Universidad Nacional Autónoma de México

Facultad de Ingeniería

Estructura de Datos y Algoritmos I

Actividad 1: Acordeón lenguaje C y MATLAB

Sánchez Hernández Marco Antonio

Fecha: 28/febrero/2021

Lenguaje C

Es importante mencionar que, el acordeón fue diseñado acorde al sistema operativo Linux, ya que este sistema será el utilizado en la realización de las prácticas de laboratorio, tareas y proyectos.

Comandos para compilar archivos mediante el compilador GCC

Todo archivo que contengan código fuente escrito en lenguaje C debe tener la terminación .c, de lo contrario, este no será compilado por ningún compilador (incluyendo gcc).

-Compilar y generar un ejecutable (el nombre de este será a.out).
gcc archivo.c

-El argumento *-o* permite asignar un nombre para el ejecutable.
gcc archivo.c -o nomdreAsignadoAlEjecutable

-El comando *-lNombreLibreria* incluye una librería nombrada que no forme parte de la librería estándar del lenguaje C.
gcc archivo.c -o archivo -lNombreLibreria

Una vez compilado el código, y generado el ejecutable, este último puede ser ejecutado, por medio de la terminal de comandos, haciendo uso del siguiente comando.
./nombreDelEjecutable

-El argumento *-g* indicará que el ejecutable que se genere debe contener información para depuración.
gcc -g archivo.c -o nomdreAsignadoAlEjecutable

Comandos para el depurador GDB

list / l //Permite listar 10 líneas del código.

b n //Establece un punto de ruptura.

d n //Elimina un punto de ruptura establecido.

clear //Elimina todos los puntos de ruptura.

info line n //Muestra información de la línea indicada.

run / r //Ejecuta el programa hasta donde exista un punto de ruptura, o hasta que termine.

c //Continua la ejecución.

s //Ejecuta la siguiente instrucción que esté después de un punto de ruptura.

p o print <nombreVariable> //Imprime el valor que contenga en ese momento la variable.

ignore n //Ignora el punto de ruptura indicado.

q / quit //Termina la ejecución de GDB.

Sintaxis del lenguaje C

// //Comentario de una sola línea.

/* <comentario>*/ //Comentario de múltiples líneas.

#include <nombreLibrería.h> //Incluye la librería especificada y permite el uso de las funciones definidas en esta.

Todas las líneas de código escritas, excepto los comentarios, líneas que solo contengan el carácter *{*, líneas que terminen en *}*, encabezados y constantes simbólicas, deberán terminar con *;*

Todo programa escrito en lenguaje C, debe contener la función *main*, en esta función es donde se iniciará la ejecución de un programa. La función main puede recibir un arreglo de caracteres como argumento al momento de ejecutar el código. Su sintaxis es la siguiente:

```
int main(int argc, char**argv) {

        <bloque de código>

    return(valor);

}
```

Para ingresar un argumento en la función *main*, este debe ser escrito junto al comando de ejecución en la terminal:
./nombreDelEjecutable argumento1 argumento2 ...

return() indica el valor que será regresado una vez se termine de ejecutar el código dentro de la función.

tipoDeDato identificador; //Declarar una variable.

tipoDeDato identificador = valor; //Declarar una variable y asignar un valor.

const tipoDeDato identificador; //Declara una constante a la que se le debe asignar un valor inicial, este no cambiará durante toda la ejecución del código.

const tipoDeDato identificador = valor; //Declara y asigna un valor constante que no será modificado durante toda la ejecución del código.

#define <nombre> <valor> //Constante simbólica.

tipoDeDato identificador [tamaño]; //Declara un arreglo unidimensional de cierto tamaño.

tipoDeDato identificador [tamaño] [tamaño] … [tamaño]; //Declarar un arreglo multidimensional.

tipoDeDato * apuntador, variable; //Declara un apuntador del tipo indicado.

apuntador = &variable; //Asigna la dirección de memoria de la variable al apuntador.

