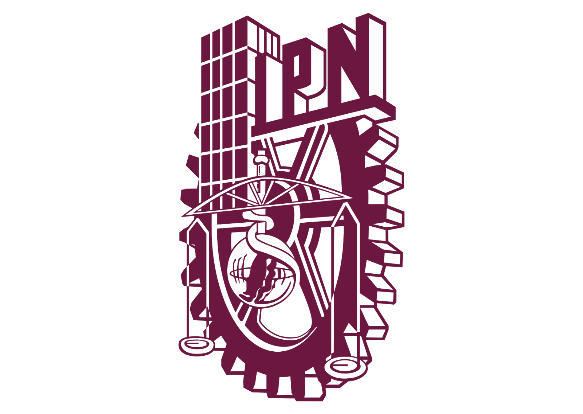
****

**INSTITUTO POLITECNICO NACIONAL**

**ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA**

**INGENIERIA EN COMUNICACIONES Y ELECTRONICA**

**FUNDAMENTOS DE PROGRAMACION**

**PROF. OSCAR CRUZ**

**Actividad 4**

**FUNCIONES DE USUARIO**

**Alumno:**

**DIAZ ANAYA EDUARDO**

**Boleta:**

**2020300206**

Desarrollar los temas de investigación para:

Definir que son las funciones de usuario

Tipos

Formas de declaración

Ejemplos

1.- **¿Qué las funciones de usuario?**

Una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir la cantidad de código. Cuando una función es invocada se le pasa el control a la misma, una vez que esta finalizó con su tarea el control es devuelto al punto desde el cual la función fue llamada.

Una función es un bloque de código que realiza alguna operación. Una función puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar argumentos a la función. Una función también puede devolver un valor como salida. Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función. La función siguiente acepta dos enteros de un llamador y devuelve su suma; a y b son parámetros de tipo int .

int sum(int a, int b)

{

return a + b;

}

Se puede invocar la función o llamarladesde cualquier número de lugares del programa. Los valores que se pasan a la función son los argumentos, cuyos tipos deben ser compatibles con los tipos de parámetro de la definición de función.

int main()

{

int i = sum(10, 32);

int j = sum(i, 66);

cout << "The value of j is" << j << endl; // 108

}

No hay ningún límite práctico para la longitud de la función, pero un buen diseño tiene como objetivo funciones que realizan una sola tarea bien definida. Los algoritmos complejos deben dividirse en funciones más sencillas y fáciles de comprender siempre que sea posible.

Las funciones definidas en el ámbito de clase se denominan funciones miembro. En C++, a diferencia de otros lenguajes, una función también pueden definirse en el ámbito de espacio de nombres (incluido el espacio de nombres global implícito). Estas funciones se denominan funciones libres o funciones no miembro; se usan en gran medida en la biblioteca estándar.

Las funciones se pueden sobrecargar, lo que significa que las distintas versiones de una función pueden compartir el mismo nombre si difieren en el número o el tipo de parámetros formales.

2.- Tipo de funciones en programación

Funcion VOID

Bajo ciertas circunstancias se deseará escribir funciones que no regresen valor alguno (esto sería algo parecido a escribir procedures en Pascal) y para ello podemos declarar a la función como void. La palabra reservada void es utilizada para declarar funciones sin valor de retorno y también para indicar que una función específica no requiere de parámetros. Por ejemplo, la función pausa() que se verá enseguida, no devolverá valor alguno y la misma no requiere de parámetros.

*// esta función requiere de la librería iostream*

void pausa(void)

{

cout << "Por favor presione <Enter> HOLA...";

cin.get();

cin.ignore(255, '\n'); *// rechazar caracteres introducidos antes de <Enter>*

}

***Notas:****se debe de aclarar que el uso de la palabra****void****dentro de los paréntesis es opcional al momento de declarar una función. Así, la función****pausa()****podría haberse declarado como****void pausa()****, y la misma puede invocarse como: pausa();.*

Funciones Anidadas

A diferencia de Pascal, el lenguaje C, C++ no permite anidar funciones, sin embargo, dentro de una función puede existir la llamada a una o más funciones declaradas previamente, que determinara en cierto punto un resultado para que sea ejecutado cuando el programador lo desee.

