Estructura de Datos

Prof. Ing. Ramón Roque Hdz.,M.C.

Modularidad y Recursión

Modularidad

- Consiste en dividir un problema en varias subtareas de menor tamaño, haciéndolo mas facil de entender y modificar.
- Para implementar modularidad se utilizan **Subrutinas** y/o **Funciones**

Módulo

- Está constituido por varias instrucciones con un propósito lógico y agrupadas bajo un ***nombre***, el cual puede invocarse desde diferentes partes de un programa.
- Un módulo puede ser:
  - Una función
  - Una subrutina (procedimiento, o subprograma)

Tipos de módulos

- **SUBRUTINA (Subprograma o Procedimiento).-** Es un módulo que NO recibe ni entrega (regresa) parámetros.
- **FUNCION.-** Es un módulo que recibe y/o entrega (regresa) parámetros.

Parámetros

- Los valores que un módulo recibe o entrega (regresa) como resultado se llaman parámetros.
- Generalmente los parámetros son ***variables locales*** (de cualquier tipo: entero, flotante, char, booleano, etc.) que se envían de un módulo a otro para que éste último pueda utilizarlas también, aún sin estar declaradas dentro de él.

5

Variables globales y locales

- Las variables ***globales*** pueden utilizarse en todos los módulos.
- Las variables ***locales*** se declaran dentro de un módulo particular y están accesibles solo dentro del cuerpo de ese módulo.

6

Ejemplo: Problema con Modularidad usando Subrutinas [sin parámetros]

- Realizar un programa que pida dos numeros y calcule e imprima las siguientes operaciones matemáticas con ellos:
  - Suma
  - Resta
  - Multiplicacion

7

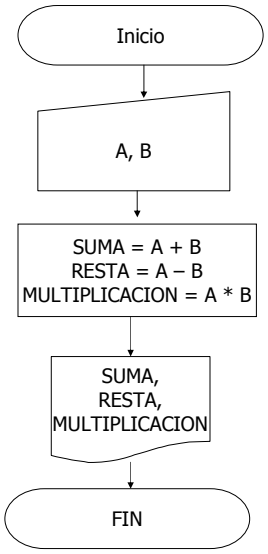


Diagrama de flujo
SIN modularidad

8

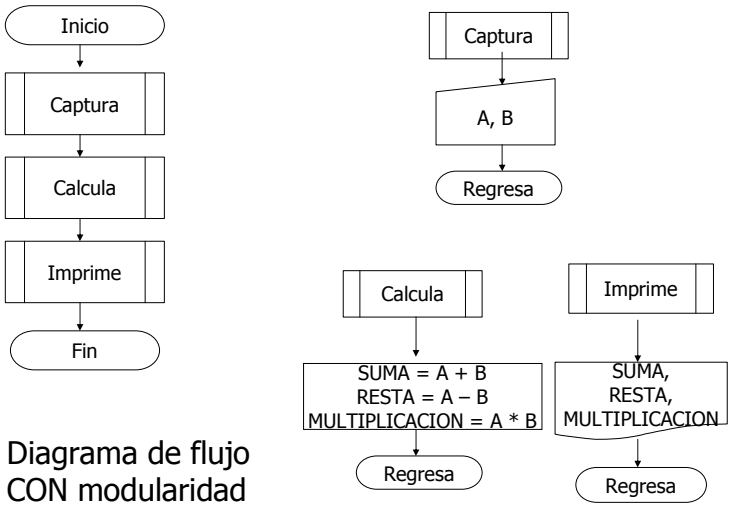


Diagrama de flujo
CON modularidad

9

Programa en C usando Subrutinas

```
#include <iostream.h>

void captura (void);
void calcula (void);
void imprime(void);

int a,b,suma,resta,multiplicacion;

void main (void)
{
  captura();
  calcula();
  imprime();
}
```

Declaracion de las subrutinas

Declaracion de las variables **GLOBALES**

Programa Principal

Llamada a las subrutinas

10

…Continuación….

```
void captura (void)
{
   cout << “Introduce el primer numero: “;
   cin >> a;
  cout << “Introduce el segundo numero: “;
   cin >> b;
}

void calcula (void)
{
  suma = a + b ;
  resta = a – b ;
  multiplicacion = a * b;
}

void imprime (void)
{
   cout << “La suma es: “ << suma;
   cout << “La resta es: “ << resta;
   cout << “La multiplicacion es: “ << multiplicacion;
}
```

Módulos con el código completo y explícito.

“void” significa que NO recibe y/o regresa parámetros.

11

Programa en C usando Funciones

```
#include <iostream.h>

int  sumar (int x, int y);
int  restar (int x, int y);
int  multiplicar (int x, int y);

void main (void)
{
    int a,b,suma, resta, multiplicacion;
    cout << "Introduce el primer numero: ";
    cin >> a;
    cout << "Introduce el segundo numero: ";
    cin >> b;

    suma = sumar(a,b);
    resta = restar(a,b);
    multiplicacion = multiplicar(a,b);

    cout << "La suma es: " << suma <<endl;
    cout << "La resta es: " << resta << endl;
    cout << "La multiplicacion es: " << multiplicacion << endl;
}
```

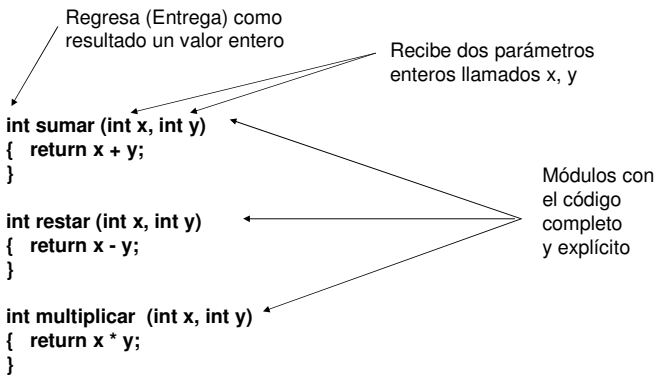
Declaración de las Funciones

Variables **LOCALES**

Se mandan llamar las funciones

12

…Continuación….



