Funciones y Programación Modular

Rodolfo Christian Catunta Uturunco (Elemental Bolivia)

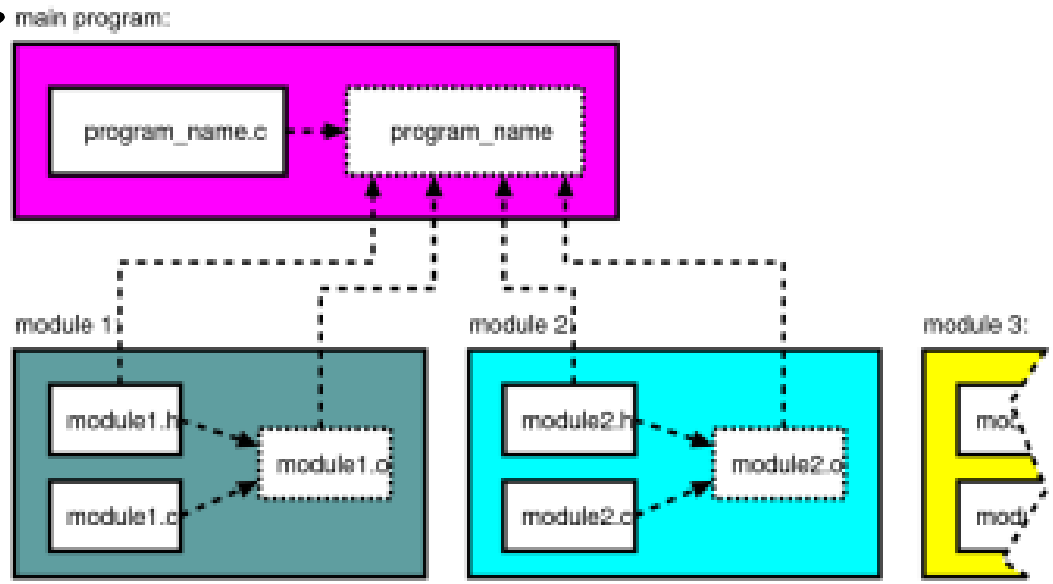
10 de Marzo de 2025

Contenido

Contenido

¿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

```cpp
#include <iostream>
#include <cmath> // Funciones de matematica

using namespace std;

int main(){
  int x=4,y=10;
  int maximo=max(x,y); // maximo entre x e y
  int minimo=min(x,y); // minimo entre x e y

  double a=14.21,b=2.5;
  double raiz=sqrt(a); // sqrt obtiene la raiz cuadrada
  double potencia=pow(a,b); // pow eleva el numero a a la
    potencia b
  double valor_abs=abs(a); // obtiene el valor absoluto de un
     numero
  return 0;
}
```

pow(a,b);
nombre parámetros

Imprimir maximo y minimo

square root

power = potencia

valor absoluto

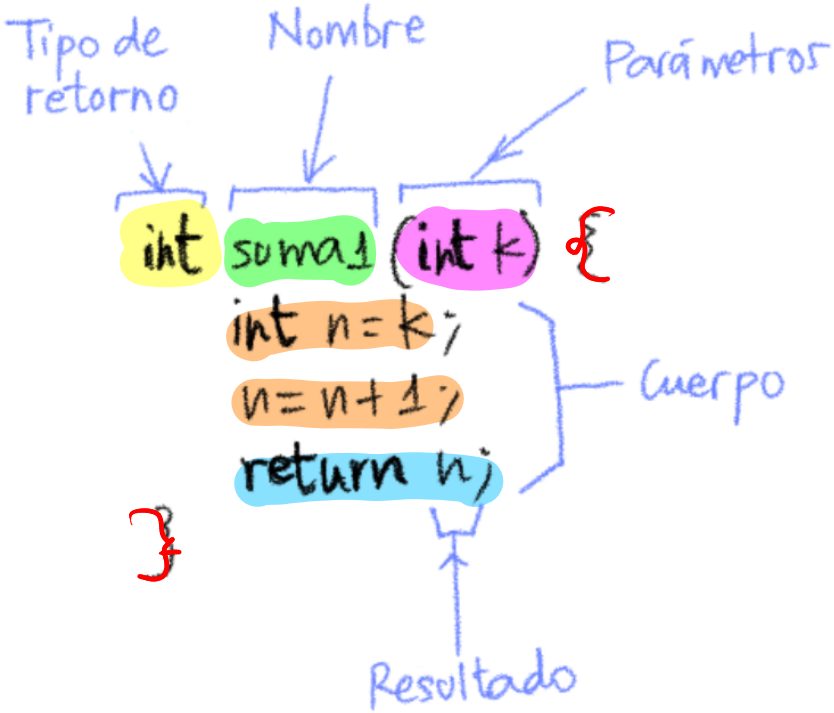
Imprimir raiz, potencia, valor_abs

No usar pow con valores grandes

Creación de una función

Para crear una función se debe tener en cuenta:

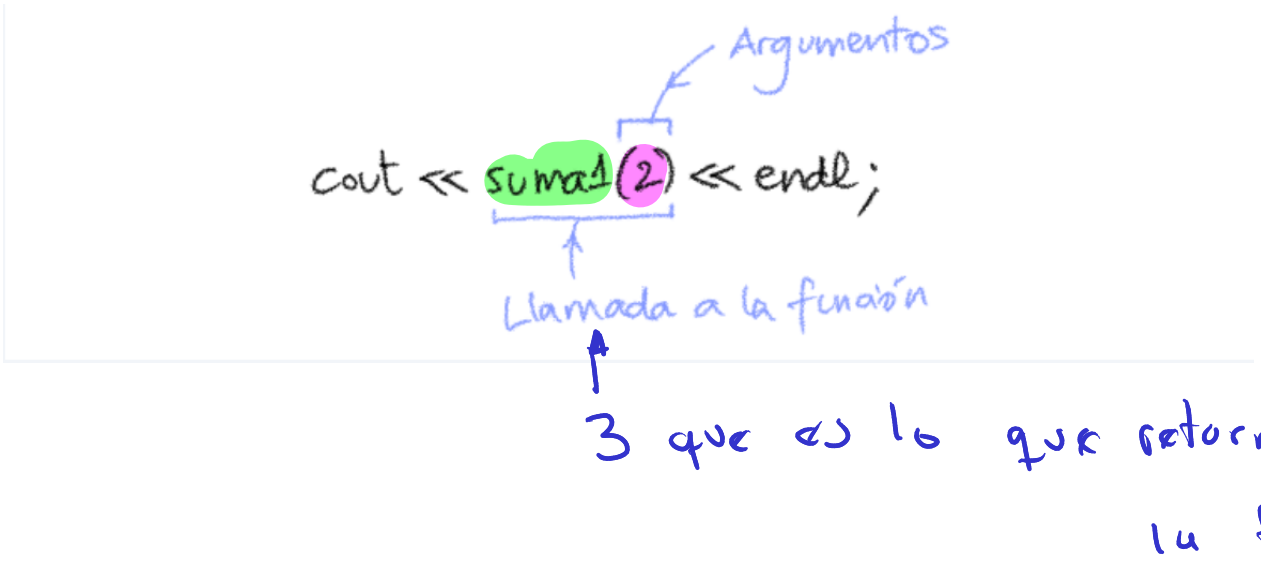
- El tipo de dato de retorno
- El nombre de la función (debe ser expresiva)
- Los parámetros de entrada y sus tipos
- El valor de retorno



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



¿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

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

```cpp
#include <iostream>
using namespace std;

int h=17,m=59; // Variables globales

void aumentarHora(){// Funcion sin retorno, no se envian parametros
  m=m+1;
  if(m>59){
    m=0;
    h=h+1;
  }
  if(h>23) h=0;
}

int main(){
  aumentarHora(); // Ahora h es igual a 18 y m es igual a 0
  cout<<h<<":"<<m<<endl;
  return 0;
}
```

Tipo de Retorno NULO, NADA, VACIO

También llamado proceso / procedimiento

Paso de Parámetros

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 envió 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 envió es común cuando se desean tener mas de una variable de retorno.

Aproximación 2: Paso de Parámetros por referencia

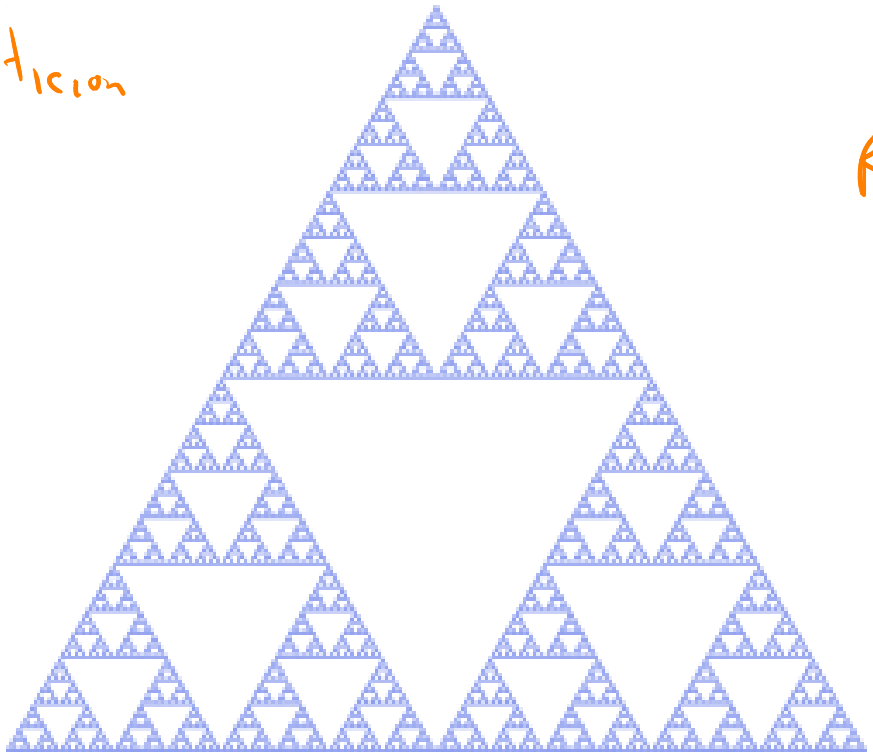
```cpp
#include <iostream>
using namespace std;

void aumentarHora(int &h, int &m){// h y m son recibidos como
     variables
  m=m+1;
   if(m>59){
     m=0;
     h=h+1;
   }
   if(h>23) h=0;
}

int main(){
   int h=23,m=59;
   aumentarHora(h,m); // Ahora h es igual a 18 y m es igual a
     0
   cout<<h<<":"<<m<<endl;
   return 0;
}
```

