Funciones

https://www.emertxe.com/embedded-systems/c-programming/c-sample-programs/

Capítulo 2: Funciones Parte 1

P1: Fundamentos de funciones - Ejemplo 1

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta es funciones. Técnicamente, una función se llama un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más.

Este código muestra cómo escribir una función usando C.

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t074_ch2_function_basics.c
 *   Title          : Functions Basics - Example 1
 *   Description    : Generalmente nos encontramos con una situación en la que terminamos
 *                    escribiendo el mismo código una y otra vez, lo que nos lleva a
 *                    preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta
 *                    es funciones. Técnicamente, una función se llama un grupo de
 *                    instrucciones escritas para realizar una funcionalidad específica
 *                    (por eso se la llama función). Al usar funciones, podemos lograr
 *                    muchas ventajas, como reutilización, dividir y vencer, modularidad,
 *                    capacidad de prueba, abstracción y muchas más.
 *                    Este código muestra cómo escribir una función usando C
 *-----------------------------------------------------------------------------------------*/

#include

/* Definición de función para aceptar 2 números del usuario y mostrar la suma de ellos */
void add()
{
    int num1, num2;
    int sum;

    printf("Introduzca dos números: ");
    scanf("%d%d", &num1, &num2);

    sum = num1 + num2;

    printf("La suma es %dn", sum);
}

int main()
{
 /* Llamada de función para la adición */
    add();

    return 0;
}
```

P2: Fundamentos de funciones - Argumentos

Resumen:

Este código muestra cómo escribir una función usando C que acepte argumentos.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t075_ch2_function_args.c
 *   Title          : Functions Basics - Arguments
 *   Description    : Este código muestra cómo escribir una función usando C que acepte
 *                    argumentos.
 *-------------------------------------------------------------------------------------------*/

#include

/*
 * Definición de función para aceptar 2 números del llamador y mostrar la suma de ellos
 * Todo lo que escribimos entre paréntesis se llama argumento(s) formal(es).
 */
void add(int num1, int num2)
{
    int sum;

    sum = num1 + num2;

    printf("The sum is %dn", sum);
}

int main()
{
    int num1, num2;

    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);

 /*
  * Llamada de función para aceptar 2 números del usuario y mostrar la suma de ellos
  * Todo lo que escribimos entre paréntesis se llama argumento(s) real(es).
  */
    add(num1, num2);

    return 0;
}
```

P3: Fundamentos de funciones: argumentos y retorno

Resumen:

Este código muestra cómo escribir una función usando C que acepta argumentos y devuelve un valor al que la llama.

```
/*-------------------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t076_ch2_function_args_and_return.c
 *   Title          : Functions Basics - Arguments and Return
 *   Description    : Este código muestra cómo escribir una función usando C que acepta argumentos y
 *                    devuelve un valor al llamador.
 *-------------------------------------------------------------------------------------------------------*/

#include
```

```c
/*
 * Definición de función para aceptar 2 números del llamador y mostrar la suma de ellos
 * Todo lo que escribimos entre paréntesis se llama argumento(s) formal(es).
 * El tipo de datos de retorno de la función es entero. Asegúrese de que el tipo de datos de los objetos
 * devueltos coincida con el tipo de datos de retorno
 */
int add(int num1, int num2)
{
    int sum;

    sum = num1 + num2;

    /* Devuelve el valor calculado al llamador */
    return sum;
}

int main()
{
    int num1, num2;
    int sum;

    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);

    /*
     * Llamada de función para aceptar 2 números del usuario y mostrar la suma de ellos
     * Todo lo que escribimos entre paréntesis se llama argumento(s) real(es).
     */
    sum = add(num1, num2);

    printf("The sum is %dn", sum);

    return 0;
}
```

P4: Fundamentos de funciones: ignorar el valor de retorno

Resumen:

Este código muestra cómo escribir una función usando C que acepta argumentos y devuelve un valor al llamador. El llamador puede ignorar el valor de retorno si es necesario.

```c
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t077_ch2_function_ignore_return.c
 *   Title          : Conceptos básicos de funciones: ignorar el valor de retorno
 *   Description    : Este código muestra cómo escribir una función usando C que acepta
 *                    argumentos y devuelve un valor al llamador. El llamador puede ignorar
 *                    el valor de retorno si es necesario.
 *-----------------------------------------------------------------------------------------*/

#include

/*
 * Definición de función para aceptar 2 números del llamador y mostrar la suma de ellos
 * Todo lo que escribimos entre paréntesis se llama argumento(s) formal(es).
 * El tipo de datos de retorno de la función es entero. Asegúrese de que el tipo de
 * datos de los objetos devueltos coincida con el tipo de datos de retorno
 */
int add(int num1, int num2)
```

```
{
 int sum;

 sum = num1 + num2;

 /* Devuelve el valor calculado al llamador */
 return sum;
}

int main()
{
 int num1, num2;
 int sum = 0;

 printf("Enter two numbers: ");
 scanf("%d%d", &num1, &num2);

 /*
  * Se llama a la función Add y se ignora su valor de retorno.
  */
 add(num1, num2);

 printf("The sum is %dn", sum);

    return 0;
}
```

Chapter 4 : Functions - Part 2

P1: Functions Basics - Pass by References - Example 1

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En esos casos, el paso por referencia es muy útil. Este ejemplo muestra un uso simple del paso por referencia.

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t091_ch2_function_pass_by_ref.c
 *   Title          : Functions Basics - Pass by References - Example 1
 *   Description    : Using functions we can achieve many advantages like Reusability,
 *                    Divide and conquer, Modularity, Testability, Abstraction and many more.
 *                    There could be instance were we need to deal with a huge amount of data,
 *                    return multiple values from functions, have common pool of data across
 *                    different functions etc., then Pass by reference is very helpful This
 *                    example show a simple usage of pass by reference
 *-----------------------------------------------------------------------------------------*/

#include

void modify(int *ptr)
{
    /* Changing the value of x with help of pointer */
    *ptr = 10;
}

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

```
    printf("x = %dn", x);

    /* The reference (address) of the x is sent to the modify function */
    modify(&x);

    printf("x = %dn", x);

    return 0;
}
```

P2: Functions Basics - Pass by References - Example 2

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil. Este ejemplo muestra el ejemplo más común analizado: "Intercambiar" 2 variables.

```
/*-------------------------------------------------------------------------------------------
 *   Author          : Emertxe (https://www.emertxe.com)
 *   Date            : Thursday 23 Mar 2016 16:00:04 IST
 *   File            : c_t092_ch2_function_pass_by_ref.c
 *   Title           : Functions Basics - Pass by References - Example 2
 *   Description     : This example show the most common discussed example "Swap" 2 variables.
 *-------------------------------------------------------------------------------------------*/

#include

void swap(int *ptr1, int *ptr2)
{
    int temp;

    temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

int main()
{
    int a = 5, b = 10;

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

    /* Send the references of a and b to swap function */
    swap(&a, &b);

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

    return 0;
}
```

P3: Functions Basics - Pass by References - Example 3

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil.

Este programa de código ilustra el método para devolver más de un valor mediante el paso por referencia. El lenguaje C no admite la devolución de múltiples valores mediante la palabra clave return.

La solución es pasar la dirección de las variables de resultado por referencia desde la función que realiza la llamada. En este ejemplo, sum y prod se pasan por referencia desde la función main().

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t093_ch2_function_pass_by_ref.c
 *   Title          : Functions Basics - Pass by References - Example 3
 *   Description    : Using functions we can achieve many advantages like Reusability,
 *                    Divide and conquer, Modularity, Testability, Abstraction and many more.
 *                    There could be instance were we need to deal with a huge amount of data,
 *                    return multiple values from functions, have common pool of data across
 *                    different functions etc., then Pass by reference is very helpful. This code
 *                    program illustrates the method of returning more than one value using pass
 *                    by reference. C language does not support returning multiple values using the
 *                    return keyword.
 *                    The solution is to pass the address of result variables by reference from the
 *                    calling function. In this example, sum and prod are passed by reference from
 *                    main() function.
 *-----------------------------------------------------------------------------------------*/

