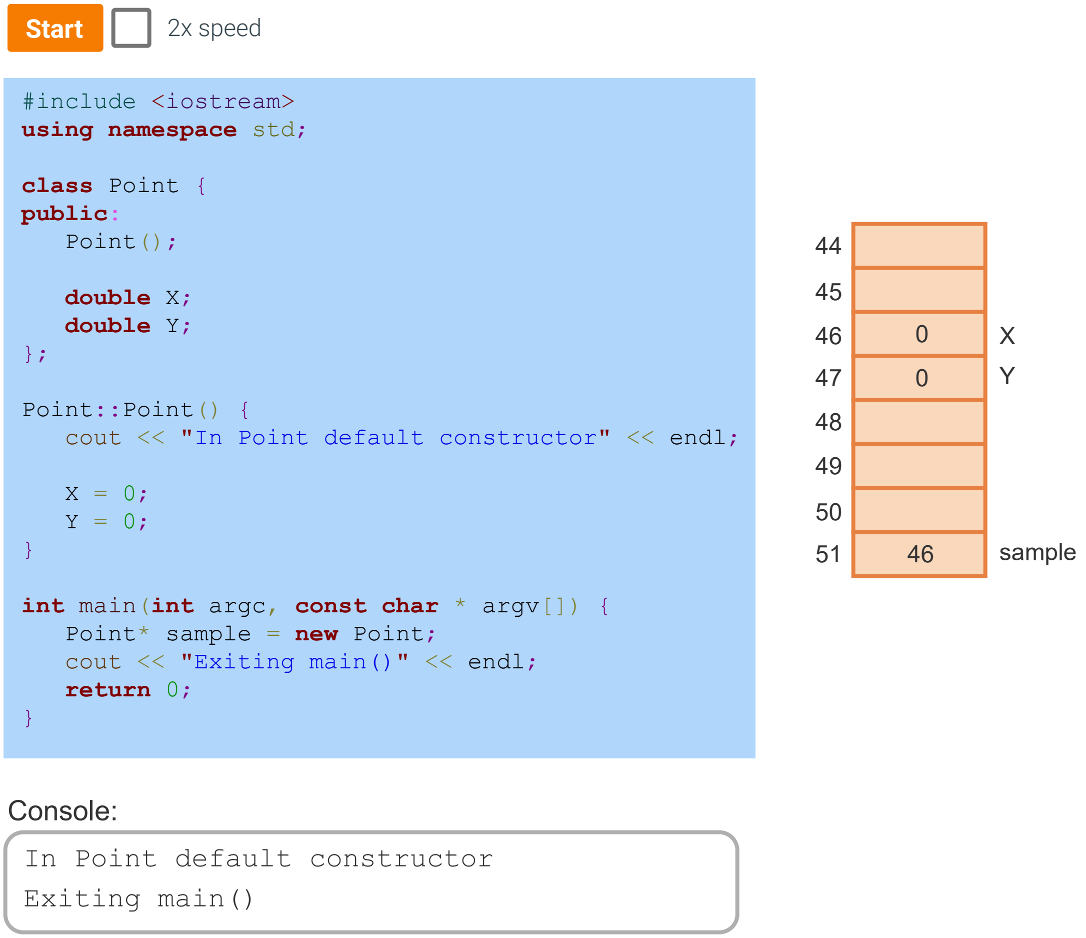
8.3 Operators: new, delete, and ->

The new operator

The ***new operator*** allocates memory for the given type and returns a pointer to the allocated memory. If the type is a class, the new operator calls the class's constructor after allocating memory for the class's member variables.

PARTICIPATION ACTIVITY | 8.3.1: The new operator allocates space for an object, then calls the constructor.



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

class Point {
public:
   Point();

   double X;
   double Y;
};

Point::Point() {
   cout << "In Point default constructor" << endl;

   X = 0;
   Y = 0;
}

int main(int argc, const char * argv[]) {
   Point* sample = new Point;
   cout << "Exiting main()" << endl;
   return 0;
}
```

Console:

```
In Point default constructor
Exiting main()
```

Feedback?

PARTICIPATION ACTIVITY | 8.3.2: The new operator.

1) The new operator returns an int. ✓

○ True

○ False

2) When used with a class type, the new operator allocates memory after calling the class's constructor. ✓

○ True

○ False

3) The new operator allocates, but does not deallocate, memory. ✓

○ True

○ False

Feedback?

Constructor arguments

The new operator can pass arguments to the constructor. The arguments must be in parentheses following the class name.

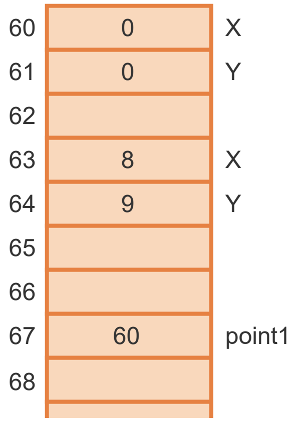
PARTICIPATION ACTIVITY | 8.3.3: Constructor arguments. ✓

Start □ 2x speed

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

class Point {
public:
   Point(double xValue = 0, double yValue = 0);
   void Print();

   double X;
   double Y;
};

Point:: Point(double xValue, double yValue) {
   X = xValue;
   Y = yValue;
}

void Point::Print() {
   cout << "(" << X << ", ";
   cout << Y << ")" << endl;
}
```

