CURSOS INTERSEMESTRALES

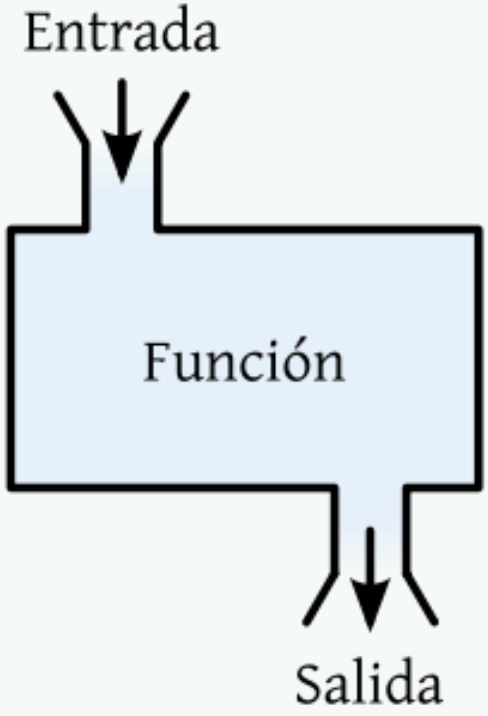
PROTECO

C++ básico

Funciones

Introducción

Una función es un bloque de código que realiza una tarea en específico y que puede retornar un valor.



PROTECO

Introducción

Las funciones son utilizadas para descomponer un problema en problemas más simples u operaciones que se realizan con mucha frecuencia.

PROTECO

Sintáxis

Prototipo: Indica la forma de la función

```
tipo_r nombreFuncion(tipo_d parametros);
```

Definición: Indica cómo se realiza la función

```
tipo_r nombreFuncion(tipo_d parametros) {
      //código de la función
      return;
}
```

PROTECO

Sintáxis

Llamada: Se coloca en el punto donde queremos que la función se ejecute.

```
nombreFuncion(parametros);
variable = nombreFuncion(parametros);
```

PROTECO

Sintáxis

Donde:

tipo_r: es el tipo de dato que retorna la función

nombreFuncion: el nombre de la función

tipo_d: es el tipo de dato de los parámetros

parametros: parámetros que recibe la función

La palabra reservada **return** termina la ejecución de la funcion, e indica el valor que regresará, esto es, su resultado.

PROTECO

Variables locales y globales

Variables globales: Son aquellas que son accesibles en cualquier parte del programa.

Variables locales: Son aquellas que son accesibles en la función donde están definidas o declaradas.

Se recomienda hacer poco uso de las variables globales, ya que sus variaciones pueden provocar comportamiento inesperado durante la ejecución del programa.

PROTECO

Variables locales y globales

```
//Variables.cpp
//Código para ejemplificar variables locales y
//globales

#include <iostream>

int var_global = 0;

int funcion(int a) {
     int var_local = 1;
     return 0;
}
```

Cabe mencionar que “a” es una variable local de “funcion”.

PROTECO

Paso de parámetros

Paso por valor: Se crea una copia temporal del parámetro al momento de entrar en la función. Todos los tipos de dato primitivos (int, float, double, char, etc.) se pasan de esta manera.

Paso por referencia: Se modifica directamente los valores de los parámetros. Los arreglos, ya sea unidimensionales o bidimensionales se pasan así.

PROTECO

Ejemplos (paso por valor)

Función sin parámetros y sin valor de retorno

```
//Prototipo
void imprimeMensaje();

//Definición
void imprimeMensaje() {
      cout << “Hola mundo” << endl;
}

//Llamada
imprimeMensaje();
```

PROTECO

Ejemplos (paso por valor)

Función con parámetros sin valor de retorno

```
//Prototipo
void imprimeSuma(int a,int b)

//Definición
void imprimeSuma(int a,int b) {
      cout << a + b << endl;
}

//Llamada
imprimeSuma(a,b);
```

PROTECO

Ejemplos (paso por valor)

Función con parámetros y valor de retorno

```
//Prototipo
int suma(int a, int b);

//Definición
int suma(int a, int b) {
      return a + b;
}

//Llamada
variable = suma(a,b);
```

PROTECO

Ejemplos (paso por referencia)

Función con arreglos

```
void llenarArreglo(int arreglo[],int tamano) {
     //Código de la función
}
```

