Práctica 0: Conceptos básicos

Programación 2

Curso 2018-2019

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. Estos conceptos introductorios se trabajarán en el *Tema 1* de teoría.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 15 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

Esta práctica consiste en una colección de ejercicios que se describen a continuación. Deben aparecer todos en el mismo fichero fuente `prac0.cc`. En dicho fichero **no debe haber** una función `main` en el momento de la entrega. En la web de la asignatura se publicará el fichero fuente `mainPrac0.cc`,[1] 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. Para compilar `mainPrac0.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 mainPrac0.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 ampliando el fichero `mainPrac0.cc` con pruebas adicionales.

2 Ejercicios

A continuación se detallan los siete 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. **Sentencia condicional**

   Diseña una función que reciba dos números enteros positivos `a` y `b` como parámetros y devuelva `true` si `a` es múltiplo de `b` o bien `b` es múltiplo de `a`, y `false` si ninguno de los dos es múltiplo del otro.

   - El prototipo de la función será: `bool multiple(int a, int b)`
   - No hace falta comprobar que `a` y `b` son positivos
   - Si imprimes un booleano por pantalla, no mostrará `true` y `false`, sino `1` y `0`

[1] Este fichero no se deberá incluir en la entrega, solamente el fuente `prac_0.cc` realizado por ti.

Ejemplos:

Llamada	Valor devuelto
`multiple(15,3)`	`true`
`multiple(17,5)`	`false`
`multiple(5,76)`	`false`
`multiple(12,72)`	`true`

2. **Sentencia repetitiva**

   Diseña una función que reciba como parámetro un número entero positivo `n` e imprima por pantalla una `N`, utilizando caracteres `*`, que tenga forma cuadrada (es decir, tantos caracteres de alto como de ancho) en función de un valor n, que debe ser mayor o igual que 4. En caso contrario, la función debe mostrar por pantalla:

   Terminal
   ```
   ERROR
   ```

   La función debe devolver el número de asteriscos que ha impreso.

   - El prototipo de la función será: `int printN(int n)`
   - Si se produce error, la función debe devolver `0`

   Ejemplos:

Llamada	Salida	Valor devuelto
`printN(1)`	`ERROR`	`0`

`printN(4)`	`*  *`
	`** *`
	`* **`
	`*  *`
		`10`

`printN(6)`	`*    *`
	`**   *`
	`* *  *`
	`*  * *`
	`*   **`
	`*    *`
		`16`

3. **Números primos**

   Diseña una función que, dado un número entero n, imprima por pantalla los n primeros números primos y devuelva su suma. Debes utilizar una función auxiliar que indique si un número es primo o no.

   - El prototipo de la función principal será: `int firstPrimes(int n)`
   - El prototipo de la función auxiliar será: `bool isPrime(int p)`
   - El número n será siempre mayor que 0. No es necesario comprobarlo
   - Un número es primo si es mayor que 1 y solamente es divisible por sí mismo y por 1. Por tanto, el 1 no es primo (ni el 0, por supuesto)
   - Los números se mostrarán por pantalla separados por un espacio en blanco entre sí

Ejemplos:

Llamada	Salida	Valor devuelto
`firstPrimes(1)`	`2`	`2`
`firstPrimes(3)`	`2 3 5`	`10`
`firstPrimes(6)`	`2 3 5 7 11 13`	`41`
`firstPrimes(10)`	`2 3 5 7 11 13 17 19 23 29`	`129`

4. **Goldbach**

   La conjetura de Goldbach dice que todo número par mayor que 2 puede escribirse como la suma de dos números primos. Diseña una función que, dado un número entero n, devuelva en parámetros pasados por referencia los dos números primos cuya suma es n. Por ejemplo, si n es 28 debe devolver 5 y 23.

   - El prototipo de la función será: `void Goldbach(int n,int &p1,int &p2)`
   - En el parámetro `p1` se debe devolver el menor número primo que cumpla la conjetura y en el parámetro `p2` el otro número primo. Puede darse el caso que `p1` y `p2` sean el mismo número. Por ejemplo, $6 = 3 + 3$
   - Puede haber varias parejas de `p1` y `p2` que cumplan la conjetura, pero se debe devolver aquella que contenga el menor número primo
   - El número n será siempre par y mayor que 2. No hace falta comprobarlo
   - Debes reutilizar la función auxiliar `bool isPrime(int p)` del ejercicio anterior para comprobar si un número es primo

   Ejemplos:

Llamada	p1	p2
`Goldbach(6,p1,p2)`	3	3
`Goldbach(12,p1,p2)`	5	7
`Goldbach(458,p1,p2)`	19	439
`Goldbach(32896,p1,p2)`	53	32843

5. **Búsqueda de un elemento en un vector**

   Dado un vector de enteros, su tamaño y un número entero n, diseña tres funciones para buscar n en el vector.

