WORKING WITH CLASSES

Constructors & Destructors cont…

2

PRACTICE EXERCISE

CAR & ENGINE EXAMPLE

```
class Engine{
  private:

    int cylinders; // Number of cylinders
    double liters; // Capacity in liters

 public:

   Engine(int c = 4, double l = 1.3); // Constructor
};

Engine::Engine(int c, double l) : cylinders(c), liters(l)
{
 cout<<"Constructor Engine called "<<endl;
 cout<<"Cylinders: "<<cylinders<<endl;
 cout<<"Liters: "<<litres<<endl;
}
```

```
class Car
{
private:
      int modelNumber;    // Model number
      Engine motor;        // Car's engine

          public:
     // Constructor
     Car(int m, int c, double l);
};

Car::Car(int m, int c, double l):modelNumber(m),
motor(c, l)
{
        cout<<"Constructor Car called"<<endl;
        cout<<"Model No: "<<modelNumber<<endl;
}

int main(){
        Car c(2,3,2.3);
}
```

4

CLASSES & CLASS VARIABLE/MEMBER VARIABLES

A SIMPLE CLASS

```
class Circle{
    double radius;

    public:
        Circle(){ radius=0.0;}
        double area();
};
```

In OOP

- Class is the blueprint /template
- Use the template to create Objects
- Class includes class variables
  - Also known as data members / member variables
- Class Constructors
  - Executes / invoked when an instant is created using the class /create a new instance of a class.
- Member Functions
  - Get different functionalities

DEFINING CONSTRUCTORS

```
Circle ()
{
  radius = 0;
}
```

```
Circle (): radius (0)
{  }
```

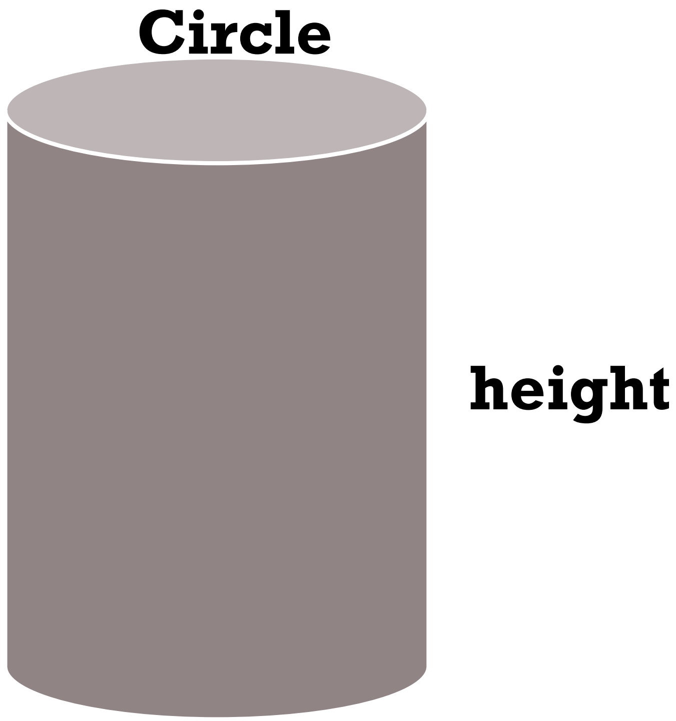
```
Circle (int r)
{
  radius = r;
}
```

```
Circle (int r): radius (r)
{  }
```

DEFINING CLASS OBJECTS WITHIN A CLASS

Member Object

```
class Cylinder {
   Circle base;
   double height;
};
```



class Cylinder has a member object whose type is another class (In this example: Circle)

DEFINING CLASS OBJECTS WITHIN A CLASS

```
class Circle{
    double radius;
        public:
            Circle():radius(0) {
                cout<<"leaving the circle constructor" <<endl;
            }
            Circle (double r):radius(r){
                cout<<"leaving the Circle constructor: Radius
        passed as a variable"<<endl;
             }
             double area();
};
```

DEFINING CLASS OBJECTS WITHIN A CLASS

```
class Cylinder {
   Circle base;
   double height;
 public:
   Cylinder(double r, double h) : height(h), base(r)
    {
        cout<<"leaving the Cylinder constructor"<<endl;
    }
   double volume() {return base.area() * height;}
};
```

DEFINING CLASS OBJECTS WITHIN A CLASS

```
Cylinder Cyl1(3,4);
```

1. Create a class object "Cyl1" of the type "Cylinder".
2. The constructor will be called.

   ```
   Cylinder(double h, double r) : height(h), base(r)
   ```

3. Create a double variable named height and assign it a value of h.
4. Create a class object name "base" and call the corresponding constructor to initialize the variables of object "base".
5. The Circle constructor will be called.

