C PROGRAMMING

CODE WITH HARRY(20-40)NOTES

Recursive Functions: Recursion In C: C Tutorial In Hindi #21

Recursive Functions :

Recursive functions or Recursion is a process when a function calls a copy of itself to work on smaller problems.

Recursion is the process in which a function calls itself directly or indirectly. And the corresponding function or function which calls itself is called as recursive function.

- *Any function which calls itself is called recursive function.*
- *This makes life of programmer easy by dividing complex problem into simple or easier problems.*
- *A termination condition is imposed on such functions to stop them executing copies of themselves forever or infinitely.*
- *Any problem which can be solved recursively can also be solved iteratively.*

Recursions are used to solve tower of Hanoi, Fibonacci series, factorial finding etc.

Base condition in recursion :

- *The case at which the function doesn't recur is called the base case.*

So, that's all about Recursions in C.

Code as described/written in the video

```
#include <stdio.h>
int factorial(int number){

    if (number == 1 || number == 0)
    {
        return 1;
    }
    else
```

```
   {
      return (number * factorial(number - 1));
   }}
int main(){
   int num;
   printf("Enter the number you want the factorial of\n");
   scanf("%d", &num);
   printf("The factorial of %d is %d\n", num, factorial(num));

   return 0;}
```

Copy

Recursive Case :

- *The instances where the function keeps calling itself to perform a subtask i.e. solving problem by dividing it in small parts, is called the recursive case.*

Now let me summarize whole recursions. So Recursion is a process in which any function keeps calling itself till any termination condition is satisfied and in simple words you can think Recursions as same like iteration because in both of them repetition occurs till any condition is satisfied or becomes false.

And the most important thing during using recursions is it's termination condition because most of time the condition given in recursive function is wrong and because of that the function is executed infinitely or for forever.

```
#include<stdio.h>int factorial(int number){
          if (number ==1 || number == 0)
          {
                    return 1;
          }
          else
          {
                    return number*factorial(number-1);          //Recursion of Function
          }}
int main(){
          int num;
          printf("Enter a no. :");
```

```
        scanf(%d, &num);
        printf("\nThe factorial of %d is %d", num, factorial(num));

        return 0;}
```

Arrays In C: C Tutorial In Hindi #23

Arrays :

An array is a collection of data items of the same data type. And it is also known as subscript variable.

- *Items are stored at contiguous memory locations in Arrays.*
- *It can also store the collection of derived data types such as pointers, structures etc.*
- *The C Language places no limits on the number of dimensions in an array i.e. we can create any no. of dimension array E.g. 2d array, 3d array etc.*

Most Commonly used dimensions of Array :

- *A one-dimensional array is like a list.*
- *A two-dimensional array is like a table.*

*Some texts refer to **one-dimensional arrays as vectors** and **two-dimensional arrays as matrices** and use the general term **Arrays** when the no. of dimensions is unspecified or unimportant.*

So that's all about Arrays in C Language.

Code as described/written in the video

```
#include <stdio.h>
int main(){
   // printf("Hello World\n");
   int marks[2][4] = {{45, 234, 2, 3},
                      {3, 2, 3, 3}};

   // for(int i = 0; i < 4; i++)
```

```
  // {
  //     printf("Enter the value of %d element of the array\n", i);
  //     scanf("%d", &marks[i]);
  // }

  for (int i = 0; i < 2; i++)
  {
      for (int j = 0; j < 4; j++)
      {
          /* code */

          // printf("The value of %d, %d element of the array is %d\n", i, j,
marks[i][j]);
          printf("%d ", marks[i][j]);
      }
      printf("\n");
  }

  // marks[0] = 34;
  // printf("Marks of student 1 is %d\n", marks[0]);
  // marks[0] = 4;
  // marks[1] = 24;
  // marks[2] = 34;
  // marks[3] = 44;
  // printf("Marks of student 1 is %d\n", marks[0]);
  return 0;}
```
Copy

Why do we need Arrays ?

Code that use arrays for managing large no. of same type variables is more organized and readable.

- *Arrays allow us to create many variables by just a single line. It means there is no need to create or specify each and every variable.*

Advantage of Arrays :

- *It is used to represent multiple data items of same type by using only single name.*
- *Accessing an item in a given array is very fast.*

Properties of Array :

- *Data in an array is stored in contiguous memory locations in RAM.*
- *Each element of an array is of same size i.e. their data types are same so memory consumed by each is also same.*
- *Any element of the array with given index can be accessed very quickly by using its address which can be calculated using the base address and the index.*

Index No. – It is the special type of no. which allows us to access variables of Arrays i.e. index no. provides a method to access each element of an array in a program.

Example for One-dimensional Array :

```
#include<stdio.h>
int main(){
          //One dimensional array
          int marks[10], sum=0;
          printf("Enter marks of 10 students : \n\n");
          for (int i=0;i<=9,i++)
          {
          printf("Marks of %d student : ", i+1);
          scanf("%d", &marks[i]);
          sum += marks[i]
          }

          int average = sum/10;
          printf("\nThe average marks of 10 students are %d", average);
          return 0;}
```
Copy

Example for Two-Dimensional Array :

```
#include<stdio.h>
int main(){

          //Two dimensional array
          int Matrice[3][3;
          printf("This Program will print no. from 1-9 in matrice form : \n\n");
          for (int i=0;i<3,i++)
          {
                    for (int j=0;j<3;j++)
                    {
                              printf("Enter no. (1-9) : ");
                              scanf("%d", &Matrice[i][j]);
                    }
          }
          printf("\n\n\n");
          for (int i=0;i<3;i++)
          {
                    for (int j=0;j<3;j++)
                    {
                              printf("%d\t", Matrice[i][j]);
                    }
                    printf("\n");
          }

          printf("\n\n\n So that's the matrice form of no from 1-9");
          return 0;}
```
Copy

Now let me tell you in brief about Arrays :

So as we already seen that array is a collection of data items and generally we use arrays when we have to make lots of variables at that time instead of creating each variable, so we can easily declare one array and can access it's elements using it's index no. and can do whatever we wish to do with that data.

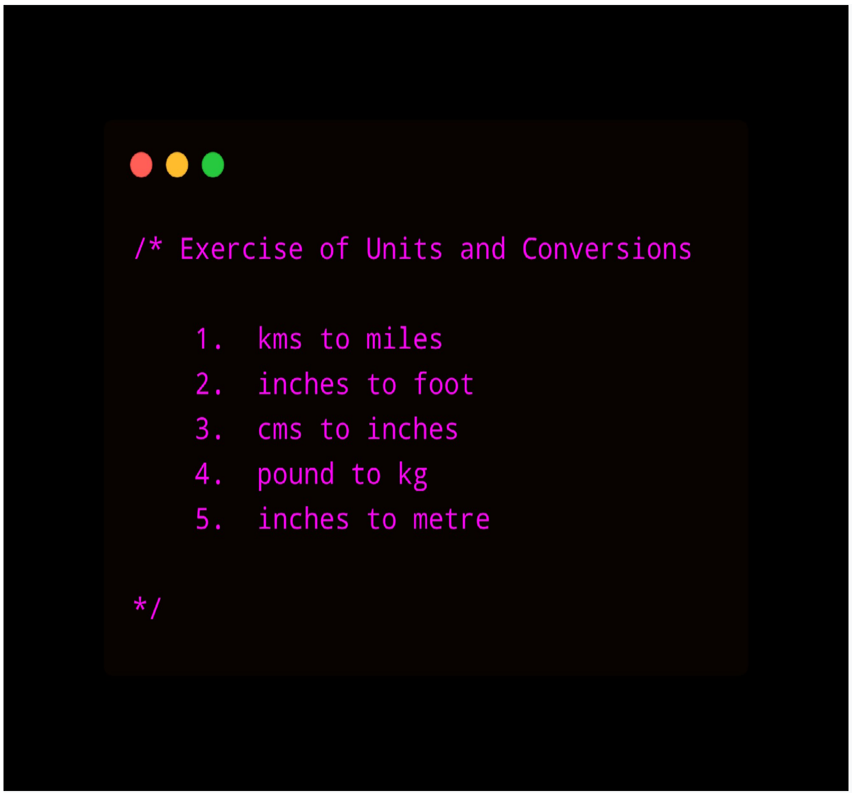
So, that's the use of arrays. And not only this we should have a good command over arrays because it is one of the most important topic in C language. Generally if we access each element of array at that time we use loops for fast calculations or operations.