   (a) La primera de ellas es la función `search`, que devuelve `true` si n está en el vector y `false` en caso contrario.

   - El prototipo de la función será: `bool search(int v[], const int TAMVECTOR, int n)`

   Ejemplos:

Llamada	Valor devuelto
`search({1,2,3,4,5,6,5,4,3,2,1},11,7)`	`false`
`search({1,2,3,4,5,6,5,4,3,2,1},11,2)`	`true`

   (b) La segunda es la función `position`, que devuelve un valor entero indicando la posición que ocupa n dentro del vector o `-1` en caso de no haberlo encontrado.

   - El prototipo de la función será: `int position(int v[],const int TAMVECTOR, int n)`
   - Si número n está en la primera posición del vector, la función devolverá 0, es decir, el índice del primer elemento del vector
   - Si n aparece más de una vez, devuelve la posición de la primera aparición

Ejemplos:

Llamada	Valor devuelto
`position({1,2,3,4,5,6,5,4,3,2,1},11,7)`	`-1`
`position({1,2,3,4,5,6,5,4,3,2,1},11,2)`	`1`

(c) Finalmente, la tercera función es `count`, que devuelve el número de veces que n aparece en el vector. Este valor puede ser `0` si n no está en el vector.

- El prototipo de la función será: `int count(int v[],const int TAMVECTOR, int n)`

Ejemplos:

Llamada	Valor devuelto
`count({1,2,3,4,5,6,5,4,3,2,1},11,7)`	`0`
`count({1,2,3,4,5,6,5,4,3,2,1},11,2)`	`2`

6. **Mínimo y máximo de un vector**

Dado un vector con números positivos, diseña dos funciones para encontrar el mínimo y máximo valor de dicho vector.

(a) La primera función se llamará `minOdds` y recibe un vector de números y su tamaño. Deberá devolver el mínimo de los números impares que contenga el vector o `-1` si no contiene ningún número impar.

- El prototipo de la función será: `int minOdds(int v[],const int TAMVECTOR)`
- Puedes asumir que todos los números que contiene el vector `v` son mayores o iguales a 0. No es necesario comprobarlo
- Puede haber números repetidos en el vector

Ejemplos:

Llamada	Valor devuelto
`minOdds({3,4,7,2,275,5,1},7)`	`1`
`minOdds({2,4,8,0,24,8},6)`	`-1`
`minOdds({3,7,8,51,3,2},6)`	`3`

(b) La segunda función se llamará `posMaxMultFive` y recibe un vector de números y su tamaño. Deberá devolver la posición en el vector del máximo de los números múltiplos de 5 que tenga el vector, o `-1` si no tiene ningún múltiplo de 5.

- El prototipo de la función será: `int posMaxMultFive(int v[],const int TAMVECTOR)`
- ¡Ojo! Debe devolver la posición que ocupa en el vector el número, no el número en sí
- Puedes asumir que todos los números que contiene el vector `v` son mayores o iguales a 0. No es necesario comprobarlo
- Puede haber números repetidos en el vector

Ejemplos:

Llamada	Valor devuelto
`posMaxMultFive({3,4,7,2,275,5,1},7)`	`4`
`posMaxMultFive({2,3,8,17,24,8},6)`	`-1`
`posMaxMultFive({3,7,8,50,3,2},6)`	`3`

7. **Extraer datos**

Diseña una función que recibirá como parámetro una cadena de caracteres con datos sobre un alumno, separados por el carácter dos puntos (`:`), con la siguiente información: número identificador, nombre, nota numérica y observaciones. Un ejemplo de cadena sería el siguiente:

```
1234:Perez Perez, Pedro:3.4:Good work, but did not submit the last assignment
```

La función debe extraer los datos y guardarlos en un registro del siguiente tipo:

```
struct Student
{
  int number;
  string name;
  double mark;
  string obs;
};
```

Una vez guardados en el registro, la función debe escribir por pantalla el contenido del mismo, indicando el nombre (`name`), nota (`mark`), observaciones (`obs`) y número identificador (`number`). Por ejemplo, la salida por pantalla para la entrada anterior sería:

Terminal

```
Name: Perez Perez, Pedro
Mark: 3.4
Obs: Good work, but did not submit the last assignment
Number: 1234
```

- El prototipo de la función será: `void extractStudentData(char data[])`
- La cadena `data` estará bien formada, por lo que terminará siempre en `\0` y se podrá aplicar sobre ella las funciones de la librería `string.h` que se consideren necesarias

Ejemplos:

`data`	Salida

`33452:Cruz Pi, Teo:7.9:No comment`	`Name: Cruz Pi, Teo`
	`Mark: 7.9`
	`Obs: No comment`
	`Number: 33452`

`12:Mas Mora, Ana:9.4:Superb!`	`Name: Mas Mora, Ana`
	`Mark: 9.4`
	`Obs: Superb!`
	`Number: 12`