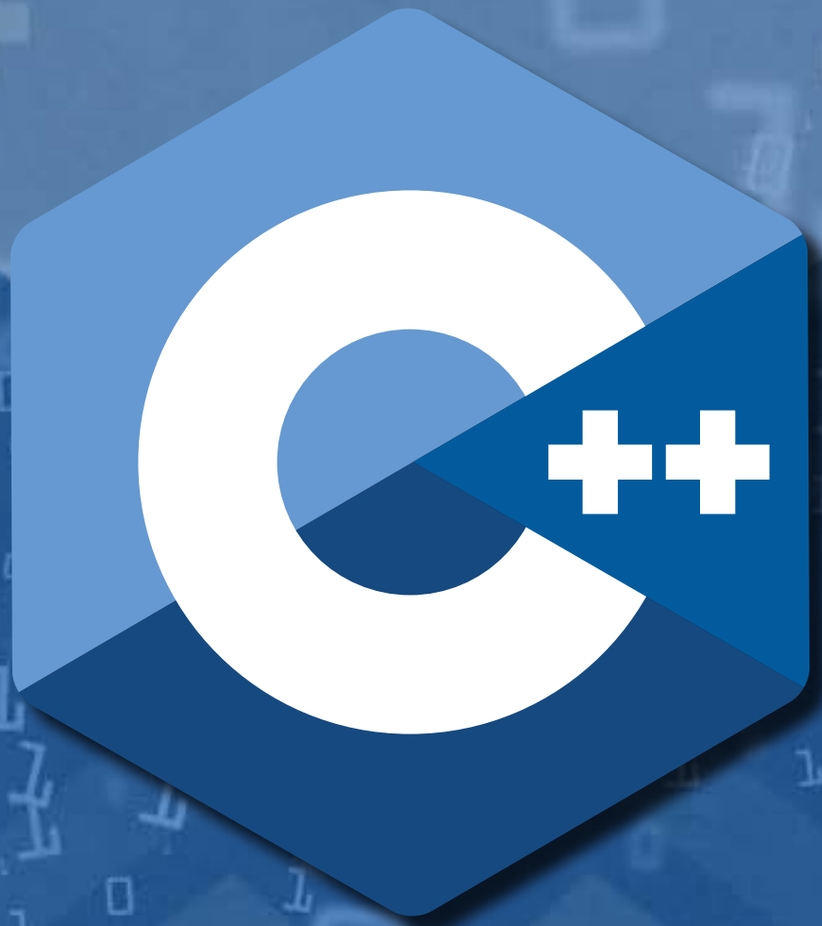
CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.



Entrar

Material de Actividad de Aprendizaje 1

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

Índice:

Material Actividad de Aprendizaje 1.

1. Elementos básicos del lenguaje C++.

C, predecesor de C++,

1.3. Estructura de un programa en C++.

La estructura de un programa en C++ está definida por los siguientes componentes.

Un programa en C++ está definido por funciones (grupo de instrucciones que pueden o no hacer algún cálculo), donde la función principal debe ser llamada **main.**

La composición general de un programa en C++ es:

Directivas de preprocesamiento.
Declaraciones globales.
Función main.
Funciones definidas por el usuario.
Comentarios para entender el funcionamiento del programa.

Directivas de preprocesador

Las directivas de preprocesador, como #define y #ifdef, se utilizan normalmente para facilitar, cambiar los programas de origen y facilitar la creación en diferentes entornos de ejecución. Las directivas del archivo de código fuente indican al preprocesador que realice acciones específicas. Por ejemplo, el preprocesador puede reemplazar tokens en el texto, insertar el contenido de otros archivos en el código fuente o suprimir la compilación de la parte del archivo quitando secciones de texto. Las líneas de preprocesador se reconocen y se realizan antes de expansión de macro. Por consiguiente, si una macro se expande en algo que se parece a un comando de preprocesador, el preprocesador no reconoce a ese comando.

