

MANUAL DE LABORATORIO

ESTRUCTURA DE DATOS

ESIS - FAIN – UNJBG

AUTOR: ING. ISRAEL NAZARETH CHAPARRO CRUZ

Presentación

El presente manual de laboratorio contiene 40 páginas efectivas de prácticas de laboratorio y 29 páginas de código. Las prácticas de laboratorio corresponden a los temas de Arreglos, Matrices, Listas Enlazadas, Pilas, Colas y Árboles de Búsqueda Binaria del curso de Estructura de Datos turno mañana del semestre académico 2022-I. Cada práctica de laboratorio incluye código básico para empezar a trabajar en la implementación de las estructuras de datos, seguida por los enunciados de los problemas y la implementación en C++ de los aspectos básicos del tema y/o implementación completa de las operaciones básicas. Los enunciados de los problemas son una recopilación que ha tenido como fuente principal la plataforma educativa gratuita <https://omegapl.com/> que cuenta con casi 2 millares de enunciados, de los cuales se ha seleccionado cuidadosamente aquellos pertinentes para el curso y se los ha presentado en las prácticas aumentando cada vez el nivel de dificultad. Adicionalmente se han utilizado algunos enunciados de la *librería de material educativo en Ciencia de la Computación de la Stanford University* (disponible en: <http://cslibrary.stanford.edu/>). Los problemas de nivel básico a medio-alto puntúan el 100% de la práctica, los problemas de nivel avanzado a extremo están allí para aquellos alumnos que deseen conquistarlos y resolverlos. Al respecto, ha habido una cantidad de alumnos considerable que se ha volcado sobre estos ejercicios pese a ya haber obtenido el máximo puntaje de la práctica, lo que demuestra el buen apetito y gusto por los algoritmos de los alumnos y la necesidad de tener un manual de laboratorio curado de problemas de estructuras de datos. Una versión digital puede ser encontrada en <https://github.com/ichaparroc/IA-UNJBG-2022/blob/main/README.md>

El Autor

PRÁCTICA/LABORATORIO N° 01

ARREGLOS UNIDIMENSIONALES Y BIDIMENSIONALES EN MEMORIA DINÁMICA

Objetivos:

- Utilizar [Dev-C++](#)
- Experimentar con punteros y variables en memoria estática.
- Comprobar que un arreglo/matriz es lo mismo que un puntero al primer elemento del arreglo/matriz.
- Utilizar punteros para manejar variables en memoria dinámica
- Resolver problemas utilizando memoria dinámica.

0. Ejercicios con punteros

A continuación, se presentan algunos fragmentos de código 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:	
<pre>int variable = 1234; cout<<'\n'<<"int variable = 1234"<<'\n'; cout<<'t'<<"variable: "<<variable<<'\n'; cout<<'t'<<"&variable: "<<&variable<<'\n';</pre>	<pre>int variable = 1234 variable: 1234 &variable: 0x6ffdde4</pre>
Puntero y asignación a dirección de memoria	
<pre>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';</pre>	<pre>int *puntero; puntero = &variable; puntero: 0x6ffdde4 *puntero: 1234 &puntero: 0x6ffdde4 &puntero: 0x6ffdd8</pre>
Puntero y asignación a variable	
<pre>*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';</pre>	<pre>*puntero = variable puntero: 0x6ffdde4 *puntero: 1234 &puntero: 0x6ffdde4 &puntero: 0x6ffdd8</pre>
Puntero y cambios en variable	
<pre>*puntero = 4321; cout<<'\n'<<"*puntero = 4321"<<'\n'; cout<<'t'<<"puntero: "<<puntero<<'\n'; cout<<'t'<<"*puntero: "<<*puntero<<'\n'; cout<<'t'<<"variable: "<<variable<<'\n';</pre>	<pre>*puntero = 4321 puntero: 0x6ffdde4 *puntero: 4321 variable: 4321</pre>
Cambios en variable y puntero	
<pre>variable = 1234; cout<<'\n'<<"variable = 1234"<<'\n'; cout<<'t'<<"puntero: "<<puntero<<'\n'; cout<<'t'<<"*puntero: "<<*puntero<<'\n'; cout<<'t'<<"variable: "<<variable<<'\n';</pre>	<pre>variable = 1234 puntero: 0x6ffdde4 *puntero: 1234 variable: 1234</pre>

