UNIT- III: Functions: Introduction, Function Definition, Function Declaration, Return values and their Types, Function Calls, Categories of Functions, nesting of Functions, Recursion, Passing arrays to Functions, Storage Classes.
Structure: Basics of Structures, Structures to Functions, Arrays of Structures, Structures with in Structures, Arrays with in structures, Unions.

UNIT-III

1. What is a function? Why we use functions in C language? Give an example.

Ans:

Function in C:

A function is a block of code that performs a specific task. It has a name and it is re usable .It can be executed from as many different parts in a program as required, it can also return a value to calling program.

All executable code resides within a **function**. It takes input, does something with it, then give the answer. A C program consists of one or more functions.

A computer program cannot handle all the tasks by itself. It requests other program like entities called functions in C. We pass information to the function called **arguments which** specified when the function is called. A function either can return a value or returns nothing. Function is a subprogram that helps reduce coding.

Simple Example of Function in C

```
#include<stdio.h>
#include <conio.h>
int adition (int, int); //Function Declaration
int addition (int a, int b) //Function Definition
{
int r;
r=a + b;
return (r);
}
int main()
```

```
{
int z;
z= addion(10,3); //Function Call
printf ("The Result is %d", z);
return 0;
}
```

Output: The Result is 13

Why use function

Basically there are two reasons because of which we use functions

1. Writing functions avoids rewriting the same code over and over. For example - if you have a section of code in a program which calculates the area of triangle. Again you want to calculate the area of different triangle then you would not want to write the same code again and again for triangle then you would prefer to jump a "section of code" which calculate the area of the triangle and then jump back to the place where you left off. That section of code is called ‘function'.

2. Using function it becomes easier to write a program and keep track of what they are doing. If the operation of a program can be divided into separate activities, and each activity placed in a different function, then each could be written and checked more or less independently. Separating the code into modular functions also makes the program easier to design and understand.

2. Distinguish between Library functions and User defined functions in C and Explain with examples.

Ans:

Types of Function in C:

(i). Library Functions in C

C provides library functions for performing some operations. These functions are present in the c library and they are predefined.

For example sqrt() is a mathematical library function which is used for finding the square root of any number .The function scanf and printf() are input and output library function similarly we have strcmp() and strlen() for string manipulations. To use a library function we have to include some header file using the preprocessor directive #include.

For example to use input and output function like printf() and scanf() we have to include stdio.h, for math library function we have to include math.h for string library string.h should be included.

(ii). User Defined Functions in C

A user can create their own functions for performing any specific task of program are called user defined functions. To create and use these function we have to know these 3 elements.

1. Function Declaration
2. Function Definition
3. Function Call

1. Function declaration

The program or a function that calls a function is reffered to as the calling program or calling function. The calling program should declare any function that is to be used later in the program this is known as the function declaration or function prototype.

2. Function Definition

The function definition consists of the whole description and code of a function. It tells that what the function is doing and what are the input outputs for that. A function is called by simply writing the name of the function followed by the argument list inside the parenthesis.
Function definitions have two parts:
Function Header
The first line of code is called Function Header.
int sum(int x, int y)
It has three parts
(i). The name of the function i.e. sum
(ii). The parameters of the function enclosed in parenthesis
(iii). Return value type i.e. int
Function Body
Whatever is written with in { } is the body of the function.

3. Function Call

In order to use the function we need to invoke it at a required place in the program. This is known as the function call.

3. Write some properties and advantages of user defined functions in C?

Ans:

Properties of Functions

- Every function has a unique name. This name is used to call function from “main()” function.
- A function performs a specific task.
- A function returns a value to the calling program.

Advantages of Functions in C

- Functions has top down programming model. In this style of programming, the high level logic of the overall problem is solved first while the details of each lower level functions is solved later.
- A C programmer can use function written by others
- Debugging is easier in function
- It is easier to understand the logic involved in the program
- Testing is easier

4. Explain the various categories of user defined functions in C with examples?

Ans:

A function depending on whether arguments are present or not and whether a value is returned or not may belong to any one of the following categories:

(i) Functions with no arguments and no return values.

(ii) Functions with arguments and no return values.

(iii) Functions with arguments and return values.

(iv) Functions with no arguments and return values.

(i) Functions with no arguments and no return values:-

When a function has no arguments, it does not return any data from calling function. When a function does not return a value, the calling function does not receive any data from the called function. That is there is no data transfer between the calling function and the called function.

Example

```
#include <stdio.h>
#include <conio.h>
void printmsg()
```

```
{
printf ("Hello ! I Am A Function .");
}
int main()
{
printmsg();
return 0;
};
```

Output : Hello ! I Am A Function .

(ii) Functions with arguments and no return values:-

When a function has arguments data is transferred from calling function to called function. The called function receives data from calling function and does not send back any values to calling function. Because it doesn't have return value.

Example

```
#include<stdio.h>
#include <conio.h>
void add(int,int);

void main()
{
int a, b;

printf("enter value");

scanf("%d%d",&a,&b);

add(a,b);
}
```

```
void add (intx, inty)
{
int z ;
z=x+y;

printf ("The sum =%d",z);
}
```

output : enter values 2 3

The sum = 5

(iii) Functions with arguments and return values:-

In this data is transferred between calling and called function. That means called function receives data from calling function and called function also sends the return value to the calling function.

Example

```
#include<stdio.h>
#include <conio.h>
int add( int, int);
main()
{
 int a,b,c;

printf(“enter value”);

scanf(“%d%d”,&a,&b);

c=add(a,b);
printf ("The sum =%d",c);
 }
int add (int x, int y)
```

```
{
int z;
z=x+y;

return z;
}
```

output : enter values 2 3

The sum = 5

(iv) Function With no Argument And Return Type:-

When function has no arguments data can not be transferred to called function. But the called function can send some return value to the calling function.

Example

```
#include<stdio.h>
#include <conio.h>
int add( );
main()
{
 int c;

c=add();
printf ("The sum =%d",c);
 }
int add ()
{
int x,y,z;

printf("enter value");

scanf("%d%d",&a,&b);
```

```
z=x+y;

return z;
}
```

Output: enter values 2 3

The sum = 5

5. What is inter function communication?

Ans:

Inter function communication

A function is a self-contained block or sub program that perform a special task when it is called. Whenever a function is called to perform a specific task, then the called function performs that task and the result is returns back to the calling function. The data flow between the calling and called functions to perform a specific task is known as **inter function communication.**

6. Explain different methods for transferring data between calling and called function.

Ans:

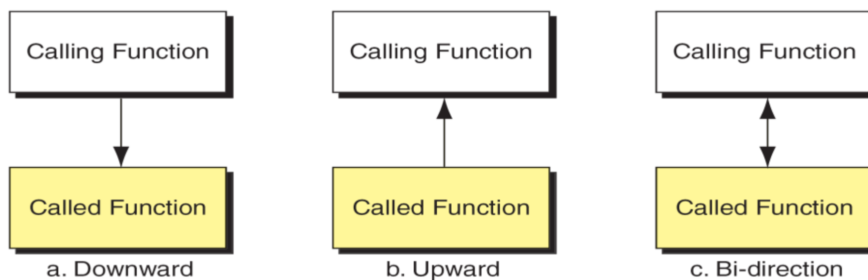
Different methods for transferring data between calling and called function.

