Universidad de Puerto Rico
Recinto Universitario de Mayagüez
Departamento de Ingeniería Eléctrica y de Computadoras

LABORATORIO III
ICOM 4015 ADVANCED PROGRAMMING

1. SCOPE RULES

El alcance de una variable, es la región de código en la cual una declaración esta activa. Por ejemplo, cuando se declara una variable como local en un bloque, esta solo puede ser referenciada en ese mismo bloque, o en bloques internos a este. Es una buena práctica limitar el alcance de las variables lo más posible para dar mantenimiento al código.

El propósito principal de las reglas de alcance es permitir el desarrollo de secciones independientes de código de tal manera que aún cuando Los identificadores idénticos son usados para referenciar a diferentes objetos no exista un conflicto, y también para permitir al programa para re-usar la memoria previamente asignada a diferentes variables.

La declaración de variables estáticas (static) tienen un efecto diferente dependiendo de si la variable es declarada globalmente o a nivel de función. Cuando las variables estáticas son declaradas en un alcance global, estas sólo pueden ser accesadas por funciones definidas dentro de este archivo fuente. Pero si las variables son declaradas como estáticas dentro de la definición de una función, a estas se les asignan unos espacios de almacenamiento y son inicializadas antes de que comience la ejecución del programa. Estas permanecen hasta que el programa finaliza.

A continuación se muestran una serie de ejemplos, donde se ilustra claramente los diferentes alcances de las variables:

Ejemplo 1

```
#include <iostream.h>
void Increment(void);
void Print();
 int global;

int main(void) {
   int i;
   for (i = 0; i != 10; i ++)
       Increment();
   Print();
   return 0;}

void Increment(void) {
    static int number = 100;
 / / number es iniciada a 100 sólo en el primer llamado a la función Increment().

  cout << number++ << endl;     / / en cada llamado el valor de number es de salida.
  cout << global << endl;
    / / en cada llamado el valor de global es el mismo, puesto que no se modifica.
}

void Print() {
   cout << global << endl;
    / / global puede ser accesada por cualquiera de las funciones del programa.}
```

Ejemplo 2

```
// A scope example
# include <iostream>
void a(void);
void b(void);
void c(void);
int x=1; //variable global

int main(){
  int x=5;//Variable local de Main
  cout<<"El valor de x dentro de main es: "<<x<<endl;
  {int x=7; //empieza un nuevo bloque
  cout<<"El valor de x dentro de un bloque en main es: " << x << endl;}
  //fin del bloque
  cout<<"El valor de x dentro de main es: "<<x<<endl;

  a(); // la funcion a tiene una variable local x
  b(); // la funcion b tiene un static local x
  c(); // la funcion c usa la variable global x

  a(); // la funcion a reinicializa automaticamente la variable local x
  b(); // la variable static local x retiene el valor previo
  c(); // la variable local x tambien retiene su valor previo

  cout<<"El valor de x dentro de main es: "<<x<<endl;
  return 0;}
void a(void){
```

```
  int x=25; //inicializa el valor de x cada vez que se llama la funcion
  cout<<endl<<"El valor de la variable local x en la funcion a es: "
      << x << endl;
  cout<<"Antes de entrar a la funcion a"<<endl;
  ++x;
  cout<<"El valor de la variable local x e s: "<<x<<endl<<"despues de
entrar a la funcion a"<<endl;}


void b(void){
  static int x=50; //Inicializa la variable solo la primera vez que se
llama la funcion b;
  cout<<endl<<"El valor de la variable local static x en la funcion b
es: "<<x<<endl;
  cout<<"Entrando a la funcion b"<<endl;
  ++x;
  cout<<"El valor de la variable local static x es:
"<<x<<endl<<"Saliendo de la funcion b"<<endl;}


void c(void){

  cout<<endl<<"El valor de la variable global x en la funcion c es: "
<< x <<endl;
  cout<<"Entrando a la funcion c"<<endl;
  x*=10;
  cout<<endl<<"El valor de la variable global x en la funcion c es: "
<< x <<endl;
  cout<<"Saliendo de la funcion c"<<endl; }
```

2. VALUE PARAMETERS AND REFERENCE PARAMETERS

Los parametros por valor son una copia del valor del argumento que se le pasa a la funcion al momento de llamarla. El valor de la copia puede cambiar o no, lo cual no afecta el valor de la variable que se uso para llamar la funcion.

Un parametro por referencia es un alias para el correspondiente argumento. Para indicar que un parámetro es por referencia en el tipo de parámetro en el “function prototype” se le antepone (&).

Ejemplo 3

```
//Example call-by-value and call-by-reference
#include<iostream>

int cuadradovalor(int);
void cuadradoreferencia(int &);

int main(){
  int x=2, z=4;
  cout<<"x= "<<x<<" Antes de llamar a la funcion Cuadrado por valor \n"
      <<"Valor retornado por cuadrado por valor: "
```

```
      <<cuadradovalor(x)<<endl
      <<"x= "<<x<<" Despues de llamar la funcion \n"<<endl;

  cout<<"z=  "<<z<<"  Antes  de  llamar  la  funcion  Cuadrado  por
referencia"<<endl;
  cuadradoreferencia(z);
  cout<<"z= "<<z<<" Despues de llamar a la funcion"<<endl;
  return 0;}

int cuadradovalor(int a){
  return a*=a;}

void cuadradoreferencia(int &cRef){
  cRef*=cRef;}
```

Ejercicios

1. El máximo común divisor de dos números es el entero mas grande que divida a los dos números. Escribe un programa utilizando funciones que calcule el máximo común divisor de dos números

2. Escribe un programa utilizando funciones qu e lea un número entero positivo y devuelva el número al revés. Por ejemplo, si el input es 7631, el output debe ser 1367.