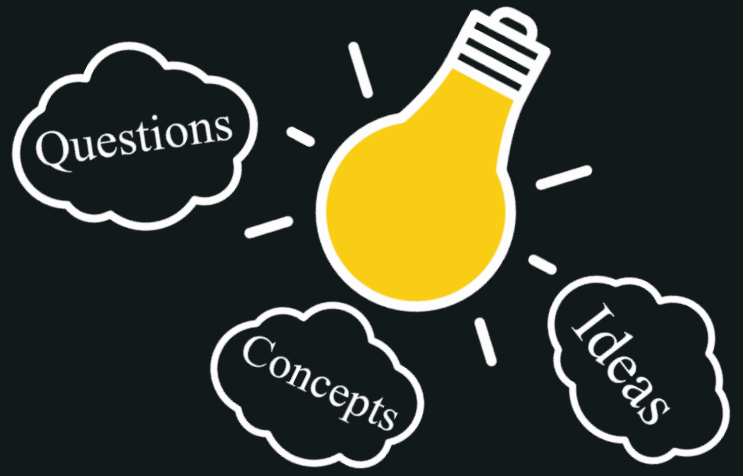


Polymorphism

C++ Programming

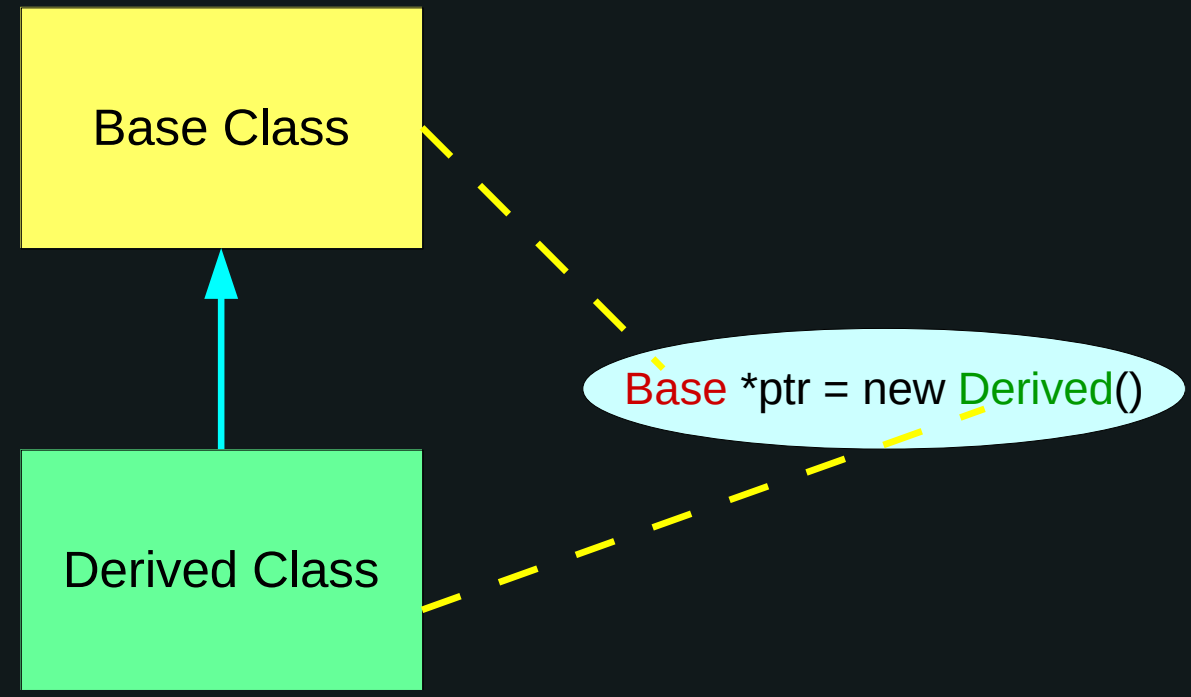
VEDINESH

Base Class Pointer And Derived Class Object?