#include

void sum_prod(int x, int y, int *sum_ptr, int *prod_ptr)
{
    /*
     * The operation happens on x and y and the result is stored in the address
     * referenced (pointed) by pointer sum_ptr (pointing to address of sum)
     */
    *sum_ptr = x + y;
    /*
     * The operation happens on x and y and the result is stored in the address
     * referenced (pointed) by pointer prod_ptr (pointing to address of prod)
     */
    *prod_ptr = x * y;
}

int main()
{
    int a = 2, b = 5, sum, prod;

    /* The values of a and b are sent along with the references of sum and product */
    sum_prod(a, b, &sum, &prod);

    printf("sum = %d, prod = %dn", sum, prod);

    return 0;
}
```

P4: Functions Basics - Pass by References - Passing arrays - Example 1

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil.

Este programa ilustra el método de pasar un bloque de datos con la ayuda de matrices a una función.

```
/*-----------------------------------------------------------------------------------------
```

```
 *   Author          : Emertxe (https://www.emertxe.com)
 *   Date            : Thursday 23 Mar 2016 16:00:04 IST
 *   File            : c_t094_ch2_fuction_pass_array.c
 *   Title           : Functions Basics - Pass by References - Passing arrays - Example 1
 *   Description     : This program illustrates the method of passing a block of data with the
 *                     help of arrays a function.
 *-----------------------------------------------------------------------------------------*/

#include

/*
 * Accept the base address of the array in a variable ptr.
 * This is the second interpretation of an array. The compiler see this as a pointer to
 * the first element of an big variable 'a'.
 * Hence the writing any number as size inside the square bracket makes no sense. But
 * the square brackets are required for the compiler not to interpret ptr as normal
 * integer variable
 */
void print_array(int ptr[])
{
    int i;

    /*
     * The next issue is we will not be able to know the size of the array being passed to
     * the function because of the same reason stated above. Hence we end up in writing a
     * hard coded value as loop terminator.
     * This is not preferred for modular programming approach, so its better send the size
     * along with the base address of an array. Shown in next example
     */
    for (i = 0; i < 5; i++)
    {
     printf("%d ", ptr[i]);
    }
    puts("");
}

int main()
{
    int a[5] = {1, 2, 3, 4, 5};

    /* Pass the base address of the array */
    print_array(a);

    return 0;
}
```

P5: Functions Basics - Pass by References - Passing arrays - Example 2

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil.

Este programa ilustra el método de pasar un bloque de datos con la ayuda de matrices a una función.

```
/*-----------------------------------------------------------------------------------------
 *   Author          : Emertxe (https://www.emertxe.com)
 *   Date            : Thursday 23 Mar 2016 16:00:04 IST
 *   File            : c_t095_ch2_fuction_pass_array.c
 *   Title           : Functions Basics - Pass by References - Passing arrays - Example 2
 *   Description     : This program illustrates the method of passing a block of data with the
 *                     help of arrays a function.
```

```
 *-----------------------------------------------------------------------------------------*/

#include

/*
 * As mentioned in the previous example, The base address of the array is stored in
 * a pointer. Hence we can use the pointer notation to accept an array in a function
 */
void print_array(int *ptr, int n)
{
    int i;

    for (i = 0; i < n; i++)
    {
        /* Accessing a pointer like an array */
        printf("%d ", ptr[i]);
    }
    puts("");
}

int main()
{
    int a[5] = {1, 2, 3, 4, 5};
    int b[3] = {1, 2, 3};

    /* Pass the base address of arrays with their sizes */
    print_array(a, 5);
    print_array(b, 3);

    return 0;
}
```

P6: Functions Basics - Pass by References - Passing arrays - Example 3

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil.

Este programa ilustra el método de pasar un bloque de datos con la ayuda de matrices a una función.

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t096_ch2_fuction_pass_array.c
 *   Title          : Functions Basics - Pass by References - Passing arrays - Example 3
 *   Description    : This program illustrates the method of passing a block of data with the
 *                    help of arrays a function.
 *-----------------------------------------------------------------------------------------*/

#include

/*
 * You should be able to understand this example based on explanation given on the
 * previous examples
 */

void print_array(int *arr, int n)
{
    int i;

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

