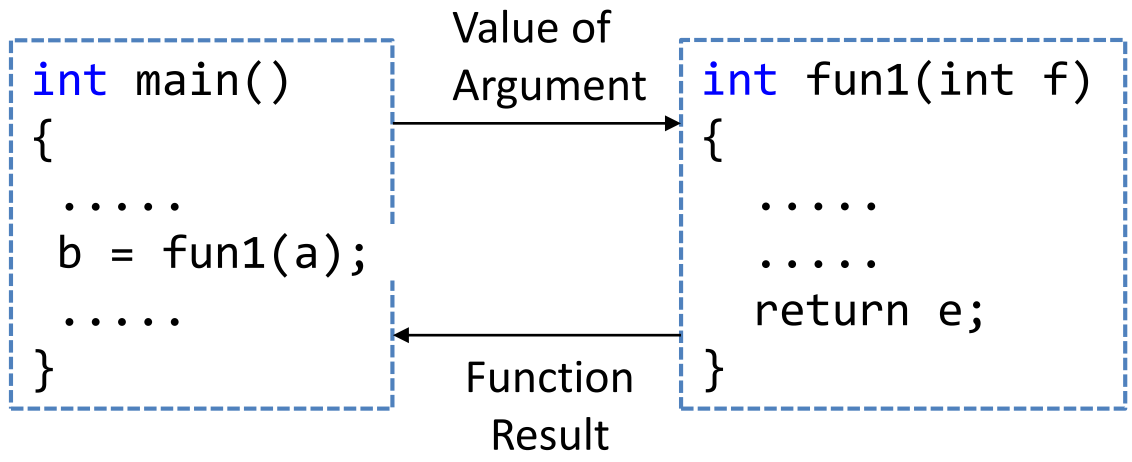
# Passing Objects as Function Arguments

# Function with argument and returns value

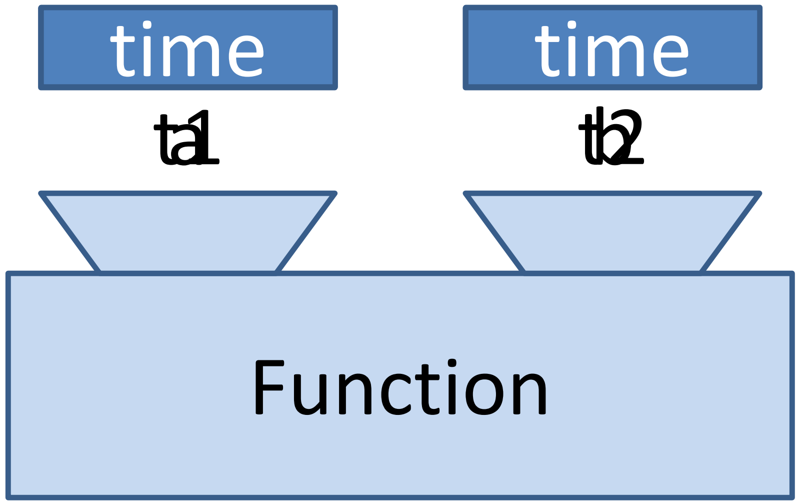
```cpp
#include <iostream>
using namespace std;

int add(int, int);

int main(){
  int a=5,b=6,ans;
  ans = add(a,b);
  cout<<"Addition is="<<ans;
  return 0;
}
int add(int x,int y)
{
  return x+y;
}
```



```cpp
int main()
{
 .....
 b = fun1(a);
 .....
}
```

```cpp
int fun1(int f)
{
  .....
  .....
  return e;
}
```

# Object as Function arguments



```
void add(int x, int y)
{
   statements…
}
int main()
{
   int a=5,b=6;
   add(a,b);
}
```

```
void addtime(time x, time y)
{
   statements…
}
int main()
{
   time t1,t2,t3;
   t3.addtime(t1,t2);
}
```

