	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Saavedra Hernandez Honorato

Asignatura: Fundamentos de programación

Grupo: 2

No de Práctica(s): Guía práctica de estudio 07: Fundamentos de Lenguaje C

Integrante(s): Rivas Ruiz Blanca Estela

Alamo Mendoza Maria Fernanda

Semestre: SEGUNDO SEMESTRE

Fecha de entrega: 10/04/18

Observaciones:

CALIFICACIÓN: _____

Guía práctica de estudio 07: Fundamentos de Lenguaje C ,

Objetivo:
Elaborar programas en lenguaje C utilizando las instrucciones de control de tipo secuencia, para realizar la declaración de variables de diferentes tipos de datos, así como efectuar llamadas a funciones externas de entrada y salida para asignar y mostrar valores de variables y expresiones.

Introducción:
Una vez que un problema dado ha sido analizado (se identifican los datos de entrada y la salida deseada), que se ha diseñado un algoritmo que lo resuelva de manera eficiente (procesamiento de datos), y que se ha representado el algoritmo de manera gráfica o escrita (diagrama de flujo o pseudocódigo) se puede proceder a la etapa de codificación. La codificación se puede realizar en cualquier lenguaje de programación estructurada, como lo son Pascal, Python, Fortran o PHP. En este curso se aprenderá el uso del lenguaje de programación C. Dentro del ciclo de vida del software, la implementación de un algoritmo se encuentra dentro de la etapa de codificación del problema. Esta etapa va muy unida a la etapa de pruebas así como tenemos el ciclo del software:

Desarrollo:

Lenguaje de programación C:

El lenguaje de C es de propósito general basado en el paradigma estructurado. El teorema del programa estructurado, demostrado por Böhm-Jacopini, dicta que todo programa puede desarrollarse utilizando únicamente 3 instrucciones de control:

Para crear un programa en C se siguen tres etapas principales: edición, compilación y ejecución.

- ***Edición:*** Se escribe el código fuente en lenguaje C desde algún editor de textos.
- ***Compilación:*** A partir del código fuente (lenguaje C) se genera el archivo en lenguaje máquina (se crea el programa objeto o ejecutable).
- ***Ejecución:*** El archivo en lenguaje máquina se puede ejecutar en la arquitectura correspondiente.

Un programa en C consiste en una o más funciones, de las cuales una de ellas debe llamarse main( ) y es la principal.

Licencia GPL de GNU:
se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
*/
```

Comentarios:Son esenciales para el lenguaje de programación realizar comentarios para documentar el programa. En C tenemos dos tipos de comentarios: el comentario por línea y el comentario por bloque.
El comentario por línea inicia cuando se insertan los símbolos '//' y termina con el salto de línea (hasta donde termine el renglón) El comentario por bloque inicia cuando se insertan los símbolos '/*' y termina cuando se encuentran los símbolos '*/'. Cabe resaltar que el comentario por bloque puede abarcar varios renglones.
Ejemplo:

```
//hack for ie browser (assuming that ie is a browser)
```

```
/**
* For the brave souls who get this far: You are the chosen ones,
* the valiant knights of programming who toil away, without rest,
* fixing our most awful code. To you, true saviors, kings of men,
* I say this: never gonna give you up, never gonna let you down,
* never gonna run around and desert you. Never gonna make you cry,
* never gonna say goodbye. Never gonna tell a lie and hurt you.
*/
```

