Informática I para Bachillerato

C/C++

Funciones

José Luis Alonzo Velázquez

CIMAT

Sesión 11

¿Que es una función?

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.

Sintaxis

```
<tipo> [clase::] <nombre> ( [Parámetros] )
{
    cuerpo;
}
```

Sintaxis

```
<tipo> [clase::] <nombre> ( [Parámetros] )
{
    cuerpo;
}
```

Ejemplo

```
// regresar el cuadrado de un número
double cuadrado(double n)
{
    return n*n;
}
```

Parámetros por valor

La función cuadrado() es un clásico ejemplo que muestra el paso de parámetros por valor, en ese sentido la función cuadrado() recibe una copia del parámetro n. En la misma función se puede observar que se realiza un calculo ($n * n$), sin embargo el parámetro original no sufrirá cambio alguno, esto seguirá siendo cierto aún cuando dentro de la función hubiera una instrucción parecida a $n = n * n$; o $n* = n$;.

Parámetros por valor

La función cuadrado() es un clásico ejemplo que muestra el paso de parámetros por valor, en ese sentido la función cuadrado() recibe una copia del parámetro n. En la misma función se puede observar que se realiza un calculo ($n * n$), sin embargo el parámetro original no sufrirá cambio alguno, esto seguirá siendo cierto aún cuando dentro de la función hubiera una instrucción parecida a $n = n * n$; o $n* = n$;.

Ejemplo

```
1 // regresar el cuadrado de un número
2 double cuadrado(double n)
3 {
4     return n*n;
5 }
```

Ejemplo

```
1 // regresar el cuadrado de un número
2 double cuadrado2(double &n)
3 {
4     n *= n;
5     return n;
6 }
```

Ejemplo

```
1 // regresar el cuadrado de un número
2 double cuadrado2(double &n)
3 {
4     n *= n;
5     return n;
6 }
```

Parámetros por referencia

La función cuadrado2() es un clásico ejemplo que muestra el paso de parámetros por referencia, en ese sentido la función cuadrado2() recibe el parámetro n. En la misma función se puede observar que se realiza un calculo ($n * n$), sin embargo el parámetro original sufrirá cambio, esto seguirá siendo cierto aún cuando dentro de la función hubiera una instrucción parecida a $n = n * n$; o $n* = n$;.

Parámetros constantes

Los parámetros usados por una función pueden declararse como constantes (const) al momento de la declaración de la función. Un parámetro que ha sido declarado como constante significa que la función no podrá cambiar el valor del mismo (sin importar si dicho parámetro se recibe por valor o por referencia).

Parámetros constantes

Los parámetros usados por una función pueden declararse como constantes (const) al momento de la declaración de la función. Un parámetro que ha sido declarado como constante significa que la función no podrá cambiar el valor del mismo (sin importar si dicho parámetro se recibe por valor o por referencia).

Ejemplo

```
1 int funcionX( const int n );
2 void printstr( const char *str );
```

Parámetros con valor por defecto

Los parámetros usados por una función pueden declararse con un valor por defecto. Un parámetro que ha sido declarado con valor por defecto es opcional a la hora de hacer la llamada a la función.

Parámetros con valor por defecto

Los parámetros usados por una función pueden declararse con un valor por defecto. Un parámetro que ha sido declarado con valor por defecto es opcional a la hora de hacer la llamada a la función.

Ejemplo

```
1 void saludo( char* mensaje = "Hola sudafrica 2010" );
2
3 la misma puede ser invocada como:
4
5 saludo(); // sin parámetro
6 saludo("Sea usted bienvenido a C++"); // con parámetro
```

Funciones sobrecargadas

C++, a diferencia del C estándar, permite declarar funciones con el mismo nombre y a esto se conoce como **sobrecarga de funciones**. Las funciones sobrecargadas pueden coincidir en tipo, pero al menos uno de sus parámetros tiene que ser diferente. En todo caso, si usted trata de declarar funciones sobrecargadas que coincidan en tipo y número de parámetros el compilador no se lo permitirá. Para poner un ejemplo vamos a considerar el caso de dos funciones cuyo nombre será divide, ambas regresarán el cociente de dos números, salvo que una de ellas operará sobre números enteros y la otra lo hará sobre números reales (de punto flotante).

Observación

Nota: cuando en los programas se hace una llamada a una función sobrecargada, el compilador determina a cual de las funciones invocar en base al tipo y número de parámetros pasados a la función.