VEDINESH

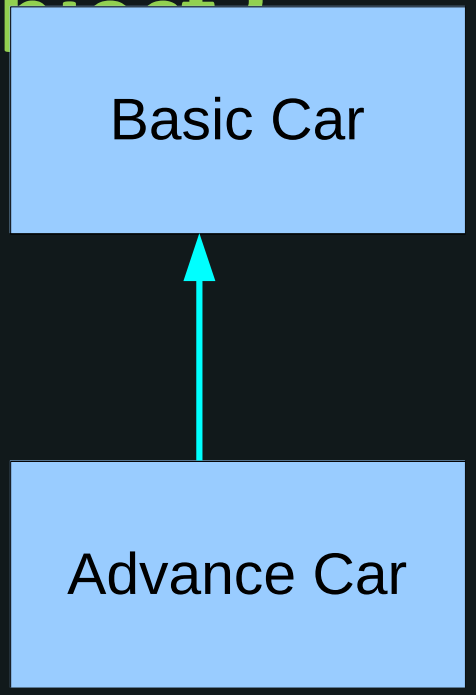
Base *ptr = NULL;

ptr = new Derived();



Base Class Pointer And Derived Class Object?

VEDINESH




```
class basicCar
{
  public:
            void  body( );
            void  door( );
            void  windows( );
            void  tyres( );
}
```

```
class advCar: public basicCar
{
  public:
            void  ABS( );
            void  PS( );
            void  EngineV8( );
            void  AT( );
}
```

```
void main( )
{
   basicCar *ptr;
   ptr = new advCar( );

    ptr->body( );
    ptr->doors( );
    ptr->windows( );
    ptr->types( );

    ptr->ABS( );
    ptr->PS( );
    ptr->EngineV8( );
    ptr->AT( );
```

If a pointer is of BASE CLASS and it is pointing derived class object then, it can't access derived class functionality

Virtual Function

VEDINESH

Virtual means existing in appearance but not in reality

Virtual Function means fun. existing in class but can't be used.

Program that appears to be calling a function of one class may in reality be calling a function of different class .

Virtual Function

VEDINESH

```
class Derv1: public base
{
  public:
            void  show( )
            {
               cout << "Derived1";
            }
}
```

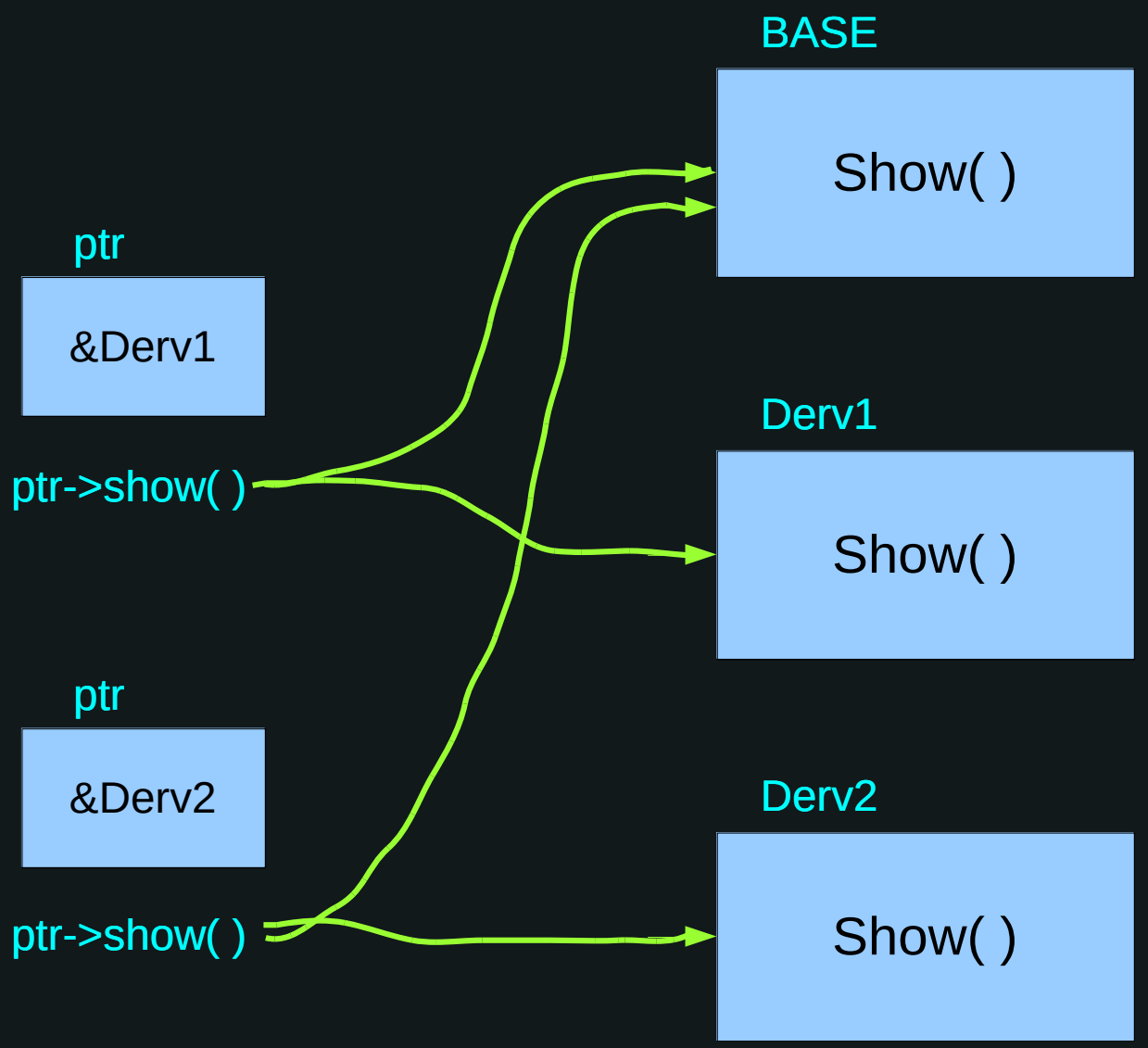
```
class Derv2: public base
{
  public:
            void  show( )
            {
               cout << "Derived2";
            }
}
```

```
class base
{
  public:
            void  show( )
            {
               cout << "Base";
            }
}
```

