Constructor and Destructor Functions-Chapter 2.1 and 2.2
Teach yourself C++, Herbert Schildt

PREPARED BY: LEC TASMIAH TAMZID ANANNYA, CS DEPT, AIUB

Constructor Functions

- A constructor is a member function of a class which initializes objects of a class.
- In C++, Constructor is automatically called when object(instance of class) is created.
- It is special member function of the class.

Constructor Functions

❑A constructor is a member function of a class which initializes objects of a class.

❑In C++, Constructor is automatically called when object(instance of class) is created.

❑ It is special member function of the class.

How constructors are different from a normal member function?

❑A constructor has the same name as the class.

❑It does not have a return type.

❑A class’s constructor is called each time an object of that class is created.

Default Constructors

- **Default constructor is the constructor which doesn’t take any argument. It has no parameters**.

```
class construct
{
public:
   int a, b;

      // Default Constructor
   construct() ;
};

construct::construct()
{
      cout<<“Constructing…”<<endl;
      a = 10;
      b = 20;  }

int main()
{
    // Default constructor called automatically when the object is created
   construct c;
   cout << "a: "<< c.a << endl << "b: "<< c.b;
   return 0;
}
```

Default Constructors

- **Default constructor is the constructor which doesn’t take any argument. It has no parameters**.

```
class construct
{
public:
   int a, b;

      // Default Constructor
   construct() ;
};

construct::construct()
{
      cout<<“Constructing…”<<endl;
      a = 10;
      b = 20;  }

int main()
{
    // Default constructor called automatically when the object is created
   construct c;
   cout << "a: "<< c.a << endl << "b: "<< c.b;
   return 0;
}
```

```
Constructing…
a:10
b:20
```

Default Constructors

Notice how contruct() is defined.

```
construct::construct()
{
      cout<<“Constructing…”<<endl;
      a = 10;
      b = 20;
   }
```

It has no return type.

It is illegal for a constructor to have a return type.

Parameterized Constructors

- It is possible to pass arguments to constructors.
- Typically, these arguments help initialize an object when it is created.
- To create a parameterized constructor, simply add parameters to it the way you would to any other function.
- When you define the constructor’s body, use the parameters to initialize the object.

Parameterized Constructors

```
class Point
{
        int x, y;

    public:

        // Parameterized Constructor

        Point(int x1, int y1)
        {
            x = x1;
            y = y1;
        }

        int getX()   { return x; }

        int getY()   {return y;}

 };
```

```
int main()

{

    // Constructor called

    Point p1(10, 15);


    // Access values assigned by constructor

cout << "p1.x = " << p1.getX() << ", p1.y = " <<
p1.getY();

    return 0;

}
```

Parameterized Constructors

- ❑ When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function.

In the previous example:

```
int main()
{
   // Constructor called
   Point p1;    //error because the object does not pass any parameter to the constructor
   // Access values assigned by constructor
cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
   return 0;
}
```

Parameterized Constructors

You can also send any variables in the constructor.

```
class Point
{
   private:
      int x, y;

public:

      // Parameterized Constructor

      Point(int x1, int y1)
      {
         x = x1;
         y = y1;
       }

      int getX()        {  return x;  }

      int getY()        {   return y; }
   };
```

```
int main()
{
    int a,b;
    cin>>a>>b;
   // Constructor called
   Point p1(a, b);

   // Access values assigned by constructor
cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
   return 0;
}
```

Note:

For global objects, an object’s constructor is called once, when the prgram first begins execution.

For local objects, the constructor is called each time the declaration statement is executed.

Destructors

❑ The complement of a constructor is called destructor.

❑Destructor is a member function which destructs or deletes an object.

❑**When is destructor called?**
A destructor function is called automatically when the object goes out of scope:
(1) the function ends
(2) the program ends
(3) a block containing local variables ends

❑**How destructors are different from a normal member function?**
- ❑Destructors have same name as the class preceded by a tilde (~)
- ❑Destructors don’t take any argument and don’t return anything

Constructors and Destructors

```
class Line {
      double length;
   public:
      double getLength( );
      Line(double len);   // This is the constructor declaration
      ~Line();  // This is the destructor: declaration
};
Line::Line(double len) {
    length=len;
    cout << "Object is being created" << endl;}
Line::~Line(void) {
   cout << "Object is being deleted" << endl;}
double Line::getLength() {return length;}
```

```
int main() {
 Line line(6.0);
 cout << "Length of line : " << line.getLength();
 return 0;
}
```

Constructors and Destructors

```
class Line {
      double length;
   public:
      double getLength( );
      Line(double len);   // This is the constructor declaration
      ~Line();  // This is the destructor: declaration
};
Line::Line(double len) {
    length=len;
    cout << "Object is being created" << endl;}
Line::~Line(void) {
   cout << "Object is being deleted" << endl;}
double Line::getLength() {return length;}
```

```
int main() {
 Line line(6.0);
 cout << "Length of line : " << line.getLength();
 return 0;
}
```

Output:

```
Object is being created
Length of line : 6
Object is being deleted
```

Practice on Constructor and Destructor

❑ Create a class called **box** whose constructor function is passed three double values, length, width and height of the box. Have the **box** class compute the volume of the box and store it in another **double** variable. Include a member function of the class called **show_volume()** that will display the volume of each box.

❑ Practice all the previous examples using constructors instead of **setvalues** functions.

