
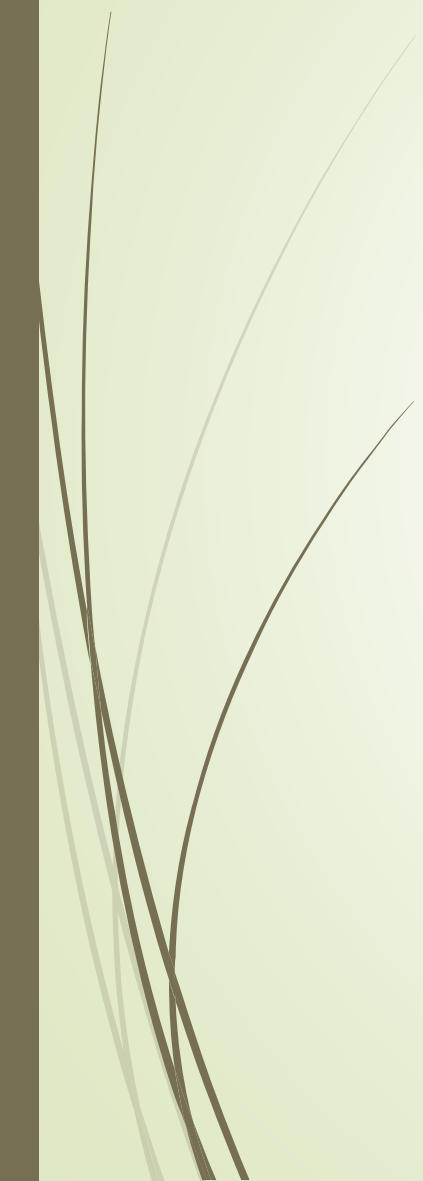


Cont. FUNCIONES

Catedrático: Ing. David Rajo

- 
- 
- Funciones externas
 - Ámbito
 - Recursividad
 - Archivos de cabecera



Funciones Externas



- Una función externa no es siempre totalmente global. En el lenguaje C++ se aplica la siguiente regla:
 - El alcance de una variable externa (o función) inicia en el punto de declaración hasta el fin del archivo (módulo) donde fue declarada.
 - Las funciones externas deben ser definidas antes de la función llamadora que las invoca o en un archivo externo conocidos como archivos de cabecera o encabezados



Considerar el siguiente código:

```
main() { ... }  
int que_alcance;  
float fin_de_alcance[10];  
void que_global() { ... }  
char solitaria;  
float fn() { ... }
```




Visibilidad de la funciones

- La función `main()` no puede ver a las variables `que_alcance` o `fin_de_alcance`, pero las funciones `que_global()` y `fn()` si pueden. Solamente la función `fn()` puede ver a solitaria.
- Esta es también una de las razones por las que se deben poner los prototipos de las funciones antes del cuerpo del código.
- Por lo que en el ejemplo la función `main` no conocerá nada acerca de las funciones



Ejemplo

- Modificar el ejemplo de la calculadora para realizar las operaciones de manera modular (a través de funciones externas)
- Agregar la opción de potencia.



Ámbito (alcance)

- El ámbito es la zona del programa en la que es visible una variable
- Existen 4 tipos de ámbitos:
 - Programa
 - Archivo fuente
 - Función
 - Bloque
- Normalmente, la posición de la sentencia en el programa determina el ámbito

Ámbito del programa (variables globales)

- Las variables que tienen ámbito de programa pueden ser referenciadas por cualquier función en el programa completo; tales variables se llaman variables globales.
- Para hacer una variable global, declárela simplemente al principio de un programa, fuera de cualquier función.
- Si se define una variable global, cualquier línea del resto del programa, no importa cuantas funciones y líneas de código le sigan, podrá utilizar esa variable.

```
int g, h; // variables globales  
Main(){  
}
```


Ámbito de archivo fuente (palabra reservada static)

- Una variable que se declara fuera de cualquier función y cuya declaración contiene la palabra reservada static tiene ámbito de archivo fuente.
- Las variables con este ámbito se pueden referenciar desde el punto del programa en que están declaradas hasta el final del archivo fuente. Si un archivo fuente tiene más de una función, todas las funciones que siguen a la declaración de la variable pueden referenciarla

```
static int i;  
void func(void){  
...  
}
```



static

- Cuando se define una variable como static esta palabra clave significa que la variable tiene una duración estática (es almacenada en memoria cuando el programa inicia y desalojada cuando el programa termina) y se inicialización 0 a menos que le sea especificado otro valor. Cuando se modifica una variable o función en el ámbito de archivo, la palabra static especifica que la variable o función tiene un enlace interno (su nombre no es visible desde fuera del archivo fuente en donde ha sido declarada).

static en una función

- Una variable declarada static en una función mantiene su valor entre llamadas a esa función, lo que estamos indicándole al compilador es que dicha variable sea inicializada solo una vez (la primera vez que se llama a la función), y el resto de veces que se llame a la función, la variable contendrá el último valor asignado. Esta variable sólo podrá ser visible desde la función que declara dicha variable.

```
void mifuncion(){  
    static int i=0;  
    cout<<"En la entrada i vale "<<i<<endl;  
    for(int j=0;j<10;j++)  
        i++;  
    cout<<"En la salida i vale "<<i<<endl;  
}
```



Ámbito de una función (Variables locales a la función)

