MODULARIDAD

Abril, 2014
Guadalajara, Jalisco. México
Autor
Prof. Sabrina Lizbeth Vega Maldonado

Unidad 3

Las modularidad son procedimientos o subrutinas que se utilizan para facilitar la solución del problema.

En el lenguaje de C la modularidad se trabaja a través de funciones. La función principal es el main.

Hay dos aspectos importantes para tomar en cuenta acerca de la modularidad. Uno, permite que se pueda dividir el programa en segmentos lógicos funcionales haciendo que los programas complejos no crezcan desmesuradamente. Dos, permite que los segmentos de código internos así como las estructuras de control, puedan ser "abstraídas" permitiendo que la lógica de un algoritmo y los programas sean más fácil de comprender y más sencillos de mantener.

Primera Parte. Funciones

Las funciones son subrutinas que deben ser definidas para luego ser llamadas en otra función (generalmente es en la función principal llamada main).

Existen dos formas de trabajar con las funciones. La primera, se definen la(s) función(es) antes de la función principal main. La segunda es declarar un prototipo de la función, luego definir la función principal main y abajo de ésta definir las funciones.

Ejemplo de la estructura de la primera forma:

<cabeceras y/o directivas>

<declaración Variables Globales>

<definición de funciones>

<función principal main>

Ejemplo de la estructura de la segunda forma:

<cabeceras y/o directivas>

<declaración Variables Globales>

<declaración de prototipos de funciones>

<función principal main>

<definición de funciones>

La diferencia entre un prototipo de la función y su definición es que la definición ponerle un cuerpo, es decir, escribir las sentencias dentro de la función (encerradas entre las llaves) y el prototipo es sólo presentar a la función con el compilador y en lugar de abrir llaves se pone punto y coma (;) esto para poder llamarlas en otra función antes de definirlas con sus sentencias.

Las funciones en el lenguaje de C se dividen en dos categorías:

- Funciones que regresan valor
- Funciones que no regresan valo**r**

Funciones que regresan valor

- Sintaxis de la definición de la función que regresa valor:

<tipo Dato> <identificador> ([<parametros>])

{

sentencia_1;

…

sentencia_n;

}

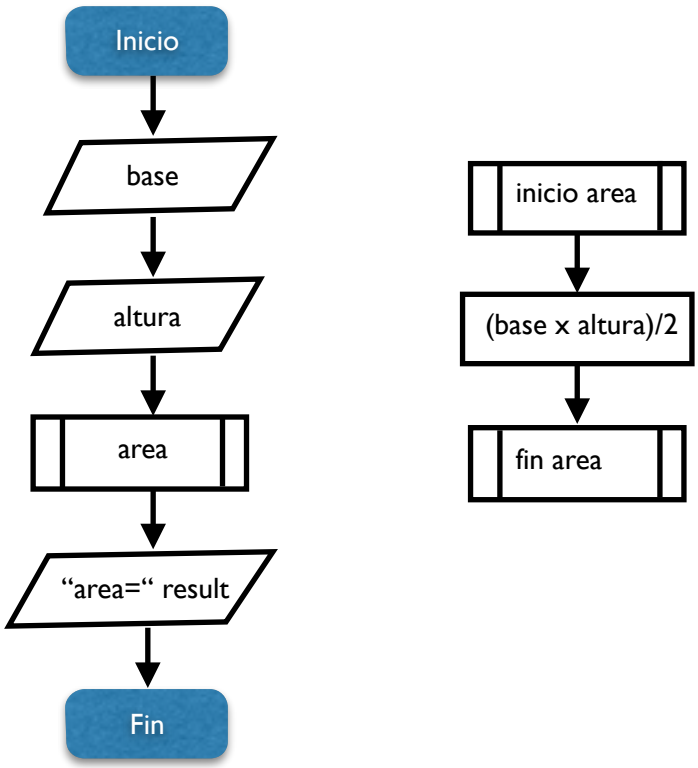
Recuerda que en la sintaxis, los elementos que se encuentran entre los símbolos menor y mayor que son los que van a cambiar en el código y los corchetes significan que es un elemento opcional.

