Heap Memory – Allocating Arrays

```cpp
int *x;
int size = 3;
x = new int[size];
for (int i = 0; i < size; i++) {
x[i] = i + 3;
}
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:

```cpp
/*
 * Creates a new sphere that contains the exact volume
 * of the sum of volume of two input spheres.
 */
Sphere joinSpheres(Sphere s1, Sphere s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    Sphere result(newRadius);
    return result;
}
```

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

- Exactly what is copied when the function is invoked?
- Does modification of the passed in object modify the caller’s object?
- Is there always a valid object passed in to the function?
- Speed
- Safety

### Alternative #1: Pass by Reference

```cpp
Sphere joinSpheres(Sphere & s1, Sphere & s2) {
    double totalVolume = s1.getVolume() + s2.getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    Sphere result(newRadius);
    return result;
}
```

### Alternative #2: Pass by Pointer

```cpp
Sphere joinSpheres(Sphere * s1, Sphere * s2) {
    double totalVolume = s1->getVolume() + s2->getVolume();
    double newRadius = std::pow((3.0 * totalVolume) / (4.0 * 3.141592654), 1.0/3.0);
    Sphere result(newRadius);
    return result;
}
```
Using the const keyword

1. Using const in function parameters:

```cpp
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:

```cpp
sphere-const.h
5 class Sphere {
6   public:
7       Sphere();
8       Sphere(double r);
9       double getRadius();
10      double getVolume();
11     void setRadius(double r);
12     ...
13       // ...
```

```cpp
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 }
...```