NOTE:- Pointer to object of derived class ae type - compatible with pointer to object of base class.

```
void main( )
{
   Derv1  dv1;
   Derv2  dv2;

    base *ptr;
    ptr = &dv1;
    ptr->show( );

    ptr = &dv2;
    ptr->show( );
 }
```

Reason


BASE
Show( )
ptr
&Derv1
ptr->show( )
Derv1
Show( )
ptr
&Derv2
Derv2
ptr->show( )
Show( )

Virtual Function

VEDINESH

```
class Derv1: public base
{
  public:
            void  show( )
            {
               cout << "Derived1";
            }
}
```

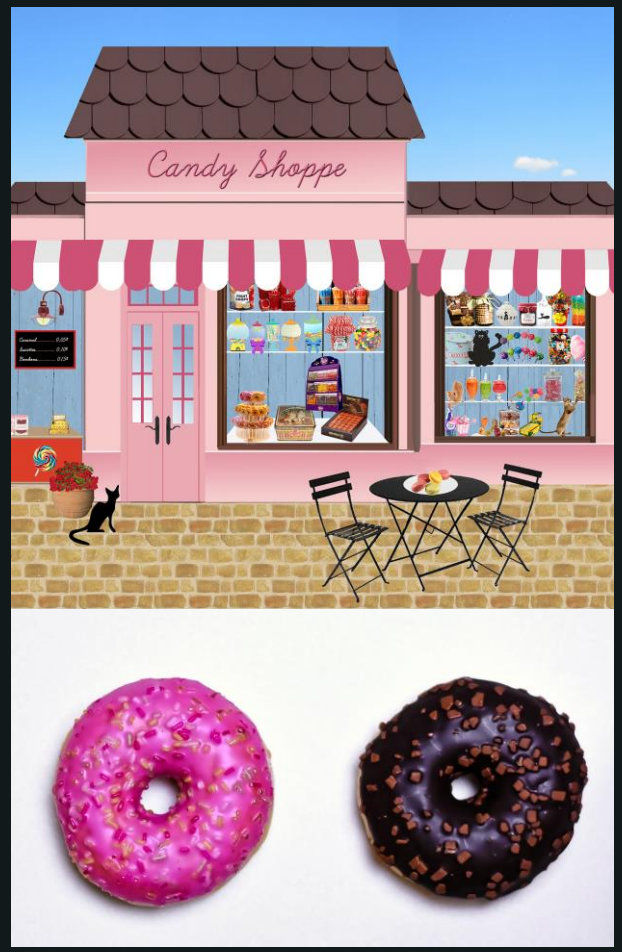
```
class Derv2: public base
{
  public:
            void  show( )
            {
               cout << "Derived2";
            }
}
```

```
class base
{
  public:
            virtual void  show( )
            {
               cout << "Base";
            }
}
```

NOTE:- Virtual makes (show) function invisible.

```
void main( )
{
   Derv1  dv1;
   Derv2  dv2;

   base *ptr;
   ptr = &dv1;
   ptr->show( );

   ptr = &dv2;
   ptr->show( );
 }
```

Why Virtual Function ?



