Introducción al Lenguaje C

Funciones y Punteros

Claudio Omar Biale
Facultad de Ciencias Exactas Químicas y Naturales
Universidad Nacional de Misiones
10/08/2012

Funciones

- Son subprogramas que realizan una operación y retornan **un** valor.
  - ¿Siempre?.
- Permiten "encapsular" una operación en particular, por lo que pueden ser reutilizadas.
- Ejemplos:
  - `getchar()`
  - `scanf()`
  - `sqrt()`
  - `abs()`

Características

- Permiten reutilizar código.
- El código es encapsulado.
  - Ej.: la implementación de `scanf()` está oculta.
- Pueden almacenarse en bibliotecas.
  - Ej.: `scanf()` se encuentra en la biblioteca `stdio`.

Definición de una Función

- Una función tiene la siguiente estructura:

```
tipo_de_datos nombre_función (parámetros) {
    cuerpo_de_la_función
}
```

  - El resultado obtenido por la función se devuelve por medio de la sentencia `return`.
  - Los “procedimientos” son funciones de un tipo especial: `void`.
  - Es posible el uso de recursividad en C.

Ejemplo

- Consideremos una función que convierte una temperatura en Celsius a una temperatura en Fahrenheit.

```
Fahrenheit = Celsius * 1.8 + 32.0
```

- Escriban una función `CaF` en lenguaje C que realice lo solicitado.

Ejemplo

- Definición e invocación:

```
#include <stdio.h>

double CaF (double v_celsius) {
     return v_celsius * 1.8 + 32.0;
}

int main (void) {
     double valor = 0.0;

     printf("Ingrese una temperatura en Celsius: ");
     scanf("%lf",&valor);
     printf("%f ", valor);
     printf("Su equivalente en Fahrenheit es: %f\n", CaF(valor));
     return 0;
}
```

- Definición
- Invocación

Ejemplo

- Definición e invocación:

```
#include <stdio.h>

double CaF (double v_celsius) {
     return v_celsius * 1.8 + 32.0;
}

int main (void) {
     double valor = 0.0;

     printf("Ingrese una temperatura en Celsius: ");
     scanf("%lf",&valor);
     printf("%f ", valor);
     printf("Su equivalente en Fahrenheit es: %f\n", CaF(valor));
     return 0;
}
```

- ¿Es correcto el código?.

Del Pequeño C ilustrado

- Para leer un `float` se utiliza: `%f`
- Para leer un `double` se utiliza: `%lf`
- Para leer un `long double` se utiliza: `%Lf`
- Para imprimir un `double` o `float` se utiliza: `%f`
- Para imprimir un `long double` se utiliza: `%Lf`
  - `Recordar`: en lugar de `f` pueden usar `a`, `e` o `g`

Prototipado o Declaración

- Describe la función, indicando:
  - el valor de retorno,
  - el nombre de la función y
  - el tipo y número de los parámetros.

- Permite declarar una función y luego en otra parte del código realizar la definición de la misma.

Volviendo a Nuestro Ejemplo...

- Declaración, invocación y definición:

```
#include <stdio.h>

double CaF (double);

int main (void) {
     double valor = 0.0;
     printf("Ingrese una temperatura en Celsius: ");
     scanf("%lf",&valor);
     printf("%f ", valor);
     printf("Su equivalente en Fahrenheit es: %f\n", CaF(valor));
     return 0;
}

double CaF (double v_celsius) {
     return v_celsius * 1.8 + 32.0;
}
```

- Declaración
- Invocación
- Definición

Recursividad

- Una función es recursiva cuando contiene una invocación a si misma.
- Deben tener un caso básico para no tener una recursión infinita.

Ejemplo

- Consideremos el clásico ejemplo del factorial de un número.
- La resolución del factorial de manera iterativa puede ser:

```
int factorial ( int valor) {
     int i, j;
     for (i = 1, j = 1; i <= valor; j *= i , i++)
          ;
     return j;
}
```

- Implementen la función `rFactorial` que permita obtener el factorial de un número de manera recursiva.

Ejemplo

- La resolución del factorial de manera recursiva puede ser:

```
int rFactorial(int n) {
     if(n == 0)
          return 1;
     else
          return n * rFactorial(n-1);
}
```

Punteros

- Es una de las características más sofisticadas del lenguaje.
- Mediante punteros podemos representar de forma efectiva distintas estructuras de datos complejas.
- Permiten además cambiar los valores pasados como argumentos *(parámetros)* a las funciones.
- ¡Trabajar con memoria dinámica!.

Punteros

- Un puntero es una variable que contiene una dirección de memoria donde se almacenan valores.

- Se declaran agregando un asterisco al tipo de datos de la variable.

```
int cant;
int * cant_p;
//cant es una variable de tipo entero
// cant_p es una variable de tipo puntero a un entero
```

Punteros

- Poseen dos operadores:
  - `Desreferenciamiento`: si `cant_p` es un puntero, la expresión `*cant_p` denota al contenido del puntero *(el valor almacenado en la dirección apuntada por el mismo)*.
  - `Enreferenciamiento`: si `cant` es una variable, la expresión `&cant` indica la dirección de memoria donde reside esa variable.

Ejemplo

- ¿Qué se imprime en el siguiente programa?:

```
#include <stdio.h>

int main (void) {
     int a = 2;
     int *pA = &a;

     printf("%d\n", a);
     printf("%x\n", a);
     printf("%p\n", &a);
     printf("%p\n", pA);
     printf("%d\n", *pA);
     printf("%p\n", &pA);

     return 0;
}
```

[1]

[1] Se presenta la impresión de valores de memoria "portable" usando el operador `%p`. Originalmente se presentó la impresión de valores de memoria usando `%x`, en Windows compila correctamente y en Linux compila con advertencias. Si se utiliza `%x` y se convierte el valor de memoria a `unsigned` se compila en Linux sin advertencias.

Ejemplo

- ¿Qué se imprime en el siguiente programa?:

```
#include <stdio.h>

int main (void) {
     int a = 10, b = 30;
     int *c = &a;

     printf("%p : %d\n", &a, a);
     printf("%p : %d\n", &b ,b);
     printf("%p : %p : %d\n", &c, c, *c);

     return 0;
}
```

Ejemplo

- ¿Qué se imprime en el siguiente programa?:

```
#include <stdio.h>

int main (void) {
     int a = 10;
     int *b = &a;

     *b = 2;
     printf("%d\n", a);
     printf("%d\n", *b);

     return 0;
}
```

Paso de Parámetros por Referencia

- Si bien C no permite pasar parámetros por referencia, el operador `&` permite simular ese tipo de pasaje de parámetro.

```
#include <stdio.h>

void absoluto(int *valor) {
     if (*valor < 0)
          *valor = - *valor;
}

int main (void) {
     int x = -10;

     absoluto(&x);
     printf("%d\n", x);

     return 0;
}
```

Introducción al Lenguaje C

Funciones y Punteros

Claudio Omar Biale
Facultad de Ciencias Exactas Químicas y Naturales
Universidad Nacional de Misiones
10/08/2012