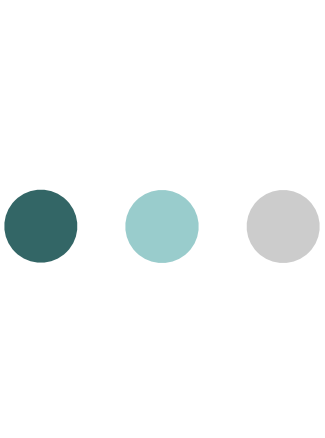
Foundation of Computer Science: Class

Kafi Rahman

Assistant Professor

Computer Science

Truman State University



13.8

Passing Arguments to Constructors

Passing Arguments to Constructors

- To create a constructor that takes arguments:
  - indicate parameters in prototype:

```
Rectangle(double, double);
```

  - Use parameters in the definition:

```
Rectangle::Rectangle(double w, double len)
{
   width = w;
   length = len;
}
```

Passing Arguments to Constructors

- Now, we can pass arguments to the constructor when we create an object:
  - `Rectangle r(10, 5);`

JIT Quiz (1 of 5)

```
// default values
void abc(int x, int y=100, int z = 50)
{
  cout<< x << " "
      << y << " "
      << z << endl;
}

// driver function
int main()
{
  // the default values will be used
  // for the other two parameters
  abc(10, 50, 20);

  return 0;
}
```

JIT Quiz (2 of 5)

```
// default values
void abc(int x = 50, int y = 100, int z)
{
  cout<< x << " "
      << y << " "
      << z << endl;
}

// driver function
int main()
{
  // the default values will be used
  // for the other two parameters
  abc(20);

  return 0;
}
```

JIT Quiz (3 of 5)

```
// default values
void abc(int x, int y = 100, int z = 50)
{
  cout<< x << " "
      << y << " "
      << z << endl;
}

// driver function
int main()
{
  // the default values will be used
  // for the other two parameters
  abc();

  return 0;
}
```

JIT Quiz (4 of 5)

```
class Movie // class of a Movie
{
private:
  // variables
  string title;
  string director;
  unsigned release_year;

  Movie(string t, string d, unsigned y)
  {
    title = t;
    director = d;
    release_year = y;
  }
public:

  // formatted string
  string to_string()
  {
    return title + "; " + director + " (" +  std::to_string(release_year) +  ")";
  }
}; // end of the class definition

int main()
{ // what about Movie myMovie; ???
  Movie entMovie ("Harry Potter", "Chris Columbus", 2001);
  cout<<entMovie.to_string();
  return 0;
}
```

JIT Quiz (5 of 5)

```
class Point
{
  private:
    int x, y;
  public:
    Point()
    {
      x = y = 0;
    }

    Point(int px, int py)
    {
      x = px;
      y = py;
    }
    string to_string()
    { return "\nx = " + std::to_string(x) + "; y = " + std::to_string(y);
    }
};


int main()
{
  Point center;
  Point left(10, 20);
  cout<<center.to_string();
  cout<<left.to_string();
  return 0;
}
```

Default Arguments

- We can specify default values for the function parameters
- The values should be provided from right to left
  - i.e., it is possible that parameter on the left will not have default value
  - the vice-versa is not supported in C++

```
// y and z have default values
void abc(int x, int y=100, int z=50)
{
  cout<< x << " "
      << y << " "
      << z << endl;
}

// driver function
int main()
{
  // the default values will be used
  // for the other two parameters
  abc(10);
  return 0;
}
```

More About Default Constructors

- If all of a constructor's parameters have default arguments, then it is a default constructor. For example:
  - `Rectangle(double = 0, double = 0);`
- Creating an object and passing no arguments will cause this constructor to execute:

  `Rectangle r;`

Classes with No Default Constructor

- When all of a class's constructors require arguments, then the class has NO default constructor.

- When this is the case, you must pass the required arguments to the constructor when creating an object.

