PRÁCTICA DE LABORATORIO Nº 01

COMPLEJIDAD TEMPORAL

Materiales:

- Utilizar Dev-C++.
- Los archivos arreglo.cpp y punteros.cpp se encuentran disponibles aquí.

Objetivos:

- Resolver problemas utilizando **arreglos en memoria dinámica**.
- Entender la complejidad temporal de las soluciones.

Indicaciones para el envío

- Para los problemas 1,2 y 3, comentar en la **primera línea** la complejidad temporal. Ejemplo: //O(n^2)
- Enviar en 1 solo archivo .zip con los archivos código fuente **1.cpp, 2.cpp, 3.cpp**.
- SOLO se deberá imprimir la salida especificada en el problema.

0. Repaso de memoria estática, memoria dinámica y punteros.

Ejecute el código del archivo arreglo.cpp en DevC++:

```
1  #include<iostream>
2  #define MAX_SIZE 518021
3
4  using namespace std;
5
6  int main()
7  {
8      int arreglo[MAX_SIZE];
9      arreglo[MAX_SIZE-1]=1;
10     cout<<arreglo[MAX_SIZE-1];
11     return 0;
12 }
```

¿Qué pasó? Cambie la directiva DEFINE en la línea 2 para que MAX_SIZE sea 518020.

¿Qué pasó? Este comportamiento es debido a que el compilador **solo puede reservar un espacio de memoria limitado en memoria estática**. Este espacio depende del compilador, procesador, espacio disponible en la pila, etc. Si el comportamiento no es el esperado, pruebe con otros valores dividiendo o multiplicando por 2.

Entonces, ¿**Cómo definir un arreglo de tamaño superior**?

Para ello, necesitamos acceder a la **memoria dinámica**, que se aloja en el heap. Además, este espacio de memoria **puede ser reservado en tiempo de ejecución**, sin embargo, debemos tener cuidado debido a que podríamos llenar este espacio, por lo que será necesario liberarlo también.

Para la administración de memoria dinámica en C++, se cuenta con 2 operadores ***new (reserva espacio)*** y ***delete (libera espacio)***.

Para esto, será necesario el uso de punteros (una variable que almacena una dirección de memoria).

A continuación, se presentan algunos fragmentos del código de punteros.cpp y los resultados de su ejecución, deberá interpretar lo que está haciendo el código y el porqué de los resultados:

Código	Resultado
Variable y Dirección de Memoria: Se declara variable en memoria estática, luego, usando el operador de dirección & para imprimir la dirección de memoria en la que se encuentra la variable. El valor de dirección mostrado podría ser diferente en su ordenador y ser diferente en cada ejecución.	

`int variable = 1234;`
 	
`cout<<'\n'<<"int variable = 1234"<<'\n';`
`cout<<'\t'<<"variable: "<<variable<<'\n';`
`cout<<'\t'<<"&variable: "<<&variable<<'\n';`
	`int variable = 1234`
	`variable: 1234`
	`&variable: 0x6ffde4`

Puntero y asignación a dirección de memoria: Se declara un puntero, luego, se asigna al puntero la dirección de variable (los punteros almacenan direcciones de memoria). Luego, al imprimir puntero nos mostrará la dirección. Para acceder a la información dentro de dicha dirección será necesario usar el operador de referencia de valor *.	

`int *puntero;`
 	
`cout<<'\n'<<"int *puntero;"<<'\n';`
 	
`puntero = &variable;`
 	
`cout<<'\n'<<"puntero = &variable;"<<'\n';`
`cout<<'\t'<<"puntero: "<<puntero<<'\n';`
`cout<<'\t'<<"*puntero: "<<*puntero<<'\n';`
`cout<<'\t'<<"&*puntero: "<<&*puntero<<'\n';`
`cout<<'\t'<<"&puntero: "<<&puntero<<'\n';`
	`int *puntero;`
	
	`puntero = &variable;`
	`puntero: 0x6ffde4`
	`*puntero: 1234`
	`&*puntero: 0x6ffde4`
	`&puntero: 0x6ffdd8`

Puntero y modificación en variable: Al hacer uso del operador de referencia de valor *, podemos cambiar el valor de la variable a la cual está apuntando el puntero.	

