Funciones

Alvaro Henry Mamani-Aliaga

Escuela Profesional de Ciencia de la Computación
Universidad Nacional de San Agustín
amamaniali@unsa.edu.pe

4 de mayo de 2020

Contenido

Introducción a funciones

- Implemente un programa que calcule el resultado elevar 3 a la potencia 4.

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

int main() {
    int tresExpCuatro = 1;
    for (int i = 0; i < 4; i++) {
        tresExpCuatro *= 3;
    }
    cout << "3^4 is " << tresExpCuatro << endl;
    return 0;
}
```

Introducción a funciones: Ctrol + c ... Ctrol + v

- Implemente un programa que calcule el resultado elevar 3 a la potencia 4.
- Posteriormente calcule el resultado de elevar 5 a la potencia 6.

```
using namespace std;

int main() {
    int tresExpCuatro = 1;
    for (int i = 0; i < 4; i++) {
        tresExpCuatro *= 3;
    }
    cout << "3^4 is " << tresExpCuatro << endl;

    int seisExpCinco = 1;
    for (int i = 0; i < 5; i++) {
        seisExpCinco *= 6;
    }
    cout << "6^5 is " << seisExpCinco << endl;

    return 0;
}
```

Introducción a funciones: Ctrol + c ... Ctrol + v (*Bad programming*)

- Implemente un programa que calcule el resultado elevar 3 a la potencia 4.
- Posteriormente calcule el resultado de elevar 5 a la potencia 6.
- Posteriormente calcule el resultado de elevar 12 a la potencia 10.

```
using namespace std;

int main() {
    int tresExpCuatro = 1;
    for (int i = 0; i < 4; i++) {
        tresExpCuatro *= 3;
    }
    cout << "3^4 is " << tresExpCuatro << endl;

    int seisExpCinco = 1;
    for (int i = 0; i < 5; i++) {
        seisExpCinco *= 6;
    }
    cout << "6^5 is " << seisExpCinco << endl;

    int doceExpDiez = 1;
    for (int i = 0; i < 10; i++) {
        doceExpDiez *= 12;
    }
    cout << "12^10 is " << doceExpDiez << endl;

    return 0;
}
```

Con una función

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

int main() {
    int tresExpCuatro = potencia(3, 4);
    cout << "3^4 is " << tresExpCuatro << endl;

    return 0;
}
```

Con una función

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

int main() {
    int tresExpCuatro = potencia(3, 4);
    cout << "3^4 is " << tresExpCuatro << endl;

    int seisExpCinco = potencia(6, 5);
    cout << "6^5 is " << seisExpCinco << endl;

    return 0;
}
```

Con una función

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

int main() {
    int tresExpCuatro = potencia(3, 4);
    cout << "3^4 is " << tresExpCuatro << endl;

    int seisExpCinco = potencia(6, 5);
    cout << "6^5 is " << seisExpCinco << endl;

    int doceExpDiez = potencia(12, 10);
    cout << "12^10 is " << doceExpDiez << endl;

    return 0;
}
```

¿Porqué definimos funciones

- **Legibilidad**: *potencia(3, 4)* es mucho mas claro que varias lineas de código calculando 3 potencia de 4.
- **Mantenimiento del código**: si cambia el comportamiento del algoritmo, solo es necesario cambiar la función.
- **Reusabilidad**: permitir que otros programadores utilicen la implementación.

Declaración de una función: Sintaxis

```
1   //
2   //  NOMBRE DE LA FUNCION
3   //          |
4   //          v
5   int     potencia        (int base, int exponente)
6   {
7       int result = 1;
8       for (int i = 0; i < exponente; i++) {
9           result = result * base;
10      }
11      return result;
12  }
```

Declaración de una función: Sintaxis

```
1   //
2   //  TIPO DE RETORNO
3   //       |
4   //       v
5           int     potencia        (int base, int exponente)
6           {
7               int result = 1;
8               for (int i = 0; i < exponente; i++) {
9                   result = result * base;
10              }
11              return result;
12          }
```

Declaración de una función: Sintaxis

```
1   //
2   //            Argumento 1
3   //                 |
4   //                 v
5   //              .________.
6   int potencia (int base, int exponente)
7   {
8       int result = 1;
9       for (int i = 0; i < exponente; i++) {
10          result = result * base;
11      }
12      return result;
13  }
```

Declaración de una función: Sintaxis

```
1   //
2   //                              Argumento 2
3   //                                   |
4   //                                   v
5   //                              ._________.
6   int potencia (int base, int exponente)
7   {
8       int result = 1;
9       for (int i = 0; i < exponente; i = i + 1) {
10          result = result * base;
11      }
12      return result;
13  }
```

