ForNextDay(17)
Stephen Cole
3553803

point2d.c

```
#include<math.h>
#include<stdio.h>

#include"point2d.h"

void setPoint(Point2D* point, const double x, const double y)
{
	point->x = x;
	point->y = y;
}

Point2D* mallocPoint2D()
{
	return (Point2D)malloc(sizeof(Point2D));
}

void freePoint2D(Point2D* pThis)
{
	free(pThis->x);
	free(pThis->y);
	free(pThis);
}

Point2D* copyPoint2D(Point2D* pThis)
{
	Point2D copy = mallocPoint2D();
	copy.x = pThis->x;
	copy.y = pThis->y;

	return &copy;
}

double getXPoint2D(Point2D* pThis)
{
	return pThis->x;
}

double getYPoint2D(Point2D* pThis)
{
	return pThis->y;
```

```
}

void print(const Point2D* point)
{
	printf("(%f,%f)\n", point->x, point->y);
}

double getDistancePoint2D(const Point2D* point1, const Point2D* point2)
{
 const double x2 = pow(point2->x - point1->x, 2);
 const double y2 = pow(point2->y - point1->y, 2);
 return sqrt(x2 + y2);
}

int main(void)
{
	Point2D point;
	setPoint(&point, 1.0, 1.0);
	print(&point);
	Point2D point2;
	setPoint(&point2, 2.0, 2.0);
	print(&point2);
	printf("distance %f\n", distance(&point, &point2));
	return 0;
}
```

point2d.h

```
#ifndef POINT_2D_H
#define POINT_2D_H
typedef struct Point2D {
	double x;
	double y;
} Point2D;

Point2D* mallocPoint2D();
void freePoint2D(Point2D* pThis);
Point2D* createPoint2D(double x, double y);
void setPoint2D(Point2D* pPt, double x, double y);
Point2D* copyPoint2D(Point2D* pThis);
double getXPoint2D(Point2D* pThis);
double getYPoint2D(Point2D* pThis);
void setPoint(Point2D* point, const double x, const double y);
void print(const Point2D* point);
double getDistancePoint2D(const Point2D* point1, const Point2D* point2);
```

#endif

Using structs and creating modules makes it much easier to manage your data, grouping all related data together into one container. As well as allowing for type specific functions to be created that will make code easier to understand and read. Grouping your data together into a struct can also reduce the amount of parameters needed to be passed into a function further reducing complexity.