Laboratorio Sesión 02: Introducción al Lenguaje de Programación C

Objetivo

El objetivo de esta sesión de laboratorio es introducir al alumno en el uso del Lenguaje de Programación C. Este lenguaje será el que utilizaremos en todos los ejemplos de programas que veremos durante el curso.

No es un tema cerrado; en las siguientes sesiones de laboratorio se irán ampliando los conceptos.

Documentación

La referencia básica para iniciarse en C es el libro:

- Brian W. Kernighan and Dennis M. Ritchie. "El Lenguaje de Programación C". Ed. Prentice Hall, 1985.

Igualmente, pueden encontrarse en la red numerosos cursos interactivos de C (en la página www de la asignatura podéis encontrar algunos). Para empezar os recomendamos que os leáis el Manual de C que utilizaban los alumnos de ISO. Está muy resumido (13 páginas), pero tiene todos los elementos básicos para empezar. Este manual lo encontraréis en la página web de la asignatura.

El Primer Programa en C

El primer programa que vamos a compilar y ejecutar es el ejemplo que aparece en el manual de C que os hemos dado y simplemente calcula la expresión a^b, siendo a y b enteros positivos. Este programa lo encontraréis en la página web de la asignatura. El código del programa es el siguiente:

```
#include <stdlib.h>
main (int argc, char *argv[])
{  int error, i, result, base, exp;

   if (argc != 3)
      error = 1;   /* se ha invocado el programa de forma incorrecta */
   else {
      result = 1;
      base = atoi(argv[1]);  /* atoi convierte un string en un entero */
      exp = atoi(argv[2]);
     if (exp > 0) {
       for (i = 0; i < exp; i++)
         result = result * base;
      }
      error = 0;
      printf("%d elevado a %d es %d \n", base, exp, result);
   }
   return (error);
}
```

Una vez que tenemos escrito el programa, los pasos necesarios para ejecutarlo son los siguientes:

1) Compilar el programa:

```
$> gcc expon.c –o expon
```

2) Ejecutar el programa:

```
$> ./expon 3 4
3 elevado a 4 es 81
```

3) Modificar el programa para que muestre por pantalla un mensaje de error cuando invocamos el programa de forma incorrecta. Para editar el programa podéis utilizar el

editor vim (se comporta de forma similar al vi). Podéis encontrar una guía rápida de uso del vi en la página web de la asignatura.

Puzzles en C

Para continuar la sesión os proponemos un conjunto de pequeños ejercicios en C que hemos llamado "puzzles". Estos puzzles consisten en realizar algunas operaciones aritméticas o lógicas con un conjunto limitado y predeterminado de operadores. **MUY IMPORTANTE: en estos puzzles sólo se pueden utilizar operaciones de asignación con los operadores indicados, no se pueden utilizar sentencias condicionales (tipo if o ?) ni iterativas (tipo while)**. Por ejemplo, el código de uno de los puzzles podría ser el siguiente:

```
#include <stdio.h>
#include <limits.h>

int BitXor(int a, int b);
int TestXor(int a, int b);

main (int argc, char *argv[])
{
   int vtest[10] = {1, -1, 0, INT_MIN, INT_MAX, 7, -13, 34, 9, 3000};
   int i, j;
   int xxxx, test;
   int error;

   error = 0;
   for (i=0; i<10; i++)
     for (j=0; j<10; j++) {
       xxxx = BitXor(vtest[i], vtest[j]);
       test = TestXor(vtest[i], vtest[j]);
       if (xxxx != test) error++;
     }
   if (!error)
     printf("\n\nLa función programada funciona correctamente.\n\n");
   else
     printf("\n\nSe han producido %d errores.\n\n", error);
}

int TestXor (int a, int b)
{
   return (a ^ b);
}

int BitXor(int a, int b)
{
   /*  Calculad a ^ b
    *  ejemplo: BitXOr(7,3) = 4
    *  Operadores legales : ~ &
    *  Numero máximo de operadores: 14
    *  Nivel de dificultad: 2
    */

   return (~a)&(b);     /* Esta solución es INCORRECTA */
}
```

En el ejemplo, hay que programar la función `BitXor`, utilizando únicamente los operadores `&` y `~`. Notad que los puzzles incluyen el código necesario para comprobar que vuestra implementación funciona correctamente.