Late Binding:-

Compiler deferr the decision untill, the program is running.
And at runtime when it come to know which class is pointed by PTR, then appropriate function would be called.
This is called Dynamic Binding / Late Binding

Virtual Function

VEDINESH

```
class boy: public person
{
  public:
            void  give( )
            {
               cout << “Brown Bun”;
            }
}
```

```
class girl: public person
{
  public:
            void  give( )
            {
               cout << “Pink Bun”;
            }
}
```

```
class person
{
  public:
            virtual void  give( )
            {
               cout << “Bun”;
            }
}
```

```
void main ( )
{
    boy b1;
    girl g1;

    person *ptr = NULL;

    ptr = &b1;
    ptr->give();

     ptr = &g1;
     ptr->give();
}
```

Abstract Class

VEDINESH

```
class boy: public person
{
  public:
          void  give( )
          {
             cout << "Brown Bun";
          }
}
```

```
class girl: public person
{
  public:
          void  give( )
          {
             cout << "Pink Bun";
          }
}
```

Define:- Abstract class is used when we never want to instantiate object of BASE class.

```
class person
{
  public:
     virtual void  give( )  =  0;
}
```

Pure Virtual Function

Define:- Abstract class exists only to act as parent of DERIVED CLASS.

```
void main ( )
{
    boy b1;
    girl g1;

    person *ptr = NULL;

    ptr = &b1;
    ptr->give();

     ptr = &g1;
     ptr->give();
}
```

Conclusion

give( )

give( )

ptr->give( )

ptr = &girl1

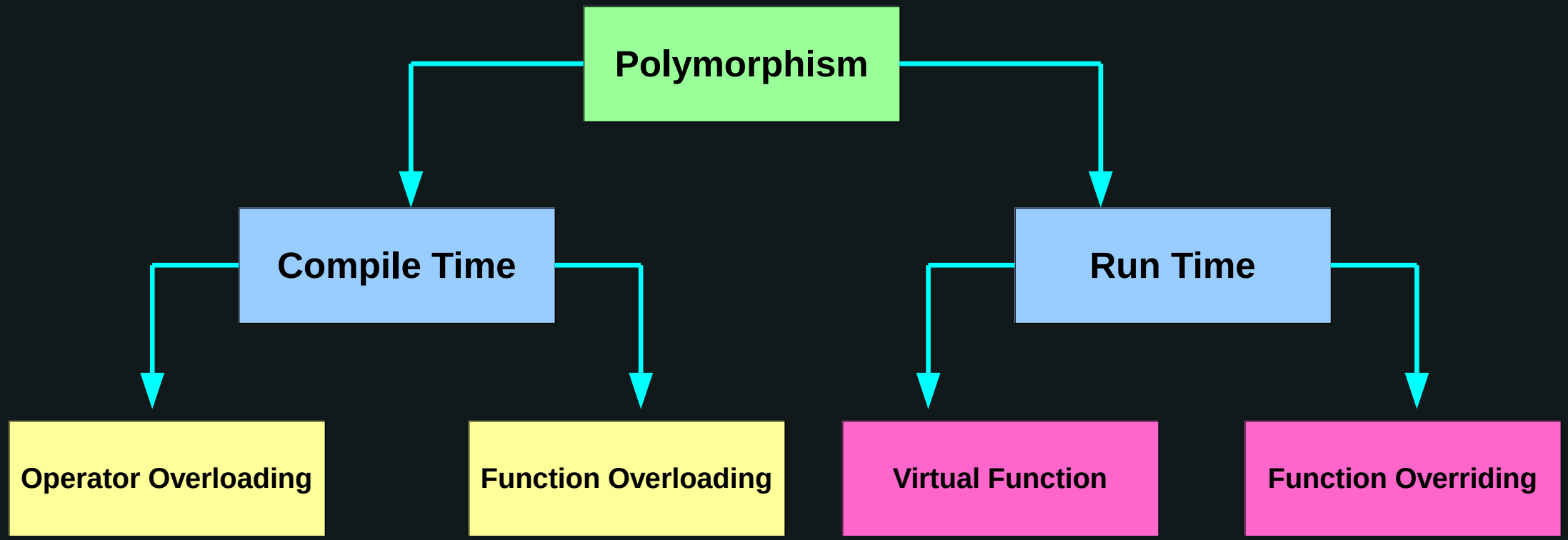
ptr = &boy1

Polymorphism

Poly means many, something existing in more than one form.