And yeah one more thing i.e. you must have seen some memes on programmers i.e. programmers count from 0 not from 1. So the reason behind these memes is this only i.e. Arrays. As in arrays we do indexing from 0 onwards na that's why these memes are created for programmers. :)

Exercise-2 (Solution)

So, Guys here is the solution of exercise 2 which I gave you some days ago in our C course.

Question: You have to make a program which can convert the following units into other system units.



```
/* Exercise of Units and Conversions

    1.  kms to miles
    2.  inches to foot
    3.  cms to inches
    4.  pound to kg
    5.  inches to metre

*/
```

Hint : In this program you can use switch case statements to choose between any of the conversion and after that you can use simple logic to make program which can convert units from one system to another.

Solution: So, here is the solution of that exercise ..

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

   char input;
  float kmsToMiles = 0.621371;
  float inchesToFoot = 0.0833333;
  float cmsToInches = 0.393701;
  float poundToKgs = 0.453592;
  float inchesToMeters = 0.0254;
  float first, second;

  while (1)
  {
     printf("Enter the input character. q to quit\n       1. kms to miles\n       2. inches to foot\n       3. cms to inches\n       4. pound to kgs\n       5. inches to meters\n");

     scanf(" %c", &input);
  //    printf("The character is '%c'", input);
     switch (input)
     {
     case 'q':
      printf("Quitting the program...");
      goto end;
      break;

      case '1':
      printf("Enter quantity in terms of first unit\n");
      scanf("%f", &first);
      second = first * kmsToMiles;
      printf("%.2f Kms is equal to %.2f Miles\n\n\n", first, second);
      break;

      case '2':
      printf("Enter quantity in terms of first unit\n");
      scanf("%f", &first);
      second = first * inchesToFoot;
      printf("%f Inches is equal to %f Foot\n", first, second);
      break;


      case '3':
      printf("Enter quantity in terms of first unit\n");
      scanf("%f", &first);
      second = first * cmsToInches;
      printf("%f Cms is equal to %f Inches\n", first, second);
      break;

      case '4':
      printf("Enter quantity in terms of first unit\n");
      scanf("%f", &first);
      second = first * poundToKgs;
      printf("%f Pounds is equal to Kgs %f \n", first, second);
      break;

      case '5':
      printf("Enter quantity in terms of first unit\n");
      scanf("%f", &first);
      second = first * inchesToMeters;
      printf("%f inches is equal to %f meters \n", first, second);
      break;

     default:
     printf("In default now");
        break;
     }
  }
  end:

   return 0;}
```
Copy

So, Hope You have understood this exercise and for better explanation Must check above video.

Pointers :

Before discussing about pointers let me tell you when we define and initialize a variable at that time we come to know about these things :

- *Memory block i.e. variable get some space in RAM and we can think of that as a block.*
- *Name of memory block or Variable's name*
- *Content of that block i.e. value in that variable*
- *Address of memory block i.e. unique address which allows us to access that variable.*

We can print address of any variable by using printf function as :

printf("%d",&variable_name);

Pointer :

- *Pointer is a variable that contains address of another variable. It means it is a variable but this variable contains address or memory address of any other variable.*
- *It can be of type int, char, array, function, or any other pointer.*
- *Its size depends on architecture.*
- *Pointers in C Language can be declared using *(asterisk symbol).*

```
#include<stdio.h>
int main(){
        int x=5;
        int *a =&x;

        printf("%d\n",&x );
        printf("%d",a );
        return 0;}
```
Copy

After checking above example you will understand the basics of pointers.

So, pointers are nothing just a variable which stores the address of other variables and by using pointers we can access other variables too and can even manipulate them.

Now let’s see about some of the operators which we use with Pointers :

- *Address of Operator (&) :*

· *It is a unary operator.*

· *Operand must be the name of the variable.*

· *& operator gives address no. of variable.*

· *& is also known as “Referencing Operator”.*

- *Indirection Operator :*

· ** is indirection operator.*

· *It is also known as “Dereferencing Operator”.*

· *It is a unary operator.*

· *It takes address as an argument.*

· ** returns the content/container whose address is it’s argument.*

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

```
int main(){
        int a=5;

        printf("%d\n",&a );
        printf("%d",a );
        return 0;}
```
Copy

In above example you will see that we printed variable ‘a’ address and its value. So in second printf statement you can see we used two unary operators i.e. * and "&" operator. As we know unary operator associativity is from right to left so first of all & of operator will be resolved and then the address of variable ‘a’ will be the argument of * operator. That’s how we can print values or can use these Address of Operator and Indirection Operator.

```
#include<stdio.h>
int main(){
        int x,*a;
        return 0}
```
Copy

When * (indirection operator) is written before any pointer variable like (*a) then that whole variable resembles like original variable i.e. the variable to whom the pointer is pointing.

Or,

We can say that *a pointer becomes the variable whose address is in a pointer.

Example 1 :

```
#include<stdio.h>
void ptrfunc(int *ptr){
        *ptr=8;}
int main(){
```

```
          int x=5;

          printf("x=%d\n",x );
          printf(&x);
          printf("x=%d",x );
          return 0;}
```
Copy

Example 2 :

```
#include<stdio.h>

int main(){
          printf("Pointer Basics\n");
          int a =76;
          int *ptra=&a
          int *ptr2= NULL;

          printf("The Address of pointer is %p\n", &ptra );
          printf("The Address of a is %p\n," &a);
          printf("The Address of a is %p\n",ptra );
          printf("The Address of some garbage is %p\n",ptr2);
          printf("The Value of a is %d\n", *ptra );
          printf("The Value of a is %d\n", *a );
          return 0;}
```
Copy

Null Pointer :

- *A pointer that is not assigned any value but NULL is known as NULL pointer.*
- *In computer programming NULL pointer is a pointer that does not point to any object, variable or function.*
- *We can use it to initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.*
- *int *ptr = NULL;*

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

```
int main(){
          printf("Pointer Basics\n");
          int a =5;
          int *p=NULL;


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

          return 0;}
```
Copy

Uses of Pointers:

- *Dynamic Memory Allocation*
- *Arrays, Functions and Structures*
- *Return multiple values from a function*
- *Pointer reduces the code and improves the performance*

That's all about basics of Pointers. In next tutorial we will discuss about Pointers Arithmetic i.e. how to manipulate pointers values.

Code as described/written in the video

