# C++ slides - I

**Objects and Classes**: Structure in C and C++, Class specification, Objects, Data hiding, Encapsulation and abstraction, namespaces, Array of objects, Passing objects as arguments, Returning object from a function, inline functions, Static data member and member function, ‘const’ member function.

# Basic program

```
#include<iostream>
using namespace std;
int main(){
cout<<"Hello world"<<endl;
}
```

# cin & cout

- cout works similar to printf
- cin works similar to scanf

**Example**

```
cout<<“Hello and welcome”;
cin>>variable;   //variable could be int, float, char array (without space)
```

# cout basics

- ‘\n’ is for new line, or you can use *endl*

  cout << endl << “message”;
- ‘\t’ is for tab
- ‘\a’ is an alarm sound
- ‘\r’ is carriage return to go to the beginning of the current line

# Input char array with blanks

```
#include<iostream>
using namespace std;
int main(){
char str[20];
cin.getline(str,sizeof(str));
cout<<str<<endl;
}
```

# Header files and namespace

# Header file and more

- #include <iostream>  // input-output stream for cin/cout
- using namespace std;

- Namespaces allow us to  differentiate same named  entities in various libraries. It is just a region of the code or library and not a function.
- ***std*** stands for standard I/O on the console screen.

# Need of namespace

```
#include <iostream>
int main() {
    int value;
    value = 0;
    double value; // Error here due to reuse of value
    value = 0.0;
}
```

# Example – namespace defines the *scope*

```
#include <iostream>
using namespace std;
namespace ns1 { int value() {return 5;}}
namespace ns2 { int value() {return -5;}}

int main() {
cout << ns1::value() << '\n';  //5 will be displayed
cout << ns2::value() << '\n';     // -5 will be displayed
}
```

# What will be the output?

```
#include <iostream>
using namespace std;
namespace ns1 { int value() {return 5;}}
namespace ns2 { int x=10; int value() {return 4;}}
 int main() {
cout << ns1::value() << endl;
cout << ns2::value() << endl;
cout<< ns2::x<<endl;
}
```

# Without namespace could cause :: pollution

```
#include<iostream>

int main(){
std::cout << “Hello there” << std::endl;
return 0;
}
```

# using namespace std;

- Thus using namespace std;  means cin/cout will be performed through standard console screen.

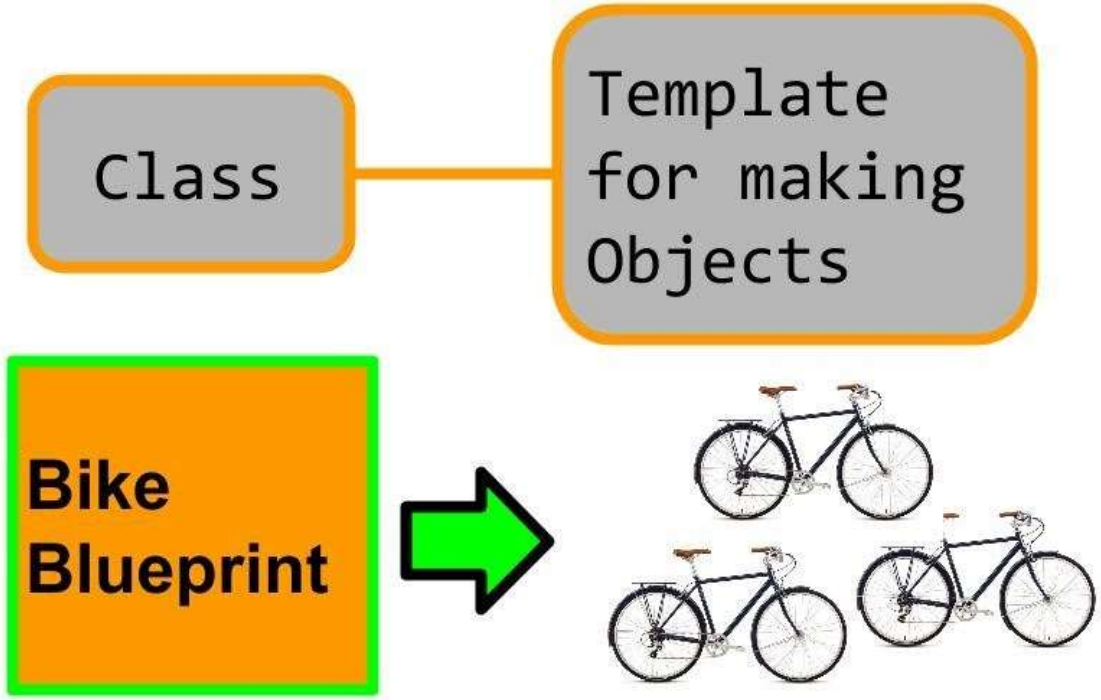
# Header file and namespace

- #include<iostream> includes all necessary files required of CIN and COUT opertions.
- using namespace std; allows us to reduce :: pollution for simpler programs