- Una variable que tiene ámbito de una función se puede referenciar desde cualquier parte de la función. Las variables declaradas dentro del cuerpo de la función se dice que son locales a la función. Las variables locales no se pueden utilizar fuera del ámbito de la función en que están definidas.

```
void funcdemo()  
{  
    int i;  
    ...  
}
```

Ámbito de bloque (Variables locales)

- Una variable declarada en un bloque tiene ámbito de bloque y puede ser referenciada en cualquier parte del bloque, desde el punto en que está declarada hasta el final del bloque. Las variables locales declaradas dentro de una función tienen ámbito de bloque de la función; no son visibles fuera del bloque
- Una variable declarada en el interior de un bloque solo es visible en el interior del bloque

```
void func(int j)
{
    if (j>3)
    {
        int i;
        for (i=0;i<50;i++)
            func2(i);
    }
    //Aquí ya no es visible la variable i
}
```



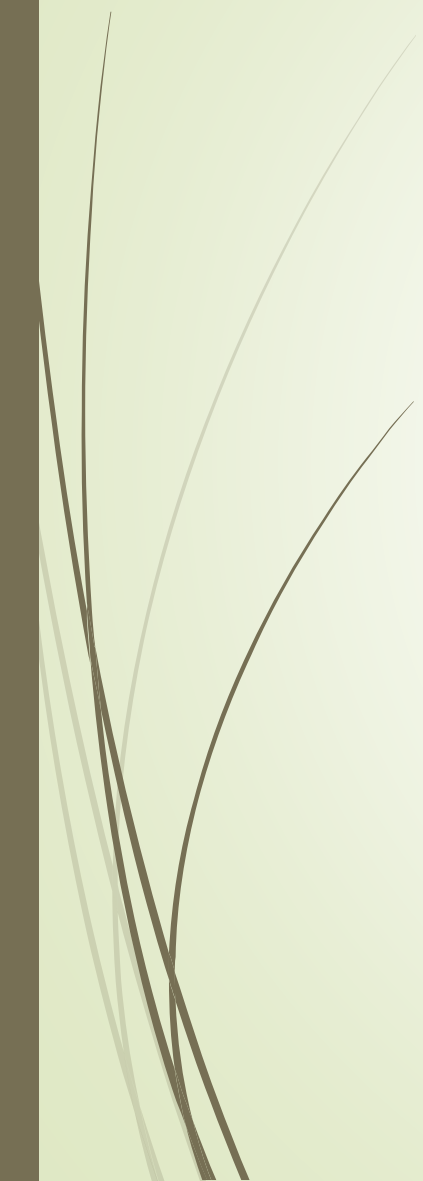
Recursividad



- Se dice que una función es recursiva cuando se define en función de si misma es decir se llama a sí misma directa o indirectamente.
- No todas la funciones pueden llamarse a si mismas, sino que deben estar diseñadas especialmente para que sean recursivas, de otro modo podrían conducir a bucles infinitos, o a que el programa termine inadecuadamente
- Por ello un proceso recursivo debe tener una condición de terminación o caso base.
- No todos los lenguajes de programación permiten usar recursividad.



Ejemplo

- Factorial de un numero recursivo
 - Contar hasta 10 de modo recursivo
- 



Tarea

- Investigar: Ventajas y desventajas de la recursividad.



Archivos de cabecera

- Son archivos externos que contienen bloques de código que se utilizan con mucha frecuencia en diferentes programas.
- Si se adopta el modelo modular entonces se querrá tener para cada módulo las definiciones de las variables, los prototipos de las funciones, etc. Sin embargo, ¿qué sucede si varios módulos necesitan compartir tales definiciones? En tal caso, lo mejor es centralizar las definiciones en un archivo, y compartir el archivo entre los módulos. Tal archivo es usualmente llamado un archivo cabecera y su extensión es *.h

Ejemplo: MiEncabezado.h

- Se crea un archivo con extensión .h y se añade el siguiente código, dentro de las directivas declaradas se coloca el cuerpo del archivo

```
#ifndef MIENCABEZADO_H_INCLUDED
```

```
#define MIENCABEZADO_H_INCLUDED
```

Aquí el cuerpo del archivo

```
#endif // MIENCABEZADO_H_INCLUDED
```

Directivas `#ifdef` , `#ifndef`

- Las directivas `#ifdef` y `#ifndef` son condicionales especializadas para comprobar si un macro-identificador está definido o no. En el caso del archivo `.h` evita inclusiones múltiples que generan la declaración de una función múltiples veces y esto deriva en un error de compilación

```
#ifndef MIENCABEZADO_H_INCLUDED
```

```
#define MIENCABEZADO_H_INCLUDED
```

```
... contenido del fichero
```

```
#endif
```

- Sin embargo, esta disposición tiene el inconveniente de que tenemos que acordarnos de definir `MIENCABEZADO_H_INCLUDED` cada vez que vayamos a incluir la cabecera
- La directiva `#define`, sirve para definir macros



Incluir archivo en programa

- Se coloca la directiva include con el nombre del archivo entre comillas y su extensión en la parte de declaraciones globales de la siguiente forma:

```
#include <iostream>
#include <cstdlib>
#include "MiEncabezado.h"
using namespace std;
int main (){
Return 0;
}
```



Ejemplo

- Calculadora utilizando archivo de cabecera definido