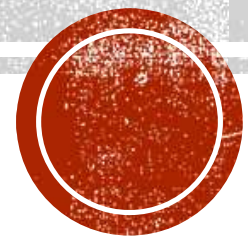
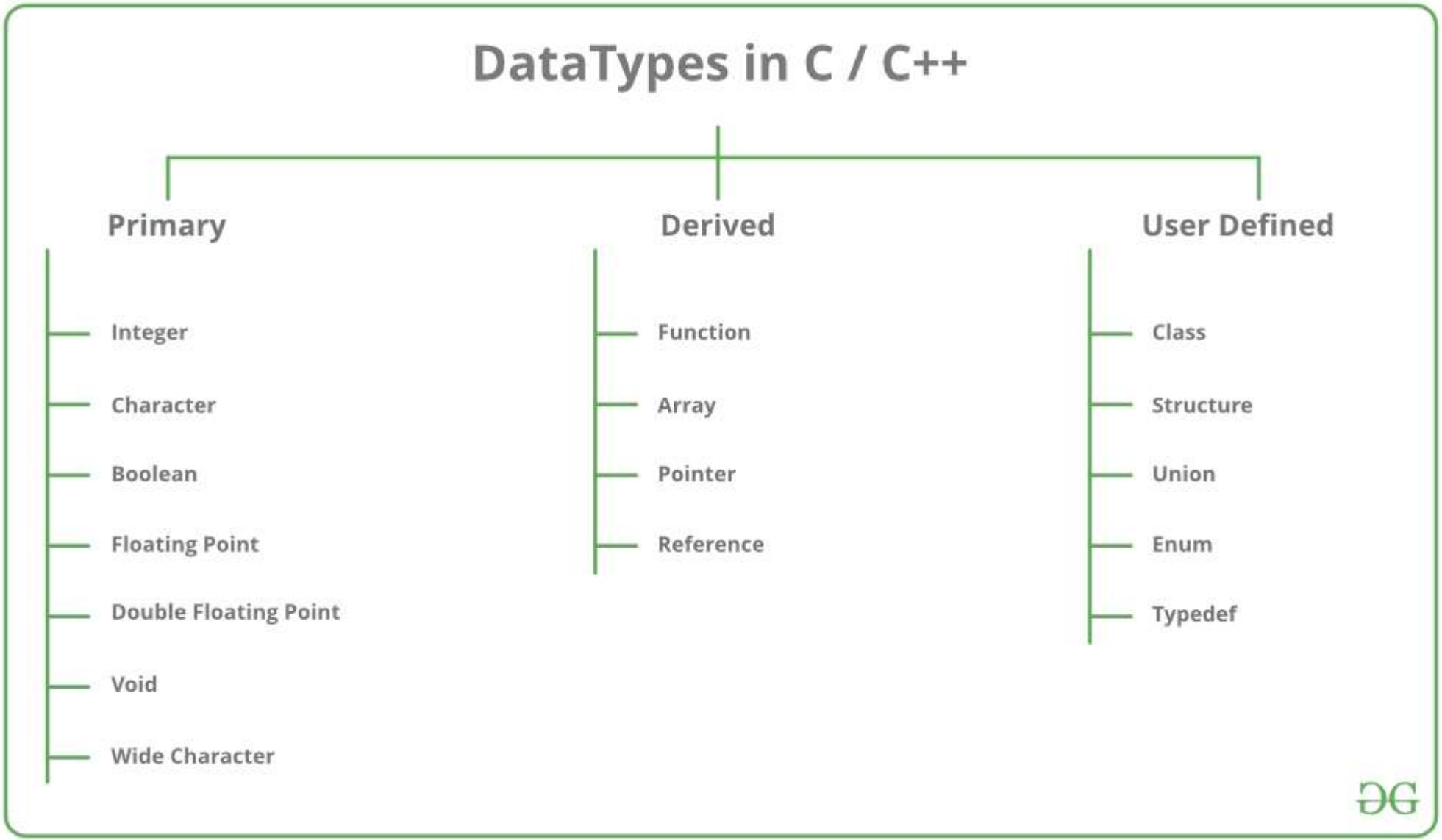
STRUCTURE & UNION



Module 5

Kirti S.