Las instrucciones de preprocesador utilizan el mismo juego de caracteres que instrucciones del archivo de código fuente, con la excepción de que las secuencias de escape no se admiten. El juego de caracteres utilizado en instrucciones de preprocesador es igual que juego de caracteres de la ejecución. El preprocesador también reconoce los valores de caracteres negativos.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

- **El preprocesador reconoce las siguientes directivas:**

# define	# error	# import	# undef
# elif	# if	# include	# using
# else	# ifdef	# line	# endif
# ifndef	# pragma		

El signo de número (#) debe ser el primer carácter de la línea, que no es un carácter de espacio en blanco que contiene la directiva; los caracteres de espacio en blanco pueden estar entre el signo de número y la primera letra de la directiva. Algunas directivas incluyen argumentos o valores. Cualquier texto que siga una directiva (excepto un argumento o un valor que forma parte de la directiva) debe ir precedido por un delimitador de comentario de una sola línea (//) o incluido entre en delimitadores de comentario (/* */). Las líneas que contienen directivas de preprocesador se pueden continuar precediendo inmediatamente la marca de fin de línea con una barra diagonal inversa (\).

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

Las directivas de preprocesador pueden aparecer en cualquier parte de un archivo de código fuente, pero solo se aplican al resto del archivo de código fuente.

La directiva **#include** indica al preprocesador que trate el contenido de un archivo especificado como si esos contenidos hubiesen aparecido en el programa de origen en el punto donde aparece la directiva.

Se usa para incluir bibliotecas de funciones al archivo actual, se pueden incluir archivos propios o del sistema. En la medida que se avance se hará claridad acerca de este tema.

```
#include  "path-spec"
```

```
#include  <path-spec>
```

La directiva **#define** crea una macro, que es la asociación de un identificador o identificador de parámetros con una cadena de token. Una vez definida la macro, el compilador puede sustituir la cadena de token para cada aparición del identificador del archivo de código fuente.

La función main () (Principal) es el punto de entrada en un programa de C++, es una función obligatoria y única, solo puede existir una función main por proyecto de C++.

Los comentarios son líneas aclaratorias no compilables, el compilador reconoce los juegos de caracteres // o /* */

Los comentarios pueden escribirse utilizando alguna de las siguientes formas:

1. Comentando en la misma línea, utiliza //, ejemplo int edad; // la edad se utilizará como un valor entero.
2. Comentando entre varias líneas, utiliza /* */, **Ejemplo:**

/* La siguiente función promedio recibe tres valores enteros y calcula y regresa el promedio de los tres a través de un valor real */

1.3.1. Tipos de datos en C++.

El concepto de tipo es muy importante en C++. Cada variable, argumento de función y el valor devuelto por una función debe tener un tipo para compilarse. Asimismo, ante de evaluar todas las expresiones (incluyendo valores literales), el compilador les da implícitamente un tipo. Algunos ejemplos de tipos incluyen int para almacenar los valores integrales, double para almacenar los valores de punto flotante (también conocidos como tipos de datos escalares), o la clases td::basic_string de biblioteca estándar para almacenar texto. Puede crear su propio tipo definiendo class o struct. El tipo específica la cantidad de memoria que se asignará para la variable (o resultado de la expresión), las clases de valores que se pueden almacenar en esa variable, cómo se interpretan estos valores (como patrones de bits), así como las operaciones que se pueden realizar en ella.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

A diferencia de algunos lenguajes, C++ no tiene tipo base universal del que derivan el resto de los tipos. La implementación de Visual C++ de lenguaje incluye muchos tipos fundamentales, también conocidos como tipos integrados. Esto incluye los tipos numéricos como int, double, long, bool, más los tipos char y wchar_t para los caracteres ASCII y Unicode, respectivamente. La mayoría de los tipos fundamentales (excepto bool, double, wchar_t y tipos relacionados) tienen versiones sin signo, que modifican el intervalo de valores que la variable puede almacenar. Por ejemplo, int, que almacena un entero de 32 bits con signo, puede representar un valor comprendido entre -2.147.483.648 y 2.147.483.647.unsigned int, que también se almacena como 32 bits, puede almacenar un valor comprendido entre 0 y 4.294.967.295. El número total de valores posibles en cada caso es igual; solo el intervalo es diferente.

El compilador reconoce los tipos fundamentales, que tiene reglas integradas que rigen las operaciones que se pueden realizar en ellos, y cómo se pueden convertir en otros tipos fundamentales.

C++ no soporta un gran número de tipos de datos predefinidos, pero tiene la capacidad para crear sus propios tipos de datos. Todos los tipos de datos simples o básicos de C++ son, esencialmente, números. Los tres tipos de datos básicos son:

- Enteros.
- Números de coma flotante (reales).
- Caracteres.