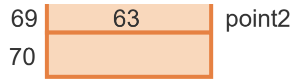
Console:

```
(0, 0)
(8, 9)
```



60	0	X
61	0	Y
62		
63	8	X
64	9	Y
65		
66		
67	60	point1
68		

```
int main() {
   Point* point1 = new Point;
   (*point1).Print();

   Point* point2 = new Point(8, 9);
   (*point2).Print();

   return 0;
}
```



Feedback?

PARTICIPATION ACTIVITY | 8.3.4: Constructor arguments.

```
Point* point = new Point();
```

```
Point* point = new Point(10);
```

```
Point* point = new Point(0, 10);
```

```
Point* point = new Point(0, 0, 0);
```

Constructs the point (0, 0).

Constructs the point (10, 0).

Constructs the point (0, 10).

Causes a compiler error.

Reset

Feedback?

The member access operator

When using a pointer to an object, the ***member access operator*** (**->**) allows access to the object's members with the syntax `a->b` instead of `(*a).b`. Ex: If myPoint is a pointer to a Point object, `myPoint->Print()` calls the `Print()` member function.

Table 8.3.1: Using the member access operator.

Action	Syntax with dereferencing	Syntax with member access operator
Display point1's Y member value with cout	`cout << (*point1).Y;`	`cout << point1->Y;`
Call point2's Print() member function	`(*point2).Print();`	`point2->Print();`

Feedback?

PARTICIPATION ACTIVITY 8.3.5: The member access operator. ✓

1) Which statement calls point1's Print() member function? ✓

```
Point point1(20, 30);
```

○ `(*point1).Print();`

○ `point1->Print();`

○ `point1.Print();`

2) Which statement calls point2's Print() member function? ✓

```
Point* point2 = new Point(16, 8);
```

○ `point2.Print();`

○ `point2->Print();`

3) Which statement is *not* valid for multiplying point3's X and Y members? ✓

```
Point* point3 = new Point(100, 50);
```

○ `point3->X * point3->Y`

○ `point3->X * (*point3).Y`

○ `point3->X (*point3).Y`

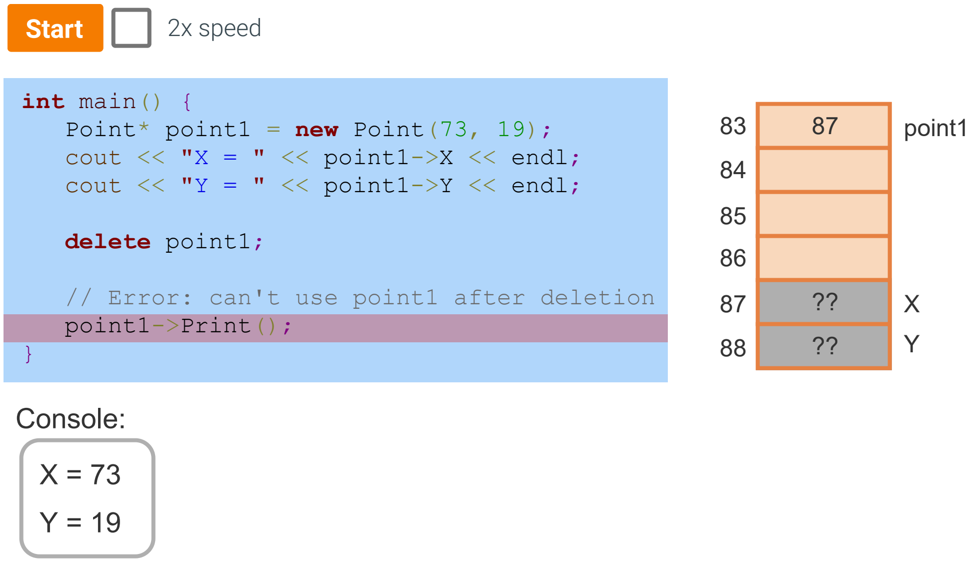
Feedback?

The delete operator

The ***delete operator*** deallocates (or frees) a block of memory that was allocated with the new operator. The statement `delete pointerVariable;` deallocates a memory block pointed to by pointerVariable. If pointerVariable is null, delete has no effect.

After the delete, the program should not attempt to dereference pointerVariable since pointerVariable points to a memory location that is no longer allocated for use by pointerVariable. Dereferencing a pointer whose memory has been deallocated is a common error and may cause strange program behavior that is difficult to debug. Ex: If pointerVariable points to deallocated memory that is later allocated to someVariable, changing *pointerVariable will mysteriously change someVariable. Calling delete with a pointer that wasn't previously set by the new operator has undefined behavior and is a logic error.

