PRO1: Lista P4

Emma Rollón

18 de marzo de 2020

Problema P37760

En este problema tenemos que tener en cuenta que en la librería cmath hay dos funciones llamadas sin y cos que calculan el sinus y cosinus de un valor real que representa un ángulo en radianes. Por tanto, si queremos utilizarlas, tendremos que incluir esa librería en la zona de cabeceras de nuestro programa.

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    ....
}
```

Pasos que dará nuestro programa:

1. mientras haya valores en la entrada:
    1.1 pasar ese valor a radianes
    1.2 escribir el resultado que nos da la llamada a sin y cos

Problema X04437

En este problema, y como en muchos otros de esta lista, sólo nos pidan la implementación de una acción/función. Sin embargo, tenemos que tener en cuenta que:

1. Para que nuestra implementación compile en local, tenemos que incluir un main (que puede no tener ninguna instrucción).

2. Para garantizar que nuestra implementación es correcta, tenemos que probar su funcionamiento en local. Por tanto, tenemos que implementar un main que haga esas pruebas.

En este problema, podemos plantear probar si la función es correcta para un conjunto de pares que representan puntos. Por tanto, nuestra entrada será una secuencia de pares de reales y, para cada uno de ellos, escribiremos su distancia al origen.

```
int main() {
    double x, y;
    while (cin >> x >> y) {
        cout << dist_or(x, y) << endl;
    }
}
```

Problema P59652

En este problema nos piden que implementemos una acción con la siguiente descripción:

```
// Pre: en la entrada hay dos enteros mayores estrictos que 0
// Post: num y den  se han leído de la entrada
//          y, al finalizar la acción, no tienen factores comunes
void read_rational(int& num, int& den) {
    ...
}
```

Fíjate que, dado que la acción tiene que leer de la entrada, la Precondición nos informa sobre los valores que deben estar disponibles en la entrada para poder llamar a esta acción. Es una precondición un tanto especial. Esta Precondición nos dice que jamás podrán hacer una llamada a read_rational si en la entrada hay un par de valores que son 0 o negativos. Por tanto, no tenemos que preocuparnos de los casos 0/0, 0/4, 5/0, ... porque éstos no cumplen la precondición (es decir, no son válidos).
La Postcondición nos informa sobre qué hace y, por tanto, nos dice qué valores tendrán sus dos parámetros de salida num y den una vez acabe la acción.

Pasos que dará la acción:

1. Leer de la entrada los valores num y den
2. Calcular el máximo común divisor
3. Dividir num y den por ese valor

Podemos, por tanto, encapsular el segundo paso en una nueva función que calcule el máximo común divisor de dos valores estrictamente mayores que 0.

```
// Pre: x, y mayores estrictos que 0
// Post: retorna el máximo común divisor de x, y
int mcd(int x, int y) {
    ...
}
```

Una vez implementada la podemos utilizar dentro de la acción read_rational.

Igual que en el ejercicio anterior, tenemos que implementar un main que nos permita comprobar que la implementación de la acción es correcta. En este caso, como es la acción la que realiza la lectura, planteamos un main que sólo trate un par de valores de la entrada.

```
int main() {
    double n, d;
    read_rational(n, d);
    cout << n << " " << d << endl;
}
```

Como los dos parámetros de read_rational son de salida, fíjate que las variables n y d con las que se hacen la llamada NO están inicializadas. Es la acción read_rational la que se encarga de darles un valor válido cuando se ejecuta.

Problema P58653

En este problema nos dan una acción implementada para que la utilicemos en la implementación de nuestro programa. Primero, tenemos que entender muy bien qué hace. Nos lo describe en el enunciado. Escrito como Pre y Post sería:

```
// Pre: c, s, b son válidos
// Post: escribe en una línea si el carácter c es o no s en función de b
void escriu_linia(char c, string s, bool b) {
    cout << s << "('" << c << "') = ";
    if (b) cout << "cert" << endl;
    else cout << "fals" << endl;
}
```

Primero, la Pre podríamos haberla dejado vacía porque los parámetros de entrada siempre han de ser válidos. Por tanto, como no tienen que cumplir ninguna otra propiedad especial, no sería necesario indicar nada.
Segundo, la Post nos describe qué hace la acción. También nos dan un ejemplo de utilización para aclarar ese funcionamiento. Vamos a ver un par más:

- para la llamada escriu_linia('6', "lletra", false), la acción escribe el string:

$$lletra('6') = fals$$

- para la llamada escriu_linia('6', "lletra", true), la acción escribe el string:

$$lletra('6') = cert$$

Por lo tanto, parece que las llamadas a esta acción nos puede ayudar a escribir lo que nos pide el problema. Como inciso, fíjate que también podríamos hacer llamadas a esta acción como escriu_linia(';', "PRO1", true) que escribiría:

$$PRO1(';') = cert$$

Es un uso posible, pero no es el que nos interesa para solucionar este problema.

Pasos que dará nuestro programa (versión 1):

1. Leer el carácter de la entrada
2. Si es letra entonces llamar a escriu_linia con string valiendo "lletra", y el booleano valiendo true;
   si no llamar a escriu_linia con string valiendo "lletra", y el booleano valiendo false;
3. Si es vocal entonces llamar a escriu_linia con string valiendo "vocal", y el booleano valiendo true;
   si no llamar a escriu_linia con string valiendo "vocal", y el booleano valiendo false;
4. ...

Una vez tenemos esta versión, nos daremos cuenta de que alguna de las comprobaciones se repiten. Por ejemplo, para saber si es minúscula, repetimos parte de la condición para saber si es letra. Lo mismo pasa si queremos saber si es mayúscula. Cuando éste es el caso, vamos a evaluar la expresión una única vez y guardamos su resultado en una variable. De ese modo, ahora podemos consultar ese valor siempre que nos haga falta sin tener que volver a evaluar la misma expresión.

Pasos que dará nuestro programa (versión 2):

1. Leer el carácter de la entrada
2. Evalúa si el carácter es letra minúscula y lo almacena.

3. Evalúa si el carácter es letra mayúscula y lo almacena.

4. Evalúa si el carácter es letra gracias a la combinación almacenada de letra minúscula o letra mayúscula y lo almacena.

5. ...

6. Si es letra entonces llamar a escriu_linia con string valiendo "lletra", y el booleano valiendo true;
   si no llamar a escriu_linia con string valiendo "lletra", y el booleano valiendo false;

7. Si es vocal entonces llamar a escriu_linia con string valiendo "vocal", y el booleano valiendo true;
   si no llamar a escriu_linia con string valiendo "vocal", y el booleano valiendo false;

8. ...

Con la estructura que te damos a continuación, piensa qué otros valores booleanos tendrías que almacenar (recuerda que sólo hay que almacenar el resultado de aquellas expresiones que utilizas más de una vez).

```
int main() {
    char c;
    cin >> c;
    bool minuscula = 'a' <= c and c <= 'z';
    bool mayuscula = 'A' <= c and c <= 'Z';
    bool letra = minuscula or mayuscula;
    ...
    if (letra) escriu_linia(c, "lletra", true);
    else escriu_linia(c, "lletra", false);
    ...
}
```