PROGRAMACIÓN ESTRUCTURADA

07

Trabajo Práctico N° 7

Modularidad: Funciones

Apellido y Nombre: ..... Fecha: ...../...../.....

CONCEPTOS A TENER EN CUENTA

Como ya se ha visto en los trabajos prácticos previos, los problemas sencillos pueden resolverse fácilmente con programas simples de pocas instrucciones. Sin embargo, al tratar con problemas significativamente más complejos será necesario diseñar programas de cientos o miles de instrucciones. Consecuentemente, la detección y eliminación de errores se vuelve una tarea difícil de realizar.

A fin de simplificar la comprobación y mantenimiento de programas, la Programación Modular propone que el conjunto de instrucciones de un programa se divida en unidades independientes (módulos, subprogramas o subrutinas) que realicen tareas específicas. Así, cada una de estas unidades o módulos puede diseñarse, verificarse y mantenerse por separado. Los programas modulares cuentan con un módulo especial, llamado principal, que coordina el trabajo de los restantes subprogramas. Según su comportamiento, los módulos pueden clasificarse en: funciones y procedimientos. Una función, al igual que en matemáticas, comprende una serie de operaciones que deben aplicarse a valores (argumentos) usados por la función para calcular un único resultado simple.

El tema de procedimientos será tratado en el siguiente práctico.

SUBPROGRAMA: *conjunto de sentencias de un programa que realiza determinada tarea y que puede ser ejecutada desde uno o más puntos del programa principal con la simple mención de su nombre y posee todas las características propias de un programa.*

CONTROL DEL PROGRAMA: *Al efectuarse el llamado a un subprograma, el control del programa se transfiere a este módulo independiente el cual después de realizar la tarea encomendada retorna nuevamente el control al programa llamador o principal continuando este la ejecución normal del resto de sus instrucciones.*

PROGRAMA PRINCIPAL (*main*): *modulo que actúa como coordinador el cual controla y relaciona a todos los subprogramas.*

EJERCICIOS RESUELTOS

1. El siguiente programa calcula el cociente entero, mediante restas sucesivas, de 2 números ingresados por el usuario. Modifíquelo para obtener un programa modular, diseñando un módulo especial para el cálculo del cociente.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
main()
{ int num1,num2,cociente;
  cout << "Ingrese dividendo: ";
  cin >> num1;
  cout << "Ingrese un divisor: ";
  cin >> num2;
  cociente=0;
  while (num1>=num2)
    { num1=num1-num2;
      cociente++;
    }
  cout << "El cociente es: " << cociente;
  system("pause");
}
```

Instrucciones asociadas a la entrada de datos

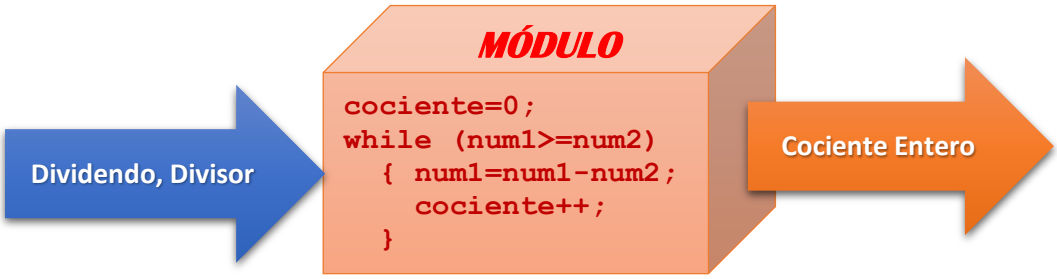
Instrucciones asociadas al cálculo del cociente

Instrucciones asociadas a la presentación de resultados

¿Qué tipo de módulo usaré? ¿qué datos necesitará este módulo para hacer el cálculo?

¿Cuáles serán las operaciones que realizará el módulo?

En primer lugar, es necesario identificar las partes del programa asociadas a la entrada, proceso y salida. En este caso, el proceso corresponde al cálculo del cociente mediante restas y será este conjunto de instrucciones el que se utilice para construir el módulo. Para ello, es preciso identificar qué datos serán requeridos por el módulo y cuál será el resultado a generar.



Los valores que deben suministrarse al módulo constituyen los parámetros o argumentos de éste. Mientras que, el resultado o valor de salida determinará el tipo de módulo que se diseñará. Cuando el resultado a generar por un módulo es un **valor simple** (entero, real, carácter o lógico) se diseñará una función. La función contendrá todas las operaciones de "cálculo" necesarias para obtener, a partir de los parámetros, el resultado final.

A continuación se presenta el código de un programa modular que cuenta con un módulo tipo función para el cálculo del cociente:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int division(int n1,int n2);

main()
{
  int num1,num2,resultado;
  cout << "Ingrese divisor: ";
  cin >> num1;
  cout << "Ingrese dividendo: ";
  cin >> num2;
  resultado=division(num1,num2);
  cout << "Resultado: " << resultado << endl;
  system("pause");
}

int division(int n1,int n2)
{
  int cociente=0;
  while(n1 >= n2)
   { n1=n1-n2;
     cociente++;
   }
  return cociente;
}
```