Sintaxis de una función

Firma de una función
valorDeRetorno nombre (parámetros);

Definición de función
valorDeRetorno nombre (parámetros){

    <Bloque de código>

}

static //Hace que el valor de una variable sea siempre el mismo desde el inicio hasta el final de la ejecución de un programa.

break; //Termina la ejecución de un bloque de código dentro de una estructura de repetición, antes de que este termine.

continue; //Reanuda la ejecución de un bloque de código dentro de una estructura de repetición.

++n //Preincremento.

n++ //Postincremento.

--n //Predecremento.

n-- //Postdecremento.

Estructura de control selectiva if-else:

if (expesión_lógica) {

    <Bloque de código que se ejecuta si la condición es verdadera>

}

else {

    <Bloque de código que se ejecuta si la condición es falsa>

```
}
```

Estructura de control selectiva switch:

```
switch (opción_a_evaluar) {

    case valor1:
        <Bloque de código a ejecutar>
    break;

    case valor2:
        <Bloque de código a ejecutar>
    break;

    …

    case valorN:
        <Bloque de código a ejecutar>
    break;

}
```

```
condición ? siSeCumple : siNoSeCumple //Operador ternario
```

Estructura de control repetitiva while

```
while (expresión_lógica){

    <Bloque de código a ejecutar>

}
```

Estructura de control repetitiva do-while

```
do {

    <Bloque de código a ejecutar>

} while (expresión_lógica);
```

Estructura de control repetitiva for

```
for (inicialización; expresión_lógica; operación por iteración) {

    <Bloque de código a ejecutar>
```

```
}
```

Tipos de datos

Caracteres

Tipo	Bits	Valor mínimo	Valor máximo
signed char	8	-128	127
unsigned char	8	0	255

Enteros

Tipo	Bits	Valor mínimo	Valor máximo
short	16	-32 768	32 767
unsigned short	16	0	65 535
int	16	-2,147,483,648	2,147,483,647
unsigned int	16	0	4 294 967 295
long	32	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
unsigned long	32	0	18 446 744 073 709 551 615

Flotantes

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

Otros

Tipo	Bits	Valor mínimo	Valor máximo
enum	16	-32 768	32 767
bool	8	0	1

Especificadores de datos

Tipo	Especificador	Uso
short unsigned short int unsigned int	%d, %i	Guardar o imprimir un valor entero en base 10
long unsigned long	%ld, %li	Guardar o imprimir un valor entero largo en base 10
unsigned short unsigned int	%o	Guardar o imprimir un valor en base 8
unsigned long	%lo	Guardar o imprimir un valor en base 8
unsigned short unsigned int	%x	Guardar o imprimir un valor en base 16
unsigned long	%lx	Guardar o imprimir un valor en base 16
float double	%f	Guardar o imprimir un valor real
long double	%lf	Guardar o imprimir un valor real

float double	%e	Guardar o imprimir un valor real en notación científica
long double	%Le	Guardar o imprimir un valor real en notación científica
float double	%g	Guardar o imprimir un valor real redondeando a tres cifras significativas
long double	%Lg	Guardar o imprimir un valor real redondeando a tres cifras significativas
char unsigned char	%c	Guardar o imprimir un carácter
unsigned char	%d, %i	Guardar o imprimir un carácter en código ASCII en base 10
unsigned char	%o	Guardar o imprimir un carácter en código ASCII en base 8
unsigned char	%x	Guardar o imprimir un carácter en código ASCII en base 16
char	%s	Guardar o imprimir un una cadena de caracteres

Secuencia de caracteres de escape

Secuencia de caracteres de escape	**Función**
\a	Alarma
\b	Retroceso
\f	Avanzar hoja
\n	Salto de línea
\r	Retroceso de carro
\t	Tabulador horizontal
\v	Tabulador vertical
‘\0’	Carácter nulo

Operadores aritméticos

Operador	**Operación**
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto o módulo

