FUNDAMENTOS DE PROGRAMACIÓN

Inicio Clase 07

Profesor: Carlos Díaz

Clase 07: Funciones

- Prototipos de función
- Llamada a una función
- Definición de una función
- Funciones con lista de parámetros vacíos
- Sobrecarga de funciones
- Variables locales y globales
- Parámetros por valor y por referencia

Ejercicio 1

Función que calcula el máximo de 2 números.

Usamos una función que devuelve un valor entero.

```
#include <iostream>
using namespace std;

int encontrarMax(int, int); // el prototipo de la función

int main()
{
  int primernum, segundonum, max;


  cout << "\nIntroduzca un numero: ";
  cin >> primernum;
  cout << "¡Estupendo! Por favor introduzca un segundo número: ";
  cin >> segundonum;

  max = encontrarMax(primernum, segundonum); // aquí se llama a la función

  cout << "\nEl máximo de los dos números es " << max << endl;

  return 0;
}

int encontrarMax(int x, int y)
{                       // inicio del cuerpo de la función
  int numMax;           // declaración de variable

  if (x >= y)           // encontrar el número máximo
    numMax = x;
  else
    numMax = y;

  return numMax;        // instrucción de devolución
}
```

Ejercicio 2

Función que calcula el máximo de 2 números.

Usamos una función que no devuelve valor.

```
#include <iostream>
using namespace std;

void encontrarMax(int, int); // el prototipo de la función

int main()
{
  int primernum, segundonum;

  cout << "\nIntroduzca un número: ";
  cin >> primernum;
  ccout << "¡Estupendo! Por favor introduzca un segundo numero: ";
  cin >> segundonum;

  encontrarMax(primernum, segundonum); // aquí se llama a la función

  return 0;
}

// en seguida está la función encontrarMax()

void encontrarMax(int x, int y)
{                    // inicio del cuerpo de función
  int numMax;        // declaración de variable

  if (x >= y)        // encontrar el número máximo
    numMax = x;
  else
    numMax = y;

  cout << "\nEl máximo de los dos números es "
       << numMax << endl;

  return;
}  // fin del cuerpo de función y fin de la función
```

Ejercicio 3

Función que convierte grados Fahrenheit a grados Celsius.

Realiza 4 conversiones.

```
#include <iostream>
using namespace std;

double convertir_temp(double); // prototipo de la función

int main()
{
  const CONVERSIONES = 4; // numero de conversiones que se harán
  int cuenta;
  double fahren;

  for(cuenta = 1; cuenta <= CONVERSIONES; cuenta++)
  {
    cout << "\nIntroduzca una temperatura en grados Fahrenheit: ";
    cin  >> fahren;
    cout << "El equivalente en grados Celsius es "
         << convertir_temp(fahren) << endl;
  }

  return 0;
}

// convertir Fahrenheit a Celsius
double convertir_temp(double in_temp)
{
  return (5.0/9.0) * (in_temp - 32.0);
}
```

Funciones con lista de parámetros vacíos

Aunque las funciones útiles que tienen una lista de parámetros vacía son limitadas en extremo , pueden ocurrir. El prototipo de función para dicha función requiere escribir la palabra clave void o no poner nada en absoluto entre los paréntesis que siguen al nombre de la función. Por ejemplo, ambos prototipos

```
int despliegue();
```

o

```
int despliegue(void);
```

indican que la función `despliegue()` no tiene parámetros y devuelve un número entero. Una función con una lista de parámetros vacía es llamada por su nombre sin nada escrito dentro del paréntesis requerido después del nombre de la función. Por ejemplo, la instrucción `despliegue();` llama en forma correcta a la función `despliegue()` cuyo prototipo se proporcionó antes.

Ejercicio 4

El volumen, v, de un cilindro está dado por la fórmula

$$v = \pi r^2 l$$

donde r es el radio del cilindro y l es su largo. Usando esta fórmula, escriba una función C++ nombrada `vol_cil()` que acepte el radio y el largo de un cilindro y devuelva su volumen.

Ejercicio 5

Escriba una función en C++ llamada `entero()` que devuelva la parte entera de cualquier número que se transmita a la función. (*Sugerencia:* Asigne el argumento transmitido a una variable entera.)

Ejercicio 6

Un algoritmo de programación útil en extremo para redondear un número real a n lugares decimales es

Paso 1: Multiplicar el número por 10^n

Paso 2: Sumar 0.5

Paso 3: Eliminar la parte fraccionaria del resultado

Paso 4: Dividir entre 10^n

Por ejemplo, usar este algoritmo para redondear el número 78.374625 a tres lugares decimales produce:

Paso 1: $78.374625 \times 10^3 = 78374.625$

Paso 2: $78374.625 + 0.5 = 78375.125$

Paso 3: Conservar la parte entera = 78375

Paso 4: 78375 dividido entre $10^3 = 78.375$

Usando este algoritmo, escriba una función en C++ que acepte un valor introducido por un usuario y devuelva el resultado redondeado a dos lugares decimales.

Ejercicio 7

Escriba una función en C++ nombrada `partefrac()` que devuelva la parte fraccionaria de cualquier número transmitido a la función. Por ejemplo, si se transmite el número 256.879 a `partefrac()`, debería devolverse el número 0.879. Haga que la función `partefrac()` llame a la función `entero()` que escribió en el ejercicio 5 . El número devuelto puede determinarse entonces como el número transmitido a `partefrac()` menos el valor devuelto cuando el mismo argumento es transmitido a `entero()`. El programa completo deberá consistir de `main()` seguido por `partefrac()` seguido por `entero()`.

