Classes and Data Abstraction: struct

There are many instances in programming where we need more than one variable in order to represent an object. For example, to represent yourself, you might want to store your name, your birthday, your height, your weight, or any other number of characteristics about yourself. You could do so like this:

```
string myName;
int myBirthYear;
int myBirthMonth;
int myBirthDay;
int myHeightInches;
int myWeightPounds;
```

However, you now have *6 independent variables* that are not grouped in any way. If you wanted to pass information about yourself to a function, you'd have to pass each variable individually. Furthermore, if you wanted to store information about someone else, you'd have to declare 6 more variables for each additional person! As you can see, this can quickly get out of control.

Fortunately, C++ allows us to create our own user-defined aggregate data types. An **aggregate data type** is a data type that groups multiple individual variables together. One of the simplest aggregate data types is the struct. A **struct** (short for structure) *allows us to group variables of mixed data types together into a single unit.*

- **Declaring and defining structs**

Because structs are user-defined, we first have to tell the compiler what our struct looks like before we can begin using it. To do this, we declare our struct using the *struct* keyword. Here is an example of a struct declaration:

```
struct Employee
{
   short id;
   int age;
   double salary;
};
```

This tells the compiler that we are defining a struct named Employee. The Employee struct contains 3 variables inside of it: a short named id, an int named age, and a double named wage. These variables that are part of the struct are called **members** (or fields). Keep in mind that Employee is just a declaration -- even though we are telling the compiler that the struct will have member variables, no memory is allocated at this time. By convention, struct names start with a capital letter to distinguish them from variable names.

In order to use the Employee struct, we simply declare a variable of type Employee:

```
Employee mohamed; // struct Employee is capitalized, variable mohamed  is not
```

This defines a variable of type Employee named mohamed. As with normal variables, defining a struct variable allocates memory for that variable.

It is possible to define multiple variables of the same struct type:

```
Employee mohamed; // create an Employee struct for Mohamed
Employee salah; // create an Employee struct for Salah
```

- ## Accessing struct members

When we define a variable such as `Employee mohamed`, Mohamed refers to the entire struct (which contains the member variables). In order to access the individual members, we use the **member selection operator** (which is a period). Here is an example of using the member selection operator to initialize each member variable:

```
Employee mohamed; // create an Employee struct for Mohamed
mohamed.id = 14; // assign a value to member id within struct Mohamed
mohamed.age = 32; // assign a value to member age within struct Mohamed
mohamed.salary = 2415.7; // assign a value to member salary within struct Mohamed

Employee salah; // create an Employee struct for Salah
salah.id = 15; // assign a value to member id within struct Salah
salah.age = 28; // assign a value to member age within struct Salah
salah.salary = 1827; // assign a value to member salary within struct Salah
```

***struct** member variables act just like normal variables, so it is possible to do normal operations on them:*

```
int totalAge = mohamed.age + salah.age;

if (mohamed.salary >salah .salary)
   cout << "Mohamed makes more than Salah\n";
else if (mohamed.salary < salah.salary)
   cout << "Mohamed makes less than Salah\n";
else
   cout << "Mohamed and Salah make the same Salary\n";

// Salah got a promotion
salah.salary += 250;

// Today is Mohamed's birthday
++mohamed.age; // use pre-increment to increment mohamed's age by 1
```

- ## Structs and functions

A big advantage of using structs over individual variables is that we can pass the entire struct to a function that needs to work with the members:

<u>Example 1:</u>

Write a C++ program to create an Employee using *structure* and function to print Information to give name, id, age, and Salary for *5* employees:

Answer:

```
#include <iostream>
using namespace std;
struct Employee
{
    string name;
    short id;
    int age;
    double salary;
};

void printInformation(Employee employee)
{
    cout << "Name:  "<<employee.name<< "\n";
    cout << "ID:   " << employee.id << "  \n";
    cout << "Age:  " << employee.age << " years\n";
    cout << "Salary: " << employee.salary << "Saudi riyals.\n";
}

int main()
{
for(int i=1;i<=5;i++)
{
  Employee employee;
    cout << "Name of   "<<i<< "   employee:";
    cin>>employee.name;
    cout << "\nID of    " <<i<< "   employee:";
    cin>>employee.id;
    cout << "\nAge of   "<< i << "  employee:";
    cin>> employee.age ;
    cout << "\nSalary of : "<<i <<"  employee:";
    cin>> employee.salary;
    printInformation(employee);
     cout << "\n";
}

    return 0;
}
```

Nested structs

Structs can contain other structs.

```
struct Employee
{
    short id;
    int age;
    double salary;
};

struct Company
{
    Employee CEO; // Employee is a struct within the Company struct
    int numberOfEmployees;
};

Company myCompany;
```

In this case, if we wanted to know what the CEO's salary was, we simply use the member selection operator twice: `myCompany.CEO.salary;`

This selects the CEO member from myCompany, and then selects the wage member from within CEO.

Example 2:

Write a program on C++ to create a StudentRecord using *structure* and give student id and grades. Create two objects (StudentRecord and Get_Data) and give the two id and grades.

