Electrónica digital II

Santiago Rúa Pérez, PhD.

18 de septiembre de 2022

FUNCIONES Y LIBRERIAS EN C

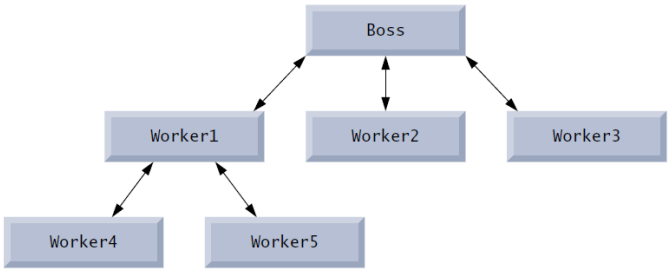
Programación modular

Objetivos

- Construir programas modulares.
- Usar funciones de la libreria math.
- Pasar información entre funciones.
- Recursividad.

Modularizar programas en C

En C, las funciones son utilizadas para modularizar programas. El objetivo es empaquetar pedazos de código para luego reutilizarlo.



Funciones de la libreria math.h

Esta libreria posibilita realizar operaciones matemáticas especializadas.

Function	Description	Example
`sqrt(x)`	square root of x	`sqrt(900.0)` is 30.0 `sqrt(9.0)` is 3.0
`cbrt(x)`	cube root of x (C99 and C11 only)	`cbrt(27.0)` is 3.0 `cbrt(-8.0)` is -2.0
`exp(x)`	exponential function e^x	`exp(1.0)` is 2.718282 `exp(2.0)` is 7.389056
`log(x)`	natural logarithm of x (base e)	`log(2.718282)` is 1.0 `log(7.389056)` is 2.0
`log10(x)`	logarithm of x (base 10)	`log10(1.0)` is 0.0 `log10(10.0)` is 1.0 `log10(100.0)` is 2.0
`fabs(x)`	absolute value of x as a floating-point number	`fabs(13.5)` is 13.5 `fabs(0.0)` is 0.0 `fabs(-13.5)` is 13.5
`ceil(x)`	rounds x to the smallest integer not less than x	`ceil(9.2)` is 10.0 `ceil(-9.8)` is -9.0
`floor(x)`	rounds x to the largest integer not greater than x	`floor(9.2)` is 9.0 `floor(-9.8)` is -10.0
`pow(x, y)`	x raised to power y (x^y)	`pow(2, 7)` is 128.0 `pow(9, .5)` is 3.0

Definición de una función

El objetivo de las funciones consiste en modularizar un software. Todas las variables definidas dentro de la función son locales. Las funciones deben crearse con proptotipo y la ejecución.

```
1  #include <stdio.h>
2
3  int square(int y); // function prototype
4
5  int main(void)
6  {
7    // loop 10 times and calculate and output square of x each time
8    for (int x = 1; x <= 10; ++x) {
9      printf("%d ",square(x)); // function call
10   }
11   puts("");
12 }
13
14 int square(int y) // y is a copy of the argument to the function
15 {
16   return y * y; // returns the square of y as an int
17 }
18
```

Ejemplo: realizar una funcion que reciba tres numeros enteros y retorne cual es el mayor.

Solucion Ejemplo

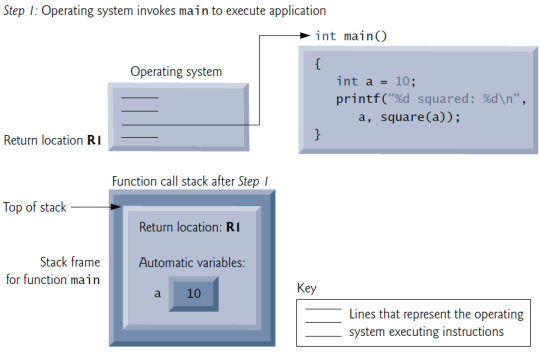
```c
#include <stdio.h>

int maximum(int x,int y, int z);

int main (void){
  int number1; // first integer entered by the user
  int number2; // second integer entered by the user
  int number3; // third integer entered by the user

  printf("%s", "Enter three integers: ");
  scanf("%d%d%d", &number1, &number2, &number3);

  // number1, number2 and number3 are arguments
  // to the maximum function call
  printf("El numero maximo es: %d\n", maximum(number1,number2,number3));
}

// Definicion de la funcion
// x, y and z are parameters
int maximum(int x, int y, int z)
{
  int max = x; // assume x is largest
  if (y > max) { // if y is larger than max,
    max = y; // assign y to max
  }
  if (z > max) { // if z is larger than max,
    max = z; // assign z to max
  }
  return max;
}
```