```
#include <stdio.h>
int main(){
   printf("Lets learn about pointers\n");
   int a=76;
   int *ptra = &a;
   int *ptr2 = NULL;
   printf("The address of pointer to a is %p\n", &ptra);
   printf("The address of a is %p\n", &a);
   printf("The address of a is %p\n", ptra);
   printf("The address of some garbage is %p\n", ptr2);
   printf("The value of a is %d\n", *ptra);
   printf("The value of a is %d\n", a);
   return 0;}
```

Arrays And Pointer Arithmetic In C: C Tutorial In Hindi #27

In this tutorial we will learn about, arrays and pointer arithmetic

There are four arithmetic operators that can be used on Pointers :

- ++
- --
- +
- -

Pointers arithmetic is not same as normal arithmetic i.e. if you want to add 1 to any no. then you will get that no. after adding 1 in it. But in Pointers it is a bit different as:

Base Address : *The first byte address of any variable is known as base address.*

It means suppose you have int type variable then it's size in my architecture is 4 byte (It may vary in yours) then 4 consecutive blocks will be created in RAM. So when we will point it with any pointer variable at that time the address of 1 block of 4 will come in that pointer variable. It means base address is the first block address of any data type variable.

Pointers Arithmetic :

- *We can't add, multiply or divide two addresses. (Subtraction is possible)*

E.g:

*int a, b, *p, *q;*

&a&b;   // Not Possible*

&a+&b;   // Not Possible

*p*q;   // Not Possible*

- *We can't multiply & divide an address with integer value.*

E.g :

*&a*5;   // Not Possible*

p/5;   // Not Possible

- *We can add or subtract integer to/from an address.*

E.g :

int a,b;   // Let's Suppose &a is 1000

*int *p, *q;*

p=&a;

q=&b;

p+1;   // It is possible [1000+1 = 1004]

Now you may think why we got 1004 as an output after adding 1 in 1000. So as we know a is an integer. So it'll take 4 bytes of memory & we know pointer always takes base address, but it access other bytes also by itself only.

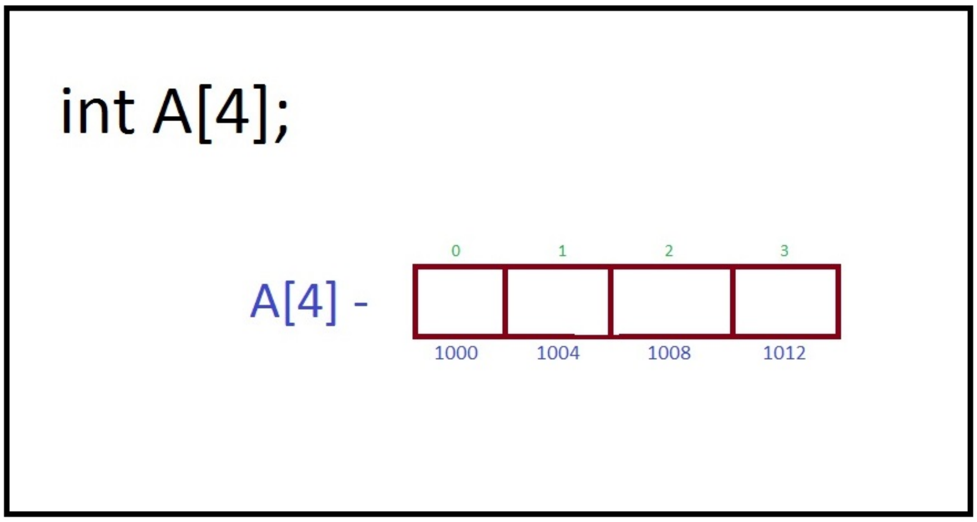
So, when we will add 1000+1 then from 1000 to 1003 all address are allocated to int a. So the next block of memory is after 1003 i.e. 1004. So that's how Pointers arithmetic is done.

To Find arithmetic (+,-) of any pointer :

*Pointer + n = pointer + size of (type of pointer)*n*

Arrays and Pointers :

- *Array always consumes memory location in contiguous fashion/manner.*



Suppose address of A[0] is 1000 then the address of next variable in Array will be 1004 (if int type) because we know pointer always takes the base address and when it is incremented or decremented then it contain the address of next block but not of very next byte.

- *Pointer when incremented always point to next block of its own type.*

When we add or increment in pointer than literal addition doesn't take place instead of that it start pointing next block.

E.g. :

int age[50];

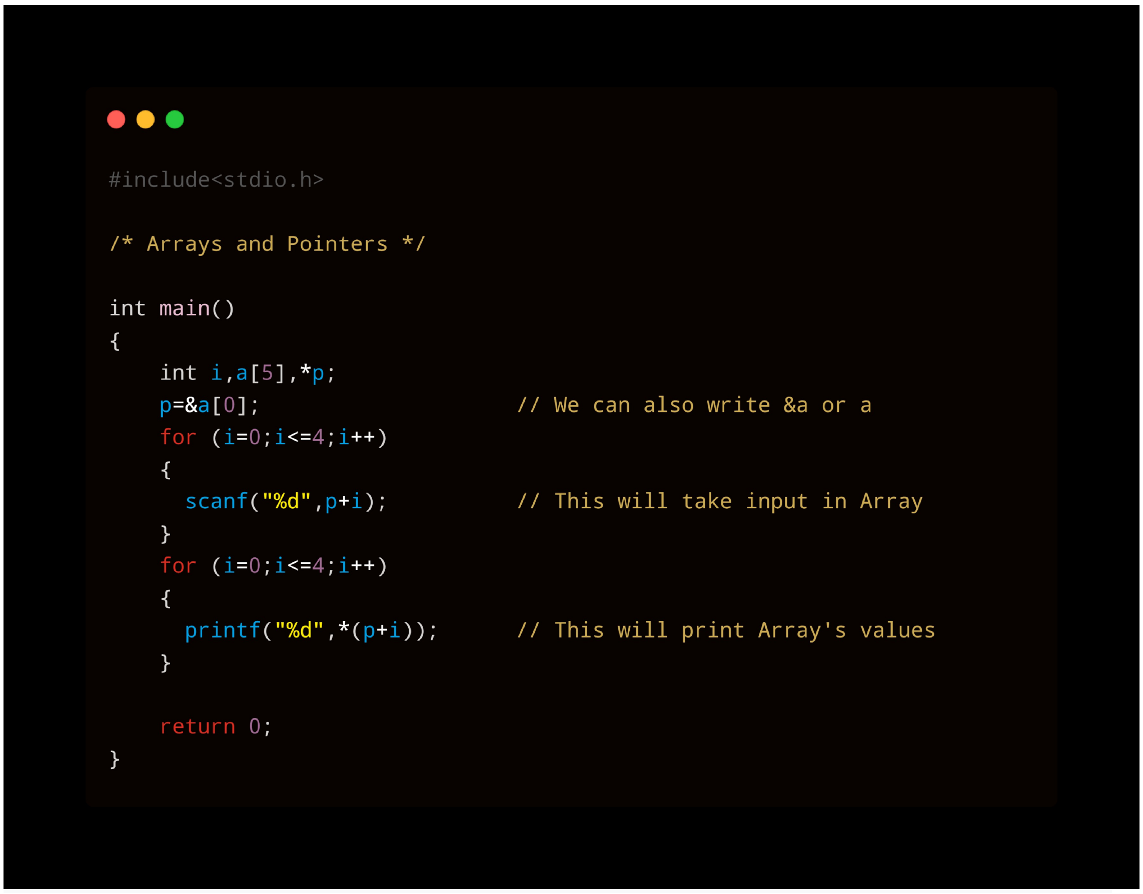
*int *p;*

p= age;

It means, Either we write &age or &age[0] or age. They all means that the first element of array or index 0 is being used or pointed by pointer.

So to access the other Array variables we can simply do increment or decrements in pointer variable an can access other Array variables too.

Like age[1] = p+1;



```
#include<stdio.h>

/* Arrays and Pointers */

int main()
{
    int i,a[5],*p;
    p=&a[0];                    // We can also write &a or a
    for (i=0;i<=4;i++)
    {
      scanf("%d",p+i);          // This will take input in Array
    }
    for (i=0;i<=4;i++)
    {
      printf("%d",*(p+i));      // This will print Array's values
    }

    return 0;
}
```

In the above example we have used p+I instead of &a[i] because they both mean the same.

That's how we can do Pointers Arithmetic and can use Arrays and Pointers in C Language.

Code as described/written in the video

```
#include <stdio.h>int main(){
    // char a = '3';
    // char* ptra = &a;
    // printf("%d\n", ptra);
    // ptra--;
    // printf("%d\n", ptra);
    // printf("%d", ptra-2);
    int arr[] = {311,52,3,4,5,6,67};
    int* arrayptr = arr;
    printf("Value at position 3 of the array is %d\n", arr[3]);
    printf("The address of first element of the array is %d \n", &arr[0]);
    printf("The address of first element of the array is %d \n", arr);
    printf("The address of second element of the array is %d \n", &arr[1]);
    printf("The address of second element of the array is %d \n", arr + 1);
    printf("The address of third element of the array is %d \n", &arr[2]);
    printf("The address of third element of the array is %d \n", arr + 2);
    // arr--; // this line will throw an error


    printf("The value at address of first element of the array is %d \n",
*(&arr[0]));
    printf("The value at address of first element of the array is %d \n", arr[0]);
    printf("The value at address of first element of the array is %d \n", *(arr));
    printf("The value at address of second element of the array is %d \n",
*(&arr[1]));
    printf("The value at address of second element of the array is %d \n", arr[1]);
    printf("The value at address of second element of the array is %d \n", *(arr
+ 1));