PARTICIPATION ACTIVITY 8.3.6: The delete operator.



```
int main() {
   Point* point1 = new Point(73, 19);
   cout << "X = " << point1->X << endl;
   cout << "Y = " << point1->Y << endl;

   delete point1;

   // Error: can't use point1 after deletion
   point1->Print();
}
```

83	87	point1
84		
85		
86		
87	??	X
88	??	Y

Console:

```
X = 73
Y = 19
```

Feedback?

PARTICIPATION ACTIVITY 8.3.7: The delete operator.

1) The delete operator can be used on any pointer.

○ True

○

○ False

2) The statement `delete point1;` throws an exception if point1 is null. ✓

○ True

○ False

3) After the statement `delete point1;` executes, point1 will be null. ✓

○ True

○ False

Feedback?

Allocating and deleting object arrays

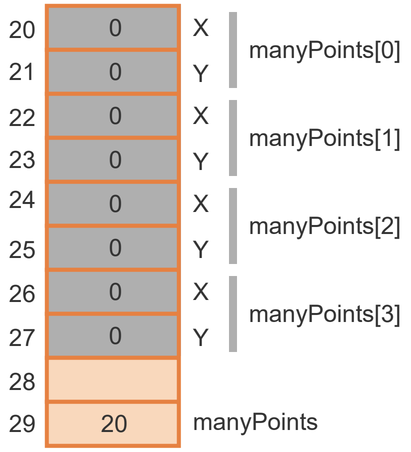
The new operator creates a dynamically allocated array of objects if the class name is followed by square brackets containing the array's length. A single, contiguous chunk of memory is allocated for the array, then the default constructor is called for each object in the array. A compiler error occurs if the class does not have a constructor that can take 0 arguments.

The ***delete[] operator*** is used to free an array allocated with the new operator.

PARTICIPATION ACTIVITY | 8.3.8: Allocating and deleting an array of Point objects. ✓

Start □ 2x speed

```
int main() {
   // Allocate points
   int pointCount = 4;
   Point* manyPoints = new Point[pointCount];

   // Display each point
   for (int i = 0; i < pointCount; ++i)
      manyPoints[i].Print();

   // Free all points with one delete
   delete[] manyPoints;

   return 0;
}
```



20	0	X	manyPoints[0]
21	0	Y	
22	0	X	manyPoints[1]
23	0	Y	
24	0	X	manyPoints[2]
25	0	Y	
26	0	X	manyPoints[3]
27	0	Y	
28			
29	20	manyPoints	

Console:

```
(0, 0)
(0, 0)
(0, 0)
(0, 0)
```

Feedback?

PARTICIPATION ACTIVITY | 8.3.9: Allocating and deleting object arrays. ✓

1) The array of points from the example above _____ contiguous in memory. ✓

○ might or might not be

○ is always

2) What code properly frees the dynamically allocated array below? ✓

```
Airplane* airplanes = new
Airplane[10];
```

○ `delete airplanes;`

○ `delete[] airplanes;`

○
```
for (int i = 0; i < 10;
++i) {
   delete airplanes[i];
}
```

3) The statement below only works if the Dalmatian class has _____. ✓

```
Dalmatian* dogs = new Dalmatian[101];
```

○ no member functions

○ only numerical member variables

○ a constructor that can take 0 arguments

Feedback?

CHALLENGE ACTIVITY | 8.3.1: Operators: new, delete, and ->. ✓

Start

Type the program's output.

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

class Car {
   public:
      Car(int distanceToSet);
   private:
      int distanceTraveled;
};

Car::Car(int distanceToSet) {
   distanceTraveled = distanceToSet;
   cout << "Traveled: " << distanceTraveled << endl;
}

int main() {
   Car* myCar1 = nullptr;
   Car* myCar2 = nullptr;

   myCar1 = new Car(75);
   myCar2 = new Car(85);

   return 0;
}
```

1 2

Check Next

Feedback?

CHALLENGE ACTIVITY 8.3.2: Deallocating memory

Deallocate memory for kitchenPaint using the delete operator.

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

class PaintContainer {
   public:
      ~PaintContainer();
      double gallonPaint;
};

PaintContainer::~PaintContainer() { // Covered in section on Destructors.
   cout << "PaintContainer deallocated." << endl;
}

int main() {
   PaintContainer* kitchenPaint;

   kitchenPaint = new PaintContainer;
   kitchenPaint->gallonPaint = 26.3;

   /* Your solution goes here  */

```

Run

View your last submission

Feedback?

Exploring further:

- operator new[] Reference Page from cplusplus.com
- More on operator new[] from msdn.microsoft.com
- operator delete[] Reference Page from cplusplus.com
- More on delete operator from msdn.microsoft.com
- More on -> operator from msdn.microsoft.com