```
    {
        printf("%d ", arr[i]);
    }
    puts("");
}

void square_array(int a[], int n)
{
    int i;

    for (i = 0; i < n; i++)
    {
        *(a + i) = (*(a + i)) + (*(a + i));
    }
}

int main()
{
    int a[5] = {1, 2, 3, 4, 5};

    square_array(a, 5);
    print_array(a, 5);

    return 0;
}
```

P7: Functions Basics - Pass by References - Returning Arrays - Example 1

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil.

Este programa de código ilustra el método para devolver más de un valor mediante el paso por referencia. Por lo tanto, en ese contexto, podemos pasar la matriz desde el elemento principal y obtenerla completada desde la función.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t097_ch2_fuction_return_array.c
 *   Title          : Functions Basics - Pass by References - Returning Arrays - Example 1
 *   Description    : This code program illustrates the method of returning more than one value
 *                    using pass by reference. So on that context we can pass the array from
 *                    main get it filled from the fuction.
 *-------------------------------------------------------------------------------------------*/

#include

void print_array(int *arr, int n)
{
    int i;

    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    puts("");
}

void get_user_input(int *a, int n)
```

```
{
    int i;

    printf("Enter %d integers: ", n);

    for (i = 0; i < n; i++)
    {
        scanf("%d", a + i);
    }
}

int main()
{
    int a[5];

    get_user_input(a, sizeof(a) / sizeof(a[0]));
    print_array(a, 5);

    return 0;
}
```

P8: Functions Basics - Pass by References - Returning Arrays - Example 2

Resumen:

Puede haber casos en los que necesitemos trabajar con una gran cantidad de datos, devolver múltiples valores de funciones, tener un conjunto común de datos entre diferentes funciones, etc. En ese caso, el paso por referencia es muy útil.

Este programa de código ilustra el método para devolver más de un valor mediante el paso por referencia. Por lo tanto, podemos definir una matriz en una función y devolver su dirección al llamador.

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t098_ch2_fuction_return_array.c
 *   Title          : Functions Basics - Pass by References - Returning Arrays - Example 2
 *   Description    : This code program illustrates the method of returning more than one value
 *                    using pass by reference. So can define an array in a function and return
 *                    its address to the caller.
 *-----------------------------------------------------------------------------------------*/

#include

void print_array(int *arr, int n)
{
    int i;

    for (i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    puts("");
}

int *get_user_input(int size)
{
    /*
     * Note the definition below. The memory allocation for the array 'a' would be
     * automatic, which would get get destroyed when the function return becoming illegal
     * to access. By adding static storage modifier will help us get a memory which will
     * be available through out the program. Will see more on this in later examples
```

```
     */
    static int a[size];
    int i;

    for (i = 0; i < size; i++)
    {
        scanf("%d", a + i);
    }
}

int main()
{
    int *ptr;

    /* A function to get user inputs for 5 integer space */
    ptr = get_user_input(5);
    print_array(ptr, 5);

    return 0;
}
```

P9: Functions Basics - Function Prototypes - Example 1

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más. Este programa ilustra el uso del prototipo de función y el comportamiento predeterminado.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t099_ch2_funtion_prototype.c
 *   Title          : Functions Basics - Function Prototypes - Example 1
 *   Description    : This program illustrates the use of the function prototype and the default
 *                    behaviour
 *-------------------------------------------------------------------------------------------*/

#include

/*
 * The compiler expect how a function is defined, like its no of arguments, the type
 * of the arguments and a function return type.
 * Now fortunately the compiler see the definition of 'display' function before it is
 * called. So when any function calls it, the compiler has clear idea about its definition.
 * What if this function is defined in some other file or any function calls it before
 * the compiler knows about its definition?. Please look into the next example.
 */
void display(void)
{
    int count = 0;

    printf("count = %dn", ++count);
}

int main()
{
```

```
    display();

    return 0;
}
```