Función con matrices

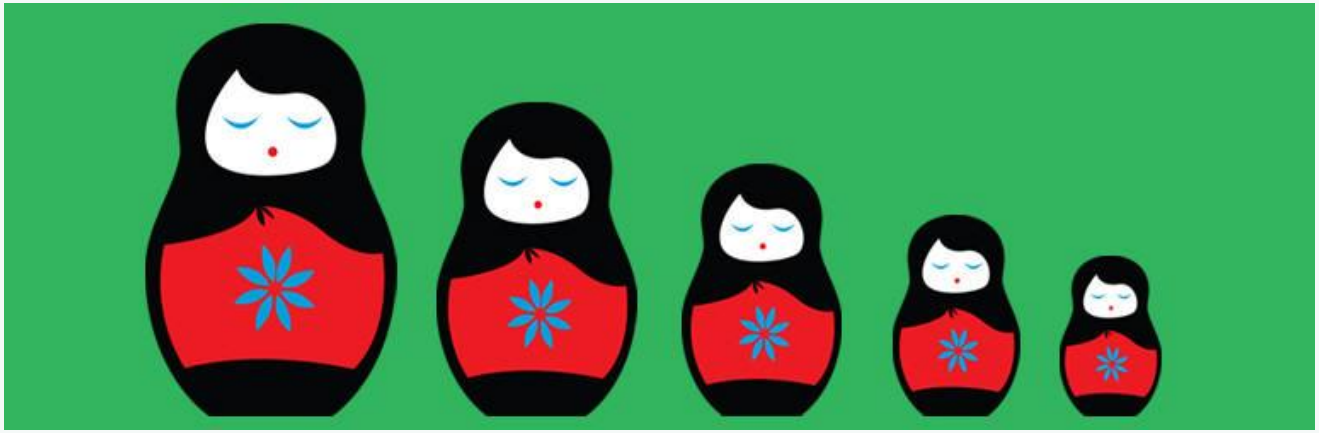
```
void leerMatriz(int matriz[][col],int ren) {
     //Código de la función
}
```

PROTECO

Recursividad

La recursividad consiste en definir una función en términos de si misma.

En programación, consiste en la técnica de aprovechar la capacidad de las funciones de llamarse a sí mismas.



PROTECO

Recursividad

Un ejemplo clásico es el cálculo del factorial de un número, definido de la siguiente manera:

$$f(n) = \begin{cases} 1, & si\ n = 0 \\ n * f(n-1), & si\ n > 0 \end{cases}$$

Si queremos calcular f(3), se haría lo siguiente:

$$f(3) = 3 * f(2)$$
$$f(3) = 3 * 2 * \mathrm{f}(1)$$
$$f(3) = 3 * 2 * 1 * \mathrm{f}(0)$$
$$f(3) = 3 * 2 * 1 * 1$$
$$f(3) = 3 * 2 * 1$$
$$f(3) = 3 * 2$$
$$f(3) = 6$$

PROTECO

Recursividad

En código resultaría de la siguiente forma.

```
int factorial(int n) {
      if (n == 0) {
            return 1;
      }
      else {
            return n * factorial(n-1);
      }
}
```

PROTECO

Funciones inline

Las llamadas a funciones tienden a reducir el rendimiento del programa, debido a los saltos que se realizan en la memoria.

```
funcion1();
```

```
funcion1() {
    …
    return;
}
```

```
funcion2();
```

```
funcion2() {
    …
    return;
}
```

PROTECO

Funciones inline

Las funciones inline combaten este problema, ya que durante la compilación, el código de estas funciones se copia en los lugares donde fue llamada, evitando así los saltos.

Debido a esta operación, el programa resultante crece en tamaño, y se recomienda que las funciones inline sean funciones pequeñas o excesivamente utilizadas.

PROTECO

Funciones inline

Ejemplos

```
inline int suma(int a,int b) {
      cout << a + b << endl;
}

inline void cuenta(int a) {
      for(int i = 0; i < a; i++) {
            cout << i << endl;
      }
}
```

PROTECO

Bibliotecas

Conforme nuestro código va creciendo, se vuelve más tedioso dar mantenimiento al código.

Las bibliotecas nos permiten separar el código y tenerlo en diferentes archivos, haciendo más sencillo el darle mantenimiento.

PROTECO

Bibliotecas

Para crear una biblioteca de funciones, se requiere de lo siguiente:

- Un archivo .cpp que contendrá las definiciones de las funciones a incluir.
- Un archivo .hpp que contendrá los prototipos de las funciones a incluir.

Para incluirlos en nuestro código que contiene la función principal, se utiliza la directiva de preprocesador:

```
#include “mibiblioteca.h”
```

PROTECO