Programación Estructurada

Transformación del diseño al código

La estructura de un programa en el lenguaje de C es la siguiente:

<cabecera>

<declaraciones globales>

<definición de funciones>

Cabecera

La cabecera son las directivas de pre-procesador, esto significa que en el lenguaje de C ya existen muchos procesos programados que podemos utilizar indicando en qué librería se encuentra y otros que podemos definir nostros mismos.

Las librerías ya programadas en el lenguaje de C son archivos con extensión .h que contienen un conjunto de funciones las cuales están clasificadas de acuerdo a su área o naturaleza. Por lo que podemos decir que las funciones que se programaron para los procesos de entrada y salida de datos se encuentran en la librería stdio.h ; las funciones que preparan a la consola para la programación en un ambiente de texto se encuentra en las librería conio.h ; las funciones que se programaron para los procesos matemáticos se encuentran en la librería math.h ; entre otras.

Las directivas de pre-procesamiento más usada es: #include. Se utiliza para incluir los archivos .h o librerías que ya se encuentran programadas en el lenguaje de C. Otra directiva que se utiliza es , #define la cual sirve para definir macros y constantes.

A continuación se describe la sintaxis (forma general de escribir en el lenguaje) de estas dos directivas:

La sintaxis de la cabecera #include es la siguiente:

#include <identificador de la librería**>**

Ejemplo:

#include <stdio.h>

Nota: sólo para este caso será necesario poner los símbolos de menor y mayor que. Para las demás sintaxis, estos símbolos significarán el elemento que debe cambiar.

La sintaxis de la cabecera #define es la siguiente:

#define <identificador constante> <valor>

Ejemplo:

#define CIERTO 1

Declaraciones globales

Una declaración en el lenguaje de C sirve para indicarle al compilador que necesita guardar un espacio en memoria para ese elemento declararlo así como identificarlo para ser usado posteriormente. Todo lo que se declare en esta área será global, esto quiere decir que se podrá usar en cualquier función definida dentro del programa. El elemento más común usado en el área de declaraciones globales es la declaración de variables globales.

La sintaxis (forma general de escribir en el lenguaje) para una declaración de una variable global es la siguiente:

<tipoDato> <identificador variable>**;**

Ejemplo:

int numero;

Definición de funciones

Una función es un proceso en el que se programaran los pasos para resolver un problema en algún lenguaje de programación. En el lenguaje de C es necesario que exista por lo menos una función principal, la cual es la primera que se ejecuta dentro del programa y de ahí se podrán mandar llamar otras funciones definidas en el programa

La función principal tiene características especiales y sólo se define en el programa. Veamos la sintaxis de esta función.

Sintaxis definición función principal:

```
void  main ( )
{
        <sentencia_1>;
          …
        <sentencia_n>;
}
```

Otras funciones será necesario definirlas primero y después mandarlas llamar por lo que la sintaxis para la definición de la función es diferente que para la llamada de la función. En la unidad 3 se estudiará el tema de funciones así que aquí nos concentraremos sólo en la función principal.

Por otro lado, dentro del programa podemos escribir comentarios que no formen parte de las instrucciones del lenguaje de programación, para hacer esto posible es necesario utilizar los siguientes símbolos:

// El símbolo de dos diagonales servirá para poner un comentario de una sola línea.

/* Este símbolo de diagonal-asterisco al inicio del comentario y asterisco-diagonal al final del comentario servirá para cuando el comentario tenga varias líneas. */

El compilador no tomará en cuanta lo que este expresado como comentario y de esta manera se pueden poner anotaciones en nuestro programa.

Un ejemplo sencillo que imprime un mensaje en la pantalla usando la estructura del programa en el lenguaje de C es el siguiente:

```
/*ejemplo usando la función principal con el tipo de dato void
   programa realizado para el curso de programación estructurada
*/

//cabecera
#include <stdio.h>
#include <conio.h>

//no hay declaraciones globales

//definición de la función principal
void main( )
{
        printf(" Mi primer mensaje");
        getch( );
}
```

En este ejemplo usamos la librería stdio porque estamos haciendo uso de la función printf para imprimir en pantalla, esta función está definida dentro del archivo stdio.h. La funcion getch sirve para detener la ejecución del programa hasta que el usuario oprima una tecla y se encuentra en el archivo conio.h por lo tanto también es necesario incluir esta librería.

La función principal la definimos como void y el identificador de esta función siempre debe de ser main también las funciones siempre deben llevar paréntesis así que en este caso se abren y cierran paréntesis antes de iniciar la función.

El símbolo de la llave abierta significa que va a iniciar la función y el símbolo de la llave cerrada significa que terminó la función.

La función principal también se le puede definir con el tipo de dato entero, su sintaxis sería:

```
int  main ( )
{
        <sentencia_1>;
          …
        <sentencia_n>;
        return 0;
}
```

Esta forma de definir la función principal significa que dicha función regresará un valor entero. Recordemos que la función principal no la mandamos llamar nosotros como programadores sino que al ejecutarse el programa es lo primero que se manda llamar de forma automática por lo que es necesario indicarle que regrese el valor entero cero.

