Soluciones OIFem II Nivel 1

Concurso 1

Aprendiendo a contar

Teoría

- bucles

Solución

Leemos el número de entrada n e imprimimos los números del 1 a n con ayuda de un bucle.

Código

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

int main ()
{
  int n;
  cin>>n;
  for(int i=1;i<=n;i++)cout<<i<<"\n";
  return 0;
}
```

Hola

Solución

Primero leemos el número de casos `t`. Para cada caso, leemos la palabra que debemos decidir si es un anagrama de la palabra "hola".

Para ser un anagrama de "hola", la palabra debe tener exactamente las mismas letras, y por lo tanto, sabemos que si la palabra tiene un número diferente de 4 letras, entonces **no** será un anagrama.

Cuando la palabra tiene exactamente 4 letras, debemos mirar si estas letras son las mismas que 'h', 'o', 'l', 'a'. Para hacerlo, miramos si entre las cuatro letras hay una que es igual a 'h', otra que sea igual a 'o', otra igual a 'l', y por último la 'a'.

Código

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

int main()
{
    int t;
    cin >> t;
    string s;
    while (t--) {
        cin >> s;
        if (s.length() == 4) {
            int cont = 0;
            if (s[0] == 'h' or s[1] == 'h' or s[2] == 'h' or s[3] == 'h')
                ++cont;
            if (s[0] == 'o' or s[1] == 'o' or s[2] == 'o' or s[3] == 'o')
                ++cont;
            if (s[0] == 'l' or s[1] == 'l' or s[2] == 'l' or s[3] == 'l')
                ++cont;
            if (s[0] == 'a' or s[1] == 'a' or s[2] == 'a' or s[3] == 'a')
                ++cont;
            if (cont == 4) cout << "SI\n";
            else cout << "NO\n";
        } else {
            cout << "NO\n";
        }
    }
    return 0;
}
```

Konnichiwa OIE 2018

Solución

Primero leemos el número de casos n. Para cada caso se nos da el valor m de la moneda y se nos indica el tipo de moneda. Los euros los pasamos a yenes multiplicando por 130 y los yenes a euros dividiendo entre 130. Por último, tenemos que imprimir los valores nuevos y su correspondiente tipo de moneda.

Código

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

int main ()
{
  int n,m;
  string l;
  cin>>n;
  while(n--){
    cin>>m>>l;
    if(l=="E"){
```

```
        cout<<m*130<<" Y"<<"\n";
      }
      else{
        cout<<m/130<<" E"<<"\n";
      }
    }
    return 0;
}
```

Lomo queso

Solución

Leemos n, el número de personas que han pedido un lomo queso. Entonces, como cada lomo queso tiene dos lomos, necesita cocinar $2 \cdot n$ lomos. Se pueden cocinar 4 filetes a la vez.

Si n es un número par, entonces se necesitan n rondas para cocinar los filetes por los dos lados. En una ronda se cocinan 4 caras de filete, y en dos rondas se cocinan 4 filetes enteros. Como se necesitan $2n$ filetes, habrá $\frac{2n \text{ filetes}}{4 \text{ filetes}} \cdot 2 \text{ rondas} = n \text{ rondas}$. Cada ronda son 20 segundos que tarda una cara de filete en cocinarse. Luego, se tardará $n \cdot 20$ **segundos** en total.

En cambio, si $n \geq 3$ es un número impar, el argumento que hemos seguido para el caso n par no es el óptimo porque al final hay dos rondas en las que solo se utilizan 2 espacios para cocinar (que son del bocadillo de la última persona porque son impares). Entonces, el algoritmo óptimo es usar siempre todos los espacios disponibles. Observamos que en 3 rondas se pueden cocinar 6 filetes enteros de la siguiente manera:

- 1a ronda: cocinamos una cara de los filetes $1, 2, 3, 4$.
- 2a ronda: cocinamos una cara de los filetes $1, 2, 5, 6$.
- 3a ronda: cocinamos una cara de los filetes $3, 4, 5, 6$.

Así pues, con 3 rondas hemos cocinado 6 filetes. Entonces, $n-3$ es un número par, y podemos aplicar el argumento para el caso par. Luego, en total necesitamos $2n$ filetes que se cocinan en $(n-3) \cdot 20 + 3 \cdot 20 = n \cdot 20$ **segundos**, ya que para $(n-3)$ personas y $2 \cdot (n-3)$ filetes se tarda $(n-3) \cdot 20$, y para 3 personas y 6 filetes se tarda $3 \cdot 20$ porque son 3 rondas.

El caso $n = 1$ no se incluye en el caso n impar porque se necesitan 40 **segundos** para cocinar los filetes: 20 segundos cada cara de los dos filetes que tenemos que hacer.

Código

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

int main() {
    int n;
    while (cin >> n) {
        if (n != 1) {
            cout << 20*n << '\n';
        } else {
            cout << 40 << '\n';
```

```
        }
    }
    return 0;
}
```