- Sintaxis de la llamada de la función que regresa valor:

<variable> = <identificador> ([<argumentos>]);

Veamos un ejemplo que calcula el área de un triángulo cuya formula es base por altura entre dos. Los datos de entrada serán la base y la altura del triángulo, los procesos será el área e imprimir el resultado en pantalla, los datos de salida serán el resultado del área. Se ha decidido hacer el proceso del área en una función que regrese valor.

El diagrama de flujo se muestra a continuación:



También se presenta el pseudocódigo:

```
inicio area
        regresa (base x altura) / 2
fin area
inicio
        imprimir “Escribe la base del triangulo”
        guardar base
```

imprimir “Escribe la altura del triangulo”

guarda altura

result = area

imrpimir “Area = “ result

Fin

En el pseudocódigo no importa si se escribe primero el programa principal o el modulo de la(s) funcion(es).

Por último se presenta el código en el lenguaje C, primero, el ejemplo definiendo la función antes de la función main

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


//declaracion variables globales
float base, altura, result;
//definición de la función:
float area (  ) {
      return ( (base*altura)/2 );
}
//definición del main:
int main ( ) {
      printf (“Escribe la base del triangulo: “);
      scanf ( “%f”, &base);
      printf (“Escribe la altura del triangulo: “);
      scanf( “%f”, &altura);
      result = area( );
      printf ( “Area = %f”, result);
      return 0;
}
```

Segundo, el ejemplo declarando los prototipo de la función y definiendo ésta después de la función main

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

//declaración variables globales
float base, altura, result;

//declaración prototipo de la función
float area (  );

//definición del main:
int main ( ) {
      printf (“Escribe la base del triangulo: “);
      scanf ( “%f”, &base);
      printf (“Escribe la altura del triangulo: “);
      scanf( “%f”, &altura);
      result = area( );
      printf ( “Area = %f”, result);
      return 0;
}

//definición de la función:
float area (  ) {
      return ( (base*altura)/2 );
}
```

Funciones que no regresan valor

- Sintaxis de la definición de la función que no regresa valor:

void <identificador> ([<parámetros>])

{

sentencia_1;

…

sentencia_n;

}

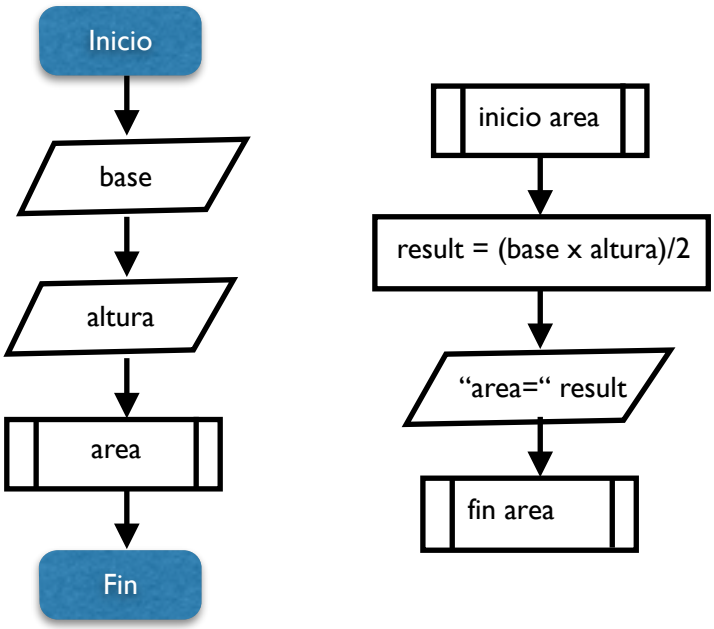
Recuerda que en la sintaxis, los elementos que se encuentran entre los símbolos menor y mayor que son los que van a cambiar en el código y los corchetes significan que es un elemento opcional.

- Sintaxis de la llamada de la función que regresa valor:

<identificador> ([<argumentos>]);

Veamos el mismo ejemplo, así que sólo tenemos que hacer los ajustes necesarios al diagrama de flujo y pseudocódigo para trabajar con una función que no regrese valor, esto quiere decir que la impresión en pantalla será desde la función.