- potencia(2, 3) => 2 al cubo
- potencia(3, 2) => 3 al cuadrado

Declaración de una función: Sintaxis

```
1   int potencia (int base, int exponente)     // <- FIRMA DE LA FUNCION
2   {
3       int result = 1;
4       for (int i = 0; i < exponente; i = i + 1) {
5           result = result * base;
6       }
7       return result;
8   }
```

Firma de la función

```
int potencia (int base, int exponente)
```

Declaración de una función: Sintaxis

```
1   int potencia (int base, int exponente)
2   {                                                    //          |
3       int result = 1;                                  //          |
4       for (int i = 0; i < exponente; i = i + 1) {      //          |
5           result = result * base;                      //    <---  |   CUERPO DE LA FUNCION
6       }                                                //          |
7       return result;                                   //          |
8   }                                                    //          |
```

Cuerpo de la función

```
{
    int result = 1;
    for (int i = 0; i < exponente; i = i + 1) {
        result = result * base;
    }
    return result;
}
```

Declaración de una función: Sintaxis

```
1   int potencia (int base, int exponente)
2   {
3       int result = 1;
4       for (int i = 0; i < exponente; i = i + 1) {
5           result = result * base;
6       }
7       return result;      // <- SENTENCIA RETURN - VALOR DE RETORNO
8   }
```

Declaración de una función: Sintaxis

```
1   #include <iostream>
2   using namespace std;
3
4   int potencia (int base, int exponente)                  //          |
5   {                                                       //          |
6       int result = 1;                                     //          |
7       for (int i = 0; i < exponente; i = i + 1) {         //          |
8           result = result * base;                         //     <--- |   DECLARACION DE LA FUNCION
9       }                                                   //          |
10      return result;                                      //          |
11  }                                                       //          |
12
13  int main() {
14      //
15      //                INVOCACION DE LA FUNCION
16      //                          |
17      //                          V
18      int tresExpCuatro = potencia(3, 4);
19      cout << "3^4 is " << tresExpCuatro << endl;
20
21      return 0;
22  }
```

Valor de retorno

- El **tipo de retorno** debe ser del mismo que el valor que esta siendo retornado.

```
1   //
2   //  TIPO DE RETORNO
3   //       |
4   //       v
5           int potencia (int base, int exponente)
6           {
7               int result = 1;
8               for (int i = 0; i < exponente; i = i + 1) {
9                   result = result * base;
10              }
11              return result;      // <-  SENTENCIA RETURN - VALOR DE RETORNO
12          }
```

Valor de retorno

- El **tipo de retorno** debe ser del mismo que el valor que esta siendo retornado.

```
1   int f()
2   {
3       return "hello";      // error
4   }
```

```
1   string f()
2   {
3    return "hola"; // correcto
4   }
```

Valor de retorno

- El **tipo de retorno** debe ser del mismo que el valor que esta siendo retornado.
- Si no hay valor para retornar, el **TIPO DE RETORNO** de la función debe ser **void**

```
1   void imprimirNumero(int num)
2       cout << "el numero es " << num << endl;
3   }
4   int main() {
5       int x = 4;
6       imprimirNumero(x); // el numero es 4
7       return 0;
8   }
```

Valor de retorno

- El **tipo de retorno** debe ser del mismo que el valor que esta siendo retornado.
- Si no hay valor para retornar, el **TIPO DE RETORNO** de la función debe ser **void**
  - OJO, el tipo **void** no se usa para declarar variables.

```
1    int main() {
2        void x; // ERROR
3        return 0;
4    }
```

Valor de retorno

- El **tipo de retorno** debe ser del mismo que el valor que esta siendo retornado.
- Si no hay valor para retornar, el **TIPO DE RETORNO** de la función debe ser **void**
  - OJO, el tipo **void** no se usa para declarar variables.
- La sentencia **return** no necesariamente requiere estar al final
- Las funciones terminan apenas se ejecute la sentencia **return**.

```
1   void imprimirNumeroSiEsPar(int num) {
2       if (num %2 == 1) {
3           cout << "Numero impar" << endl;
4           return;
5       }
6       cout << "Numero par; el numero es " << num << endl;
7   }
8   int main() {
9       int x = 4;
10      imprimirNumeroSiEsPar(x);
11      // Numero par; el numero es 4
12      int y = 5;
13      imprimirNumeroSiEsPar(y);
14      // Numero impar
15
16      return 0;
17  }
```

Tipo de dato del argumento

```
1    void imprimir(int x)
2    {
3        cout << x << endl;
4    }
```

- *imprimir(3)* está correcto.
- *imprimir(“hola”)* no compilará.

