```
int func_1 (int x, int y)
{
int a,b;
   a=x+5*y;
   b=func_2 (a);
return b;
}
```

≡ α

Dirección: α

```
int func_2 (int z)
{
    return z*3;
}
```

- Una función posee una dirección.
- Es decir, una función sería direccionable.
- Una función posee un proto**Tipo** (argumentos y tipo de dato a devolver).

una función puede ser caracterizada por su dirección y prototipo (o contenido).

Punteros

Declaración de un Puntero a Función

- Al declarar un puntero debemos definir a que tipo de elemento esta apuntando
- En caso de simples variables estos elementos podrán ser: int, float, etc.
- En el caso de funciones, el elemento está definido por el prototipo de la función, por ejemplo:

int _____ (int, int); int * _____ (void *, void *);

void _____ (char *); char * _____ (void);

- Como todo puntero, un puntero a función es una variable, y como tal requiere un nombre, es decir poder invocar a la variable a través de un nombre o identificador.

int ***pnt*** (void *, void *); Sin embargo, esta es la declaración de una función que retorna un puntero a int.

- Falta algo, para poder discernir entre ambos tipos de punteros.

int **(***pnt***)** (void *, void *) Declaración de Puntero a Función.

Ejemplos de declaración de Puntero a Función (PaF)

void (*pnt)();	**pnt** es un **PaF**, sin parámetros, que devuelve **void**.
void (*pnt)(int);	**pnt** es un **PaF** que recibe un **int** y devuelve **void**.
int (*pnt)(int, char);	**pnt** es un **PaF**, que recibe un **int** y un **char** y retorna **int**.
int* (*pnt)(int*, char*);	**pnt** es un **PaF**, que recibe punteros a **int** y **char**, y devuelve un puntero a **int**.
float (*pnt (char))(int);	**pnt** es un **PaF** que recibe un **char** y devuelve un PaF que recibe un **int** como argumento y devuelve un **float**.
int (* apnt[10])(int);	**apnt** es un array de 10 **PaF** que recibe un **int** y devuelve un **int**.
void func (int ,int (*pnt)(void *, void *));	**func** es una función que devuelve void y recibe como argumo un int y un **PaF, pnt,** que recibe dos punteros a void y devuelve un **int**.

Algunos ejemplos simples, basados en las 4 operaciones básicas

Archivos pnt_func x.c – parte común

```
int suma (int x,int y)
{
        return x+y;
}

int resta (int x,int y)
{
        return x-y;
}
```

```
int suma (int,int);

int resta (int,int);

int multi (int,int);

int divi (int,int);
```

```
int multi (int x,int y)
{
        return x*y;
}

int divi (int x,int y)
{
        if (!y)  return 0;

        return x/y;
}
```

Archivos pnt_func1.c

Llamado simple a las funciones a través de un PAF.

```
int main(void) {
int (*p) (int,int);     // declaración de la variable tipo PaF
int a=10, b=5, x;

        p=suma;      // asignación del PaF
        x=p (a,b);     // llamado a la función a través del PaF
        printf ("\n\nLa suma de %d y %d es %d",a,b,x);

        p=resta;
        x=p (a,b);
        printf ("\n\nLa resta de %d y %d es %d",a,b,x);

        p=multi;
        x=p (a,b);
        printf ("\n\nEl producto de %d y %d es %d",a,b,x);

        p=divi;
        x=p (a,b);
        printf ("\n\nLa division de %d y %d es %d",a,b,x);

return 0;}
```

Archivos pnt_func2.c

Llamado simple a las funciones a través de un array de PAF.

```
int main(void) {

        // declaración e inicialización del array de PaF
int (*p[ ]) (int,int) ={suma,resta,multi,divi,NULL};

        // declaración auxiliar de array de string
char *str[]={"La suma","La resta","El producto","La division", NULL};

int a=10, b=5, x, i;

        // recorrido y ejecución del array de PaF
        for (i=0 ; p[i] != NULL ; i++)
        {
            x=p[i] (a,b); // llamado a la función a través del PaF
            printf ("\n\n%s de %d y %d es %d",*(str+i),a,b,x);
        }

return 0;
}
```

Archivos pnt_func3.c

Ejemplo simple de un PAF como argumento.

```
// declaración e implementación de un PaF como argumento
void func ( int a , int b , int i , int (*pnt)(int,int))  {
char *str[]={"La suma","La resta","El producto","La division"};
        printf ("\n\n%s de %d y %d es %d",*(str+i),a,b,pnt(a,b));
}

int main(void) {

int (*p[ ]) (int,int) ={ suma , resta , multi , divi };
int a=10, b=5;
char op;
        do {        // Menú de opción
            printf ("\n\n\n Datos- a:%d b: %d",a,b);
            printf ("\n\n1-suma\n2-resta\n3-producto\n4-division");
            printf ("\n\nIngrese Operacion:");
            op=getch() -'1';        // ajuste de índice
          }while ( (op<'1')||(op>'4'));

        func (a , b , op , p[ op ]);

return 0;}
```

Algunas funciones estándar en C que usan PaF

void **qsort** (void * base, size_t num, size_t size,
int (*** comp**) (const void *, const void *));

void * **bsearch** (const void * key, const void * base, size_t num, size_t size,
int (*** comp**) (const void *, const void *));

key Puntero al objeto a buscar dentro del array.

base puntero al primer elemento del array a ser ordenado

num Número de elementos que conforman el array, apuntado por *base*.

Size Tamaño en bytes de cada elemento en el array.

comp Función que compara dos elementos dentro del array.

Su prototipo es: int comp (const void * elem1, const void * elem2);

La función recibe dos punteros void, a los elementos a ser comparados.
Esta función debe retornar un valor representativo del la comparación entre elem1 y elem2