Prototipo de la función
Argumentos de la función
Tipo de dato de la función
Módulo Principal
Llamada o Invocación
Parámetros Actuales
Definición de la función
Parámetros Formales
Valor de retorno

Como puede observarse al escribir un programa modular se añaden algunos elementos a la estructura básica de programa:

- Prototipo del módulo (función o procedimiento): define qué módulos serán utilizados en el programa especificando su tipo, nombre y parámetros.
- Llamada o invocación del módulo (función o procedimiento): indica qué módulo será utilizado en ese momento y cuáles serán los valores con los que operará.
- Definición del módulo (función o procedimiento): indica qué datos serán utilizados por el módulo (parámetros formales), el conjunto de operaciones a ejecutar, y el/los resultados a generar. En el caso de las funciones se incluye la instrucción *return* para devolver el resultado del cálculo, ya que las funciones sólo pueden obtener un ÚNICO valor como resultado.

También puede observarse que la estructura del programa principal (*main*) se simplifica al reducir a una sola línea el cálculo del cociente. La "complejidad" de este cálculo queda confinada al módulo que implementa la operación.

1. Dado el siguiente programa, determina su propósito y modifícalo para realizar el cálculo mediante una función.

```
#include <iostream>
#include <stdlib.h>

using namespace std;

main()
{ int n,m;
  bool b;
     cout << "Ingrese dato: ";
     cin >> n;
     n=2*n-1;
     m=0;
     b=n>0;
     while (b==true)
     {   m=m+n%2*n;
           n=n-1;
           if (!(n>0))
                 b=false;
     }
     cout << "Salida: " << m << endl;
     system("pause");
}
```

2. El siguiente programa utiliza los módulos *VALIDAR* y *AREA_TRAPECIO* para verificar que los datos ingresados sean positivos y calcular el área de un trapecio isósceles, respectivamente. Tomando este programa como referencia, codifica uno que calcule el perímetro y área de un triángulo rectángulo conociendo el valor de 2 catetos (datos ingresados por el usuario).

```
#include <iostream>

using namespace std;

bool validar(float a,float b,float c);
float area_trapecio(float bme,float bma,float h);

main()
{ float bmenor,bmayor,altura;
  float area;
  do {
    cout << "Ingrese base menor: ";
    cin >> bmenor;
    cout << "Ingrese base mayor: ";
    cin >> bmayor;
    cout << "Ingrese altura: ";
    cin >> altura;
  } while (validar(bmenor,bmayor,altura)==false);
  area=area_trapecio(bmenor,bmayor,altura);
  cout << "Area: " << area << endl;
  system("pause");
}

bool validar(float a, float b, float c)
{
  if (a>0 && b>0 && c>0)
     return true;
  else
     return false;
}

float area_trapecio(float bme,float bma,float h)
{
   return (bme+bma)/2*h;
}
```

3. Codifica un programa que permita ingresar los lados de un triángulo, determinar su tipo, calcular su perímetro y su área. Para ello, diseñe los módulos *EXISTE, TIPO, CONTORNO y HERON*. El módulo *EXISTE* debe determinar si el valor de los lados permite la construcción del triángulo. El módulo *TIPO* debe determinar si el triángulo es equilátero (1), isósceles (2) o escaleno (3). El módulo *CONTORNO* debe calcular el perímetro del triángulo. Por último, el módulo *HERON* debe aplicar la fórmula de Herón para el cálculo del área del triángulo.
4. Codifica un módulo, llamado ***CARACTER***, que determine si un carácter corresponde a: minúscula (1), mayúscula (2), dígito (3) u otro símbolo (4).
5. Codifica un módulo, llamado ***NÚMERO***, que permita convertir un dígito (carácter '0', '1', …, '9') a su correspondiente valor numérico.
6. Codifica un módulo, llamado ***PRODUCTO***, que calcule el producto de 2 números enteros positivos mediante sumas sucesivas. Considera que el cálculo se realizará usando estructuras MIENTRAS y criterio de finalización por BANDERA. Además, asume que los valores a multiplicar son ingresados y **validados** en el programa principal. Indica el pasaje de parámetros utilizado.