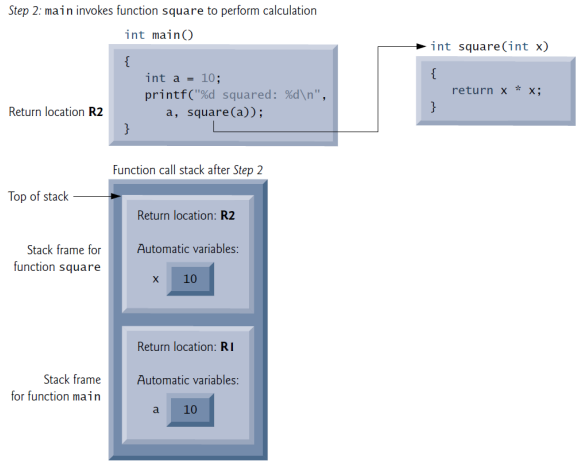
Pila llamadas a funciones

Es importante entender el concepto de una pila. Las pilas funcionan como el último dato que entra a la pila es el primero en salir. En cada llamado de función se debe almacenar el contexto donde se está trabajando y la dirección de memoria de donde fue llamado. Supongamos que se tiene el siguiente código.

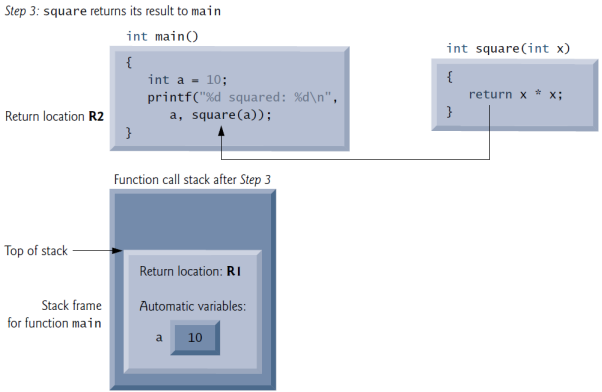
```
1   #include <stdio.h>
2
3   int square(int y); // function prototype
4
5   int main(void)
6   {
7     int a = 10; // value to square (local automatic variable in main)
8
9     printf("%d squared: %d\n", square(a) ); // display a squared
10  }
11
12  int square(int y) // y is a copy of the argument to the function
13  {
14    return y * y; // returns the square of y as an int
15  }
16
```

Pila llamadas a funciones





Pila llamadas a funciones



Scanf y Printf

Data type	`printf` conversion specification	`scanf` conversion specification
Floating-point types		
`long double`	`%Lf`	`%Lf`
`double`	`%f`	`%lf`
`float`	`%f`	`%f`
`unsigned long long int`	`%llu`	`%llu`
`long long int`	`%lld`	`%lld`
`unsigned long int`	`%lu`	`%lu`
`long int`	`%ld`	`%ld`
`unsigned int`	`%u`	`%u`
`int`	`%d`	`%d`
`unsigned short`	`%hu`	`%hu`
`short`	`%hd`	`%hd`
`char`	`%c`	`%c`

Headers

Cada libreria contiene un header. Estos header contienen los prototipos de las funciones en la libreria y las definiciones de varios tipos de datos.