P10: Functions Basics - Function Prototypes - Example 2

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más. Este programa ilustra el uso del prototipo de función y el comportamiento predeterminado.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t100_ch2_funtion_prototype.c
 *   Title          : Functions Basics - Function Prototypes - Example 2
 *   Description    : This program illustrates the use of the function prototype and the default
 *                    behaviour
 *-------------------------------------------------------------------------------------------*/

#include

int main()
{
    /*
     * The 'display' function is called before the compiler can know about its
     * definition
     */
    display();

    return 0;
}

/*
 * The code would work fine though you get compiler warning.
 * But note that, this will never be the same in all the cases.
 * The compiler would lead to an error it finds a major mismatch than its assumption
 * Please see the next example.
 */
void display(void)
{
    int count = 0;

    printf("count = %dn", ++count);
}
```

P11: Functions Basics - Function Prototypes - Example 3

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para

realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más. Este programa ilustra el uso del prototipo de función y el comportamiento predeterminado.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t101_ch2_funtion_prototype.c
 *   Title          : Functions Basics - Function Prototypes - Example 3
 *   Description    : This program illustrates the use of the function prototype and the default
 *                    behaviour
 *-------------------------------------------------------------------------------------------*/

#include

int main()
{
    int num1 = 5, num2 = 10;
    double res;

    res = average(num1, num2);

    printf("average = %fn", res);
}

/*
 * This will lead to a compiler error. How to solve this??
 * Function Prototype - Please look into next example
 */
double average(int x, int y)
{
    double avg;

    avg = (x + y) / 2.0;

    return avg;
}
```

P12: Functions Basics - Function Prototypes - Example 4

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más. Este programa ilustra el uso del prototipo de función y el comportamiento predeterminado.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Thursday 23 Mar 2016 16:00:04 IST
 *   File           : c_t102_ch2_funtion_prototype.c
 *   Title          : Functions Basics - Function Prototypes - Example 4
 *   Description    : This program illustrates the use of the function prototype and the default
 *                    behaviour
 *-------------------------------------------------------------------------------------------*/
```

```c
#include

/*
 * Function prototype
 * With the below line the compilers is aware about the exception of the 'average' function
 * when called before definition.
 * These are also called as declarations
 */
double average(int x, int y);
/*
 * Note: The variable names are not important in prototypes, but its types are.
 * The above prototype could be written as
 */
double average(int, int);
/*
 * Ya ya, I know :), the declarations can be made any number of times
 */

int main()
{
    int num1 = 5, num2 = 10;
    double res;

    res = average(num1, num2);

    printf("average = %fn", res);
}

double average(int x, int y)
{
    double avg;

    avg = (x + y) / 2.0;

    return avg;
}
```

Chapter 6 : Functions - Part 3

P1: Functions Basics - Function Linking - Implicit

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más.

Este programa ilustra la vinculación implícita de las funciones estándar por parte del compilador.

```c
/*----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Wednesday 05 April 2017 16:00:04 IST
 *   File           : c_t111_ch2_functions_linking.c
 *   Title          : Functions Basics - Function Linking - Implicit
 *   Description    : We generally encounter a situation where we end up writing the same code
 *                    again and again leading up to a question, can we avoid this some how?,
 *                    Yes, the answer is functions. Technically a function be called as group of
 *                    instructions written to do a specific functionality (hence called as
```

```
 *                   function).
 *                   Using functions we can achieve many advantages like Re usability, Divide and
 *                   conquer, Modularity, Testability, Abstraction and many more.
 *                   This program illustrates the implicit linking of the standardfunctions by
 *                   the compiler.
 *-----------------------------------------------------------------------------------------------*/

int main()
{
    /*
     * 'printf' is a standard built-in library function.
     * If you note, we have not included the function prototype for this, so we might expect
     * warning by the compiler!. In order to avoid it we have to add a header file which has
     * the declaration of 'printf', and this is nothing but "stdio.h"
     * Interestingly you would observe that the compiler would compile the code and it would
     * run without even including the header file!.
     * Why? Well the answer is simple, when the compiler see a function call before it sees
     * the definition, it implicitly assumes then to be in libc library and if it finds it
     * proceeds with the compilation, else we will get an error while linking
     */
    printf("hellon");

    /* Please do see the next example for explicit linking */

    return 0;
}
```

P2: Functions Basics - Function Linking - Explicit

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más.

Este programa ilustra la necesidad de vincular explícitamente las funciones estándar por parte del compilador.