Operadores lógicos a nivel de bits

Operador	**Operación**
>>	Corrimiento a la derecha

<<	Corrimiento a la izquierda
&	Operador AND
|	Operador &
~	Complemento ar-1

Operadores lógicos

Operador	Operación
!	No
&&	Y
||	O

Operadores de relación

Operador	Operación
==	Igual que
!=	Diferente de
<	Menor que
>	Mayor que
<=	Menor igual que
>=	Mayor igual que

funciones de la librería stdio.h

printf(“cadena de caracteres”); //Imprime una cadena de caracteres en la salida estándar.

printf(“%identificador”, nombreDeVariable); //Imprime el valor de una variable con el formato especificado por el identificador en la salida estándar.

scanf(“%identificador”, &nombreDeVariable); //Lee de la entrada estándar el valor y lo asigna, con el formato dado por el identificador, a la variable especificada.

FILE * nombre; //Apuntador hacia un archivo.

*FILE fopen (char*nombre_archivo, char*modo); //Abre un archivo en determinado modo.

fclose(FILE* apArch); //Cierra el archivo abierto por un apuntador.

Modos para abrir un archivo

Modo	Descripción
r	Abre un archivo existente en modo lectura
w	Abre o crea un archivo en modo escritura
a	Abre un archivo existente en para añadir datos
r+	Abre un archivo existente en modo lectura /

	escritura
w+	Crea un archivo en modo lectura / escritura
a+	Abre o crea un archivo en modo lectura / escritura
rb	Abre un archivo existente en modo lectura y binario
wb	Abre o crea un archivo existente en modo lectura y binario

char*fgets(char*buffer,int tamaño, FILE* apArch); //Lee una cadena de caracteres desde un archivo especificado.

char*fputs(char*buffer, FILE* apArch); //Escribe una cadena de caracteres en un archivo especificado.

int fprintf(FILE*apArch,char*formato,...); //La función es igual que printf();, sin embargo, la salida será en un archivo.

int fscanf(FILE*apArch,char*formato,...); //La función es igual que scanf();, sin embargo, la entrada provendrá de un archivo.

int fread(void*ap,size_t tam, size_t nelem,FILE*archivo); //Lee de uno o varios elementos que poseen la misma longitud a partir de un apuntador.

int fwrite(void*ap,size_t tam, size_t nelem,FILE*archivo); //Escribe en un archivo uno o varios elementos que poseen la misma longitud almacenados en un apuntador.

MATLAB (básico)

Comandos para la ventana de comandos

Para compilar y ejecutar un código escrito en MATLAB es necesario que el archivo que lo contenga posea la terminación .m

archivo.m

clc - Limpia la ventana de comandos.

Clear - Limpia las variables que se encuentren en el Workspace.

close all - Cierra todas las ventanas abiertas desde la ventana de comandos.

Sintaxis

% o %% - Comentario de una línea.

<código>; - No mostrará el resultado en la terminal de comandos de esa línea.

… - Permite escribir el código en líneas separadas.

format short g – Redondea los puntos flotantes a una sola cifra significativa.

Tipos de datos

En MATLAB, a diferencia de C, no es necesario especificar el tipo de dato para cada variable, MATLAB interpretará el tipo de dato a partir del valor asignado a esta.

Numéricos: todos los valores numéricos ingresados serán interpretados por MATLAB como punto flotante de doble precisión.

Caracteres y cadenas: para asignar el valor a una variable se debe escribir entre comillas simples ‘ ’ o dobles “ ”.

Operadores aritméticos

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
mod	Resto o módulo
^	Potencia

Operadores lógicos

Operador	Operación
~	No
&&	Y
||	O

Condicional if-else

```
if <expresión>
    <Bloque de código>
elseif <expresión>
    <Bloque de código>
else
    <Bloque de código>
end
```

Bucle for

```
for <expresión>
    <Bloque de código>
end
```

Bucle while

```
while <expresión>
    <Bloque de código>
end
```

function [y1, …, yN] = nombreFuncion(x1, …, xN) //Declarar una función.

