	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.I. Marco Antonio Martínez Quintana

Asignatura: Fundamentos de Programación

Grupo: 3

No de Práctica(s): 7. Fundamentos de lenguaje C.

Integrante(s): Hernández Vázquez Daniela

No. de Equipo de cómputo empleado: No aplica

No. de Lista o Brigada: 26

Semestre: 1

Fecha de entrega: Lunes 16 de noviembre de 2020

Observaciones:

CALIFICACIÓN: _____

Práctica 7: 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.

Actividades:

- Crear un programa en lenguaje C que tenga definidas variables de varios tipos, se les asigne valores adecuados (por lectura o asignación directa) y muestre su valor en la salida estándar.
- En un programa en C, asignar valores a variables utilizando expresiones aritméticas; algunas con uso de cambio de tipo (cast)
- Elaborar expresiones relacionales/lógicas en un programa en C y mostrar el resultado de su evaluación.

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*:

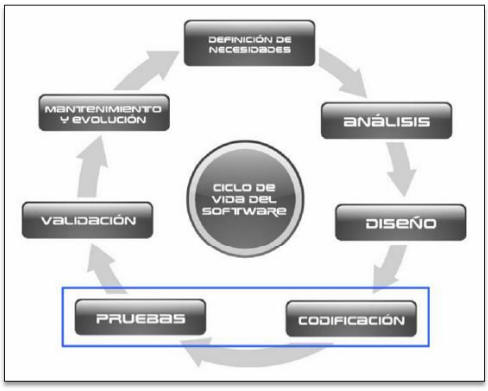

Figura 1: Ciclo de vida del software, resaltando las etapas de *codificación* y *pruebas*, las cuales se cubrirán en esta práctica.

Lenguaje de programación C

El proceso de desarrollo del lenguaje C se origina con la creación de un lenguaje llamado BCPL, que fue desarrollado por Martin Richards.
BCPL tuvo influencia en un lenguaje llamado B, el cual se usó en 1970 y fue inventado por Ken Thompson, esto permitió el desarrollo de C en 1971, el cual lo inventó e implementó Dennis Ritchie.
C es un lenguaje de programación de propósito general que ofrece como ventajas economía de expresión, control de flujo y estructuras de datos y un conjunto de operadores.
C es un lenguaje 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:

- Secuencia
- Selección
- iteración

Por otro lado, C es un lenguaje compilado, es decir, existe un programa (llamado compilador) que, a partir de un código en lenguaje C, genera un código objeto (ejecutable).
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.
Al momento de ejecutar un programa objeto (código binario), se ejecutarán únicamente las instrucciones que estén definidas dentro de la función principal. La función principal puede contener sentencias, estructuras de control y comentarios. Dentro de las sentencias se encuentran la declaración y/o asignación de variables, la realización de operaciones básicas, y las llamadas a funciones.

Licencia GPL de GNU

```
/*
 *
 * 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

Es una buena práctica en cualquier lenguaje de programación realizar comentarios para documentar el programa. En C existen 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

```
// Comentario por línea
// Otro comentario por línea
/* Comentario por bloque */
/* Comentario por bloque
   que puede ocupar
   varios renglones */
```

Codigo con comentarios

```
#include <stdio.h>

main() {
    // Este código compila y ejecuta
    /* pero no muestra salida alguna
       debido a que un comentario
       ya sea por línea o por bloque */
    // no es tomado en cuenta al momento
    // de compilar el programa,
    /* sólo sirve como documentación en el */
    /*
       código fuente
    */
}
```

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 declarar variables en C se sigue la siguiente sintaxis:

[modificadores] tipoDeDato identificador [= valor];

Por lo tanto, una variable puede tener modificadores (éstos se analizarán más adelante y son opcionales), 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).

También 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.

tipoDeDato identificador1[= valor], identificador2[= valor];

Tipos de datos

Los tipos de datos básicos en C son:

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

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

Si se omite el clasificador por defecto se considera 'signed'.

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

Las variables reales siempre poseen signo.
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:

Tipo de dato	*Especificador de formato*
Entero	%d, %i, %ld, %li, %o, %x
Flotante	%f, %lf, %e, %g
Carácter	%c, %d, %i, %o, %x
Cadena de caracteres	%s

El especificador de dato se usa para guardar o imprimir el valor de una variable.
Para imprimir un valor entero en base 10 se pueden usar los especificadores %d o %i (%ld ó %li para enteros largos), para imprimir un valor entero en base 8 se utiliza el especificador %o y para imprimir un valor entero en base 16 se utiliza el especificador %x.
Un valor real se puede imprimir con el especificador %f para reales de precisión simple, %lf para reales de doble precisión, %e (notación científica) y %g (redondea la parte fraccionaria a 3 dígitos significativos).
Una variable de tipo carácter se puede imprimir con el especificador %c, con los especificadores %i o %d para imprimir el valor del código ASCII del carácter en base 10, con el especificador %o para imprimir el valor del código ASCII del carácter en base 8 o con el especificador %x para imprimir el valor del código ASCII del carácter en base 16.
El lenguaje C también permite manejar cadenas de caracteres y éstas se pueden imprimir con el especificador %s.
NOTA: Las cadenas de caracteres se manejarán cuando se aborde el tema de arreglos.

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:

- 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 guion 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:

```c
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.

```c
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",caracterA);
    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

Los modificadores que se pueden agregar al inicio de la declaración de variables son **const** y **static**.
El modificador **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.

Ejemplo

```
const int NUM_MAX = 1000;
const double PI = 3,141592653589793 3;
```

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

```c
#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;
    static char a = 'a';
    printf("Valor constante: %i\n", constante);
    printf("Valor estático: %c\n", a);
    // El valor de la variable declarada como constante no puede cambiar.
    // La siguiente línea genera un error al compilar si se quita el comentario:
    // constante = 30;
    // las variables estáticas sí pueden cambiar de valor
    a = 'b';
    printf("\nValor estático: %c\n", a);
    return 0;
}
```

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:

Operador	*Operación*	*Uso*	*Resultado*
>>	Corrimiento a la derecha	8 >> 2	2
<<	Corrimiento a la izquierda	8 << 1	16
&	Operador AND	5 & 4	4
|	Operador OR	3 | 2	3
~	Complemento ar-1	~2	1

Moldeo o cast

El resultado de una operación entre dos tipos de datos iguales puede dar como resultado un tipo de dato diferente, en esos casos es necesario moldear el resultado. A este proceso se le conoce como *cast*.

Ejemplo:

```
// Si se tienen 2 enteros
int cinco = 5, dos = 2;

// La operación de división entre dos enteros
// genera un valor real, en este caso hay que
// moldear (cast) el resultado del lado derecho del
// igual para que corresponda con el lado izquierdo
// y se pueda asignar.
double res = (double)cinco/dos;

// Si no se hiciese el cast el resultado se truncaría.
```

Código operadores

```c
#include <stdio.h>

/*
   Este programa muestra la manera en la que se realiza un moldeo o cast.
   También muestra como 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
*/

int main(){
    short ocho, cinco, cuatro, tres, dos, uno;

    // 8 en binario: 0000 0000 0000 1000
    ocho = 8;
    // 5 en binario: 0000 0000 0000 0101
    cinco = 5;
    // 4 en binario: 0000 0000 0000 0100
    cuatro = 4;
    // 3 en binario: 0000 0000 0000 0011
    tres = 3;
    // 2 en binario: 0000 0000 0000 0010
    dos = 2;
    // 1 en binario: 0000 0000 0000 0001
    uno = 1;
    printf("Operadores aritméticos\n");
    double res = (double)cinco/dos;        // Cast
    printf("5 / 2 = %lf\n",res);
    printf("5 modulo 2 = %d\n",cinco%dos);
    printf("Operadores lógicos\n");
    printf("8 >> 2 = %d\n",ocho>>dos);
    printf("8 << 1 = %d\n",ocho<<1);
    printf("5 & 4 = %d\n",cinco&cuatro);
    printf("3 | 2 = %d\n",tres|dos);

    printf("\n");
    return 0;
}
```

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.

Operador	*Operación*	*Uso*	*Resultado*
==	Igual que	'h' == 'H'	Falso
!=	Diferente a	'a' != 'b'	Verdadero
<	Menor que	7 < 15	Verdadero
>	Mayor que	11 > 22	Falso
< =	Menor o igual	15 <= 22	Verdadero
> =	Mayor o igual	20 >= 35	Falso

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

Operador	*Operación*	*Uso*
!	No	! p
&&	Y	a > 0 && a < 11
||	O	opc == 1 || salir != 0

Lenguaje C posee operadores para realizar incrementos y decrementos de un número.

El operador ++ agrega una unidad (1) a su operando. Es posible manejar preincrementos (++n) o posincrementos (n++).

El operador – resta una unidad (1) a su operando. Se pueden manejar predecrementos (--n) o posdecrementos (n--).

NOTA: Lenguaje C maneja los resultados booleanos (Verdadero o falso) con números enteros, cuando el resultado de una comparación es falso el valor regresado es cero, cuando la comparación es verdadera el valor regresado es 1.

Código expresiones lógicas

```c
#include <stdio.h>
/*
   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.

Conclusiones

Como la introduccion nos señalaba al principio, una vez que ya ha sido idetificado y debidamente analizado un problema y se ha diseñado un algoritmo para su resolucion, se puede proceder a la etapa de codificación. Este es un paso muy, pero muy importante ya que como lo hemos visto en practicas anteriores podemos ver este como una parte de "la caja negra", aquel paso faltante entre la introducción de los datos y la recepción de los datos de salida.
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, ya que de noser bien realizada la codificación las pruebas fallarán.

Referencias

- El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

- Carlos Guadalupe (2013). Aseguramiento de la calidad del software (SQA). [Figura 1]. Consulta: Junio de 2015. Disponible en: https://www.mindmeister.com/es/273953719/aseguramiento-de-la-calidad-del-software-sqa