Academic Year:-2023-24


DataTypes in C / C++
Primary
Integer
Character
Boolean
Floating Point
Double Floating Point
Void
Wide Character
Derived
Function
Array
Pointer
Reference
User Defined
Class
Structure
Union
Enum
Typedef

WHAT IS STRUCTURE

- Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.
- Unlike an array, a structure can contain many different data types (int, float, char, etc.).
- **Structure in c is a user-defined data type** that enables us to store the collection of different data types. Each element of a structure is called a member.
- The **,struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

SYNTAX OF STRUCTURE

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memberN;
};
```

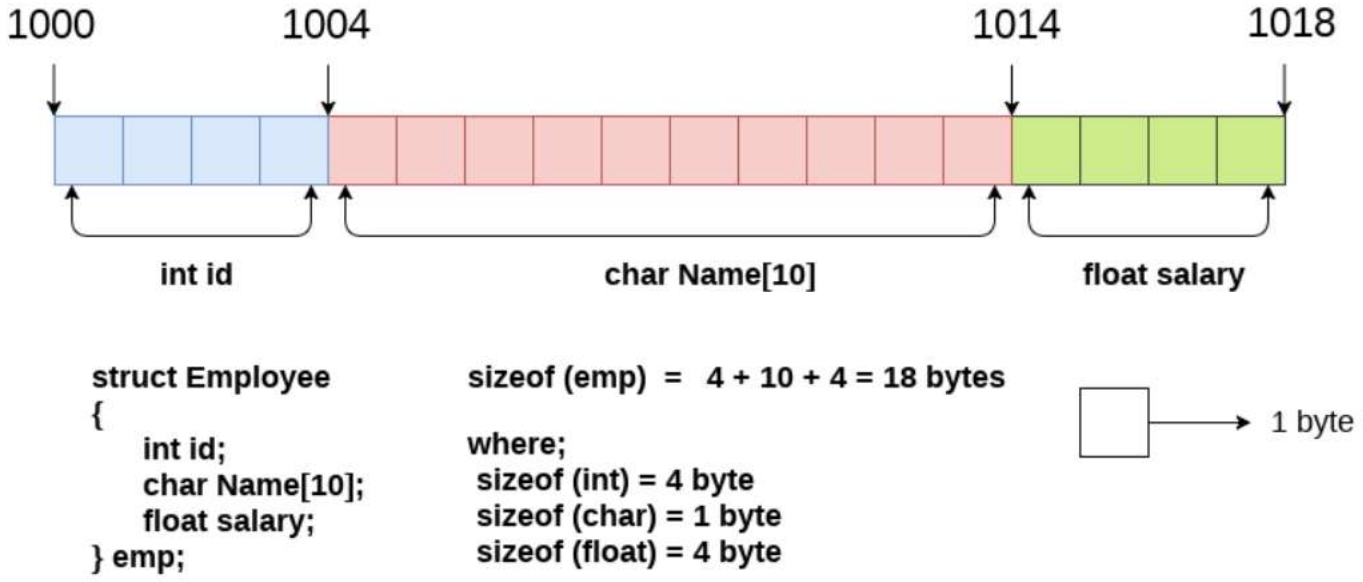
EXAMPLE

- Example to define a structure for an entity employee in c.

```
struct employee
{   int id;
    char name[10];
    float salary;
};
```

sizeof(struct employee)

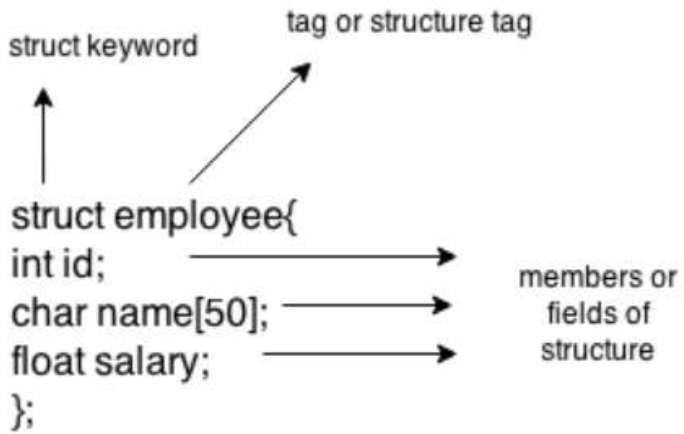
The following image shows the memory allocation of the structure employee that is defined in the above example.



https://www.tutorialspoint.com/cprogramming/c_data_types.htm

CONT..

- Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



DECLARING STRUCTURE VARIABLE

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1. BY STRUCT KEYWORD WITHIN MAIN() FUNCTION

- Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{   int id;
    char name[50];
    float salary;
};
```

- Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

```
# include <stdio.h>
int main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1, b2, b3 ;
}
```

```
# include <stdio.h>
struct book
{
char name ;
float price ;
int pages ;
} ;

int main( )
{
struct book b1, b2, b3 ;
}
```

2. BY DECLARING A VARIABLE AT THE TIME OF DEFINING THE STRUCTURE.

- Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1;
```

Which approach is good

- If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.
- If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in main() function.

EXAMPLE

```
#include<stdio.h>

#include <string.h>

struct employee

{   int id;

    char name[50];

}e1,e2;  //declaring e1,e2 variable for struct
ure
```

CONT..

- Note the following points while declaring a structure type:

(a) The closing brace (}) in the structure type declaration must be followed by a semicolon (;).

(b) It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the 'form' of the structure.

(c) Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined. In very large programs they are usually put in a separate header file, and the file.

ACCESSING MEMBERS OF THE STRUCTURE

- There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

```c
// Create a structure
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
}
```

OUTPUT:

13 B Some text

EXAMPLE

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {

  struct myStructure s1 = {13};

printf("%d %c %s", s1.myNum,
s1.myLetter, s1.myString);

  return 0;
}
```

OUTPUT:

13

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
} s1 = {13, 'B', "Some text"};


int main() {
  // Create a structure variable and
assign values to it


  // Print values
  printf("%d %c %s", s1.myNum,
s1.myLetter, s1.myString);

  return 0;
}
```

OUTPUT:

13 B Some text

EXAMPLE

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  struct myStructure s1 = {13, 'B', "Some text"};
  struct myStructure s2 = {14, 'C', "Some other text"};

  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);
    printf("%d %c %s", s2.myNum, s2.myLetter, s2.myString);

  return 0;
}
```

Separate memory allocated for s1 and s2.

EXAMPLE

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  struct myStructure s1;
  struct myStructure s2 = {14, 'C', "Some other text"};
                                    s1=s2;

  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

      printf("%d %c %s", s2.myNum, s2.myLetter, s2.myString);


  return 0;
}
```

OUTPUT:

14 C Some other text

14 C Some other text

EXAMPLE

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main(){
  struct myStructure s1;
  struct myStructure s2 = {14, 'C', "Some other text"};
                                    s1.myNum=1;

                                    s1.myLetter='a';


      printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

      printf("%d %c %s", s2.myNum, s2.myLetter, s2.myString);


  return 0;
}
```

OUTPUT:

1 a ☺

14 C Some other text

EXAMPLE

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];

}e1;  //declaring e1 variable for structure
```

```
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, "Saniya Jaiswal");//copying
string into char array
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);

return 0;
}
```

```
OUTPUT:
employee 1 id : 101
employee 1 name : Saniya Jaiswal
```

```c
# include <stdio.h>
int main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1, b2, b3 ;
printf ( "Enter names, prices & no. of pages of 3 books\n" ) ;
scanf ( "%c %f %d", &b1.name, &b1.price, &b1.pages ) ;
scanf ( "%c %f %d", &b2.name, &b2.price, &b2.pages ) ;
scanf ( "%c %f %d", &b3.name, &b3.price, &b3.pages ) ;
printf ( "And this is what you entered\n" ) ;
printf ( "%c %f %d\n", b1.name, b1.price, b1.pages ) ;
printf ( "%c %f %d\n", b2.name, b2.price, b2.pages ) ;
printf ( "%c %f %d\n", b3.name, b3.price, b3.pages ) ;
return 0 ;
}
```

OUTPUT:

```
Enter names, prices and no. of pages of 3 books
A 100.00 354
C 256.50 682
F 233.70 512
And this is what you entered
A 100.000000 354
C 256.500000 682
F 233.700000 512
```

TYPEDEF IN C

- The typedef is a keyword used in C programming to provide some meaningful names to the already existing variable in the C program.
- It behaves similarly as we define the alias for the commands. In short, we can say that this keyword is used to redefine the name of an already existing variable.
- Syntax of typedef

  typedef <existing_name> <alias_name>

- In the above syntax, '**existing_name**' is the name of an already existing variable while '**alias name**' is another name given to the existing variable.

CONT...

- For example, suppose we want to create a variable of type unsigned int, then it becomes a tedious task if we want to declare multiple variables of this type. To overcome the problem, we use a typedef keyword.