Arreglo como puntero	
<pre>int arreglo[10] = {0,1,2,3,4,5,6,7,8,9}; cout<<'\n'<<"int arreglo[10] = {0,1,2,3,4,5,6,7,8,9}"<<'\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'; }</pre>	<pre>int arreglo[10] = {0,1,2,3,4,5,6,7,8,9} puntero = arreglo arreglo: 0x6ffd0 &arreglo[0]: 0x6ffd0 *puntero+0: 0 puntero+0: 0x6ffd0 *puntero+1: 1 puntero+1: 0x6ffd04 *puntero+2: 2 puntero+2: 0x6ffd08 *puntero+3: 3 puntero+3: 0x6ffdac *puntero+4: 4 puntero+4: 0x6ffdb0 *puntero+5: 5 puntero+5: 0x6ffdb4 *puntero+6: 6 puntero+6: 0x6ffdb8 *puntero+7: 7 puntero+7: 0x6ffdbc *puntero+8: 8 puntero+8: 0x6ffdc0 *puntero+9: 9 puntero+9: 0x6ffdc4</pre>
Matriz como puntero	
<pre>int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}}; cout<<'\n'<<"int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}}"<<'\n'; cout<<"matriz: "<<matriz[0]<<'\n'; cout<<"&matriz: "<<&matriz[0]<<'\n'; cout<<"*matriz[0]: "<<*matriz[0]<<'\n'; cout<<"&matriz[0][0]: "<<&matriz[0][0]<<'\n'; cout<<"*matriz[0][0]: "<<*matriz[0][0]<<'\n'; cout<<"&matriz[0][1]: "<<&matriz[0][1]<<'\n'; cout<<"*matriz[0][1]: "<<*matriz[0][1]<<'\n'; cout<<"&matriz[0][2]: "<<&matriz[0][2]<<'\n'; cout<<"*matriz[0][2]: "<<*matriz[0][2]<<'\n'; cout<<"&matriz[1]: "<<&matriz[1]<<'\n';</pre>	<pre>int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}} matriz: 0x6ffd70 &matriz: 0x6ffd70 matriz[0]: 0x6ffd70 &matriz[0]: 0x6ffd70 matriz[0][0]: 1 &matriz[0][0]: 0x6ffd70 &matriz[0][1]: 0x6ffd74 &matriz[0][2]: 0x6ffd78 &matriz[1]: 0x6ffd7c</pre>
Puntero sobre matriz	
<pre>puntero = matriz[0]; cout<<'\n'<<"puntero = matriz[0]"<<'\n'; for(int i = 0; i < 3; i++) { for(int j = 0; j < 3; j++) { cout<<'\t'<<"*puntero+<<j<<": "<<i<<")"; "<<*puntero+j+(i*3); cout<<'\t'<<"*puntero+<<j<<": "<<(i*3)<<"; "<<*puntero+j+(i*3)<<'\n'; } }</pre>	<pre>puntero = matriz[0] *puntero+0*(0*3): 1 puntero+0*(0*3): 0x6ffd70 *puntero+1*(0*3): 2 puntero+1*(0*3): 0x6ffd74 *puntero+2*(0*3): 3 puntero+2*(0*3): 0x6ffd78 *puntero+0+(1*3): 4 puntero+0+(1*3): 0x6ffd7c *puntero+1+(1*3): 5 puntero+1+(1*3): 0x6ffd80 *puntero+2+(1*3): 6 puntero+2+(1*3): 0x6ffd84 *puntero+0+(2*3): 7 puntero+0+(2*3): 0x6ffd88 *puntero+1+(2*3): 8 puntero+1+(2*3): 0x6ffd8c *puntero+2+(2*3): 9 puntero+2+(2*3): 0x6ffd90</pre>
Puntero y variable en memoria dinámica	
<pre>puntero = new int(123); cout<<'\n'<<"puntero = new int(123)"<<'\n'; cout<<"*puntero: "<<*puntero<<'\n'; cout<<"puntero: "<<puntero<<'\n'; delete puntero;</pre>	<pre>puntero = new int(123) *puntero: 123 puntero: 0x1c1550</pre>
Puntero y arreglo en memoria dinámica	
<pre>int *array = new int[10]{0,1,2,3,4,5,6,7,8,9}; int *array = new int[10]{0,2,4,6,8,10,12,14,16,18}; cout<<'\n'<<"*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<<": "<<array+i<<'\n'; } delete [] array; cout<<"delete [] array;"<<'\n';</pre>	<pre>int *array = new int[10]{0,1,2,3,4,5,6,7,8,9} *array+0: 0 array+10: 0x1c1550 *array+1: 1 array+11: 0x1c1554 *array+2: 2 array+12: 0x1c1558 *array+3: 3 array+13: 0x1c155c *array+4: 4 array+14: 0x1c1560 *array+5: 5 array+15: 0x1c1564 *array+6: 6 array+16: 0x1c1568 *array+7: 7 array+17: 0x1c156c *array+8: 8 array+18: 0x1c1570 *array+9: 9 array+19: 0x1c1574 delete [] array;</pre>
Puntero y matriz en memoria dinámica	
<pre>int filas = 3; int columnas = 3; int **matrix = new int*[filas]; for(int i = 0; i < filas; i++) { matrix[i] = new int[columnas]{1*(i+1),2*(i+1),3*(i+1)}; } cout<<'\n'<<"int **matrix = new int*[filas];"<<'\n'; cout<<"&matrix[0]: "<<&matrix[0]<<'\n'; cout<<"*matrix[0]: "<<*matrix[0]<<'\n'; cout<<"&matrix[0][0]: "<<&matrix[0][0]<<'\n'; cout<<"*matrix[0][0]: "<<*matrix[0][0]<<'\n'; cout<<"&matrix[0][1]: "<<&matrix[0][1]<<'\n'; cout<<"*matrix[0][1]: "<<*matrix[0][1]<<'\n'; cout<<"&matrix[0][2]: "<<&matrix[0][2]<<'\n'; cout<<"*matrix[0][2]: "<<*matrix[0][2]<<'\n'; cout<<"&matrix[1]: "<<&matrix[1]<<'\n'; cout<<"*matrix[1]: "<<*matrix[1]<<'\n'; for(int i = 0; i < filas; i++) { delete[] matrix[i]; } delete[] matrix;</pre>	<pre>int **matrix = new int*[filas]; for(int i = 0; i < filas; i++) matrix[i] = new int[columnas]{1*(i+1),2*(i+1),3*(i+1)}; matrix: 0x1c1550 &matrix: 0x6ffd0 matrix[0]: 0x1c1570 &matrix[0]: 0x1c1550 matrix[0][0]: 1 &matrix[0][0]: 0x1c1570 &matrix[0][1]: 0x1c1574 &matrix[0][2]: 0x1c1578 &matrix[1]: 0x1c1558</pre>

1. Carreteras

Descripción

En Perú hay 1000 ciudades, numeradas de 0 al 999, y algunas de esas ciudades están unidas por carreteras. Dadas todas las carreteras que hay en Perú, escribe un programa que determine si dos ciudades están conectadas directamente por una carretera.

Entrada y Salida

Un número **n** ($1 \leq n \leq 100,000$), indicando la cantidad de carreteras en Perú.

Las siguientes líneas contienen dos números **a** y **b**, indicando que hay una carretera conectando a la ciudad con la ciudad y viceversa.

La siguiente línea contiene un número **q** ($1 \leq q \leq 10^6$) que representa la cantidad de preguntas a contestar.

Le siguen **q** preguntas, cada una representada por dos números **s** y **e**. Por cada pregunta el programa debe imprimir "YES" si hay una carretera que conecta **s** con **e**. En caso contrario imprimir "NO".

Ejemplo

Entrada	Salida
3	YES
0 1	NO
3 1	
2 0	
2	
1 0	
5 9	

2. Omitiendo la intersección

Descripción

Dadas dos secuencias de **N** enteros, escribe un programa que reimprima las secuencias pero omitiendo los enteros que aparecen en ambas secuencias. El orden de aparición de los enteros debe conservarse.