```
/*-----------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Wednesday 05 April 2017 16:00:04 IST
 *   File           : c_t111_ch2_functions_linking.c
 *   Title          : Functions Basics - Function Linking - Implicit
 *   Description    : This program illustrates the implicit linking of the standardfunctions by
 *                    the compiler.
 *-----------------------------------------------------------------------------------------------*/

#include
/* Header file containing the declaration for sqrt function */
#include

int main()
{
    double res;
    double val = 10;

    /*
     * Will compiler do the same thing as mentioned in the previous example
```

```
     * Well yes, but the sqrt function is not part of the libc library, and hence
     * we need to explicitly link the libm library while compiling
     * See the sample output section
     */
    res = sqrt(val);

    printf("%fn", res);

    return 0;
}
```

P3: Functions Basics - Recursion

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se llama un grupo de instrucciones escritas para realizar una funcionalidad específica (de ahí que se la llame función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más.

Una función que se llama a sí misma para realizar una tarea determinada se llama recursión. Un concepto que se puede usar para resolver un problema de manera elegante. Muchas operaciones complejas se pueden implementar fácilmente con ella. Pero, como todos sabemos, todo tiene sus ventajas y desventajas. Para tener recursión, necesitamos una buena cantidad de memoria y nos toma más tiempo realizar la tarea debido al cambio de contexto.

Este programa ilustra cómo implementar una función recursiva simple para generar un factorial de un número dado.

```
/*-----------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Wednesday 05 April 2017 16:00:04 IST
 *   File           : c_t113_ch2_function_recursion.c
 *   Title          : Functions Basics - Recursion
 *   Description    : A function calling itself to perform given task is called as Recursion.
 *                    A concept which could be used to solve a with it. But as we all know that,
 *                    every thing as an advantage and disadvantage. In order to have recursion we
 *                    need good amount of memory and take more time to preform the task because of
 *                    context switching.
 *                    This program illustrates the how to implement a simple recursive function for
 *                    a generating a factorial of a given number
 *-----------------------------------------------------------------------------------------------*/

#include

int factorial(int n)
{
    /*
     * Base condition:
     * Every recursive solution should have a finite limit. When a function is called
     * a stack frame is created and grows till it reaches the depth (a function call within
     * a function). The last frame which is created would get destroyed first in a LIFO manner.
     * If the depth of the function call doesn't end within amount of allocated stack, it would
     * lead to segment violation (sometimes called as stack overflow). So the decided limit
     * up to which a stack can grow is called base condition
     */
    if (n == 0)
    {
        return 1;
```

```
    }
    else
    {
        /*
         * Recursive condition:
         * A condition which call the function (self) again is called as recursive condition
         */
        return n * factorial(n - 1);
    }
}
```

P4: Functions Basics - Recursion - No Base Condition

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se llama un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se la llama función). Al usar funciones, podemos lograr muchas ventajas, como

reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más.

Una función que se llama a sí misma para realizar una tarea determinada se llama recursión. Un concepto que se puede usar para resolver un problema de manera elegante. Muchas operaciones complejas se pueden implementar fácilmente con ella. Pero, como todos sabemos, todo tiene ventajas y desventajas. Para tener recursión, necesitamos una buena cantidad de memoria y lleva más tiempo realizar la tarea debido al cambio de contexto.

Este programa ilustra cómo la recursión de la función podría provocar un desbordamiento de pila.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Wednesday 05 April 2017 16:00:04 IST
 *   File           : c_t113_ch2_function_recursion.c
 *   Title          : Functions Basics - Recursion
 *   Description    : This program illustrates the how to function recursion would lead to stack
 *                    overflow.
 *-------------------------------------------------------------------------------------------*/

#include

int test(int n)
{
    /*
     * The base condition missing!!
     */

    /*
     * Recursive condition:
     * A condition which call the function (self) again is called as recursive condition
     * Since the base condition is missing, this would keep on calling (say infinite,
     * theoretically) leading to stack overflow
     */
    return n + test(n - 1);
}

int main()
{
    int res;

    res = test(3);
```

```
    printf("res = %dn", res);

    return 0;
}
```

