Structures

A structure is a derived data type. It is constructed using objects of other types.

A structure is a collection of related variables under one name. It may contain variables of many different data types.

Structures are used to define objects and their attributes.

Here is the syntax for the structure definition:

struct structure_tag {

$$\left.\begin{array}{l}\text{data\_type var\_name;}\\ \ldots\end{array}\right\}\ \text{Data members}$$

};

Where:

Keyword **struct** – introduces the structure definition.

Structure tag – names the structured definition and is used with the keyword **struct** to declare variables of the structure type.

Structure Members – the variables declared within braces of the structure definition. These members are the attributes of the structure.

For example,

Define the structure definition of a student.

First: Identify the possible attributes of a student.

Id number, name, age, tuition fee

Second: Define the structure.

```
struct Student{
      int idnumber;
      char name[30];
      int age;
      float tuitionfee;
};
```

By convention, the name of the structure always starts with a big letter.

Note: Members of the same structures must have unique names, but two different structures may contain members of the same name without conflict.

For example,

Student has attributes id number, name, age and tuition fee.

Employee has attributes id number, name, age and salary.

Based on the comparison, both objects have the same attributes (i.e. id number, age and name) and only differ on the fourth attribute that is tuition fee for student and salary for employee.

For the structure definitions:

```
struct Student{
      int idnumber;
      char name[30];
      int age;
      float tuitionfee;
};

struct Employee{
      int idnumber;
      char name[30];
      int age;
      float salary;
};
```

These two structures will work without conflict since to refer to a structure member, you need to refer first to the structure where it belongs to.

Declaration of Structure Variable

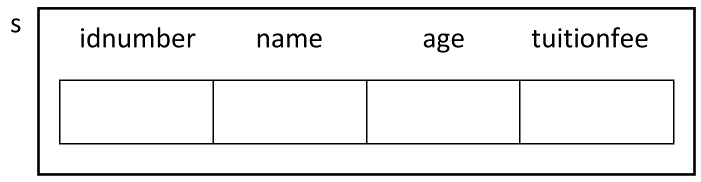
Here is the syntax:

struct structure_tag var_name;

For example,

```
struct Student s;
```

Variable s is the structure variable. To declare, always specify the name of the structure with the struct keyword. With the above example, s is allocated memory with the size of struct Student. Below is an illustration of s If using the previous definition of struct Student.



The Use of typedef

Since to declare a structure variable you always need to have the struct keyword and the structure tag, there would always be two words before the variable. So to simplify, we can use typedef.

The keyword **typedef** provides a mechanism for creating synonyms (or aliases) for previously defined data types. There are three ways to use typedef for structures. Here are the examples:

```
1. typedef struct Student Stude;
2. typedef struct Student{

         int idnumber;

         char name[30];

         int age;

         float tuitionfee;

   }Stude;

3. typedef struct {

         int idnumber;

         char name[30];

         int age;

         float tuitionfee;

   }Stude;
```

When using the third example, the structure tag becomes optional. In this way, there is only one name to use in declaring a structure variable, that is Stude.

From the above examples, Stude is not a variable name but an alias for struct Student. So instead of declaring,

```
      struct Student s;
```

We can use,

```
      Stude s;
```

Note: Creating a new name with typedef does not create a new data type; typedef simply creates a new type name, which may be used as an alias for an existing type names.

Initializing Structures

Structures can be initialized using initializer lists as with arrays. To initialize a structure, follow the variable name in the structure declaration with an equal sign and a brace-enclosed, comma-separated list of initializers.

For example,

```
Stude s = {101, “Jose Rizal”, 18, 24500.25};
```

To illustrate:

idnumber	name	age	tuitionfee
101	Jose Rizal	18	24500.25

However, remember to initialize with values that correctly match the data type. Otherwise, an error may occur.

Accessing Structures Members

To access the data members, two operators can be used:

1. dot operator (.)

   The dot operator is used to directly access the structure member.

   Here is the syntax:

   structureVariable.memberName

   For example,

   ```
   s.age = 20;
   
   strcpy(s.name,“Marlowe”);
   
   printf(“%d”,s.age);
   
   printf(“%s”,s.name);
   ```

2. structure pointer operator / arrow operator (->)

   The arrow operator is used when using a pointer to indirectly access the structure members.

   Here is the syntax:

   structurePointer->memberName

   For example:

   ```
   struct Student s, *sptr;
   
   sptr=&s;
   
   sptr->age=20;
   
   strcpy(sptr->name,”Marlowe”);
   ```

Nested Structures

It is possible to place structures inside another structure.

For example,

```
        struct Birth{
              int month, day, year;
        };
        struct Stud{
              char name[20];
              int age;
              float grade;
              struct Birth bday;
        };
        struct Stud s;
```

Structure variable bday is of type struct Birth. How can we access an inner structure variable?

For example, to access day,

```
        s.bday.day = 6;
        printf("%d", s.bday.day);
```

Here is a sample exercise. Consider the following structures:

```
        struct name{
              char fname[30], lname[30], mi[2];
        };
        struct birth{
              int month, day, year;
        };
        struct bio{
              struct name Nm;
              struct birth Bd;
              int idno;
        };
        struct bio x;
```

Identify if the following statements are valid or not:

1. x.bio.idno=1012; //invalid
2. strcpy(x.Nm.fname, “Christopher”); //valid
3. strcpy(x.name.lname, “Smith”); //invalid
4. x.Bd.month=6; //valid
5. bio.birth.year=1975; //invalid

Structures to Functions

When structures or individual structure members are passed to functions, they are passed by value.

Here is an example:

```
void display(Stude s);
```

```
void display(Stude s){
    printf("%d %s %s %s %d %.2f\n", s.idnumber, s.name.firstname,
        s.name.middlename, s.name.lastname, s.age, s.tuitionfee);
}
```

```
display(s);
```

To pass a structure call by reference, pass the address of the structure variable.

Here is an example:

```
void input(Stude *s);
```

```
void input(Stude *s){
    printf("ID#: ");
    scanf("%d",&s->idnumber);
    fflush(stdin);
    printf("First Name: ");
    gets(s->name.firstname);
    printf("Middle Name: ");
    gets(s->name.middlename);
    printf("Last Name: ");
    gets(s->name.lastname);
    printf("Age: ");
    scanf("%d",&s->age);
    printf("Tuition fee: ");
    scanf("%f",&s->tuitionfee);
}
```

```
input(&s);
```

From the above example, since input() has to accept new values for s then it needs to pass s by reference. Thus, the function uses a pointer as its parameter and the arrow operator to access the elements.