Entrada

Un entero **N** seguido de dos líneas donde cada una tiene **N** enteros. Puedes suponer que $0 \leq N \leq 100,000$ y que los enteros de las secuencias están en el rango de -10^9 a $+10^9$.

Salida

Dos líneas que contengan las secuencias de enteros originales, pero omitiendo los enteros que aparecen en ambas secuencias.

Ejemplo

Entrada	Salida
5 1 2 3 2 -1 3 1 4 1 6	2 2 -1 4 6

3. Incrementando intervalos

Descripción

Cuentas con un arreglo de **N** enteros que valen inicialmente 0. Se te darán **M** intervalos con extremos y para cada uno deberás incrementar los elementos del arreglo que están de la posición **I** a la posición **F**, sin incluir esta última. ¿Qué valor tendrán los elementos del arreglo después de procesar todos los intervalos?

Entrada

Dos enteros **N**, **M** seguido de **M** parejas de enteros que denotan los extremos de cada intervalo. Puedes suponer que $0 \leq N, M \leq 100,000$ y que $0 \leq I \leq F \leq N$.

Salida

Los enteros del arreglo después de procesar todos los intervalos.

Ejemplo

Entrada	Salida
4	1 2 2 1
2	
1 4	
0 3	

4. Partiendo cadenas

Descripción

Tienes una cadena que has ido construyendo a lo largo de los años y ya es muy larga. Desafortunadamente, en tu ciudad acaban de promulgar una ley que obliga a todos los dueños de cadenas largas a reducir su tamaño. La política es que el dueño de la cadena se verá obligado a borrar todas las apariciones de un carácter de su elección (el carácter debe aparecer en la cadena, sino es trampa), lo cual tiene el efecto de partir la cadena en pedazos. Por ejemplo, si se borran todas las apariciones de "t" de la cadena "gatitos" entonces se forman las cadenas "ga", "i", "os" mientras que si se borran todas las apariciones de "a" se forman las cadenas "g" y "titos". Tu deseo es que, después de aplicar la operación, tengas una cadena del mayor tamaño posible. Escribe un programa que te ayude a conseguir tu objetivo.

Entrada

Una cadena de caracteres ASCII gráficos seguida de un salto de línea. Puedes suponer que la cadena tiene a lo mucho 10^7 caracteres.

Salida

Un entero que es el tamaño de la cadena más larga que puedes conservar después de aplicar la operación.

Ejemplo

Entrada	Salida	Descripción
a@aaaaaa!aaaaaa@a	11	<p>Si decides borrar las apariciones de "a", las cadenas resultantes son "@", "!" y "@".</p> <p>Si decides borrar las apariciones de "@", las cadenas resultantes son "a", "aaaaa!aaaaa" y "a".</p> <p>Si decides borrar las apariciones de "!", las cadenas resultantes son "a@aaaaaa" y "aaaaa@a".</p>
gatitos	6	

5. Triples

Descripción

Dada una secuencia de enteros positivos, se necesita encontrar el número de triples en la secuencia.

Un triple, se logra con tres enteros x, y, z de la secuencia tal que $x+y=z$, por ejemplo: 1, 2, 3 es un triple porque $1+2=3$. 3, 4, 5 no es un triple porque $3+4 \neq 5$.

Entrada

En la primera línea un entero N que representa la cantidad de números en la secuencia. En la segunda línea, N enteros positivos de 32 bits separados por espacios. Puedes suponer que $0 \leq N \leq 100,000$

Salida

Un entero seguido de un salto de línea que indique el número de triples en el conjunto.

Ejemplo

Entradas	Salidas
6 1 2 3 4 5 6	6
6 1 2 4 8 16 32	0
3 1000000000 2000000000 1000000000	1
5 1 1 1 2 2	6

6. El tablero de Bety

Descripción

Bety compro un tablero electrónico muy poco común. Este tablero tiene forma de cuadricula con **n** filas y **m** columnas. El tablero electrónico cuenta con 4 operaciones, "Up", "Down", "Left", "Right".

"Up" toma la primera fila, y la pone al abajo de la última fila. "Down" toma la última fila, y la pone arriba de la primera fila. "Left" toma la primera columna, y la pone después de la última columna". "Right" toma la última columna, y la pone antes de la primera columna.

Antes de empezar a jugar con el tablero, Bety quiere saber cómo quedara después de aplicar **k** operaciones.

Entrada

En la primera fila estaran los enteros **n**, **m** y **k**. ($0 < n, m < 100$; $0 < k < 1,000,000$)

Las siguientes **k** lineas, tendran un entero representando las operaciones que se aplicaran al tablero. "Up" sera representado con el entero 1, "Down" con el entero 2, "Left" con el entero 3 y "Right" con el entero 4.

Las siguientes **n** lineas tendran **m** enteros, las cuales representaran la configuración inicial del tablero.

Salida

Una matriz de tamaño **n*m** la cual representara la configuración final del tablero.

Ejemplo

Entradas	Salidas
3 3 1	4 5 6
1	7 8 9
1 2 3	1 2 3
4 5 6	
7 8 9	
3 3 1	2 3 1
3	5 6 4
1 2 3	8 9 7
4 5 6	
7 8 9	

7. Ordenando columnas

Descripción

Dada una matriz A de **m** filas y **n** columnas. Escribe un programa que imprima A con cada una de sus columnas ordenada de manera creciente.

Entrada

Dos enteros **m,n** que representan el número de filas y columnas de la matriz A. A continuación, los **m×n** enteros que representan cada una de las entradas de la matriz. Puedes asumir que cada entrada en la matriz es un entero cuyo valor está en el intervalo $[-100,100]$ y que $1 \leq m,n \leq 100$.

Salida

La matriz A con cada una de sus columnas ordenada de manera creciente.

