C or C++: STRUCT a.k.a. Records

An ARRAY is an aggregate data structure that contains a sequence of data items, all with the same base datatype (e.g., int arr[10];)

A RECORD (called a struct in C/C++) is an aggregate data structure that contains a collection of data items, each with it's own field name and data type. A struct is useful because it simplifies keeping several related pieces of information located together in a single data object.

```
struct employeeType
{
    string  firstName;
    string  lastName;
    string  address1;
    string  address2;
    double  salary;
    string  deptID;
};
```

Note semicolon ;
struct employeeType is a data type definition.
It is not a variable declaration.
No memory is allocated.

You can create variables of type employeeType:

```
employeeType   fred;
```

The value of fred is the collection of field values.
Use the '.' dot operator to reference specific fields.

```
fred.firstname = "Fred";
fred.lastName  = "Barker";
fred.address1  = "123 Fourth Street";
fred.address2  = "Seattle, WA 98119";
fred.salary    = 38000.00;
fred.deptID    = "Sales";
```

Struct values can be assigned:

```
employeeType   myFriend;
myFriend  =  fred;            // copies each field
```

Arrays of structs or structs containing arrays are allowed:

```
employeeType  roster[25];
roster[0]  =  fred;
roster[0].salary = roster[0].salary + 1000.00;
```

Structs can be passed as parameters or returned as a function value:

```
void printEmployeeName(const employeeType &emp) // by reference
{
     cout << "Employee name: "
          << emp.firstname << " " << emp.lastname << endl;
}
```

C++ "CLASS" construct for defining an "object"

```
class Point {
public:              // Allow following members to be referenced
                     //   directly by objects of type Point
   int  x;
   int  y;
};


int main()
{
     Point   topLeft, bottomRight;

     // Legal because x and y are a public members of Point
     topLeft.x = 100;
     topLeft.y = 200;

     bottomRight = topLeft;

}
```

C or C++: STRUCT construct is supported in all versions of C and C++

Both struct and class declarations are allowed in C++.

```
     struct Point {
        int  x;
        int  y;
     };
```

// Access visibility alternatives: public, private, protected

In a **CLASS**, all members are **assumed 'private'** unless explicitly made 'public' or 'protected'.

In a **STRUCT**, all members are **assumed 'public'** unless explicitly made 'private' or 'protected'.

A **struct** is usually used simply as **data aggregate construct** without any constructors or other function methods.

Constructor Initializations

Class definitions should use constructor functions ("methods") to automate (and guarantee) initialization of the object data members

Class definition using constructor functions defined inline within the class declaration:

```
class Point {
public:              // Allow following members to be referenced
                     //   directly by objects of type Point

   int  m_x, m_y;

   // Constructor functions ...  [InLine]: Overloaded prototypes

   Point()                                     // Default constructor
     {
        m_x = 0;
        m_y = 0;
     }

   Point(const int xarg, const int yarg)       // Convert constructor
     {
        m_x = xarg;
        m_y = yarg;
     }
};
```

```
int main()
{

     Point   topLeft;                     // Uses default  constructor
     Point   bottomRight(100, 200);       // Uses convert  constructor

     topLeft = bottomRight;               // Assignment is valid

     topLeft.m_x = 150;
     topLeft.m_y = 400;

}
```

Private data members and access methods
Control "visibility" of internal details
Require main() "client" to use access methods

```
class Point {
public:
   Point()                                     // Default constructor
   {
      m_x = 0;                                 // Constructors are
      m_y = 0;                                 // almost always public
   }

   Point(const int xarg, const int yarg)       // Convert constructor
   {
      m_x = xarg;
      m_y = yarg;
   }
                                               // Public access methods
   int  getx() const                           // "getters" & "setters"
   {                                           //   for some private
      return(m_x);                             //   member variables
   }

   int  gety() const       { return(m_y); }    // one-line style is
   void setx(const int x)  { m_x = x;     }    // common for inline
   void sety(const int y)  { m_y = y;     }    // access methods

private:                                       // private data members
   int  m_x, m_y;
};
```

```
int main()           // Main program: "client" using Point objects
{
     Point   topLeft;
     Point   bottomRight(100, 200);

//   topLeft.m_x = 150;        // NO -- restricted, m_x is private

     topLeft.setx(150);        // client must use access methods
     topLeft.sety(400);

     cout << "BottomRight is positioned at ";
     cout << bottomRight.getx();
     cout << ", ";
     cout << bottomRight.gety();
     cout << endl;
}
```

