CTIC - UNI
Unidad de Capacitación

Sesión 5: Programación Modular

Víctor Melchor Espinoza

Agenda

- Definición de función. Llamada a una función.
- Funciones y procedimientos.
- Variables locales y globales.
- Parámetros por valor y por referencia.
- Recursividad.

Introducción

Hasta el momento, cada programa que hemos elaborado ha consistido de una función llamada **main**, que para llevar a cabo sus tareas ha utilizado funciones estándar de biblioteca.

Las funciones son los componentes básicos de los programas de C++. Cada función es esencialmente un pequeño programa, con sus propias declaraciones y sentencias.

La Biblioteca <math.h>

Función	Descripción	Ejemplo
sqrt(x)	Raiz cuadrada de x	sqrt(16); // da 4.0
fabs(x)	Valor absoluto de x	fabs(-19.2); // da 19.2
ceil(x)	Redondea x al entero más pequeño que no sea menor que x	ceil(-9.8) es -9.0
floor(x)	Máximo entero de x en formato real	floor(9.2) es 9.0
sin(x)	Seno de x (x en radianes)	sin(0.0) es 0.0
cos(x)	Coseno de x(x en radianes)	cos(0.0) es 1.0
tan(x)	Tangente de x(x en radianes)	tan(0.0) es 0.0
log(x)	Logaritmo natural de x (de base e)	log(2.718282) es1.0

Introducción

•Un programa en C++ está formado generalmente por un conjunto de funciones. Estas funciones posteriormente consisten de muchas sentencias de programación.

•Mediante el uso de funciones, una gran tarea se puede descomponer en otras más pequeñas.

•Las funciones son importantes debido a su reutilización. Es decir, los usuarios pueden desarrollar un programa de aplicación en base a lo que otros han hecho. Ellos no tienen por qué empezar de cero.

Definición de una función

Una función se puede definir de la siguiente forma:

```
tipo_retorno  nombre_funcion(declaracion de argumentos)
{
    declaraciones
    sentencias
}
```

El **tipo_retorno** de una función es el tipo de valor que la función retorna. Las siguientes reglas gobiernan el tipo de retorno:

Las funciones no pueden retornar arrays, pero no hay otra restricción en el tipo de retorno.

Ejemplo de escritura de una función

Ejemplo:

Elabore un programa que solicite dos números enteros de entrada y obtenga como salida la suma de ambos.

Invocación a funciones

Para llamar a una función se especifica su nombre y la lista de argumentos sin escribir el tipo de dato.

```
nombre_funcion(arg1, arg2,…, argN);
```

Lugares de la llamada:

Puede realizar la llamada una o varias veces en el programa, siempre desde una función que puede ser:

- La función main.
- Otra función.
- La misma función: recursividad.

Clasificación de los parámetros

Tipos de parámetros:

Parámetros formales: aparecen en la definición de la función.

Parámetros actuales: los que se pasan en la llamada a la función.

Parámetros actuales vs formales

En una llamada habrá un argumento actual por cada argumento formal, respetando el orden de la declaración.

Los parámetros actuales pueden ser:

Constantes.
Variables simples.
Expresiones complejas.

Los parámetros actuales deben ser del mismo tipo de datos que el argumento formal correspondiente.

Cuando se pasa un valor a una función se copia el argumento real en el argumento formal.

Reutilización

Ejemplo:

Elabore un programa que incluya la función promedio. El programa leerá tres números y calcula sus promedios un par a la vez.

```
Ingrese tres números:  3.5    9.6    10.2

Promedio de  3.5   y   9.6:  6.55
Promedio de  9.6   y   10.2:  9.9
Promedio de  3.5   y  10.2:  6.85
```

La sentencia return

El tipo de retorno puede ser cualquier especificador de tipo válido.

La sentencia **return** se puede utilizar para devolver un valor desde la función llamada a la función llamadora;

Si es necesario, la expresión se convertirá al tipo de retorno de la función. Sin embargo, si la expresión no puede convertirse en el tipo de retorno de la función de acuerdo a las reglas de conversión de tipos de datos incorporados implícitamente, se tratará de un error de sintaxis.

Uso de la sentencia return

¿Qué devuelve el siguiente segmento?

```
int   funcion(void)
{
      double d;
      d = 4.6;
      return  d;
}
```

Tipo de retorno no void

Si el tipo de retorno no es **void**, es necesaria una instrucción de retorno, al final de la función. De lo contrario, se utilizará el cero como valor de retorno predeterminado y un mensaje de advertencia será producido por el sistema.