Ejemplo

Entradas	Salidas
3 4 4 10 -1 2 -100 3 4 23 56 78 22 9	-100 3 -1 2 4 10 4 9 56 78 22 23
6 6 -84 54 49 -63 93 2 -87 -86 -31 -33 8 -100 -50 -36 -62 29 -47 -85 71 63 -2 30 29 31 -13 92 90 44 -31 32 86 -95 -33 27 -23 -19	-87 -95 -62 -63 -47 -100 -84 -86 -33 -33 -31 -85 -50 -36 -31 27 -23 -19 -13 54 -2 29 8 2 71 63 49 30 29 31 86 92 90 44 93 32

```
1: #include<iostream>
2:
3: using namespace std;
4:
5: int main()
6: {
7:     int variable = 1234;
8:
9:     cout<<'\n'<<"int variable = 1234"<<'\n';
10:    cout<<'\t'<<"variable: "<<variable<<'\n';
11:    cout<<'\t'<<"&variable: "<<&variable<<'\n';
12:
13:    int *puntero;
14:
15:    cout<<'\n'<<"int *puntero;"<<'\n';
16:
17:    puntero = &variable;
18:
19:    cout<<'\n'<<"puntero = &variable;"<<'\n';
20:    cout<<'\t'<<"puntero: "<<puntero<<'\n';
21:    cout<<'\t'<<"*puntero: "<<*puntero<<'\n';
22:    cout<<'\t'<<"&puntero: "<<&puntero<<'\n';
23:    cout<<'\t'<<"&puntero: "<<&puntero<<'\n';
24:
25:    *puntero = variable;
26:
27:    cout<<'\n'<<"*puntero = variable"<<'\n';
28:    cout<<'\t'<<"puntero: "<<puntero<<'\n';
29:    cout<<'\t'<<"*puntero: "<<*puntero<<'\n';
30:    cout<<'\t'<<"&puntero: "<<&puntero<<'\n';
31:    cout<<'\t'<<"&puntero: "<<&puntero<<'\n';
32:
33:    *puntero = 4321;
34:
35:    cout<<'\n'<<"*puntero = 4321"<<'\n';
36:    cout<<'\t'<<"puntero: "<<puntero<<'\n';
37:    cout<<'\t'<<"*puntero: "<<*puntero<<'\n';
38:    cout<<'\t'<<"variable: "<<variable<<'\n';
39:
```

```

40:     variable = 1234;
41:
42:     cout<<'\n'<<"variable = 1234"<<'\n';
43:     cout<<'\t'<<"puntero: "<<puntero<<'\n';
44:     cout<<'\t'<<"*puntero: "<<*puntero<<'\n';
45:     cout<<'\t'<<"variable: "<<variable<<'\n';
46:
47:     int arreglo[10] = {0,2,4,6,8,10,12,14,16,18};
48:     cout<<'\n'<<"int arreglo[10] = {0,2,4,6,8,10,12,
49:     14,16,18}"<<'\n';
50:     puntero = arreglo;
51:     cout<<'\n'<<"puntero = arreglo"<<'\n';
52:     cout<<"arreglo: "<<arreglo<<'\n';
53:     cout<<"&arreglo[0]: "<<&arreglo[0]<<'\n';
54:     for(int i = 0; i < 10; i++)
55:     {
56:         cout<<'\t'<<"*(puntero+<<i<<"):
57:             "<<*(puntero+i);
58:         cout<<'\t'<<"puntero+<<i<<":
59:             "<<puntero+i<<'\n';
60:     }
61:
62:     int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
63:     cout<<'\n'<<"int matriz[3][3] = {{1,2,3},{4,5,6},
64:     {7,8,9}}"<<'\n';
65:     cout<<"matriz: "<<matriz<<'\n';
66:     cout<<"&matriz: "<<&matriz<<'\n';
67:     cout<<"matriz[0]: "<<matriz[0]<<'\n';
68:     cout<<"&matriz[0]: "<<&matriz[0]<<'\n';
69:     cout<<"matriz[0][0]: "<<matriz[0][0]<<'\n';
70:     cout<<"&matriz[0][0]: "<<&matriz[0][0]<<'\n';
71:     cout<<"matriz[0][1]: "<<matriz[0][1]<<'\n';
72:     cout<<"&matriz[0][1]: "<<&matriz[0][1]<<'\n';
73:     cout<<"matriz[0][2]: "<<matriz[0][2]<<'\n';
74:     cout<<"&matriz[1]: "<<&matriz[1]<<'\n';

```

```
75:     {
76:         for(int j = 0; j < 3; j++)
77:         {
78:
79:             cout<<' \t'<<"*(puntero+"<<j<<"+"<<i<<
80:                 " <<*(puntero+j+(i*3)):"*3));
81:         }
82:
83:         puntero = new int(123);
84:         cout<<' \n'<<"puntero = new int(123)"<<' \n';
85:         cout<<"*puntero: "<<*puntero<<' \n';
86:         cout<<"puntero: "<<puntero<<' \n';
87:         delete puntero;
88:
89:         int *array = new int[10]{0,2,4,6,8,10,12,14,16,
90:             18};
91:         cout<<' \n'<<"int *array = new int[10]{0,2,4,6,8,
92:             10,12,14,16,18}"<<' \n';
93:         for(int i = 0; i < 10; i++)
94:         {
95:             cout<<' \t'<<"*(array+"<<i<<"):"<<*(array+i);
96:             cout<<' \t'<<"array+i"<<i<<":
97:                 "<<array+i<<' \n';
98:
99:             delete [] array;
100:            cout<<"delete [] array;"<<' \n';
101:
102:            int filas = 3;
103:            int columnas = 3;
104:            int **matrix = new int*[filas];
105:            for(int i = 0; i < filas; i++)
106:            {
107:                matrix[i] = new int[columnas]{1+i*3,2+i*3,
108:                    3+i*3};
109:            }
110:            cout<<' \n'<<"int **matrix = new
111:                int*[filas];"<<' \n';
```

```
107:     cout<<'\\t'<<"for(int i = 0; i < filas;  
108:         i++)"<<'\\n';  
109:     cout<<'\\t'<<"matrix[i] = new  
110:         int[columnas]{1*(i+1),2*(i+1),3*(i+1)};"<<'\\n';  
111:     cout<<'\\n'<<"matrix: "<<matrix<<'\\n';  
112:     cout<<"&matrix: "<<&matrix<<'\\n';  
113:     cout<<"matrix[0]: "<<matrix[0]<<'\\n';  
114:     cout<<"&matrix[0]: "<<&matrix[0]<<'\\n';  
115:     cout<<"&matrix[0][1]: "<<&matrix[0][1]<<'\\n';  
116:     cout<<"&matrix[0][2]: "<<&matrix[0][2]<<'\\n';  
117:     cout<<"&matrix[1]: "<<&matrix[1]<<'\\n';  
118:     for (int i = 0; i < filas; i++)  
119:     {  
120:         delete[] matrix[i];  
121:     }  
122:     delete[] matrix;  
123: }
```

PRÁCTICA/LABORATORIO N° 02
LISTAS SIMPLEMENTE ENLAZADAS

Objetivos:

- Utilizar [Dev-C++](#)
- Implementar estructuras (nodo) y clases (lista_simple).
- Resolver problemas utilizando listas simplemente enlazadas.

