CPP101

Coding Club, IIT Guwahati

Coding Club
IIT Guwahati

```cpp
#include <iostream>
#include <functional>

int main()
{
    int C;
    std :: cin >> C;

    std::function<void(int)> Advanced = [&](int C) {
        C++;
    };

    Advanced(C++);

    return 0;
}
```

Week 2

Day 1

Global Scope in C++

In C++ you can declare global variables which can be used in any function unlike the variables declared inside the main function
To declare a global variable just declare it in the same way as you declare a normal variable but outside the main function

For example this program gives the output :

```
Call from func1: 10
Call from main: 10
```

```cpp
#include <iostream>
int var = 10;
void func1(){
  std:: cout << var << '\n';
}
int main(){
  std:: cout<<"Call from func1: ";
  func1();
  std:: cout<<"Call from main: ";
  std:: cout << var << '\n';
}
```

To use a global variable inside a function that have same name for local and global variable you can use scope resolution operator :: Read more about it here

Also note that you can only do declarations and initialisations in global scope. Trying to call a function in global scope will give error. Read a discussion here

Static keyword

Generally, the value of local variable is deleted once the function ends, but by using static it does not happen. Understand it using this code :

Memory layout of C++ program

Before understanding the difference between static and global variables you need to first understand how C++ organizes it's memory to execute it's code. It basically divides the memory into four parts namely Code segment, data segment, stack and heap. Read more about it here . Read this discussion here also
BSS(Block Started by Symbol) is the section in the data segment where all the uninitialised global/static variables go. You can learn more about the bss section here. Even the global/static variables that you initialize to zero also go in this section. Read about it here
Why do the compiler initialize all the variables in the bss section to 0. You can learn about it here

Difference b/w global and static

The difference is that the static variables are only accessible in the function that they are declared in while the global variable can be accessed anywhere. Both are initialised in the bss segment in the memory. Read more about in this Quora answer here . So the only difference in a global and static variable is the only the scope.
Now you would already know from last section that the variables in bss segment are initialised to 0, so the value of any global/static variables are initialised to 0 while the in any global/static object all the variables are set to 0
Also you can look at the difference between between global and static global here
There is also a class of functions that can only access the static variables of a class and these are known as static functions. There are also class of variables that have only read-only access to the class variables and they are known as constant functions. Learn more about them here

```cpp
#include <iostream>
void func1(){
    static int var = 1;
    std:: cout<<var<<'\n';
    var++;
}
int main(){
  std:: cout<<"Value of var before calling func1: ";
  func1();
  std:: cout<<"Value of var after calling func1: ";
  func1();
}
```

```
Value of var before calling func1: 1
Value of var after calling func1: 2
```

Here when we called the func1() again the compiler doesn't initialise var again but uses the earlier var because it wasn't deleted from the memory.

You can read more about it here

Volatile in C++

If you want to prevent C++ from using unwanted optimisations while using your variables you can use 'volatile' keyword.
It tells the compiler that the value of this variable can change anytime unexpectedly so as to be sort of be prepared for it
You can read more about it here and here

lvalue and rvalue

lvalues are objects having an identifiable location in the memory whereas rvalues are the data values stored at some memory address. You can learn more about them here.

Day 2

Enum in C++

Enumeration is a user defined data type that consists of integral constants
By default all the variables inside enum are given values from 0 to n-1 where n is the size of the enum set but you can change this by initialising them explicitly
For example consider the output of the following example

```cpp
#include <iostream>

enum languages{
  Java,
  Python,
  CPP,
  Assembly = 12,
};
int main(){
  std:: cout<< Java << '\n'; //Outputs 0
  std:: cout<< Python << '\n'; //Outputs 1
  std:: cout<< CPP << '\n'; //Outputs 2
  std:: cout<< Assembly << '\n'; //Outputs 12
}
```

You can learn more about enumeration here .
You can also have a look at this GFG article on enum here

Clarification for last example of gfg article :
In the last example they first declared an int i and in the for loop initialised it with the value of the variable ‘Jan’ which was zero then they looped till its value is equal to ‘Dec’ which was 11

Extern in C++

Suppose that you are making a project where you need to import variable and functions from another c++ file. How will you tell your C++ compiler that it needs to get this variable from another file and not initialise it here ? It is where extern comes to help extern tells the compiler that this particular variable of that type exists somewhere in some file that are being imported and it don't has to worry to initialise it here