1.3.2. Tipos de datos numéricos.

Los tipos de datos numéricos en C++ están divididos en enteros y reales, y a su vez, estos tipos enteros están divididos de acuerdo al tamaño de almacenamiento que ocupa.

Los tipos de datos enteros se dividen en:

Tipo	Contenido
Char	El tipo **char** es un tipo entero que normalmente contiene miembros de juego de caracteres de la ejecución (en Microsoft C++, esto es ASCII).
	El compilador de C++ trata las variables de tipo **char, signed char** y unsigned **char** como si tuvieran tipos diferentes. Las variables de tipo char se promueven a int como si fueran de tipo **signed char** de forma predeterminada, a menos que se use la opción de la compilación /J. En este caso se tratan como tipo **unsigned char** y se promueven a int sin la extensión de signo.
Bool	El tipo **bool** es un tipo entero que puede tener uno de los dos valores **true** o **false**.Su tamaño no está especificado.
Short	El tipo **short int** (o simplemente **short**) es un tipo entero que es mayor o igual que el tamaño del tipo char y menor o igual que el tamaño del tipo **int**.

Tipo	Contenido
	Los objetos de tipo **short** se pueden declarar como **signed short** o **unsigned short**. **Signed short** es un sinónimo de **short**.
Int	El tipo **int** es un tipo entero que es mayor o igual que el tamaño del tipo **short int** y menor o igual que el tamaño del tipo **long**.
	Los objetos de tipo **int** se pueden declarar como **signed int** o **unsigned int.Signed int** es un sinónimo de **int.**
long	El tipo **long** (o **long int**) es un tipo entero que es mayor o igual que el tamaño del tipo **int**.
	Los objetos de tipo ***long*** se pueden declarar como **signed long** o u**nsigned long.Signed long** es un sinónimo de **long**.
Long long	Mayor que **long** sin signo.
	Los objetos de tipo **long long** se pueden declarar como **signed long long ounsigned long long**. Signed long long es un sinónimo de long **long**.

1.1.3 Tipos de datos numéricos reales.

Tipo	Contenido
Float	El tipo **float** es el tipo flotante menor.
Double	El tipo **double** es un tipo flotante mayor o igual que el tipo **float**, pero es menor o igual que el tamaño del tipo **long double**.
Long Double 1	El tipo **long double** es un tipo flotante que es igual que el tipo **double.**

1.1.4 Tipos de datos carácter:

Los tipos de datos de carácter en C++ se dividen en carácter y cadenas.

Los tipos carácter son:

Tipo	Contenido
Char	char El tipo **char** es un tipo entero que normalmente contiene miembros de juego de caracteres de la ejecución (en Microsoft C++, esto es ASCII).
__wchar_t	Una variable de __wchar_t indica un tipo de caracteres anchos o de carácter multibyte. De forma predeterminada, wchar_t es un tipo nativo pero puede utilizar/Zc: wchar_t- para crear wchar_t una definición para unsigned short. Utilice el prefijo L antes de un carácter o una constante de cadena para notificar la constante de tipo de caracteres anchos.

1.1.5 Tipos referencias.

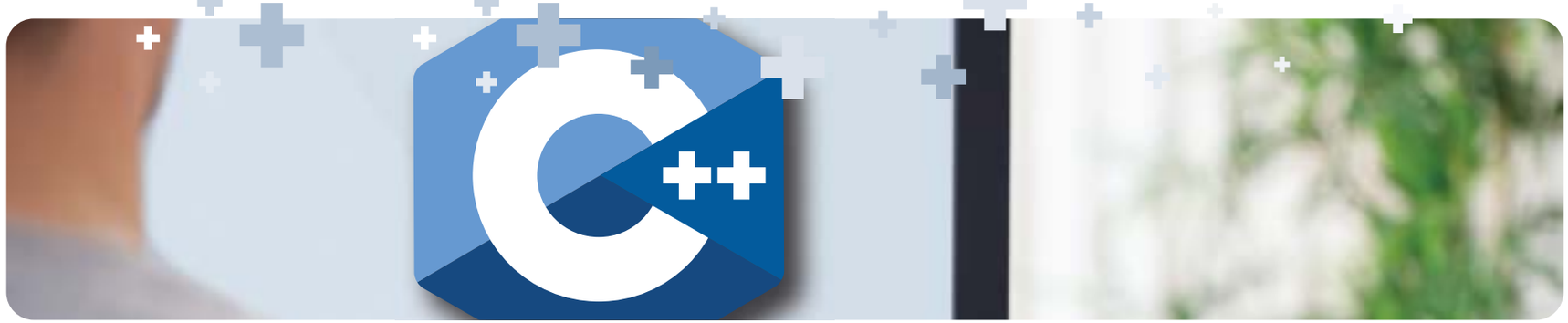
void (C++)

