A **structure** is a collection of values of different data types. In contrast with Pascal and other programming languages, a C structure is equivalent to a record.

Structure declaration – forms the template that may be used to create structure variable.

Structure elements/components – are variables that make up the structure.

The **syntax** of structure declaration

```
struct structure_tag_name {
       data_type_1 variable_1;
       data_type_2 variable_2;
       …
       data_type_n variable_n;
} structure_variable(s);
```

Where

- **structure_tage_name** is the name of the structure template, **NOT** a variable name.
- **structure_variable(s)** is/are a comma separated list of variable names.

Note: Either structure_tag_name or structure_variable(s) are optional but **NOT** both!

4 DIFFERENT WAYS TO DECLARE STRUCTURE VARIABLES:

1. WITH tagname, WITHOUT structure variables.
   Example:

```
struct Name {
       char Fname[20];
       char Mname[20];
       char Lname[20];
};
```

   This statement formally declares structure variables Person1 and Person2.

```
struct Name Person1, Person2;
```

Note: The structure template (the structure declaration should be defined globally!);

2. WITHOUT tagname, WITH structure variables
   Example:

```
struct {
       char Fname[20];
       char Mname[20];
       char Lname[20];
} Person1, Person2;
```

   This time, Person1 and Person2 being the structure variables are "variables" themselves to be used for containing data.

3. WITH tagname, WITH structure variables
   Example:

```
struct Name{
       char Fname[20];
       char Mname[20];
       char Lname[20];
}Person1, Person2;

struct Name BankMgr, BankTeller;
```

4. Using **typedef**: WITHOUT tagname, WITH structure variable – now as the user-defined data type.
   Example:

```
typedef struct{
       char Fname[20];
       char Mname[20];
       char Lname[20];
}Person1;

Person1 BankMgr, BankTeller;
```

Note: Person1 in the structure template definition is NOT a structure variable but a user defined data type which could be used to declare structure variables, e.g. BankMgr, BankTeller.

[You can also use typedef with tagname, with structure variable. More on this when we use linked lists.]

MORE on typedef construct:
Type definitions

- Used to give names to simple and structured types
- Can be used in subsequent declarations and definitions

Syntax:

```
typedef type-specifier new-type;
```

Examples:

```
typedef int ordinal;
typedef char *stringtype;
typedef float number[10];

ordinal x, y; /* x and y are type
ordinal (which is basically of type
int) */

stringtype str; /* str is a character
pointer */
```

```
number expenses; /*expenses is a 10-
element array of float*/
```

Referencing Structure Elements/Components
To access individual components, the **dot** notation is used. (The dot is called the **member operator).**

```
structure_variable.element_name;
```

Examples:
```
printf("%s\n", BankMgr.Fname);
printf("%s", BankMgr.Lname);
fgets(BankMgr.Mname, 20, stdin);
```

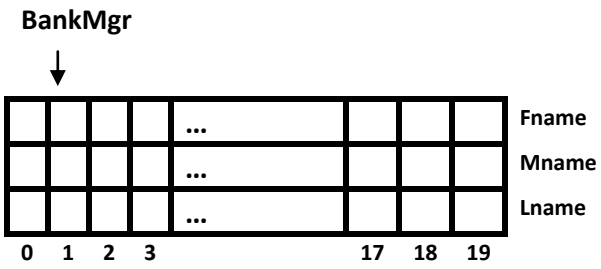
Question: How can we visualize a structure in the memory?

Let's look at this declaration as an example.

```
typedef struct{
        char Fname[20];
        char Mname[20];
        char Lname[20];
}Person1;

Person1 BankMgr, BankTeller;
```

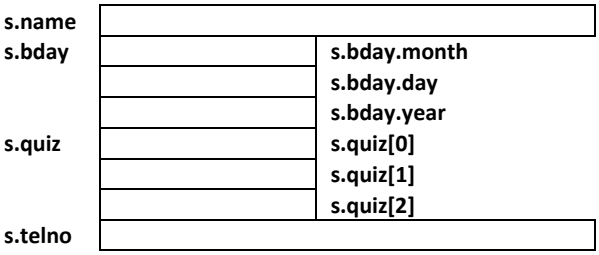
In the array discussion we had, we said that the elements are stored sequentially. The structure is saved in the similar manner. Every variable is allocated in sequential order. The structure variable points to the first element of the structure.



Structure within Structure

```
typedef struct {
        int month, day, year;
} birth;

typedef struct {
        char name[20];
        birth bday;
        int quiz[3], telno;
} student;
student s;
```



Array of Structures

```
student cs11[100];
```

Below is a code fragment that reads in values to store in array cs21:

```
int i, j;
for( i = 0; i < 100; i++ )
{
        printf("Student no %d: ", i+1);
        printf("Enter name: ");
        gets(cs11[i].name);

        printf("Birthday\n");
        printf("Enter month in number: ");
        scanf("%d", &cs11[i].bday.month);
        printf("Enter day in number: ");
        scanf("%d", &cs11[i].bday.day);
        printf("Enter year in number: ");
        scanf("%d", &cs11[i].bday.year);

        printf("Quiz scores\n");
        for( j = 0; j < 3; j++ )
        {
                printf("Quiz %d: ", j+1);
                scanf("%d",
&cs11[i].quiz[j]);
        }

        printf("Enter telephone number: ");
        scanf("%d", &cs11[i].telno);
}
```

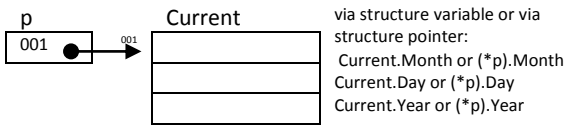
Structure Pointers

```
struct Date {
        char Monthr[10];
        int Day, Year;
} Current; /*variable declaration*/

struct Date *p; /*structure pointer
declaration*/

p = &Current; /*places the address of
the structure Current into the pointer
variable p*/
```

To access the components: via structure variables or structure pointer.



Note: Use the dot operator to access structure elements when operating on the structure itself. Use the arrow operator when referencing a structure through a pointer, i.e. p->Month instead of (*p).Month. (The latter form is considered archaic.)

Structure as Function Parameters
Examples:

A. Pass by Value

```
…
void display_student ( student ss ) {
        printf("Name: %s\n", ss.name);
        printf("Bday: %d-%d-%d\n",
ss.bday.month, ss.bday.day, ss.bday.year);
        printf("Quizzes: %d %d %d\n",
ss.quiz[0], ss.quiz[1], ss.quiz[2]);
        printf("Telno: "%d\n", ss.telno);
}//end of function

…
main()
{
        student s;
        /*function call*/
        display_student(s);
} //end of main
```

B. Pass by Reference

```
…
void display_student ( student *ss ) {
        printf("Name: %s\n", ss->name);
        printf("Bday: %d-%d-%d\n", ss-
>bday.month, ss->bday.day, ss->bday.year);
        printf("Quizzes: %d %d %d\n", ss-
>quiz[0], ss->quiz[1], ss->quiz[2]);
        printf("Telno: "%d\n", ss->telno);
}//end of function

…
main()
{
        student s;
        /*function call*/
        display_student(&s);
} //end of main
```