Polymorphism

VEDINESH



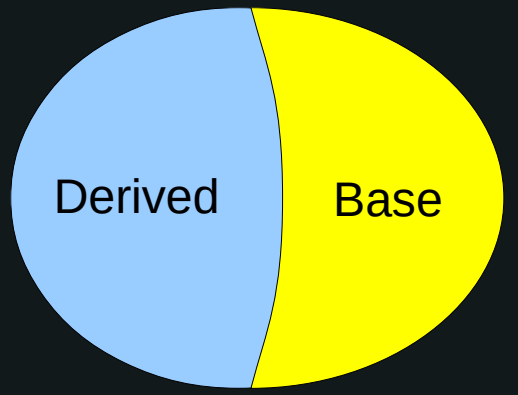
Virtual Destructor

VEDINESH

```
class base
{
  public:
          ~base( )
          {
             cout << “Base Class Destroyed”;
          }
}
```

```
class derived: public base
{
  public:
          ~derived( )
          {
             cout << “Derived Class Destroyed”;
          }
}
```

```
void main ( )
{
   base *b1 =  new derived;

   delete b1;
}
```



Note:- In this case only base class destructor is called.

Virtual Destructor

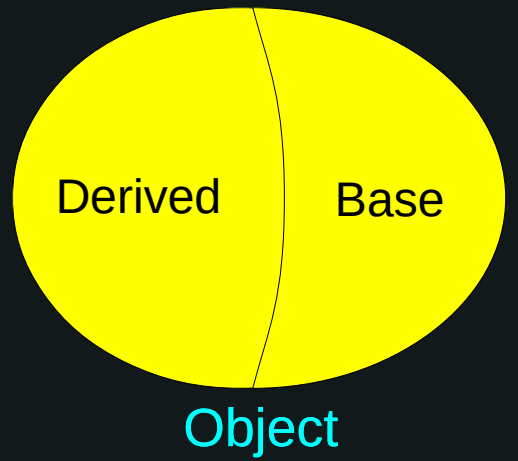
VEDINESH

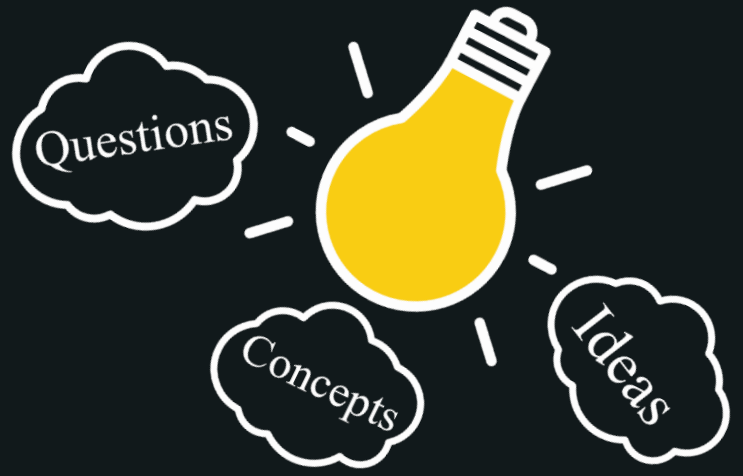
```
class base
{
  public:
          virtual ~base( )
          {
             cout << “Base Class Destroyed”;
          }
}
```

```
class derived: public base
{
  public:
          ~derived( )
          {
             cout << “Derived Class Destroyed”;
          }
}
```

```
void main ( )
{
   base *b1 =  new derived;

   delete b1;
}
```