   ```
   Circle (double r):radius(r){
   cout<<"leaving the Circle constructor: Radius passed as a variable"<<endl; }
   ```

6. End of the Circle constructor.
7. End of the Cylinder constructor.
8. End of the main() function.

```
class Engine{
   private:

     int cylinders; // Number of cylinders
     double liters; // Capacity in liters

 public:

    Engine(int c = 4, double l = 1.3); // Constructor
};

Engine::Engine(int c, double l) : cylinders(c), liters(l)
{
 cout<<"Constructor Engine finished initializing the
variables"<<endl;
 cout<<"Cylinders: "<<cylinders<<endl;
 cout<<"Liters: "<<litres<<endl;
}
```

```
class Car
{
private:
      int modelNumber;    // Model number
      Engine motor;         // Car's engine

          public:
     // Constructor
     Car(int m, int c, double l);
};

Car::Car(int m, int c, double l):modelNumber(m),
motor(c, l)
{
        cout<<"Constructor Car finished initializing the
variables"<<endl;
        cout<<"Model No: "<<modelNumber<<endl;
}

int main(){
        Car c(2,3,2.3);
}
```

```
class Engine{
  private:

   ❻ int cylinders; // Number of cylinders
   ❼ double liters; // Capacity in liters

 public:

   Engine(int c = 4, double l = 1.3); // Constructor
};

❺ Engine::Engine(int c, double l) : cylinders(c),
liters(l)
{ ❽
 cout<<"Constructor Engine finished initializing the
variables"<<endl;
 cout<<"Cylinders: "<<cylinders<<endl;
 cout<<"Liters: "<<litres<<endl;
} ❾
```

```
class Car
{
private:
     ❹ int modelNumber;    // Model number
     ❺ Engine motor;       // Car's engine

         public:
     // Constructor
     Car(int m, int c, double l);
};

❸ Car::Car(int m, int c, double l):
modelNumber(m), motor(c, l)
{ ❿
       cout<<"Constructor Car finished initializing the
variables"<<endl;
       cout<<"Model No: "<<modelNumber<<endl;
} 11

❶ int main(){
       ❷ Car c(2,3,2.3);
}12
```

WHEN TO USE CONSTRUCTORS/DESTRUCTORS

- Order of calls depends on order of execution (when execution enters and exits scope of objects)
- Generally, destructor calls reverse order of constructor calls
- Global scope objects
  - Constructors are called before any other function (including main)
  - Destructors are called when main terminates (or exit function called)
    - Not called if program terminates with abort
- Automatic local objects
  - Constructors are called when objects defined and each time execution enters scope
  - Destructors are called when objects leave scope / execution exits block in which object defined
    - Not called if program ends with exit or abort

STATIC KEYWORD IN C++

- Allocate storage only once in a program lifetime in static storage area.
- The scope exists till the program ends (program lifetime).
- Static Keyword can be used to define,
  - Variable in functions
  - Class Objects
  - member Variable in class
  - Methods in class

STATIC VARIABLES IN A FUNCTION

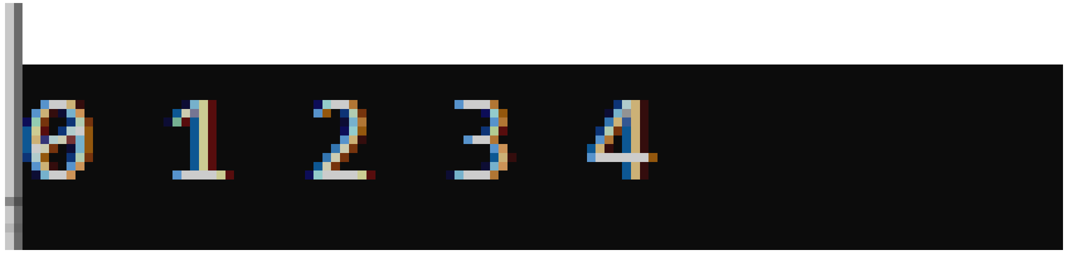
- The variables declared as static are initialized only once.
- They are allocated space in separate static storage.
- Space for **it gets allocated for the lifetime of the program**.
- The value of variable in the previous call gets carried through the next function call.
- Even if the function is called multiple times, space for the static variable is allocated only once

EXAMPLE

```
void demo()
{
    static int count = 0;
    cout << count << " ";

      count++;
}

int main()
{
    for (int i=0; i<5; i++)
        demo();
}
```



```
0 1 2 3 4
```

STATIC VARIABLES IN A CLASS

- The static variables **in a class are shared by the objects.**
- Can not have multiple copies of same static variables for different objects.
- Can not be initialized using constructors. because these are not dependent on object initialization
- A static variable inside a class should be initialized explicitly always outside the class using the class name and scope resolution operator outside the class.