```
//
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 16
//
```

```
// TODO: Fix this. Fix what?
```

```
// no comments for you
// it was hard to write
// so it should be hard to read
```

NOTA. Al iniciar el programa se deben agregar todas las bibliotecas que se van a utilizar en el mismo, es decir, funciones externas necesarias para ejecutar el programa. En lenguaje C la biblioteca estándar de entrada y salida está definida en 'stdio.h' (standard in out) y provee, entre otras, funciones para lectura y escritura de datos que se verán a continuación.

Declaración de variables:

Para poder declarar una variable dentro del lenguaje de C, se debe seguir la siguiente sintaxis:

[modificadores] tipoDeDato identificador [= valor];

Debe declarar el tipo de dato que puede contener la variable, debe declarar el identificador (nombre o etiqueta) con el que se va a manejar el valor y se puede asignar un valor inicial a la variable (opcional).

De la misma forma, es posible declarar varios identificadores de un mismo tipo de dato e inicializarlos en el mismo renglón, lo único que se tiene que hacer es separar cada identificador por comas.

Tipos de datos:

Los tipos de datos básicos son:

- Caracteres: codificación definida por la máquina.
- Enteros: números sin punto decimal.
- Flotantes: números reales de precisión normal.
- Dobles: números reales de doble precisión.

Las variables enteras que existen en lenguaje C son:

Tipo	*Bits*	*Valor Mínimo*	*Valor Máximo*
signed char	8	-128	127
unsigned char	8	0	255
signed short	16	-32 768	32 767
unsigned short	16	0	65 535
signed int	32	-2,147,483,648	2 147 483 647
unsigned int	32	0	4 294 967 295
signed long	64	9 223 372 036 854 775 808	9 223 372 036 854 775 807
unsigned long	64	0	18 446 744 073 709 551 615
enum	16	-32 768	32 767

Las variables reales que existen en lenguaje C son:

Tipo	*Bits*	*Valor Mínimo*	*Valor Máximo*
float	32	3.4 E-38	3.4 E38
double	64	1.7 E-308	1.7 E308
long double	80	3.4 E-4932	3.4 E4932

Para poder acceder al valor de una variable se requiere especificar el tipo de dato. Los especificadores que tiene lenguaje C para los diferentes tipos de datos son:

ejemplo:

```
{
int a,b;
scanf("%d %d, &a,&b);
}
```

Identificador: Un identificador es el nombre con el que se va a almacenar en memoria un tipo de dato. Los identificadores siguen las siguientes reglas, esta dos reglas son muy importantes:
-Debe iniciar con una letra [a-z].
-Puede contener letras [A-Z, a-z], números [0-9] y el carácter guión bajo (_).
NOTA: A pesar de que variables como 'areadeltriangulo' o 'perimetro_del_cuadrado' son declaraciones válidas como identificadores, es una buena práctica utilizar la notación de camello para nombrar las variables como convención. En la notación de camello los nombres de cada palabra empiezan con mayúscula y el resto se escribe con minúsculas (a excepción de la primera palabra, la cual inicia también con minúscula). No se usan puntos ni guiones para separar las palabras. Además, las palabras de las constantes se escriben con mayúsculas y se separan con guión bajo.

Código declaración de variables

```c
#include <stdio.h>
/*
   Este programa muestra la manera en la que se declaran y asignan variables
   de diferentes tipos: numéricas (enteras y reales) y caracteres.
*/

int main() {
    // Variables enteras
    short enteroNumero1 = 32768;
    signed int enteroNumero2 = 55;
    unsigned long enteroNumero3 = -789;
    char caracterA = 65;        // Convierte el entero (ASCII) a carácter
    char caracterB = 'B';

    // Variables reales
    float puntoFlotanteNumero1 = 89.8;
    // La siguiente sentencia genera un error al compilar
    // porque los valores reales siempre llevan signo
    // unsigned double puntoFlotanteNumero2 = -238.2236;
    double puntoFlotanteNumero2 = -238.2236;
    return 0;
}
```

La biblioteca 'stdio.h' contiene diversas funciones tanto para imprimir en la salida estándar (monitor) como para leer de la entrada estándar (teclado).
printf es una función para imprimir con formato, es decir, se tiene que especificar entre comillas el tipo de dato que se desea imprimir, también se puede combinar la impresión de un texto predeterminado:
printf("El valor de la variable real es: %lf", varReal);
scanf es una función que sirve para leer datos de la entrada estándar (teclado), para ello únicamente se especifica el tipo de dato que se desea leer entre comillas y en qué variable
se quiere almacenar. Al nombre de la variable le antecede un ampersand (&), esto indica

que el dato recibido se guardará en la localidad de memoria asignada a esa variable.
scanf ("%i", &varEntera);
Para imprimir con formato también se utilizan algunas secuencias de caracteres de escape,
C maneja los siguientes:
\a carácter de alarma
\b retroceso
\f avance de hoja
\n salto de línea
\r regreso de carro
\t tabulador horizontal
\v tabulador vertical
‘\0’ carácter nulo

Código almacenar e imprimir variables:

```c
#include <stdio.h>
/*
   Este programa muestra la manera en la que se declaran y asignan variables
   de diferentes tipos: numéricas (enteras y reales) y caracteres, así como
   la manera en la que se imprimen los diferentes tipos de datos.
*/

int main() {
    /* Es recomendable al inicio declarar
       todas las variables que se van a utilizar
       en el programa */
    // variables enteras
    int enteroNumero;
    char caracterA = 65;        // Convierte el entero a carácter (ASCII)

    // Variable reales
    double puntoFlotanteNumero;

    // Asignar un valor del teclado a una variable
    printf("Escriba un valor entero: ");
    scanf("%d", &enteroNumero);
    printf("Escriba un valor real: ");
    scanf("%lf", &puntoFlotanteNumero);

    // Imprimir los valores con formato
    printf("\nImprimiendo las variables enteras:\a\n");
    printf("\tValor de enteroNumero = %i\n", enteroNumero);
    printf("\tValor de caracterA = %c\n",charA);
    printf("\tValor de puntoFlotanteNumero = %lf\n", puntoFlotanteNumero);

    printf("\nValor de enteroNumero en base 16 = %x\a\n", enteroNumero);
    printf("\tValor de caracterA en código hexadecimal = %i\n", enteroNumero);
    printf("\tValor de puntoFlotanteNumero en notación científica = %e\n",
    puntoFlotanteNumero);
    // La función getchar() espera un carácter para continuar la ejecución
    getchar();
    return 0;
}
```

Modificadores de alcance:
Se pueden agregar al inicio utilizando:

const: const impide que una variable cambie su valor durante la ejecución del programa, es decir, permite para crear constantes. Por convención, las constantes se escriben con mayúsculas y se deben inicializar al momento de declararse.indica que la variable permanece en memoria desde su creación y durante toda la ejecución del programa, es decir, permanece estática en la memoria.

Código variables estáticas y constantes:

El modificador **static** indica que la variable permanece en memoria desde su creación y durante toda la ejecución del programa, es decir, permanece estática en la memoria.

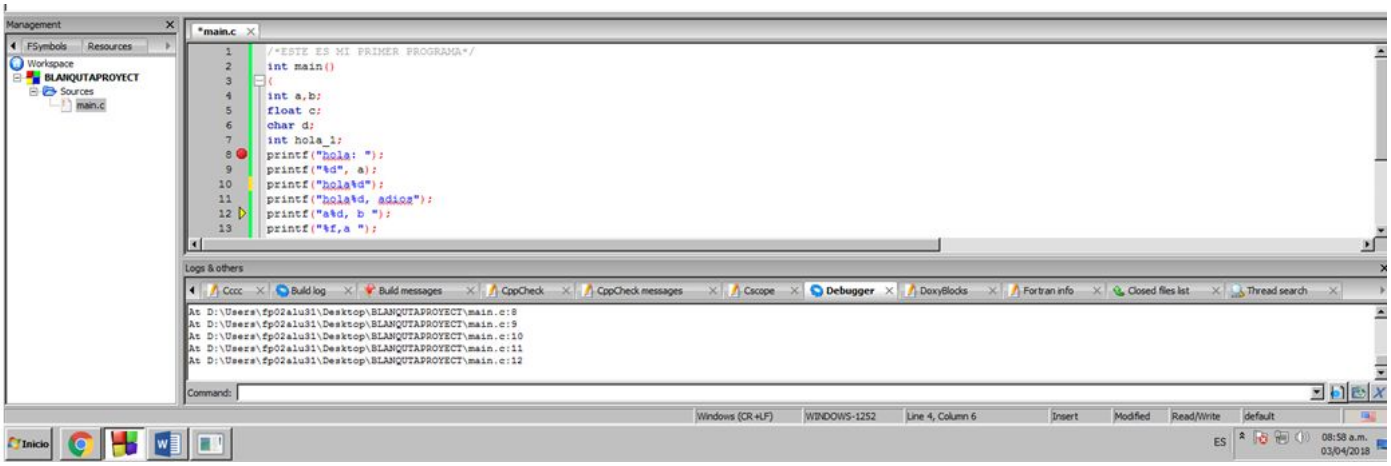
Ejemplo

