# Data structures

Arrays are used to store sequences of data having the same data type. But in many occasions we need to store a set of data elements having different data types under one name. The data structure used to store such kind of data is called data structure. Each element of a data structure can have different data type and different lengths.

**Declaration:**

```
struct StructureName {
        memberType1  member_1;
        memberType2  member_2;
        .
        .
};
```

or  using typedef

```
typedef struct {
        memberType1  member_1;
        memberType2  member_2;
        .
        .
} StructureName;
```

where StructureName is the name of the structure type and within the {} is a list of data members  (data types and their names) which is called <u>fields</u>.

We can initialize a data structure object at declaration:

## Using the data structure

The declaration of a data structure creates a new data type that can be used as any other built in data type.  For example,

```
struct student {
        char name[20];
        double gpa;
        int id;
};

student s1, s2;
```

To access the fields of the structure we use the dot "." operator. Note that the left hand side of the "." operator must be an object (not an address).

*<u>Initialization</u>*:

Similar to arrays, we can initialize a data structure using the {} when the variable is declared.
student s = {"ali", 2.3, 123};

*Assignment:*

_We can assign data to a structure by assigning field by field

```
strcpy(s1.name, "hassan");
s1.avg  = 2.3;
s1.id = 123;
```

note that we can't assign arrays using "=" operator:

```
s1.name = "hassan" ; // error
```

Read data from input stream to a structure

```
cin >> s1.name ;
cin >> s1.avg;
cin >> s1.id;
```

It is valid to assign a structure to another structure even if it include arrays as fields

```
s2  = s1; // correct even it includes an array as a field

cout <<  s2.name  << endl;
cout <<  s2.gpa << endl;
cout << s2.id << endl;
```

*Comparisons:*

To compare two structure objects, we must compare them field by field.

```
if (s1 == s2) {} ; //  error

if ( s1.name == s2.name && ... ) {}; //error arrays can't be compared

if (strcmp(s1.name, s2.name) == 0 &&
        s1.id == s2.id &&
        abs (s1.gpa - s2.gpa) < .00001 )  { ... }
```

// correct, note if we compare float numbers using "==" we might get unpredicted logical errors.

*Pointer to structure*

Again similar to fundamental data types we can use the reference (&) and dereference (*) operators to assign the address of a structure object to a pointer and dereference the pointer pointing to an object.

```
student    s = {"Ali", 2.3, 123};
student  *sptr = &s;
cout << (*sptr) .name << endl;   // (*sptr) is an object of type student
```

We can use the operator (->) to directly access the fields of a structure object through a pointer.

```
cout << sptr->name << endl;
```

Note that the dot "." operator has higher precedence than the dereference operator (*) operator. Therefore, you should note that the expression, *sptr.name is not the same as (*sptr).name . The *sptr.name actually mean *(sptr.name)

We can use the new operator to dynamically allocate object:

```
student * s = new student;
```

and the delete operator to free memory,

```
delete s;
```

*Arrays of structures*

We can build arrays of data structure exactly as we did with the fundamental data types,

```
student A[20];   // declaring a static array that can hold up to 20 student object
student *A = new student[20]; //dynamically allocating memory for a 20 element of student.
```

Read in data from keyboard :

```
for (int i=0; i<20; i++) {
        cin >> A[i].name >> A[i].avg >> A[i].id;
}
```

print the average gpa for all 20 students:

```
float sum = 0;
for ( int i=0; i<20; i++) {
        sum += A[i].gpa;
}
cout << "Avg GPA = " << sum / i;
```

*Arrays of Pointers to structures*

also we can build array of pointers to structure object

```
student *A[100];
for (int i=0; i<100; i++) {
        A[i] = new student;
}
```

fill the array with data from keyboard

```
for (int i=0; i<100; i++) {
        cin >> (*A[i]).name >> (*A[i]).gpa >> (*A[i]).id;
}
```

or using the (->) operator

```
for (int i=0; i<100; i++) {
        cin >> A[i]->name >> A[i]->gpa >> A[i]->id ;
}
```

Print the name of the student who has the highest score

```
int hi = 0;;
for (int i=1; i<100; i++) {
        if (A[i]->gpa > A[hi]->gpa) {
                hi = i;
}

cout << A[hi] -> name ;
```

*Nested structures:*

An element of a structure can itself be another structure.

```
struct address {
        char street[20];
        char city[20];
        char country[20];
};

struct employee {
        char name[20];
        address addr;
        int id;
        float salary;
};

employee e;
cin >> e.a.street;
cin >> e.a.city;
cin >> e.a.country;
cin >> e.name;
cin >> id;
cin >> salary;
```

*Structures and function*

Structure objects can be passed and returned to/from function exactly the same as fundamental data type.

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

a function that return true if two points are equal is:

```
bool pointsAreEqual(point p1, point p2) {
        return (p1.x == p2.x && p1.y == p2.y)
}
```

a function that returns the point in the middle of the two points passing by value

```
point midpoint(point p1, point p2) {  // passing p1 and p2 by value
        point mid;
        mid.x = (p1.x + p2.x) / 2;
        mid.y = (p1.y + p2.y) /2;
        return mid;
}
```

use the function

```
point p1 = {2, 4};
point p2 - {4, 6};
point p3 = midpoint(p1, p2);
```

pass by reference

```
void midpoint(point p1, point p2, point &m) {  // passing p1 and p2 by value
        m.x = (p1.x + p2.x) / 2;
        m.y = (p1.y + p2.y) /2;
}
```

use the function

```
point p1 = {2, 4};
point p2 - {4, 6};
point p3;
midpoint(p1, p2, p3);
```

pass by address

```
void midpoint(point *p1, point *p2, point *m) {  // passing p1 and p2 by value
        m->x = (p1->x + p2->x) / 2;
        m->y = (p1->y + p2->y) /2;
}
```

use the function

```
point p1 = {2, 4};
point p2 - {4, 6};
point p3;
midpoint(&p1, &p2, &p3);
```