Cuando se utiliza como un tipo de valor devuelto de función, la palabra clave void especifica que la función no devuelve ningún valor. Cuando se utiliza para la lista de parámetros de una función, void especifica que la función no toma ningún parámetro. Cuando se utiliza en la declaración de un puntero, void especifica que el puntero es "universal".
Si el tipo de puntero es void *, el puntero puede señalar a cualquier variable que no se declare con la palabra clave const o volatile. Un puntero void no se puede desreferenciar a menos que se convierta en otro tipo. Un puntero void se puede convertir en cualquier otro tipo de puntero de datos.
Un puntero void puede señalar a una función, pero no a un miembro de clase en C++.
No se puede declarar una variable de tipo void.

- **Ejemplo**

```
// void.cpp
void vobject;   // C2182
void *pv;   // okay
int *pint; int i;
int main() {
pv = &i;    // Cast optional in C required in C++
pint = (int *)pv;
}
```

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.



1.2 Constantes y variables.

Variable: el nombre simbólico de una cantidad de datos, de forma que el nombre se pueda utilizar para tener acceso a los datos que se hace referencia en el ámbito del código donde se define. En C++, “variable” se utiliza normalmente para hacer referencia a las instancias de tipos de datos escalares, mientras que las instancias de otros tipos normalmente se denominan “objetos”.

Constantes:

Los elementos invariables de un programa se denominan “literales” o “constantes.” Los términos “literal” y “constante” se utilizan aquí indistintamente. Los literales tienen cuatro categorías principales: enteros, carácter, flotante, y literales de cadena.

Así:

```
157 // integer constant
0xFE // integer constant
'c' // character constant
0.2 // floating constant
0.2E-01 // floating constant
"dog" // string literal
```

Las constantes enteras son los datos constantes que no tienen ninguna parte fraccionaria o exponente. Comienzan siempre con un dígito. Pueden especificar signo o tipos sin signo.

```
123
987
1027
-125
-236
```

Las constantes de coma flotante especifican los valores que deben tener una fracción. Estos valores contienen los separadores decimales (.) y pueden contener exponentes.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

Las constantes de coma flotante tienen una “mantisa”, que especifica el valor de número, un exponente “,” de los que especifica la magnitud del número, y un sufijo opcional que especifica el tipo de constante. La mantisa es especificada como una secuencia de dígitos seguido de un punto, seguido de una secuencia opcional de dígitos que representan a la parte fraccionaria del número. Por ejemplo:

```
18.46
38.1

18.46e0      // 18.46
18.46e1      // 184.6
```

Las constantes de coma flotante tienen una “mantisa”, que especifica el valor de número, un exponente “,” de los que especifica la magnitud del número, y un sufijo opcional que especifica el tipo de constante. La mantisa es especificada como una secuencia de dígitos seguido de un punto, seguido de una secuencia opcional de dígitos que representan a la parte fraccionaria del número. Por ejemplo:

```
18.46
38.1

18.46e0      // 18.46
18.46e1      // 184.6
```

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

El exponente se puede especificar mediante E o E, que tienen el mismo significado, seguido de un signo opcional (+ o -) y una secuencia de dígitos. Si un exponente está presente, el separador decimal final es innecesario en números enteros como 18E0.

Una constante de carácter es cualquier carácter imprimible entre comillas simples ‘a’.

Para la gran mayoría de compiladores de C/C++, el código fuente y los juegos de caracteres de ejecución son ASCII.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

El juego de caracteres básico de origen consta de 96 caracteres: el carácter de espacio; los caracteres de control que representan la tabulación horizontal, la tabulación vertical, el avance de página y la nueva línea; y los 91 caracteres siguientes:

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRST UVWXYZ
0123456789
_{}[]#()<>%:;.?*+-/^&|~!=,\'"

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

El juego de caracteres básico de ejecución está formado por caracteres del juego de caracteres básico y también los caracteres de control que representan alerta, retroceso, retorno de carro y null.