Tipo de dato del argumento

```
1   void imprimir(string x)
2   {
3       cout << x << endl;
4   }
```

- *imprimir(3)* no compilará.
- *imprimir(“hola”)* está correcto.

Tipo de dato del argumento

```
1   void imprimir(int x)
2   {
3       cout << x << endl;
4   }
5   void imprimir(string x)
6   {
7       cout << x << endl;
8   }
```

- *imprimir(3)* está correcto.
- *imprimir("hola")* está correcto.

Sobrecarga de funcioón

```
1   void imprimir(int x)
2   {
3       cout << x << endl;
4   }
5   void imprimir(string x)
6   {
7       cout << x << endl;
8   }
```

- Varias funciones con el mismo nombre, pero diferente cantidad/tipo de argumentos.
- La función que se invocará es aquella que tiene el mismo tipo.

Sobrecarga de funcioón

```
1   void imprimir(int x)
2   {
3       cout << "Entero " << x << endl;
4   }
5   void imprimir(string x)
6   {
7       cout << "String " << x << endl;
8   }
```

- *imprimir(3)* imprime $->$ “Entero 3”.
- *imprimir(“hola”)* imprime $->$ “String hola”.

Sobrecarga de función

```
1   void imprimir(int x)
2   {
3       cout << " 1 entero " << x << endl;
4   }
5   void imprimir(int x, int y)
6   {
7       cout << "2 enteros " << x << " y " << y << endl;
8   }
```

- *imprimir(3)* imprime $->$ “1 entero 3”.
- *imprimir(2, 3)* imprime $->$ “2 enteros 2 y 3”

Recursividad

- Función que se llama a sí misma.
- Ejemplo: *fib(n) = fib(n-1) + fib(n-2)* puede ser facilmente expresado usando resursividad.

```
1   int fibonacci(int n) {
2       if (n == 0 || n == 1) {
3           return 1;
4       } else {
5           return fibonacci(n-2) + fibonacci(n-1);
6       }
7   }
```

Recursividad

- Función que se llama a sí misma.
- Ejemplo: *fib(n) = fib(n-1) + fib(n-2)* puede ser facilmente expresado usando resursividad.

```
1   int fibonacci(int n) {
2       if (n == 0 || n == 1) {                 //     |
3           return 1;                           //  <--|   CASO BASE
4       } else {                                //     |
5           return fibonacci(n-2) + fibonacci(n-1);
6       }
7   }
```

Recursividad

- Función que se llama a sí misma.
- Ejemplo: *fib(n) = fib(n-1) + fib(n-2)* puede ser facilmente expresado usando resursividad.

```
1   int fibonacci(int n) {
2       if (n == 0 || n == 1) {
3           return 1;
4       } else {
5           return fibonacci(n-2) + fibonacci(n-1);   // <====      PASO RECURSIVO
6       }
7   }
```

Variables Globales

- ¿Cuantas veces la función ***f*** es invocada? Se debe usar una variable global para determinarlo.
  - La variable global puede ser accedida desde cualquier función.

```
1   int numeroLlamadas = 0;
2
3   void f() {
4       ++numeroLlamadas;
5   }
6
7   int main() {
8       f(); f(); f();
9       cout << numeroLlamadas << endl; // 3
10  }
```

Ámbito

- Ámbito de una variable se refiere al rango se sentencias en el que dicha variable es **visible**.
- Una variable es **visible** en una sentencia si el valor de dicha variable en la sentencia es válido.

```
1   int numeroLlamadas = 0;
2
3   int potencia(int base, int exponent) {
4       numeroLlamadas = numeroLlamadas + 1;
5       int result = 1;
6       for (int i = 0; i < exponent; i++) {
7           result *= base;
8       }
9       return result;
10  }
11
12  int max(int num1, int num2) {
13      numeroLlamadas = numeroLlamadas + 1;
14      int result;
15      if (num1 > num2) {
16          result = num1;
17      }
18      else {
19          result = num2;
20      }
21      return result;
22  }
```

Ámbito

- Ámbito: en el lugar donde la variable fue declarada, determinará donde dicha variable será accedida.
- *numeroLlamadas* tiene un **ámbito global**, esto significa que puede ser accedido desde cualquier función.

```
1   int numeroLlamadas = 0;                 //  <==== VARIABLE CON AMBITO GLOBAL
2
3   int potencia(int base, int exponent) {
4       numeroLlamadas = numeroLlamadas + 1;
5       int result = 1;
6       for (int i = 0; i < exponent; i++) {
7           result *= base;
8       }
9       return result;
10  }
11
12  int max(int num1, int num2) {
13      numeroLlamadas = numeroLlamadas + 1;
14      int result;
15      if (num1 > num2) {
16          result = num1;
17      }
18      else {
19          result = num2;
20      }
21      return result;
22  }
```