Header	Explanation
`<math.h>`	Contains function prototypes for math library functions.
`<setjmp.h>`	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
`<signal.h>`	Contains function prototypes and macros to handle various conditions that may arise during program execution.
`<stdarg.h>`	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
`<stddef.h>`	Contains common type definitions used by C for performing calculations.
`<stdio.h>`	Contains function prototypes for the standard input/output library functions, and information used by them.
`<stdlib.h>`	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers and other utility functions.
`<string.h>`	Contains function prototypes for string-processing functions.
`<time.h>`	Contains function prototypes and types for manipulating the time and date.

Las comillas dobles para librerias propias.

Random Number Generation

La función rand() se encuentra en la libreria stdlib.h. Esta función generar un entero entre 0 y `RAND_MAX` el cual por defecto es 32767.

Ejemplo: Hacer un programa que simule lanzar los dados 20 veces e imprima su valor.

Las función rand() arroja número pseudoaleatorios. Para generar número aleatorios reales es importante usar la funcion srand(seed) para indicar la semilla a usar.

Random Number Generation - Solución

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6    // loop 20 times
7    for (unsigned int i = 1; i <= 20; ++i) {
8
9      // pick random number from 1 to 6 and output it
10     printf("%10d", 1 + (rand() % 6));
11
12     // if counter is divisible by 5, begin new line of output
13     if (i % 5 == 0) {
14       puts("");
15     }
16   }
17 }
18
```

Random Number Generation - Ejemplo 2

Ejemplo: Un jugador lanza el dado dos veces. Despues de lanzarlos realiza la suma de ambos dados. Si la suma es 7 u 11 en el primer lanzamiento, entonces el jugador gana. Si es 2, 3, o 12 entonces pierde. Si la suma es 4, 5, 6, 8, 9, o 10, entonces esa suma se convierte en el punto del jugador. Para ganar debe continuar lanzando los dados hasta que hagas el puntaje. El jugador pierde si lanza una suma de 7.

Hint: Haga una funcion que haga el lanzamiento de los dados y retorne la suma.

Hint 2: utilice enum para crear los posibles casos: `enum Status { CONTINUE, WON, LOST };`

Random Number Generation - Solución

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> // contains prototype for function time

// enumeration constants represent game status
enum Status { CONTINUE, WON, LOST };

int rollDice(void); // function prototype

int main(void)
{
  // randomize random number generator using current time
  srand(time(NULL));

  int myPoint; // player must make this point to win
  enum Status gameStatus;// can contain CONTINUE, WON, or LOST
  int sum = rollDice(); // first roll of the dice

  switch(sum) {
    case 7: case 11: // 7 and 11 is a winner
    gameStatus = WON;
    break;
    case 2:  case 3: case 12:// 2, 3 and 12 is a loser
    gameStatus = LOST;
    break;
    default:
    gameStatus = CONTINUE; // player should keep rolling
    myPoint = sum; // remember the point
    printf("Point is %d\n", myPoint);
    break; // optional
  }
```

Random Number Generation - Solución

```
1         while (CONTINUE == gameStatus) { // player should keep rolling
2         sum = rollDice(); // roll dice again
3         if (sum == myPoint) { // win by making point
4           gameStatus = WON;
5         }
6         else {
7           if (7 == sum) { // lose by rolling 7
8             gameStatus = LOST;
9           }
10        }
11      }
12      if (WON == gameStatus) { // did player win?
13        puts("Player wins");
14      }
15      else { // player lost
16        puts("Player loses");
17      }
18    }
19
20    int rollDice(void){
21      int die1 = 1 + (rand() % 6); // pick random die1 value
22      int die2 = 1 + (rand() % 6); // pick random die2 value
23
24      printf("Player rolled %d + %d = %d\n", die1, die2, die1 + die2);
25      return die1 + die2; // return sum of dice
26    }
27
```