La sentencia return

•Una función llamadora puede ignorar libremente el valor de retorno.

```
int  siguiente(int  i){
    return      i+1;
}
int  main() {
    int j;
    j = siguiente(4);
    siguiente(5);
    return 0;
}
```

Tipo de retorno void

Si el tipo de retorno es void, la sentencia return es opcional. Sin embargo, luego de return no debe seguir ninguna expresión, de lo contrario, se tratará de un error de sintaxis .

La sentencia return

En el siguiente fragmento detecte un error de sintaxis.

```
void  compara(int  i)
{
      if (i== 3){
              cout<<“i es igual  a 3 \n";  return i;
            }
     else if (i> 3){
            cout<<“i no es igual a 3 \n";
            return;}
      i= -1;
}
int main(){
     compara(2);
     return 0;
}
```

La función exit

Una manera de terminar un programa es ejecutar una sentencia return en main.

Otra forma es llamar a la función **exit,** la cual pertenece a <stdlib.h>.Para indicar la terminación normal pasamos

```
exit(0);
```

C++ nos permite pasar EXIT_SUCCESS:

```
exit (EXIT_SUCCESS);
```

Pasamos EXIT_FAIURE para indicar una terminación anormal:

```
exit (EXIT_FAILURE);
```

Funciones como procedimientos

Algunas funciones no tienen ningún parámetro.

La palabra **void** dentro de los paréntesis indica que la función no tiene argumentos.

Para llamar a una función sin argumentos, escribiremos el nombre de la función seguida por los paréntesis.

Ejemplo:

Elabore un programa que invoque a una función, la cual imprimirá un mensaje en pantalla.

Argumentos actuales y formales

El tipo de datos del argumento actual de la función llamadora puede ser diferente de la del argumento formal de la función llamada, siempre y cuando sean compatibles.

El valor de un argumento actual se convierte en el tipo de datos de su definición formal de acuerdo con las reglas de conversión de datos incorporados implícitamente en la fase de interfaz de la función.

Ejemplo

Ejemplo:

¿Son correctas las dos llamadas a la función doble?

```
int  doble(int  i)
      { return 2*i;
}
int main(){
     doble(5);
     doble(5.0);
     return 0;
}
```

Ámbito de las variables

Variables locales

Las variables que declaramos justo al principio del cuerpo de una función se denominan variables locales.

```
#include  <stdio.h>
int  suma(int  a, int  b) {
      int   i,s;
      s=0;
      for  (i=a; i<=b;i++)
            s += i;
      return s;
}
int main(){
   int   i;
   for (i=1; i<=10; i++)
      cout<<“Suma de los “<<i<<“ primeros naturales:
      ”<<suma(1,i));
    return 0;
}
```

Ámbito de las variables

Variables globales

Las variables globales se declaran fuera del cuerpo de cualquier función y son accesibles desde cualquier punto del programa posterior a su declaración. ¿Cuál es la salida del siguiente programa?

```
#include  <stdio.h>
int   j=1;
int  doble(void)
      { j *= 2;
      return   j ;
}
int main(){
     int   j;
     for (j=0; j<5; j++)
            cout<<doble()<<“\n”;
     return 0;
}
```

Prototipos de funciones

Un prototipo de función es una declaración de una función que incluye el tipo de retorno y los tipos de sus parámetros. No es obligatorio pero si aconsejable.

Ejemplo:

```
double   fuerza (double t);
doble    Aceleracion(double  t, double mu, double m);
```

•Un prototipo de función no contiene ninguna definición de función.

• Permite la comprobación de errores entre las llamadas a una función y la definición de la función correspondiente.

Prototipos de funciones

Si la función se llama antes que sea definida, se asume que el tipo de retorno de la función es **int**.

Por lo tanto, se requiere un prototipo de función si el tipo de retorno de la función no es **int** y la función se llama antes de procesar su definición, o si la definición de la función se encuentra en un archivo diferente o en una biblioteca.

Paso de parámetros

En el caso de que un parámetro sea una variable (no una expresión de variables y/o constantes), al volver al contexto de la función desde la que se llamó pueden ocurrir dos casos:

•Que el valor de la variable no se haya modificado en la función llamada.

•Que el valor de la variable se haya modificado en la función llamada.

Paso de parámetros por Valor

En las llamadas por valor se hace una copia del valor del argumento en el parámetro formal.

La función opera internamente con estos últimos.

Como las variables locales a una función (y los parámetros formales lo son) se crean al entrar a la función y se destruyen al salir de ella, cualquier cambio realizado por la función en los parámetros formales no tiene ningún efecto sobre los argumentos.