Chapter13 : Functions - Part 4

P1: Functions - Command Line Arguments - Example 1

Resumen:

Generalmente nos encontramos con una situación en la que terminamos escribiendo el mismo código una y otra vez, lo que nos lleva a preguntarnos: ¿podemos evitar esto de alguna manera? Sí, la respuesta son las funciones. Técnicamente, una función se denomina un grupo de instrucciones escritas para realizar una funcionalidad específica (por eso se denomina función). Al usar funciones, podemos lograr muchas ventajas, como reutilización, dividir y vencer, modularidad, capacidad de prueba, abstracción y muchas más.

Este programa ilustra cómo usar el argumento de la línea de comandos.

```
/*-----------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t187_ch5_functions_cmd_line.c
 *   Title          : Functions - Command Line Arguments - Example 1
 *   Description    : This program illustrates how to use the command line argument
 *-----------------------------------------------------------------------------------------------*/

#include

/*
 * argc (argument count)  - Contains number of arguments including program name
 * argv (argument vector) - Contains the actual arguments.
 * argv[0] is the program name
 */

int main(int argc, char *argv[])
{
    int i;

    printf("No of args = %dn", argc);

    puts("List of arguments");

    for (i = 0; i < argc; i++)
    {
        printf("%d: %sn", i, argv[i]);
    }

    return 0;
}
```

P2: Functions - Command Line Arguments - Example 2

Resumen:

Este programa ilustra cómo usar el argumento de la línea de comandos.

```c
/*------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t188_ch5_functions_cmd_line.c
 *   Title          : Functions - Command Line Arguments - Example 2
 *   Description    : This program illustrates how to use the command line argument
 *------------------------------------------------------------------------------------------*/

#include
#include

static void usage(char *program)
{
    printf("Usage:n");
    printf("%s   ..... n", program);

    exit(1);
}

int main(int argc, char *argv[])
{
    int i, sum = 0;

    if (argc == 1)
    {
        usage(argv[0]);
    }

    for (i = 1; i < argc; i++)
    {
        sum += atoi(argv[i]);
    }

    printf("average = %fn", (double) sum / (argc - 1));

    return 0;
}
```

P3: Functions Pointers - Example 1

Resumen:

Este programa ilustra cómo utilizar los punteros de función.

```c
/*------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t189_ch5_function_pointers.c
 *   Title          : Functions Pointers -  Example 1
 *   Description    : This program illustrates how to use the function pointers
 *------------------------------------------------------------------------------------------*/

#include

int add(int a, int b)
{
    return a + b;
}

int sub(int a, int b)
{
    return a - b;
}
```

```
int main()
{
    int res;
    int (*fptr)(int, int);

    /* Assign add functions address */
    fptr = &add;

    /* Invoke add function */
    res = (*fptr)(5, 10);
    printf("res = %dn", res);

    /* Assign sub function address */
    fptr = sub;
    /* Call sub func. Notice that u need not derefence fptr */
    res = fptr(5, 10);

    printf("res = %dn", res);

    return 0;
}
```

P4: Functions Pointers - Example 2

Resumen:

Se utiliza 'atexit' para registrar una función que se llamará al finalizar el proceso normal. Se llamará a my_exit antes de que finalice el programa.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t190_ch5_function_pointers.c
 *   Title          : Functions Pointers -  Example 2
 *   Description    : 'atexit' is used to register a function to be called at normal process
 *                    termination. my_exit will get called before the program terminates.
 *-------------------------------------------------------------------------------------------*/