Variables Locales vs Globales

Todas las variables definidas dentro de una función estan solo al alcance de esa función y se conocen como variables locales. Por otro lado las variables globales tienen alcance a todos. Cuando se le pone el prefijo de estático conserva su valor en el tiempo.

```c
// Fig. 5.16: fig05_16.c
// Scoping.
#include <stdio.h>

void useLocal(void); // function prototype
void useStaticLocal(void); // function prototype
void useGlobal(void); // function prototype

int x = 1; // global variable

int main(void)
{
   int x = 5; // local variable to main

   printf("local x in outer scope of main is %d\n", x);

   { // start new scope
      int x = 7; // local variable to new scope

      printf("local x in inner scope of main is %d\n", x);
   } // end new scope

   printf("local x in outer scope of main is %d\n", x);

   useLocal(); // useLocal has automatic local x
   useStaticLocal(); // useStaticLocal has static local x
   useGlobal(); // useGlobal uses global x
   useLocal(); // useLocal reinitializes automatic local x
   useStaticLocal(); // static local x retains its prior value
   useGlobal(); // global x also retains its value

   printf("\nlocal x in main is %d\n", x);
}
```

```c

// useLocal reinitializes local variable x during each call
void useLocal(void)
{
   int x = 25; // initialized each time useLocal is called

   printf("\nlocal x in useLocal is %d after entering useLocal\n", x);
   ++x;
   printf("local x in useLocal is %d before exiting useLocal\n", x);
}

// useStaticLocal initializes static local variable x only the first time
// the function is called; value of x is saved between calls to this
// function
void useStaticLocal(void)
{
   // initialized once
   static int x = 50;

   printf("\nlocal static x is %d on entering useStaticLocal\n", x);
   ++x;
   printf("local static x is %d on exiting useStaticLocal\n", x);
}

// function useGlobal modifies global variable x during each call
void useGlobal(void)
{
   printf("\nglobal x is %d on entering useGlobal\n", x);
   x *= 10;
   printf("global x is %d on exiting useGlobal\n", x);
}
```

Librerias propias

El objetivo de las librerias propias en modularizar el programa con funciones y archivos creados por nosotros mismos. Independizar estas funciones del código generan mayor modularidad.

- Cada libreria tiene dos archivos: un .h o header de las funciones, y un .c donde esta definida la función.
- El archivo .h debe comenzar indicandole al procesador el procedimiento a compilar. Es decir:

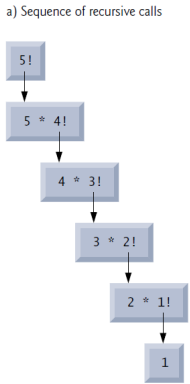
```
1   #ifndef _LIBRERIAPROPIA
2   #define _LIBRERIAPROPIA
3
4   // ------ Variables globales y definiciones ------//
5
6   // ------ Prototipos de funciones ----------//
7
8   ...
9   ...
10
11  #endif
12
```

- El archivo .c debe hacer un llamado a su propio header donde estan los prototipos e incluir librerias propias del sistema operativo necesarias.
- Posteriormente pueden ser llamadas desde el programa principal.

Ejemplo: Crear una libreria propia con dos o tres funciones de las antes creadas. Compilar usar Makefile

Recursividad

Consiste en llamarse asi mismo. Una función recursiva es aquella que tiene un llamado a si misma. **Ejemplo**: factorial.



Como implementarlo?

Factorial - Solución

```c
#include <stdio.h>

unsigned long long int factorial(unsigned int number);

int main(void)
{
  // during each iteration, calculate
  // factorial(i) and display result
  for (unsigned int i = 0; i <= 21; ++i) {
    printf("%u! = %llu\n", i, factorial(i));
  }
}

// recursive definition of function factorial
unsigned long long int factorial(unsigned int number)
{
  // base case
  if (number <= 1) {
    return 1;
  }
  else { // recursive step
    return (number * factorial(number - 1));
  }
}
```

Tarea: implementar la serie de Fibonnaci.

FUNCIONES Y LIBRERIAS EN C

GRACIAS