Ámbito

- Ámbito: en el lugar donde la variable fue declarada, determinará donde dicha variable será accedida.
- *numeroLlamadas* tiene un **ámbito global**, esto significa que puede ser accedido desde cualquier función.
- *result* tiene un ámbito de función. Cada función tiene su propia variable *result* y dicha variable es independiente una de la otra.

```
1   int numeroLlamadas = 0;                 //  <==== VARIABLE CON AMBITO GLOBAL
2
3   int potencia(int base, int exponent) {
4       numeroLlamadas = numeroLlamadas + 1;
5       int result = 1;                     //  <==== VARIABLE CON AMBITO DE FUNCION
6       for (int i = 0; i < exponent; i++) {
7           result *= base;
8       }
9       return result;
10  }
11
12  int max(int num1, int num2) {
13      numeroLlamadas = numeroLlamadas + 1;
14      int result;                         //  <==== VARIABLE CON AMBITO DE FUNCION
15      if (num1 > num2) {
16          result = num1;
17      }
18      else {
19          result = num2;
20      }
21      return result;
22  }
```

Ámbito

- Los bubles y las condicionales tambien tienen su propio ámbito.
  - El ámbito de los contadores de un bucle es el cuerpo de dicho bucle.

```
1   double squareRoot(double num) {
2       double low = 1.0;
3       double high = num;
4
5       for (int i = 0; i < 30; i = i + 1) {
6           double estimate = (high + low) / 2;
7           if (estimate*estimate > num) {
8               double newHigh = estimate;
9               high = newHigh;
10          } else {
11              double newLow = estimate;
12              low = newLow;
13          }
14      }
15
16      return (high + low) / 2;
17  }
```

Paso por valor

- Se refiere cuando se pasa una variable como argumento de una función, en este caso **se pasa unicamente una copia de dicha variable**, solo se pasa el valor.
- Los cambios a la variable en dicha función, solo ocurren dentro de dicha función

```
1   // paso por valor
2   void incremento(int a) {
3       a = a + 1;
4       cout << "a dentro de la funcion incremento " << a << endl;
5   }
6
7   int main() {
8       int q = 3;
9       incremento(q); // cambios en la funcion incremento NO afectaran a la variable q
10      cout << "q en la funcion main " << q << endl;
11
12      return 0;
13  }
```

Salida:

a dentro de la funcion incremento 4
q en la funcion main 3

Paso por referencia

- Si se requiere modificar la variable original, que está siendo pasada como argumento, se debe utilizar el paso por referencia. (*int &a* en lugar de *int a*)
- En el paso por referencia se pasa **UNA REFERENCIA de la variable**. No se pasa una copia.

```
1   // paso por referencia
2   void incremento(int &a) {      //  <=== OJO 'int &a' en lugar de 'int a'
3       a = a + 1;
4       cout << "a dentro de la funcion incremento " << a << endl;
5   }
6
7   int main() {
8       int q = 3;
9       incremento(q); // cambios en la funcion incremento SI afectaran a la variable q
10      cout << "q en la funcion main " << q << endl;
11
12      return 0;
13  }
```

Salida:

a dentro de la funcion incremento 4
q en la funcion main 4

Ejemplo: función de intercambio - *swap*

- Implemente un programa para que intercambie los valores de dos variables.

```
1   int main() {
2       int q = 3;
3       int r = 5;
4       swap(q, r);
5       cout << "q " << q << endl; // q 5
6       cout << "r " << r << endl; // r 3
7   }
```

Ejemplo: función de intercambio - *swap*

- ERROR, si se usa el paso por valor no cambiarán los valores de las variables.

```
1   void swap(int a, int b) {
2       int t = a;
3       a = b;
4       b = t;
5   }
6
7   int main() {
8       int q = 3;
9       int r = 5;
10      swap(q, r);
11      cout << "q " << q << endl; // q 3
12      cout << "r " << r << endl; // r 5
13  }
```

Ejemplo: función de intercambio - *swap*

- CORRECTO, se debe usar el paso por referencia y de esta forma cambiarán los valores de las variables.

```
1   void swap(int &a, int &b) {     //  <=== OJO 'int &' en lugar de 'int '
2       int t = a;
3       a = b;
4       b = t;
5   }
6
7   int main() {
8       int q = 3;
9       int r = 5;
10      swap(q, r);
11      cout << "q " << q << endl; // q 5
12      cout << "r " << r << endl; // r 3
13  }
```

Gracias!!!