#include
#include

void my_exit(void)
{
    printf("Exiting programn");
}

int main()
{
    /* Registering my_exit function */
    atexit(my_exit);

    exit(1);

    puts("Before Exit");

    return 0;
}
```

P5: Functions Pointers - Example 3

Resumen:

Este ejemplo muestra cómo pasar un puntero de función como argumento a una función.

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t191_ch5_function_pointers.c
 *   Title          : Functions Pointers -  Example 3
 *   Description    : This example shows how to pass function pointer as an argument to function.
 *-----------------------------------------------------------------------------------------*/

#include

int addition(int a, int b)
{
    return a + b;
}

int subtraction(int a, int b)
{
    return a - b;
}

int operation (int x, int y, int (*functocall)(int,int))
{
    int g;

    g = functocall(x,y);

    return g;
}

int main ()
{
    int m,n;
    int (*minus)(int, int) = subtraction;

    m = operation (7, 5, addition);
    n = operation (20, 5, minus);

    printf("%d %dn", m, n);

    return 0;
}
```

P6: Functions Pointers - Example 4

Resumen:

Este ejemplo muestra el uso del puntero de función mediante la función incorporada qsort.

```
/*-----------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t192_ch5_function_pointers.c
 *   Title          : Functions Pointers -  Example 4
 *   Description    : This example shows usage of function pointer using qsort builtin function.
 *-----------------------------------------------------------------------------------------*/

#include
```

```
#include

int sort_ascending(const void *a, const void *b)
{
    return *(int *)a > *(int *)b;
}

int sort_descending(const void *a, const void *b)
{
    return *(int *)a < *(int *)b;
}

int sort_string(const void *a, const void *b)
{
    return strcmp(*(char **)a, *(char **)b);
}

void print_int_array(int *arr, unsigned int size)
{
    int i = 0;

    for (i = 0; i < size; i++)
    {
        printf("%dn", arr[i]);
    }
}

void print_str_array(char **arr, unsigned int size)
{
    int i = 0;

    for (i = 0; i < size; i++)
    {
        printf("%sn", arr[i]);
    }
}

int main(void)
{
    int array1[5] = { 9, 2, 6, 1, 7 };
    char *str_array1[] = {"hello", "apple", "zebra", "india", "cricket" };
    size_t strings_len = sizeof(str_array1) / sizeof(char *);
    int size = 5;
    int i = 0;

    qsort(array1, size, sizeof(int), sort_ascending);
    printf("Ascendingn");
    print_int_array(array1, size);

    qsort(array1, size, sizeof(int), sort_descending);
    printf("Descendingn");
    print_int_array(array1, size);

    i = strings_len + 1;

    qsort(str_array1, strings_len, sizeof(char *), sort_string);
    printf("nString sortn");
    print_str_array(str_array1, strings_len);

    return 0;
}
```

P7: Functions - Variable Arguments

Resumen:

Este ejemplo muestra cómo utilizar la función de argumento variable.

```
/*-------------------------------------------------------------------------------------------
 *   Author         : Emertxe (https://www.emertxe.com)
 *   Date           : Fri 14 April 2017 16:00:04 IST
 *   File           : c_t193_ch5_function_var_arg.c
 *   Title          : Functions - Variable Arguments
 *   Description    : This example shows how to use Variable argument fuction.
 *-------------------------------------------------------------------------------------------*/

#include
#include

double calc_mean(int no_of_args, ...)
{
    va_list ap;
    double temp, sum = 0;
    int i;

    va_start(ap, no_of_args);

    for (i = 0; i < no_of_args; i++)
    {
        /* Fetch the next argument (of double type) */
        temp = va_arg(ap, double);
        sum += temp;
    }

    va_end(ap);

    return (sum / no_of_args);
}

int main()
{
    double res;

    res = calc_mean(3, 3.0, 4.0, 5.0);
    printf("avg = %lfn", res);

    res = calc_mean(2, 1.0, 2.0);
    printf("avg = %lfn", res);

    return 0;
}
```