Quedaría de la siguiente forma el diagrama de flujo:



El pseudocódigo se modificaría como sigue:

inicio area

result asigna (base x altura) / 2

imprimir (“Area = “ result)

fin area

inicio

imprimir “Escribe la base del triangulo”

guardar base

imprimir “Escribe la altura del triangulo”

guarda altura

area

Fin

El código en el lenguaje C usando la definición de la función área antes que la función principal queda:

```
#include <stdio.h>
float base, altura, result;


void area (  ) {
      result =  (base*altura)/2;
      printf ( “Area = %f”, result);
}
int main ( ) {
      printf (“Escribe la base del triangulo: “);
      scanf ( “%f”, &base);
      printf (“Escribe la altura del triangulo: “);
      scanf( “%f”, &altura);
      area( );
      return 0;
}
```

Un buen ejercicio sería que hicieras los cambios para escribir este mismo programa declarando primero el prototipo de la función área, después definiendo la función main y al final definir la función área.

Hasta ahora hemos visto como trabajar con funciones que regresan algún valor (entero, decimal, etc.) según los tipos de datos que conocemos, así como funciones que no regresan valor con la palabra reservada void para el lenguaje de C.

El criterio a utilizar para elegir una forma u otra será dependiendo dónde queremos imprimir el o los resultados. Si los resultados los queremos imprimir en la función main, las otras funciones deberán ser del tipo que regresa valor. Si los resultados los queremos imprimir dentro de cada función, las funciones tendrán que ser de tipo void.

Nos hace falta ver ese elemento en la sintaxis que está entre corchetes tanto los parámetros como los argumentos ¿Para que se usan y qué son cada uno?. Lo estudiaremos en la segunda parte de este documento.

Parte Dos. Parámetros

Los parámetros permiten establecer a una función valores que se encuentra fuera de ella y que quedan fuera de su alcance, es decir, cuando los valores o datos se declaran en variables locales (dentro de las funciones), esto quiere decir que no son globales.

Los parámetros se declaran dentro de los paréntesis de la función, si son varios se separan con una coma (,). ¿Cuántos parámetros es necesario declarar en una función? tantos como valores estén fuera de su alcance y necesite la función para realizar su proceso.

Los argumentos se utilizan en la llamada son los datos que se le van a pasar a los parámetros desde afuera.

En el lenguaje de C, existen dos tipos de parámetros: por valor y por referencia.

Parámetros por valor

El paso de parámetros por valor se evalúan de acuerdo al tipo de dato declarado en el parámetro y el tipo de dato del argumento de la llamada, si coincide se copia el valor del argumento al

parámetro sin importar el identificador tanto del parámetro como del argumento ya que no se ven entre sí. Sólo se hace una copia

Tomando el ejemplo del área del triángulo, el diagrama de flujo y pseudocódigo para una función que regresa valor no necesita modificaciones.

El código en C se presenta a continuación:

```
//cabecera
#include <stdio.h>
/*** no hay declaración de variables globales***/
//definición de la función con parámetros:
float area ( float base, float altura ) {
    return ( (base*altura)/2 );
}
//definición del main:
int main ( ) {
    float b, h, result; //declaración variables locales
    printf (“Escribe la base del triangulo: “);
    scanf ( “%f”, &b);
    printf (“Escribe la altura del triangulo: “);
    scanf( “%f”, &h);
    result = area( b, h );  //llamada con argumentos
    printf ( “Area = %f”, result);
    return 0;
}
```

Recuerda que las variables locales solamente las puede ver la función donde se declaran, por lo que en este ejemplo las variables “b”, “h” y “result” solamente puede verlas y trabajar con ellas la función main, fuera de esta función no son validas, ni si quiera las reconoce el compilador.

Es el mismo caso para los parámetros “base” y “altura”, solamente se ven en la función área.