`*puntero = 4321;`
 	
`cout<<'\n'<<"*puntero = 4321"<<'\n';`
`cout<<'\t'<<"puntero: "<<puntero<<'\n';`
`cout<<'\t'<<"*puntero: "<<*puntero<<'\n';`
`cout<<'\t'<<"variable: "<<variable<<'\n';`
	`*puntero = 4321`
	`puntero: 0x6ffde4`
	`*puntero: 4321`
	`variable: 4321`

Modificación en variable y puntero: Si realizamos el cambio en la variable a la cual está apuntando un puntero, podemos ver que el cambio también se ve reflejado cuando usamos el operador de referencia de valor * en el puntero.	

`variable = 1234;`
 	
`cout<<'\n'<<"variable = 1234"<<'\n';`
`cout<<'\t'<<"puntero: "<<puntero<<'\n';`
`cout<<'\t'<<"*puntero: "<<*puntero<<'\n';`
`cout<<'\t'<<"variable: "<<variable<<'\n';`
	`variable = 1234`
	`puntero: 0x6ffde4`
	`*puntero: 1234`
	`variable: 1234`

Arreglo como puntero:

Si declaramos un arreglo en memoria estática, nos daremos cuenta que dicho arreglo en realidad es un puntero. Si asignamos a puntero el valor de arreglo, no es necesario usar el operador de dirección &. Además, si imprimimos arreglo obtendremos una dirección de memoria, que es igual a la dirección de memoria de arreglo[0]. Luego, podemos usar aritmética de punteros, donde arreglo[x]=*(puntero+x).

```
int arreglo[10] = {0,2,4,6,8,10,12,14,16,18};
cout<<'\n'<<"int arreglo[10] = {0,2,4,6,8,10,12,14,16,18}"<<'\n';

puntero = arreglo;
cout<<'\n'<<"puntero = arreglo"<<'\n';
cout<<"arreglo: "<<arreglo<<'\n';
cout<<"&arreglo[0]: "<<&arreglo[0]<<'\n';
for(int i = 0; i < 10; i++)
{
    cout<<'\t'<<"*(puntero+"<<i<<"): "<<*(puntero+i);
    cout<<'\t'<<"puntero+"<<i<<": "<<puntero+i<<'\n';
}
```

```
int arreglo[10] = {0,2,4,6,8,10,12,14,16,18}

puntero = arreglo
arreglo: 0x6ffda0
&arreglo[0]: 0x6ffda0
        *(puntero+0): 0 puntero+0: 0x6ffda0
        *(puntero+1): 2 puntero+1: 0x6ffda4
        *(puntero+2): 4 puntero+2: 0x6ffda8
        *(puntero+3): 6 puntero+3: 0x6ffdac
        *(puntero+4): 8 puntero+4: 0x6ffdb0
        *(puntero+5): 10        puntero+5: 0x6ffdb4
        *(puntero+6): 12        puntero+6: 0x6ffdb8
        *(puntero+7): 14        puntero+7: 0x6ffdbc
        *(puntero+8): 16        puntero+8: 0x6ffdc0
        *(puntero+9): 18        puntero+9: 0x6ffdc4
```

Puntero y variable en memoria dinámica

Con el operador new podemos declarar una variable en memoria dinámica. El comportamiento es similar al estudiando en los ejemplos anteriores, excepto que solo tenemos a puntero señalándonos dicha dirección, por lo que debemos tener mucho cuidado en no perder el valor de puntero.

```
puntero = new int(123);
cout<<'\n'<<"puntero = new int(123)"<<'\n';
cout<<"*puntero: "<<*puntero<<'\n';
cout<<"puntero: "<<puntero<<'\n';
delete puntero;
```

```
puntero = new int(123)
*puntero: 123
puntero: 0x1c1550
```

Puntero y arreglo en memoria dinámica:

Finalmente, podemos utilizar un puntero para declarar un arreglo en memoria dinámica y utilizar aritmética de punteros para acceder a sus valores.

```
int *array = new int[10]{0,2,4,6,8,10,12,14,16,18};
cout<<'\n'<<"int *array = new int[10]{0,2,4,6,8,10,12,14,16,18}"<<'\n';
for(int i = 0; i < 10; i++)
{
    cout<<'\t'<<"*(array+"<<i<<"): "<<*(array+i);
    cout<<'\t'<<"array+i"<<i<<": "<<array+i<<'\n';
}
delete [] array;
cout<<"delete [] array;"<<'\n';
```

```
int *array = new int[10]{0,2,4,6,8,10,12,14,16,18}
        *(array+0): 0   array+i0: 0x171570
        *(array+1): 2   array+i1: 0x171574
        *(array+2): 4   array+i2: 0x171578
        *(array+3): 6   array+i3: 0x17157c
        *(array+4): 8   array+i4: 0x171580
        *(array+5): 10  array+i5: 0x171584
        *(array+6): 12  array+i6: 0x171588
        *(array+7): 14  array+i7: 0x17158c
        *(array+8): 16  array+i8: 0x171590
        *(array+9): 18  array+i9: 0x171594
delete [] array;
```

Ahora, comprendes cómo funcionan los punteros y cómo declarar arreglos en memoria dinámica, donde tendrás acceso a toda la memoria disponible en tu ordenador.

A continuación, resuelve los siguientes problemas utilizando arreglos en memoria dinámica:

1. Conjetura de Collatz (https://es.wikipedia.org/wiki/Conjetura_de_Collatz) (5 puntos)

La conjetura de Collatz, conocida también como conjetura 3n+1 o conjetura de Ulam (entre otros nombres), fue enunciada por el matemático Lothar Collatz en 1937, y a la fecha no se ha resuelto:

Consideremos un algoritmo que toma como entrada un número entero positivo **n**.

Si **n** es par, el algoritmo lo divide por dos, y si **n** es impar, el algoritmo lo multiplica por tres y le añade uno. El algoritmo repite esta operación hasta que **n** sea uno. Por ejemplo, la secuencia para **n**=3 es la siguiente:

3→10→5→16→8→4→2→1

Tu tarea es simular la ejecución del algoritmo para un valor dado de **n**.

Entrada

La única línea de entrada contiene un número entero **n**.

Salida

Imprime una línea que contiene todos los valores de **n** durante el algoritmo.

Restricciones

$1 \leq \mathbf{n} \leq 10^6$

Complejidad Objetivo

O(n)

Ejemplo

Entrada	Salida
3	3 10 5 16 8 4 2 1

La conjetura dice que siempre alcanzaremos el 1. Sin embargo no ha podido ser demostrada matemáticamente aún. En mayo de 2020 se comprobó la conjetura para todas las secuencias de números menores que 2^{68}.

El premio por resolver la conjetura es de 900 mil dólares (120 millones de yenes).

2. Repeticiones (6 puntos)

Se le da una secuencia de ADN: una cadena formada por los caracteres A, C, G y T. Su tarea consiste en encontrar la repetición más larga de la secuencia. Se trata de una subcadena de longitud máxima que sólo contiene un tipo de carácter.

Entrada

La única línea de entrada contiene una cadena de **n** caracteres.

Salida

Imprime un entero: la longitud de la repetición más larga.

Restricciones

$1 \leq n \leq 10^6$

Complejidad Objetivo

O(n)

Ejemplo

Entrada	Salida
ATTCGGGA	3

3. Array creciente (9 puntos)

Se le da un array de **n** enteros. Quieres modificar el array para que sea creciente, es decir, que cada elemento sea al menos tan grande como el elemento anterior.

En cada movimiento, puede aumentar el valor de cualquier elemento en uno. ¿Cuál es el número mínimo de movimientos necesarios?

Entrada

La primera línea de entrada contiene un número entero **n**: el tamaño del array.

A continuación, la segunda línea contiene **n** enteros $\mathbf{x_1,x_2,...,x_n}$: el contenido del array.

Salida

Imprime el número mínimo de movimientos.

Restricciones

$1 \leq n \leq 2 \cdot 10^5$

$1 \leq x_i \leq 10^9$

Complejidad Objetivo

O(n)

Ejemplo

Entrada	Salida
5 3 2 5 1 7	5