Common project configuration: multiple source code files

Separate class code into a ".h" interface file and a ".cpp" implementation file for the methods

```
// point.h
// Header Definition and Interface File

class Point {
public:
   // Constructor method prototypes...
   Point();
   Point(const int xarg, const int yarg);

   // Access method prototypes
   int getx() const;
   int gety() const;

   void setx(const int x);
   void sety(const int y);

private:
   int  m_x, m_y;
};
```

point.h

Separate "header file" In a multi-file project.

Defines the Interface to the class.

```
// main.cpp

#include <iostream>
using namespace std;

#include "point.h"              // Clients include the header file
                                // Specifies the object "interface"
int main()
{
     Point   topLeft;
     Point   bottomRight(100, 200);

//   topLeft.m_x = 150;         // NO -- restricted,
                                // m_x is private
     topLeft.setx(150);
     topLeft.sety(400);

     cout << "BottomRight is positioned at ";
     cout << bottomRight.getx();
     cout << ", ";
     cout << bottomRight.gety();
     cout << endl;
}
```

"Client"

Uses public Interface to compute with variables and values (objects) of the class.

Separate file in a multi-file project.

```
// point.cpp                 -- use separate file for implementation
// ADT Implementation file

#include "point.h"      // Implementation file includes header file

//   Point ::        Scope Resolution operator
//                   Use prefix for all method definitions

Point::Point()                        // Default constructor
{
   m_x = 0;
   m_y = 0;
}

// Alternative declaration form using "header initialization"
//
// Point :: Point()
//     :  m_x(0), m_y(0)
// {
//
// }

// Convert constructor
Point::Point(const int xarg, const int yarg)
{
   m_x = xarg;
   m_y = yarg;
}

// Other method implementation, including the Access methods
int Point::getx() const
{
   return(m_x);
}

int Point::gety() const
{
   return(m_y);
}

void Point::setx(const int x)
{
   m_x = x;
}

void Point::sety(const int y)
{
   m_y = y;
}
```

point.cpp

Separate class Implementation file in a multi-file project.

Refine ADT, provide additional functionality
such as ability to move a point or print point coordinates

```
#include <iostream>
using namespace std;
#include "point.h"

int main()
{
     Point   topLeft;
     Point   bottomRight(100, 200);

     cout << "BottomRight is positioned at ";
     bottomRight.print();
     cout << endl;

     topLeft.move( 50, -50);       // move to nearby location

     topLeft.move( bottomRight );  // move to new location

     cout << "TopLeft is now at ";
     topLeft.print();
     cout << endl;
}
```

```
// point.h
// Header Definition and Interface File

class Point {
public:
   // Constructor functions prototypes...
   Point();
   Point(const int xarg, const int yarg);

   // Access methods
   int getx() const;
   int gety() const;

   void setx(const int x);
   void sety(const int y);
                                    // more public methods
   void print() const;              // Output coordinates to cout

   void move(const Point &d);       // Move point to new location
   void move(const int deltaX, const int deltaY);

private:
   int  m_x, m_y;
};
```

```
// point.cpp
// ADT Implementation file

#include <iostream>
using namespace std;
#include "point.h"


Point::Point() : m_x(0), m_y(0)
     { }

Point::Point(const int xarg, const int yarg) : m_x(xarg), m_y(yarg)
     { }

int Point::getx() const  { return(m_x); }
int Point::gety() const  { return(m_y); }

void Point::setx(const int x)  { m_x = x; }
void Point::sety(const int y)  { m_y = y; }

// -------------------------------------------------------

void Point::print() const       // output coordinates
{
   cout << '[' << m_x << ", " << m_y << ']' ;
}

                        // Move point to new location nearby
void Point::move(const int deltaX, const int deltaY)
{
   m_x = m_x + deltaX;
   m_y += deltaY;
}

void Point::move(const Point &d)   // Move to new location
{
   m_x = m_x + d.getx();      // Either use method
   m_y = m_y + d.m_y;         //  or access members directly
}
```