    return 0;}
```
Copy

← PreviousNext →

Exercise-3 Solution :

Question : Write a program which produces Fibonacci Series of numbers using both Recursive and Iterative approach ?

Solution :

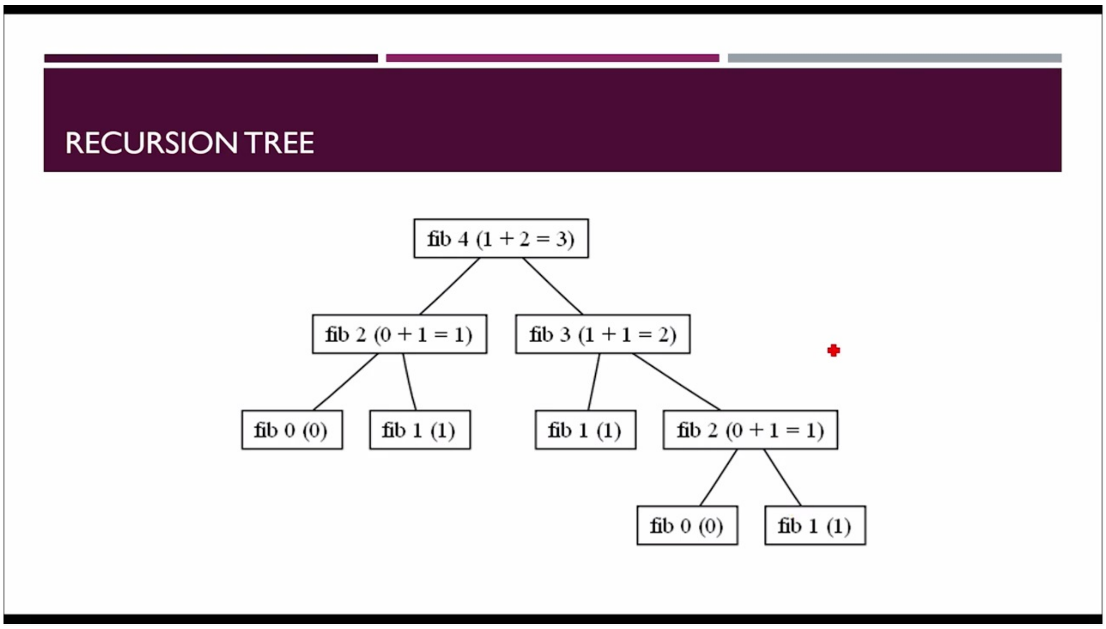
```
#include <stdio.h>

int fib_recursive(int n){
    if(n==1 || n==2){
        return n-1;
    }
    else{
        return fib_recursive(n-1) + fib_recursive(n-2);
    }   }

int fib_iterative(int n){
    int a = 0;
    int b = 1;

    for (int i = 0; i < n-1; i++)
    {
        b = a+b; //1 for iteration 1
        a = b-a; //1 for iteration 1

    }

    return a;}
int main(){
    int number;
    printf("Enter the index to get fibonacci series\n");
    scanf("%d", &number);
    printf("The value of fibonacci number at position no %d using iterative approach is %d\n",number, fib_iterative(number));
    printf("The value of fibonacci number at position no %d using recursive approach is %d\n", number, fib_recursive(number));
    return 0;}
```

Copy

← PreviousNext →

Why is Recursion not always good ?

Iterative approach takes less time as in case of Fibonacci Series it doesn't call same no. again and again but in Recursive approach it calls same no. again and again i.e. many times.



RECURSION TREE

- *Recursion is a good approach when it comes to problem solving.*
- *However, programmer needs to evaluate the need and impact of using recursive/iterative approach while solving a particular problem.*
- *In case of Factorial calculation, recursion saved a lot of lines of code.*
- *However in case of Fibonacci Series recursive approach called many functions again and again causing time waste.*

So, programmer must have to evaluate which approach he/she should use to solve any particular problem.

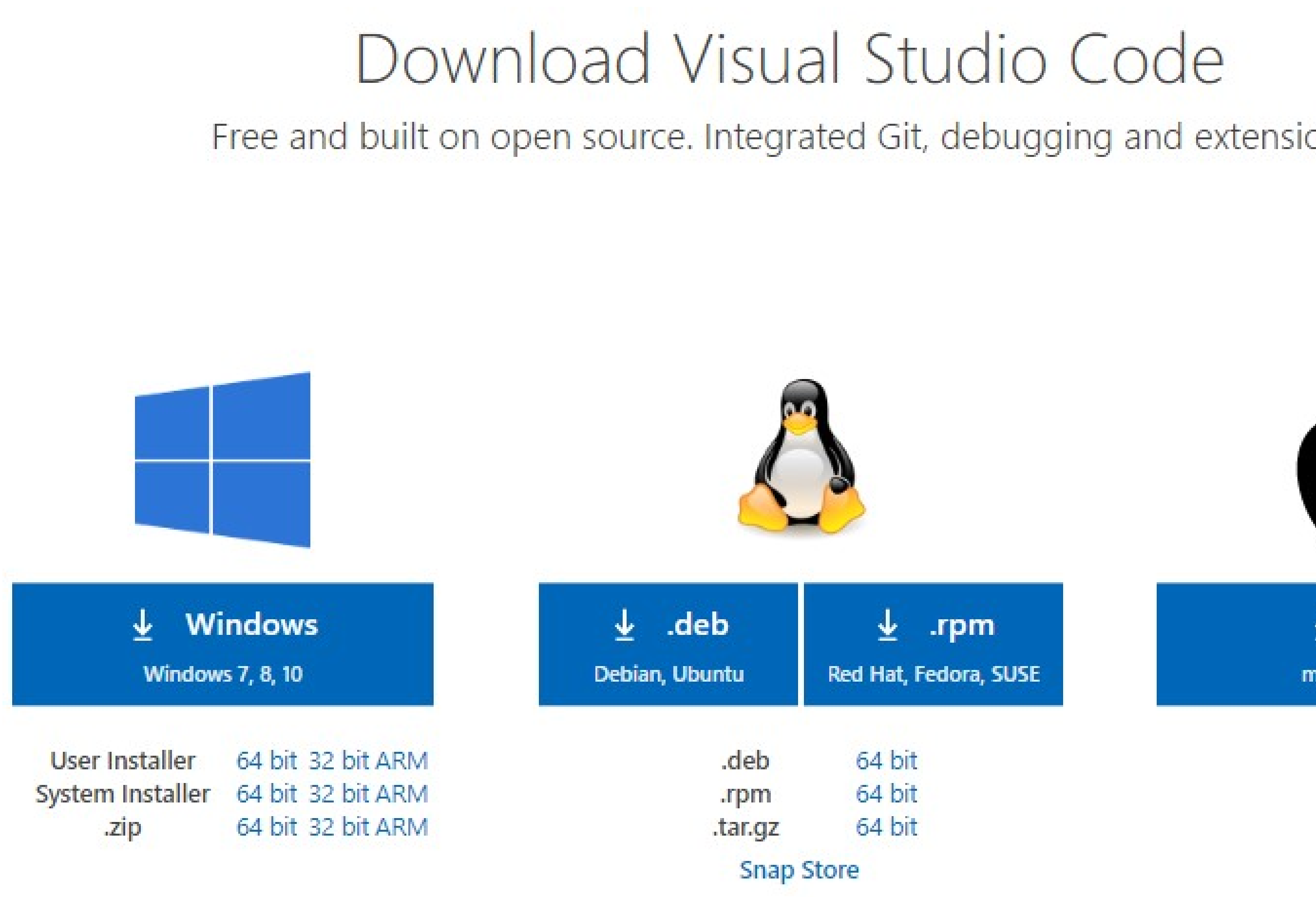
Code as described/written in the video

Copy

← PreviousNext →

Call by Value & Call By Reference In C: C Tutorial In Hindi #31

*To run a **C program**, we need an IDE's like Visual Studio Code or Codeblocks. For this series, we are using Virtual Studio Code (VS Code). **Visual Studio Code** is a fast source code editor and provides the tools that a developer needs for a quick code-build-debug cycle. To download VS Code, click on **Download Virtual Studio Code** . For guidance, check the tutorial **Install & Configure VS Code***



Download Visual Studio Code

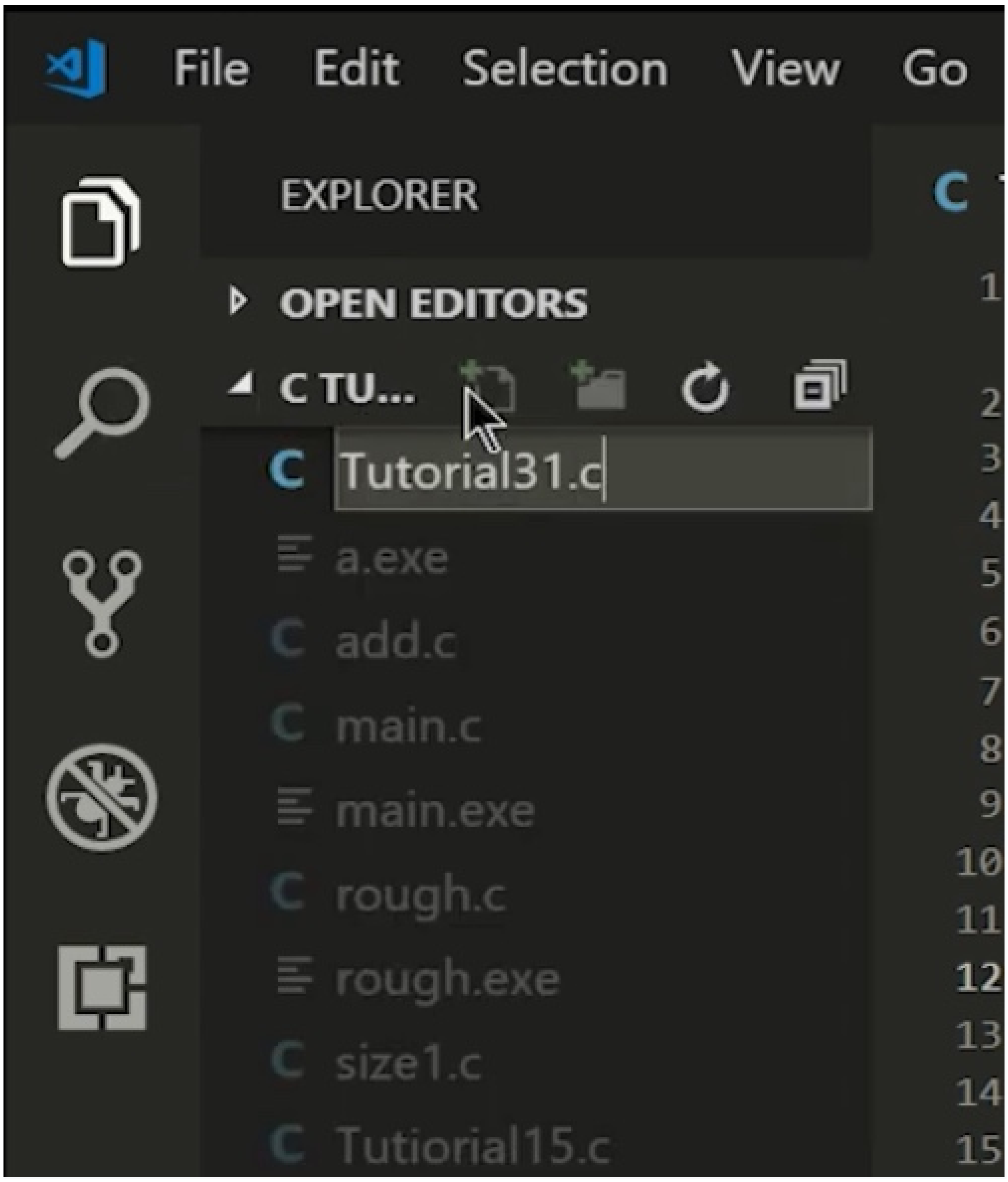
Free and built on open source. Integrated Git, debugging and extensio

User Installer	64 bit	32 bit	ARM
System Installer	64 bit	32 bit	ARM
.zip	64 bit	32 bit	ARM

.deb	64 bit
.rpm	64 bit
.tar.gz	64 bit

Snap Store

*Now open VS Code and create a file with name **"tutorial31.c"***



Function Calls In C programming:-

*In today's tutorial, we will explore the function calls. By now we are well familiar with how to use functions in C. But, if we observe carefully, whenever we called a function and passed something to it, we have always passed the 'values' of variables to the called function. Such function calls are called 'call by value'. Similarly, we have also learned that variables are stored somewhere in memory. So instead of passing the value of a variable, can we pass the location an address of the variable to a function. Such function calls are called '**call by reference**'.*

*This call by reference functions needs the knowledge of a concept called ‘**pointers**’. It is the use of pointers that makes the C programming an excellent language.*

What are the Pointers?

A pointer in C is a variable that allocates memory dynamically. It holds the value that is the address of another variable, i.e., direct address of the memory location. The syntax of a pointer variable declaration is

type *variable_name;