# Classes in C++

- **Class:** A class in C++ is the building block of object oriented programming
- It is User Defined Datatype (UDT) which has data & functions
- **Object** is an instance of a Class i.e. variable of UDT.

# What is an Object?



| CLASS | OBJECT |
| --- | --- |
| Class is a data type | Object is an instance of Class. |
| It generates OBJECTS | It gives life to CLASS |
| Does not occupy memory location | It occupies memory location. |
| It cannot be manipulated because it is not available in memory *(except static class)* | It can be manipulated. |

Object is a class in *"runtime"*

# Structure in C++

```
#include <iostream>
using namespace std;
struct Person{int age;};
int main(){
    Person p1;    // No need to write struct Person p1 in C++.
    cout << "Enter age: ";cin >> p1.age;
    cout <<"Age: " << p1.age << endl;
    return 0;
}
```

# Structures in C++ vs in C

1. Functions can be defined inside structure in C++
2. Using ***struct*** keyword not required in C++
3. C++ structures can have static members
4. C++ allows data hiding by using access modifiers

```
#include <iostream>
using namespace std;
struct Person{
int age;                    //variable
int setAge(int a){age = a;}    //function
int display() {cout <<"Age: " << age << endl;} //function
};
int main(){
    Person p1;
    p1.setAge(20); p1.display();
    return 0;
}
```

# Need more knowledge for 3 & 4

1. Functions can be defined inside structure in C++
2. Using ***struct*** keyword not required in C++
3. C++ structures can have static members
4. C++ allows data hiding by using access modifiers

# struct vs class

- Classes in C++ are similar to struct for syntax
- *struct* – everything is public by default
- *class* – everything is private by default.

What is the output of the following program?

```
#include <iostream>
using namespace std;
class Person{
int age;
int setAge(int a){age = a;}
int display() {cout <<"Age: " << age << endl;}
};
int main(){
    Person p1;
    p1.setAge(20); p1.display();
    return 0;
}
```

# Answer - Output – Compilation Error

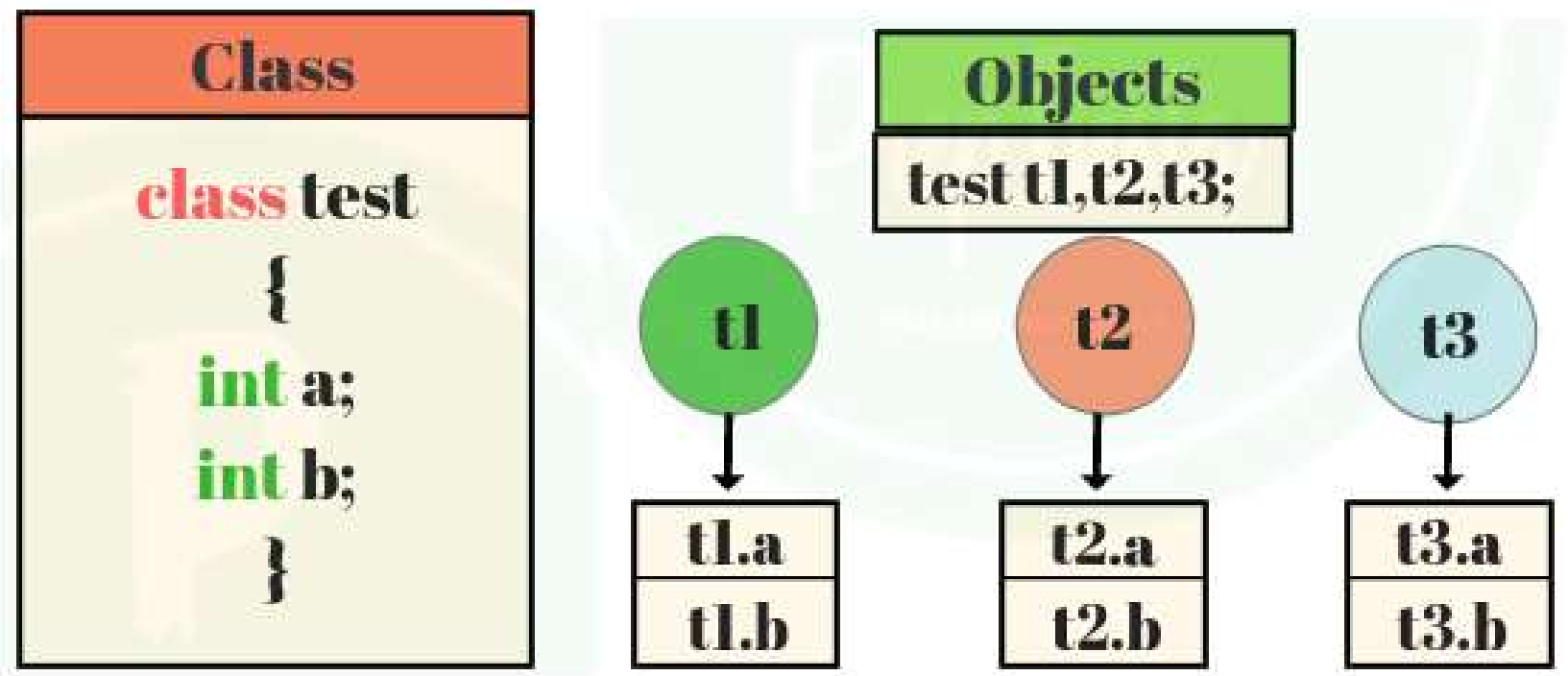
[Error] 'int Person::setAge(int)' is private

