FUNCTION RETURNING a POINTER

C [Level1] :Function return pointer

ISM
We inspire you to learn

- Function can return a pointer as it retun int,float and other data types.
- So function return value should be catch by the pointer variable.

Sample program is as follows:

```
Int *fun()
{
        int i=20;
        return (&i);
}
Main()
{
        int *p;
        p=fun();
        printf("\n address is::%u\t value is::%d",&p,*p);
}
```

POINTER TO FUNCTION

```
#include<stdio.h>
void fun()
{
        printf("\n we are in function");
}
main()
{
        void (*fun_ptr)();
        fun_ptr=fun;
        printf("\n we are in main");
        (*fun_ptr)();
        printf("\n");
}
```


Premier Embedded Training Centre in the country

COMMAND-LINE ARGUMENTS

- In C it is possible to accept cmd line arguments.
- Cmd args are given after the name of the program.
- For giving cmd line arguments to the c program at first we need to under stand the full declaration of the main program.
- Main accepts two arguments. They are as follows

  argc for number of cmd line arguments.(argument count)

  argv[] for full list of all cmd line arguments. It is pointer array.(argument vector)

Syntax:

Int main(int argc, char *argv[])

•argv[0] contains the program name or empty string.

•argv[argc] contains NULL POINTER.

Sample program:

```
int main(int argc,char *argv[])
{
        int i;
        char *c;
        for(i=1;i<argc;i++)
        {
                c=argv[i];
                printf("\n enterd strings are::%s",c);
        }
}
```

FUNCTION with VARIABLE NUMBER of ARGUMENTS

•If we have a requirement like we don't know how many arguments will be passed to the function.
•One way is accept a pointer to an array.
•Another way is create a function will accept any number of values and then return.
•For that at first we need include following library file:

#include<stdarg>

It has 4 parts

1)va_list

Which stores the list of arguments.
Syntax: va_list arguments;

2) va_start:

Which is used to initializes the list.

Syntax: va_start(arguments,num);

3)va_arg:

Which returns the next argument in the list.

Syntax: va_arg (arguments, double);

4) va_end:

Which cleans up the variable argument list.

Syntax: va_end (arguments);

NOTE:

Whenever a function declared to have an indeterminate number of arguments, in place of the last argument we should place an ellipsis (which looks like …)

```
int fun(int x, …)
{

}
```

```
#include <stdarg.h>
// this function will take the number of values to average followed by all of the numbers to average
double average ( int num, ... )
{
   va_list arguments;
   double sum = 0; int x;
/* Initializing arguments to store all values after num */
   va_start ( arguments, num );
   /* Sum all the inputs; we still rely on the function caller to tell us how many there are */
   for ( x = 0; x < num; x++ )
   {
      sum += va_arg ( arguments, double );
   }
   va_end ( arguments );                // Cleans up the list
   return (sum / num);
}
int main()
{
 //this computes the average of 13.2, 22.3 and 4.5 (3 indicates the number of values to average)
   printf( "%f\n", average ( 3,13.2, 22.3, 4.5 ) );
   /* here it computes the average of the 5 values 3.3, 2.2, 1.1, 5.5 and 3.3*/
   printf( "%f\n", average ( 5,3.3, 2.2, 1.1, 5.5, 3.3 ) );
}
```