Recitation 05

From C to C++ - Constructors/Destructors

George Mappouras

10/13/2017

Classes

- What is a class?
  A fancy struct…

```
class NAME{
        int var_name;
        double other_var;
        …
};
```

Classes (2)

- What is special about them?
  (1) Methods: Functions inside the class
  (2) “private”, “public” (or “protected”) declarations of variables and methods

```
class Grades {
   private:
            int g;
   public:
            void add_grade(int x){
              g+=x;
            }
};
```

Classes (2)

- What is special about them?
  (1) Methods: Functions inside the class
  (2) “private”, “public” (or “protected”) declarations of variables and methods

```
class Grades {
  private:
          int g;
  public:
          void add_grade(int x){
            g+=x;
          }
};
```

can only be accessed within the class

can be accessed from anywhere

Objects

- And what is an object?
  An instance of a class

```
int main(){
  Grades student1;
  Grades student2;
  …
  return 0;
}
```

Objects

- And what is an object?
  An instance of a class

```
int main(){
  Grades student1;
  Grades student2;
  student1.add_grade(3);     → legal
  student2.g=3;      → illegal
  return 0;
}
```

Static Decleration

```
class Grades {
   private:
                int g;
                static int num_of_students;
   public:
          int getG(){
                   return g;
                }
                Grades& add_grade(int x){
                  g+=x;
             return *this
                }
};
int Grades::num_of_students=0;
```

Static Decleration

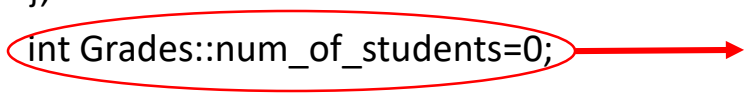
```
class Grades {
   private:
            int g;
            static int num_of_students;
   public:
         int getG(){
                return g;
              }
              Grades& add_grade(int x){
               g+=x;
          return *this
              }
};
int Grades::num_of_students=0;
```

a new box for every object instance of this class

unique box that is the same for every object instance of this class

Static Decleration

```
class Grades {
   private:
          int g;
          static int num_of_students;
   public:
          int getG(){
                return g;
          }
          Grades& add_grade(int x){
                g+=x;
                return *this
          }
};
int Grades::num_of_students=0;
```



initializing and creating the box for num_of_students

Example

```
class BankAccount {
 private:
   static unsigned long nextAccountNumber;
   unsigned long accountNumber;
   double balance;
 public:
   unsigned long getAccountNumber(){
     return accountNumber;
   }
   double showBalance(){
     return balance;
   }
 unsigned long getNextAccountNumber(){
     return nextAccountNumber;
   }
   void deposit(double amount){
     balance += amount;
   }
void initAccount(){
     accountNumber = nextAccountNumber;
     nextAccountNumber++;
   }
};
```

```
unsigned long BankAccount::nextAccountNumber = 0;

int main (){
  BankAccount account1;
  account1.initAccount();
  account1.deposit(10);
  BankAccount account2;
  account2.initAccount();
  account2.deposit(100);
  unsigned long num1 = account1.getAccountNumber();
  unsigned long numNext1 =account1.getNextAccountNumber();
  unsigned long num2 = account2.getAccountNumber();
  unsigned long numNext2 = account2.getNextAccountNumber();
  frpintf(stdout,"%lu, %lu, %lu, %lu\n", num1,num2,numNext1,numNext2);
  frpintf(stdout, "%lu, %lu\n",account1.showBalance(),
account2.showBalance()),
  return 0;
}
```

Example

```
class BankAccount {
 private:
   static unsigned long nextAccountNumber;
   unsigned long accountNumber;
   double balance;
 public:
   unsigned long getAccountNumber(){
     return accountNumber;
   }
   double showBalance(){
     return balance;
   }
 unsigned long getNextAccountNumber(){
     return nextAccountNumber;
   }
   void deposit(double amount){
     balance += amount;
   }
void initAccount(){
     accountNumber = nextAccountNumber;
     nextAccountNumber++;
   }
};
```

```
unsigned long BankAccount::nextAccountNumber = 0;

int main (){
  BankAccount account1;
  account1.initAccount();
  account1.deposit(10);
  BankAccount account2;
  account2.initAccount();
  account2.deposit(100);
  unsigned long num1 = account1.getAccountNumber();
  unsigned long numNext1 =account1.getNextAccountNumber();
  unsigned long num2 = account2.getAccountNumber();
  unsigned long numNext2 = account2.getNextAccountNumber();
  frpintf(stdout,"%lu, %lu, %lu, %lu\n", num1,num2,numNext1,numNext2);
  frpintf(stdout, "%lu, %lu\n",account1.showBalance(),
account2.showBalance()),
  return 0;
}
```