```
typedef unsigned int unit;
```

- In the above statements, we have declared the unit variable of type unsigned int by using a typedef keyword.
- Now, we can create the variables of type unsigned int by writing the following statement:

```
unit a, b;
```

- instead of writing:

```
unsigned int a, b;
```

- Till now, we have observed that the typedef keyword provides a nice shortcut by providing an alternative name for an already existing variable. This keyword is useful when we are dealing with the long data type especially, structure declarations.

EXAMPLE

```c
#include <stdio.h>
int main()
{
typedef unsigned int unit;
unit i,j;
i=10;
j=20;
printf("Value of i is :%d",i);
printf("\nValue of j is :%d",j);
return 0;
}
```

OUTPUT:
Value of i is :10
Value of j is :20

USING TYPEDEF WITH STRUCTURES

- Consider the below structure declaration:

```
struct student
{
char name[20];
int age;
};
struct student s1;
```

CONT...

- In the above structure declaration, we have created the variable of student type by writing the following statement:

```
struct student s1;
```

- The above statement shows the creation of a variable, i.e., `s1`, **but the statement is quite big. To avoid such a big statement, we use the typedef** keyword to create the variable of type student.

```
struct student
{
char name[20];
int age;
};
typedef struct student stud;
stud s1, s2;
```

- In the above statement, we have declared the variable stud of type struct student. Now, we can use the stud variable in a program to create the variables of type struct student.

CONT...

- **The above typedef can be written as:**

typedef struct student

{

char name[20];

int age;

} **stud;**

stud s1,s2;

From the above declarations, we conclude that typedef keyword reduces the length of the code and complexity of data types. It also helps in understanding the program.

EXAMPLE

```
#include <stdio.h>
typedef struct student
{
char name[20];
int age;
}stud;

int main()
{
stud s1;
printf("Enter the details of student s1: ");
printf("\nEnter the name of the student:");
scanf("%s",&s1.name);
printf("\nEnter the age of student:");
scanf("%d",&s1.age);
printf("\n Name of the student is : %s", s1.name);
printf("\n Age of the student is : %d", s1.age);
return 0;
}
```

OUTPUT:

```
Enter the details of student s1:
Enter the name of the student: Peter
Enter the age of student: 28
Name of the student is : Peter
Age of the student is : 28
```

C ARRAY OF STRUCTURES

- Why use an array of structures?
- Consider a case, where we need to store the data of 3 students. We can store it by using the structure as given below.

```
#include<stdio.h>
struct student
{
    char name[20];
    int id;
    float marks;
};
void main()
{
    struct student s1,s2,s3;
    printf("Enter the name, id, and marks of student 1 ");
    scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
    printf("Enter the name, id, and marks of student 2 ");
    scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
    printf("Enter the name, id, and marks of student 3 ");
    scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
    printf("Printing the details....\n");
    printf("%s %d %f\n",s1.name,s1.id,s1.marks);
    printf("%s %d %f\n",s2.name,s2.id,s2.marks);
    printf("%s %d %f\n",s3.name,s3.id,s3.marks);
}
```

Output

Enter the name, id, and marks of student 1 James 90 90

Enter the name, id, and marks of student 2 Adoms 90 90

Enter the name, id, and marks of student 3 Nick 90 90

Printing the details....

James 90 90.000000

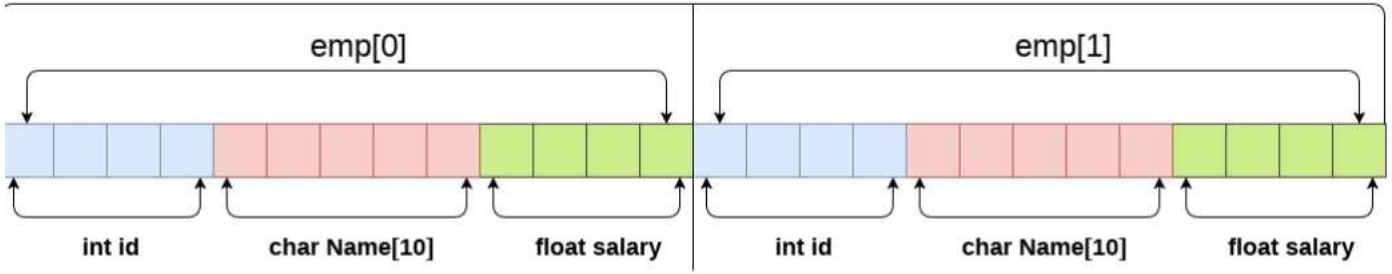
Adoms 90 90.000000

Nick 90 90.000000

CONT..

- In the above program, we have stored data of 3 students in the structure. However, the complexity of the program will be increased if there are 20 students. In that case, we will have to declare 20 different structure variables and store them one by one. This will always be tough since we will have to declare a variable every time we add a student. Remembering the name of all the variables is also a very tricky task. However, c enables us to declare an array of structures by using which, we can avoid declaring the different structure variables; instead we can make a collection containing all the structures that store the information of different entities.
- Array of Structures in C
- An array of structure in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Array of structures




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