Now we use public: specifier for data and functions in class

```
#include <iostream>
using namespace std;
class Person{
public:        // Everything after public:  becomes public
int age;
int setAge(int a){age = a;}
int display() {cout <<"Age: " << age << endl;}
};
int main(){
    Person p1;    p1.setAge(20);
    p1.display();   return 0;
} // everything public in class means it becomes a struct
```

Output =  Age: 20

# Each object has its own variables/functions



# Access specifiers in C++

- Public
- Private
- Protected (for later until we cover inheritance)

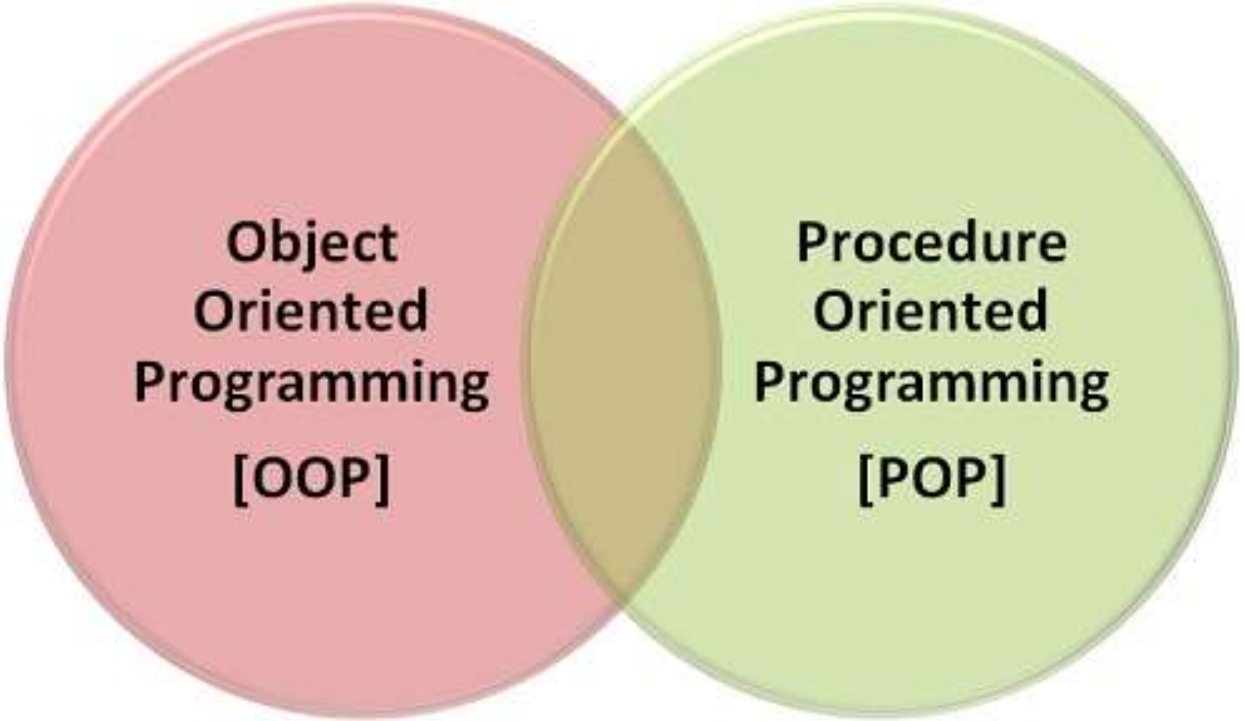
Data private and functions public.