EXAMPLE:

```
Type ClassName::variable_name = value;
```

ACCESSING STATIC VARIABLES IN A CLASS

- A member function can refer to a static data member of the same class directly.
- A non-member function can refer to a static data member using either the notation

ClassName::staticMemberName or
objectName.staticMemberName

STATIC CLASS OBJECTS

- Like variables, objects also when declared as static have a scope till the lifetime of program.
- Exists the scope at the end of program.

EXERCISE

```
int main()
{
  int x=0;
  if(x==0){
    static StaticObject obj;
  }
    cout << "End of main\n";
 }
```

```
class StaticObject
{
   int i;
   public:
      StaticObject():i(0)
      {
         cout << "Inside
Constructor"<<endl;
      }
      ~StaticObject()
      {
         cout << "Inside Destructor\n";
      }
};
```

STATIC FUNCTIONS IN A CLASS

- Work for the class as whole rather than for a particular object of a class.
  - Does not depend on object of class.
- It can be called using an object and the direct member access . operator.
  - Recommended to invoke the static members using the class name and the scope resolution operator.
- Allowed to access only the static data members or other static member functions
  - Can not access the non-static data members or member functions of the class.

STATIC MEMBER FUNCTIONS

- Like static data members, static member functions are associated with a class and not with any particular object of that class.
- Static member functions do not have a this pointer.
- Static member functions cannot be declared const.
- A member function can call a static member function of the same class directly.
- A non-member function can call a static member function using either the below notation

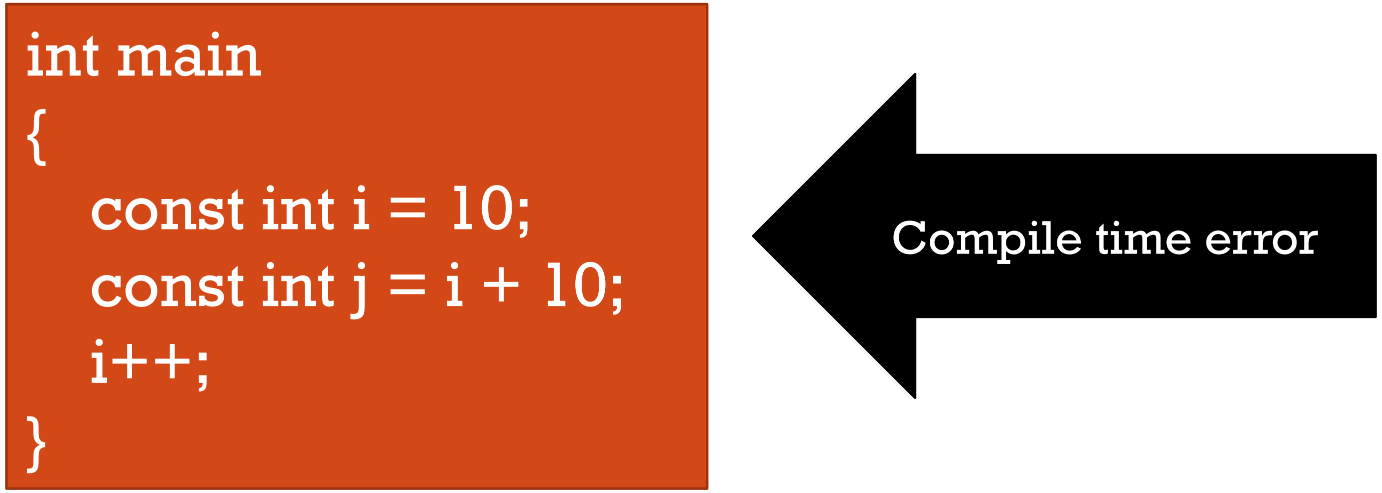
ClassName::staticMemberName() or
objectName.staticMemberName()

CONSTANTS

- Variables
- Pointers
- Function arguments and return types
- Class Data members
- Class Member functions
- Objects

CONSTANT VARIABLES IN C++

- Cannot change its value of the variable.
- Value of the variable will not change during its lifetime.
- Constant variables must be initialized while they are declared.



```
int main
{
    const int i = 10;
    const int j = i + 10;
    i++;
}
```

Compile time error

POINTERS WITH CONST KEYWORD

Declaration can be done in TWO ways.