```
int *ptr;   /* This is the pointer to an integer */
```

Copy
*Here, * is used to denote the pointer variable, and to return the address of the variable, we use operator **'&'**.*

Let us now get back to function calls that are the main focus of this tutorial. The function calls are of two types. In function calls, one of the important concepts is the formal and actual parameters.

- *Formal Parameter: Formal parameters are the local variable which are assigned values from the arguments when the function is called.*
- *Actual Parameter: When a function is called, the values(expression) that are passed in the call are called arguments or actual parameters.*

There are two ways we generally pass the arguments to functions:

***(a)	Call by values***

In this method, the ‘call by value’ of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. In this function call, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function. Actual and formal arguments will be created in a different memory location. The following program is the example of ‘Call by Value’.

```
void swap(int x, int y){int temp;
temp=x;
x=y;
y=temp;}void main(){ int r=10, v=20; swap(r, v);  // passing value to
functionprintf("\nValue of r: %d",r);printf("\nValue of v: %d",v);}
```

Copy

(b) Call by reference

In this method, the addresses of actual arguments in the calling function are copied into formal arguments of the called function. This means that using these addresses we could access the actual arguments and hence we would be able to manipulate them. The changes that are made to the parameter affect the argument. This is because the address is used to access the actual argument. Formal and actual arguments will be created in the same memory location. The following program is the example of 'Call by Reference'.

```
void swap(int *x, int *y){int temp;
 temp=*x;*x=*y;*y=temp;}void main(){ int r=10, v=20; swap(&r, &v);  // passing
value to functionprintf("\nValue of r: %d",r);printf("\nValue of v: %d",v)
```

Copy

Usually, we use call by value function call. This means that in general, we cannot alter the actual arguments. But if desired, we can use call by reference for that purpose. We can make a function return more than one value at a time by using call by reference which is not possible ordinarily.

Conclusions

From the above discussion, we can draw the following conclusions:

- *If we want the value of an actual argument to not get changed in the function being called, pass the actual argument by value.*
- *If we want the value of an actual argument should get changed in the function being called, then pass the actual argument by reference.*
- *If a function is to be made to return more than one value at a time then use call by reference method for that purpose.*

Code as described/written in the video

```
#include <stdio.h>
void changeValue(int* address){
    *address = 37565;}
```

```
int main(){
    int a = 34, b =56;
    printf("The value of a now is %d\n", a);
    changeValue(&a);
    printf("The value of a now is %d\n", a);
    return 0;}