Earlier





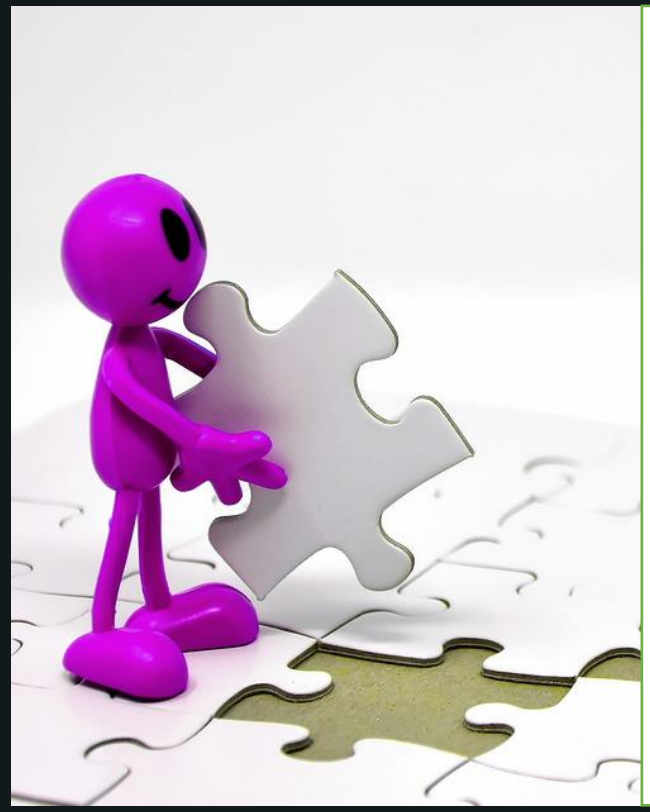
Friend Function & Class

C++ Programming

VEDINESH

Friend

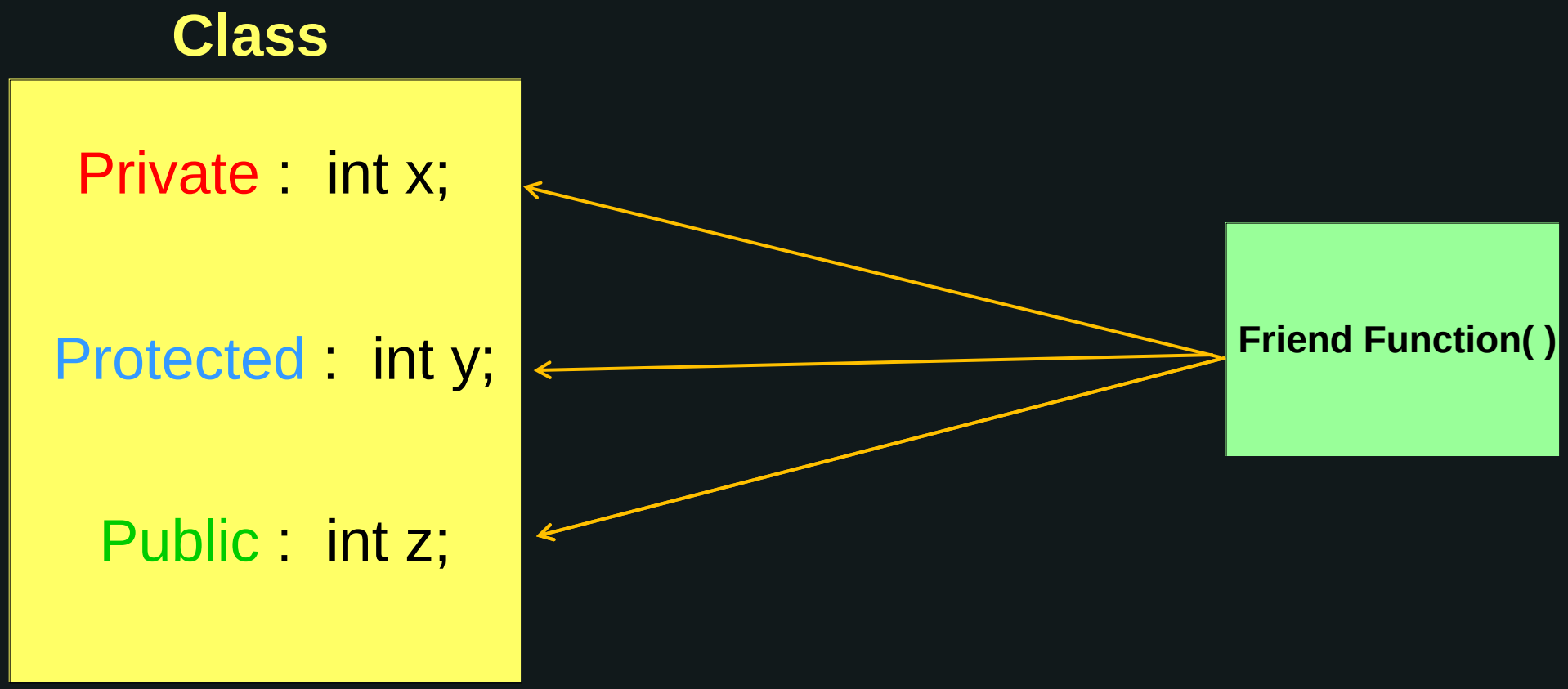
Friend



- Reduce Work Load
- Speed Up Work
- Less Chances Of mistake
- High Standard of output
- Better Work Management

Why Friend Function

VEDINESH



Friend Function

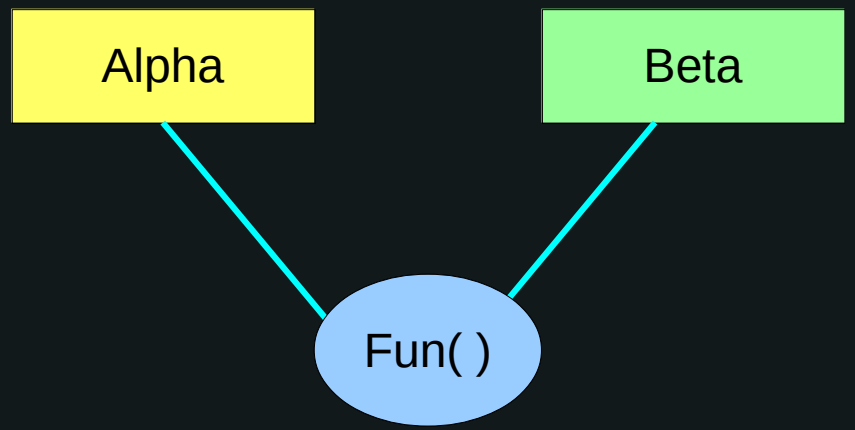
VEDINESH

```
class Alpha
{
  private:
            int a1;
  public:
            Alpha( int arg = 0)
            {  a1 = arg; }
            friend void Fun( );
};
```

```
class Beta
{
  private:
            int b1;
  public:
            Beta( int arg = 0)
            {  b1 = arg; }
            friend void Fun( );
};
```

```
 void Fun( )
 {
    Alpha a(8);
    Beta b(2);

    int x = a.a1 +  b.b1;

    cout << "Sum "  << x;
}
```

```
void main( )
{
   Fun( );
}
```



NOTE:- Friend function connecting two class or more(act as bridge).

Questions

VEDINESH

Class can have more than one friend function ? Yes

