Fundamentos de la programación

NRC: 200274
Horario:

- Martes: 11:00 am - 12:55 pm, DUCT2 LC10
- Jueves: 11:00 am - 12:55 pm, DUCT2 LC08

Nombre: Ashley Lizbeth Barrera Hermosillo
Código: 220916338

Tema: Programación modular
Fecha: 25/10/2024

La programación modular es una técnica para el diseño de software que se enfoca en separar la funcionalidad del programa en módulos independientes e intercambiables, de tal forma que cada uno contenga todo lo necesario para ejecutar un aspecto específico de la funcionalidad deseada.

Los módulos han recibido los nombres de subrutinas, procedimientos, macros, métodos (en el caso de OOP), o, más comúnmente en lenguajes modernos, **funciones**. Por lo general, una función puede servir para uno de los siguientes propósitos:

- **Control del programa**. Son funciones que se usan para subdividir el control del programa, por lo que su implementación, el código que corren, es única al programa siendo escrito.
- **Tarea específica**. Son funciones diseñadas para lograr una tarea específica, por lo que están optimizadas para resolver ese problema y pueden ser usadas en cualquier programa. Por ejemplo, las funciones de las librerías estándar cumplen con este propósito.

Cada función tiene, al menos, tres características:

1. Un identificador.
2. Una lista de parámetros (que puede estar vacía).
3. Un tipo de valor de regreso.
   y generalmente se presentan de la forma:

```
tipo_regreso identificador(lista, de, parametros) {}

o
```

```
funcion identificador(lista, de, parametros): tipo_regreso {}
```

Para ejecutar una función, una función externa debe de llamarla mediante su identificador y proveerle las variables que satisfagan su lista de parámetros, acomodando espacio para el valor que regresará la función.

```
funcion_externa(){
        var = funcion_interna();
}
```

Por lo general, se espera que una función que no tenga un valor de regreso tenga efectos secundarios, es decir, que ejecute un cambio externo, cómo cambiar un archivo de texto, y que una función que si tenga valor de regreso no tenga efectos secundarios.

Dentro de la memoria, cada función ocupa un lugar en el *stack*, o pila. El programa comienza con la función principal (*main*), la que a su vez puede llamar a funciones internas, las cuales se van apilando encima de la anterior, solo siendo removidas cuándo su ejecución termina, cómo si fuera una pila de platos, dónde solo se puede quitar el plato que esté hasta arriba.

Por ejemplo, dado el siguiente pseudocódigo:

```
int b(){
        return 9
}

int a(int x){
        return x * b()
}

void main(){
        print(a(4))
}
```

la memoria luciría de esta manera:

Memoria	**Stack**				
0x08			b()		
0x04		a(4)	a(4)	a(4)	
0x00	main()	main()	main()	main()	main()

Una función puede llamarse a sí misma, lo que se llama recursividad. Esta es una alternativa a declarar bucles, por lo que, al igual que ellos, si no se condiciona correctamente corre el riesgo de formar un ciclo sin escape y desbordar la memoria, matando el programa.

Alcance de funciones y variables

Una función solo puede usar las variables y llamar a las funciones que ella conoce que existen, es decir, las que estén a su alcance.

El alcance de las variables y las funciones es establecido por el lugar en el que se definen:

- Una variable o función definida dentro de una función es **local** a esa función, y solo esta disponible para ella. Ella puede ser declarada en el cuerpo de la función o en su lista de parámetros.
- Por el contrario las variables y funciones declaradas fuera de una función son globales, y pueden ser accedidas por todas las funciones en el mismo archivo, la misma clase, o la misma librería, dependiendo del lenguaje. Una función o variable global puede ser importada a otro archivo, el procedimiento varía según el lenguaje.

```
int global = 10;

void funcion(int x){
        print(local_de_main) // No esta en su alcance
        print(x)             // Si lo esta
        print(global)        // Si lo esta
}

void main(){
        int local_de_main = 5

        funcion(8)

        print(global)        // Esta a su alcance
        print(local_de_main) // Si lo esta
        print(x)             // No lo esta
}
```

En lenguajes cómo C y C++, una función debe de ser declarada antes de la función que la va a utilizar, por lo que se pueden declarar **prototipos** de funciones, los cuáles solo contienen las tres características esenciales de una función (identificador, parámetros y tipo de regreso) sin la implementación.

```
float dividir(float a, float b){} // PROTOTIPO

void main(){
```

```
        print(dividir(9, 2))
}

float dividir(float a, float b){ // IMPLEMENTACION
        if(b == 0) return 0
        return a / b;
}
```

Fuentes

GeeksforGeeks. (11 de septiembre de 2024). *Function Call Stack in C*. GeeksforGeeks. https://www.geeksforgeeks.org/function-call-stack-in-c/

Libretexts. (7 de julio de 2020). *3.1: Modular Programming*. Engineering LibreTexts. https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_and_Computation_Fundamentals/Programming_Fundamentals_(Busbee_and_Braunschweig)/03%3A_Functions/3.01%3A_Modular_Programming?readerView