Answer:

```
#include <iostream>
using namespace std;
struct StudentRecord
{               int id;
                char grade;
};
StudentRecord Get_Data (StudentRecord  in_student);
int main ()
{
        StudentRecord          Student1;
        Student1 = Get_Data (Student1);
        cout<< Student1.id<< ","<<Student1.grade<< endl;
        return 0;
}
StudentRecord Get_Data (StudentRecord in_student)
{
        cout<<"Enter ID: ";
        cin>> in_student.id;
        cout<<"Enter Grade: ";
        cin>> in_student.grade;
        return  (in_student);
}
```

Example 3:

Write a program on C++ to create a Point using *structure* and give abscissa and ordinate of a point. Create two objects and give the abscissa and ordinate of a point.

Answer:

```
#include<iostream>

using namespace std;

struct Point

{

    double x;

    double y;

};

Point read_point()

{

    Point p;

    cout<<"Enter x=";

    cin>>p.x;

    cout<<"Enter y=";

    cin>>p.y;

    return p;

}

void print_point(Point p) // function for print of type void with argument an object

{

    cout<<"(x,y)=("<<p.x<<","<<p.y<<")\n";

}

int main()

{

Point p;

p=read_point();
print_point(p);

    return 0;

}
```

Example 4:

Write a program on C++ to create a Point using *structure* and give abscissa and ordinate of a point. Create two objects and give the abscissa and ordinate of a point and a function to compute the distance between two points in the space.

Answer:

```
#include<iostream>

#include<cmath>

using namespace std;

struct Point

{

   double x;

   double y;

};

Point read_point()

{

   Point p;

   cout<<"Enter x=";

   cin>>p.x;

   cout<<"Enter y=";

   cin>>p.y;

   return p;

}

void print_point(Point p)

{

   cout<<"(x,y)=("<<p.x<<","<<p.y<<")\n";

}
```

```
double distance(Point p1,Point p2) // function distance of type double and arguments two objects

{

   return sqrt(pow((p1.x-p2.x),2)+pow((p1.y-p2.y),2));

}

int main()

{

   Point a1,a2;

a1=read_point();

a2=read_point();

cout<<"The distance between\n";

print_point(a1);

cout<<"and\n";

print_point(a2);

cout<<"is:"<<distance(a1,a2);

   return 0;

}
```

Example5:

Write a program on C++ to create a Student using *structure* and give id, age and grade. Create two objects (Student and Student read_student) and two functions: one print_student and the second pass_check of type bool.

Answer:

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

struct Student{
   int id;
   int age;
   double grade;
};


struct Student read_student() {
   struct Student s;
   cout << "Enter ID:";
   cin >> s.id;
   cout << "Enter age:";
   cin >> s.age;
   cout << "Enter grade:";
   cin >> s.grade;
   return s;
}

void print_student(struct Student s) {
   cout << "Student ID :" << s.id << endl;
   cout << "Age:" << s.age << endl;
   cout << "Grade:" << s.grade << endl;
}

bool pass_check(struct Student s) {
   return s.grade >= 60;
}

int main() {
   struct Student s;
   do {
      s = read_student();
   } while( pass_check( s ) );
   cout << "Student failed" << endl;
   return 0;
}
```

Example5:

Write a program on C++ to create a Point, Rectangle, Rectangle_four_corners and Point read_point using *structure* with fonctions area and circumference to read rectangle and print it.
Answer:

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

struct Point {
  double x;
  double y;
};

struct Rectangle {
  struct Point topLeft;
  double length;
  double width;
};

struct Rectangle_four_corners {
  struct Point topLeft, topRight, bottomLeft, bottomRight;
};

struct Rectangle_four_corners four_corners(struct Rectangle r) {
  struct Rectangle_four_corners r2;
  r2.topLeft = r.topLeft;
  r2.topRight.x = r.topLeft.x + r.length;
  r2.topRight.y = r.topLeft.y;

  r2.bottomLeft.x = r.topLeft.x;
  r2.bottomLeft.y = r.topLeft.y - r.width;

  r2.bottomRight.x = r.topLeft.x + r.length;
  r2.bottomRight.y = r.topLeft.y - r.width;
  return r2;
};

struct Point read_point() {
  struct Point p;
  cout << "Enter x coordinate:";
  cin >> p.x;
  cout << "Enter y coordinate:";
  cin >> p.y;
  return p;
}

double area(struct Rectangle r) {
  return r.length * r.width;
}

double circumference(struct Rectangle r) {
  return 2* (r.length + r.width);
}

struct Rectangle read_rectangle() {
  struct Rectangle r;
  r.topLeft = read_point();
  cout << "Enter the length:";
  cin >> r.length;
  cout << "Enter the width:";
  cin >> r.width;
  return r;
}
void print_point(struct Point p) {
  cout << "(x,y) =" << "( " << p.x << " , " << p.y << " )" << endl;
}
void print_rectangle(struct Rectangle r) {
  struct Rectangle_four_corners r4c = four_corners(r);

  cout << "Top Left point is:" ;
  print_point(r.topLeft);
  cout << "Length :" << r.length << endl;
  cout << "Width :" << r.width << endl;
  cout << "Top right point is :";
  print_point(r4c.topRight);
  cout << "Bottom left:";
  print_point(r4c.bottomLeft);
  cout << "Bottom right:";
  print_point(r4c.bottomRight);

}
int main() {
  struct Rectangle r;

  r = read_rectangle();
  print_rectangle(r);
  return 0;
}
```