- Pointer to a constant variable
  - The pointer is pointing to a const variable.

```
const int* u;
```

- Constant Pointer
  - Cannot change the pointer, to which it points to
  - Can change the value that it points to, by changing the value of the variable.
  - Useful when one want to change the value but not the memory.
    - The pointer will always point to the same memory location, as it is defined with const keyword, but the value at that memory location can be changed.

```
int x = 1;
int* const w = &x;
```

CONSTANT FUNCTION ARGUMENTS AND RETURN TYPES

- The return type or arguments of a function can be defined as constant.
- Member functions should not modify member data.

```
void func(const int i)
{
   i++;    // error
}

const int funcCont()
{
   return 6;
}
```

DEFINING CLASS DATA MEMBERS AS CONSTANT

- Data members are not initialized during declaration.
- The initialization is done in the constructor.
- Once initialized cannot change the value.

Exercise:

Write a function called setValue within the class "TestConst" and try to change the variable 'var'.

Compiles correctly

```
class TestConst
{
    const int var;
    public:
    TestConst(int x):var(x)
    {
        cout << "Const Var: " <<
var<<endl;
    }
};

int main()
{
    TestConst testObj1(3);
    TestConst testObj2(2);
}
```

DEFINING CLASS OBJECT AS CONSTANTS

- During the object lifetime Data members can never be changed.

```
const class_name object;
```

CONSTANT OBJECTS

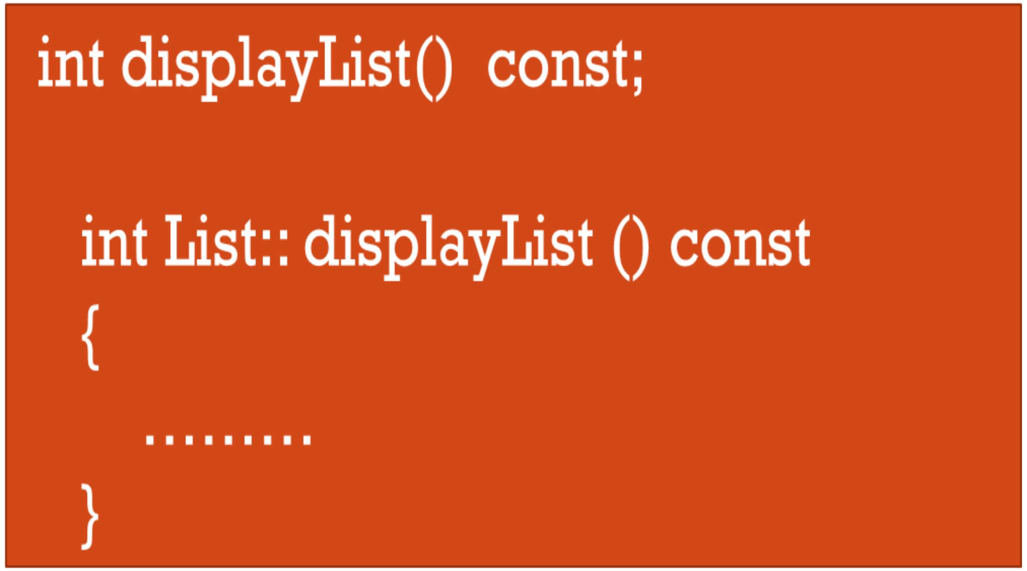
- Objects are seldom passed by value in C++.
- If you want to pass objects, they are passed by passing a **const** reference
- Passing by **const** reference
  - Avoids copying the object
  - Protects the object from unintentional changes.

```
void displayTime(const Time &t)
```

DEFINING CLASS MEMBER FUNCTION AS CONSTANTS

- **const keyword** must be added to both the function signature and implementation.
- A constant member functions never modifies data members in an object.

```
returnType functionName() const
```



```
int displayList()  const;

  int List:: displayList () const
  {
     .........
  }
```

THE "THIS" POINTER IN C++

- How do objects work with functions and data members of a class?
- When an object is created
  - Each object gets its own copy of the data member.
  - All-access the same function definition as present in the code segment.
- Each object gets its own copy of data members and share a single copy of member functions.
- How the proper data members are accessed and updated?
  - An implicit pointer 'this' (value is the address of the object that generated the call)
  - Passed as a hidden argument to all non-static member function calls and
  - Available as a local variable within the body of all non-static functions.
  - 'this' pointer is not available in static member functions as static member functions can be called without any object (with class name).

IN RECTANGLE EXAMPLE...

- this points to the object that invoked getWL().

```
void Rectangle::getWL()
{
    cout<<"Width is "<<this->width<<endl;
    cout<<"Length is "<<this->length<<endl;
}
```

HOMEWORK

Friend Functions(What, Why & How)

Forum post discussion