Esta es la tabla de los caracteres de escape.

Carácter	ASCII Representación	ASCII Valor	Secuencia de escape
Nueva línea	NL (LF)	10 o 0x0a	\n
Tabulación horizontal	HT	9\	t
Tabulación vertical	VT	11 o 0x0b	\v
Retroceso	BS	8\	b
Retorno de carro	CR	13 o 0x0d	\r

Carácter	ASCII Representación	ASCII Valor	Secuencia de escape
Avance de página	FF	12 o 0x0c	\f
Alerta	BEL	7\	a
Barra diagonal inversa	\	92 o 0x5c	\\
Signo de interrogación	?	63 o 0x3f	\?
Comilla simple	'	39 o 0x27	\'

Carácter	ASCII Representación	ASCII Valor	Secuencia de escape
Comilla doble	"	34 o 0x22	\"
Número octal	ooo	—\	ooo
Número hexadecimal	hhh	—\	xhhh
Carácter nulo	NUL	0\	0

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

Si el carácter que sigue a la barra diagonal inversa no específica una secuencia de escape válida, el resultado es implementación definida. En Microsoft C++, el carácter que sigue a la barra diagonal inversa se toma literalmente, como si el escape no estuviese presente y se emite una advertencia de nivel 1 (“secuencia de escape de carácter no reconocida”).

Un literal de cadena consta de cero o más caracteres de juego de caracteres de origen incluido entre comillas ("). Un literal de cadena representa una secuencia de caracteres que, en conjunto, forman una cadena terminada en null.

Los literales de cadena pueden contener cualquier carácter gráfico de juego de caracteres de origen excepto comillas ("), la barra inversa (\), o el carácter de nueva línea. Pueden contener las mismas secuencias de escape descritas en Literales de caracteres de C++.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

Variables:

Una variable representa los valores guardados en memoria durante la ejecución de un programa C++, hay que recordar que los datos de un programa se almacenan en memoria RAM (Randon Acces Memory) durante su ejecución y de manera persistente en el disco duro.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

El nombre de una variable se conoce como identificador. Un identificador es una secuencia de caracteres utilizados para denotar uno de los siguientes:

- Objeto o nombre de variable.
- Nombre de clase, estructura o unión.
- Nombre de tipo enumerado.
- Miembro de una clase, estructura, unión o enumeración.
- Función o función miembro de clase.
- Nombre de definición de tipos.
- Nombre de etiqueta.
- Nombre de macro.
- Parámetro de macro.

Hay ciertas reglas que se deben cumplir para asignarle el nombre a una variable y que el compilador las pueda reconocer sin que ocasionen errores.

- Los nombres de variables deben empezar con letras (a-z o A-Z), o el carácter _ esto es necesario para que el compilador pueda detectar que se trata de una variable.

- Después del primer carácter se pueden usar caracteres numéricos o letras.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

- No se deben usar caracteres especiales en los nombres de variables
- Las palabras reservadas del lenguaje C++ no se deben usar como nombres de variables.

Aquí encontraran una lista de las palabras reservadas de C++

Aquí encontraran una lista de los caracteres que se pueden usar para nombrar variables.

Nombres de variables bien escogidos ayudan a documentar el programa, por ejemplo una variable llamada montoPrestamo, por simple lógica podríamos saber que contiene el valor de un préstamo.

Estos datos de acuerdo a su valor y naturaleza ocupan un espacio en la memoria en bytes para su optimización C++ define tipos de datos vistos anteriormente.

1.2.1 Declaraciones de variables.

C++ es un lenguaje fuertemente tipado y también es de tipos estáticos; cada objeto tiene un tipo y ese tipo nunca cambia (no confundir con objetos de datos estáticos).

Al declarar una variable en el código, debe especificar explícitamente su tipo o utilizar la palabra clave auto para indicar al compilador que deduzca el tipo de inicializadores.

Al declarar una función en el código, debe especificar el tipo de cada argumento y su valor devuelto o void si no se devuelve ningún valor por la función.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.