0. Ejercicios con listas simplemente enlazadas

A continuación, se presentan algunos fragmentos de código c++ de la implementación de listas simplemente enlazadas, para cada fragmento de código debe comentar el código y adicionalmente, para la función main() debe presentar los resultados parciales en consola.

Código	Comentario
Estructura “nodo”:	
<pre>struct nodo { int key; nodo *next; };</pre>	
Clase “lista_simple”	
<pre>class lista_simple { public: nodo *head; lista_simple() { head = NULL; } void insertar_inicio(int k); void insertar_fin(int k); void imprimir(); void eliminar_key(int k); int buscar_key(int k); };</pre>	
Función “insertar inicio” de la clase “lista_simple”	
<pre>void lista_simple::insertar_inicio(int k) { nodo *x = new nodo; x->key = k; x->next = this->head; head = x; this->imprimir(); }</pre>	

Función “insertar_fin” de la clase “lista_simple”	
<pre>void lista_simple::insertar_fin(int k) { if(this->head == NULL) { this->insertar_inicio(k); } else { nodo *x = this->head; while(x->next != NULL) { x = x->next; } nodo *y = new nodo; y->key = k; y->next = x->next; x->next = y; } this->imprimir(); }</pre>	
Función “imprimir” de la clase “lista_simple”	
<pre>void lista_simple::imprimir() { nodo *x = this->head; int count = 0; while(x != NULL) { cout<<x->key<<'\t'; x = x->next; count++; } cout<<endl<<count<<" elementos"<<endl; }</pre>	
Función “eliminar_key” de la clase “lista_simple”	
<pre>void lista_simple::eliminar_key(int k) { nodo *x = this->head; if(x != NULL) { if(x->key == k) { this->head = this->head->next; delete x; } else { while(x->next != NULL and x->next->key != k) { x = x->next; } if(x != NULL) { nodo *y = x->next; x->next = x->next->next; delete y; } } } this->imprimir(); return; }</pre>	
Función “buscar_key” de la clase “lista_simple”	
<pre>int lista_simple::buscar_key(int k) { nodo *x = this->head; while(x != NULL and x->key != k) { x = x->next; } if(x != NULL) { return 1; } else { return 0; } }</pre>	

Función main()		
<pre>int main() { lista_simple lista; lista.insertar_inicio(55); lista.insertar_inicio(44); lista.insertar_inicio(33); lista.insertar_inicio(22); lista.insertar_inicio(11); lista.eliminar_key(11); lista.eliminar_key(33); lista.eliminar_key(55); cout<<lista.buscar_key(22)<<endl; cout<<lista.buscar_key(33)<<endl; lista.insertar_fin(66); }</pre>		

1. Del código presentado anteriormente en el punto 0, implementar las funciones:

- insertar_anter_de_key(int k) //inserta un nodo anterior a key, key está en lista.
- insertar_despues_de_key(int k) // inserta un nodo posterior a key, key está en lista.
- eliminar_inicio() //elimina el primer elemento de la lista.
- eliminar_fin() //elimina el último elemento de la lista.
- eliminar_anter_de_key(int k) // elimina el nodo anterior a key, key está en lista.
- eliminar_despues_de_key(int k) // elimina el nodo posterior a key, key está en lista.

2. Implementar un SISTEMA (con menú) en consola utilizando listas simplemente enlazadas ORDENADAS (class lista_simple_ordenada), en donde cada nodo contenga un string identificador (KEY) y un flotante (BALANCE). Deberá incluir las funciones:

- insertar(string k, float b) //considerar que se debe mantener el orden ascendente (a<Z).
- eliminar(string k) //puede que k no esté en la lista
- buscar(string k) //considerar que la lista está ordenada, donde devolverá el “balance”.
- imprimir() //KEY y BALANCE.
- sumar_balance() //sumar todos los balance.
- sumar_balance_par() //sumar todos los balance cuyo key sea par.
- invertir() //invertir el orden de los NODOS (luego de esto, solo funcionará imprimir()).

3. Reubicando elementos de una lista

Descripción

Tienes una secuencia de N enteros con valores iniciales $0,1,2,\dots,N-1$ y deseas modificarla. Existen dos tipos de operaciones que realizarás sobre ella: la operación $v\ A\ x$ quitará el entero v de su posición actual y lo colocará justo atrás de x , mientras que la operación $v\ D\ x$ quitará el entero v de su posición actual y lo colocará justo después de x . Escribe un programa que procese M modificaciones en sucesión e imprima la secuencia final resultante.

Entrada

Dos enteros N,M seguidos de M triplets $v\ C\ x$ donde C puede ser A o D . Puedes suponer que $2 \leq N,M \leq 10^5$ y que $0 \leq v \neq x < N$.

Salida

Los N enteros de la secuencia final resultante.

Ejemplo

Entrada	Salida
5 2 3 A 1 1 D 4	0 3 2 4 1

4. De una lista a otra

Descripción

Tienes **N** listas de enteros e inicialmente la primera lista guarda un **1**, la segunda lista guarda un **2**, etc. Realizarás **M** operaciones sobre las listas: cada operación consiste en transladar a la lista **I** todos los enteros almacenados en la lista **J**, de modo que los elementos trasladados se agregan al final de la lista **I** en el orden en el que aparecían en la lista **J** y esta última queda vacía. Escribe un programa que muestre el contenido final de todas las listas.

Entrada

Dos enteros **N,M** seguidos de **M** parejas de enteros **I,J** separados por un espacio. Puedes suponer que **1≤N,M≤100,000** y que **1≤I≠J≤N**.