The data flow between the calling and called functions can be divided into three strategies:

(i) Downward flow

(ii) Upward flow

(iii) Bi-directional flow.



(i) Downward flow:-

The calling function sends data to the called function is represented as downward flow. No data flows in opposite directions. It is only one way communication. The data items are passed from calling to called function and called function may change the values, but the original values in calling function remains unchanged. This is also known as call by value mechanisnm in 'C' language.

Ex:-

```
void downFun(int x,int y);

int main(void)

{

  int a=5;

 downFun(a,15);

  printf("%d",a);

  return 0;

  }

  Void downFun(int x ,int y)

  {

  x=x+y;

  return;

  }
```

(ii) Upward flow:-

The called function sends data to the calling function is represented as upward flow. Here the called function does not receive any data prior from calling function. It is also one way communication only. Since the data items are passed from only called to calling function, so that it may read data from keyboard and then passed it to calling function.

Ex:-

```
void upFun(int *ax,int *ay);
```

```
int main(void)
{
int a, b;
upFun(&a,&b);
printf("%d%d",a,b);
return 0;
}
void upFun(int *ax ,int *ay)
{
*ax=23;
*ay=8;
return;
}
```

(iii) Bidirectional flow:-

The calling function sends data to the called function, after performing task the called function sends back the result to called function is represented as bidirectional flow. Here the data items are passed in two ways. This is also known as call by value mechanisnm in ‘C’ language.

```
void biFun(int *ax, int *ay);
int main(void)
{
int a=2, b=6;
biFun(&a,&b);
return 0;
}
void biFun(int *ax ,int *ay)
```

```
{
*ax=*ax+2;
*ay=*ay/*ax;
return;
}
```

7. Explain the Parameter Passing Mechanisms in C-Language with examples.

Ans:

Most programming languages have 2 strategies to pass parameters. They are

(i) pass by value

(ii) pass by reference

(i) Pass by value (or) call by value :-

In this method calling function sends a copy of actual values to called function, but the changes in called function does not reflect the original values of calling function.

Example program:

```
#include<stdio.h>
void fun1(int,int);
void main( )
{
int a=10, b=15;
fun1(a,b);
printf("a=%d,b=%d", a,b);
```

```
getch():
}
void fun1(int x, int y)
{
x=x+10;
y= y+20;
}
```

Output: a=10 b=15

The result clearly shown that the called function does not reflect the original values in main function.

(ii) Pass by reference (or) call by address :-

In this method calling function sends address of actual values as a parameter to called function, called function performs its task and sends the result back to calling function. Thus, the changes in called function reflect the original values of calling function. To return multiple values from called to calling function we use pointer variables.

Calling function needs to pass '&' operator along with actual arguments and called function need to use '*' operator along with formal arguments. Changing data through an address variable is known as indirect access and '*' is represented as indirection operator.

Example program:

```
#include<stdio.h>
void fun1(int,int);
void main( )
{
```

```
int a=10, b=15;
fun1(&a,&b);
printf("a=%d,b=%d", a,b);
getch():
}
void fun1(int *x, int *y)
{
*x = *x + 10;
*y =  *y + 20;
}
```

Output: a=20 b=35

The result clearly shown that the called function reflect the original values in main function. So that it changes original values.

8. Differentiate actual parameters and formal parameters.

Ans:

Actual parameters	Formal parameters
The list of variables in calling function is known as **actual parameters**.	The list of variables in called function is known as **formal parameters**.
Actual parameters are variables that are declared in function call.	Formal parameters are variables that are declared in the header of the function definition.
Actual parameters are passed without using type	Formal parameters have type preceding with them.
main( ) {	return_typefunction_name(formal parameters) {

…..	…..
function_name (actual parameters);	function body;
…..	…..
}	}

Formal and actual parameters must match exactly in type, order, and number.

Formal and actual parameters need not match for their names.

9. Write a C program for exchanging of two numbers using call by reference mechanism.

Ans:

```
#include<stdio.h>

void swap (int, int);

void main( )

{

int a=10, b=15;

swap(&a, &b);

printf("a=%d,b=%d", a,b);

getch();

}

void swap(int *x, int *y)

{

int temp;

temp = *x;
```

```
*x = *y;

*y =  temp;

}
```

Output: **a = 15 b=10**

10. Define scope? Explain local and global variable with examples?

Ans:

Scope

The scope of a variable is the portion of a program in which the variable may be visible or available. There are 3 types of scopes in which a variable can fall:

1. Block scope
2. Function scope or local scope
3. Global scope or file scope

The longevity or lifetime or extent of a variable refers to the duration for which the variable retains a given value during the execution of a program.

The same variable name may appear in different scopes. It is the duty of the compiler to decide whether the different occurrences of the variable refer to the same variable or not.

1. Block Scope:

A variable is said to have block scope, if it is recognised only with in the block where it is declared.The following example demonstrates this concept:

```
#include<stdio.h>
main()
{
{/*Block-1*/
int a;
a=10;
printf("%d\t%d",a,b);
}
{/*Block-2*/
int b;
b=20;
printf("%d\t%d",a,b);
}
}
```

2. Local Scope:

A variable is said to have local scope or function scope, if it is recognised only with in the function where it is declared. The following example demonstrates this concept:

```
#include<stdio.h>
void fun(void);
main()    //function-1
{
int a;
a=10;
printf("a=%d\tb=%d",a,b);
fun();
}
void fun()  //function-2
{
int b;
b=20;
printf("a=%d\tb=%d",a,b);
}
```

Global Scope:

A variable is said to have global scope or file scope, if it is recognised with in blocks and all the functions defined in a program. A global variable can be declared outside of all functions with in a file. The following example demonstrates the concept of global variable:

```
#include<stdio.h>

void fun(void);

int c=30;  //global variable

int d;  //global variable

main()

{

int a;

a=10;

d=40;

printf("\n With in main()");

printf("\n Local variable a=%d",a);
```

```
printf("\n Global variables c=%d\td=%d",c,d);
fun();
}
void fun()
{
int b;
b=20;
d=60;
printf("\n With in fun()");
printf("\n Local variable b=%d",b);
printf("\n Global variables c=%d\td=%d",c,d);
}
```