Veamos ahora el ejemplo anterior usando esta forma de definir a la función principal:

```
/*ejemplo usando la función principal con el tipo de dato int
   programa realizado para el curso de programación estructurada
*/

//cabecera
#include <stdio.h>

//no hay declaraciones globales

//definición de la función principal
int main( )
{
        printf(" Mi primer mensaje");
        return 0;
}
```

Como vemos en este ejemplo ya no es necesario llamar a la función getch por lo que tampoco será necesario incluir la librería conio.h

Estamos en el último paso para realizar un programa. La formalidad del pseudocódigo (Rodríguez,2003) permite transitar fácilmente al código fuente de la mayoría de los lenguajes de programación. En esta lectura, veremos cómo transformar el pseudocódigo a código en el lenguaje C.

La tabla que a continuación se presenta es una comparativa que describe cómo pasar de los principales elementos del pseudocódigo al código en el lenguaje C.

Pseudocódigo	Código en C	Observaciones	Ejemplo
INICIO <sentencia_1> … <sentencia_n> **FIN**	**int main( )** **{** <sentencia_1>**;** … <sentencia_n>**;** **return 0;** **}**	Es el inicio y fin del proceso principal que se ejecutará. La sentencia es el elemento más simple de la programación ya sean pasos, operaciones o cálculos que se realizan.	Pseudocódigo: **INICIO** total: entero *total ← total + 1* ***FIN*** (*incrementar total en 1)* Código: int main ( ) **{** *int total;* *total = total + 1;* *return 0;* ***}***
	;	Es el elemento que se utiliza en el código para indicar que una sentencia llegó a su fin. En el pseudocódigo no es necesario.	
IMPRIME	**printf (**“<mensaje>”**);** **ó** **printf(**“<formato>”, <variable>**);** Nota: se puede combinar ambas formas	En el pseudocódigo se puede poner alguna otra palabra que represente que se va a mostrar en pantalla un mensaje o valor de una variable	Pseudocódigo: **INICIO** total: entero *total ← total + 1* ***IMPRIME** “el valor es: , total* ***FIN*** Código: int main ( ) **{** *int total;* *total = total + 1;* *printf(“El valor es: %d, total);* *return 0;* ***}***

Autor
Prof. Sabrina Lizbeth Vega Maldonado

GUARDA	**scanf(**"<formato>"**,** <variable>**);**	En el pseudocódigo se puede poner alguna otra palabra que represente que se va ingresar un valor dentro de la variable	Pseudocódigo: **INICIO** numero: entero **IMPRIME** "ingresa un numero entero: " ***GUARDA*** *numero* ***FIN*** Código: int main ( ) { int numero; printf("ingresa un numero entero:); scanf("%d", numero); return 0; }
Pseudocódigo	**Código en C**	**Observaciones**	**Ejemplo**
SI <condición> **INICIA_SI** <sentencia_1> … <sentencia_n> **FIN_SI**	**if (**<condición>**)** **{** <sentencia_1>**;** … <sentencia_n>**;** **}**	Este elemento se le conoce como estructura de control selectiva simple en donde lo que está entre los símbolos de menor y mayor que es lo que cambia como se puede ver en el ejemplo	Pseudocódigo: **SI** valor > 10 **INICIA_SI** valor ← valor * 2 **FINALIZA_SI** Código: ***if (valor >10)*** ***{*** *valor = valor * 2;* **}**
SI <condición> **INICIA_SI** <sentencia_1> … <sentencia_n> **FIN_SI** **OTRA_FORMA** **INICIA_OTRA_FORMA** <sentencia_1> … <sentencia_n> **FIN OTRA_FORMA**	**if (**<condición>**)** **{** <sentencia_1>**;** … <sentencia_n>**;** **}** **else** **{** <sentencia_1>**;** … <sentencia_n>**;** **}**	Este elemento se le conoce como estructura de control selectiva doble en ella si se cumple la condición se realiza una o varias sentencias, de lo contrario cuando la condición sea falsa se realizan las sentencias que están en la sección OTRA_FORMA en el código se llama ELSE	Pseudocódigo: **SI** valor > 10 **INICIA_SI** valor ← valor + 2 **FINALIZA_SI** **OTRA_FORMA** **INICIA OTRA_FORMA** Valor ← valor - 2 **FIN OTRA_FORMA** Código: ***if (valor >10)*** ***{*** *valor = valor + 2;* **}** **else** **{** valor = valor - 2; **}**