Salida

Para cada lista (de la primera a la última), una línea con los enteros almacenados después de realizar todas las operaciones.

Ejemplo

Entrada	Salida
8	2
3	5 4
1 3	6
5 4	7 1 3
7 1	8

5. Nodos alcanzables

Descripción

Dados **N** nodos numerados del **0** al **N-1** donde cada uno tiene un enlace a su siguiente nodo, escribe un programa que determine si existe una forma de llegar del nodo **0** al nodo **N-1**.

Entrada

Dos enteros **N,M** seguidos de **M** parejas de enteros **X,Y** que denotan un enlace del nodo **X** al **Y**. Pueden existir nodos que no tengan un nodo siguiente. Puedes suponer que **1≤M≤N≤100**.

Salida

El valor **1** si existe una forma de llegar del nodo **0** al **N-1** y el valor **0** en otro caso.

Pista

Los nodos pueden tener un enlace ordenado ($0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow N-1$) y otro según las entradas.

Ejemplo

Entrada	Salida
3 2 0 1 1 2	1
3 3 1 0 2 1 0 1	0

```
1: #include<iostream>
2:
3: using namespace std;
4:
5: struct nodo
6: {
7:     int key;
8:     nodo *next;
9: };
10:
11: class lista_simple
12: {
13:     public:
14:         nodo *head;
15:
16:     public:
17:         lista_simple()
18:     {
19:             this->head = NULL;
20:         }
21:         void imprimir();
22:         void insertar_inicio(int k);
23:         void insertar_fin(int k);
24:         void eliminar(int k);
25:         int buscar_key(int k);
26:     };
27:
28: int main()
29: {
30:     lista_simple lista;
31:     lista.insertar_inicio(44);
32:     lista.insertar_inicio(33);
33:     lista.insertar_inicio(22);
34:     lista.insertar_inicio(11);
35:     lista.insertar_fin(55);
36:
37:
38:     //11 22 33 44 55
39:     lista.eliminar(11);
```

```
40:     cout<<endl<<lista.buscar_key(11);
41:     cout<<endl<<lista.buscar_key(33);
42:     lista.eliminar(33);
43:     lista.eliminar(55);
44:     lista.eliminar(22);
45:     lista.eliminar(44);
46:     lista.eliminar(66);
47: }
48:
49: void lista_simple::imprimir()
50: {
51:     nodo *x = this->head;
52:     int count=0;
53:     while(x != NULL)
54:     {
55:         cout<<x->key<<'\t';
56:         x = x->next;
57:         count++;
58:     }
59:     cout<<endl<<count<<" elementos"<<endl;
60: }
61:
62: void lista_simple::insertar_inicio(int k)
63: {
64:     nodo *x = new nodo;
65:     x->key = k;
66:     x->next = this->head;
67:     this->head = x;
68:     this->imprimir();
69: }
70:
71: void lista_simple::insertar_fin(int k)
72: {
73:     if(this->head == NULL)
74:         this->insertar_inicio(k);
75:     else
76:     {
77:         nodo *x = this->head;
78:         while(x->next != NULL)
```

```

79:             x = x->next;
80:             nodo *y = new nodo;
81:             y->key = k;
82:             y->next = x->next;
83:             x->next = y;
84:         }
85:         this->imprimir();
86:     }
87:
88: void lista_simple::eliminar(int k)
89: {
90:     if(this->head != NULL)
91:     {
92:         nodo *x = this->head;
93:         if(x->key == k)
94:         {
95:             this->head = x->next;
96:             delete x;
97:         }
98:         else
99:         {
100:             while(x->next != NULL && x->next->key != k)
101:                 x=x->next;
102:                 if(x->next != NULL)
103:                 {
104:                     nodo *y = x->next;
105:                     x->next = x->next->next;
106:                     delete y;
107:                 }
108:             }
109:         }
110:         this->imprimir();
111:     }
112:
113: int lista_simple::buscar_key(int k)
114: {
115:     nodo *x = this->head;
116:     while(x != NULL && x->key != k)

```

```
117:         x = x->next;
118:     if(x != NULL)
119:         return 1;
120:     else
121:         return 0;
122: }
```

PRÁCTICA/LABORATORIO N° 03

LISTAS CIRCULARES Y LISTAS DOBLEMENTE ENLAZADAS

Objetivos:

- Utilizar [Dev-C++](#).
- Implementar listas circulares doblemente enlazadas y listas doblemente enlazadas.
- Resolver problemas utilizando listas circulares doblemente enlazadas.

0. Ejercicios con listas doblemente enlazadas:

A continuación, se presenta el archivo *main.cpp*, donde se instancia el objeto *lista* de la clase *lista_doble* y se ejecutan algunas operaciones:

```
main.cpp | lista_doble.h
1 #include<iostream>
2 #include "lista_doble.h"
3
4 using namespace std;
5
6 int main()
7 {
8     lista_doble lista;
9     lista.insertar_fin(11);
10    lista.insertar_fin(22);
11    lista.insertar_fin(33);
12    lista.insertar_fin(44);
13    lista.insertar_fin(55);
14 }
```

Sin embargo, ¿Dónde está la clase *lista_doble*?