Consider the example given below :

A.cpp

```cpp
#include <iostream>
#include "B.cpp"

int main(){
    extern int a;
    std:: cout<<a<<'\n';
}
```

B.cpp

```cpp
int a = 10;
```

Outputs : 10

A.cpp

```cpp
#include <iostream>
#include "B.cpp"

int main(){
    int a;
    std:: cout<<a<<'\n';
}
```

B.cpp

```cpp
int a = 10;
```

Outputs : -243433471 (Some garbage value)

Read more about extern here. Also read about it's usefulness here

Day 3

OOPS

We already know that OOPS was one of the new feature added to C++. Also, we have learnt about classes and objects in the previous week. To recall, classes are user-defined data structure, a blueprint, which groups together data and functions together, whereas objects are the instances of the class. Object oriented programming allows classes and objects to interact with each other in a better way, binding the data and functions that operate on them so that no other part of the code can access this data except that function.

Constructor and Destructor

Constructor : A constructor is a special method which is called whenever any object is created. It could be used to intitialise the object with values either given by the user or any default value. You can watch this video for better understanding.
The types of constructor are:

- Default Constructors : Covered in the video above.
- Parameterized Constructors : Covered in the video above.
- Copy Constructors : Allows initialization of objects using another object of same class. You can read this article or watch this video.
- Move Constructor : Read here or watch this video

Destructor : Contrary to constructor, destructor is a method called when an object is destroyed. It can be used to free up the memory. You can read more about destructor here or watch this video.

This article also summarises Constructor and Destructor in C++.

OOPS Concept:

OOPS allows the code to be more readable and reusable using the core concepts of Encapsulation, Abstraction, Inheritance and Polymorphism. Further, I will use an example of a city to briefly explain each of these concepts.

- Encapsulation : The wrapping of code and data together into a single unit is known as encapsulation. In our example, police department of a city can be seen as an encapsulation. They have all the crime records and in order to access them we first need to contact someone from their department and then only we can get the data.

- Abstraction : Hiding internal details and showing functionality is known as abstraction. In our example, the sewage system of a city uses abstraction. The wastewater is collected in a network of pipes and sent for treatment without the citizens having much knowledge.

- Polymorphism : The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. In our example, parks in a city can have multiple usage. Citizens can exercise and meditate in a park, children an play in a park.

- Inheritance : The capability of a class to derive properties and characteristics from another class is called Inheritance. In our example, There are many hospital in a city which specialise in different things, there may be a hospital for treatment of fracture or for treatment treatment of heart related issues. However, the basic structure remains the same for every hospital which is inherited.

More content on Encapsulation, Abstraction, Polymorphism and Inheritance in the following days of this week.

Day 4

Abstraction

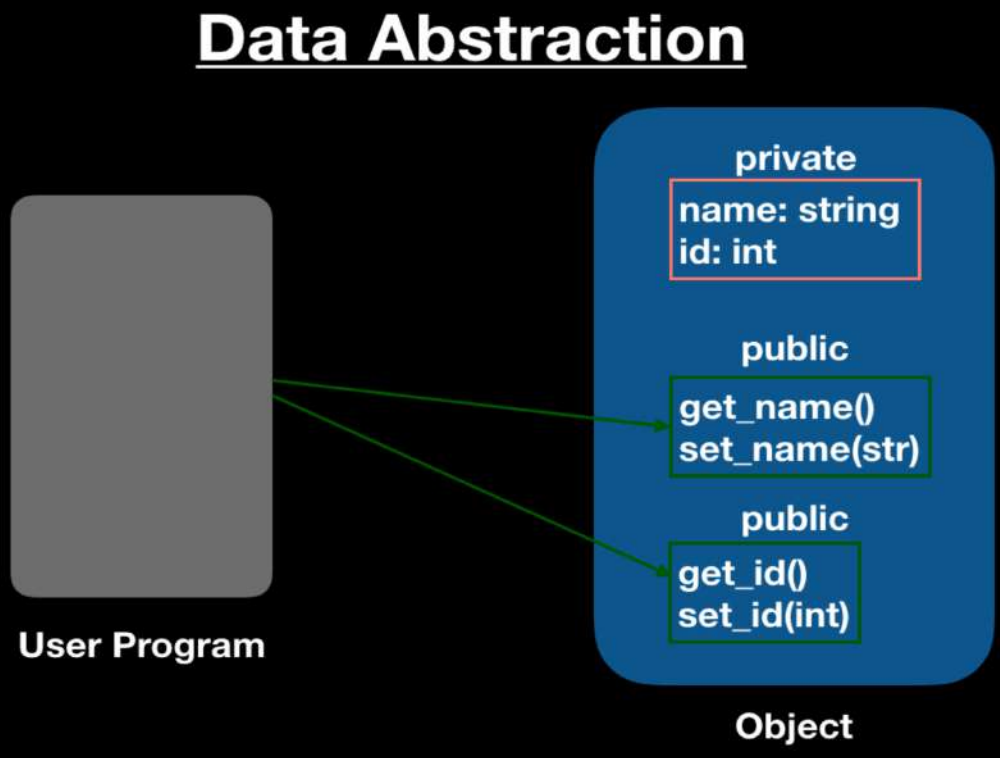
In OOPS, Abstraction exposes necessary functionalities to external objects and hides implementation details. This helps programmers use complex logic inside an object without understanding it's implementation. In other words, it is the process of hiding internal details of an application from the outer world so that it is only needed to know what the code does, and not thinking how it is being done!