```
#include<stdio.h>
#include <string.h>
struct student{
int rollno;
char name[10];
};
int main(){
int i;
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```

Enter Records of 5 students

Enter Rollno:1

Enter Name:Sonoo

Enter Rollno:2

Enter Name:Ratan

Enter Rollno:3

Enter Name:Vimal

Enter Rollno:4

Enter Name:James

Enter Rollno:5

Enter Name:Sarfraz

Student Information List:

Rollno:1, Name:Sonoo

Rollno:2, Name:Ratan

Rollno:3, Name:Vimal

Rollno:4, Name:James

Rollno:5, Name:Sarfraz

NESTED STRUCTURE IN C

- C provides us the feature of nesting one structure within another structure by using which, complex data types are created.
- For example, we may need to store the address of an entity employee in a structure. The attribute address may also have the subparts as street number, city, state, and pin code.
- Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee. Consider the following program.

EXAMPLE

```
#include<stdio.h>
struct address
{
   char city[20];
   int pin;
   char phone[14];
};
struct employee
{
   char name[20];
   struct address add;
};
```

```
void main ()
{
   struct employee emp;
   printf("Enter employee information?\n");
   scanf("%s %s %d %s",&emp.name,&emp.add.city, &emp.add.pin,
&emp.add.phone);
   printf("Printing the employee information....\n");
   printf("name: %s\nCity: %s\nPincode: %d\nPhone:
%s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
}
```

Output

Enter employee information?

Arun

Delhi

110001

1234567890

Printing the employee information....

name: Arun

City: Delhi

Pincode: 110001

Phone: 1234567890

THE STRUCTURE CAN BE NESTED IN THE FOLLOWING WAYS.

- By separate structure
- By Embedded structure

1) Separate structure

Here, we create two structures, but the dependent structure should be used inside the main structure as a member. Consider the following example.

```
struct Date
{
   int dd;
   int mm;
   int yyyy;
};
struct Employee
{
   int id;
   char name[20];
   struct Date doj;
}emp1;
```

As you can see, doj (date of joining) is the variable of type Date. Here doj is used as a member in Employee structure. In this way, we can use Date structure in many structures.

2) Embedded structure

The embedded structure enables us to declare the structure inside the structure. Hence, it requires less line of codes but it can not be used in multiple data structures. Consider the following example.

```
struct Employee
{
   int id;
   char name[20];
   struct Date
    {
      int dd;
      int mm;
      int yyyy;
    }doj;
}emp1;
```

ACCESSING NESTED STRUCTURE

- We can access the member of the nested structure by Outer Structure. Nested_Structure.member as given below:

```
e1.doj.dd

e1.doj.mm

e1.doj.yyyy
```

C Nested Structure example

Let's see a simple example of the nested structure in C language.

```
#include <stdio.h>
#include <string.h>
struct Employee
{
  int id;
  char name[20];
  struct Date
   {
     int dd;
     int mm;
     int yyyy;
   }doj;
}e1;
int main( )

{
  //storing employee information
  e1.id=101;
  strcpy(e1.name, "Sonoo Jaiswal");//copying string into
char array
  e1.doj.dd=10;
  e1.doj.mm=11;
  e1.doj.yyyy=2014;

  //printing first employee information
  printf( "employee id : %d\n", e1.id);
  printf( "employee name : %s\n", e1.name);
  printf( "employee date of joining (dd/mm/yyyy) :
%d/%d/%d\n", e1.doj.dd,e1.doj.mm,e1.doj.yyyy);
  return 0;
}
```