Ejercicio 8

Todos los años que son divisibles entre 400 o son divisibles entre cuatro y no son divisibles entre 100 son años bisiestos. Por ejemplo, en vista que 1600 es divisible entre 400, el año 1600 fue un año bisiesto. Del mismo modo, en vista que 1988 es divisible entre cuatro pero no entre 100, el año 1988 también fue un año bisiesto. Usando esta información, escriba una función en C++ que acepte el año como entrada de un usuario y devuelva un uno si el año transmitido es un año bisiesto o un cero si no lo es.

Sobrecarga de funciones

C++ proporciona la capacidad de usar el mismo nombre de función para más de una función, lo cual se conoce como **sobrecarga de función**. El único requisito para crear más de una función con el mismo nombre es que el compilador debe ser capaz de determinar cuál función usar con base en los tipos de datos de los parámetros (no los tipos de datos del valor devuelto, si es que hay alguno). Por ejemplo, considere las tres funciones siguientes, todas nombradas `cdabs()`.

```
void cdabs(int x) // calcula y despliega el valor absoluto de un número entero
{
  if ( x < 0 )
    x = -x;
  cout << "El valor absoluto del número entero es " << x << endl;
}

void cdabs(float x) // calcula y despliega el valor absoluto de un número de
punto flotante
{
  if ( x < 0 )
    x = -x;
  cout << "El valor absoluto del número de punto flotante es " << x << endl;
}

void cdabs(double x) // calcula y despliega el valor absoluto de un número
en doble precision
{
  if ( x < 0 )
   x = -x;
  cout << "El valor absoluto del número de doble precisión es " << x << endl;
}
```

Ejercicio 4

Sobrecarga de la función área, para calcular el área de un rectángulo y circulo.

```
#include <iostream>
using namespace std;
double area(double, double);
double area(double);
int main()
{
    double radio, base, altura;
    cout<<"Calculo de areas"<<endl;
    cout<<"Escriba el radio ";
    cin>>radio;
    cout<<"El area del circulo es "<<area(radio)<<endl;
    cout<<"Escriba la base ";
    cin>>base;
    cout<<"Escriba la altura ";
    cin>>altura;
    cout<<"El area del triangulo es "<<area(base,altura)<<endl;
    return 0;
}
double area(double b, double h)
{
    return b*h/2;
}

double area(double r)
{
    return 3.14*r*r;
}
```

Variables locales y globales

Una variable con un **alcance local** es aquella a la que una instrucción de declaración hecha dentro del cuerpo de una función le ha designado ubicaciones de almacenamiento. Las variables locales sólo son significativas cuando se usan en expresiones o instrucciones dentro de la función que las declaró. Esto significa que el mismo nombre de variable puede declararse y usarse en más de una función. Para cada función que declara la variable, se crea una variable separada y distinta.

Una variable con **alcance global**, por lo general denominada **variable global**, el almacenamiento se crea mediante una instrucción de declaración localizada fuera de cualquier función. Estas variables pueden ser utilizadas por todas las funciones que se colocan físicamente después de la declaración de la variable global.

Ejercicio 5

Utiliza los mismos nombres de variable en las funciones main y valfun.

```
#include <iostream>
using namespace std;

int primernum; // crea una variable global llamada primernum

void valfun(); // prototipo de la función (declaración)

int main()
{
  int segundonum;          // crea una variable local llamada segundonum

  primernum = 10; // almacena un valor en la variable global
  segundonum = 20; // almacena un valor en la variable local

  cout << "De main(): primernum = " << primernum << endl;
  cout << "De main(): segundonum = " << segundonum << endl;

  valfun(); // llama a la función valfun

  cout << "\nDe main() de nuevo: primernum = " << primernum << endl;
  cout << "De main() de nuevo: segundonum = " << segundonum << endl;

  return 0;
}

void valfun() // no se transmiten valores a esta función
{
  int segundonum; // crea una segunda variable local llamada segundonum

  segundonum = 30; // esto sólo afecta al valor de esta variable local

  cout << "\nDe valfun(): primernum = " << primernum << endl;
  cout << "De valfun(): segundonum = " << segundonum << endl;

  primernum = 40; // esto cambia primernum para ambas funciones

  return;
}
```

Parámetros por valor y por referencia

- Hasta ahora, la forma en que hemos declarado y pasado los parámetros de las funciones se conoce como "**por valor**". Esto quiere decir que cuando el control pasa a la función, los valores de los parámetros en la llamada se copian a "objetos" locales de la función.

```
#include <iostream>
using namespace std;
int funcion(int n, int m);
int main() {
   int a, b;
   a = 10;
   b = 20;
   cout << "a,b =" << a << ", " << b << endl;
   cout << "funcion(a,b) =" << funcion(a, b) << endl;
   cout << "a,b =" << a << ", " << b << endl;
   cout << "funcion(10,20) ="   << funcion(10, 20) << endl;
   return 0;
}
int funcion(int n, int m) {
   n = n + 1;
   m = m +2;
   return n+m;
}
```

Parámetros por valor y por referencia

- Si queremos que los cambios realizados en los parámetros dentro de la función se conserven al retornar de la llamada, deberemos pasarlos "**por referencia**". Esto se hace declarando los parámetros de la función como referencias a objetos.

```
#include <iostream>
using namespace std;
int funcion(int &n, int &m);
int main() {
   int a, b;
   a = 10;
   b = 20;
   cout << "a,b =" << a << ", " << b << endl;
   cout << "funcion(a,b) =" << funcion(a, b) << endl;
   cout << "a,b =" << a << ", " << b << endl;
   cout << "funcion(10,20) ="   << funcion(10, 20) << endl; //Es ilegal pasar constantes
   return 0;
}
int funcion(int &n, int &m) {
   n = n + 1;
   m = m +2;
   return n+m;
}
```

FUNDAMENTOS DE PROGRAMACIÓN

Fin Clase 07

Profesor: Carlos Díaz