Después declarar primero una variable, no puede cambiar su tipo posteriormente. Sin embargo, puede copiar el valor devuelto del valor de variable o de una función en otra variable de un tipo diferente. Estas operaciones se denominan conversiones de tipo, que son a veces necesarias pero también son posibles orígenes de pérdida de datos o incorrección.

- **La forma de declarar variables en C++ es la siguiente:**

  Tipo identificador;

  - **Ejemplos:**

  ```
  char c;
  int b;
  ```

También se le pueden dar valores de inicialización

```
char c='a';
int b=10;

int result = 0;              // Declare and initialize an integer.
double coefficient = 10.8;   // Declare and initialize a floating

auto name = "Lady G.";       // Declare a variable and let compiler
                             // deduce the type.
auto address;                // error. Compiler cannot deduce a type
                             // without an intializing value.
age = 12;                    // error. Variable declaration must
                             // specify a type or use auto!
result = "Kenny G.";         // error. Can't assign text to an int.
string result = "zero";      // error. Can't redefine a variable with
                             // new type.
int maxValue;                // Not recommended! maxValue contains
                             // garbage bits until it is initialized.
```

1.2.2. Funciones.

Tanto en los lenguajes estructurados o en los orientados a objetos se mantiene la máxima de la programación “Divide y vencerás”, esto se debe a que se recurre a la descomposición sucesiva del problema en partes pequeñas que son más fáciles de resolver, porque su complejidad es baja en comparación con el problema completo. Las funciones se usan para encapsular la solución de estas pequeñas partes del problema.

Una función es una agrupación de sentencias que realizan una acción específica ya sea realizar un cálculo en cuyo caso devuelve valor o simplemente la realización de una tarea, por ejemplo trazar una línea en la pantalla.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

En la mayor parte de los lenguajes de programación las subrutinas están divididas en dos grupos bien diferenciados, las que devuelven valor (Funciones) y las que no (Procedimientos). En C/C++ esta distinción no existe sino meramente en lo conceptual; en la implementación solo las diferencia el tipo de retorno.

- **La forma general de una función es la siguiente:**

```
Tipo_de_dato nombre_funcion(lista de argumentos)
{
   int ..
   .
   .
   .

}
```

Bloque de instrucciones o sentencias

El tipo de datos representa el valor a devolver por parte de la función, el nombre de la función es el nombre que se la ha asignado a la función, puede ser cualquiera, sin embargo se prefiere que sea un nombre que refleje el objeto o la acción que cumple la función.

1.2.3. La función main.

La función main() es una secuencia de llamadas a otras funciones que pueden llamar a otras funciones. C++ proporciona los medios para construir nuestras propias funciones así como también bibliotecas de funciones, tales como la stdio.h de ANSI C, y las específicas de C++ como stream, iostream y otras.

Al igual que cualquier función en C++ la función main es un subprograma que puede o no devolver un valor o simplemente realizar una acción.

- Su forma es:

```
int main()
{
   int ..
   .
   .
   .

}
```

Bloque de instrucciones o sentencias

1.2.4. Funciones de la biblioteca estándar de C++.

La biblioteca estándar de C++ está compuesta por una colección de archivos que contienen funciones predefinidas que ayudan al programador a realizar tareas comunes, estos archivos son archivos de extensión .h y son llamados archivos de cabecera.

Un archivo de cabecera es un archivo especial que contiene las declaraciones de objetos y funciones de la biblioteca que son añadidos en el lugar donde se insertan. Un archivo de cabecera se inserta con la directiva #include . El nuevo ANSI C++ ha cambiado el convenio (notación) original de los archivos de cabecera. Es posible utilizar sólo los nombres de las bibliotecas sin el sufijo .h; es decir, se puede usar iostream , cmath , cassert y cstlib en lugar de iostream.h , math.h , assert.h y stdlib.h respectivamente.

