Record (struct type)

- A structured data type is a type in which each value is a collection of component items.
  - the entire collection has a single name
  - each component can be accessed individually

SYNTAX

```
struct  TypeName          // does not  allocate memory
{
        DataType   MemberName ;
        DataType   MemberName ;
        …

};
```

Often we have related information of various types that we'd like to store together for convenient access under the same identifier, for example . . .

```
struct  StudentType                    // declares a  struct data type
{                                      //  does not allocate memory
     int            id ;
     string         name ;
     int             age ;
     float           gpa;
} ;
StudentType    studentA ;              // declare  variables of StudentType
StudentType    studentB ;
```

The struct declaration names a type and names the members of the struct.
It does not allocate memory for any variables of that type!
You need to declare struct type variables, that is when the memory space is allocated (for those variables)

- **Access members of a struct variable**

Dot (period) is the member selection operator.
After the struct type declaration, the various members can be used in your program only when they are preceded by a struct variable name and a dot.

Valid operations on a struct member depend only on its type

```
studentA.age  =  18;
studentA.id    =  2081;
cin  >>  studentA.gpa;
getline (cin, studentA.name);
studentA.age++;
```

Examples of **aggregate** assignment operator with struct type

```
studentB   =   studentA ;          // assignment
```

- **Functions with struct variable**

Example 1

```
DisplayInfo(studentA);                  // function activation: value parameter

void  DisplayInfo( StudentType   oneStudent)
{
        cout << "Here is the information for " << oneStudent.name << endl;
        cout   <<  setw(10) << "ID :  "  <<  oneStudent.id << endl ;
```

```
        cout  <<  setw(10) << "Age :  "  <<  oneStudent.age << endl ;
        cout  <<  setw(10) << "GPA :  "  <<  oneStudent.gpa << endl ;
}
```

Example 2:

```
IncrementAge(studentA);      // function activation: reference parameter

void  IncrementAge (StudentType &  oneStudent )
{
        oneStudent.age++ ;
}
```

Example 3:

```
GetStudentData( studentB );          // function call, better than value returning function in this case

void  GetStudentData ( StudentType & oneStudent)
{
        cout << "Enter the name of the student : " ;
        getline(cin, oneStudent.name);

        cout << "Enter the age of the student: ";
        cin >> oneStudent.age;

        cout << "Enter the id of the student: ";
        cin >> oneStudent.id;

        cout << "Enter the gpa of the student: ";
        cin >> oneStudent.gpa;
}
```

- **Hierarchical Structures**

The type of a struct member can be another struct type. This is called nested or hierarchical structures. Hierarchical structures are very useful when there is much detailed information in each record.

```
struct  DateType
{
        int    month ;          //  Assume 1 . . 12

        int    day ;            //  Assume  1 . . 31

        int    year ;
};
```

```
struct  StudentType  // declares a  struct data type
{                          //  does not allocate memory
    int          id ;
    string       name ;
    int          age ;
    float        gpa;
    DateType     birthday;
} ;
```

```
StudentType    studentC;          // declare  variables of StudentType
```

- **Internal representation of a variable of hierarchical struct type**
- **Access member of a hierarchical structure (right association)**

```
studentC.birthday.day=21;
studentC.birthday.month=4;
studentC.birthday.year=2002;
```

Array of struct type data

const int MAX_EMPLOYEE = 100;
employee allEmployee[MAX_EMPLOYEE];
what will the internal representation of allEmployee look like?

- **member access** for one array element: read in name of the 2nd employee
  getline(cin, allEmployee[1].name);

- print the 1st two characters of the 3rd employee
  cout << allEmployee[2].name[0] << allEmployee[2].name[1] << endl;

- **array iteration** : count the total number of dependents of all employee
  int sum=0;
  for (int i=0; numOfEmployee; i++)
  sum += allEmployee[i].numOfDependents;

- **pass array of struct to function**
  declare : find FindHighestRate(int numOfEmployee, employeeType allEmployee []);
  activation: FindHighestRate(numOfEmployee, allEmployee);
  definition:

```
float  FindHighestRate(employee allEmployee[])
{
    float highest = allEmployee[0].rate;
    int     employeeId = allEmployee[0].id;

    for (int i=1; i<numOfEmployee; i++)
    {
            if (allEmployee[i].rate > highest)
            {
                    highest = allEmployee[i].rate;
                    employeeId = allEmployee[i].id;
            }
    }

    cout << "Employee: " << employeeId << " has the highest pay rate : "
            << highest << endl;
}
```

Example program for Array of Struct