## Program: passing object as argument

```
class Time
{
     int hour, minute, second;
   public :
     void getTime(){
       cout<<"\nEnter hours:";cin>>hour;
       cout<<"Enter Minutes:";cin>>minute;
       cout<<"Enter Seconds:";cin>>second;
     }
     void printTime(){
       cout<<"\nhour:"<<hour;
       cout<<"\tminute:"<<minute;
       cout<<"\tsecond:"<<second;
     }
     void addTime(Time x, Time y){
       hour = x.hour + y.hour;
       minute = x.minute + y.minute;
       second = x.second + y.second;
     }
};
```

## Program: passing object as argument

```
int main()
{
  Time t1,t2,t3;

  t1.getTime();
  t1.printTime();

  t2.getTime();
  t2.printTime();

  t3.addTime(t1,t2);
  cout<<"\nafter adding two objects";
  t3.printTime();

  return 0;
}
```

# Program: Passing object as argument

- Define class **`Complex`** with members **`real`** and **`imaginary`** . Also define function to **`setdata()`** to initialize the members, **`print()`** to display values and **`addnumber()`** that adds two complex objects.
- Demonstrate concept of passing object as argument.

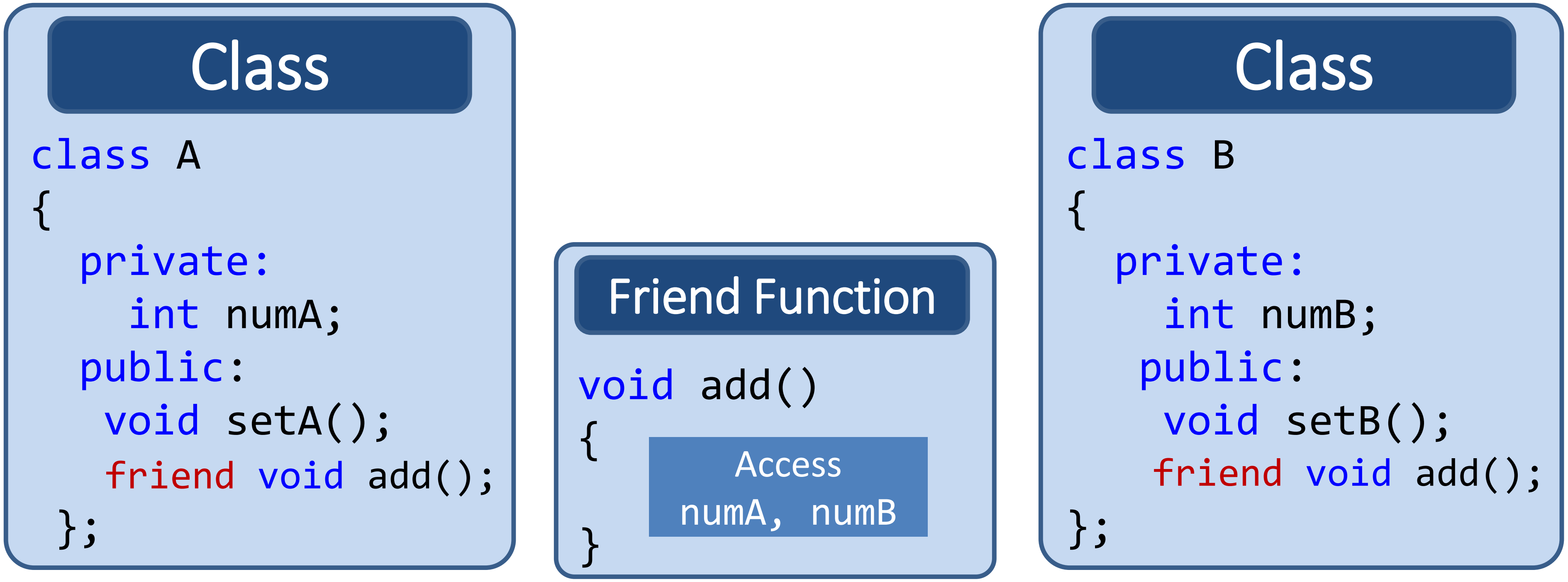
## Program: Passing object as argument

```
class Complex
{
 private:
   int real,imag;
 public:
   void readData()
   {
     cout<<"Enter real and imaginary number:";
     cin>>real>> imag;
   }
   void addComplexNumbers(Complex comp1, Complex comp2)
   {
     real=comp1.real+comp2.real;
     imag=comp1.imag+comp2.imag;
   }
   void displaySum()
   {
     cout << "Sum = " << real<< "+" << imag << "i";
   }
};
```

```
int main()
{
   Complex c1,c2,c3;
   c1.readData();
   c2.readData();
   c3.addComplexNumbers(c1, c2);
   c3.displaySum();
}
```