Types of Abstraction

OOPS can be seen as an attempt to abstract both data and control. Hence, there are two types of abstraction, data abstraction and control abstraction.

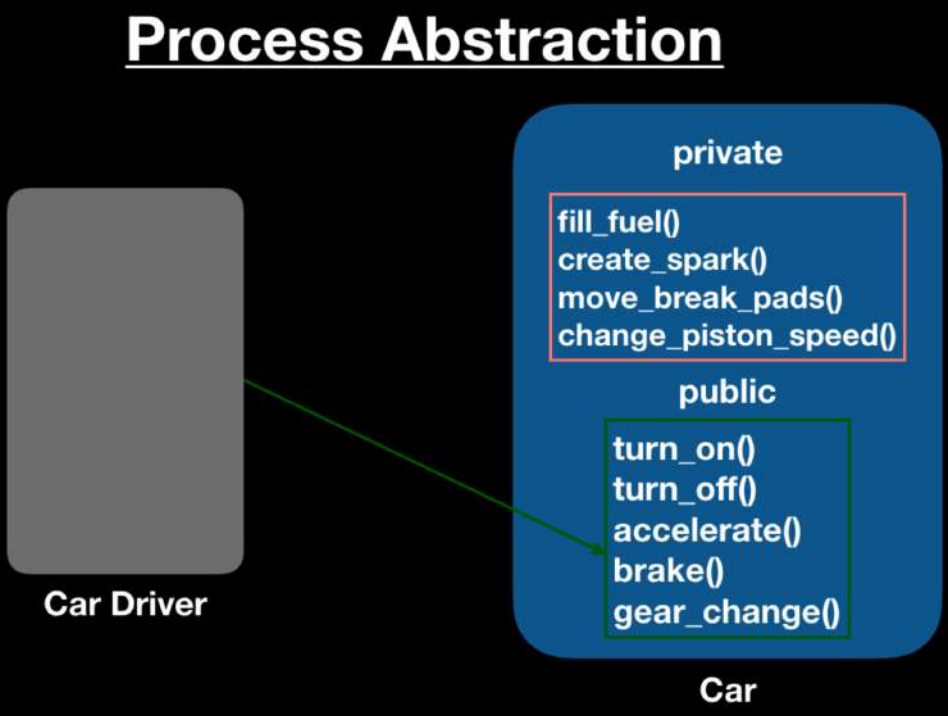
Data Abstraction

Data abstraction is about the clear separation between the properties of a data type and the concrete details of it's implementation. In other words, when an object's data is not available to the outer world, it creates data abstraction. In other words, data abstraction provides an interface to the data type that can be visible to the client to use it. At the same, inside the class, we can keep the operations working on that data entirely private, which we can change later for any updates or improvements. This way, the client code won't be impacted.