SEGUN <variable> **HACER** **INICIA SEGUN** **Caso** 1 <sentencia_1> … <sentencia_n> **Caso** 2 <sentencia_1> … <sentencia_n> **Caso** n <sentencia_1> … <sentencia_n> **FIN SEGUN**	**switch (**<variable>**)** **{** **case** valor_1**:** <sentencia_1>**;** … <sentencia_n>**;** **break;** **case** valor_2**:** <sentencia_1>**;** … <sentencia_n>**;** **break;** **…** **case** valor_n**:** <sentencia_1>**;** … <sentencia_n>**;** **break;** **}**	Este elemento se le conoce como estructura de control selectiva múltiple. En ese elemento se utiliza un valor que se encuentra en una variable para que entre al caso necesario y se debe utilizar la instrucción break para que se salga e la estructura de control sin que tenga que pasar por todos los casos. Primero tenemos un ejemplo en pseudocódigo	Pseudocódigo: **IMPRIME** ingresa un numero del 1 al 3 **GUARDA** numero **SEGUN** numero **HACER** **INICIA_SEGUN** **Caso 1** **IMPRIME** ingreso el uno **Caso 2** **IMPRIME** ingreso el dos **Caso 3** **IMPRIME** ingreso el tres **FIN SEGUN**
Pseudocódigo	**Código en C**	**Observaciones**	**Ejemplo**
Continuación de **SEGÚN** <variable> **HACER**	**C**ontinuación de **switch**	Ahora podemos observar un ejemplo en código en el lenguaje de C	Código: ***printf (*** *“ingresa un numero del 1 al 3: “* ***);*** ***scanf(“%d”, &****numero****);*** ***switch(****numero****)*** ***{*** ***case*** *1:* ***printf(****“ingreso el uno”****);*** ***break;*** ***case*** *2:* ***printf(****“ingreso el dos”****);*** ***break;*** ***case*** *3:* ***printf(****“ingreso el tres”****);*** ***break;*** ***default****:* ***printf(****“fuera de rango”****);*** ***break;*** **}**

<inicializa contador> **MIENTRAS** <condicion> **INICIA_MIENTRAS** <sentencia_1> … <sentencia_n> <incremento / decremento contador> **FIN_MIENTRAS**	<inicializa contador> **while (**<condicion>**)** **{** <sentencia_1>**;** … <sentencia_n>**;** <incremento / decremento>; **}**	Este elemento se le conoce como estructura de control repetitiva "mientras". Es necesario declarar una variable que se el contador para controlar cuantas veces entrará a realizar las sentencias antes de la estructura, también dentro de ella es necesario incrementar o decrementar el contador. Se evalua la condición para entrar a ejecutar las sentencias	Pseudocódigo: contador ← 1 **MIENTRAS** contador <=10 **INICIA_MIENTRAS** IMPRIMIR contador contador ← contador+1 **FINALIZA_MIENTRAS** Código: Int contador = 1; ***mientras (*** *contador<=10* ***)*** ***{*** ***printf(****"%d", contador****);*** *contador = contador + 1;* **}**
<inicializa contador> **HACER** <condicion> **INICIA HACER-MIENTRAS** <sentencia_1> … <sentencia_n> <incremento / decremento contador> **FIN_HACER-MIENTRAS** **MIENTRAS** <condición>	<inicializa contador> **do (**<condicion>**)** **{** <sentencia_1>**;** … <sentencia_n>**;** <incremento / decremento>; **}while (**<condicion>**);**	Este elemento se le conoce como estructura de control repetitiva "hacer-mientras". Se sigue el mismo proceso que la anterior. La diferencia es que en este caso la primera vez se realizan las sentencias sin evaluar la condición	Pseudocódigo: contador ← 1 **HACER** **INICIA_HACER-MIENTRAS** IMPRIMIR contador contador ← contador+1 **FINALIZA_HACER-MIENTRAS** **MIENTRAS** contador<=10 Código: Int contador = 1; ***do*** ***{*** ***printf(****"%d", contador****);*** *contador = contador + 1;* ***}mientras (*** *contador<=10* ***) ;***
Pseudocódigo	**Código en C**	**Observaciones**	**Ejemplo**
PARA <inicializa> **HASTA** <condición> **INCREMENTO / DECREMENTO** <valor> **INICIA_PARA** <sentencia_1> … <sentencia_n> **FIN_PARA**	**for (**<inicializa>; <condicion>; <incremento / decremento>**)** **{** <sentencia_1>**;** … <sentencia_n>**;** **}**	Este elemento se le conoce como estructura de control repetitiva "para" en donde se puede simplificar las vistas con anterioridad programando el inicializar el contador, la condición y el incremento o decremento en una misma instrucción al inicio de la estructura	Pseudocódigo: **PARA** contador ← 1 **HASTA** contador <=10 **INCREMENTO** 1 **INICIA_PARA** IMPRIMIR contador **FINALIZA_PARA** Código: ***for (*** *int contador =1; contador<=10; contador++* ***)*** ***{*** ***printf(****"%d", contador****);*** **}**

Nota: en las estructuras de control cuando las sentencias que están dentro de ellas es de solamente una línea de código, es decir solo una sentencia entonces el lenguaje permite omitir las llaves de inicio y fin.

Referencia

Rodríguez, J. (2003) Introducción a la programación. Editorial Club Universitario. Obtenida el día 1 de agosto de http://books.google.com/books?id=nLMJsInMyBwC&pg=PA17&dq=pseudoc%C3%B3digo&hl=es&ei=9_xvTNaBNZSsQPUsN2gCw&sa=X&oi=book_result&ct=result&resnum=2&ved=0CC0Q6AEwAQ#v=onepage&q=pseudoc%C3%B3digo&f=false