Universidad Nacional Autónoma de México

Facultad de ingeniería

Asignatura: Estructura de Datos y Algoritmos I

Actividad asíncrona 2: Acordeón de Lenguaje C y Ml

Alumna: María Guadalupe Martínez Pavón

Grupo:15

Fecha:01-03-2021

Acordeón de lenguaje C

Editores, compiladores y ejecución

Editores:
Un programa en C debe ser escrito en un editor de texto permite copiar, guardar, editar, ordenar un código, para después generar un programa ejecutable en la computadora por medio de un compilador.

Compiladores:
Un compilador es un tipo de traductor que transforma un programa entero de un lenguaje de programación a otro, estos dependen totalmente del hardware de la computadora y el sistema operativo que corre sobre ella.

LINUX:

En primer lugar, debemos escribir el texto del programa. Para ello ejecutamos cualquier procesador de texto y creamos el fichero bienvenida.c con el texto que hemos visto en el ejemplo anterior. Guardamos el fichero. A continuación, compilamos, el programa con el compilador de C de GNU: gcc.

Para hacerlo, abrimos una consola y desde el directorio donde hayamos guardado el fichero bienvenida.c escribimos la orden:

gcc bienvenida.c -o bienvenida.exe

Esta orden crea el fichero ejecutable bienvenida, que podemos ejecutar simplemente escribiendo

bienvenida.exe

Ejecutores:
Una vez compilado el programa, se puede distribuir para equipos que ejecuten el mismo sistema operativo y tengan la misma plataforma de hardware

Tipos de datos

Tipo	Formato	Descripción
Void		
Char	%c, %d, %i, %o, %x	codificación definida por la máquina.
Int	%d, %i, %ld, %li, %o, %x	números sin punto decimal.
Float	%f, %lf, %e, %g	números reales de precisión normal
Double	%s	números reales de doble precisión.

Declaración de variables

int dato;

char inicial;

char nombre [50];

;oat decimales;

int notas [15];

Operadores

Operadores	Uso
+	Suma
-	Resta
*	Multiplicación
/	División
%	Modulo
>>	Corrimiento a la derecha
<<	Corrimiento a la izquierda
&	Operador AND
|	Operador OR
~	Complemento ar-1

Expresiones lógicas

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

Comentarios

// Todas las líneas que empiezan por // son comentarios que sirven para explicar cómo es el // programa.

Estructura general del programa

#include <st-dio.h> /* Cabecera estándar de entrad-a/s-alida de datos/*

Prototipos /* Aquí se definen los prototipos de las funciones se usarán dentro de la función main /*

int main(){ // Función principal

instru-cci-ones;

return 0; // Buena praxis de progra-mación

}

Entrada y salida de datos:

printf( ): Muestra en pantalla los datos

scanf(-"-%i",-&v-ari-able): Guarda un dato que el ususario digitó

gets( ): Se usa para cadenas de strings, dado que el scanf solo lee hasta que haya un espacio

puts( ): Muestra datos en pantalla pero solo si está dentro de un condic-ional

Estructura de control:

Sentencia if: En esta estructura se evalúa la expresión lógica y, si se cumple (si la condición es verdadera), si no se cumple la condición, se continúa con el flujo normal del programa

```
if (expresión_lógica) { // bloque de código a ejecutar
  }
```

Sentencia if-el: se evalúa la expresión lógica y si la condición es verdadera se ejecutan las instrucciones del bloque que se encuentra entre las primeras llaves, si la condición es falsa se ejecuta el bloque de código que está entre las llaves después de la palabra reservada 'else':

```
if (expresión_lógica) {
// bloque de código a ejecutar
// si la condición es verdadera
}
else {
// bloque de código a ejecutar
```

// si la condición es falsa

}

switch-case: evalúa la variable que se encuentra entre paréntesis después de la palabra reservada switch y la compara con los valores constantes que posee cada caso (case).

```
switch (opcion_a_evaluar){
 case valor1:
     /* Código a ejecutar*/
 break;
case valor2:
     /* Código a ejecutar*/
 break;
… case valorN:
      /* Código a ejecutar*/
 break;
 default:
      /* Código a ejecutar*/
 }
```

Estructuras de repetición

While: primero valida la expresión lógica y si ésta se cumple (es verdadera) procede a ejecutar el bloque de instrucciones de la estructura, el cual está delimitado por las llaves {}

```
while (expresión_lógica) {
// Bloque de código a repetir
// mientras que la expresión
// lógica sea verdadera.
}
```

Do-while: ejecuta el bloque de código que se encuentra dentro de las llaves y después valida la condición.

```
do {
```

/* Bloque de código que se ejecuta

por lo menos una vez y se repite

mientras la expresión lógica sea

verdadera. */

} while (expresión_lógica);

For: permite realizar repeticiones cuando se conoce el número de elementos que se quiere recorrer.

for (inicialización ; expresión_lógica ; operaciones por iteración) {

/*

Bloque de código a ejecutar

***/**

}

Define: permite definir constantes o literales; se les nombra también como constantes simbólicas

#define <nombre> <valor>

Arreglos:

Undimensional: La primera localidad del arreglo corresponde al índice 0 y la última corresponde al índice n-1, donde n es el tamaño del arreglo.

tipoDeDato nombre[tamaño]

Apuntadores:

Es una variable que contiene la dirección de una variable, es decir, hace referencia a la localidad de memoria de otra variable.