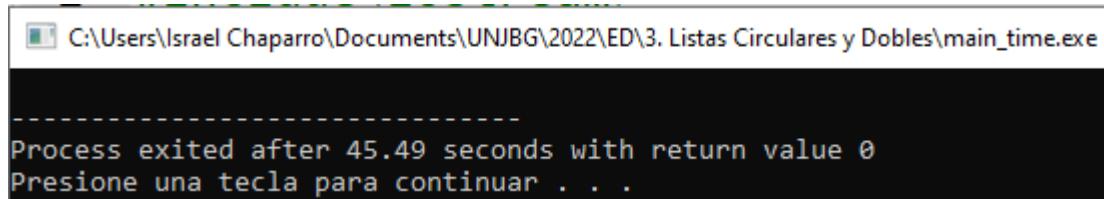
Podemos ver en la línea 2 (cabecera) la inclusión de un archivo “*lista_doble.h*”, es en este archivo donde se ha implementado la estructura *nodo_ld* y la clase *lista_doble* junto con las funciones *insertar_fin* e *imprimir*:

```
lista_doble.h
1 #include<iostream>
2
3 using namespace std;
4
5 struct nodo_ld
6 {
7     int key;
8     nodo_ld *next;
9     nodo_ld *prev;
10};
11
12 class lista_doble
13 {
14 public:
15     nodo_ld *head;
16     nodo_ld *tail;
17
18     lista_doble()
19     {
20         head = NULL;
21         tail = NULL;
22     }
23
24     void insertar_fin(int k);
25     void imprimir();
26 };
27
28 void lista_doble::insertar_fin(int k)
29 {
30     nodo_ld *x = new nodo_ld;
31     x->key = k;
32     x->next = NULL;
33     x->prev = this->tail;
34     this->tail = x;
35     if(this->tail->prev != NULL)
36         this->tail->prev->next = x;
37     else
38         this->head = x;
39     this->imprimir();
40 }
41
42 void lista_doble::imprimir()
43 {
44     nodo_ld *x = this->head;
45     int count = 0;
46     while(x != NULL)
47     {
48         cout<<x->key<<'\t';
49         x = x->next;
50         count++;
51     }
52     cout<<endl<<count<<" elementos"<<endl;
53     cout<<endl;
54 }
```

1. Implementar, en el archivo de cabecera “lista_doble.h” utilizando una estructura `nodo_Id {int key, nodo_Id *next, nodo_Id *prev}`, la clase `lista_doble` que contiene las siguientes funciones:
 - insertar_inicio(int k) //inserta un nodo al inicio de la lista.
 - imprimir_al_reves //imprime la lista del fin al inicio
 - insertar_anter_de_key(int k, int kk) //inserta un nodo anterior a k, k está en lista.
 - eliminar_anter_de_key(int k) // elimina el nodo anterior a key, key está en lista.
 - eliminar_inicio_fin() //elimina el primer elemento y el último elemento de la lista.
2. Del ejercicio 1 de la práctica 02 y 03, generar un archivo de cabecera “`lista_simple_time.h`” (puede ser útil puedes renombrar la estructura `nodo` de ese archivo a `nodo_ls`) y “`lista_doble_time.h`”, eliminando la función imprimir. Luego implemente el siguiente código:

```
main_time.cpp
1 #include<iostream>
2 #include "lista_simple_time.h"
3 #include "lista_doble_time.h"
4
5 using namespace std;
6
7 int main()
8 {
9     lista_simple lista;
10    //Lista_doble lista;
11    int k=10000;
12    for(unsigned int i=0; i<k; i++)
13    {
14        lista.insertar_fin(i);
15    }
16 }
```

Anote el tiempo de ejecución otorgado por Dev-C++.



C:\Users\Israel Chaparro\Documents\UNJBG\2022\ED\3. Listas Circulares y Dobles\main_time.exe

Process exited after 45.49 seconds with return value 0
Presione una tecla para continuar . . .

Genere una tabla comparativa de tiempos entre ejecutar, en listas simple y listas doblemente enlazadas, las funciones `insertar_inicio`, `insertar_fin`.

Función	Tiempo en Lista Simple	Tiempo en Lista Doble
Insertar_inicio		
...		

3. Ruleta de la Fortuna (resolver con Listas Circulares Doble Enlazadas)



Descripción

Tienes una ruleta con N casillas (la imagen superior corresponde a una ruleta de 12 casillas) en donde se encuentran los montos a premiarse $P_0, P_1, P_2, \dots, P_{N-1}$ ubicados en forma horaria $P_0 \rightarrow P_1 \rightarrow \dots \rightarrow P_{N-1} \rightarrow P_0 \rightarrow P_1 \rightarrow \dots$. Antes de girar la ruleta, la flecha que indica el premio a ganar se ubica justo entre P_0 y P_{N-1} pero sin estar en ninguna de ambas casillas (como se puede ver en la imagen). Al girar la ruleta con una fuerza F , la flecha avanza o retrocede F casillas, dependiendo si el valor de F es positivo (movimiento horario de la ruleta) o negativo (movimiento anti horario de la ruleta), pudiendo dar una o varias vueltas a la ruleta, hasta detenerse. Se debe mostrar cuál es premio de la casilla en la cual se detiene la flecha de la ruleta.

Entrada

Un entero N , seguido de N enteros $P_0, P_1, P_2, \dots, P_{N-1}$, finalmente un entero F . Puedes suponer que $0 < P_i, N < 1,000,000$, que $-1,000,000 < F < 1,000,000$, adicionalmente $F \neq 0$.

Salida

El premio ganador.