Output:

employee id : 101

employee name : Sonoo Jaiswal

employee date of joining (dd/mm/yyyy) : 10/11/2014

Passing structure to function

PASSING STRUCTURE TO FUNCTION

- Just like other variables, a structure can also be passed to a function. We may pass the structure members into the function or pass the structure variable at once. Consider the following example to pass the structure variable employee to a function display() which is used to display the details of an employee.

```
#include<stdio.h>
struct address
{
    char city[20];
    int pin;
    char phone[14];
};
struct employee
{
    char name[20];
    struct address add;

};
```

```
void display(struct employee);
void main ()
{
    struct employee emp;
    printf("Enter employee information?\n");
     scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.
pin, emp.add.phone);
    display(emp);
}
void display(struct employee emp)
{
  printf("Printing the details....\n");
  printf("%s %s %d %s",emp.name,emp.add.city,emp.add.pin,e
mp.add.phone);
}
```

UNION

A union is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

To define a union, you must use the union statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows.

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

CONT...

- The union tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.
- At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional.
- Here is the way you would define a union type named Data having three members i, f, and str −

```
union Data {
   int i;
   float f;
   char str[20];
} data;
```

CONT..

- Now, a variable of Data type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., **same memory location, can be used to store multiple types of data.** You can use any built-in or user defined data types inside a union based on your requirement.

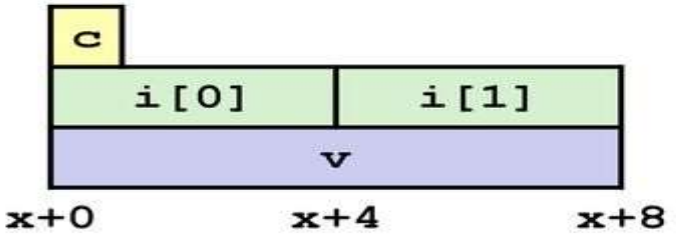
- The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.

- The following example displays the total memory size occupied by the above union –

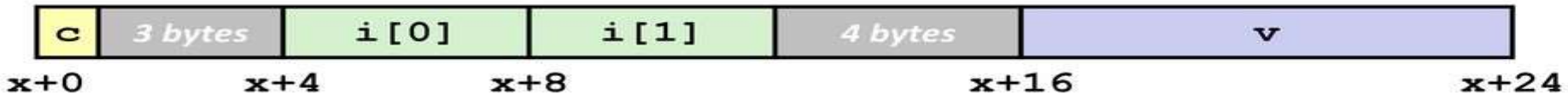
Union Allocation

- **Allocate according to largest element**

```
union U1 {
   char c;
   int i[2];
   double v;
} u;
```




```
struct S1 {
   char c;
   int i[2];
   double v;
} s;
```



EXAMPLE

```
#include <stdio.h>
#include <string.h>
 union Data {
   int i;
   float f;
   char str[20];
};
 int main( ) {
   union Data data;
   printf( "Memory size occupied by data :
%d\n", sizeof(data));
   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Memory size occupied by data : 20

ACCESSING UNION MEMBERS

- To access any member of a union, we use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use the keyword union to define variables of union type. The following example shows how to use unions in a program –

```
#include <stdio.h>
#include <string.h>

union Data {
   int i;
   float f;
   char str[20];
};
```

```
int main( ) {

   union Data data;

   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");

   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

data.i : 1917853763

data.f : 4122360580327794860452759994368.000000

data.str : C Programming

Here, we can see that the values of i and f members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of str member is getting printed very well.

CONT..

- Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having unions −

```
#include <stdio.h>
#include <string.h>

union Data {
   int i;
   float f;
   char str[20];
};
```

```
int main( ) {

   union Data data;

   data.i = 10;
   printf( "data.i : %d\n", data.i);

   data.f = 220.5;
   printf( "data.f : %f\n", data.f);

   strcpy( data.str, "C Programming");
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

CONT...

When the above code is compiled and executed, it produces the following result –

data.i : 10

data.f : 220.500000

data.str : C Programming

Here, all the members are getting printed very well because one member is being used at a time.

Structure	Union
We use the struct statement to define a structure.	We use the union keyword to define a union.
Every member is assigned a unique memory location.	All the data members share a memory location.
Change in the value of one data member does not affect other data members in the structure.	Change in the value of one data member affects the value of other data members.
You can initialize multiple members at a time.	You can initialize only the first member at once.
A structure can store multiple values of the different members.	A union stores one value at a time for all of its members
A structure's total size is the sum of the size of every data member.	A union's total size is the size of the largest data member.
Users can access or retrieve any member at a time.	You can access or retrieve only one member at a time.