11. Explain in detail about nesting of functions with example.

Ans:

Nesting of functions

The process of calling a function within another function is called nesting of function

Syntax:-

```
main()
{
………..
Function1();
……….
}
Function1();
```

```
{
 ………..
Function2();
………..
}
Function2();
{
………..
Function3();
………..
}
Function3();
{
…………
}
```

main () can call function 1() where function1 calls function2() which calls function3() and so on

Ex:

```
float ratio (int,int,int);
int difference (int,int);
void main()
{
int a,b,c,d;
printf(" enter three numbers");
```

```
scanf("%d%d%d",  &a, &b, &c );

d= ratio(a,b,c);

printf ("%f\n" ,d);

}

float ratio(int x,int y,int z);

{

int u ;

u=difference(y,z);

if (u)

return(x/(y-z));

else

return (0.0);

}

int difference (int p,int q)

{

if(p!=q)

return 1;

else

return 0;

}
```

main reads values a,b,c, and calls ratio() to calculate a/(b-c) ratio calls another function difference to test whether (b-c) is zero or not this is called nesting of function

12. What is recursive function? Write syntax for recursive functions.

Ans:

Recursion

Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied.

When a function calls itself, a new set of local variables and parameters are allocated storage on the stack, and the function code is executed from the top with these new variables. A recursive call does not make a new copy of the function. Only the values being operated upon are new. As each recursive call returns, the old local variables and parameters are removed from the stack, and execution resumes immediately after the recursive call inside the function.

The main advantage of recursive functions is that we can use them to create clearer and simpler versions of several programs.

Syntax:-

A function is **recursive** if it can call itself; either directly:

```
void f( )
{
f( );
}
```

(or) indirectly:

```
void f( )
{
g( );
}

void g( )
{
f( );
}
```

Recursion rule 1: Every recursive method must have a **base case** -- a condition under which no recursive call is made -- to prevent infinite recursion.

Recursion rule 2: Every recursive method must make progress toward the base case to prevent infinite recursion

13. Write a program to find factorial of a number using recursion.

Ans:

```
#include<stdio.h>
int fact(int);
main()
{
int n,f;
printf("\n Enter any number:");
scanf("%d",&n);
f=fact(n);
printf("\n Factorial of %d is %d",n,f);
}
int fact(int n)
{
int f;
if(n==0||n==1)  //base case
f=1;
else
f=n*fact(n-1);  //recursive case
return f;
}
```

Output:- Enter any number: 5

Factorial of 5 is 120

14. Differentiate between recursion and non-recursion.

Ans:

Recursion:- Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied.

When a function calls itself, a new set of local variables and parameters are allocated storage on the stack, and the function code is executed from the top with these new variables. A recursive call does not make a new copy of the function. Only the values being operated upon are new. As each recursive call returns, the old local variables and parameters are removed from the stack, and execution resumes immediately after the recursive call inside the function.

Ex:-

```
void main( )
{
```

```
int n=5;
fact( n);
}
int fact( )
{
if(n==0 || n==1)
return 1;
else
return(n*fact(n-1));
}
```

Non-Recursion:-

In C language we need repetition of statements more than one time in our program. There are two methods in C language to handle repeated statements.

(i) Iterative (or) non-recursion functions

(ii) Recursive functions.

(i) Iterative (or) non-recursion functions :-

Using looping statements we can handle repeated statements in ‘C’. The example of non recursion is given below.

Syntax:-

```
void main( )
{
int n=5;
res = fact(n);
printf("%d",res);
}
int fact( )
{
for(i=1;i<=n;i++)
{
f=f+1;
}
return f;
}
```

Differences:

- Recursive version of a program is slower than iterative version of a program due to overhead of maintaining stack.
- Recursive version of a program uses more memory (for the stack) than iterative version of a program.
- Some times, recursive version of a program is simpler to understand than iterative version of a program.

15. Write a program to calculate GCD of two numbers using recursion

Ans:

```
#include<stdio.h>

int gcd(int, int);

main()

{

int a,b;

printf("\n Enter any two numbers:");

scanf("%d%d",&a,&b);

printf("\nGCD=%d",gcd(a,b));

}

int gcd(inta,int b)

{

if(b==0)   //base case

return a;

else

return gcd(b,a%b);  //recursive case

}
```

16. Write a program to generate Fibonacci series using recursive and non-recursive functions.

Ans:

Recursion Program:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int f,f1,f2,n,i,res;

clrscr();

printf("enter the limit:");

scanf("%d",&n);

printf("The fibonacci series is:");

for(i=0;i<n;i++)

{

res=fib(i);

printf("%d\t",res);

}

getch();

}

int fib(int i)

{

if(i==0)

return 0;

else if(i==1)

return 1;
```

```
else
return(fib(i-1)+fib(i-2));
}
```

Non-Recursion program:

```
#include<stdio.h>
int main()
{
int n,a,b,c,i;
printf("\n Enter how many numbers:");
scanf("%d",&n);
a=0;
b=1;
printf("%d\t%d",a,b);
for(i=3;i<=n;i++)
{
c=a+b;
printf("\t%d",c);
a=b;
b=c;
}
return 0;
}
```

17. How to Pass Array Individual Elements to Functions? Explain with example program.

Ans:

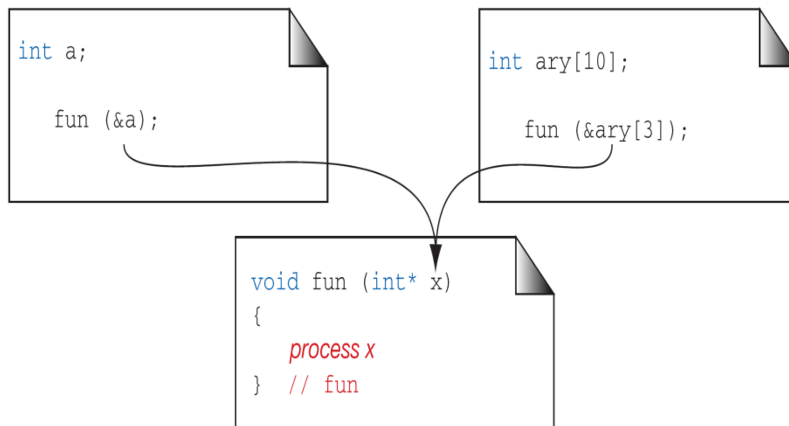
Arrays with functions:

Array element can be passed to a function by calling the function by value, or by reference. in the call by value, we pass values of array elements to the function, whereas in the call by reference, we pass address of array element to the function. These two calls are illustrated below

- The function must be called by passing only the name of the array.
- In the function definition, the formal parameters must be an array type, the size of the array does not need to be specified.
- The function prototype must show that the argument is an array.

To process arrays in a large program, we have to be able to pass them to functions. We can pass arrays in two ways: pass individual elements or pass the whole array.

Passing Individual Elements

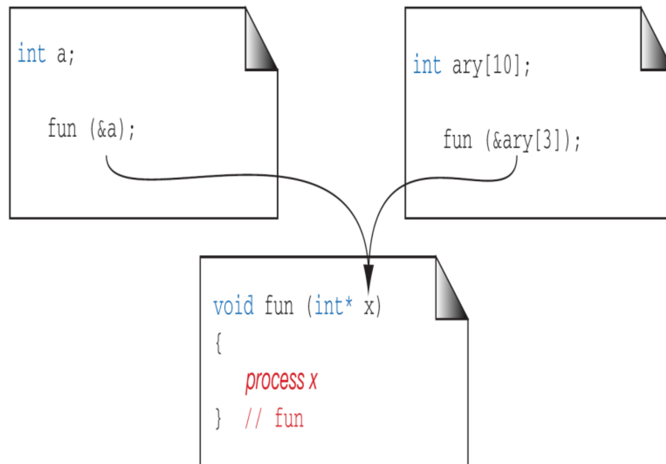


Program using call by value

