**+Instituto Superior de Formación Técnica Nº 151 logo151-trans.gif  
Carrera: Analista de Sistemas  
1 Año. Algoritmos y Estructuras de Datos I**

|  |  |
| --- | --- |
| **Trabajo Práctico Nº 2.5** | **Unidad 2** |
| **Modalidad:** Semi-Presencial | **Estratégica Didáctica:** Trabajo Grupal. |
| **Metodología de Desarrollo:** acordar | **Metodología de Corrección:** acordar docente |
| **Carácter de Trabajo:** Obligatorio – con Nota | **Fecha Entrega:** A confirmar por el Docente. |

**Unidad 2 – Instrucciones Tipos Operadores   
Funciones**

**Marco Teórico:**

1. ¿Qué entiende Por función? ¿Qué características tiene?
2. ¿Qué entiende Por procedimiento? ¿Qué características tiene?
3. ¿En que se beneficia el programador al utilizar funciones?
4. ¿Cuantas “tareas o servicios” debe realizar una función?
5. Desarrollar los conceptos de: *tipo nombre(parámetros),* dar ejemplos
6. Explicar el alcance de las variables definidas en dentro del cuerpo de la Función
7. ¿Qué se entiende por argumentos?
8. ¿Qué función cumple el Return en la Función? ¿Puede ser de cualquier tipo?
9. ¿Qué se entiende por “¿Prototipo de las Funciones”, donde se ubica en el Código?
10. ¿En qué parte del Código escribimos las funciones?
11. Los parámetros son “valores ” que pasamos a la función, pero que pasa cuando la variable tiene el mismo nombre en el código y en la función con su valor?
12. OPCIONAL: trate de Investigar el “tema de pasaje de parámetros” x “valor y referencia”, sin entrar en temas como “punteros, tratado más adelante”, conceptualice y fundamente.

**Marco Práctico:**  
Desarrollar en C++ los siguientes programas:

1. Escriba una función llamada mult() que acepte dos números en punto flotante como parámetros, multiplique estos dos números y despliegue el resultado.
2. Escriba una función llamada al\_cuadrado() que calcule el cuadrado del valor que se le transmite y despliegue el resultado. La función deberá ser capaz de elevar al cuadrado números flotantes.
3. Escriba una función nombrada funpot() que eleve un número entero que se le transmita a una potencia en número entero positivo y despliegue el resultado. El número entero positivo deberá ser el segundo valor transmitido a la función.
4. Escriba un programa en C++ que devuelva la parte fraccionaria de cualquier número introducido por el usuario. Por ejemplo, si se introduce el número 256.879, debería desplegarse el número 0.879.
5. Escriba una plantilla de función llamada despliegue() que despliegue el valor del argumento único que se le transmite cuando es invocada la función.
6. Escribir un función que Intercambie el valor de 2 variables.
7. Escriba una función nombrada cambio() que tenga un parámetro en número entero en pesos y devuelva le valor en dólares.
8. Escriba una función nombrada tiempo() que tenga un parámetro en número entero llamado totalSeg y tres parámetros enteros nombrados horas, min y seg. La función es convertir el número de segundos transmitido en un número equivalente de horas, minutos y segundos.
9. Escribir un Programa en C++ que duplique el valor dos números con una función Duplicar(int a, int b), pero a y b están definidas en el código…   
   Mostrar los valores de a y b, antes y después del llamado de la Función.  
   {Nota: EL presente ejercicio es para “Inducir” al alcance de las Variables y el pasaje de Parámetros, dado que como vimos “hasta ahora” solo le pasamos un valor y no la variable en si. Comprenda y Conceptualice el Ejercicio, escriba debajo del Código un Texto y relaciónelo con la pregunta 11 (opcional 12).

Lic. Oemig José Luis.

1. En los primeros prácticos hablamos de las cuatro estructuras básicas para expresar acciones en un lenguaje de programación: secuencia, selección, ciclo y subprograma. Se dice que los subprogramas permiten escribir por separado partes del programa y luego ensamblarlas en una forma final. En C++, los subprogramas se denominan *funciones.*

Las funciones son un conjunto de instrucciones encapsuladas en un bloque. Usualmente reciben valores llamados parámetros, los cuales se utilizan para efectuar operaciones y adicionalmente retornan un valor.

1. Los procedimientos son básicamente lo un conjunto de instrucciones que se ejecutan sin retornar ningún valor. En el contexto de C++ un procedimiento es básicamente una función que no retorna ningún valor. (void)
2. El principal beneficio de escribir funciones radica en la reutilización del código y la modularidad del mismo.
3. Es deseable que cada función resuelva *una* tarea. De ese modo se gana en reutilización.
4. Desarrollar los conceptos de: *tipo nombre(parámetros),* dar ejemplos

*tipo nombre(parámetros)*  es la forma en que se declara una función.