0, 1, 2, 2

10, 100

Reference vs Pointers

int y = 5; int z;

int & x = y;	≡	int * const x = &y;	Rule1: always initialize when declared
			Rule2: we can not change where it points
z = x;	≡	z = (*x);	Rule3: no need to dereference
x = z+1;	≡	(*x) = z+1;	Rule4: no reference to reference
y = x;	≡	y = (*x);	

int g(int & x); //assume we have a function that gets a reference to int and returns an int

z =g(x);

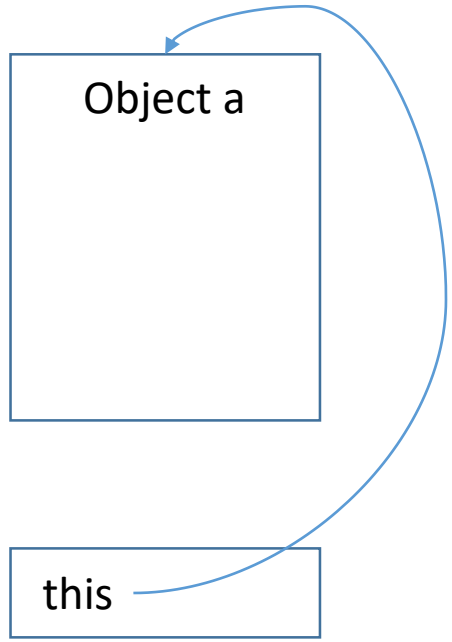
The “this” parameter and the reference “&”

```
class Grades {
   private:
          int g;
   public:
          int getG(){
             return g;
          }
          Grades& set_grade(int x){
            g=x;
            return *this;
          }
};
```

The “this” parameter and the reference “&”

```
class Grades {
   private:
          int g;
   public:
          int getG(){
             return g;
          }
          Grades& set_grade(int x){
            g=x;
            return *this;
          }
};
```

```
int main(void){
  Grades a;
  a = a.set_grade(10);

}
```



The “this” parameter and the reference “&”

```
class Grades {
   private:
          int g;
   public:
          int getG(){
             return g;
          }
          Grades& set_grade(int x){
            g=x;
            return *this;
          }
};
```

```
int main(void){
  Grades a;
  Grades b;
  a.set_grade(5);
  b = a.set_grade(2);
  fprintf(stdout, "%d,%d\n",a.getG(),b.getG());
}
```

The “this” parameter and the reference “&”

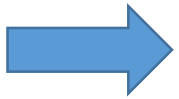
```
class Grades {
   private:
      int g;
   public:
            int getG(){
         return g;
      }
      Grades& set_grade(int x){
        g=x;
        return *this;
      }
      void add_grade(int x){
        g = x;
      }
};
```

```
int main(void){
  Grades a;
  Grades b;
  a.set_grade(5);
  b = a.set_grade(2);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());
  b.add_grade(6);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());


}
```

Exercise 1 – C to C++

C

```
typedef struct {
 int x;
 int y;
} MyClass;

int getSum (const MyClass * const this) {
 return this->x + this->y;
}
```

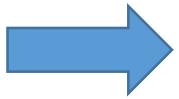


C++

Exercise 1 – C to C++

C

```c
typedef struct {
 int x;
 int y;
} MyClass;

int getSum (const MyClass * const this) {
 return this->x + this->y;
}
```



C++

```cpp
class MyClass {
 public:
  int x;
  int y;
  int getSum() const { return x + y;}
};
```

Question 14.11 : What is the output when the following code is executed?

```
1  #include <cstdio>
2  #include <cstdlib>
3
4  void f(int & y, int * z) {
5    printf("y = %d, *z = %d\n", y, *z);
6    y += *z;
7    *z = 42;
8  }
9
10 int main(void) {
11   int a = 3;
12   int b = 4;
13   int & x = a;
14   x = b;
15   printf("a = %d, b = %d, x = %d\n", a, b, x);
16   f(b, &x);
17   printf("a = %d, b = %d, x = %d\n", a, b, x);
18   return EXIT_SUCCESS;
19 }
```

