Output: Visit count: 3
   return 0;
}
```

Static Variables in Classes

- Static is used to declare variables that are shared across all instances of a class.



```
#include <iostream>
class Counter {
public:
  static int count;  // Declaration of a static class variable
  Counter() {
    count++;
  }
};

int Counter::count = 0;  // Definition of the static variable
int main() {
  Counter obj1, obj2, obj3;
  std::cout << "Total objects created: " << Counter::count << std::endl;  // Output: Total objects created: 3
  return 0;
}
```

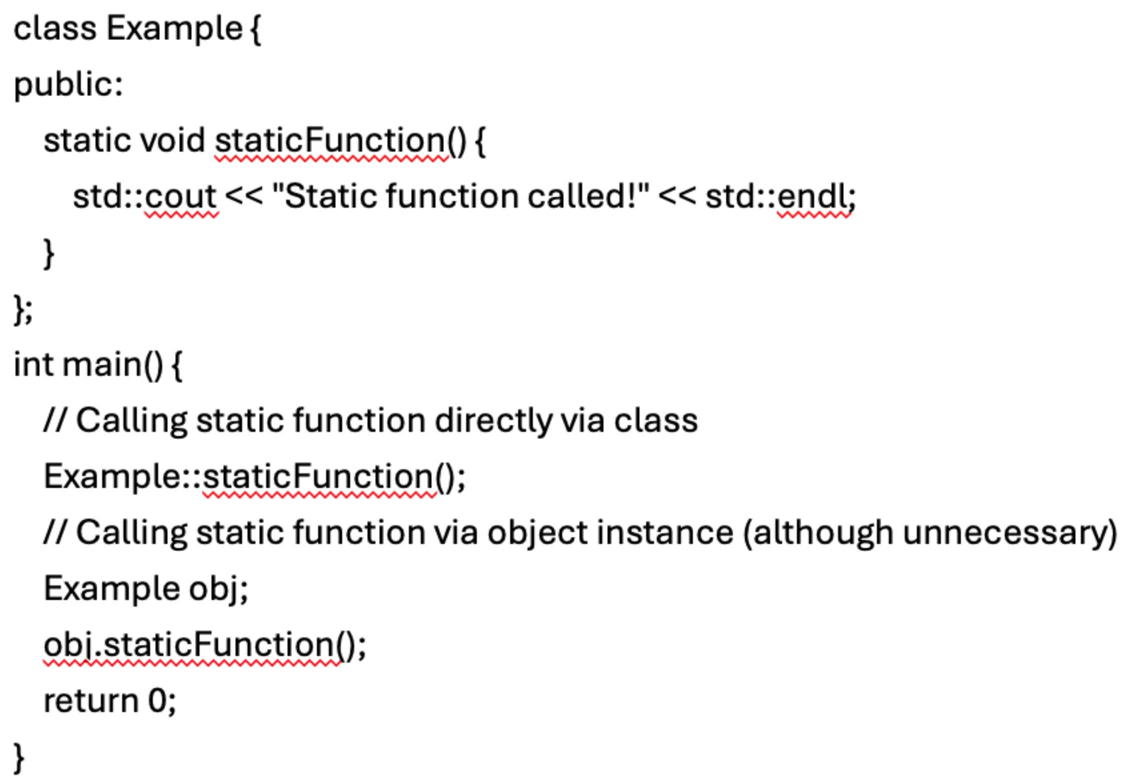
Static Member Functions in Classes

```
#include <iostream>
class Utility {
public:
    static int square(int x) {
        return x * x;
    }
};
int main() {
    int result = Utility::square(5);  // Calling static member function
    std::cout << "Square of 5: " << result << std::endl;  // Output: Square of 5: 25
    return 0;
}
```

Static Function Call

Two ways:

- Directly through the class,
- Through an object



```
class Example {
public:
  static void staticFunction() {
    std::cout << "Static function called!" << std::endl;
  }
};
int main() {
  // Calling static function directly via class
  Example::staticFunction();
  // Calling static function via object instance (although unnecessary)
  Example obj;
  obj.staticFunction();
  return 0;
}
```

Static Objects

```
class Logger {
public:
   // Member function to log messages
   void logMessage(const std::string& message) {
      std::cout << "Log: " << message << std::endl;
   }
   // Static member object
   static Logger globalLogger;  // Declaration of static object
};
// Definition of the static member object
Logger Logger::globalLogger;
int main() {
   // Using the static object to log messages
   Logger::globalLogger.logMessage("First log message.");
   Logger::globalLogger.logMessage("Second log message.");
   return 0;
}
```

```
// Class Colour definition
class Colour {
private:
  int g;  // Assume more data members
public:
  // Constructor for class Colour
  Colour(int a): g(a) {
    std::cout << "Colour object created with value: " << g << std::endl;
  }
  // Other functions (assume more)
};
```

```
// Class Point definition
class Point {
private:
  double xcoordinate;
  double ycoordinate;
  int pointID;
public:
  // Static Colour object declaration
  static Colour pointColour;

  Point(): xcoordinate(0), ycoordinate(0) {
    std::cout << "Point created!" << std::endl;
  }
};
```

```
// Define and initialize static Colour object outside the class
// Pass the required argument to the Colour constructor
Colour Point::pointColour(5);  // Argument passed to Colour constructor

int main() {
  // Creating Point objects
  Point p1, p2;
  return 0;
}
```

```
Colour object created with value: 5
Point created!
Point created!
```