Nota: Las constantes `INT_MIN` e `INT_MAX`, son el mínimo y máximo valor entero. Estas constantes están definidas en el fichero `limits.h`.

En la página web de la asignatura encontraréis todos los ficheros con los puzzles.

Algo de Linux: Compresión de ficheros

Linux dispone de varias herramientas para compactar, comprimir ficheros, etc. En la siguiente tabla mostramos algunas de esas herramientas:

comando	Descripción	Ejemplos
gzip fichero	Comprime el fichero especificado. El fichero comprimido queda con el nobre `fichero.gz`.	`gzip lista.dat`
gunzip fichero.gz	Descomprime el fichero especificado. El fichero comprimido ha de llamarse `*.gz`.	`gunzip lista.dat.gz`
zip seguro.zip file(s)	Comprime los ficheros especificados sobre el fichero indicado (`seguro.zip`).	`zip copia.zip *.c`
unzip seguro.zip	Descomprime el fichero especificado	`unzip copia.zip`
tar cvf file.tar file(s)	Almacena los ficheros especificados en el archivo indicado (`file.tar`). Si el fichero especificado es un directorio se almacena el contenido del directorio.	`tar cvf copia.tar 02/*.c`
tar xvf file.tar	Extrae todos los ficheros del archivo indicado. Los archivos son extraidos con la misma estructura que tenían originalmente. En caso necesario crea los directorios que hagan falta.	`tar xvf copia.tar`

Es bastante común combinar el comando **`tar`** y **`gzip`**. Para extraer los ficheros una vez descargados de la página web, hay que hacer lo siguiente:

```
$> gunzip Programas.Sesion02.tar.gz
$> tar xvf Programas.Sesion02.tar
$> cd Sesion02
$> ls -l
```

Trabajo previo de la sesión

Como trabajo previo de la sesión, **que debe entregarse al profesor al inicio de la sesión**, hay que realizar las siguientes tareas:

1) Aplicad la ley de Morgan a la función lógica "a+b".
2) Dados dos números en complemento a 2 (A y B, con bits de signo a_s y b_s), indicad cuándo se produce overflow al realizar la suma S = A + B (siendo s_s el bit de signo de la suma).
3) Escribid la tabla de verdad de la función lógica para sumar 3 bits (x,y,z).
4) Dados dos números en complemento a 2 (A y B), indicad en qué condiciones A es mayor que B (nota: podéis hacerlo en función del signo de A-B).
5) Dados dos números en complemento a 2 (A y B), indicad cómo podemos realizar la operación de resta (A-B) sin utilizar el operador de resta "-".
6) Dado "y",un número entero codificado en complemento a 2 con 32 bits, si realizamos la siguiente operación en C: "x = y>>31", ¿qué valores puede tomar x?
7) Dado "y",un número entero codificado en complemento a 2 con 32 bits, si realizamos la siguiente operación en C: "x = y>>32", ¿qué valores puede tomar x?
8) Explicad los parámetros, argv y argc, que aparecen en el main del programa "expon.c". ¿Para qué sirven? ¿Cómo se utilizan? ¿Porqué no se utilizan esos parámetros en los puzzles?

- Si tenéis acceso a un sistema Linux: compilad y ejecutad el programa "expon.c"; modificad el programa "expon.c".

Resultados de la sesión

Haced todos los puzzles que sea posible. En cada uno de ellos se indica el nivel de dificultad. Para sacar una buena nota de esta sesión hay que haber hecho al menos 3-4 puzzles.