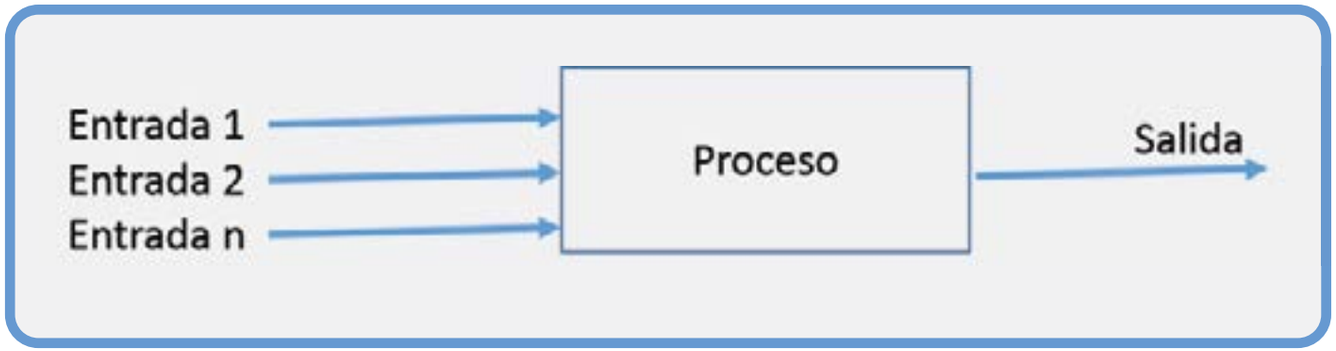
CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

- **Los archivos de cabecera que son parte de la biblioteca estándar de C++ son:**

<bitset>	<algorithm>	<fstream>	.<streambuf>
<deque>	<functional>	<ios>	<complex>
<list>	<iterator>	<iostream>	<numeric>
<map>	<locale>	<iosfwd>	<valarray>
<queue>	<memory>	<iomanip>	<exception>
<set>	<stdexcept>	<istream>.	<limits>
<stack>	<utility>	<ostream>	<new>
<vector>	<string>	<sstream><	typeinfo>

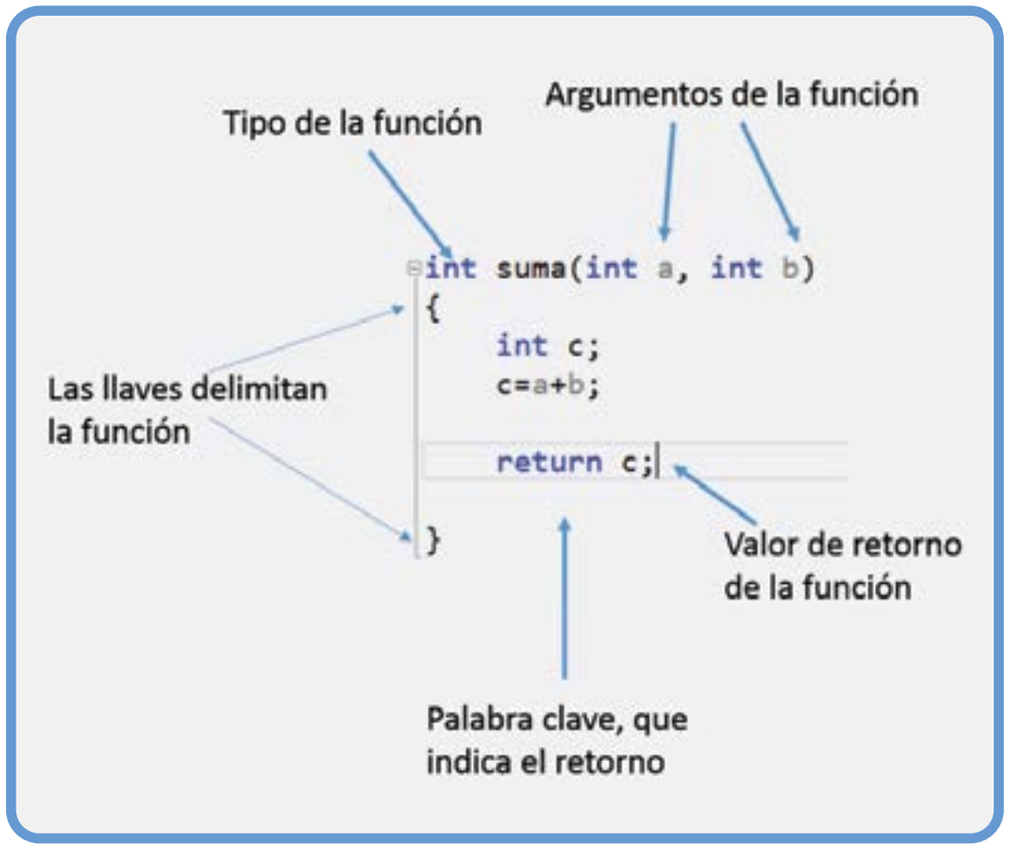
1.2.5. Funciones definidas por el usuario.

