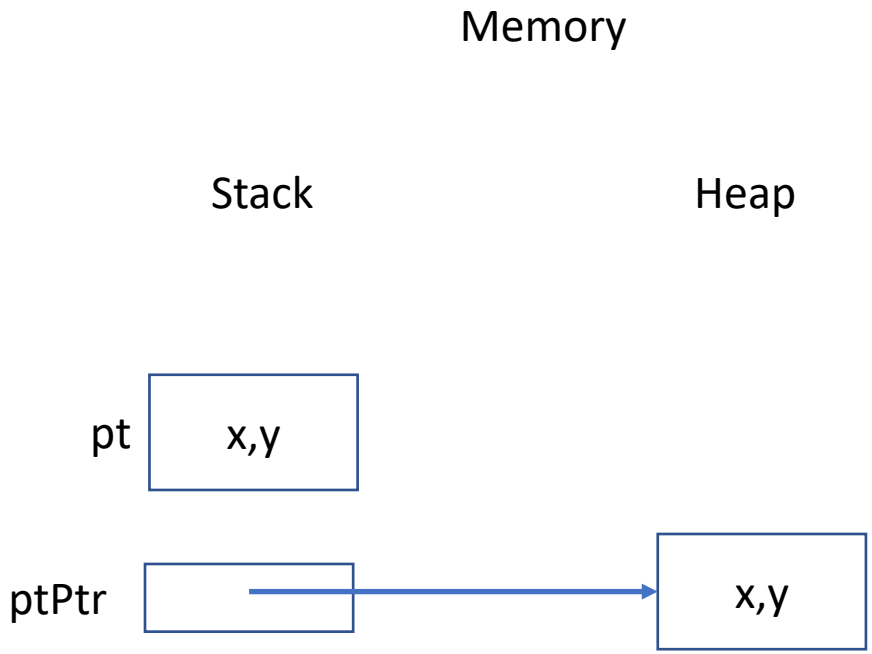
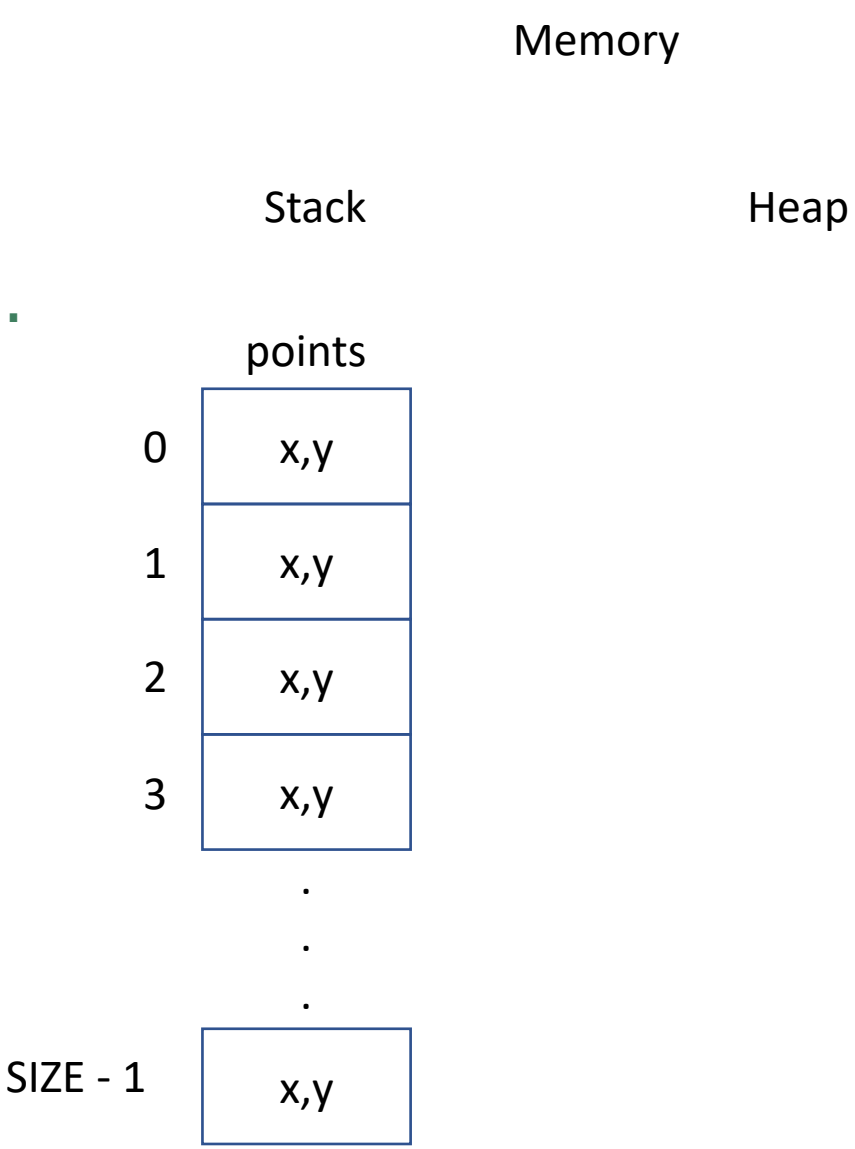
```
const int SIZE = 10;

class Point {
public:
        static const int MAX = 100;
        int x;
        int y;

        Point(){
                //rand() produces a number between 0 and RAND_MAX inclusive
                x = rand() % MAX;
                y = rand() % MAX;
        };

        string toString() const{
                stringstream out;
                out << "( " << x << ", " << y << " )";
                return out.str();
        }
};
```

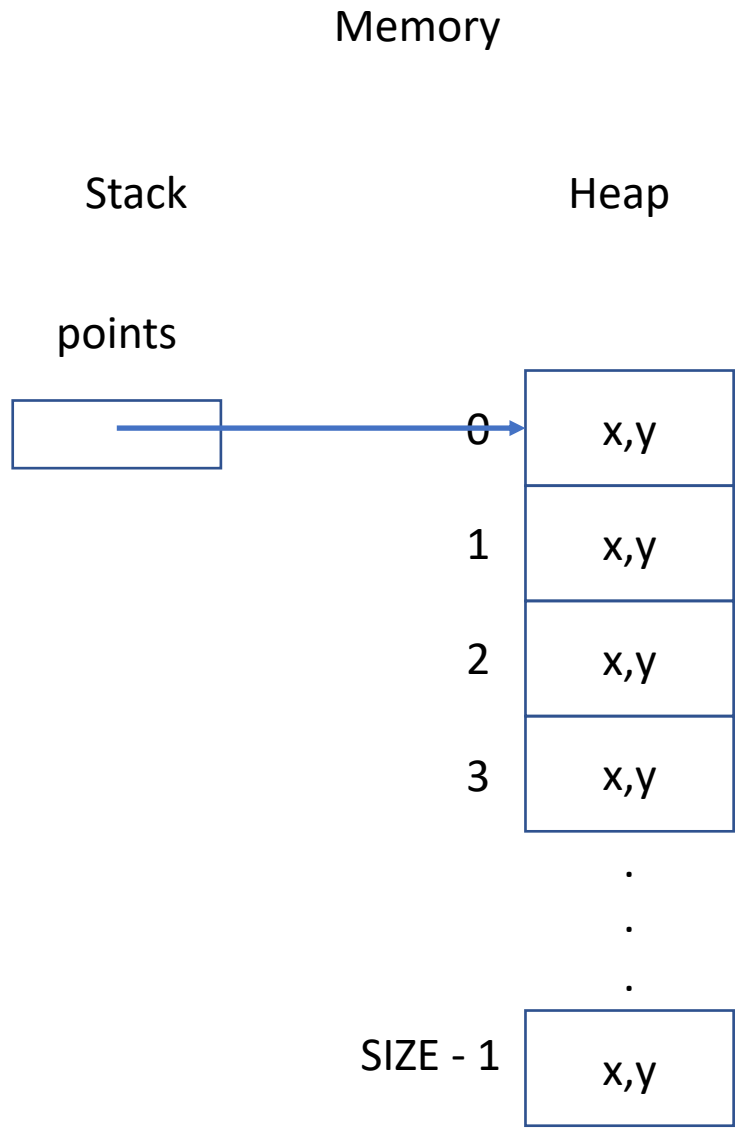
```
//Create two objects
// The first object is on the stack.
// The second object is on the heap.
void objectTest(){
        cout << "objectTest()" << endl;

        //create a point on the stack
        Point pt;
        cout << pt.toString() << endl;
        cout << "x = " << pt.x << endl;


        //create a point on the heap
        Point *ptPtr = new Point();
        cout << ptPtr->toString() << endl;
        cout << "x = " << ptPtr->x << endl;
        //deallocate
        delete ptPtr;
}
```



```
//Create and print an array of points:
// This array is static, it is ALL on the stack
// It will no longer be useful after the method returns.
void staticArrayTest(){
        cout << "staticArrayTest()" << endl;
        Point points[SIZE];

        for(int i = 0; i < SIZE; i++){
                cout << i << ": "
                     << points[i].toString()
                     << endl;
        }
        cout << endl;
}
```



```
// Create and print an array of points:
// In this case the array is on the heap and is enough room for
// the objects to be lined up contiguously in memory.
// Each object exists and has values.
// The entire array must be released when the
// program is done with it.
void dynamicArrayTest(){
        cout << "dynamicArrayTest()" << endl;
        //allocate an array of objects on the heap
        Point *points = new Point[SIZE];

        for(int i = 0; i < SIZE; i++){
                cout << i
                        << ": "
                        << points[i].toString()
                        << endl;
        }

        //free up the array
        delete[] points;
        cout << endl;
}
```



```
// Create and print an array of points:
// This is an array of pointers to objects.
// Each object is allocated separately, so must be freed separately.
// This is most closely models Java arrays of Objects
void dynamicArrayOfPointerTest(){
        cout << "dynamicArrayOfPointerTest()" << endl;

        //allocate an array of pointers to
        //objects on the heap
        Point **points = new Point*[SIZE];

        for(int i = 0; i < SIZE; i++){
                //create a new object
                points[i] = new Point();
        }

        for(int i = 0; i < SIZE; i++){
                // since the array contains pointers,
                // we use -> to access members
                cout << i << ": " << points[i]->toString() << endl;
        }


        for(int i = 0; i < SIZE; i++){
                //free object
                delete points[i];
        }
        //free up the array
        delete[] points;
        cout << endl;
}
```