Práctica 0: Conceptos básicos

Programación 2

Curso 2019-2020

Esta primera práctica del curso consiste en la realización de una serie de ejercicios cortos, con el objetivo de que repases conceptos básicos de programación en C++. Los conceptos necesarios para desarrollar esta práctica se trabajan en el *Tema 1* de teoría.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 14 de febrero**, hasta las **23:59**
- Debes entregar un único fichero llamado `prac0.cc` con el código de todas las funciones

Código de honor

Si se detecta copia (total o parcial) en tu práctica, tendrás un **0** en la entrega y se informará a la dirección de la Escuela Politécnica Superior para que adopte medidas disciplinarias

Está bien discutir con tus compañeros posibles soluciones a las prácticas
Está bien apuntarte a una academia si sirve para obligarte a estudiar y hacer las prácticas

Está mal copiar código de otros compañeros para resolver tus problemas
Está mal apuntarte a una academia para que te hagan las prácticas

Si necesitas ayuda acude a tu profesor/a
No copies

Normas generales

- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Se puede acceder a él de dos maneras:
  - Página principal del DLSI (`https://www.dlsi.ua.es`), opción "ENTREGA DE PRÁCTICAS"
  - Directamente en la dirección `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud
  - No se admitirán entregas fuera de plazo
- Tu práctica debe poder ser compilada sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas

- Si tu práctica no se puede compilar su calificación será 0
- La corrección de la práctica se hará de forma automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

prac0.cc

```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las transparencias de presentación de la asignatura (*Tema 0*)

1. Descripción de la práctica

En esta primera práctica del curso deberás desarrollar una serie de ejercicios cortos que se describen a continuación. Debes incluir todo el código en el mismo fichero fuente `prac0.cc`. En dicho fichero **no debe haber** una función `main` en el momento de la entrega, ya que los profesores incluirán su propia función principal para la corrección.

En el Moodle de la asignatura se publicará un fichero fuente `main_prac0.cc`, que contendrá una función `main` para probar tu práctica utilizando como entrada los ejemplos que aparecen en este enunciado. En este fichero podrás comentar las llamadas a funciones que todavía no hayas implementado para probar únicamente aquellas que estén completadas.

- El fichero `main_prac0.cc` no se deberá incluir en la entrega, solamente el fuente `prac0.cc` realizado por ti

Para compilar `main_prac0.cc` con tu fichero fuente `prac0.cc` y generar un único fichero ejecutable, debes utilizar la siguiente orden por línea de comando:

Terminal

```
$ g++ -Wall main_prac0.cc prac0.cc -o prac0
```

Cada ejercicio de esta práctica se corregirá y puntuará por separado utilizando pruebas adicionales diseñadas por los profesores de la asignatura. Por esta razón, no te conformes con que tu práctica funcione con los ejemplos de este enunciado. Trata de pensar en todas las posibles situaciones que se pueden dar en cada ejercicio y comprueba que funcionan añadiendo al fichero `main_prac0.cc` pruebas adicionales.

2. Ejercicios

A continuación se detallan los ocho ejercicios que componen esta primera práctica. Para todos ellos se proporciona el prototipo de la función a desarrollar, su descripción y ejemplos de entrada y salida.

1. **Borrar caracteres** (1 punto)

   Implementa una función que reciba dos parámetros: una cadena almacenada en un vector de caracteres (terminada en '\0') y un carácter. La función debe eliminar, sobre la misma cadena, todas las apariciones de dicho carácter, reorganizando la misma para que no queden huecos.

- El prototipo de la función será: `void deleteChar(char str[],char c)`
- La cadena resultante debe acabar en '\0', como la cadena original
- No se debe utilizar ninguna cadena auxiliar: se han de borrar todas las apariciones del carácter sobre la misma cadena

Ejemplos:

```
char test[]="cadena de pruebaas";
char hola[]="hola, mundo";
```

Llamada	Cadena resultante
`deleteChar(test,'a')`	`"cden de pruebs"`
`deleteChar(test,'e')`	`"cadna d prubaas"`
`deleteChar(hola,'a')`	`"hol, mundo"`
`deleteChar(hola,'o')`	`"hla, mund"`

2. **Factorial** (1 punto)

   Diseña una función que, dado un número entero positivo, devuelva el factorial de dicho número.

   - El factorial de un número entero positivo *n* es el producto de todos los números enteros positivos desde *1* hasta *n*
   - El prototipo de la función será: `unsigned factorial(unsigned n)`
   - El tipo de dato `unsigned` representa un entero sin signo, es decir, representa siempre un número positivo (incluido el 0). En general, este tipo de dato se puede usar cuando sabemos que la variable nunca va a contener números negativos
   - El número `n` será siempre mayor que 0 (no es necesario comprobarlo)

   Ejemplos:

Llamada	Valor devuelto
`factorial(1)`	1
`factorial(2)`	2
`factorial(3)`	6
`factorial(5)`	120

3. **Comprobar contraseña** (1 punto)

   Diseña una función que compruebe si una contraseña tiene un formato correcto. Se admitirán dos tipos de contraseñas:

   - Contraseña alfanumérica de entre 8 y 12 caracteres: debe contener letras mayúsculas, minúsculas y dígitos (al menos una de cada)
   - Contraseña numérica de 16 caracteres: sólo contiene dígitos

   La función devolverá `true` si la contraseña es correcta o `false` si no lo es, de acuerdo a las reglas anteriores.

   - El prototipo de la función será: `bool checkPassword(const char passwd[])`
   - La cadena `passwd` siempre acabará en '\0'
   - Las posibles letras que contendrá la cadena `passwd` serán siempre las correspondientes al alfabeto inglés (sin acentos, diéresis, etc.). No hace falta comprobarlo
   - En ningún caso se admitirán caracteres distintos a los indicados: si aparece algún carácter incorrecto la función devolverá `false`
   - Puedes utilizar las funciones `islower()`, `isupper()` e `isdigit()` de la librería `cctype`

Ejemplos:

Llamada	Valor devuelto
`checkPassword("holaMundo")`	`false`
`checkPassword("hlM7no")`	`false`
`checkPassword("hlM7no8jk43")`	`true`
`checkPassword("hlM7no8jk43aa")`	`false`
`checkPassword("hola, mundo")`	`false`
`checkPassword("0123456789012345")`	`true`
`checkPassword("01234567890123a5")`	`false`

4. **Calculadora** (1 punto)

   Diseña una función que lea los valores numéricos de un vector de enteros y los operadores de un `string` de forma alterna, realice las operaciones y devuelva el valor calculado.

   

   - El prototipo de la función será: `int calculator(const int numbers[],unsigned size,const char operators[])`
   - `numbers` es el vector de enteros candidatos a ser operados
   - `size` es la cantidad de operandos del vector `numbers` que se desea operar. Será siempre mayor que `0` y menor o igual que la cantidad de enteros en el vector (no es necesario comprobarlo)
   - `operators` es la cadena con los operadores, que pueden ser cualquiera de estos: `+-*/`. El operador `/` solo devolverá la parte entera de la división. Consideramos que todos los operadores tienen la misma prioridad, por lo que el orden de evaluación será de izquierda a derecha
   - Se dejará de leer y evaluar operaciones cuando se lea el último número, se lea el último operador o se lea un operador erróneo
   - El número de operadores del vector `operators` será siempre mayor o igual que `size-1`, es decir, hay suficientes operadores para los operandos con los que se quiere trabajar. No hace falta comprobarlo

   Ejemplos:

   ```
   const int numbers[]={10,15,4,-5};
   ```

Llamada	Operaciones	Valor devuelto