// Quick Quiz:// Given two numbers a and b,  add them then subtract them and assign them to a and b using call by reference.
// a = 4// b = 3
// after running the function, the values of a and b should be:// a = 7// b = 1
```

Copy

Passing Arrays As Function Arguments: C Tutorial In Hindi #32

With the advancement in technology, the developers are also providing us the modern IDE's like Virtual Studio Code or CodeBlocks. ***Visual Studio Code*** *is a fast source code editor, perfect for day-to-day* ***use****. It supports for hundreds of languages, in which one of them is C.For this C programming series, we are using VS Code. To download VS Code, click on* ***Download Virtual Studio Code*** *. For guidance, check the tutorial* ***Install & Configure VS Code***

Now open VS Code and create a file with name ***"tutorial32.c"***

Just like variables, array can also be passed to a function as an argument. In this tutorial, we will learn how to pass the array to a function using ***call by value*** *and* ***call by reference*** *methods (using pointers). We have already discussed how to use function calls in the previous* ***tutorial#31***

In ***C programming****, there are various problems which requires passing more than one variable of the same type to a function. For example, consider a function in which we have to pass the marks of 70 students. Such a function requires 70 numbers to be passed as the actual parameters from the main function. Here, instead of declaring 70 different numbers and then passing them as an argument into the function, we can declare and initialize an array and pass that into the function. This will resolve all the complexity and make the code easy to read.*

There are two ways of passing array to a function as argument:

Declaring Function with array as a parameter

We can pass the one dimentional and multidimensional array in function as an argument. There are multiple ways to pass one-dimensional or two-dimensional arrays as arguments in function. We pass the array to a function to make it accessible within the function. When we pass an entire array to a function, then the function can access all the elemnts of the array. Single array elements can also be passed as arguments, it could be a sized or unsized array. This can be done in the same way as we pass variables to a function. Following are the syntax of passing array as an argument.

- *Formal parameters as an unsized array*

```
void myfunc (int arr[]) {}
```

Copy

- *Formal parameters as a sized array*

```
void myfunc (int arr [100]) {}
```

Copy

- *Formal parameters as a two dimensional array.*

```
void myfunc (int arr [3][3]) {}
```

Copy

Example:-

```
int sum(int arr[]) {int sum_of_array=0;for (int i = 0; i<4; ++i) {
sum_of_array += arr[i]; }return sum_of_array; }
int main() {   int result, array[] = {23,33,44,55};
result = sum(array);     printf("Result = %d", result);   return 0;}
```

Copy

Declaring function with pointer in the parameter:-

When we pass the address of an array while calling a function then we are using call by reference function call. When we pass an address as an argument in the function, the pointer in the function receives the address of the array.

```
void myfunc (int *ptr) {}
```

Copy

Example:-

```
void display(int *ptr) {
 printf("%d", *ptr);}int main() {
  int arr[] = {1, 2, 3, 4};
  for (int i=0; i<4; i++) {
  display(&arr[i]);
 }return 0;
```

Copy

***Remember that**, when we make a change in array using pointers in the functions, the actual array will also be affected. For Example, the array element arr[1]=100, and in function we write *(ptr+1)=200, then the element which is store at arr[1] becomes equal to 200.*

So, in this tutorial we have learned about two methods to pass the array as an argument in function. The first way is the most widely used technique that is declaring blank subscript notation [ ], and the second way is basically a general method that includes the use of the concept of a pointer.

Code as described/written in the video

```
#include <stdio.h>
int func1(int array[]){
    for (int i = 0; i < 4; i++)
    {
        printf("The value at %d is %d\n", i, array[i]);
    }
    // array[0] = 382; // Very important point that if you change any value here, it gets reflected in main()
    return 0;}

void func2(int *ptr){
    for (int i = 0; i < 4; i++)
    {
        printf("The value at %d is %d\n", i, *(ptr + i));
    }
    *(ptr + 2) = 6534;}

void func3(int arr[2][2]){
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            printf("The value at %d, %d is %d\n", i, j, arr[i][j]);
        }
    }}

int main(){
    int arr[][2] = {{2, 13}, {9, 3}};
    // printf("The value at index 0 is %d\n", arr[0]);
    // func1(arr);
    // printf("The value at index 0 is %d\n", arr[0]);
    // func2(arr);
    // func2(arr);
    func3(arr);
    return 0;}
```
Copy

← PreviousNext →

Star Pattern In C - Exercise 4 Solution: C Tutorial In Hindi #33

Hope you enjoyed solving the exercise. If you havn't seen the Exercise 4 tutorial, then do not worry as I have mentioned the question below for your convenience. Just give it a read and then you may check the solution:

Question Statement:

*The task you have to perform is "**Star Pattern In C**". Printing different pattern is one of the most popular programming exercise for beginners, it not only improves the programming skills but also the thinking ability of the person. Following are the instructions you have to follow:*

Instructions:-

Take input from the user and ask the user to choose 0 for the triangular star pattern and 1 for the reserved triangular star pattern. When the user entered the number, your program should print the pattern accordingly.

Hint: Ask the user to enter the number of rows and then use nested for loop to print rows and columns of triangular star pattern.

Triangular star pattern:

```
*
**
***
****
```

Copy

Reserved Triangular Star Pattern:

```
****
***
**
*
```

Copy

Code as described/written in the video

```
#include <stdio.h>
void starPattern(int rows){
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j <= i; j++)
        {
            printf("*");
        }
        printf("\n");
    }}
void reverseStarPattern(int rows){
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j <= rows - i - 1; j++)
        {
            printf("*");
        }
        printf("\n");
    }}
int main(){
    int rows, type;
    printf("Enter 0 for star pattern and 1 for reversed star pattern\n");
    scanf("%d", &type);
    printf("How many rows do you want?\n");
    scanf("%d", &rows);
    switch (type)
    {
    case 0:
        starPattern(rows);
        break;

    case 1:
        reverseStarPattern(rows);
        break;

    default:
        printf("You have entered an invalid choice");
        break;
    }

    return 0;}
```
Copy

Strings In C: C Tutorial In Hindi #34

*The selection of right IDE for programming is one of the most important thing. To run a **C program**, we need an IDE's like Visual Studio Code or Codeblocks. For this series, we are using Virtual Studio Code (VS Code). It provides the tools that a developer needs for a quick **code**-build-debug cycle. To download VS Code, click on **Download Virtual Studio Code** . For guidance, check the tutorial **Install & Configure VS Code***

*Now open VS Code and create a file with name **"tutorial34.c"***

Conclusion:-

Today we have learned about how to use strings in C programming. Each member of the array contains one of the characters in the string. By using scanf() we are not capable of receiving multi-word strings. The way to get a multi-word string is by using the function gets().

Code as described/written in the video

```
#include <stdio.h>
void printStr(char str[]){
    int i=0;
    while(str[i]!='\0')
    {
        printf("%c", str[i]);
        i++;
    }
    printf("\n");}int main(){
    // char str[] = {'h', 'a', 'r', 'r', 'y', '\0'};
    // char str[6] = "harry";
    char str[34];
    gets(str);
    printf("Using custom function printStr\n");
    printStr(str);
    printf("Using printf %s\n", str);
    printf("using puts: \n");
    puts(str);
```

```
   return 0;}
```

Copy

In today’s tutorial, we will learn **how to declare strings** and **how to work with strings in C programming**. Character arrays or strings are used by programming languages like C or Java, to manipulate text such as words and sentences.

String in C:-

String is an array of characters. Data of the same type are stored in an array for example, Integers can be stored in an integer array, similarly, a group of characters can be stored in a character array. **Character arrays are also called strings**. A string is a one-dimensional array of characters that is terminated by a null (‘\0’).

Declaration of strings:

Declaring a string is very simple, same as declaring a one-dimensional array. Below is the syntax for declaring a string.

```
char string_name[size];
```

Copy

In the above syntax, string_name is any name given to the string variable, and size is used to define the length of the string, i.e the number of characters that the strings will store. Keep in mind that there is an extra terminating character which is the null character (‘\0’) that is used to indicate the termination of string.

```
char name[ ] = { 'H', 'A', 'R', 'R', 'Y', '\0' } ;
```

Copy

Each character in the array, like “H”, occupies one byte of memory and the last character is always ‘\0’. The null character ‘\0’ looks like two characters, but it is actually only one character, with the \ indicating that what follows it is something special. Character array elements are stored in contiguous memory locations.

Equally we can make the string by assigning character values to each member.