   ¿Cómo se modificará este módulo para trabajar con valores positivos y negativos?
7. Codifica un módulo, llamado ***COCIENTE***, que calcule el cociente de la división entera (mediante restas sucesivas) de 2 valores enteros positivos. Considera que el cálculo se realizará usando estructuras REPETIR (criterio de finalización a elección). Además, asume que los valores a operar son ingresados y **validados** en el programa principal. Indica el pasaje de parámetros usado.

   ¿Cómo se modificará este módulo para trabajar con valores positivos y negativos?

   Tomando como base este módulo, codifica el módulo ***RESTO***.
8. Codifica la función ***POTENCIA***, calculada mediante productos sucesivos, que utilice el módulo *PRODUCTO* (ejercicio 6) para resolver el producto acumulado. Considera que la base y el exponente son ingresados y **validados** en el programa principal. Indica el pasaje de parámetros utilizado.

   ¿Cómo se modificará este módulo para trabajar con valores positivos y negativos?
9. Codifica un módulo, llamado *FACTORIAL*, que implemente el cálculo utilizando el módulo *PRODUCTO* (ejercicio 6) para resolver el producto acumulado correspondiente. Asume que el valor a calcular es ingresado y **validado** en el programa principal. Indica el pasaje de parámetros utilizado.
10. Codifica un módulo, llamado ***CUADRADO***, que calcule el cuadrado (TP5, ejercicio 4.a) de un número entero *N* (positivo, negativo o cero). Considera que el valor para el cálculo es ingresado en el programa principal. Indica el pasaje de parámetros utilizado.

    Tomando como base este módulo, codifica el módulo ***CUBO***.
11. Codifica un módulo, llamado ***CONTAR_DIGITOS***, que determine cuántas cifras componen un número entero ingresado por el usuario. Implementa una versión utilizando los operadores del lenguaje (/, %) y otra usando los módulos *COCIENTE* y *RESTO* (ejercicio 7). Asume que el valor a analizar es ingresado en el programa principal. Indica el pasaje de parámetros utilizado.
12. Codifica un módulo, llamado ***INVERSO***, que calcule el inverso de un número ingresado por el usuario. Implementa una versión utilizando los operadores del lenguaje (*, /, %) y otra con los módulos *PRODUCTO* (ejercicio 6), *COCIENTE* y *RESTO* (ejercicio 7). Asume que el valor para el cálculo es ingresado en el programa principal. Indica el pasaje de parámetros utilizado.
13. Codifica un módulo, llamado ***CAPICUA***, que determine si un número entero positivo *N* es capicúa o no. Para ello, ten en cuenta que puede reusar el módulo *INVERSO*. Asume que el valor a analizar es ingresado y validado en el programa principal. Indica el pasaje de parámetros utilizado.
14. Codifica un módulo, llamado ***PRIMO***, que determine si un valor positivo ingresado por el usuario es primo o no. Considera que el valor de entrada es ingresado y **validado** en el programa principal. Indica el pasaje de parámetros utilizado.

    Implementa una segunda versión utilizando los módulos *COCIENTE* y *RESTO* (ejercicio 7).

15. Considerando la serie de Fibonacci
    a) codifica un módulo, llamado ***FIBO***, que calcule un término de la serie
    b) codifica un módulo, llamado ***ISFIBO***, que dado un número entero positivo *N*, determine si se trata o no de un valor perteneciente a la serie.
16. Considerando la calculadora desarrollada en el TP6 (ejercicio 11) y las operaciones: suma (+), resta (-), producto (*), división real (/), división entera (d), resto (m), potencia (^), raíz cuadrada (~), factorial (!); codifica un programa modular que realice los cálculos de producto, división entera, resto, potencia, raíz cuadrada y factorial mediante módulos.