Ventajas de usar Modularidad

- Un programa modular es fácil de mantener, controlar, modificar, escribir y depurar.
- La división de un problema en varios módulos permite encargar los mas complejos a los programadores mas experimentados y los mas sencillos a los programadores principiantes.
- La modularidad permite reutilizar código.

Desventajas de usar Modularidad

- No hay algoritmos formales de modularidad.
- Cada programador adapta la modularidad según su lógica.
- La programación modular requiere mas memoria y tiempo de ejecución.

Recursión

- Ocurre cuando un programa (o módulo) se llama a si mismo.
- Cuando se usa recursión, SIEMPRE se debe establecer un “Estado Básico”, es decir, un momento en el cual la solución sea directa y no recursiva.
- La solución SIEMPRE debe acercarse al “Estado Básico”

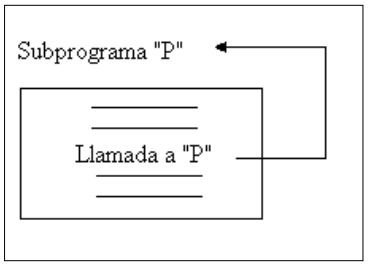
Todos los Programas recursivos…

- Poseen la solución recursiva y además el estado básico (Solución no-recursiva)
- Generalmente se usa una decisión para determinar a cual de ambas recurrir y evitar que el programa se ejecute indefinidamente.

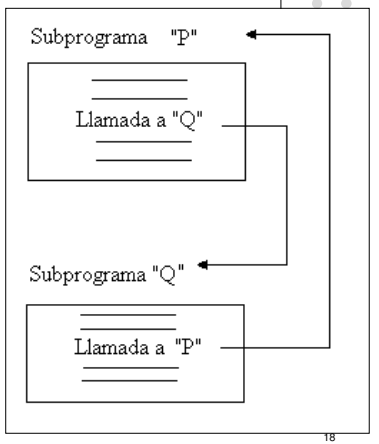
```
if (condicion)
{    …solucion recursiva…
}
else
{    …solucion no-recursiva…[Estado basico]
}
```

17

Tipos de Recursión



Recursión Directa



Recursión Indirecta

18

Ejemplo: Factorial de "N"

- Factorial de N está definido como:

*N! = N * (N-1) * …… * 1 para nums mayores de cero*

- El factorial de 0 por definición es 1

0! = 1
1! = 1 * 0! = 1 * 1 = 1
2! = 2 * 1! = 2 * 1 = 2
3! = 3 * 2! = 3 * 2 = 6
4! = 4 * 3! = 4 * 6 = 24
5! = 5 * 4! = 5 * 24 = 120

19

Definición recursiva del Factorial

Parámetro

```
Funcion Factorial (numero)
{
   Si numero > 0 entonces:
        Factorial = numero * Factorial(numero – 1)
   Si no:
        Factorial = 1
}
```

Solución Recursiva

Estado básico (Solución No Recursiva)

20

Programa Recursivo: Factorial en C

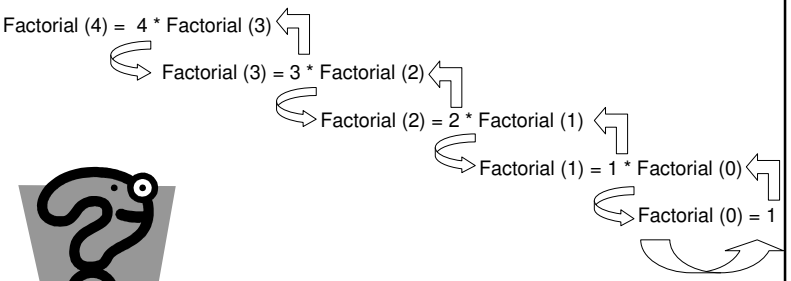
```
#include <iostream.h>

int factorial (int num);

void main (void)
{   int num, resultado;
    cout << "Numero del cual quieres el factorial: ";
     cin >> num;
     resultado =  factorial (num);
    cout << "El factorial es: " << resultado;
}

int factorial (int num)
{   if (num > 0)
    {               return num * factorial(num-1);
    }
    else
    {               return 1;
    }
}
```

Solución Recursiva

Estado básico

Entendiendo cómo se ejecuta el programa Recursivo del Factorial

- Supóngase que se desea obtener el factorial de 4:



Ventajas y desventajas de usar recursión

- Usando recursión se escriben programas mas compactos, y en ocasiones, mas comprensibles.
- La recursión utiliza más memoria en tiempo de ejecución.
- Si no se implementa correctamente, el programa produce errores.