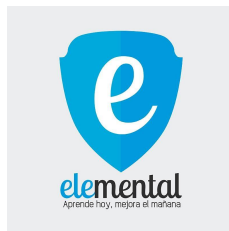


Funciones y Programación Modular

Rodolfo Christian Catunta Uturunco (Elemental Bolivia)



10 de Marzo de 2025

Contenido

- 1 Funciones
- 2 Paso de Parámetros
- 3 Recursividad

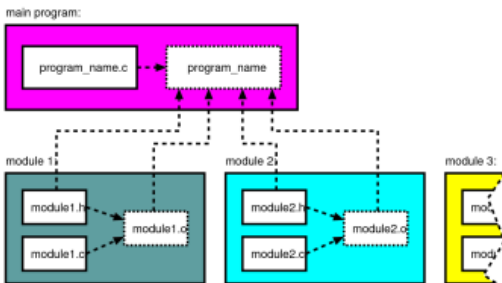
1 Funciones

2 Paso de Parámetros

3 Recursividad

¿Qué es Programación Modular?

- Es un modelo o paradigma de programación.
- Consiste en dividir un problema en subproblemas (más simples) y resolverlos por separado, para finalmente acabar resolviendo el problema.
- Este tipo de programación es la base para la creación de sistemas y software en el mundo real.



¿Qué es un módulo?

- Un modulo (o función) es un segmento de código separado del módulo principal de código, que puede ser invocado en cualquier momento desde el módulo principal o desde otro módulo.
- Los principales elementos de un módulo o función son los siguientes:
 - Entrada
 - Proceso
 - Salida



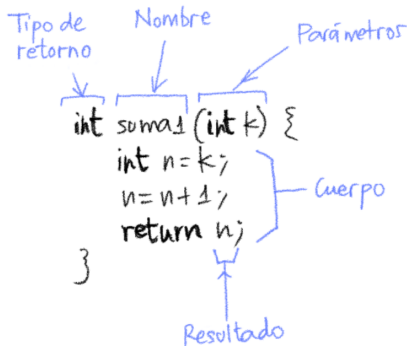
Algunas funciones de librerías

```
1 #include <iostream>
2 #include <cmath> // Funciones de matematica
3
4 using namespace std;
5
6 int main(){
7     int x=4,y=10;
8     int maximo=max(x,y); // maximo entre x e y
9     int minimo=min(x,y); // minimo entre x e y
10
11     double a=14.21,b=2.5;
12     double raiz=sqrt(a); // sqrt obtiene la raiz cuadrada
13     double potencia=pow(a,b); // pow eleva el numero a a la
14     potencia b
15     double valor_abs=abs(a); // obtiene el valor absoluto de un
16     numero
17     return 0;
18 }
```

Creación de una función

Para crear una función se debe tener en cuenta:

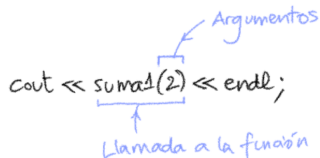
- El tipo de dato de retorno
- El nombre de la función (debe ser expresiva)
- Los parámetros de entrada y sus tipos
- El valor de retorno



Invocación de una función

Para invocar una función se debe tener en cuenta:

- El nombre de la función a invocar
- Los parámetros a enviar de entrada (argumentos)
- El uso de la salida de la función



The diagram shows the code `cout << suma1(2) << endl;` with two blue annotations. A bracket above the `2` is labeled "Argumentos". A bracket below the entire `suma1(2)` expression is labeled "Llamada a la función".

¿Por qué usar funciones?

Ventajas

- Ayudan a estructurar de mejor forma el código
- Ayuda a usar técnicas como la recursividad
- Apoya la reutilización de código
- Evita reescribir código

Desventajas

- Reduce un poco el tiempo de ejecución
- Es mas complicado realizar un calculo de la complejidad
- Puede causar RTE (Run Time Error)

Contenido

1 Funciones

2 Paso de Parámetros

3 Recursividad

Problema de Motivación

Problema

Realizar una función que reciba dos parámetros que son la hora y el minuto y aumente la hora en un minuto.

Ejemplo

aumentarHora(17,59) debe hacer que la hora ahora sea 18:00

Restricción

C++ solo puede devolver una variable o estructura de dato por función.

Aproximación 1: Variables Globales

```
1 #include <iostream>
2 using namespace std;
3
4 int h=17,m=59; // Variables globales
5
6 void aumentarHora(){// Funcion sin retorno, no se envian
    parametros
7     m=m+1;
8     if(m>59){
9         m=0;
10        h=h+1;
11    }
12    if(h>23) h=0;
13 }
14
15 int main(){
16     aumentarHora(); // Ahora h es igual a 18 y m es igual a 0
17     cout<<h<<" : "<<m<<endl;
18     return 0;
19 }
```

Existen dos tipos de envío de parámetros

- **Por Valor**

- Los parámetros recibidos son recibidos en variables nuevas.
- Si se cambia algo en la variable recibida en la función, no cambia para nada a la variable enviada al momento de la invocación.
- Es el tipo de envío mas común.

- **Por Referencia**

- Los parámetros recibidos son variables (direcciones de memoria)
- Si se cambia algo en la variable recibida en la función, también cambia lo mismo de la variable enviada al momento de la invocación.
- Es el tipo de envío es común cuando se desean tener mas de una variable de retorno.

Aproximación 2: Paso de Parámetros por referencia

```
1 #include <iostream>
2 using namespace std;
3
4 void aumentarHora(int &h, int &m){ // h y m son recibidos como
    variables
5     m=m+1;
6     if(m>59){
7         m=0;
8         h=h+1;
9     }
10    if(h>23) h=0;
11 }
12
13 int main(){
14     int h=23,m=59;
15     aumentarHora(h,m); // Ahora h es igual a 18 y m es igual a
        0
16     cout<<h<<" : "<<m<<endl;
17     return 0;
18 }
```

Contenido

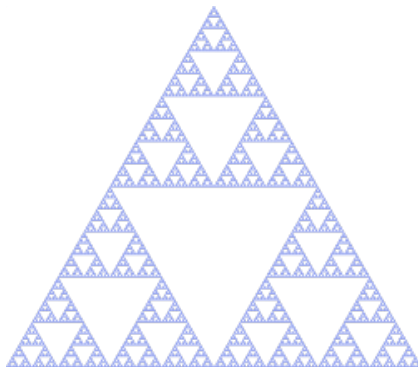
1 Funciones

2 Paso de Parámetros

3 Recursividad

Recursividad

- Es una técnica de programación en la que una función se llama a si misma una o varias veces.
- Cuando un función se llama a si misma, se dice que esa llamada es recursiva.
- En la practica se usa la recursividad para solucionar problemas que necesitan de problemas mas pequeños (casos base) para ser resueltos.



Ventajas, Desventajas y Limitantes

Ventajas

- Fácil de pensar y expresar en la mayoría de ocasiones.
- Son rápidos de programar.

Desventajas

- Ocupa mucha memoria.
- Si no se memoiza puede tender a abarcar mucho tiempo de ejecución.
- Hace complicado el calculo de la complejidad.

Limitantes

- Memoizacion.
- Pila de Funciones (Stack Overflow).

Ejemplo

```
1 int factorial(int n){
2     //Caso Base
3     if(n==0) return 1;
4     return n*factorial(n-1); // Paso Recursivo
5 }
6
7 int fibonacci(int n){
8     //Caso Base
9     if(n==1 or n==2) return 1;
10    return fibonacci[n-1]+fibonacci[n-2]; // Paso Recursivo
11 }
```