Ejemplo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  using namespace std;
5  // divide enteros
6  int divide(int a, int b){
7      printf("división entera");
8      if(b!=0){
9          return a/b;
10     }else{
11         return 0;
12     }
13 }
```

Ejemplo

```
1  // divide reales
2  double divide(double a, double b){
3      printf("división entera");
4      if(b!=0){
5          return a/b;
6      }else{
7          return 0;
8      }
9  }
10 // punto de prueba
11 int main(){
12     printf("%d",divide(10, 3));
13     printf("%f",divide(10.0, 3.0));
14 }
```

Número variable de parámetros

En C,C++ se pueden crear funciones que operen sobre una lista variable de parámetros, es decir, en donde el número de parámetros es indeterminado. En esta sección se mostrará un ejemplo de la manera en que podemos crear funciones para manejar tales asuntos, y para ello haremos uso de tres macros soportadas por C++:

1. **va_list** puntero de argumentos
2. **va_start** inicializar puntero de argumentos
3. **va_end** liberar puntero de argumentos

Número variable de parámetros

La sintaxis que usaremos para declarar funciones con lista de parámetros variables es:

```
1) tipo nombrefuncion(...)
2) tipo nombrefuncion(int num, ...)
```

Ejemplo

donde:

1. tipo es el tipo regresado por la función
2. nombrefuncion es el nombre de la función
3. int num es el número de parámetros que la función procesará
4. ... esta notación se emplea para indicar que el número de parámetros es variable

Nota: observe que la primera forma de declaración es realmente variable el número de parámetros a procesar y en estos casos se debe establecer el mecanismo para determinar cuando se ha procesado el último de los argumentos, en el segundo tipo de declaración el número total de parámetros a procesar es igual al valor del parámetro num.

Ejemplo

```
1  #include <stdio.h>
2  #include <stdarg.h>
3  using namespace std;
4  // Esta función opera sobre una lista variable de
       números enteros
5  int suma( int num, ... ){
6      int total = 0;
7      va_list argptr;
8      va_start( argptr, num );
9      while( num > 0 ){
10         total += va_arg( argptr, int );
11         num--;
12     }
13     va_end( argptr );
14     return( total );
15 }
```

Continuación Ejemplo

```
1 int main(){
2     printf("%d",suma(4, 100, 200, 300, 400));
3     return 0;
4 }
```

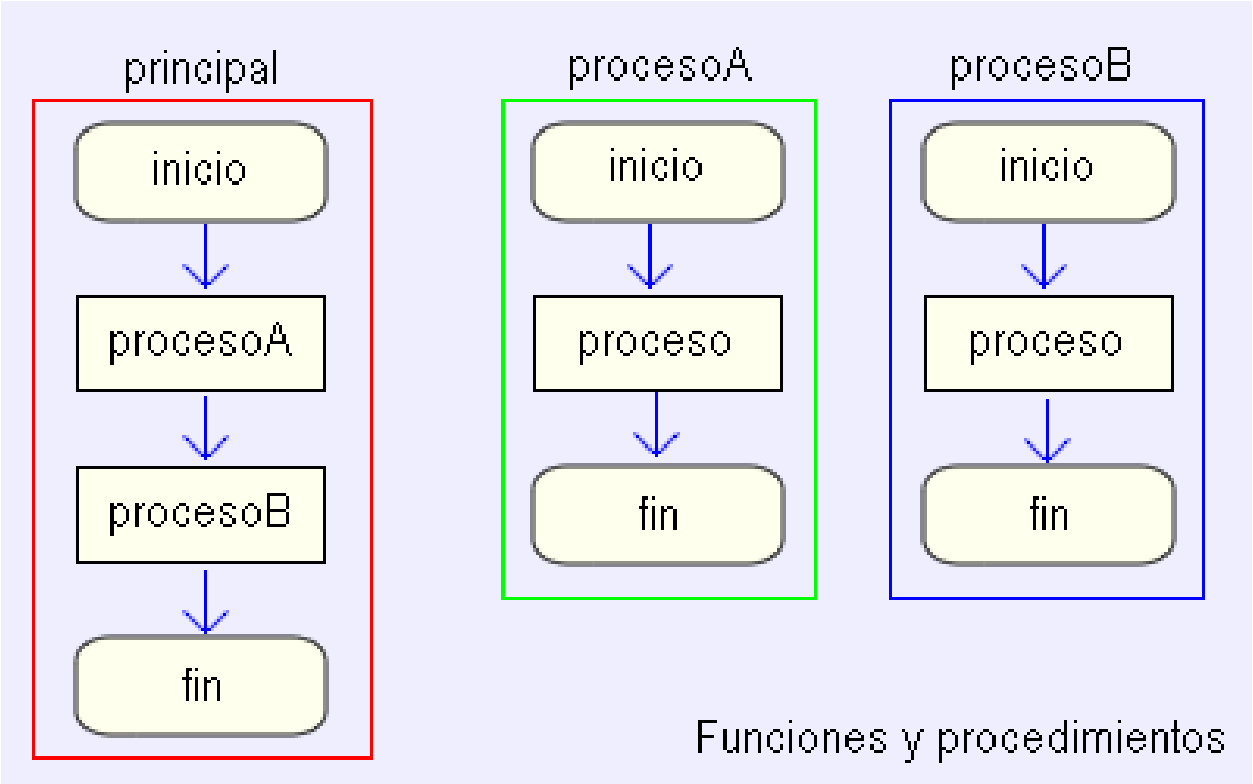

Figura : Repaso de funciones

Problema para clase

Hacer un programa que tenga una función que multiplique 4 números dados por el usuario, y imprima el resultado en pantalla.

Declaración de funciones

Antes de escribir una función es necesario informarle al Compilador los tamaños de los valores que se le enviarán en el stack y el tamaño de los valores que ella retornará al programa invocante . Estas informaciones están contenidas en la **declaración** del **prototipo** de la **función**. Formalmente dicha declaración queda dada por :