Question 14.11 - Answer

- a = 4, b = 4, x = 4
- y = 4, *z = 4
- a = 42, b = 8, x = 42

Question 14.8

```
1  #include <cstdio>
2  #include <cstdlib>
3
4  class Point {
5  private:
6    int x;
7    int y;
8  public:
9    void setLocation(int newX, int newY) {
10     x = newX;
11     y = newY;
12   }
13   int getX() const {
14     return x;
15   }
16   int getY() const {
17     return y;
18   }
19 };
20 void printPoint(const char * name, const Point & p) {
21   printf("%s: (%d,%d)\n", name, p.getX(), p.getY());
22 }
```

```
23 void f(Point & p) {
24   printPoint("p", p);
25   p.setLocation(p.getX() + 2, p.getY() - 1);
26 }
27
28 int main(void) {
29   Point p1;
30   Point p2;
31   p1.setLocation(2,4);
32   p2.setLocation(3,5);
33   f(p1);
34   f(p2);
35   printPoint("p1", p1);
36   printPoint("p2", p2);
37   return EXIT_SUCCESS;
38 }
```

Question 14.8 - Answer

```
p: (2,4)
p: (3,5)
p1: (4,3)
p2: (5,4)
```

Function Overloading

```
class Vector{
  public:
    int a[2];
    void setVector (int  num){
      int i;
      for(i=0;i<2;i++){
        a[i] = num;
      }
    }
    void setVector (int  num1, int num2){
      a[0]=num1;
      a[1]=num2;
    }
};
```

```
int main(void){
  Vector a;
  Vector b;
  a.setVector(5);
  b.setVector(1,3);
  fprintf(stdout,"%d,%d\n",b.a[0],b.a[1]);
  fprintf(stdout,"%d,%d\n",a.a[0],a.a[1]);
}
```

Operator Overloading

```
class Vector{
  public:
    int a[2];

    void operator+=(Vector & b){
      int i;
      for(i=0;i<2;i++){
        a[i]+=b.a[i];
      }
    }
};
```

```
int main(void){
  Vector a;
  Vector b;
  a.a[0]=1;
  a.a[1]=4;
  b.a[0]=3;
  b.a[1]=2;
  a+=b;
  fprintf(stdout,"%d,%d\n",a.a[0],a.a[1]);
}
```

Constructors

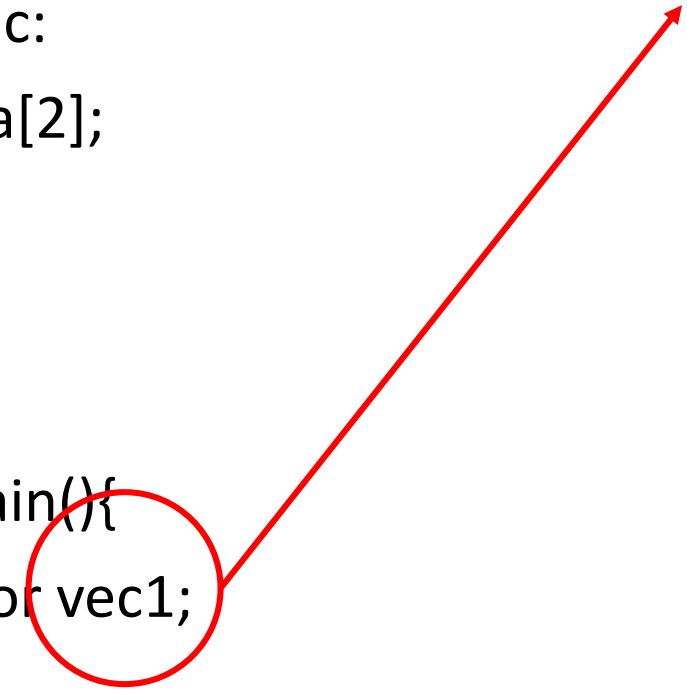
```
class Vector{
  public:
    int a[2];
…
};

int main(){
  Vector vec1;
{
```

Constructors

```
class Vector{
  public:
    int a[2];
…
};

int main(){
  Vector vec1;
{
```

Constructors

```
class Vector{
  public:
    int a[2];
…
};


int main(){
  Vector vec1;
{
```



How is this object initialized?