```
void KIT(int);

void main()

{

int i;

int marks[6]={55,66,77,88,99,100};

for(i=0;i<=5;i++)

KIT(marks[i]);

}

void KIT(int r)
```

```
{
printf("%d",r);
}
```

Here we are passing an individual element at a time to the function KIT() and getting it printed in the function KIT().Note that ,since at a time only one element is being passed, this element is collected in an ordinary integer variable r, i n the function KIT().



Program using call by reference

```
void display(int *);
void main()
{
int i;
int marks[6]={55,66,77,88,99,100};
for (i=0;i<=5;i++)
display(&marks[i]);
}
void display(int *r)
{
printf("%d",*r);
```

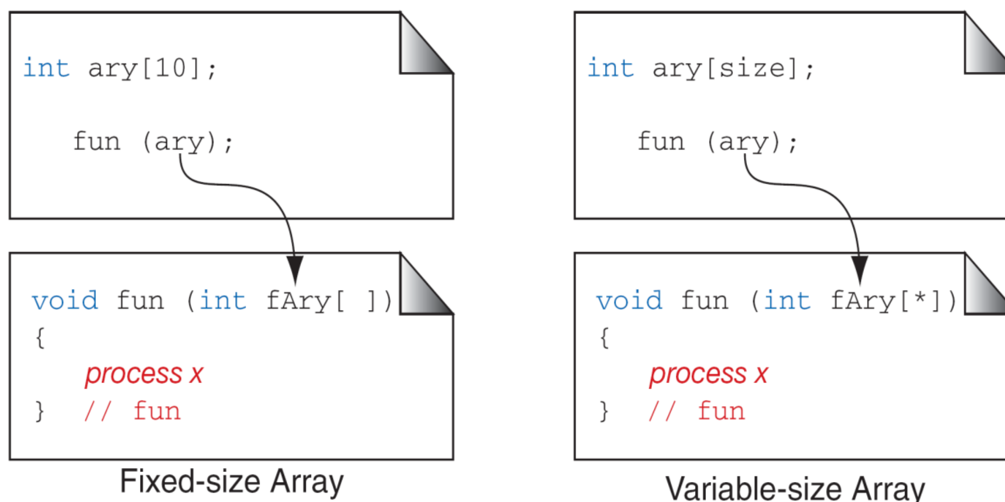
}

Here, we are passing address of individual array element to the function display().hence, the variable in which this address is collected r,is declared as a pointer a variable. and since n contains the address of array element, to print out the array element, we are using the 'value at address' operator*.

18. How can we pass the Whole Array to Functions? Explain with example program.

Ans:

Passing an entire array to a function:



Like the value of simple variables, it is also possible to the values of an array to a function. to pass an array to a function, it is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments. For example, the call largest(arr,n); will pass all the elements contained in the array as arguments. for example, the call largest(arr,n); will pass all the elements contained in the array arr of size n. the called function expecting this call must be appropriately defined. the largest function header might look like:

int largest(int array[],int size);

The function largest is defined to take two arguments, the array name and the size of the array to specify the number of elements in the array.the declaration of the formal argument array is made as int array[]; the pair of bracket informs the compiler that the argument array is an array of numbers. it is not necessary to specify the size of the array here.

Let us consider a problem of finding the largest value the largest value in an array of elements. the program is as follows

```
float largest (int*,int);

void main()

{

float arr[5]={2.5,-8.2,4.4,5.8,8.2};

printf("\n%d",largest(arr ,5);

}

float largest(int a[],int size)

{

int i:

float max;

max=a[0];

for(i=1;i<size;i++)

{

if(max<a[i])

max=a[i];

}

return(max);

}
```

When the function call largest (arr,5) is made, the values of all elements of the arr are passed to the corresponding element a in the called function. the largest function finds the largest value in the array and returns the result to main.

19. What are different types of storage classes in 'C'? (or)

Explain briefly auto and static storage classes with examples? (or)

Explain extern and register storage classes with example programs.

Ans:

Storage classes in 'C'

Variables in C differ in behavior. The behavior depends on the storage class a variable may assume. From C compiler's point of view, a variable name identifies some physical location within the computer where the string of bits representing the variable's value is stored. There are four storage classes in C:

(a) Automatic storage class
(b) Register storage class
(c) Static storage class
(d) External storage class

(a) Automatic Storage Class:-

The features of a variable defined to have an automatic storage class are as under:

Storage	Memory.
Default initial value	An unpredictable value, which is often called a garbage value.
Scope	Local to the block in which the variable is defined.
Life	Till the control remains within the block in which the variable is defined

Following program shows how an automatic storage class variable is declared, and the fact that if the variable is not initialized it contains a garbage value.

```
main( )
{
auto int i, j ;
printf ( "\n%d %d", i, j ) ;
}
```

When you run this program you may get different values, since garbage values are unpredictable. So always make it a point that you initialize the automatic variables properly, otherwise you are likely to get unexpected results. Scope and life of an automatic variable is illustrated in the following program.

