CS 225

#5: Parameters

January 26, 2018 · *Wade Fagen-Ulmschneider*

Heap Memory – Allocating Arrays

heap-puzzle3.cpp

```
5  int *x;
6  int size = 3;
7
8  x = new int[size];
9
10 for (int i = 0; i < size; i++) {
11   x[i] = i + 3;
12 }
13
14 delete[] x;
```

*: **new[]** and **delete[]** are identical to **new** and **delete**, except the constructor/destructor are called on each object in the array.

Memory and Function Calls

Suppose we want to join two Spheres together:

joinSpheres-byValue.cpp

```
11 /*
12  * Creates a new sphere that contains the exact volume
13  * of the sum of volume of two input spheres.
14  */
15 Sphere joinSpheres(Sphere s1, Sphere s2) {
16   double totalVolume = s1.getVolume() + s2.getVolume();
17
18   double newRadius = std::pow(
19     (3.0 * totalVolume) / (4.0 * 3.141592654),
20     1.0/3.0
21   );
22
23   Sphere result(newRadius);
24
25   return result;
26 }
```

By default, arguments are "passed by value" to a function. This means that:

-
-

Alterative #1: Pass by Reference

joinSpheres-byReference.cpp

```
15 Sphere joinSpheres(Sphere & s1, Sphere & s2) {
16   double totalVolume = s1.getVolume() + s2.getVolume();
17
18   double newRadius = std::pow(
19     (3.0 * totalVolume) / (4.0 * 3.141592654),
20     1.0/3.0
21   );
22
23   Sphere result(newRadius);
24
25   return result;
26 }
```

Alternative #2: Pass by Pointer

joinSpheres-byPointer.cpp

```
15 Sphere joinSpheres(Sphere * s1, Sphere * s2) {
16   double totalVolume = s1->getVolume() + s2->getVolume();
17
18   double newRadius = std::pow(
19     (3.0 * totalVolume) / (4.0 * 3.141592654),
20     1.0/3.0
21   );
22
23   Sphere result(newRadius);
24
25   return result;
26 }
```

	By Value	By Pointer	By Reference
Exactly what is copied when the function is invoked?	entire object may be large	1 pointer 8 bytes	Nothing
Does modification of the passed in object modify the caller's object?	no	yes	yes
Is there always a valid object passed in to the function?	yes	no	yes
Speed	dependent on data maybe slow	fast & constant	fast
Safety			

Using the `const` keyword

1. Using `const` in function parameters:

make a promise not modify the given variable

joinSpheres-byValue-const.cpp

```
15  Sphere joinSpheres(const Sphere  s1, const Sphere  s2)
15  Sphere joinSpheres(const Sphere *s1, const Sphere *s2)
15  Sphere joinSpheres(const Sphere &s1, const Sphere &s2)
```

Best Practice: *"All parameters passed by reference must be labeled const."* – *Google C++ Style Guide*

2. Using `const` as part of a member functions' declaration:

make a promise not modify the state of the class

sphere-const.h

```
 5  class Sphere {
 6    public:
 7      Sphere();
 8      Sphere(double r);
 9
10      double getRadius();
11      double getVolume();
12
13      void setRadius(double r);
14
 …      // ...
```

const

sphere-const.cpp

```
 …
15  double Sphere::getRadius()              {
16    return r_;
17  }
18
19  double Sphere::getVolume()              {
20    return (4 * 3.14 * r_ * r_ * r_) / 3.0;
21  }
 …
```

const

Returning from a function

Identical to passing into a function, we also have three choices on how memory is used when returning from a function:

Return by value:

```
15  Sphere  joinSpheres(const Sphere &s1, const Sphere &s2)
```

Return by reference:

```
15  Sphere &joinSpheres(const Sphere &s1, const Sphere &s2)
```

…remember: never return a reference to stack memory!

Return by pointer:

```
15  Sphere *joinSpheres(const Sphere &s1, const Sphere &s2)
```

…remember: never return a reference to stack memory!

Copy Constructor

When a non-primitive variable is passed/returned **by value,** a copy must be made. As with a constructor, an automatic copy constructor is provided for you if you choose not to define one:

All **copy constructors** will:

The **automatic copy constructor**:

1.

2.

To define a **custom copy constructor**:

sphere.h

```
 5  class Sphere {
 6    public:
 7      Sphere(const Sphere & other);   // custom copy ctor
```

CS 225 – Things To Be Doing:

1. Theory Exam #1 begins Tuesday – ensure you're registered!
2. lab_debug due Sunday (11:59pm)
3. mp1 due Monday (11:59pm)
4. Complete POTDs