What will be the output of following program?

```
#include <iostream>
using namespace std;
class Person{
int age;
public:          // Everything before public:  is private
void setAge(int a){age = a;}
int display() {cout <<"Age: " << age << endl;}
};
int main(){
    Person p1;    p1.setAge(20);
    p1.display();   return 0;
}
```

```
#include <iostream>
using namespace std;
class Person{
int age;
public:
void setAge(int a){age = a;}  //setter function
int getAge() {return age;}  //getter function
};
int main(){
    Person p1;    p1.setAge(20);
    cout <<"Age: " << p1.getAge() << endl;
}
```

# C++ versus C language



| Sr. | C language | C++ |
|---|---|---|
| 1 | Functions are basic elements | Classes are basic elements |
| 2 | Focus on global functions | Focus on encapsulation (data+functions) |
| 3 | Functions share global data | Data and function access is controlled |
| 4 | Data moves openly | Data is bounded with classes |
| 5 | Top down approach (break down into functions) | Bottom up (clubbing of data and functions together) |

# Four *features* of object oriented programming

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

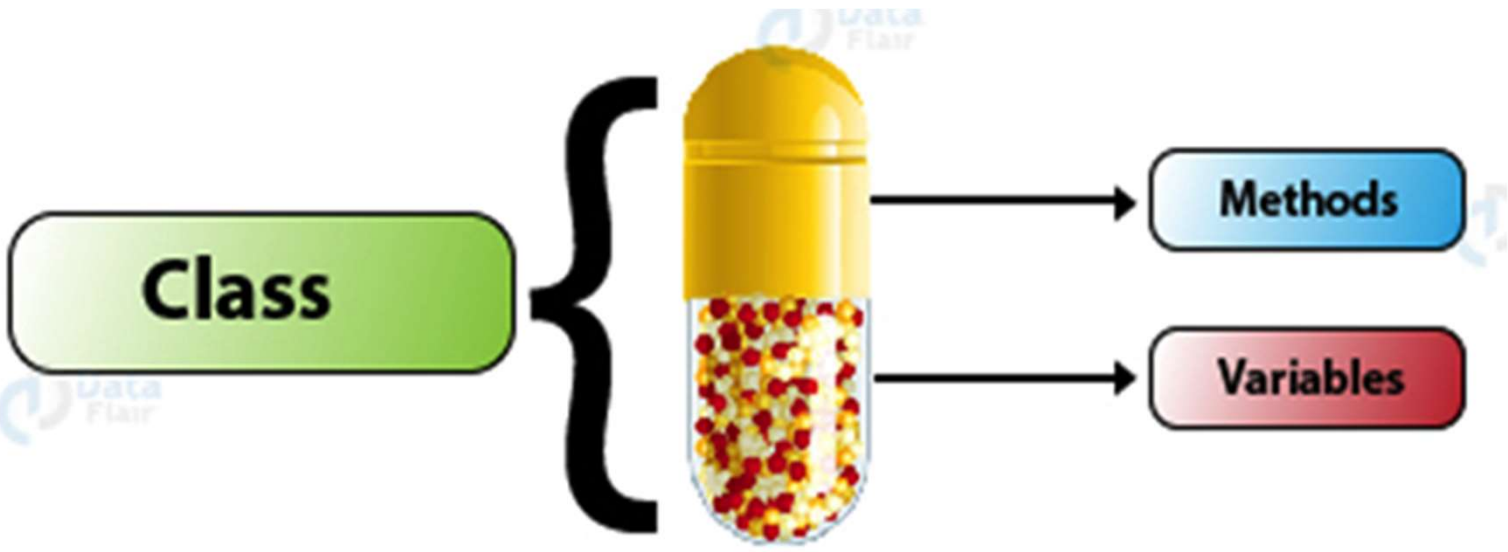
ENCAPSULATION

ABSTRACTION

INHERITANCE

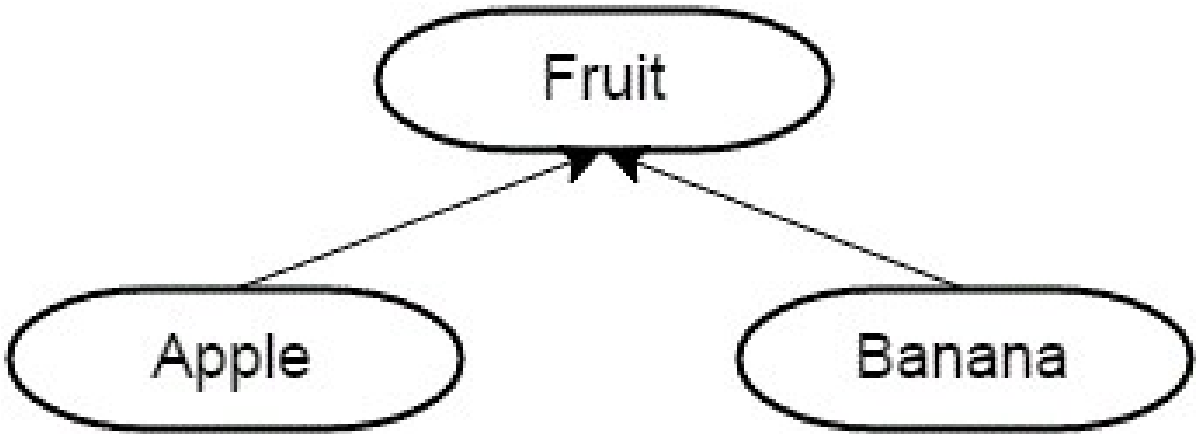
POLYMORPHISM

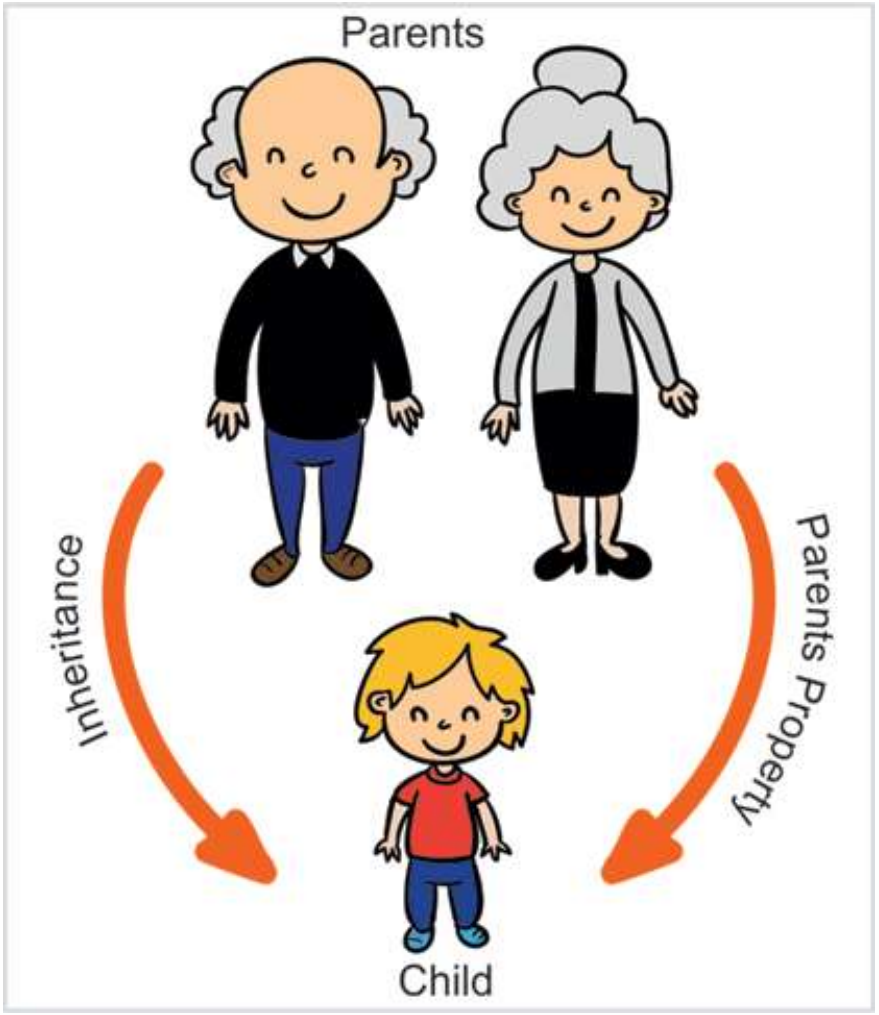
# Encapsulation



# Abstraction

# Inheritance



# Polymorphism – late binding flexibility

# :: for global variable

```
#include<iostream>
using namespace std;
int x=100;
int main(){
        int x=-100;
        cout<<"x = "<<x<<endl;
        cout<<"x = "<<::x<<endl;
        return 0;
}
```