Vectores

nombre = [n, m, o, …] //Declara un vector de un cierto tamaño, puede ser de cualquier tipo de dato, en lugar de comas pueden usarse espacios para separar los elementos que componen al vector.

nombre = (n) //Consultar el elemento contenido en el índice n.

Nota: el índice en MATLAB inicia en 1, a diferencia de los demás lenguajes de programación, como C, donde el primer índice es 0.

nombre = n:o:p //Declara un vector que inicia en n y que termine en m incrementando en o.

nombre linspace (n, m, o) //Declara un vector que inicia en n, que termina en m y que contiene o puntos, es decir, se divide en 10 puntos.

vectorUno + vectorDos //Suma de dos vectores.

vectorUno – vectorDos //Resta de dos vectores.

vector + n //n es un número, se sumará el número a cada componente del vector.

vector – n //n es un número, se resta el número a cada componente del vector.

vector * n //n es un número, corresponde al producto escara de un vector.

vector / n //n es un número, se divide cada una de las componentes del vector.

vectorUno .* vectorDos //Multiplicación entre dos vectores.

vectorUNo ./ vectorDos //División entre dos vectores.

Nota: para realizar operaciones entre vectores se debe tener en cuenta que, los vectores a operar deben ser del mismo tamaño, de lo contrario, se mostrará un error.

vector = [vectorUno vectorDos … vectorN] //Concatenar vectores.

vector = [vector vectorUno … vectorN] //Concatena un nuevo vector al vector especificado.

Matrices

nombre = [<elementos primer renglón> ; <elementos segundo renglón> ; … <elementos renglón n>] //Declara una variable de n renglones y m columnas.

nombre (<renglón>, <columna>) //Permite consultar el valor almacenado en el índice indicado.

matriz1 + matriz2 //Suma de matrices.

matriz1 – matriz2 //Resta de matrices.

matriz1 .* matriz2 //Multiplicación de matrices término a término.

matriz1 ./ matriz2 //División de matrices término a término.

Nota: para realizar estas operaciones, las matrices deben ser del mismo orden. Las operaciones entre un escalar y una matriz se realizarán término a término y con los mismo operadores que los anteriores.

matriz1 * matriz2 //Multiplicación de matrices (obedece las reglas del álgebra de matrices).

Plot

plot(x, y) //Plotear una gráfica en 2D.

plot(x, y, 'LineWidth', n) //Plotear una gráfica en 2D con un grosos de línea n.

plot(x, y, '<Color>') //Plotear una gráfica en 2D de un cierto color.

plot(x, y, 'b--') //Plotear una gráfica en 2D con una línea discontinua.

plot(x, y, 'b:') //Plotear una gráfica en 2D con una línea punteada.

plot(x, y, 'Marker','o') //Plotear una gráfica en 2D con marcadores circulares en los puntos de intersección del eje x y el eje y.

plot(x ,y, 'Marker', 'o', 'MarkerEdgeColor', '<color>') //Plotear una gráfica en 2D con marcadores circulares en los puntos de intersección del eje x y el eje y de un cierto color especificado.

Nota: Todos los argumentos anteriores pueden ser mezclados en una sola línea de código.

Referencias

Ayra B. (s. f.). Opengenus. *Boolean (bool or_Bool) datatype in* C. Recuperado el 28 de febrero de 2021, de https://iq.opengenus.org/boolean-in-c/

MathWorks (s. f.). *Tipos de Datos*. Recuperado el 28 de febrero de 2021, de https://la.mathworks.com/help/matlab/data-types.html

Solano J., García E., Nakayama A., Arteaga T., Castañeda M. (2018). Laboratorio de Computación Salas A y B. *Manual de prácticas del laboratorio de Fundamentos de programación*. Recuperado de http://lcp02.fi-b.unam.mx/

Vachev T. (s. f.). Udemy. *MATLAB Basics for Beginners – Learn from Top Experts*. Recuperado el 28 de febrero de 2021, de https://www.udemy.com/share/101voQAEceclhaQH0B/