Una función también se puede considerar un subprograma por lo que se podría aplicar el siguiente esquema:



Para obtener las entradas, en adelante parámetros o argumentos, la función usa la lista de argumentos, que no es otra cosa que variables que contienen los datos de entrada, la forma de una función con lista de argumentos, que suma dos números enteros y devuelve un valor entero es la siguiente.

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.



Donde a, b son dos variables enteras, y son los argumentos o parámetros de la función, las llaves delimitan los bloques de código en C++, por lo que se usan aquí para delimitar el alcance de la función, las líneas de código en adelante sentencias, terminan en (“;”) , por último una función que devuelve valor, debe terminar con la palabra clave “return” que indica el fin de la función y debe devolver un valor del tipo de la función.

1.3. Funciones de entrada y salida.

C++, no define explícitamente un método de entrada y salida (E/S), la E/S la proporciona la biblioteca estándar, la directiva

```
#include <iostream>
```

Añade el contenido del archivo de cabecera <iostream> que permite manipular los flujos de entrada y salida.

Al añadir el archivo <iostream> podemos usar los siguientes símbolos

```
std::cin
std::cout
std::endl
std::cerr
```

CONCEPTUALIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN C++.

Con estos símbolos podemos capturar el valor de variables de entrada por teclado (cin) y mostrar (salida) el valor de variables en pantalla (cout). Una forma fácil de entender estas y otras funciones de C/C++ es leerla así; cin -> C de C++ y “in” de “Dentro de” en inglés y cout -> C de C++ y “out” de “fuera” en inglés.

- **Un uso típico de estas funciones sería:**

```
#include <iostream>

int main()
{
    int a,b,c;
    std::cout<<"digite valores para las variables a y b"<<std::endl;
    std::cin>>a;
    std::cin>>b;
    std::cout<<"El contenido de la variable a es:"<<a<<std::endl;
    std::cout<<"El contenido de la variable b es:"<<b<<std::endl;

}
```

Donde std es el espacio de nombre, donde están contenidas las especificaciones de los símbolos cin, cout y endl; que son los que se muestran en la figura.

Webgrafía:

- CPlayMasH. (2013). Programación en C - TIPOS DE DATO - Datos Primitivos. Consultado el 19 de abril de 2014, en http://www.youtube.com/watch?v=KoKQppboxRY
- Valderrama, T. (2009). Funciones en C, parte I. Consultado el 19 de abril de 2014, en http://www.youtube.com/watch?v=fOBbxNZfR2E
- Visual Studio. (2010). Visual Studio Express. Consultado el 19 de abril de 2014, en http://www.visualstudio.com/es-es/downloads/download-visual-studio-vs#DownloadFamilies_4
- Code Block. (2013). Downloads. Consultado el 19 de abril de 2014, en http://www.codeblocks.org/downloads
- Ebrary biblioteca. (2013). C++ ahora. Consultado el 19 de abril de 2014, en http://site.ebrary.com/lib/senavirtualsp/docDetail.action?docID=10491298&adv.x=1&p00=programacion&f00=all&p01=%22C%2B%2B+%28Lenguaje+De+Programaci%C3%B3n%29%22&f01=subject.
- Microsoft Developer Network. (2013). Liberales de C++. Consultado el 19 de abril de 2014, en http://msdn.microsoft.com/es-es/library/c70dax92.aspx?sentenceGuid=2a5a82ae3b9433b8cda529f8708863f9#mt2

Bibliografía:

- J. Aguilar, L. Sánchez García. (2006). Programación en C++: un enfoque práctico. Madrid, España: Editorial. McGraw-Hill.
- J. Aguilar, L. Sánchez García. (2006).Programación en C++: algoritmos, estructuras de datos y objetos. (2a. Edicion).Madrid, España: Editorial. McGraw-Hill.
- J. Aguilar, L. Zahonero Martínez. (2010). Programación en C, C++, Java y UML. Madrid, España: Editorial. McGraw-Hill.
- R. Winder. (1995). Desarrollo de software con C++.
Editorial. Ediciones Díaz de Santos.