```
main( )
```

```
{
auto int i = 1 ;
{
auto int i = 2 ;
{
auto int i = 3 ;
printf ( "\n%d ", i ) ;
}
printf ( "%d ", i ) ;
}
printf ( "%d", i ) ;
}
```

Static Storage Class:-

The features of a variable defined to have a **static** storage class are as under:

Storage	Memory.
default initial value	Zero.
Scope	Local to the block in which the variable is defined.
Life	Value of the variable persists between different function calls

The following program demonstrates the details of static storage class:

main( )
{
increment( ) ;
increment( ) ;
increment( ) ;
}
 	
increment( )
{
auto int i = 1 ;
printf ("%d\n", i) ;
i = i + 1 ;
}
	main( )
	{
	increment( ) ;
	increment( ) ;
	increment( ) ;
	}
	
	increment( )
	{
	static int i = 1 ;
	printf ("%d\n", i) ;
	i = i + 1 ;
	}

The output of the above programs would be:

1 1 1	1 2 3

Extern Storage Class:-

The features of a variable whose storage class has been defined as external are as follows:

Storage	Memory
default initial value	Zero
Scope	Global
Life	As long as the program execution does not come to end

External variables differ from those we have already discussed in that their scope is global, not local. External variables are declared outside all functions, yet are available to all functions that care to use them. Here is an example to illustrate this fact.

Ex:

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

```
extern int i;
void main()
{
printf("i=%d",i);
}
```

Register Storage Class:-

The features of a variable defined to be of **register** storage class are as under:

Storage	**CPU Registers**
default initial value	An unpredictable value, which is often called a garbage value.
Scope	Local to the block in which the variable is defined.
Life	Till the control remains within the block in which the variable is defined.

A value stored in a CPU register can always be accessed faster than the one that is stored in memory. Therefore, if a variable is used at many places in a program it is better to declare its storage class as register. A good example of frequently used variables is loop counters. We can name their storage class as register.

```
main( )
{
register int i ;
for ( i = 1 ; i <= 10 ; i++ )
printf ( "\n%d", i ) ;
}
```

20. Enumerate the scope rules in C .

Ans:

Scope: Scope of a variable is the part of program in which it can be used

Scope rules

The rules are as under:

- Use **static** storage class only if you want the value of a variable to persist between different function calls.
- Use **register** storage class for only those variables that are being used very often in a program. Reason is, there are very few CPU registers at our disposal and many of them might be busy doing something else. Make careful utilization of the scarce resources. A typical application of **register** storage class is loop counters, which get used a number of times in a program.
- Use **extern** storage class for only those variables that are being used by almost all the functions in the program. This would avoid unnecessary passing of these variables as arguments when making a function call. Declaring all the variables as **extern** would amount to a lot of wastage of memory space because these variables would remain active throughout the life of the program.
- If we don't have any of the express needs mentioned above, then use the **auto** storage class. In fact most of the times we end up using the **auto** variables, because often it so happens that once we have used the variables in a function we don't mind loosing them.

21. What is a structure? Write the syntax for structure declaration.

Ans:

Definition:

A structure is a collection of one or more variables of different data types, grouped together under a single name. By using structures variables, arrays, pointers etc can be grouped together.

Structures can be declared using two methods as follows:

(i) Tagged Structure:

The structure definition associated with the structure name is referred as tagged structure. It does not create an instance of a structure and does not allocate any memory.

The general form or syntax of tagged structure definition is as follows,

struct TAG	**Ex:-**	struct student
{		{

```
Type varaible1;                         int htno[10];
Type variable2;                         char name[20];
……                                      float marks[6];
……                                  };
Type variable-n;
};
```

Where,

- struct is the keyword which tells the compiler that a structure is being defined.
- Tag_name is the name of the structure.
- variable1, variable2 … are called members of the structure.
- The members are declared within curly braces.
- The closing brace must end with the semicolon.

(ii) Type-defined structures:-

- The structure definition associated with the keyword typedef is called type-defined structure.

- This is the most powerful way of defining the structure.

The syntax of typedefined structure is

```
typedef struct                  Ex:-    typedef struct
{                                       {
Type varaible1;                         int htno[10];
Type variable2;                             char name[20];
……                                          float marks[6];
……                                      }student;
Type variable-n;
}Type;
```

where

- typedef is keyword added to the beginning of the definition.
- struct is the keyword which tells the compiler that a structure is being defined.
- variable1, variable2…are called fields of the structure.
- The closing brace must end with type definition name which in turn ends with semicolon.

Variable declaration:

Memory is not reserved for the structure definition since no variables are associated with the structure definition. The members of the structure do not occupy any memory until they are associated with the structure variables.

After defining the structure, variables can be defined as follows:

For first method,

struct TAG v1,v2,v3….vn;

For second method, which is most powerful is,

Type v1,v2,v3,….vn;

Alternate way:

```
struct TAG
{
Type varaible1;
Type variable2;
  ……
  ……
Type variable-n;
} v1, v2, v3;
```

Ex:

```
struct book
{
```

```
char name[30];
int pages;
float price;
}b1,b2,b3;
```

22. Declare the C structures for the following scenario:

(i) College contains the following fields: College code (2characters), College Name, year of establishment, number of courses.

(ii) Each course is associated with course name (String), duration, number of students. (A College can offer 1 to 50 such courses)

Ans:

(i). Structure definition for college :-

```
Struct college
{
        char code[2];
        char college_name[20];
        int year;
        int no_of_courses;
};
```

Variable declaration for structure college :-

```
void main( )
{
struct college col1,col2,col3;
....
}
```

(ii). Structure definition for course :-

```
struct course
{
        char course_name[20];
        float duration;
        int no_of_students;
};
```

Variable declaration for structure course :-

```
void main( )
{
struct course c1,c2,c3;
….
}
```

23. How to initialize structures in ‘C’? Write example.

Ans:

The rules for structure initialization are similar to the rules for array initialization. The initializers are enclosed in braces and separated by commas. They must match their corresponding types in the structure definition.

The syntax is shown below,

struct tag_name variable = {value1, value2,… value-n};

Structure initialization can be done in any one of the following ways

(i) Initialization along with Structure definition:-

Consider the structure definition for student with three fields name, roll number and average marks. The initialization of variable can be done as shown below,