Control Abstraction

When we hide the internal implementation of the different methods related to the client operation, it creates control abstraction. We can also this in the low level. A software code is a collection of methods written in a programming language. Most of the time, some methods are similar and repeated multiple times. So, the idea of control abstraction is to identify all such methods and bundle them into a single unit.



You can read more about abstraction from this article or this video.

Now look at an example code to make your concepts clear

```cpp
#include <iostream>
using namespace std;
// declaring class
class square
{
    private :
    float side;
    public :
    // function to get value of side from the user
    void get_side(void)
    {
        cout << "Enter side of square: " << '\n';
        cin >> side;
    }
    float area(void) {return(side*side);}
    float perimeter(void){return(4*side);}
};
//implementation of the class
int main ()
{
// creating object of declared class
    square s;
    s.get_side();
    cout << "Area = " << s.area() << '\n';
    cout << "Perimeter = " << s.perimeter() << '\n';
}
```

Day 5

Inheritance

The capability of a class to derive properties and characteristics from another class is known as inheritance. It is one of the most important features of OOPS. It is a feature or a process in which, new classes are created from the existing classes. The new class created is called derived class or child class and the existing class is known as the base class or parent class. The derived class now is said to be inherited from the base class.
Read this for a thorough explanation with examples-
Inheritance - GFG

There are 3 modes of inheritance -

- Public mode - Here, public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class
- Protected mode - Here, both public members and protected members of the base class will become protected in the derived class.
- Private mode - Here, both public members and protected members of the base class will become Private in the derived class.

Consider this example :

- We can access pub from the derived class PublicDerived only but not from PrivateDerived or ProtectedDerived because it is private and protected in that class respectively
- Also pvt is not accessible even to the derived classes
- prot is accessible to derived classes but not outside the classes

```
class Base{
    private:
        int pvt=10;
    protected:
        int prot=20;
    public:
        int pub=30;
};

class PublicDerived : public Base{
};

class PrivateDerived : private Base{
};

class ProtectedDerived : protected Base{
};
```

```
int main(){
    PublicDerived obj1;
    PrivateDerived obj2;
    ProtectedDerived obj3;
    obj1.pub;
    // valid
    obj1.prot;
    // gives error because prot was protected member in base class
    // and in derived class is also protected
    obj2.pub;
    // gives error because although pub was public member in base
    // class but in derived class it is private
    obj3.pub;
    // gives error because although pub was public
    // member in base class but derived class is protected
}
```

To read more about modes of inheritance with examples-
Modes of inheritance - CPP

There are 5 types of inheritance -

- Single - A class is allowed to inherit from only one class
- Multiple - A class can inherit from more than one class
- Multilevel - A derived class is created from another derived class
- Hierarchical - More than one subclass is inherited from a single base class
- Hybrid/Virtual - By combining more than one type of inheritance

To read more about types of inheritance with examples-
Types of inheritance - CPP

More resources for learning inheritance:

- Inheritance - CPP - Youtube
- Inheritance - Youtube (Hindi)

Virtual Base Class

Suppose, two classes B and C inherit class A, now suppose we make another class D which inherits B nd C, so indirectly inheriting A two times, which is undesirable. To prevent this duplicate inheritance, class A is declared as a virtual base class. To learn more about virtual base classes, see this, this, this and this.

Day 6

Polymorphism

Polymorphism is one of the main features of OOPS. It refers to the ability of a C++ function or object to perform in different ways, and allows us to reuse code by creating one function that's usable for multiple uses. A real-life example of polymorphism is a person who at the same time can have different characteristics. A man at the same time is a father, a husband, and an employee.
Polymorphism in C++ is categorised into two types.

Compile Time Polymorphism

- Function Overloading
  A single function is used to perform many tasks with the same name and different types or number of arguments. Look at the following example :

```
class Geeks {

public:
    void func(int x)
    {
        cout << "value of x is " << x <<'/n';
    }
    void func(double x)
    {
        cout << "value of x is " <<x << '/n';
    }
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y <<'/n';
    }
};

int main()
{
    Geeks obj1;
    obj1.func(7); // Calls the first definition of func
    obj1.func(9.132); // Calls the second definiton of func
    obj1.func(85, 64); // Calls the third defintion of func
    return 0;
}
```

```
Output :
value of x is 7
value of x is 9.132
value of x and y is 85, 64
```

Whenever we call a function with the given arguments the C++ compiler search for all the functions with given name and match which satisfies the argument type and number if none of them matches then it tries to do standard data type conversion to match the argument types like char, short are promoted to int and float is promoted to double if then also no there is no match then it gives an error.