13.9

Destructors

Destructors

- Member function automatically called when an object is destroyed
- Destructor name is ~classname, e.g., ~Rectangle
- Has no return type; takes no arguments
- Only one destructor per class, i.e., it cannot be overloaded
- If constructor allocates dynamic memory, destructor should release it

Destructors: example

```
class Student
{
  private:
    int student_ID;
    string * student_name;
  public:
    Student(int id, string * name)
    { student_ID = id;
      student_name = new string;
      *student_name = *name;
    }

    ~Student()
    {
      delete student_name;
    }

    string to_string()
    { return *student_name + " (" + std::to_string(student_ID) + ");";
    }
};
```

Destructors: example

```
class Student
{
  private:
    int student_ID;
    string * student_name;
  public:
    Student(int id, string * name)
    { student_ID = id;
      student_name = new string;
      *student_name = *name;
    }

    ~Student()
    {
      delete student_name;
    }

    string to_string()
    { return *student_name + " (" + std::to_string(student_ID) + ");";
    }
};


int main()
{
  string name = "Harry Potter";
  Student top_student (100, &name);
  cout<< top_student.to_string() << endl;
  return 0;
}
```

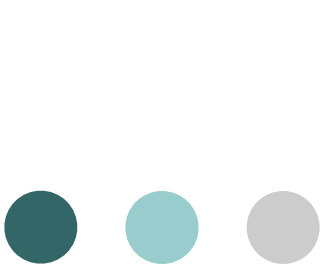
Constructors, Destructors, and Dynamically Allocated Objects

- When an object is dynamically allocated with the new operator, its constructor executes:

```
Rectangle *r = new Rectangle(10, 20);
```

- When the object is destroyed, its destructor executes:

```
delete r;
```



13.10

Overloading Constructors

Overloading Constructors

- A class can have more than one constructor
- Overloaded constructors in a class must have different parameter lists:
  - Rectangle();
  - Rectangle(double);
  - Rectangle(double, double);

Only One Default Constructor and One Destructor

- Do not provide more than one default constructor for a class: one that takes no arguments and one that has default arguments for all parameters
  - `Square();`
  - `Square(int = 0);  // will not compile`
- Since a destructor takes no arguments, there can only be one destructor for a class

Overloading Constructors

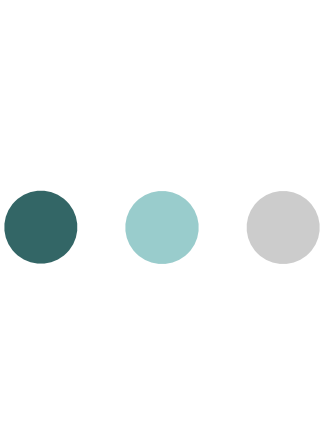
```
1  // This class has overloaded constructors.
2  #ifndef INVENTORYITEM_H
3  #define INVENTORYITEM_H
4  #include <string>
5  using namespace std;
6
7  class InventoryItem
8  {
9  private:
10    string description; // The item description
11    double cost;        // The item cost
12    int units;          // Number of units on hand
13 public:
14    // Constructor #1
15    InventoryItem()
16       { // Initialize description, cost, and units.
17         description = "";
18         cost = 0.0;
19         units = 0; }
20
21    // Constructor #2
22    InventoryItem(string desc)
23       { // Assign the value to description.
24         description = desc;
25
26         // Initialize cost and units.
27         cost = 0.0;
28         units = 0; }
```

Overloading Constructors

```
29
30      // Constructor #3
31      InventoryItem(string desc, double c, int u)
32        { // Assign values to description, cost, and units.
33          description = desc;
34          cost = c;
35          units = u; }
36
37      // Mutator functions
38      void setDescription(string d)
39         { description = d; }
40
41      void setCost(double c)
42         { cost = c; }
43
44      void setUnits(int u)
45         { units = u; }
46
47      // Accessor functions
48      string getDescription() const
49         { return description; }
50
51      double getCost() const
52         { return cost; }
53
54      int getUnits() const
55         { return units; }
56  };
57  #endif
```

Member Function Overloading

- Non-constructor member functions can also be overloaded:
  - `void setCost(double);`
  - `void setCost(double, double);`
- Must have unique parameter lists as for constructors



13.11

Using Private Member Functions

Using Private Member Functions

- A private member function can only be called by another member function
- It is used for internal processing by the class
  - the object of the class will not be able to call a private member function.

Using Private Member Functions (cont)

```
20  class Circle
21  {
22    private:
23      int x, y;
24      double r;
25      // private function
26      double square(double v)
27      {
28        return v * v;
29      }
30    public:
31      // constructor that takes three parameters
32      // also is a default constructor
33      Circle(int x_param=0, int y_param=0, double r_param=0)
34      {
35        x = x_param; y = y_param; r = r_param;
36      }
37      // calculates and regurns the area of the cirlce
38      double getArea()
39      {
40        const double PI = 3.14159f;
41        return PI * square(r);
42      }
43  };
```