```
struct student
{
char name [5];
```

int roll_number;

float avg;

} s1= {“Ravi”, 10, 67.8};

The various members of the structure have the following values.

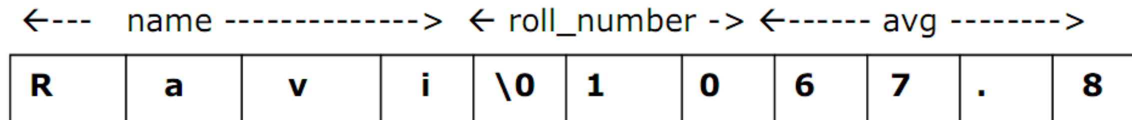


Figure 5.2 Initial Value of S1

(ii) Initialization during Structure declaration:-

Consider the structure definition for student with three fields name, roll number and average marks. The initialization of variable can be done as shown below,

typedef struct

{

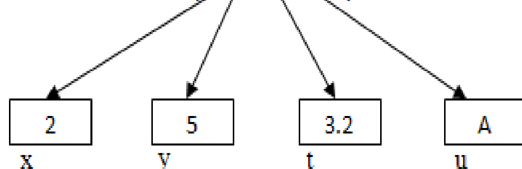
int x;

int y;

float t;

char u;

} SAMPLE;



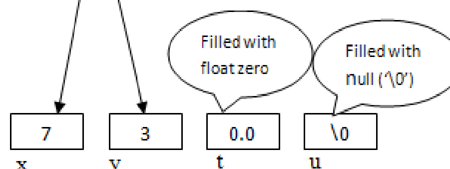


Figure Initializing Structures

The figure shows two examples of structure in sequence. The first example demonstrates what happens when not all fields are initialized. As we saw with arrays, when one or more initializers are missing, the structure elements will be assigned null values, zero for integers and floating-point numbers, and null ('\0') for characters and strings.

24. Define a structure type *personal*, that would contain person name, date of joining and salary. Write a program to initialize one person data and display the same.

Ans:

```
struct personal
{
    char name[20];
    int day;
    char month[10];
    int year;
    float salary;
};
void main( )
{
    Struct personal person = { "RAMU", 10 JUNE 1998 20000};
    printf("Output values are:\n");
    printf("%s%d%s%d%f", person.name, person.day, person.month, person.year, person.salary );
    getch( );
}
```

Output:-

RAMU 10 JUNE 1998 20000

25. How to access the data for structure variables using member ('.') operator?Explain with an example.

Ans:

We know that variables can be accessed and manipulated using expressions and operators. On the similar lines, the structure members can be accessed and manipulated. The members of a structure can be accessed by using dot(.) operator.

dot (.) operator

Structures use a dot (.) operator to refer its elements. Before dot, there must always be a structure variable. After the dot, there must always be a structure element. The dot operator is also known as member operator(period operator).

The **syntax** to access the structure members as follows,

structure_variable_name . structure_member_name

Consider the example as shown below,

```
struct student

{

char name [5];

int roll_number;

float avg;

};

struct student s1= {“Ravi”, 10, 67.8};
```

The members can be accessed using the variables as shown below,

```
s1.name --> refers the string “ravi”

s1.roll_number --> refers the roll_number 10

s1.avg     --> refers avg 67.8
```

26. Define a structure type *book*, that would contain book name, author, pages and price. Write a program to read this data using member operator ('.') and display the same.

Ans:

```
struct book
{
        char name[20];
        int day;
        char month[10];
        int year;
        float salary;
};
void main( )
{
        struct book b1;
        printf("Input values are:\n");
        scanf("%s %s %d %f", b1.title, b1.author, b1.pages, b1.price);
        printf("Output values are:\n");
        printf("%s\n %s\n %d\n %f\n", b1.title, b1.author, b1.pages, b1.price);
        getch( );
}
```

Output:-

```
Input values are:   C& DATA STRUCTURES     KAMTHANE      609      350.00
Output values are:
   C& DATA STRUCTURES
   KAMTHANE
   609
   350.00
```

27. How to pass a structure member as an argument of a function? Write a program to explain it.

Ans:

Structures are more useful if we are able to pass them to functions and return them.

By passing individual members of structure

This method is to pass each member of the structure as an actual argument of the function call. The actual arguments are treated independently like ordinary variables. This is the most elementary method and becomes unmanageable and inefficient when the structure size is large.

```
#include<stdio.h>
main ( )
{
float arriers (char *s, int n, float m);
typedef struct emp
{
char name[15];
int emp_no;
float salary;
}record;
record e1 = {"smith",2,20000.25};
e1.salary = arriers(e1.name,e1.emp_no,e1.salary);
}
float arriers(char *s, int n, float m)
{
m = m + 2000;
printf("\n%s %d %f ",s, n, m);
return m;
}
```

Output

```
smith 2     22000.250000
```

28. How to pass an entire structure as an argument of a function? (or)
Write a program to pass entire structure as an argument of a structure.

Ans:

Structures are more useful if we are able to pass them to functions and return them.

Passing Whole Structure

This method involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in the original structure. It is therefore, necessary for the function to return the entire structure back to the calling function.

The general format of sending a copy of a structure to the called function is:

return_type function_name (structure_variable_name);

The called function takes the following form:

```
data_typefunction_name(struct_typetag_name)
{
………
………
return(expression);
}
```

The called function must be declared for its type, appropriate to the data type it is expected to return.

The structure variable used as the actual argument and the corresponding formal argument in the called function must of the same struct type.

The return statement is necessary only when the function is returning some data back to the calling function. The expression may be any simple variable or structure variable or an expression using simple variables.

When a function returns a structure, it must be assigned to a structure of identical type in the calling function. The called functions must be declared in the calling function appropriately.

```
#include <stdio.h>

#include <string.h>

struct student
```

```
{
    int id;
    char name[20];
    float percentage;
};
void func(struct student record);
int main()
{
struct student record;
 record.id=1;
strcpy(record.name, "Raju");
record.percentage = 86.5;
return 0;
}
voidfunc(struct student record)
{
    printf(" Id is: %d \n", record.id);
    printf(" Name is: %s \n", record.name);
    printf(" Percentage is: %f \n", record.percentage);
}
```

Output:

Id is: 1

Name is: Raju

Percentage is: 86.500000

29. Write a program for illustrating a function returning a structure.

```
typedef struct
{
char name [15];
intemp_no;
float salary;
} record;
#include<stdio.h.>
#include<string.h>
void main ( )
{
record change (record);
record e1 = {"Smith", 2, 20000.25};
printf ("\nBefore Change %s %d %f ",e1.name,e1.emp_no,e1.salary);
e1 = change(e1);
printf ("\nAfter Change %s %d %f ",e1.name,e1.emp_no,e1.salary);
}
record change (record e2)
{
strcpy (e2.name, "Jones");
e2.emp_no = 16;
e2.salary = 9999;
return e2;
}
```

Output:

Smith 2 20000.25
Jones 16 9999.99

30. What is an array of structure? Declare a variable as array of structure and initialize it?

(or)

When the array of structures is used? Write syntax for array of structure.

Ans:

An array is a collection of elements of same data type that are stored in contiguous memory locations. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items.

As we have an array of integers, we can have an array of structures also. For example, suppose we want to store the information of class of students, consisting of name, roll_number and marks, A better approach would be to use an array of structures.

Array of structures can be declared as follows,