For example obj1.func(“hello world”) would have given an error in the above program. Since this all happens during the compiling time it comes under compile time polymorphism

Learn more about function overloading here

Not all functions can be overloaded learn more about it here . Also changing the return type doesn’t mean function overloading see it here
Also we need to make it clear that it not necessary that you define these functions in a class. Function overloading can be done to normal functions as well.

Commonly used overloaded function include the sqrt() function that can take int, double, float, long long. Same applies to min() and max() function in STL of C++

- Operator Overloading
  We can give a special meaning to an operator for a particular class without changing its original meaning for the rest of the program. For example consider the ‘+’ operator in C++ when used between two int adds them and when used between two strings concatenates them. Similarly the ‘*’ operator is used to multiply two numbers as well as used to dereference pointers or iterators

Learn more about operator overloading here

Similarly like functions there are operators that can’t be overloaded. Learn more about it here

You can also overload the << operator in C++ to output the objects of your own classes. Learn it here

Now let’s discuss an example to understand it more clearly

```cpp
#include <iostream>
using namespace std;
class Complex {
public:
    double real, imag;
    Complex(double r = 0, double i = 0)
    {
        real = r;
        imag = i;
    }
    // Overloading the + operator to add two complex numbers
    // by making the + operator a member function of the class
    Complex operator+(Complex const& obj)
    {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }

};
//overloading the << operator to print out the complex number
ostream &operator<<(ostream &out, const Complex &c)
{
    out << c.real;
    out << "+i(" << c.imag <<")"<< endl;
    return out;
}

int main()
{
    Complex c1(5, 5), c2(2, 4);
    cout << c1+c2;
}
```

Here we have made a class of complex numbers and overloaded the ‘+’ operator to make the + operator add two complex number. Without overloading the ‘+’ operator we would not be able to add the two complex number and program would have given and error

Similarly we also overloaded the << operator to output the complex number as we want.

As a programming exercise you can overload the operators - , ~ , * , / operators to do the operation on subtraction, conjugation, multiplication and division of complex numbers respectively. If you feel stuck you can refer to the solution here

Day 7

Runtime Polymorphism

- Function Overriding - In function overriding, we have multiple definitions of the same function, and the program has to discover which definition of a function it needs to use based on the information in the main() function. We use this to selectively execute a function.
  This all happens during the runtime so it comes under runtime polymorphism. If we want to override a function we have to declare the function as virtual function in the base class which we will discuss in the next section. Now consider the following example where we want to get the output of the overridden function. Consider the following example

```cpp
#include <iostream>
class Base{
    public:
        void print(){
            std::cout << "Base" << std::endl;
        }
};
class Derived : public Base{
    public:
        void print(){
            std::cout << "Derived" << std::endl;
        }
};
int main(){
    Base *b = new Derived();
    b->print();
    // Output: Base
    // Reason: Base class pointer is pointing to Derived class object
    //         and calling print() function of Base class
    // in order to overcome this problem we can use virtual function
    return 0;
}
```

Learn more about function overriding here and here

- Virtual Functions

A virtual function is declared by keyword virtual. The return type of virtual function may be int, float, void etc.. A virtual function is a member function in the base class and it is called during runtime. Declaring a virtual function means that this function can be overridden by some function in the derived class and in that case the function of the derived class is called and if there is no definition in the derived class of the function then the function of the base class is called. Virtual functions provide a way for the compiler to decide which function to call at the run-time based on the object (using a V-table), which facilitates Runtime-Polymorphism.