TipoDeDato *apuntador, variable;

apuntador = &variable;

Multidimencionales:

tipoDato nombre[tamaño][tamaño]…[tamaño];

Lectura y escritura de datos

Apuntador a archivo: es un hilo común que unifica el sistema de Entrada/Salida (E/S) con un buffer donde se transportan los datos.

FILE *F;

Abrir un archivo: La función fopen() abre una secuencia para que pueda ser utilizada y la asocia a un archivo.

***FILE fopen(char *nombre_archivo, char *modo);**

Cerrar Archivo: cierra una secuencia que fue abierta mediante una llamada a fopen().

int fclose(FILE *apArch);

Acordeón de Lenguaje Ml

Tipo de datos

Básico:

Tipos	Ejemplo
int	True, false
String	“Hola Mundo”
Char	#"a", #"\n", #"\065"
Int	0, ~1, 0xFF
word	0w25, 0wx0F
Real	1.2, 2E2, 1.1E~3

Operadores para los datos

Operador	Tipo	Descripción
~	int	menos unario
+, -, *, div, mod	int, word	aritmética de enteros
+, -, *, /	real	aritmética de punto flotante
=, <>, , <=, >=	int, word, real	operadores relacionales
not	bool	negación
andalso, orelse	bool	conjunción y disyunción de corto circuito
^	string	concatenación de cadenas

Lista de enteros:
$[1,2,3]$: int list

Lista de cadena de caracteres:
["Lupita", "Andre"]: string list

Lista de compuestas:
$[(1,2),(3,5),(5,3)]$: (int *int) list

Funciones

Para definir una función, usamos la clave **Fun**, ejemplo:

fun cuad x = x * x; val cuad = fn : int -> int

Nota: La función interfiere directamente por el uso de **“*”**

Cuando queremos hacer que la función sea real utilizamos:

fun cuadr x: real = x * x; val cuadr = fn : real -> real

La aplicación de funciones se escribe separando los argumentos con espacios a la derecha del nombre de la función

fun sum3 x y z = x + y + z; val sum3 = fn: int -> int -> int -> int

Concordancia de patrones; también podemos escribir expresiones λ, usamos la palabra fn en lugar de la letra griega y el operador => en lugar del punto:

val id = fn x => x; val id = fn : 'a -> 'a

Para funciones de varios argumentos podemos escribir la expresión en formato largo. Ejemplo:

(fn x => fn y => x+y) 3 4; val it = 7: int

Tuplas y listas

Las tuplas o n-adas de elementos utilizando paréntesis y separando los elementos con comas:

(1, "hola", true); val it = (1,"hola",true) : int * string * bool

El tipo int × string × bool, podemos usar tuplas para definir funciones "normales":

fun mul (x,y) = x * y; val mul = fn : int * int -> int

Cuando no utilizamos tuplas como argumentos de las funciones, las funciones están currificadas:

fun mulc x y = x * y; val mulc = fn : int -> int -> int

Para acceder directamente a los elementos de una tupla usamos las proyecciones. Para la proyección i-esima escribimos #i seguido de la tupla:

#2(true, 2.1, "hola"); val it = 2.1: real

Una lista se escribe entre corchetes y sus elementos separados por comas:

[1,2,3]; val it = [1,2,3] : int list

También podemos usar el operador :: para especificar una lista

1 :: 2 :: 3 :: nil; val it = [1,2,3] : int list

Nota: nil representa una lista vacía.

Utilizando la notación con :: podemos escribir fácilmente funciones recursivas sobre listas:

fun tam nil = 0 = | tam (x::xs) = 1 + tam xs; val tam = fn : 'a list -> int

también podemos escribir fácilmente las funciones car y cdr:

fun car (x::xs) = x;

fun cdr (x::xs) = xs;

Referencias:

ML (lenguaje de programación) - ML (programming language) - qaz.wiki. (s. f.). Recuperado 26 de febrero de 2021, de https://es.qaz.wiki/wiki/ML_(programming_language)

Alegsa, L. (2010, 12 diciembre). Definición de ML (lenguaje de programación). Recuperado 28 de febrero de 2021, de https://www.alegsa.com.ar/Dic/ml_lenguaje_programacion.php

Castellanos, A. P. (s. f.). ML (META LENGUAJE). Recuperado 28 de febrero de 2021, de https://prezi.com/4dh3sclvnusj/ml-meta-lenguaje/

Gongora, P. A. (s. f.). Laboratorio de Lenguajes de Programación. Recuperado 28 de febrero de 2021, de http://turing.iimas.unam.mx/~pedroG/Lab_ML.pdf

B. (2018, 5 agosto). Lenguaje C Cheat Sheet. Recuperado 26 de febrero de 2021, de https://cheatography.com/briana/cheat-sheets/lenguaje-c/

StuDocu. (s. f.). AcordeÃ3n - Resumen ProgramaciÃ3n Avanzada I Lenguaje Lenguaje Estructurado Diferencia de Comentario de Comentario Termina sentencia Plantilla del preprocesador. Recuperado 28 de febrero de 2021, de https://www.studocu.com/es-mx/document/universidad-autonoma-de-guadalajara/programacion-avanzada-i/resumenes/acordeon-resumen-programacion-avanzada-i/3023133/view