```
name[0]='H';
name[1]='A';
name[2]='R';
name[3]='R';
```

```
name[4]='Y';
name[5]='\0';
```
Copy

The placeholder for string variables is %s.

Note: *There is difference between '\0' and '0'. The ASCII value of '\0' is 0, and for '0', the value is 48. The terminating null '\0' and '0' are not same.*

The null ('\0') is important in C programming because it is the only way the functions that work with a string can know where the string ends. When a string not terminated by a '\0', then it is not really a string but merely a collection of characters.

Example of string in C:-

```
#include<stdio.h>int main(){
   // declare and initialize string
   char str[] = "CodeWithHarry";
   printf("%s",str);
   return 0;} //Output:- CodeWithHarry
```
Copy

In order to read a string that contains the spaces, we use the gets() function. The purpose of gets is to ignore the whitespaces. When a newline is reached, gets stops reading. For example:

```
#include <stdio.h>int main() {char name[50];printf("Enter your name:
");gets(name);printf("My name is %s ",name);return 0;}
```
Copy

String Functions In C & string.h Library: C Tutorial In Hindi #35

*So, lets open our VS Code and create a file with name **"tutorial35.c"***

In this tutorial, we will learn to manipulate strings in C using different library functions. C provides us the useful string handling library functions. The string.h library is used to perform string operations. It provides several functions for manipulating character strings.We need to often manipulate or change the strings according to the need of a problem, the string.h library makes handling string in programming simple and easy to understand.

Following are some commonly used string handling functions:

strcat( ):-

This function is used to concatenates the source string at the end of the target string. For example, "Hello" and "World" on concatenation would result into a string "HelloWorld". Here is an example of strcat( ):

```
int main( ) { char  s[ ] = "Hello" ;char  t[30] = "World" ;strcat ( t, s ) ;printf
( "String = %s", t ) ;}//Output: string = HelloWorld
```

Copy

strlen( ):-

This function is used to counts the number of characters present in a string. Its example is given below:

```
int main( ) { char  str[ ] = "Harry" ;int  str_length;
str_length= strlen ( str ) ;printf ( " length = %d", str_length );}//Output: length =
5
```

Copy

strcpy( ):-

This function is used to copies the contents of one string into another. The base addresses of the source and target strings should be given to this function. Here is an example of strcpy( ):

```
int main( ) { char  s[ ] = "CodeWithHarry" ;char  t[20] ;strcpy ( t, s ) ;printf ( "\n
Source string = %s", s ) ;printf ( "\n Target string = %s", t ) ; }//And here is the
output...//Source string = CodeWithHarry//Target string = CodeWithHarry
```
Copy

strcmp( ):-

This function is used to compares two strings to find out whether they are same or different. The strcmp() will compare two strings character by character until there is a mismatch or end of one of the strings is reached. If both of the strings are identical, strcmp( ) returns a value zero. If they are not identical, it will return the numeric difference between the ASCII values of the first non-matching pairs of characters. Here is a example of strcmp( ).

```
#include <stdio.h>#include <string.h>int main(){char string1[ ] = "Harry" ;char
string2[ ] = "Code" ;int a;
a= strcmp ( string1, string2 ) ;printf ("\n%d", a) ;return 0;}//Output:5
```
Copy

strrev():-

This function is used to show the reverse of the string. Following are the example of strrev():

```
#include<stdio.h>#include<string.h>int main(){char str[50] =
"1234";printf("After reversing string is =%s",strrev(str));return 0;}//Output: After
reversing string is = 4321
```
Copy

*There are many other built-in string funtions like **strlwr , strupr ,strcat , strdup and strset,** these functions are also very useful in manipulating the strings. In today’s tutorial we have discussed only few of the string functions. You can explore more about string functions by searching it on the internet.*

Code as described/written in the video

```
#include <stdio.h>#include <string.h>
int main(){
    char s1[] = "harry";
    char s2[] = "ravi";
    char s3[54];
    printf("The strcmp for s1, s2 returned %d ", strcmp(s1, s2));
    // puts(strcat(s1, s2));
    // printf("The length of s1 is %d\n", strlen(s1));
    // printf("The length of s2 is %d\n", strlen(s2));
    // printf("The reversed string s1 is: ");
    // puts(strrev(s1));
    // printf("The reversed string s2 is: ");
    // puts(strrev(s2));

    // strcpy(s3 ,strcat(s1, s2));
    // puts(s3);


    // allow user to enter two strings and then concatenate them by saying that
    // str1 is a friend of str2
    return 0;}
```
Copy

← PreviousNext →

Structures In C: C Tutorial In Hindi #37

*To run a **C program**, we need an IDE's like Visual Studio Code or Code blocks. For this series, we are using Virtual Studio Code (VS Code). **Visual Studio Code** is a fast source code editor and provides the tools that a developer needs for a quick code-build-debug cycle. To download VS Code, click on **Download Virtual Studio Code** . For guidance, check the tutorial **Install & Configure VS Code***

*Now open VS Code and create a file with name **"tutorial37.c"***

*We have seen in the previous tutorials that how variables store one piece of information and how arrays store the information of the same data type. Variables and arrays can handle a great variety of situations. But quite often we have to deal with the collection of dissimilar data types. For dealing with dissimilar data types, C provides a data type called **'structure'**. A structure gathers together, different information that belongs to different data types.*

What is a Structure in C?

***Structures** are usually used when we want to store dissimilar data together.For example, we want to store data about a book. Book has its title, author name, number of pages and price. All of the book attributes belong to different data types. One way to store the data is to construct individual arrays, and another method is to use a structure variable. Structure elements are always stored in contiguous memory locations.*

The general form of a structure declaration statement is given below:

```
struct <structure name> {
structure element 1;
structure element 2;
structure element 3;} struct_variable;//Orstruct book  b1, b2, b3 ;
```

Copy

Before the final semicolon, we specify the structure variables but it is optional. We can also specify the structure variable in main body.

*The keyword **struct** should be used to define variables of structure type.*

Following is the example of declaring struct in C:

```
struct Books{  char title[20];  char author_name[100];  float price; int pages;}
book1;
```

Copy

How to Access the Structure Elements?

*As we use subscript to access individual elements of an array. But in the case of structures, to access any element, we use the **operator (.)**. This dot operator is coded between the structure variable name and the structure member that we wish to access.*

***Note** that before the dot operator there must always be a structure variable and after the dot operator there must always be a structure element.*

Example:-

```
#include <stdio.h>struct book {char title[20];  char Author_name[100];  float
price;  int pages;} ;int main( ) {struct book book1 = { "Cprogramming", "ABC",
130.00, 550 } ;printf ( "\n Title = %s", book1.title ) ;printf ( "\n Name = %s",
book1.Author_name ) ;printf ( "\n Price = %.2f",book1.price ) ; printf ( "\n
Pages = %d", book1.pages ) ;return 0;}
```

Copy

Additional Features of Structures:-

- *We can assign the values of a structure variable to another structure variable of the same type using the assignment operator.*
- *Structure can be nested within another structure.*
- *We can pass the structure variable to a function. We can pass the individual structure elements or the entire structure variable into the function as an argument.*
- *We can have a pointer pointing to a struct just like the way we can have a pointer pointing to an int, or a pointer pointing to a char.*

Where are structures useful?

***Structures** can be used for a variety of purposes like:*

1. *Structures are used to store a large amount of data.*
2. *They are used to send data to the printer.*
3. *For placing the cursor at an appropriate position on screen, we can use structure.*
4. *It can be used in drawing and floppy formatting.*
5. *We use structures in finding out the list of equipment attached to the computer*

Summary:-

In this tutorial, we have learned about structures in C. Structures are usually used when we want to store dissimilar data together. Its members can be accessed through a structure variable using a dot (.) operator. We can declare many structure variables for same structure and memory will be allocated for each separately.

Code as described/written in the video

```
#include <stdio.h>#include <string.h>struct Student{
    int id;
    int marks;
    char fav_char;
    char name[34];} harry, ravi, shubham;// struct Student harry, ravi, shubham;
void print(){
    printf("%s", harry.name);}
int main(){
    harry.id = 1;
    ravi.id = 2;
    shubham.id = 3;
    harry.marks = 66;
    ravi.marks = 466;
    shubham.marks = 46;
    harry.fav_char = 'p';
    ravi.fav_char = 'y';
    shubham.fav_char = 'o';
    strcpy(harry.name, "Harry Potter student of the year");
    strcpy(shubham.name, "Shubham Kumar");
    // printf("Harry got %d marks\n", harry.marks);
    // printf("Harry's name is: %s\n", harry.name);
    // printf("Shubham got %d marks\n", shubham.marks);
    // printf("Shubham's name is: %s\n", shubham.name);
    print();
    // Quick Quiz
    // print all the information of a given student

    return 0;}
```
Copy

← PreviousNext →

Typedef In C: C Tutorial In Hindi #38

*To run a **C program**, we need an IDE's like Visual Studio Code or Code blocks. For this series, we are using Virtual Studio Code (VS Code). **Visual Studio Code** is a fast source code editor and provides the tools that a developer needs for efficient programming. To download VS Code, click on **Download Virtual Studio Code** . For guidance, check the tutorial **Install & Configure VS Code***

*Now open VS Code and create a file with name **"tutorial38.c"***

In C programming, we use the typedef declarations to create shorter and meaningful names for types already defined by C like int, float or char. In this tutorial, we will learn about the role of typedef in C language more in detail.

What is typedef in C?

*A **typedef** is a keyword that is used to assign alternative names to existing datatypes. We use typedef with user defined datatypes, when names of the datatypes become slightly complicated to use in programs. Typedefs can be used to:*

- *Provide the clarity in the code*
- *it makes easier to change the underlying data types that you use*
- *Typedefs makes the code more clear and easier to modify.*

Following is the syntax for using typedef,