Palabras abcd

Teoría

- bucles
- string

Solución

Para cada caso contemplamos un string con n elementos. Inicializamos cuatro contadores cona=0,conb=0,conc=0,cond=0, que contarán cuantas veces aparece cada letra en el substring que estemos contemplando.Además inicializamos buena=0, que contará el número de substrings buenos. Iremos substring a subring con un for. Añadimos el elemento nuevo s[i] al substring sumando uno al corresponiente contador y eliminamos el elemento s[i-4](,cuando i mayor igual que 4,) del substring, restando uno al contador correspondiente. Si todos los contadores son uno significa que el substring contemplado tiene las 4 letras diferentes y por tanto podemos sumar uno a buena. Al final devolvemos el valor de buena para cada caso.

Código

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

int main ()
{
  int t,n;
  string s;
  cin>>t;
  while(t--){
    cin>>n>>s;
    int buena=0;
    int cona=0,conb=0,conc=0,cond=0;
    for(int i=0;i<n;i++){
      if(i>=4){ //ya no nos interesa el primer elemento del substring anterior,
      ↪  así que restamos su valor
        if(s[i-4]=='a')cona--;
        else if(s[i-4]=='b')conb--;
        else if(s[i-4]=='c')conc--;
        else cond--;
      }
        if(s[i]=='a')cona++;//añadimos el valor del nuevo elemento
        else if(s[i]=='b')conb++;
        else if(s[i]=='c')conc++;
        else cond++;
        if(cona==conb && conc==conb && conc==cond)buena++;//añadimos uno al
        ↪  contador cuando todas las letras aparecen una vez
    }
```

```
      cout<<buena<<"\n";
    }
    return 0;
}
```

Números vecinos
Solución

Debemos computar el máximo entre dos números enteros consecutivos de una secuencia dada. Además, como nos dicen que la secuencia acaba en 0, el cual no forma parte de la secuencia, entonces debemos comprobar que los números que leamos sean diferentes de 0.

La variable `sum` contiene el valor de la suma de los dos números consecutivos que se están considerando en ese momento. Por eso restamos el valor de `first` al final del bucle y sumamos `second` al principio ya que estamos actualizando su valor para los siguientes dos números consecutivos. Se puede visualizar como una ventana deslizante con dos números: para calcular la suma actual, debemos restar el que ya no estará en la ventana, y sumar el nuevo número que entra.

Entonces, comparamos la suma actual `sum` con el valor de la máxima suma hasta el momento `max`.

Nota: Este problema también se podría implementar usando vectores.

Código

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

int main()
{
    int first = 0, second, sum = 0, max;
    cin >> first;
    if (first != 0 and cin >> second and second != 0) {
        max = first + second;
        sum = second;
        first = second;

        while (cin >> second and second != 0) {
            sum += second;

            if (sum > max)
                max = sum;

            sum -= first;
            first = second;
        }
        cout << max << '\n';
    }
}
```