```
static int num;
static float resultado = 0;
```

NOTA: Cuando se llegue al tema de funciones se verá la utilidad de las variables estáticas.

Código variables estáticas y constantes

```
#include <stdio.h>
/*
   Este programa muestra la manera en la que se declaran y asignan
   las variables estáticas y las constantes.
*/
int main() {
    const int constante = 25;
```



Operadores:

Los operadores aritméticos que maneja lenguaje C se describen en la siguiente tabla:

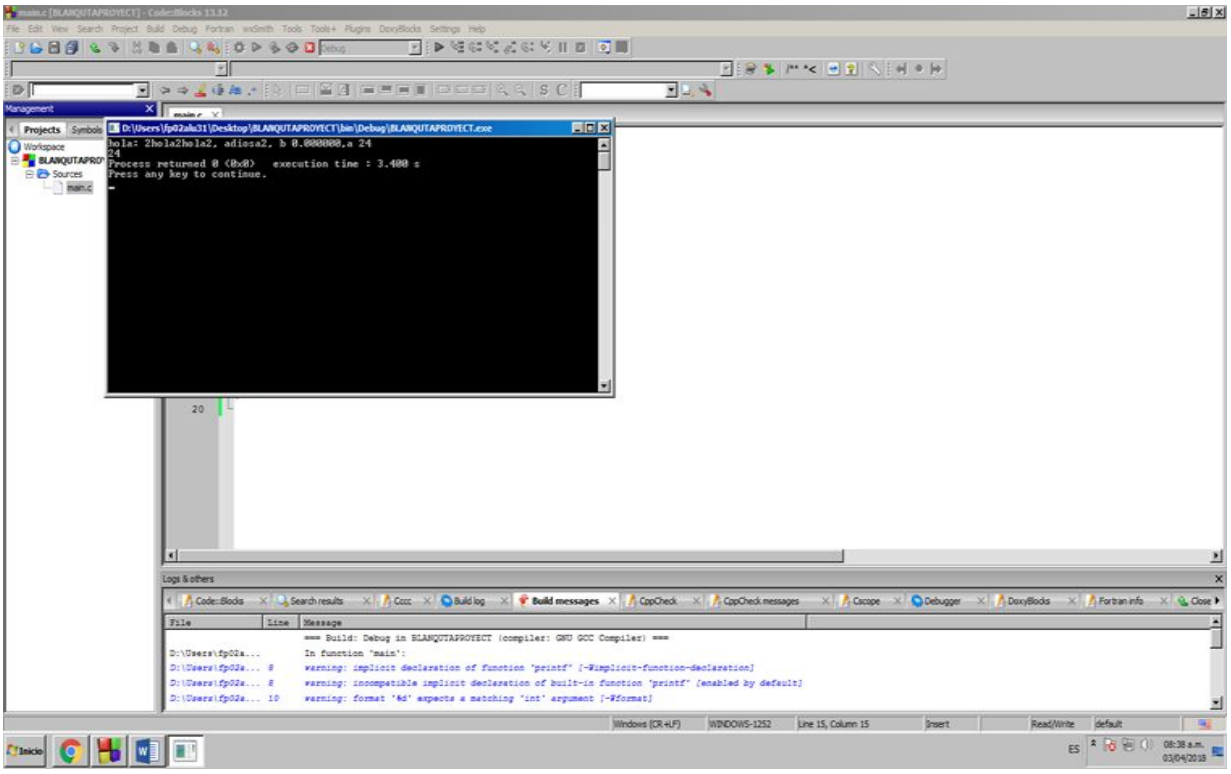
Operador	*Operación*	*Uso*	*Resultado*
+	Suma	125.78 + 62.5	188.28
-	Resta	65.3 - 32.33	32.97
*	Multiplicación	8.27 * 7	57.75
/	División	15 / 4	3.75
%	Módulo	4 % 2	0

Los operadores lógicos a nivel de bits que maneja el lenguaje C se describen en la siguiente tabla:

Moldeo o cast :

Código operadores:

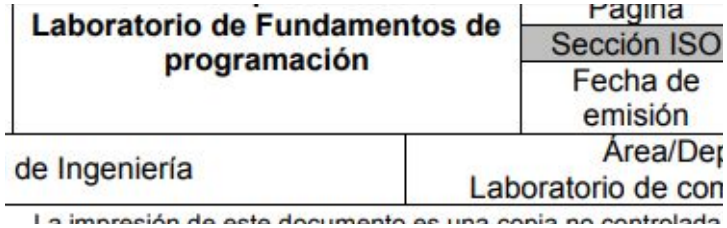
Este programa muestra la manera en la que se realiza un moldeado o cast. También muestra cómo manipular números a nivel de bits:
- Corrimiento de bits a la izquierda y a la derecha
- Operador AND a nivel de bits - Operador OR a nivel de bits



Expresiones lógicas:
Las expresiones lógicas están constituidas por números, caracteres, constantes o variables que están relacionados entre sí por operadores lógicos. Una expresión lógica puede tomar únicamente los valores verdadero o falso.

Los operadores de relación permiten comparar elementos numéricos, alfanuméricos, constantes o variables

Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples.



Código expresiones lógicas:

Este programa ejemplifica el manejo de operaciones relacionales y los operadores lógicos. También muestra la manera de incrementar o decrementar una variable y la diferencia entre hacerlo antes (pre) o después (pos).