Can a same function can become friend of multiple classes ? Yes

Friend function can be invoked as normal function ? Yes

Do we need to create object of class in order to access friend function ? No

Friend Class

VEDINESH

Class Suzi

Class Jena

Class Suzi

Class Jena

Can Access and use features and functionalities of each other.

Friend Class

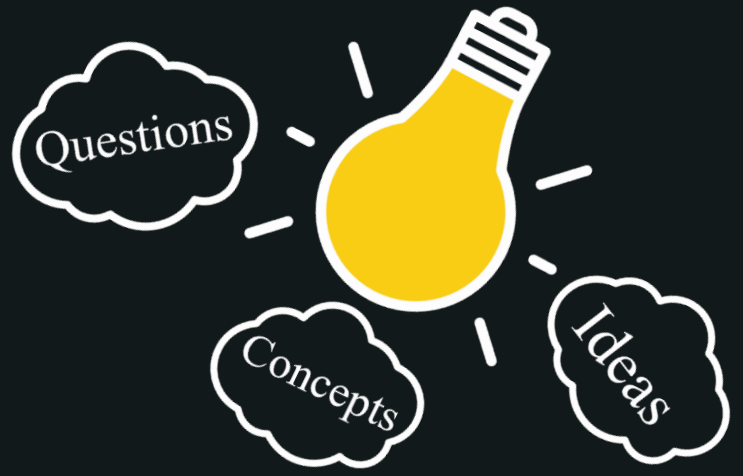
VEDINESH

```
class Alpha
{
 private:
            int a1;
 public:
            Alpha( int arg = 0)
            {  a1 = arg; }

            friend class Beta;
};
```

```
class Beta
{
 private:
            int b1;
 public:
            Beta( int arg = 0)
            {  b1 = arg; }

            void Sum( )
            {
              Alpha alpha_obj (3);

              int sum = alpha_obj.a1 + b1;

              return sum;
            }
};
```

```
 void main( )
 {
    Beta beta_obj(7);

    beta_obj.Sum( );
}
```

NOTE:-Now all memeber functions of Beta class can access private data of Alpha .