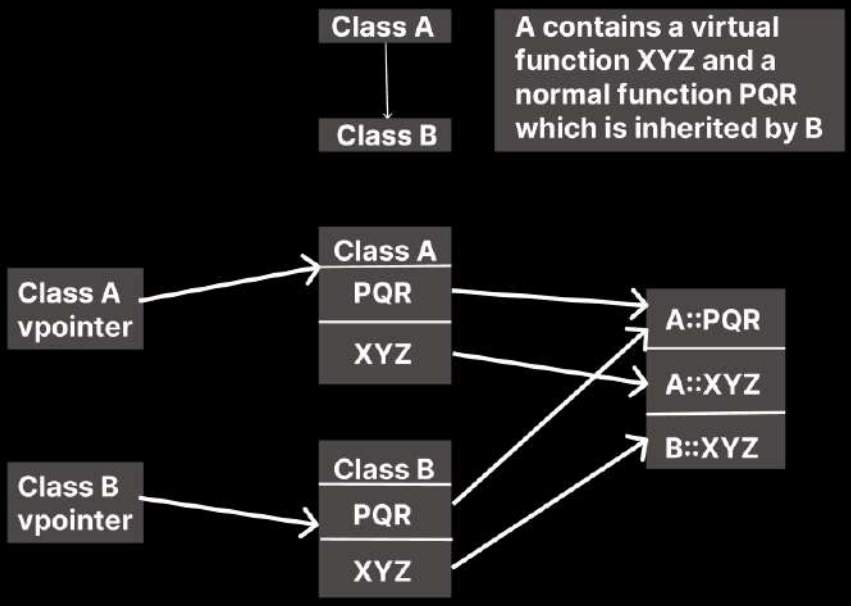
```cpp
#include <iostream>
class Base{
    public:
        virtual void print(){
            std::cout << "Base" << std::endl;
        }
};
class Derived : public Base{
    public:
        void print() override{
            std::cout << "Derived" << std::endl;
        }
};
int main(){
    Base *b = new Derived();
    b->print();
    // Output: Derived
    // Reason : Because the print() function is virtual in Base class,
    //          so the print() function of Derived class overrides it.
    return 0;
}
```

Learn more about virtual functions here, here , here and learn about their implementation through this

Learn how virtual functions work internally here

Virtual Tables in C++

It is an implementation in C++ to facilitate dynamic dispatch. Dynamic dispatch is a process, of selecting which polymorphic function to call at runtime by the compiler. There is a term known as static dispatch, which occurs at compile time, and comes under compile time polymorphism. (Note: Dynamic dispatch is not the same as late binding! see this) For every class that contains virtual functions, the compiler constructs a virtual table, a.k.a vtable. The vtable contains an entry for each virtual function accessible by the class and stores a pointer to its definition. Only the most specific function definition callable by the class is stored in the vtable. Entries in the vtable can point to either functions declared in the class itself, or virtual functions inherited from a base class. Every time the compiler creates a vtable for a class, it adds an extra argument to it: a pointer to the corresponding virtual table, called the vpointer. (Note: It is always a good idea to make destructors of base classes virtual) Read this to learn more about vtables.



VTable of class A, B

Pure Virtual Functions

These functions are virtual, but the condition being that they don't have any definition in the base class. They are are used in cases when one needs to make a template class with some function definitions, which can be extended to make functions with different definitions in different classes. You can learn more about pure virtual functions from this and this.

Encapsulation

PIn normal terms Encapsulation is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them.

Lets understand it by an example:

In a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly the sales section handles all the sales related activities and keep records of all the sales. Now there may arise a situation when for some reason an official from finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of sales section and the employees that can manipulate them are wrapped under a single name “sales section”.
An example with code is here.

Data Hiding

Data hiding is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding.

Note:
Encapsulation is not same as data hiding. It refers to the bundling of related fields and methods together. It can be used to achieve data hiding. It in itself is not data hiding.
For further explanation refer here.

Access Modifiers

We use access modifiers to achieve data hiding using Encapsulation.
To learn more about access modifiers refer here.
For more examples refer here.

Friend Functions and Classes

While creating classes and functions it may so occur that you want to give some outside functions or classes access to the private and protected data of the class to be used inside it. You can do so by using 'friend' keyword.
Consider the following example

```cpp
#include <iostream>

class Base {
    private :
        int x;
    public :
        Base(int x) : x(x) {}
        void print() { std::cout << x << std::endl; }
        friend class Derived;
};

class Derived{
    private :
        int y;
    public :
        Derived(int y) : y(y) {}
        void print(Base& b) { std::cout << b.x<<" "<<y<< std::endl; }
};

int main() {
    Base b(10);
    Derived d(20);
    b.print();
    // Output: 10
    d.print(b);
    // Output: 10 20
    return 0;
}
```

Learn more about friend classes and functions here, here and here