# :: for function def

```
class A{
public: void fun();   //only declaration
};
void A::fun(){      //function definition
cout<<"in fun()"<<endl;
}
int main(){
A obj;  obj.fun();
}
```

# Constructor and Destructor functions

- Constructor/Destructor is a member function which has to be public.

- They are called automatically to construct (initialize) and destruct (delete) the object variables.

# Constructor - basics

- Same name as the class
- No return statement
- Automatically called when object is created
- Compiler uses default empty constructor if no constructor is defined

# Destructor

- C++ destructors are used to de-allocate the memory allotted by constructor.
- It has same name as class preceded by a tilde sign ~

# Basic constructor and destructor

```
class Test {
public:
    Test() {cout<<“in constructor”<<endl;}
~Test() { cout << “In destructor” << endl; }
};
int main() {
Test c;   cout <<“In main”<<endl;;
}
```

# Static variable in Class

```
class X{
static int i;
public:void show(){cout<<"i ="<<i<<endl;}
};
int X::i=1;
int main(){
X obj; obj.show();
}
```

# Static function in Class

```
class X {
public:
static void f(){cout<<"In static f()"<<endl;}
};
int main(){
    X::f();   // direct call without an object
}
```

# Array of objects

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

class A{
        int a; char c;
        };
int main(){
A a[3];         // each of a[i] will have an int and a char
}
```

# Passing and returning an object

```
class A{
int i;
public:  A(){i=10;}
void show(){cout<<"i = "<<i<<endl;}
A makedouble(A obj){A temp; temp.i = 2*obj.i; return temp;}
};
int main(){
A a1,a2;  a1.show();  a2 = a1.makedouble(a1);  a2.show();
}
```

# Inline functions

```
inline int cube(int s){ return s*s*s; }
int main() {
cout << "The cube of 3 is: " << cube(3) << endl;
}
```

# Inline function properties

- Reduces function-call overhead
- Asks the compiler to copy code into program instead of using function call
- Compiler can ignore inline
- Should be used for small, often used functions

# ‘const’ member function

```
int main(){
    const int i = 10;
    const int j = i + 10;     // works fine
    i++;    // this leads to Compile time error
}
```

# Const class variable

```
class Test{
    const int i;
    public:
    Test(int x):i(x) {}   //initialized using constructor
    void show(){cout<<"i="<<i<<endl;}
};
int main(){Test t(190);t.show(); }
```

# Const function

- The idea of *const* functions is not to allow them to modify the object on which they are called.
- It is recommended to make as many functions *const* as possible so that accidental changes to objects are avoided.

# Const class *member* function

```
class A{
public:  int x;
void func() const{
                    x = 0; // [Error] can't modify object variable
                }
};
int main(){}
```

# Extra concepts

Const function and object

# *const* function should be a member function

```
#include<iostream>
using namespace std;
int i = 99;
void fun() const{}
int main(){}

//What will be the output?
```

# Output

- [Error] non-member function 'void fun()' cannot have cv-qualifier

- Since there is no class where this function belongs to, you will get an error.

# const class object

- In *const* class object, member variables cannot be modified
- Calling member functions that change the value of member variables is also prohibited.

```
// Example of const object
class Test{
public:
    int i;
    Test(): i(0) {}
    void setValue(int a) { i=a;}
};
int main(){
    const Test t; // calls default constructor
     //t.i = 5; // [Error]
    //t.setValue(5); // [Error]
}
```

# String basics - 1

```
int main(){
string name;
cout<<"Enter name: ";
getline(cin,name);  // cin>>name; will only take first word
if(name.compare("rocky sharma")==0)
 cout<<"Same"<<endl;
 else cout<<"Different"<<endl;
}
```

# String concat (connect)

```
int main(){
string name1,name2;
cout<<"string 1: "; cin>>name1;
cout<<"string 2: "; cin>>name2;
cout<<"String concat = "<<name1+", "+ name2;
}
// + means connect strings together
```