```
struct tag_name arrayofstructure[size];
```

Let’s take an example, to store the information of 3 students, we can have the following structure definition and declaration,

```
struct student
{
char name[10];
int rno;
float avg;
};

struct student  s[3];
```

Defines array called s, which contains three elements. Each element is defined to be of type struct student.

For the student details, array of structures can be initialized as follows,

```
struct student s[3]={{“ABC”,1,56.7},{“xyz”,2,65.8},{“pqr”,3,82.4}};
```

Ex: An array of structures for structure employee can be declared as

```
struct employee emp[5];
```

Let’s take an example, to store the information of 5 employees, we can have the following structure definition and declaration,

```
struct employee
{
int empid;
char name[10];
float salary;
};
struct employee  emp[5];
```

Defines array called emp, which contains five elements. Each element is defined to be of type struct employee.

For the employee details, array of structures can be initialized as follows,

```
struct employee emp[5] = {{1,"ramu"25,20000},
        {2,"ravi",65000},
        {3,"tarun",82000},
        {4,"rupa",5000},
        {5,"deepa",27000}};
```

31. Write a C program to calculate student-wise total marks for three students using array of structure.

Ans:

```
#include<stdio.h>

struct student

{

char rollno[10];

char name[20];

float sub[3];

float total;

};

void main( )

{

int i, j, total = 0;

struct student s[3];

printf("\t\t\t\t Enter 3 students details");

for(i=0; i<3; i++)

{
```

```
printf("\n Enter Roll number of %d Student:",i);
gets(s[i].rollno);
printf(" Enter the name:");
gets(s[i].name);
printf(" Enter 3 subjects marks of each student:");
total=0;
for(j=0; j<3; j++)
{
scanf("%d",&s[i].sub[j]);
total = total + s[i].sub[j];
}
}
printf("\n*******************************************");
printf("\n\t\t\t Student details:");
printf("\n*******************************************");
for(i=0; i<n; i++)
{
printf ("\n Student %d:",i+1);
printf ("\n Roll number:%s\n  Name:%s",s[i].rollno,s[i].name);
printf ("\nTotal marks =%f", s[i].total);
}
getch( );
}
```

32. Write a C program using array of structure to create employee records with the following fields: emp-id, name, designation, address, salary.

Ans:

```
#include<stdio.h>
struct employee
{
int emp_id;
char name[20];
char designation[10];
char address[20];
float salary;
}emp[3];
void main( )
{
int i;
printf("\t\t\t\t Enter 3 employees details");
for(i=0; i<3; i++)
{
scanf("%d",&emp[i].emp_id);
gets(emp[i].name);
gets(emp[i].designation);
gets(emp[i].address);
scanf("%f",&emp[i].salary);
}
printf("\n******************************************");
printf("\n\t\t\t Employee details:");
```

```
printf("\n*******************************************");
for(i=0; i<3; i++)
{
printf("%d",emp[i].emp_id);
puts(emp[i].name);
puts(emp[i].designation);
puts(emp[i].address);
printf("%f",emp[i].salary);
}
getch( );
}
```

33. What is structure within structure? Give an example for it. (or)

Write a C program to illustrate the concept of structure within structure.

Ans:

Nested Structure (Structure within structure)

A structure which includes another structure is called nested structure or structure within structure. i.e a structure can be used as a member of another structure. There are two methods for declaration of nested structures.

(i) The syntax for the nesting of the structure as follows

```
struct tag_name1
{
type1 member1;
 …….
…….
```

```
};
struct tag_name2
{
type1 member1;
……
……
struct tag_name1 var;
……
};
```

The syntax for accessing members of a nested structure as follows,

outer_structure_variable . innerstructurevariable . membername

(ii) The syntax of another method for the nesting of the structure is as follows

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;

        struct structure_nm
        {
                <data-type> element 1;
                <data-type> element 2;
                - - - - - - - - - - -
                - - - - - - - - - - -
                <data-type> element n;
        }inner_struct_var;
}outer_struct_var;
```

Example :

```
struct stud_Res
{
	int rno;
	char nm[50];
	char std[10];
	struct stud_subj
	{
		char subjnm[30];
		int marks;
	}subj;
}result;
```

In above example, the structure stud_Res consists of stud_subj which itself is a structure with two members. Structure stud_Res is called as 'outer structure' while stud_subj is called as 'inner structure.'

The members which are inside the inner structure can be accessed as follow :

result.subj.subjnm

result.subj.marks

Program to demonstrate nested structures.

```
#include <stdio.h>
#include <conio.h>
struct stud_Res
{
	int rno;
	char std[10];
	struct stud_Marks
	{
		char subj_nm[30];
		int subj_mark;
	}marks;
}result;
void main()
{
	clrscr();
	printf("\n\t Enter Roll Number : ");
	scanf("%d",&result.rno);
	printf("\n\t Enter Standard : ");
	scanf("%s",result.std);
	printf("\n\t Enter Subject Code : ");
```

```
        scanf("%s",result.marks.subj_nm);
        printf("\n\t Enter Marks : ");
        scanf("%d",&result.marks.subj_mark);
        printf("\n\n\t Roll Number : %d",result.rno);
        printf("\n\n\t Standard : %s",result.std);
        printf("\nSubject Code : %s",result.marks.subj_nm);
        printf("\n\n\t Marks : %d",result.marks.subj_mark);
        getch();
}
```

Output:

Enter roll number : 1
Enter standard :Btech
Enter subject code : GR11002
Enter marks : 63
Roll number :1
Standard :Btech
Subject code : GR11002
Marks : 63

34. Write a C program using nested structures to read 3 employees details with the following fields; emp-id, name, designation, address, da ,hra and calculate gross salary of each employee.

Ans:

```
#include<stdio.h>

struct employee

{

int emp_id;

char name[20];

char designation[10];

char address[20];

struct salary
```

```
{
float da;
float hra;
}sal;
}emp[3];
void main( )
{
int i;
printf("\t\t\t\t Enter 3 employees details");
grosssalary = 0;
for(i=0; i<3; i++)
{
scanf("%d",&emp[i].emp_id);
gets(emp[i].name);
gets(emp[i].designation);
gets(emp[i].address);
scanf("%f",&emp[i].sal.da);
scanf("%f",&emp[i].sal.hra);
grosssalary = grosssalary + emp[i].sal.da + emp[i].sal.hra;
}
printf("\n*******************************************");
printf("\n\t\t\t Employee details with gross salary:");
printf("\n*******************************************");
for(i=0; i<3; i++)
```

```
{
printf("%d",emp[i].emp_id);
puts(emp[i].name);
puts(emp[i].designation);
puts(emp[i].address);
printf("%f",emp[i].sal.da);
printf("%f",emp[i].sal.hra);
printf("%f",grosssalary);
}
getch( );
}
```

35. Distinguish between Arrays within Structures and Array of Structure with examples.

Ans:

Arrays within structure

It is also possible to declare an array as a member of structure, like declaring ordinary variables. For example to store marks of a student in three subjects then we can have the following definition of a structure.

```
struct student
{
char name [5];
int roll_number;
```

int marks [3];

float avg;

};

Then the initialization of the array marks done as follows,

struct student s1= {“ravi”, 34, {60,70,80}};

The values of the member marks array are referred as follows,

s1.marks [0] --> will refer the 0th element in the marks

s1.marks [1] --> will refer the 1st element in the marks

s1.marks [2] --> will refer the 2ndt element in the marks

Array of structure

An array is a collection of elements of same data type that are stored in contiguous memory locations. A structure is a collection of members of different data types stored in contiguous memory locations. An array of structures is an array in which each element is a structure. This concept is very helpful in representing multiple records of a file, where each record is a collection of dissimilar data items.

