Ejercicio 1

Frente a Zoe hay tres casas de diferentes colores. Se sabe junto a qué color de casa se vio por última vez a `Fluffy`. Las casas están numeradas 1, 2 y 3. Determina el número de la casa con la que necesita hablar sobre su mascota, `Fluffy`.

Formato de datos de entrada y salida
Se pasan cuatro colores a la entrada estándar: los colores de las tres casas y el color de la casa junto a la que se vio a `Fluffy`. Los colores están separados por un solo espacio. Se garantiza que las tres casas tienen colores diferentes.
La línea `Enter the house X` debe imprimirse en el flujo de salida estándar, donde `X` es el número de la casa: 1, 2 o 3, junto a la cual se vio a Fluffy. Si no hay una casa del color deseado entre las casas, debe imprimir la línea: `Seek somewhere else`. - Buscar en otro lugar.
Después de la frase, imprima el carácter de fin de línea.

Ejemplos de datos de entrada y salida

- **Entrada:** red green blue red
- **Salida**: Enter the house 1
- **Entrada:** white blue yellow yellow
- **Salida**: Enter the house 3
- **Entrada:** red green blue black
- **Salida**: Seek somewhere else

Ejercicio 2

Se aprendio a comprobar que los objetos son iguales. Para averiguar que no son iguales, se utiliza el operador `!=`.

Escriba un programa que pruebe la capacidad del usuario para sumar números enteros. En su entrada se introducen tres enteros: dos términos y una suma. Si la suma ingresada por el usuario es incorrecta, imprima la cadena `“Error. The sum of 5 and 3 is 9”`. En lugar de X, Y y Z, se deben mostrar los dos términos y su suma. De lo contrario, no emita

nada.

Formato de datos de entrada y salida

De la entrada estándar, el programa recibe tres enteros en el rango de -1'000'000 a +1'000'000, entre los cuales hay un espacio. La cadena de salida debe terminar con un carácter de fin de línea.

Ejemplos de datos de entrada y salida

- **Entrada**: 1 2 3
- **Salida**:
- **Entrada**: 5 3 10
- **Salida**: Error. The sum of 5 and 3 is 9
- **Entrada**: -5 2 -3
- **Salida**:
- **Entrada**: 8 -10 2
- **Salida**: Error. The sum of 8 and -10 is -2

Ejercicio 3

Se aprendio a comparar objetos desiguales. Con objetos iguales, solo el operador de comparación es `==`.

Escriba un código que compare las mitades de una naranja. El programa lee dos números reales: los tamaños de las mitades. Si los números son iguales, el programa debería mostrar: `La naranja se divide en dos partes iguales`. Si el primer número es mayor, haga que el programa muestre: `Cortar X de la primera parte`, donde `X` es la diferencia entre los pedazos de la naranja (Note que esta diferencia debe ser positiva). Si el segundo número es mayor, generar `Cortar X de la segunda parte`. Reemplace `X` con la diferencia de tamaño.

Teoria (while)

Para que el programa realice una acción mientras existe alguna condición, se utiliza un bucle `while`.

```
while (hay en la heladera torta) {
    ir a la heladera;
}
```

Digamos que el usuario ingresa el número `n`. Entonces, con un bucle `while`, puede calcular la suma de números de `1` a `n`.

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

int main() {
    int n;
    cin >> n;
    int sum = 0;
    int i = 1;
    while (i <= n) {
        sum += i;
        ++i;
    }
    cout << sum << endl;
}
```

El siguiente programa usa un ciclo `while` para encontrar el `mcd`, el máximo común divisor de dos enteros positivos `a` y `b`. El cuerpo del ciclo se repite hasta que una de las variables `a` o `b` se vuelve cero. La segunda variable contendrá el valor mcd:

```
#include <iostream>

using namespace std;

int main() {
    int a, b;
    cin >> a >> b;
    // Los numeros a y b deben ser naturales
    while (b != 0) {
        int c = b;
        b = a % b;
        a = c;
    }
    cout << a << endl;
}
```

Teoria (do-while)

El ciclo `while` tiene su analogo `do-while`:

```
do {
    ir a la heladera;
} while (hay en la heladera torta);
```

En un bucle `do-while`, ejecuta el cuerpo del bucle al menos una vez, ya que al final se cumple la condición de continuar el bucle. En un bucle `while`, la condición de continuación del bucle se comprueba desde el principio, por lo que es posible que el cuerpo del bucle no se ejecute ni una sola vez si la condición es inicialmente `falsa`.

Mire el fragmento de código de un juego interactivo en el que el usuario adivina un número:

```
int secret = 5; // el numero secreto
int answer;     // respuesta del usuario
do {
    cout << "Guess the number: "s << endl;
    cin >> answer;
} while (answer != secret);
cout << "You are right!"s << endl;
```

El usuario ingresa un número y el código repite la operación hasta que se adivina el número. Si el usuario adivina el número, el programa le dirá: `“You are right!”`.
Una vez que comprenda el ciclo `while`, puede pasar al ciclo `for`.

Ejercicio 4

Aquí está el código del juego. El jugador debe ingresar un número mientras intenta adivinarlo. La respuesta correcta es 5. Si se adivina el número 5, el programa muestra `You are right!`. Si el usuario comete un error, el programa muestra `Guess the number:`
.
Pero hay un error en el código, por lo que el programa no funcionará correctamente. Encuentre el error y corríjalo.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    int a = 5;
    int b;
    do {
```

```
        cout << "Guess the number: "s << endl;
        cin >> b;
    } while (a = b);
    cout << "You are right!"s << endl;
}
```

Ejercicio 5

Escriba una programa que imprima dos números ingresados por el usuario desde el primero hasta el segundo inclusive. En lugar de un ciclo `while`, use `for`.

Restricciones
Los números introducidos son no negativos, es decir, mayores o iguales a 0. El primer número siempre es menor o igual que el segundo.

Entrada

```
0 5
```

Salida

```
0
1
2
3
4
5
```

Ejercicio 6

Usando la solucion del ejercicio 5: imprimir todos los números entre los dos dados. Modifique lo pero ahora el primer número puede ser mayor que el segundo. En este caso, debe imprimir los números en orden inverso.
Entrada

```
-1 2
```

Salida

```
-1
0
1
2
```

Entrada

```
2 -1
```

Salida

```
2
1
0
-1
```

Ejercicio 7

Para buscar documentos, debe poder dividir el texto en palabras. Inicialmente, lo haremos de una forma no estandar: escriba su propio algoritmo.
Las palabras están separadas por espacios. Se garantiza exactamente un espacio entre palabras. La línea comienza con una palabra (no un espacio) y termina con el final de la línea (`\n`). Solo se ingresa una línea, el final de la línea también es uno, y alguna palabra siempre termina antes.
Lea toda la línea desde la terminal hasta una nueva línea. Para cada palabra, imprima el índice después del último carácter de la palabra. Recuerde, la indexación comienza desde cero.

Restricciones
Hay un espacio entre las palabras. No hay espacios al principio o al final de la cadena. Hay exactamente una línea en la consulta.

Entrada

```
green parrot
```

Salida

```
5
12
```

Los símbolos con estos índices están precedidos por las palabras verde y loro, respectivamente. (Cuente letra por letra comenzando desde 0 y verifique el resultado mostrado arriba)

Codigo Inicial

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main() {
    string query;
    getline(cin, query);

        // Su codigo aqui
}
```

Ejercicio 8

Ahora escriba las palabras encerradas entre corchetes. Por ejemplo, si la entrada fue un `green parrot`, la salida debería ser:

```
[green]
[parrot]
```

Puedes reutilizar el algoritmo del problema anterior. Allí ibas a lo largo de la línea y verificaste si el caracter era un espacio. Si el carácter no era un espacio, simplemente lo saltabas y seguía adelante. Ahora este caso tendrá que ser procesado.

Cree una variable de cadena, `word`, en la que agregará carácter por carácter de la cadena original, si este carácter no es un espacio. Si se encuentra un espacio, muestre `[`, la palabra acumulada y `]` y asigne una cadena vacía a la variable, `word`, para reiniciar el proceso con la siguiente palabra de la cadena.

Restricciones
Lo mismo que en el ejercicio anterior:

- Hay un espacio entre las palabras.
- Una línea comienza con una palabra y termina con el final de una línea.
- Hay exactamente una línea en la consulta.

Teoria (ejercicio 9 - 10)

A veces hay situaciones en las que al programador le gustaría detener la iteración actual del bucle y pasar al siguiente paso. Para decirle al programa que `"vaya al siguiente paso en el ciclo"`, use la instrucción `continue`. En el siguiente ejemplo, el ciclo debe ejecutarse `num_iters` veces. Al comienzo de cada iteración, el usuario ingresa los índices de caracteres de la cadena `str` desde el teclado. El programa debe comparar los caracteres en los índices especificados y mostrar el resultado en la pantalla. Pero si la entrada no es válida, es decir, los índices ingresados son negativos o mayores que la longitud de `str`, entonces no se puede comparar nada y solo necesita pasar a la siguiente iteración del ciclo:

```
string str = "Drawing indices for fun and profit"s;
int num_iters = 0;

// leemos cuantas veces quisieramos repetir el ciclo
cin >> num_iters;

for (int i = 0; i < num_iters; ++i) {
    int index1, index2;
    // pedimos los indices al usuario
    cin >> index1 >> index2;

    // si index1 negativo o mayor que la longitud de nuestra string
    // entonces continuar con este paso del ciclo no es posible
    if (index1 < 0 || index1 >= str.size()) {
        // al ordenar continue, el programador pide pasar a la
        // siguiente iteración (paso) sin terminar la actual
        continue;
    }