Static Member &Function

C++ Programming

VEDINESH

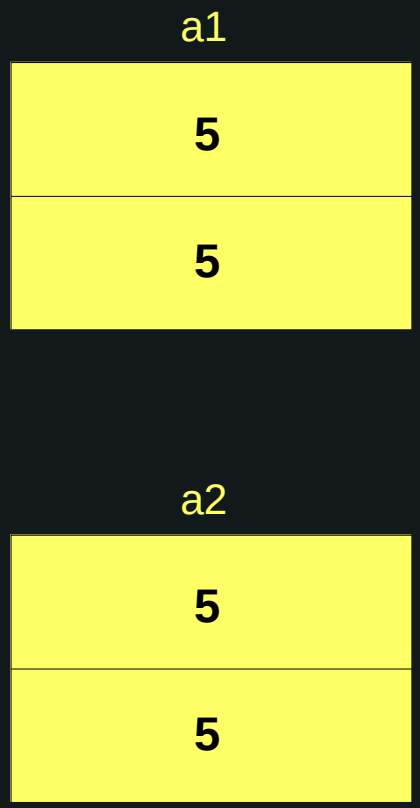
Static Members

VEDINESH

```
class Alpha
{
  private:
            int a;
            int b;
  public:
            Alpha( )
            {
              a = 5;
              b = 5;
            }
};
```

```
 void main( )
 {
    Alpha a1;
    Alpha a2;
}
```

NOTE:- Each Object will create separate copy of itself in memory .



Static Member

VEDINESH

```
class Alpha
{
 private:
            int a;
            int b;
 public:
            Alpha( )
            {
              a = 5;
              b = 5;
              stat ++;
            }

            static int stat;
};
```

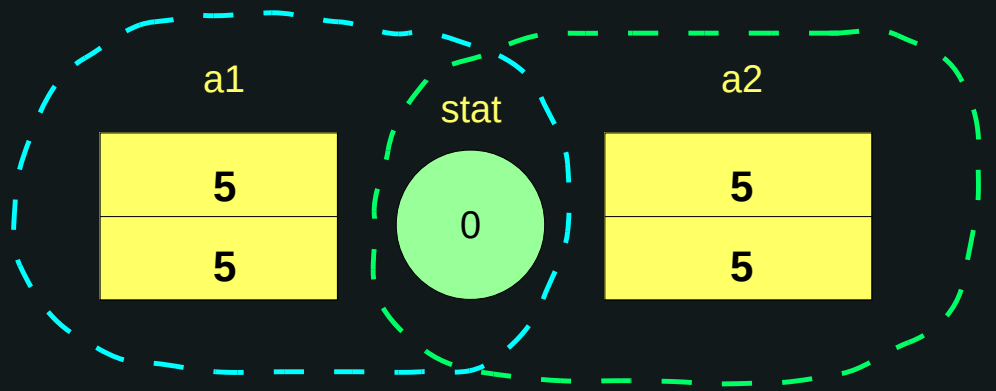
```
int  Alpha ::  stat = 0;
```

```
 void main( )
 {
    Alpha a1;
    Alpha a2;

    cout <<  a1.stat;

    cout <<  a2.stat;

    cout <<  Alpha::stat;
}
```

NOTE:- Static member would be allocated memory only once .

NOTE:- And that memory is shared by both the objects.

NOTE:- Static Data members belong a class & common to all objects.



Static Member Function

```
class Alpha
{ private:
            int a;
            int b;
  public:
            Alpha( )
            {  a = 5;
               b = 5;
            }

            static int stat;

            static int getStat( )
           {  stat ++ ;
            return stat ; }
};

int  Alpha ::  stat = 0;
```

```
void main( )
{
    cout << Alpha :: getStat( );
    Alpha a1;
    Alpha a2;
    cout << Alpha :: getStat( );
    cout << a1.getStat( );
    cout << a2.getStat( );
 }
```

NOTE:- Static member can only access static members .