Ex:

An array of structures for structure *employee* can be declared as

struct employee emp[5];

Let’s take an example, to store the information of 5 employees, we can have the following structure definition and declaration,

```
struct employee
{
int empid;
char name[10];
float salary;
};
struct employee  emp[5];
```

Defines array called emp, which contains five elements. Each element is defined to be of type struct student.

For the student details, array of structures can be initialized as follows,

```
struct employee emp[5] = {{1,"ramu"25,20000},
        {2,"ravi",65000},
        {3,"tarun",82000},
        {4,"rupa",5000},
        {5,"deepa",27000}};
```

36. Write a C program using structure to create a library catalogue with the following fields: Access number, author's name, Title of the book, year of publication, publisher's name, and price.

Ans:

```
struct library
{
        intacc_no;
        char author[20];
        char title[10];
        intyear_pub;
        charname_pub[20];
        float price;
};
void main( )
{
        struct  library  lib;
        printf("Input values are:\n");
        scanf("%d  %s  %s  %d  %s%f",  &lib.acc_no,   lib.author,  lib.title,  &lib.year_pub,
        lib.name_pub, &lib.price);
        printf("Output values are:\n\n");
        printf("Access number = %d\n Author name = %s\n
```

```
        Book Title = %s\n  Year of publication =  %d \n Name of publication = %s\n Price = %f ",
         lib.acc_no,  lib.author,  lib.title, lib.year_pub,  lib.name_pub,  lib.price);
        getch( );
}
```

37. What is self referential structure? Explain through example.

Ans:

Self-referential structure

A structure definition which includes at least one member as a pointer to the same structure is known as self-referential structure.

It can be linked together to form useful data structures such as lists, queues, stacks and trees.

It is terminated with a NULL pointer (0).

The syntax for using the self referential structure as follows,

```
struct tag_name
{
Type1 member1;
Type2 member2;
……….
struct tag_name *next;
};
```

Ex:-

```
struct node
{
int data;
struct node *next;
```

} n1, n2;

38. Explain unions in C language? Differentiate structures and unions.

Ans:

A union is one of the derived data type. Union is a collection of variables referred under a single name. The syntax, declaration and use of union is similar to the structure but its functionality is different.

The general format or syntax of a union definition is as follows,

Syntax:

```
union union_name
{
        <data-type> element 1;
        <data-type> element 2;
        ………………
}union_variable;
```

Example:

```
union techno
{
int comp_id;
char nm;
float sal;
}tch;
```

A union variable can be declared same way as structure variable.

union tag_name var1, var2...;

A union definition and variable declaration can be done by using any one of the following

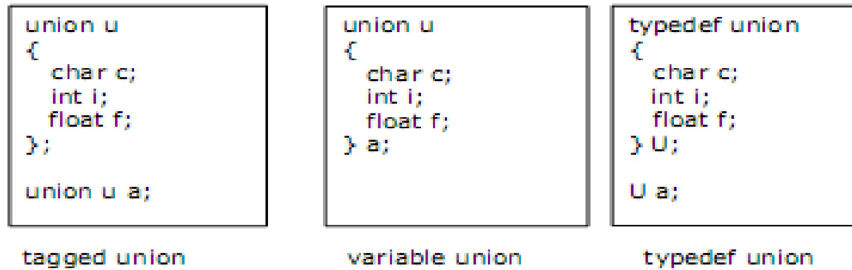


Figure 5.7 Types of Union Definitions

We can access various members of the union as mentioned below, and memory organization is shown below,

a.c

a.i

a.f

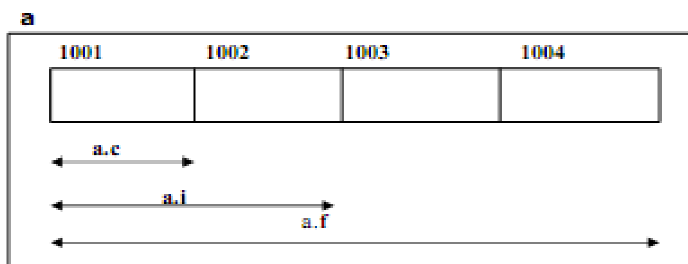


Figure 5.8: Memory Organization Union

In the above declaration, the member f requires 4 bytes which is the largest among the members. Figure 5.8 shows how all the three variables share the same address. The size of the union here is 4 bytes.

A union creates a storage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supersedes the previous members' value.

Difference between structure and union:-

	Structure	**Union**
(i) Keyword	struct	union
(ii) Definition	A structure is a collection of logically related elements, possibly of I different types, having a single name.	A union is a collection of logically related elements, possibly of different types, having a single name, shares single memory location.

(iii) Declaration	struct tag_name
	{
	type1 member1;
	type1 member2;
	……………
	};
	struct tag_name var;
		union tag_name
		{
		type1 member1;
		type1 member2;
		……………
		};
		union tag_name var;

(iv) Initialization	Same.	Same.
(v) Accessing	Accessed by specifying Structure variablename . member	Accessed by specifying unionvariablename. member

	name	name
(vi)Memory Allocation	Each member of the structure occupies unique location, stored in contiguous locations.	Memory is allocated by considering the size of largest member. All the members share the common location
(vii) Size	Size of the structure depends on the type of members, adding size of all members. sizeof (st_var);	Size is given by the size of largest member storage. sizeof(un_variable)
(viii) Using pointers	Structure members can be accessed by using dereferencing operator dot and selection operator(->)	same as structure.
	We can have arrays as a member of structures. All members can be accessed at a time.	We can have array as a member of union. Only one member can be accessed at a time.
	Nesting of structures is possible.	same.

	It is possible structue with in union as a member.	It is possible union may contain structure as a member