Ejemplo

Programa que calcula el máximo de dos números:

```
int  maximo(int  a, int b)
      { int max;
      if (a>b)
          max = a;
      else
          max = b;
      return max;
}
```

```
int main(){
     int x=2,y=4,mayor;
     mayor = maximo( x , 2*y);
     return 0;
}
```

Ejercicio

Elabore un programa que utilice una función para calcular la potencia: x elevado a la y (con x un real, y un entero).

```
…
int main(){
    int x, y ,p;
    …
    p = potencia( x , y);
    …
    return 0;
}
```

Ejercicio

Elabore un programa que invoque a una función entera CP que indica si un número es cuadrado perfecto.

```
…
void main(){
    int  n, perfecto;
    …
    cout<<n;
    perfecto = CP( n);
    …
    if (perfecto)
        …
    return 0;
}
```

Paso de parámetros por Referencia

Hasta ahora las funciones sólo devolvían un valor, pero…¿qué pasa si tienen que devolver más?

En este caso, se requiere pasar parámetros por referencia.

Declaración

```
void nombreFuncion(tipo e1,…,tipo eN,tipo *s1, …, tipo *sN)
{
    //cuerpo de la función
}
```

Llamada a la función

```
nombreFuncion(e1,…,eN, &s1, …, &sN);
```

Paso de parámetros por Referencia

En este tipo de llamadas los argumentos contienen direcciones de variables.

Dentro de la función, la dirección se utiliza para acceder al argumento actual.

En las llamadas por referencia, cualquier cambio en la función tiene efecto sobre la variable cuya dirección se pasó en el argumento. No hay un proceso de creación destrucción de esa dirección.

Mediante **&** podemos pasar direcciones de variables en lugar de valores y trabajar internamente en la función con los contenidos mediante el operador *****.

Ejercicio

Elabore un programa que utilice una función Cambiar para intercambiar el valor de dos variables.

Ejercicio

Elabore un programa que calcule la hipotenusa de un triángulo rectángulo.

Consideraciones

1. Utilice una función **Lectura** para leer los valores de a y b.
2. Defina una función hipotenusa a la que se le pasará tres parámetros.
3. Imprima el valor de h.

Recursividad

Una función es recursiva si se puede llamar a sí misma.

Ejemplo:

Calcule n! recursivamente, usando:

n! = n x (n-1)!

Seguimiento de la función factorial

Para ver como trabaja la recursión, analicemos la traza de la ejecución de la sentencia:

i = fact(3);

Aquí se detalla lo que sucede:

fact(3)	determina que 3 no es menor que 1, por lo que llama a
fact(2)	determina que 2 no es menor que 1, por lo que llama a
fact(1)	como 1 es menor o igual que 1, retorna 1, causando
fact(2)	retorna 2x1 = 2, lo que causa
fact(3)	retorna 3x2 = 6

Ejemplo de Recursividad

Ejemplo:

Escriba una función que calcule la potencia,

$$P = x^n \qquad x \in \mathbb{R}, n \in \mathbb{Z}$$

usando:

$$x^n = x \times x^{n-1}$$

Seguimiento de la función potencia

La llamada a la función potencia(5,3) puede ser ejecutada como sigue:

potencia(5,3) determina que 3 no es igual a 0, por lo que llama a
potencia(5,2) determina que 2 no es igual a 0, por lo que llama a
potencia(5,1) determina que 1 no es igual a 0, por lo que llama a
potencia(5,0) como 0 es igual que 0, retorna 1, causando
potencia(5,1) retorna 5x1 = 5, lo que causa
potencia(5,2) retorna 5x5 = 25, lo que causa
potencia(5,3) retorna 5x25 = 125.

Macros

La sentencia #define se puede utilizar para definir macros.

Una macro es un identificador equivalente a una expresión, sentencia o grupo de sentencias.

Una macro no es una función. El preprocesador sustituye todas las referencias a la macro que aparezcan dentro de un programa antes de realizar la compilación.

Ejemplo:

Escriba un programa que calcule el máximo de dos números.

```
#define maximo(x , y)   ((x>y) ? x : y )
```

Función rand

Retorna un número pseudoaleatorio entre 0 y RAND_MAX.

¿Qué salida produce el siguiente segmento de código:?

```
#include  <stdlib.h>
…
   numero = rand() % 11;
   numero = rand() % (N+1);
…
}
```

Rango de inicio distinto de cero

¿Qué salida produce el siguiente segmento de código:?

```
#include  <stdlib.h>
…
   numero = rand() % 11 + 20 ;
   numero = rand() % (N-M+1) + M;
…
}
```

La función srand

Si ejecutamos varias veces nuestro programa, la secuencia de números aleatorios se repite.

Para evitar este problema usaremos la función srand al que le pasaremos como parámetro un número inicial.

Hay dos números que se utilizan habitualmente para ello. Podemos usar indistintamente:

La fecha/hora del sistema:

```
srand(time(NULL));
```

El número de proceso del programa:

```
srand(getpid());
```

GRACIAS!!