Parameter passing

```
struct student {
   string name;
   picture pic;
   bool printed; // print flag
};
```

Function definition

```
bool print_student1(student s) {
   if (!s.printed)
     cout << s.name << endl;
   return true;
}
```

Example of use

```
student a;
… // initialize a
bool b = print_student1(a);
```

What happens when
we run code like this?

Parameter passing

```
struct student {
   string name;
   picture pic;
   bool printed; // print flag
};
```

Function definition

```
void print_student2(student s) {
   if (!  s.printed) {
     cout <<  s.name << endl;
     s.printed = true;
   }
}
```

Example of use

```
student *b;
… // initialize b
print_student2(b);
cout <<  b.printed << endl;
```

Parameter passing

```
struct student {
   string name;
   picture pic;
   bool printed; // print flag
};
```

Function definition

```
void print_student3(student s) {
   if (!  s.printed) {
     cout <<   s.name << endl;
     s.printed = true;
   }
}
```

Example of use

```
student c;
… // initialize c
print_student3(c);
cout << c.printed << endl;
```

Return values

```
struct student {
   string name;
   picture pic;
   bool printed; // print flag
};
```

Function definition

```
bool print_student1(student s) {
   if (!s.printed)
     cout << s.name << endl;

   return true;
}
```

Example of use

```
student a;
bool b = print_student1(a);
…
```

What happens when
we run code like this?

Return by _____ or _____ or _____

Returns

```
struct student {
   string name;
   picture pic;
   bool printed; // print flag
};
```

Function definition

```
student * print_student4(student &s) {
   student w = s;
   if (!w.printed) {
     cout << w.name << endl;
     s.printed = w.printed = true;
   }
   return &w;
}
```

Example of use

```
student c;
student * d;
// initialize c
d = print_student4(c);
```

Returns

```
struct student {
   string name;
   picture pic;
   bool printed; // print flag
};
```

Function definition

```
student & print_student5(student &s) {
   student w = s;
   if (!w.printed) {
     cout << w.name << endl;
     s.printed = w.printed = true;
   }
   return w;
}
```

Example of use

```
student c, d;
// initialize c
d = print_student5(c);
```

Lesson: don't return 1) a pointer to a local variable nor 2) a local variable by reference.

Constructor (다시 보기)

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

class sphere {
public:
  sphere();
  sphere(double r);
  void setRadius(double newR);
  double getDiameter() const;
  …
private:
  double theRadius;
  int numAtts;
  string * atts;
};
```

```
…
// default constructor
sphere::sphere() {
  theRadius = 1.8;
  numAtts = 3;
  atts = new string[numAtts];

  atts[0].assign(“red”);
  // atts[0] = “red”;
  …
}
```

What do you want the object to look like when you declare it?

```
sphere a;
```

Copy constructor - utility

```
class sphere {
public:
  sphere();
  sphere(double r);
  void setRadius(double newR);
  double getDiameter() const;
  …
private:
  double theRadius;
  int numAtts;
  string * atts;
};
```

Use 1

```
sphere myFun(sphere s) {
  … // play with s
  return s;
}

int main() {
   sphere a, b;
   // initialize a
   b = myFun(a);
   return 0;
}
```

Use 2

```
int main() {



}
```

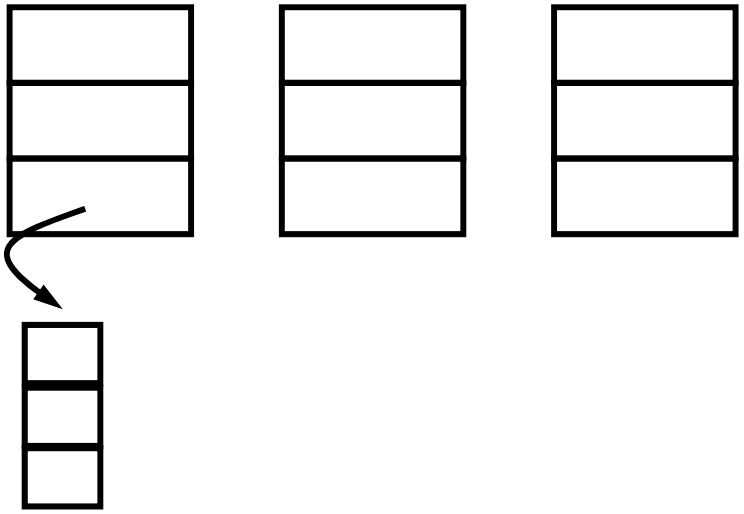
Copy constructor

```
class sphere {
public:
  sphere();
  sphere(double r);
  sphere(const sphere & o
  void setRadius(double n
  double getDiameter() co
  …
private:
  double theRadius;
  int numAtts;
  string * atts;
};
```

```
…
// copy constructor
sphere::sphere(const sphere &orig) {




}
```



Copy constructor discussion

```
class sphere {
public:
  sphere();
  sphere(double r);
  sphere(const sphere & orig);
  void setRadius(double newR);
  double getDiameter() const;
  …
private:
  double theRadius;
  int numAtts;
  string * atts;
};
```

1. Why is the cctor's param pbr?
2. What does it mean that the cctor's param is const?
3. Why did we need to write a custom cctor?