Funciones de tipo Puntero (\*)

En muchas ocasiones se desea que ciertas funciones regresen una referencia o puntero hacia un tipo (sea este estructurado o no) específico de dato en lugar de un valor específico. En tales casos, la función se deberá declarar como para que regrese un puntero. Por ejemplo, supongamos que deseamos crear una función para convertir un número entero en notación decimal a una cadena de caracteres en forma de números binarios, luego, la función mencionada podría escribirse para que reciba el número entero como parámetro y regrese un puntero a una cadena de caracteres conteniendo la conversión. Para ser más puntuales, vamos a escribir un programa en donde se verá la función binstr(), y cuyo objetivo será precisamente convertir números decimales en cadenas binarias.

*Nota: observe que en la sintaxis para declarar funciones tipo puntero se debe de poner el símbolo \* después del tipo y antes del nombre de la función que se está declarando. Esto se puede ver en el programa, ya que la función binstr se declara como: char \*binstr(unsigned int);*

#include *<iostream>*

#include *<cstring>*

**using** **namespace** std;

*// declaración de prototipo*

char \*binstr(unsigned int);

*// punto de prueba*

int main()

{

int n = 128;

cout << "decimal = " << n << ", binario = " << binstr(n) << endl;

cin.get();

}

*// definición de función binstr()*

*// nota: esta funcion requiere de la librería estándar string*

char \*binstr(unsigned int n)

{

**static** char buffer[65];

int i = 0;

strcpy(buffer, "0");

**if** (n > 0) {

**while** (n > 0) {

buffer[i] = ( n & 1 ) + '0';

i++;

n >>= 1;

}

buffer[i] = '\0';

strrev(buffer);

} *// fin (n > 0)*

**return** buffer;

}

3.- Formas de declaración de Funciones

La declaración da a conocer la función al compilador, de forma que a partir del punto de declaración, ya se pueden realizar invocaciones a la misma. A su vez, la definición estará en algún otro punto del programa, tal vez en una librería externa (en forma ya compilada) o en otro módulo de programa (como texto fuente).

Una función puede ser declarada varias veces en un mismo programa, y las declaraciones pueden aparecer en cualquier orden; en un fichero fuente o en varios, pero en cualquier caso antes de su uso, es decir: antes de cualquier invocación a la función. Además de declarar el nombre de la función y el tipo devuelto (por defecto se supone int ) se declaran también el tipo de los parámetros.

extern int funcion1 ();        // no acepta ningún argumento  
extern int funcion1 (void);    // mejor que la anterior  
funcion2 (char, int);          // por defecto supone que devuelve int  
int funcion2 (char, int);      // mejor que la anterior  
char funcion3 (char c, int i); // incluye nombres de parámetros

El especificador <tipo-devuelto> es opcional. Por defecto se supone int, así que las declaraciones que siguen son equivalentes:

**int** func (<tipo> <parámetro>, ...)  
func (<tipo> <parámetro>, ...)

*Nota: los compiladores MS Visual C++ y Borland C++ admiten que ciertos especificadores opcionales acompañen a la declaración de funciones y otros objetos. Tales especificadores son de utilidad en circunstancias específicas*

Recuerde que las declaraciones deben realizarse antes que cualquier uso de la función. A su vez, las definiciones pueden estar en cualquier sitio, aunque en algunos casos puede haber excepciones.

Las declaraciones de funciones tienen un nombre específico: se denominan prototipo . El primero de los anteriores es válido, aunque desaconsejado (herencia del C); es el denominado estilo clásico Kernighan & Ritchie. El segundo y tercero, son los aceptados en C++.

*Nota: es importante resaltar que en la declaración de parámetros no está permitido incluir funciones, es decir, las funciones no pueden pasar como argumentos a otras funciones. Sin embargo, C++ dispone de recursos cuando esto es necesario; pasar un puntero o una referencia a la función.*

El viejo estilo K&R tiene la desventaja de no permitir al compilador comprobar el número y tipo de los argumentos utilizados en las llamadas a la función. Este problema fue eliminado con la introducción de los prototipos que utilizan la forma completa, en la que se especifica el número y tipo de cada argumento aceptado por la función. El compilador usa estos datos para comprobar la validez de las llamadas a la función y como se ilustra en el ejemplo, es capaz en su caso, de realizar dentro de ciertos límites, un modelado de tipo ("Casting") de los argumentos para garantizar que coinciden con el tipo esperado.

BIBLIOGRAFIAS.

<https://docs.microsoft.com/es-es/cpp/cpp/functions-cpp?view=vs-2019>

<https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Funciones>

<https://www.zator.com/Cpp/E4_4_1.htm>

<http://progra.usm.cl/apunte/materia/funciones.htm>