# Friend Function

- In C++ a **Friend Function** that is a "friend" of a given class is allowed **access to private and protected data** in that class.
- A friend function is a function which is declared using **friend** keyword.



**Class**

```
class A
{
  private:
    int numA;
  public:
   void setA();
   friend void add();
 };
```

**Friend Function**

```
void add()
{
     Access
     numA, numB
}
```

**Class**

```
class B
{
  private:
    int numB;
   public:
    void setB();
    friend void add();
};
```

# Program: Friend Function

```
class numbers {
  int num1, num2;
  public:
   void setdata(int a, int b);
   friend int add(numbers N);
};
void numbers :: setdata(int a, int b){
  num1=a;
  num2=b;
}
int add(numbers N){
  return (N.num1+N.num2);
}
```

```
int main()
{
  numbers N1;
  N1.setdata(10,20);
  cout<<"Sum = "<<add(N1);
  return 0;
}
```

# Program: Friend Function

```
class Box {
   double width;
public:
   friend void printWidth( Box );
   void setWidth( double wid );
};
void Box::setWidth( double wid ) {
   width = wid;
}
void printWidth(Box b) {
  cout << "Width of box : " << b.width;
}
int main( ) {
Box box;
box.setWidth(10.0);
printWidth( box );
return 0;
}
```

## Constructors

## What is constructor ?

A **constructor** is a block of code which is,

- similar to member function
- has same name as class name
- called automatically when object of class created

A **constructor** is used to initialize the objects of class as soon as the object is created.

## Types of Constructors

# Types of Constructors

1) Default constructor
2) Parameterized constructor
3) Copy constructor

# Program: Types of Constructor

- Create a class **`Rectangle`** having data members **`length`** and **`width`**. Demonstrate default, parameterized and copy constructor to initialize members.

# Program: Types of Constructor

This is constructor overloading

```
class rectangle{
   int length, width;
   public:
   rectangle(){ // Default constructor
     length=0;
     width=0;
   }
   rectangle(int x, int y){// Parameterized
                                          constructor
     length = x;
     width = y;
   }
   rectangle(rectangle &_r){ // Copy constructor
     length = _r.length;
     width = _r.width;
   }
};
```

## Program: Types of Constructor (Cont…)

```
int main()
{
  rectangle r1; // Invokes default constructor
  rectangle r2(10,20); // Invokes parameterized
                          constructor
  rectangle r3(r2); // Invokes copy constructor
}
```

# Destructor

## Destructor

- **Destructor** is used to destroy the objects that have been created by a constructor.
- The syntax for **destructor** is same as that for the constructor,
  - the class name is used for the name of destructor,
  - with a **tilde (~)** sign as prefix to it.

```
class car
{
    float mileage;
  public:
   car(){
     cin>>mileage;
   }

   ~car(){
cout<<" destructor";
   }

};
```

**Destructor**

- never takes any argument nor it returns any value nor it has return type.
- is invoked automatically by the complier upon exit from the program.
- should be declared in the public section.

# Program: Destructor

```
class rectangle
{
   int length, width;
   public:
   rectangle(){ //Constructor
    length=0;
    width=0;
    cout<<”Constructor Called”;
   }
   ~rectangle() //Destructor
   {
    cout<<”Destructor Called”;
   }
// other functions for reading, writing and
processing can be written here
};
```

```
int main()
{
   rectangle x;
// default
constructor is
called
}
```