```
int main (){
    int num1, num2, res;
    char c1, c2;
    double equis = 5.5;
    num1 = 7;
    num2 = 15;

    c1 = 'h';
    c2 = 'H';

    printf("Expresiones de relación\n");
    printf("¿ num1 es menor a num2 ? -> \t%d\n",num1<num2);
    printf("¿ c1 es igual a c2 ?     -> \t%d\n",c1==c2);
    printf("¿ c1 es diferente a c2 ? -> \t%d\n",c1!=c2);

    printf("\nExpresiones lógicas\n");
    printf("¿ num1 es menor a num2 Y c1 es igual a 'h' ? -> \t%d\n",
    num1<num2 && c1 == 'h');
    printf("¿ c1 es igual a 's' O c2 es igual a 'H' ?    -> \t%d\n",
    c1 == 's' || c2 == 'H');
    printf("¿ c1 es igual a 's' O c2 es igual a 'S' ?    -> \t%d\n",
    c1 == 's' || c2 == 'S');

    printf("\nIncrementos y decrementos\n");
    printf("num1++  -> \t%d\n",num1++);
    printf("--num1  -> \t%d\n",--num1);
    printf("++equis -> \t%lf\n",++equis);

    return 0;
```

Depuración de programas:

Cuando un programa falla (no termina su ejecución de manera correcta) y la información enviada por el compilador es muy general, se puede ejecutar el programa en un contexto controlado para saber, exactamente, dónde está fallando.

Se revisará este tema en la guía práctica de estudio “Depuración de programas” para conocer las diferentes herramientas que nos ayudan a encontrar los errores de un programa.