C Programs: Structures

Structure Types

Structures. A structure is a collection of data values, possibly of different types that jointly describes one physical or logical object.

Example. A person has a first name, middle name, last name, a gender and a birth date. If that person is a student, (s)he will also have a major, a status (freshman, sophomore, etc...) and a GPA.

There is more than one value that describes a person or a student. Generally speaking, one can use multiple variables to describe each individual aspect of the person/student, but handling this would be inconvenient, for example, in situations, where you need to pass the information about a person into a function.

struct type definition in C. C allows us instead to create a special new data type that combines all desired values into a single object. The struct declaration has the following syntax:

```c
typedef struct {
    <Type> <Name>;
    ...
    <Type> <Name>;
} <structName>;
```

Here, typedef and struct are keywords; <Type> <Name> pairs declare individual fields of the structure (they are just like a regular variable declaration) and <structName> is a C identifier that can be used to refer to the newly declared data type.

Example. Consider the following two declarations:

```c
typedef struct {
    char firstName[10];
    char lastName[10];
    int yearOfBirth;
} Person;
```
The typedef declaration declares a structure type `studentType`, which consists of five fields, `firstName`, `lastName`, `yearOfBirth`, `GPA` and `major`.

The second declaration, creates two variables, `s1` and `s2` of type `studentType`.

Access to fields

Each field of a `struct` type variable is a separate independent value. To access it, we use the . (dot) operator. The syntax is:

```C
<StructVarName>.<FieldName>
```

For example, to access the first name of a student in the variable `s1`, we write

```C
s1.firstName
```

This can be used both on left- and right-hand sides of the assignment statement.

Example. In the example below, information about one student is read from standard input stream, while information about another student is assigned directly.

```C
typedef struct {
    char firstName[10];
    char lastName[10];
    int yearOfBirth;
    float GPA;
    char major[10];
} studentType;

studentType s1, s2;

scanf("%s", s1.firstName); /* note the lack of use of & */
scanf("%s", s1.lastName);
scanf("%d", &s1.yearOfBirth); /* note the use of & */
scanf("%f", &s1.GPA);
scanf("%s", s1.major);

strcpy(s2.firstName, "Bob");
strcpy(s2.lastName, "Smith");
s2.yearOfBirth = 1990;
s2.GPA = s1.GPA;
strcpy(s2.major, "CS");
```

Access to whole structure

Unlike arrays, struct type variables are NOT POINTERS. Therefore, variables of struct types:

- can appear on the right- and left-hand sides of the assignment statements:

```C
s1 = s2; /* all the values from s2 are copied into s1 */
```
• are passed to function **by value** only.
• can be used as **return types of functions**.