What happens when an object is passed to a function?

```
class student {
        int id;
        public:
         int setId (int i) {id = i;}
          int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
        student ob;
         ob.setId(50);
        show(ob);
}
```

What happens when an object is passed to a function?

```
class student {
        int id;
        public:
         int setId (int i) {id = i;}
          int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
        student ob;
         ob.setId(50);
        show(ob);
}
```

```
class student {
        int id;
        public:
         student(int i) {
          cout<<“Constructor…”<<endl;
          id=i;}
         int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
        student ob(3);
        show(ob);
}
```

What happens when an object is passed to a function?

- ❑When a copy of an object is made to be used in a function, the constructor function is not called.
- ❑Because a constructor function is generally used to initialize some aspect of an object.
- ❑Thus, it must not be called when making a copy if an already existing object passed to a function.
- ❑Doing so will alter the content of the object.
- ❑However, when the function terminates and the copy is destroyed the destructor function is called.
- ❑Because, the copy might perform some operation that must be undone when it goes out of scope.

What happens when an object is passed to a function?

```
class student {
        int id;
        public:
         int setId (int i) {id = i;}
          int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
        student ob;
         ob.setId(50);
        show(ob);
}
```

```
class student {
        int id;
        public:
         student(int i) {
          cout<<"Constructor…"<<endl;
          id=i;}
         int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
        student ob(3);
        show(ob);
}
```

What happens when an object is passed to a function?

```
class student {
          int id;
          public:
           int setId (int i) {id = i;}
            int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
          student ob;
           ob.setId(50);
          show(ob);
}
```

```
class student {
          int id;
          public:
           student(int i) {
            cout<<“Constructor…”<<endl;
            id=i;}
           int getId() {return id;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
          student ob(3);
          show(ob);
}
```

Output:

```
Constructor...
3
```

What happens when an object is passed to a function?

```
class student {
                int id;
                public:
                student(int i) {
                cout<<"Constructing.."<<endl;
                 id=i;}
                 int getId() {return id;}
                  ~student(){
                   cout<<"Destructing.."<<endl;}
};

void show(student st)
{
    cout<<st.getId()<<endl; }

int main(){
                student ob(3);
                show(ob);
}
```

What happens when an object is passed to a function?

```
class student {
        int id;
        public:
        student(int i) {
        cout<<"Constructing.."<<endl;
         id=i;}
         int getId() {return id;}
          ~student(){
           cout<<"Destructing.."<<endl;}
};

void show(student st)
{
   cout<<st.getId()<<endl; }

int main(){
        student ob(3);
        show(ob);
}
```

Output:

```
Constructing..
3
Destructing..
Destructing..
```

Overloading Constructors

❑ **It is possible to overload constructor functions.**

```
class Point
{
   int x, y;
   public:
       //overloading constructors
       Point(){x=0; y=0;}     //no initializations
       Point(int x1, int y1) //initialization
        {x=x1; y=y1;}
       int getX(){return x;}
       int getY(){return y;}
};
```

```
int main()
{
    Point p0;
    Point p1(10, 20);
    cout<<"p0.x:"<<p0.getX()<<" "<<"p0.y:"<<p0.getY()<<endl;
    cout<<"p1.x:"<<p1.getX()<<" "<<"p1.y:"<<p1.getY()<<endl;
    return 0;
}
```

Overloading Constructors

```
class date
{
   int day, month, year;
   public:
   date(char *str);
   date(int d, int m, int y);
};
date::date(char *str)
{
   cout<<str<<endl;
}
```

```
date::date(int d, int m, int y)
{
   day=d; month=m; year=y;
   cout<<day<<" "<<month<<" "<<year<<endl;
}
int main() {
   date sdate("10/10/2018");
   date idate(10, 10, 2018);
}
```

Default Arguments and Constructor

- ❑ **You can also give constructor functions default arguments.**

Default Arguments and Constructor

- ❑ **You can also give constructor functions default arguments.**

```
class Point
{
   int x, y;
   public:

       Point(int x1=0, int y1=0) //initialization
        {x=x1; y=y1;}
       int getX(){return x;}
       int getY(){return y;}
};
```

Default Arguments and Constructor

□ **You can also give constructor functions default arguments.**

```
class Point
{
   int x, y;
   public:

       Point(int x1=0, int y1=0) //initialization
        {x=x1; y=y1;}
       int getX(){return x;}
       int getY(){return y;}
};
```

```
int main()

{

    Point p0;  //declare without initialization

    Point p1(10, 20);   //declare with initial value

    cout<<"p0.x:"<<p0.getX()<<" "<<"p0.y:"<<p0.getY()<<endl;

    cout<<"p1.x:"<<p1.getX()<<" "<<"p1.y:"<<p1.getY()<<endl;

    return 0;

}
```

Default Arguments and Constructor

❑ **You can also give constructor functions default arguments.**

Use default arguments instead of overloading constructors

```
class Point
{
  int x, y;
  public:

      Point(int x1=0, int y1=0) //initialization
       {x=x1; y=y1;}
      int getX(){return x;}
      int getY(){return y;}
};
```

```
int main()

{

   Point p0;  //declare without initialization

   Point p1(10, 20);   //declare with initial value

   cout<<"p0.x:"<<p0.getX()<<" "<<"p0.y:"<<p0.getY()<<endl;

   cout<<"p1.x:"<<p1.getX()<<" "<<"p1.y:"<<p1.getY()<<endl;

   return 0;

}
```