- We need a Constructor:
  1. If **none is defined** a **simple default constructor** is used.
  2. If **no copy constructor** is defined a **default copy constructor exists** and can be used
  3. If we **define any** constructor **the simple default constructor cannot be used**
  4. If we **define a copy** constructor the **default copy constructor cannot be used**

Constructors and Overloading

```
class Vector{
  public:
    int a[2];
Vector(){
        a[0]=0; a[1]=0;
    }
};
```

Constructors and Overloading

```
class Vector{
  public:
    int a[2];
Vector(){
        a[0]=0; a[1]=0;
    }
};
```

```
class Vector{
  public:
    int a[2];
    Vector(){
        a[0]=0; a[1]=0;
    }
    Vector(int x, int y){
        a[0]=x; a[1]=y;
    }

};
```

Constructors and Overloading

```
class Vector{
  public:
    int a[2];
Vector(){
        a[0]=0; a[1]=0;
    }
};
```

```
class Vector{
  public:
    int a[2];
    Vector(){
        a[0]=0; a[1]=0;
    }
    Vector(int x, int y){
        a[0]=x; a[1]=y;
    }

};
```

```
class Vector{
  public:
    int a[2];
    Vector(){
        a[0]=0; a[1]=0;
    }
    Vector(int x, int y){
        a[0]=x; a[1]=y;
    }
    Vector (Vector & p){
        a[0] = p.a[0];
        a[1] = p.a[1];
    }
};
```

Constructors and Overloading

```
class Vector{
  public:
    int a[2];
Vector(){
        a[0]=0; a[1]=0;
    }
};
```

```
int main(){
  Vector myVect(2,3);
   Vector newVect(myVect);
}
```

```
class Vector{
  public:
    int a[2];
    Vector(){
        a[0]=0; a[1]=0;
    }
    Vector(int x, int y){
        a[0]=x; a[1]=y;
    }

};
```

```
class Vector{
  public:
    int a[2];
    Vector(){
        a[0]=0; a[1]=0;
    }
    Vector(int x, int y){
        a[0]=x; a[1]=y;
    }
    Vector (Vector & p){
        a[0] = p.a[0];
        a[1] = p.a[1];
    }
};
```

Constructors – Order of initialization

```
class Vector{
  public:
    int a;
    int b;
    Vector(int x, int y): a(x), b(y) {}
};
```

Constructors – Order of initialization

```
class Vector{
  public:
    int a;
    int b;
    Vector(int x, int y): a(x), b(y) {}
};
```

Find the error below:

```
Vector vec1;
Vector vec2(2,3);
```

Constructors – Order of initialization

```
class Vector{
  public:
    int a;
    int b;
    Vector(int x, int y): a(x), b(y) {}
};
```

Find the error below:

```
Vector vec1;
Vector vec2(2,3);
```

Constructors - Example

```
class Vector{
  public:
    int a;
    int b;
    void setVector (int  num1, int num2){
      a=num1;
      b=num2;
    }
    Vector(int num1, int num2):a(num1),
b(num2){}
    Vector(int num1){
      a=num1; b=num1;
      fprintf(stdout,"I am constructing a vector
object\n");
    }
};
```

```
int main(void){
  Vector a(3);
  Vector b(3,2);
  b.setVector(1,3);
  fprintf(stdout,"%d,%d\n",b.a,b.b);
  fprintf(stdout,"%d,%d\n",a.a,a.b);
}
```

Constructors - Example

```
class Vector{
  public:
    int a;
    int b;
    void setVector (int  num1, int num2){
      a=num1;
      b=num2;
    }
    Vector(int num1, int num2):a(num1),
b(num2){}
    Vector(int num1){
      a=num1; b=num1;
      fprintf(stdout,"I am constructing a vector
object\n");
    }
};
```

```
int main(void){
  Vector a(3);
  Vector b(3,2);
  b.setVector(1,3);
  fprintf(stdout,"%d,%d\n",b.a,b.b);
  fprintf(stdout,"%d,%d\n",a.a,a.b);
}
```

```
I am constructing a vector object
1,3
3,3
```

Destructors

```
class Vector{
  public:
    int a;
    int b;
    Vector(int x, int y): a(x), y(b) {}
    ~Vector() {
        fprintf (stdout, “Destroyed vector with values: %d, %d\n”,a,b);
     }
};
```

Destructors

Destructors are invoked:

1) When we explicitly delete an object (equivalent to free)

2) An object goes out of scope

3) We reach the end of our program (which is pretty much the same as point 2)

Why are they important:

If we dynamically allocate memory inside an object we need to explicitly delete that memory with our destructor

Example from your book (new, new[], delete, delete[])

```
class Point{
   int x;
   int y;
};
class Polygon{
   Point *points;
   size_t numPoints;
public:
    Polygon(size_t n): points(new Point[n]),numPoints(n) {}
   ~Polygon(){
        delete[] points;
    }
};
```

Deep Copy Constructor for Polygon

```
class Polygon{
   Point *points;
   size_t numPoints;
public:
    Polygon(const Polygon & p): points(?),  numPoints(?) {
            ?
     }
}
```

Deep Copy Constructor for Polygon

```
class Polygon{
   Point *points;
   size_t numPoints;
public:
    Polygon(const Polygon & p): points(new Points[p.numPoints]), numPoints(p.numPoints) {
            for (size_t i=0; i<numPoints; i++)
                    points[i] = p.points[i];
    }
}
```

Example

```
class Grades {
   private:
      int g;
   public:
      int getG(){
         return g;
      }
      Grades set_grade(int x){
        g=x;
        return *this;
      }
      void add_grade(int x){
        g = x;
      }
      Grades(const Grades & grade): g(grade.g) {fprintf(stdout,"Constructing %d\n", g);}
      Grades(int num):g(num){fprintf(stdout,"Constructing %d\n", g);}
      ~Grades(){fprintf(stdout,"Deleteting %d\n",g);}
};
```

```
int main(void){
  Grades a(1);
  Grades b(1);
  a.set_grade(5);
  b = a.set_grade(2);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());
  b.add_grade(6);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());

}
```

Example

```
class Grades {
   private:
      int g;
   public:
      int getG(){
        return g;
      }
      Grades set_grade(int x){
       g=x;
       return *this;
      }
      void add_grade(int x){
       g = x;
      }
      Grades(const Grades & grade): g(grade.g) {fprintf(stdout,"Constructing %d\n", g);}
      Grades(int num):g(num){fprintf(stdout,"Constructing %d\n", g);}
      ~Grades(){fprintf(stdout,"Deleteting %d\n",g);}
};
```

```
int main(void){
  Grades a(1);
  Grades b(1);
  a.set_grade(5);
  b = a.set_grade(2);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());
  b.add_grade(6);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());

}
```

Example

```
class Grades {
   private:
      int g;
   public:
      int getG(){
         return g;
      }
      Grades set_grade(int x){
        g=x;
        return *this;
      }
      void add_grade(int x){
        g = x;
      }
      Grades(const Grades & grade): g(grade.g) {fprintf(stdout,"Constructing %d\n", g);}
      Grades(int num):g(num){fprintf(stdout,"Constructing %d\n", g);}
      ~Grades(){fprintf(stdout,"Deleteting %d\n",g);}
};
```

```
int main(void){
  Grades a(1);
  Grades b(1);
  a.set_grade(5);
  b = a.set_grade(2);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());
  b.add_grade(6);
  fprintf(stdout,"%d,%d\n",a.getG(),b.getG());

}
```

Example - Answer

Constructing 1
Constructing 1
Constructing 5
Deleteting 5
Constructing 2
Deleteting 2
2,2
2,6
Deleteting 6
Deleteting 2

- A `+=` operator, which takes a `const Point &`, and increases this Points's x by the passed in Point's x, and this Point's y by the passed in Point's y. It should then return a reference to this object.
- A `==` operator, which takes a `const Point &`, and determines if it has the same coordinates as this Point.
- A `*=` operator which takes an int and scales (multiplies) this Point's x and y by the passed in integer. This operator should return a reference to this Point.

```
class Point {
private:
  int x;
  int y;
public:
  void setLocation(int newX, int newY) {
    x = newX;
    y = newY;
  }
  int getX() const {
    return x;
  }
  int getY() const {
    return y;
  }
};
```

```
class Point {
private:
  int x;
  int y;
public:
  void setLocation(int newX, int newY) {
    x = newX;
    y = newY;
  }
  int getX() const {
    return x;
  }
  int getY() const {
    return y;
  }
  Point & operator+=(const Point & rhs) {
    x += rhs.x;
    y += rhs.y;
    return *this;
  }
  bool operator==(const Point & rhs) const {
    return x == rhs.x && y == rhs.y;
  }
  Point & operator*=(int scale) {
    x *= scale;
    y *= scale;
    return *this;
  }
};
```