    // la misma logica para el index2
    if (index2 < 0 || index2 >= str.size()) {
        continue;
    }
    // muestra el resultado de comparar los caracteres de la cadena
    // en los índices especificados
    cout << (str[index1] == str[index2]) << endl;
}
```

Este código sin `continue` seria muy difícil de manejar.

Pero a veces se desea no solo pasar a la siguiente iteración, sino salir completamente del ciclo. Para finalizar un ciclo antes de tiempo, use la instrucción `break`. Así es como se verá un programa que imprime el índice de la primera letra `a` en la palabra ingresada. En este caso, la línea `Yes!` se imprimirá en cualquier caso, si se encuentra la letra `a` y si no.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string animal;

    // leemos el nombre del animal
    cin >> animal;
```

```
    for (int i = 0; i < animal.size(); ++i) {
        // si la letra actual de la string es a,
        if (animal[i] == 'a') {
            // entonces mostramos el indice i en la pantalla
            // y se termina el ciclo
            cout << i << endl;
            break;
        }
    }

    cout << "Yes!"s << endl;
}
```

Ejercicio 9

Según una encuesta realizada por `Most Accurate Statistics Bureau`, los encuestados hacen un promedio de 10 visitas al refrigerador por noche. Solo una circunstancia puede detenerlos: `Ya se acabo la torta de la heladera`.

Escriba un programa que simule las aventuras nocturnas del ciudadano. Puede averiguar si hay un pastel leyendo el número de `cin`. Solo tomará dos valores:

- 1 - todavía hay torta,
- 0 - se acabó la torta.

Si se encuentra un pastel, imprima la cadena `Om-nom-nom :P`. Si se acabó el pastel, escriba `Sin pastel :(` y finalice el ciclo con una declaración de `break`. Tambien, debe ignorarse ingresar más de 10 unidades, ya que el ciudadano promedio ya se ha quedado sin pastel en este caso.

Ejemplos
Entrada

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Salida

```
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
```

```
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
```

Cuente la cantidad de `1` en esta entrada y la cantidad de `Om-nom-nom :P`.

Entrada

```
1 1 1 0 0 0 0 0 1 1
```

Salida

```
Om-nom-nom :P
Om-nom-nom :P
Om-nom-nom :P
No cake :(
```

Entrada

```
0
```

Salida

```
No cake :(
```

Codigo Inicial

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

int main() {
    // Escriba un ciclo for, que realice 10 idas a la heladera
    // Si la torta no se encuentra escriba el siguiente texto:
    cout << "No cake :(" << endl;

    // Pero si se encuentra, entonces:
    cout << "Om-nom-nom :P" << endl;
}
```

Ejercicio 10

El operador de `continue` se usa a menudo justo al comienzo de un ciclo para eliminar casos innecesarios. Le permite simplificar el código y eliminar el largo `if`.
En el código - un programa para calcular el beneficio neto de la empresa. Este lee

ingresos y gastos de `cin`, pero suma solo cantidades positivas: ingresos. Todas las cantidades negativas se ignoran.
El programa funciona muy bien, pero resultó que la persona que revisó el código no conoce la declaración de `continue`, por lo que el cliente exige que se elimine.
¡Qué se puede hacer, la palabra del cliente es la ley! Empeora el código de tu programa sin cambiar su comportamiento deshaciéndote de la instrucción `continue`.

Es decir, el programa debe realizar lo mismo pero ahora sin utilizar la palabra clave `continue`.

Codigo inicial

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

int main() {
    int bill;
    int sum = 0;

    do {
        cin >> bill;
        // Esta construccion del if y continue nos permite ignorar casos
innecesarios
        if (bill <= 0) {
            continue;
        }

        cout << "Income: " << bill << endl;
        sum += bill;

    // la señal del final del programa será una cuenta de cero
    } while (bill != 0);

    cout << "Total income: " << sum << endl;
}
```

Ejercicio 11

Te dan una base y un exponente. Tu tarea es calcular la potencia del número. En donde la entrada es una sola linea con dos numeros separados por espacios. El primer numero sera siempre la base y el segundo el exponente.

Entrada

```
2 3
```

Salida

```
8
```

Ejercicio 12

Suponga que tiene un número `no negativo` y tiene que calcular su factorial. Este numero es solicitado al usuario con el flujo de entrada `cin`.

Restricciones
El usuario puede equivocarse e introducir un numero negativo! Su codigo debe lidiar con este caso e indicarle al usuario que esto no es permitido y que lo intente de nuevo.

Si se introduce un numero muy grande el programa debe advertirle al usuario que el resultado se demorara en ser calculado y pedirle que lo intente de nuevo.

Entrada

```
5
```

Salida

```
120
```