* *tipo*: se refiere al tipo de retorno de la función. El valor de retorno es devuelto mendiante la instrucción *return*.
* *nombre*: es el nombre de la función. Para los nombres rigen las mismas reglas que para nombrar variables.
* *parámetros*: son variables que se utilizan para proporcionarle datos a la función.

(se denominan también *parámetros formales*)

1. Explicar el alcance de las variables definidas en dentro del cuerpo de la Función.

Las variables declaradas dentro del cuerpo de la función son de carácter local: esto es, no se conocen fuera de ésta.

1. La llamada a una función se realiza del siguiente modo:

nombre(argumentos)

Donde, como siempre, *nombre* se refiere al nombre de la función y los ***argumentos*** son valores particulares que se reemplazan en el lugar de los parámetros. Se denominan también *parámetros actuales*.

1. La función de la orden *return* es la de retornar el resultado. Dicho resultado debe tener el mismo tipo que antecede al nombre de la declaración de la función.
2. El Prototipo de una Función no es más que la cabecera, terminada en “;”.

Estos prototipos se deben colocar antes de la función principal *main*. Estas declaraciones son necesarias debido a la regla de C++ requerida para declarar un identificador antes de poder usarlo.

1. Una vez escritos los prototipos, se escribe la función main y, a contiuación de ésta, se escribe la implementación de las funciones previamente declaradas.
2. Los parámetros son “valores ” que pasamos a la función, pero que pasa cuando la variable tiene el mismo nombre en el código y en la función con su valor?

Las variables que forman parte del argumento de la función son locales a ésta, es decir: se utilizan dentro de la función y “no se ven” desde afuera (a menos que sea una referencia, lo cual explicaremos a continuación). Por ejemplo, en el código:

#include <iostream>

using namespace std;

void intercambiar(int, int); //Protipo de la función para intercambiar los valores

int main(void)

{

int a = 2,b = 3;

cout<<"Valores originales a = "<<a<<" y b = "<<b<<endl<<endl;

intercambiar(a,b); //Llmado a la función intercambiar

cout<<"Luego de la funcion a = "<<a<<" y b = "<<b<<endl<<endl;

return 0;

}

void intercambiar(int a, int b)

{

int z;

z = a;

a = b;

b = z;

}

Dará como salida:

Valores originales a = 2 y b = 3

Luego de la funcion a = 2 y b = 3

Esto muestra que el alcance de la declaración de los parámetros de la función es local a la misma.

1. OPCIONAL: trate de Investigar el “tema de pasaje de parámetros” x “valor y referencia”, sin entrar en temas como “punteros, tratado más adelante”, conceptualice y fundamente.

* Cuando se imprime el valor de las variables a y b se observa que de hecho no fueron alteradas después de la llamada a la función intercambiar( ), esto se debe a que en realidad lo que le pasamos a la función fue una *copia* de los valores de las variables a y b, por lo tanto era de esperar que las copias se cambiaran pero que finalmente los elementos originales permanecieran intactos. Esto es lo que se denomina “pasaje por valor”.
* Por otro lado, cuando se pasan valores por referencia, no se pasa una copia de los valores originales, sino que se crea (como su nombre lo indica) una referencia que tiene la misma dirección en memoria del elemento original, esto quiere decir dos cosas, en primer lugar la variable con la cual se trabajará al interior de la función que recibe la referencia será por definición un sinónimo de la variable original, esto quiere decir que será idéntica en cuanto a la dirección de memoria, su contenido, etc; Por otro lado esto también significa que cualquier modificación que hagamos en la función al contenido de la referencia afectará de forma inmediata a las variables originales.

Para pasar valores por referencia a una función, solo se debe anteponer el símbolo & (ampersand) en el nombre de la variable que será la referencia de las variables originales, es decir, el prototipo de la nueva función intercambiar( ) que ahora trabaja con referencias, será:

void intercambiar(int &i , int &j ); //Protipo de la función

**Si ahora modificamos nuestro código:**

#include <iostream>

using namespace std;

void intercambiar(int &a, int &b); //Protipo de la función para intercambiar los valores

int main(void)

{

int a = 2,b = 3;

cout<<"Valores originales a = "<<a<<" y b = "<<b<<endl<<endl;

intercambiar(a,b); //Llmado a la función intercambiar

cout<<"Luego de la funcion a = "<<a<<" y b = "<<b<<endl<<endl;

return 0;

}

void intercambiar(int &a, int &b)

{

int z;

z = a;

a = b;

b = z;

}

Dará como salida:

Valores originales a = 2 y b = 3

Luego de la funcion a = 3 y b = 2