```
1 tipo del valor de retorno   nombre_de_la_función(lista
      de tipos de parámetros)
```

Ejemplo

```
1 float mi_funcion(int i, double j);
2
3 double otra_funcion(void);
4
5 otra_mas(long p);
6
7 void la_ultima(long double z,char y,int x,unsigned
    long w);
```

Observaciones

El primer término del prototipo da, como hemos visto el tipo del dato retornado por la función; en caso de obviarse el mismo se toma, por omisión, el **tipo int**. Sin embargo, aunque la función devuelva este tipo de dato, para evitar malas interpretaciones es conveniente explicitarlo .

Observaciones

El primer término del prototipo da, como hemos visto el tipo del dato retornado por la función; en caso de obviarse el mismo se toma, por omisión, el **tipo int**. Sin embargo, aunque la función devuelva este tipo de dato, para evitar malas interpretaciones es conveniente explicitarlo .

Ya que el "default" del tipo de retorno es el **int**, debemos indicar cuando la función NO retorna nada, esto se realiza por medio de la palabra **VOID** (sin valor). De la misma manera se actúa, cuando no se le enviarán argumentos. Más adelante se profundizará sobre el tema de los argumentos y sus características.

Observaciones

La declaración debe anteceder en el programa a la definición de la función. Es normal, por razones de legibilidad de la documentación, encontrar todas las declaraciones de las funciones usadas en el programa, en el **HEADER** del mismo, junto con los include de los archivos *.h que tienen los prototipos de las funciones de Librería.

Observaciones

La declaración debe anteceder en el programa a la definición de la función. Es normal, por razones de legibilidad de la documentación, encontrar todas las declaraciones de las funciones usadas en el programa, en el **HEADER** del mismo, junto con los include de los archivos *.h que tienen los prototipos de las funciones de Librería.

Si una ó más de nuestras funciones es usada habitualmente, podemos disponer su prototipo en un archivo de texto, e incluirlo las veces que necesitemos, según se vera más adelante.

Definición de las funciones

La definición de una función puede ubicarse en cualquier lugar del programa, con sólo dos restricciones: debe hallarse luego de dar su prototipo, y no puede estar dentro de la definición de otra función (incluida main()). Es decir que a diferencia de Pascal, en C las definiciones no pueden anidarse.

Definición de las funciones

La definición de una función puede ubicarse en cualquier lugar del programa, con sólo dos restricciones: debe hallarse luego de dar su prototipo, y no puede estar dentro de la definición de otra función (incluida main()). Es decir que a diferencia de Pascal, en C las definiciones no pueden anidarse.

NOTA

No confundir definición con llamada; una función puede llamar a tantas otras como desee. La definición debe comenzar con un encabezamiento, que debe coincidir totalmente con el prototipo declarado para la misma, y a continuación del mismo, encerradas por llaves se escribirán las sentencias que la compone.

```
#include <stdio.h>
/* Declaración observe que termina en  ";"  */
float mi_funcion(int i, double j );
int main(){
float k ;
int p ;
double z ;
.....
k = mi_funcion( p, z ); /* LLAMADA a la función */
.....
}
/* fin de la función main() */
/* Definición  observe que NO lleva  ";"  */
```

```
1 float mi_funcion(int i, double j ){
2 float n;
3 ......................
4 printf("%d", i ); /* LLAMADA a otra función */
5 ......................
6 return ( 2 * n ); /* RETORNO devolviendo un valor float
      */
7 }
```

Problema para clase

Hacer una función que manda llamar otra función que a su ves manda llamar otra función.

Problema extra clase

Utilizar un archivo header y ver como funciona para poder usarlo en varios proyectos.

 Como Programar en C/C++, Deitel (Prentice Hall), 2da Edición.

 Programming Principles and Practice Using C++, Bjarne Stroustrup.

 http://www.codeblocks.org

 http://www.wxwidgets.org