```
typedef <previous_name> <alias_name>
```

Copy

*In the above syntax, **'previous_name'** is the name of an already existing variable while **'alias_name'** is another name given to the existing variable.*

*For example, suppose we want to create a variable of type **unsigned long**, then it becomes a time taking task if we want to declare multiple variables of this type. To overcome this problem, we use **a typedef** keyword.*

```
typedef unsigned long ul;
```

Copy
The above example defines a term ul for an unsigned long datatype. Now this ul identifier can be used to define unsigned long type variables.

```
ul a, b, c;
```
Copy

Let's see an example.

```
#include <stdio.h>int main() {
        typedef unsigned long ul;
        ul a = 5, b = 8;
        printf("a = %d\n", a);
        printf("b = %d\n", b);
        return 0;}
```
Copy
There are ***various applications of typedef****. The following are the applications of the typedef.*

- *Typedef can be used with an array mostly with multi-dimensional array. It will increases the readability.*
- *As we know, the typedef can be implemented for defining a user-defined data type with a specific name and type. We can also use a typedef with structures of C language.*

```
typedef struct{
   structure element1;
   structure element2;
   structure element3;} name_of_type;
```
Copy
Here name_of_type can be implemented by declaring a variable of this structure type.

```
name_of_type type1, type2;
```
Copy

- *typedef can be used for providing a pseudo name to pointer variables as well.*

```
typedef int* ptr
ptr a, b, c;
```

Copy

Advantages of typedef

- *Typedef increases the readability of the code. If we are using structure and function pointer in our code, it will increase the readability of code.*
- *With the help of typedef, we can use the same name for the different types in different scopes.*
- *In the case of structure, if we use the typedef then we do not require to write struct keyword at the time of variable declaration.*
- *Typedef increases the portability of the code.*

Code as described/written in the video

```
#include <stdio.h>
typedef struct Student{
    int id;
    int marks;
    char fav_char;
    char name[34];} std;
int main(){
    // int *a, b;
    typedef int* intPointer;
    intPointer a, b;
    int c = 89;
    a = &c;
    b = &c;

    // std s1, s2;
    // s1.id = 56;
    // s2.id = 89;
    // printf("Value of s1's Id is %d\n", s1.id);
    // printf("Value of s2's Id is %d\n", s2.id);

    // typedef <previous_name> <alias_name>;
    // typedef unsigned long ul;
    // typedef int integer;
    // ul l1, l2, l3;
    // integer a = 4;
    // printf("Value of a is %d", a);
    return 0;}
```

Copy

Unions In C: C Tutorial In Hindi #39

*To run a **C program**, we need an IDE's like Visual Studio Code or Code blocks. For this series, we are using Virtual Studio Code (VS Code). To download VS Code, click on **Download Virtual Studio Code** . For guidance, check the tutorial **Install & Configure VS Code***

*Now open VS Code and create a file with name **"tutorial39.c"***

Today we will explore about unions and its role in C programming. In this tutorial we will learn:

- *how to create unions*
- *access its members*
- *learn the similarities between unions and structures*
- *learn the differences between unions and structures.*

Just like Structures, the union is a user-defined data type. All the members in union share the same memory location. The union is a data type that allows different data belong to different data types to be stored in the same memory locations. One of the advantages of using a union is that it provides an efficient way of reusing the memory location, as only one of its members can be accessed at a time. A union is used in the same way we declare and use a structure.

Defining union

We use the union keyword to define the union. The syntax to define union in C is given below.

```
  union union_name
  {
   datatype member1;
   datatype member2; };
```
Copy

Following is the example of Union in C

```
union books{
   int pages;
   float price;
```

```
    char title[20];}b1;
```
Copy

This declares a variable b1 of type union books. This union contains three members each with a different data type, price belongs to float data type, pages belong to integer data type and title belongs to character datatype. However, we can use only one of them at a time. This is because, only one location is allocated for all the union variables, irrespective of their size.

How to access the members of a union

We use ". "operator to access the members of a union.

Example: Accessing Union Members

```
#include <stdio.h>#include <string.h>
union Book {
   int pages;
   float price;
   char title[20];};
int main( ) {
   union Book b1;
   b1.pages = 100;
   printf( "Pages: %d\n", b1.pages);
   b1.price = 250.5;
   printf( "Price : %.1f\n", b1.price);
   strcpy( b1.title, "C Programming");
   printf( "Title : %s\n", b1.title);

   return 0;}
```
Copy

What are the similarities between Structure and Union

1. *Structure and union are user-defined data types used to store data of different types.*
2. *The members of structure and union can be objects of any type, including other structures and unions or arrays.*
3. *A union or a structure can be passed by value to functions and returned by value by functions.*

4. *‘.’ operator is used for accessing union and structure members.*

What are the differences between Structure and Union

1. *The keyword union is used to define a union and a keyword struct is used to define the structure*
2. *Each member within a structure is assigned a unique storage area of location whereas memory allocated is shared by individual members of the union.*
3. *Individual members can be accessed at a time in structure whereas only one member can be accessed at a time in union*
4. *Altering the value of the member will not affect other members of the structure, whereas altering the value of any member will affect other member's values.*
5. *Several members of a structure can be initialized at once, whereas only one member can be initialized in union*

Code as described/written in the video

```
#include <stdio.h>#include <string.h>union Student{
    int id;
    int marks;
    char fav_char;
    char name[34];};int main(){
    union Student s1;
    strcpy(s1.name, "Harry");
    s1.fav_char = 'u';
    s1.marks = 45;
    s1.id = 1;

    printf("The id is %d \n", s1.id);
    printf("The marks is %d \n", s1.marks);
    printf("The fav_char is %c \n", s1.fav_char);
    printf("The name is %s \n", s1.name);



    return 0;}
```

Copy

← *PreviousNext* →

C Language Array Reversal Exercise 5: Solution: C Tutorial In Hindi #40

Today we are going to discuss the solution of problem given in Tutorial #36. I hope that you have solved the problem and have learned the usage of arrays completely. The problem and instructions are again displayed below:-

Instruction:

In this task you have to write a C program that will reverse an array of integers. For that purpose, create the function that will take array as an argument and reverse an array.

Your program should print the array before and after reversal.

For Example:

Before Reversal: 1, 2, 3, 4, 5, 6, 67

After Reversal: 67, 6, 5, 4, 3, 2, 1

Code as described/written in the video

```
#include <stdio.h>// target: 67,6,5,4,3,2,1
// 7// 1,2,3,4,5,6,67// 67,2,3,4,5,6,1// 67,6,3,4,5,2,1// 67,6,5,4,3,2,1

void arrayRev(int arr[]){
    int temp;
    for (int i = 0; i < 7/2; i++)
    {
        //swap item arr[i] with arr[6-i]
        temp = arr[i];
        arr[i] = arr[6-i];
        arr[6-i] = temp;
    }}
void arrayPrint(int arr[]){
    for (int i = 0; i < 7; i++)
    {
        printf("The value of element %d is %d\n", i, arr[i]);
    }}int main(){
    int arr[] = {1,2,3,4,5,6,67};
    printf("Before reversing the array\n");
    arrayPrint(arr);
    arrayRev(arr);
    printf("\nAfter reversing the array\n");
    arrayPrint(arr);


    return 0;}
```
Copy

← PreviousNext →