Contenido

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:

Recursion ≡ Repeticion



Recusion vs Iteración

DFS vs BFS

Pilas vs Colas

Ventajas, Desventajas y Limitantes

Ventajas

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

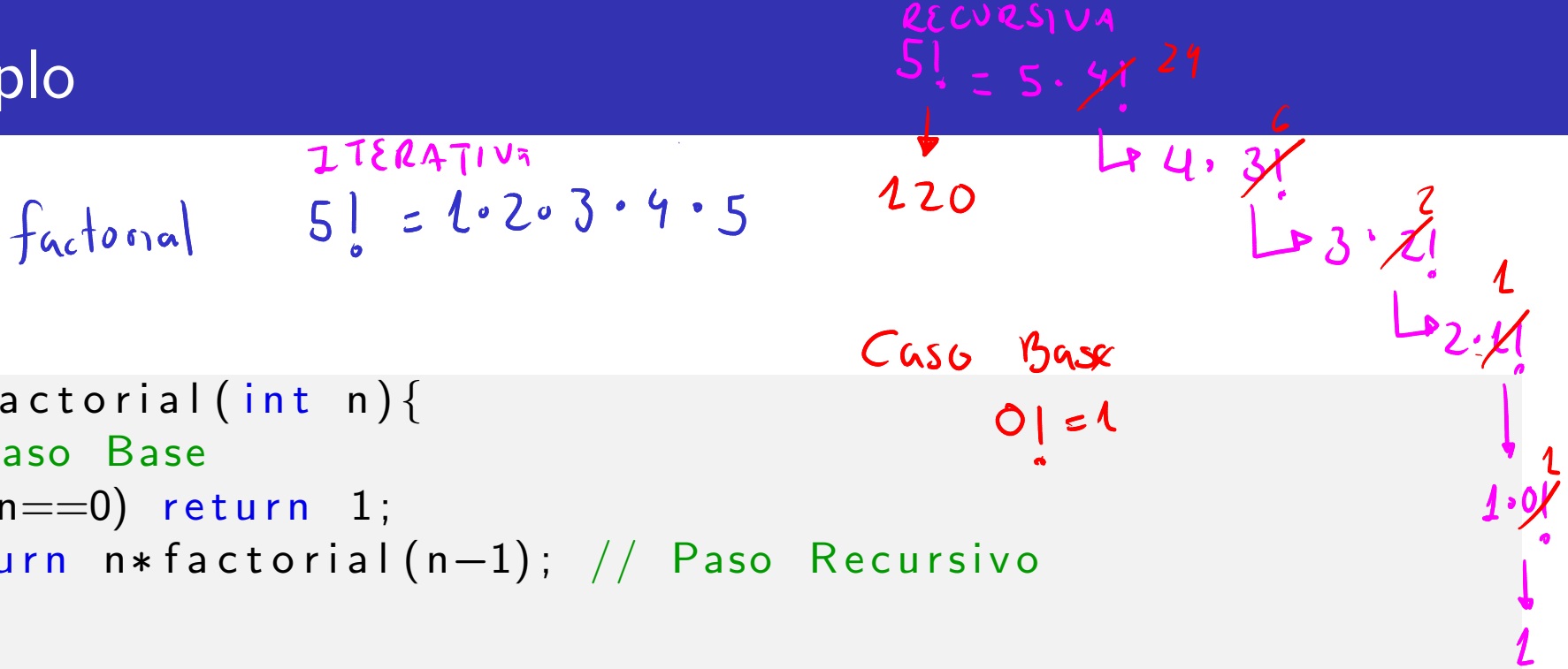
Desventajas

- 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




```java
int factorial(int n){
  //Caso Base
  if(n==0) return 1;
  return n*factorial(n-1); // Paso Recursivo
}

int fibonacci(int n){
  //Caso Base
  if(n==1 or n==2) return 1;
  return fibonacci[n-1]+fibonacci[n-2]; // Paso Recursivo
}
```