`calculator(numbers,4,"+*/")`	10 + 15 * 4 / (-5)	-20
`calculator(numbers,2,"+*/")`	10 + 15	25
`calculator(numbers,3,"+*/")`	10 + 15 * 4	100
`calculator(numbers,4,"+:/")`	10 + 15	25
`calculator(numbers,1,"+:/")`	10	10

5. **Construir un número** (1 punto)

   Diseña una función que convierta un vector de números en un único número que representa la concatenación de todos ellos. Por ejemplo, un vector que contiene los números `1`, `3` y `7`, se convertiría al número `137`.

   

   - El prototipo de la función será: `unsigned buildNumber(const unsigned numbers[],unsigned size)`
   - Todos los números proporcionados estarán entre 0 y 9, incluidos (no es necesario comprobarlo)
   - Si el vector está vacío (`size` es 0), la función devolverá `0`

Ejemplos:

Llamada	Valor devuelto
`buildNumber({1,2,3,4,5},5)`	12345
`buildNumber({4,3,0,2,2,2},6)`	430222
`buildNumber({0,4,2},3)`	42

6. **Números amigos** (1 punto)

   Implementa una función que permita comprobar si dos números naturales son amigos. Dos números x e y son amigos si son mayores que 0 y la suma de los divisores propios de x da como resultado y, y viceversa. Los divisores propios de un número n son todos los divisores de n, incluido el 1 como divisor, pero no él mismo. Ejemplo:

220	1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 (su suma es 284)
284	1, 2, 4, 71, 142 (su suma es 220)

   Como curiosidad, si $x = y$ y se cumple que x (o y) coincide con la suma de sus divisores propios (es decir, que x es amigo de él mismo), se dice que es un número perfecto. Por ejemplo: $6 = 1 + 2 + 3$.

   - El prototipo de la función será: `bool friends(unsigned x, unsigned y)`
   - Devolverá `true` si x e y son amigos (o si $x = y$ y es perfecto). En cualquier otro caso devolverá `false`

   Ejemplos:

Llamada	Valor devuelto
`friends(2,2)`	`false`
`friends(220,284)`	`true`
`friends(20,54)`	`false`
`friends(6,6)`	`true`

7. **Los vecinos suman** (2 puntos)

   Diseña una función que reciba como parámetros una matriz cuadrada (mismo número de filas que de columnas) de enteros y una posición en la matriz (dos enteros: fila y columna), y devuelva la suma de los vecinos de esa posición de la matriz.

   - El prototipo de la función será: `int sumNeighbors(int m[][kMATSIZE],int row,int column)`
   - La matriz será siempre de 5×5, por lo que debes añadir esta declaración antes de la función: `const unsigned kMATSIZE=5;`
   - Ten en cuenta que una posición no es vecina de sí misma
   - Tanto `row` como `column` deben tener valores entre `0` y `kMATSIZE-1`. En caso contrario la función debe devolver `-1` para indicar el error

   Ejemplos:

```
int matrix[kMATSIZE][kMATSIZE] = {  1, 2, 3, 4, 5,
                                    6, 7, 8, 9,10,
                                   11,12,13,14,15,
                                   16,17,18,19,20,
                                   21,22,23,24,25 };
```

Llamada	Valor devuelto
`sumNeighbors(matrix,2,2)`	104
`sumNeighbors(matrix,0,0)`	15
`sumNeighbors(matrix,4,4)`	63

8. **Imprimir una X (2 puntos)**

Diseña una función que dibuje una X en pantalla usando el carácter 'X' (mayúscula) y espacios en blanco. La función recibirá un número entero como parámetro que indicará el tamaño de la X a dibujar, que tendrá el mismo número de filas que de columnas.

- El prototipo de la función será: `void printX(int n)`
- El parámetro `n` deberá ser un número positivo e impar. En caso contrario se deberá mostrar el mensaje `ERROR` por pantalla y terminar la ejecución de la función sin imprimir nada más

Ejemplos:

Llamada	Salida por pantalla
`printX(1)`	`X`
`printX(2)`	`ERROR`

`printX(5)`	`X   X`
	` X X`
	`  X`
	` X X`
	`X   X`