Sabiendo esto podemos concluir que los parámetros y las variables locales que servirán para los argumentos se pueden identificar con

el mismo nombre porque no se ven entre ella. Sólo se asegura que el contenido de la primera variable local que está como argumento coincida el tipo de dato con el primer parámetro declarado en la función que se está mandado llamar para hacer la copia de la variable local (sin importar su identificador) al parámetro. Entonces si les ponemos el mismo identificador hay que tener cuidado de no confundir los parámetros con los argumentos, recuerda que los argumentos deben estar declarados como variables locales en la función donde se va a hacer la llamada.

A continuación se presenta el código en C para el diagrama de flujo y pseudocódigo que trabaja con una función que no regresa valor:

```
#include <stdio.h>

/**** no debe haber declaración de variables globales***/

void area ( float base, float altura ) {

      float result; //declaración variable local en la función area

      result =  (base*altura)/2; //esta operación usa los parámetros

      printf ( "Area = %f", result);

}

int main ( ) {

      float  base, altura; //declaración variables locales en la función main

      printf ("Escribe la base del triangulo: ");

      scanf ( "%f", &base);

      printf ("Escribe la altura del triangulo: ");

      scanf( "%f", &altura);

      area(base, altura ); //llamada de la función con argumentos

      return 0;

}
```

Parámetros por referencia

En el paso de parámetros por referencia se pasa a la función que tiene declarados los parámetros las direcciones de memoria de los argumentos que se encuentran en la llamada, en lugar de su valor.

A diferencia del paso por valor, aquí no se realizan copias de los datos o valores sino que se trabaja sobre la dirección de memoria de las variables locales (argumentos) que pasamos, lo que nos permite modificar el valor en cuestión, por lo que se trabaja con apuntadores para manipular la dirección de memoria.

Primero se presenta el mismo ejemplo para una función que regresa valor con parámetros con paso por referencia en el lenguaje de C:

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

/*** no hay declaración de variables globales***/

//definición de la función área con parámetros:
float area ( float *base, float *altura ) {
    return ( (*base * *altura)/2 );
}

//definición del main:
int main ( ) {
    float b, h, result; //declaración variables locales
    printf (“Escribe la base del triangulo: “);
    scanf ( “%f”, &b);
    printf (“Escribe la altura del triangulo: “);
    scanf( “%f”, &h);
    result = area( &b, &h );  //llamada con argumentos
    printf ( “Area = %f”, result);
    return 0;
}
```

Segundo se presenta el mismo ejemplo para una función que no regresa valor con parámetros con paso por referencia en el lenguaje de C:

```
#include <stdio.h>

/**** no debe haber declaración de variables globales***/

void area ( float *base, float *altura ) {

      float result; //declaración variable local en la función area

      result =  (*base * *altura)/2; //esta operación usa los parámetros

      printf ( “Area = %f”, result);

}

int main ( ) {

      float  base, altura; //declaración variables locales en la función main

      printf (“Escribe la base del triangulo: “);

      scanf ( “%f”, &base);

      printf (“Escribe la altura del triangulo: “);

      scanf( “%f”, &altura);

      area(&base, &altura ); //llamada de la función con argumentos

      return 0;

}
```

Para indicar que vamos a usar el apuntador de la dirección en memoria del parámetros se usa el símbolo del asterisco (*) el cual es el mismo para la multiplicación por lo que cuando se usa el parámetro “*base” y “*altura” se le debe anteponer este símbolo para este caso en medio de estos dos parámetros está el símbolo para la multiplicación.

En el caso de la llamada las variables locales que se vuelen argumentos se les debe anteponer el símbolo ampersan (&) para trabajar con la dirección de memoria de la variable local en lugar de una copia de los valores.

Bibliografía

Corona Nakamura, María Adriana. Ancona Valdez, Maria de los Angeles. (2011). Diseño de algoritmos y su codificación en lenguaje C. Editorial McGraw-Hill/Interamericana. ISBN 9786071505712

Deitel & Deitel. (2004). Como programar en C/C++ y Java. Editorial Pearson Educación. ISBN 9702605318

García-Bermejo Giner, José Rafael. (2008). Programación estructurada en C. Editorial Pearson Prentice Hall. ISBN 9788483224236