Ejemplo

Entrada	Salida
5 10 100 20 50 200 6	10
5 10 100 20 50 200 -6	200

```
1: #include<iostream>
2:
3: using namespace std;
4:
5: struct nodo_ld
6: {
7:     int key;
8:     nodo_ld *next;
9:     nodo_ld *prev;
10: };
11:
12: class lista_doble
13: {
14:     public:
15:         nodo_ld *head;
16:         nodo_ld *tail;
17:
18:         lista_doble()
19:     {
20:             head = NULL;
21:             tail = NULL;
22:     }
23:
24:         void insertar_fin(int k);
25:         void imprimir();
26: };
27:
28: void lista_doble::insertar_fin(int k)
29: {
30:     nodo_ld *x = new nodo_ld;
31:     x->key = k;
32:     x->next = NULL;
33:     x->prev = this->tail;
34:     this->tail = x;
35:     if(this->tail->prev != NULL)
36:         this->tail->prev->next = x;
37:     else
38:         this->head = x;
39:     this->imprimir();
```

```
40: }
41:
42: void lista_doble::imprimir()
43: {
44:     nodo_ld *x = this->head;
45:     int count = 0;
46:     while(x != NULL)
47:     {
48:         cout<<x->key<<'\t';
49:         x = x->next;
50:         count++;
51:     }
52:     cout<<endl<<count<<" elementos"<<endl;
53:     cout<<endl;
54: }
55:
56: struct nodo
57: {
58:     int key;
59:     nodo *next;
60: };
61:
62: class lista_circular
63: {
64:     public:
65:         nodo *head;
66:
67:         lista_circular()
68:     {
69:             head = NULL;
70:     }
71:
72:         void insertar_inicio(int k);
73:         void insertar_fin(int k);
74:         void eliminar_key(int k);
75:         void imprimir();
76:         int buscar_key(int k);
77: };
78:
```

```
79: void lista_circular::insertar_inicio(int k)
80: {
81:     nodo *x = new nodo;
82:     x->key = k;
83:     x->next = this->head;
84:     this->head = x;
85:     if(x->next == NULL)
86:
87:         x->next = x;
88:     else
89:     {
90:         x = x->next;
91:         while(x->next != this->head->next)
92:             x = x->next;
93:         x->next = this->head;
94:     }
95: }
96:
97: void lista_circular::insertar_fin(int k)
98: {
99:     nodo *x = new nodo;
100:    x->key = k;
101:    if(this->head == NULL)
102:    {
103:        this->head = x;
104:        x->next = x;
105:    }
106:    else
107:    {
108:        nodo *y = this->head;
109:        while(y->next != this->head)
110:            y = y->next;
111:        y->next = x;
112:        x->next = this->head;
113:    }
114: }
115:
116: void lista_circular::imprimir()
117: {
```

```
118:     if(this->head != NULL)
119:     {
120:         nodo *x = this->head;
121:         cout<<x->key<<'\t';
122:         while(x->next != this->head)
123:         {
124:             x = x->next;
125:             cout<<x->key<<'\t';
126:         }
127:     }
128:     cout<<endl;
129: }
130:
131: int lista_circular::buscar_key(int k)
132: {
133:     if(this->head ==NULL)
134:     {
135:         return 0;
136:     }
137:     else
138:     {
139:         nodo *x = this->head;
140:         while(x->next != this->head && x->key != k)
141:             x = x->next;
142:         if(x->key == k)
143:         {
144:             return 1;
145:         }
146:         else
147:         {
148:             return 0;
149:         }
150:     }
151: }
152:
153: void lista_circular::eliminar_key(int k)
154: {
155:     if(this->head != NULL)
156:     {
```

```
157:         nodo *x = this->head;
158:         if(this->head->key == k)
159:         {
160:             if(this->head->next == this->head)
161:             {
162:                 this->head = NULL;
163:                 delete x;
164:             }
165:             else
166:             {
167:                 this->head = this->head->next;
168:                 nodo *y = this->head;
169:                 while(y->next != x)
170:                     y = y->next;
171:                 y->next = this->head;
172:                 delete x;
173:             }
174:         }
175:         else
176:         {
177:             while(x->next != this->head && x->next-
>key != k)
178:                 x = x->next;
179:             if(x->next != this->head)
180:             {
181:                 nodo *y = x->next;
182:                 x->next = x->next->next;
183:                 delete y;
184:             }
185:         }
186:     }
187: }
```

PRÁCTICA/LABORATORIO N° 04
PILAS Y APLICACIONES CON PILAS

Objetivos:

- Utilizar [Dev-C++](#).
- Implementar pilas en arreglos y listas simplemente enlazadas. (1 y 2) – 5 pts.
- Resolver problemas utilizando pilas. (3,4,5,6,7) – 5 pts.
- Resolver problemas de tratamiento de expresiones aritméticas usando pilas (8,9). – 5 pts.
- Resolver el problema de las torres de Hanoi usando pilas (10). – 5 pts.

1. Implementar, en el archivo de cabecera “pila.h” utilizando un arreglo, la clase “pila_array”, que contiene las siguientes funciones:

- push(int k) //inserta k en pila, si no hay espacio escribirá “overflow”.
- pop() //elimina un dato de la pila (y lo devuelve), si no hay elementos escribirá “underflow”.
- stack-empty() //retorna TRUE si la lista está vacía, sino retorna FALSE.

2. Implementar, en el mismo archivo “pila.h” del ejercicio 1, utilizando listas simplemente enlazadas, la clase “pila_list” que contiene las siguientes funciones:

- push(int k) //inserta k en pila.
- pop() //elimina un dato de la pila (y lo devuelve), si no hay elementos escribirá “underflow”.
- stack-empty() //retorna TRUE si la lista está vacía, sino retorna FALSE.

3. Máximo elemento de la pila (archivo 3.cpp)

Tienes una secuencia vacía y se te dan preguntas. Cada pregunta puede ser de alguno de los siguientes tipos:

- 1 x** -Agrega el elemento x al tope de la pila.
- 2** -Borra el elemento en el tope de la pila.
- 3** -Imprime el máximo elemento en la pila.

Entrada

La primera línea contiene un entero **N**. Las siguientes **N** líneas cada una de ellas tendrá alguna pregunta del estilo mencionado anteriormente (se garantiza que cada una de las preguntas son válidas).

Salida

Para cada pregunta de tipo 3, imprimir el máximo elemento en la pila.

Ejemplo

Entrada	Salida
10	26
1 97	91
2	
1 20	
2	
1 26	
1 20	
2	
3	
1 91	
3	

4. A jugar 2048 <https://play2048.co/> (archivo 4.cpp)

De tanto jugar 2048 comenzaste a tener sueños raros. El sueño se parece mucho al juego: tienes una torre gigante donde comenzarán a caer bloques con números que tienen potencias de dos (2, 4, 8, 16, 32...).

Cada que cae un bloque, ocurre lo siguiente: Mientras los dos bloques que están hasta arriba de la torre sean iguales, ambos se combinan en un sólo bloque con valor igual a la suma de los bloques que se combinaron. Esto ocurre hasta que ya no haya bloques iguales juntos. En ese momento, cae el siguiente bloque.

Escribe un programa que, dada la secuencia de bloques que caerán, diga cómo queda la torre de bloques al final.

Entrada

Un entero **N** que es la cantidad de bloques que caerán, seguido de **N** líneas con los números escritos en cada uno de esos bloques.

Salida

Un entero: el tamaño de la torre al final, seguido de los valores de los bloques; uno por línea y de arriba hacia abajo.

Ejemplo

Entrada	Salida	Descripción
5	2	La torre se va viendo así: (puedes ver al margen izquierdo como el suelo) 2 <- cae un 2 2 2 <- cae un 2 4 <- se combinan los bloques 2 y 2 4 2 <- cae un 2 4 2 2 <- cae un 2 4 4 <- se combinan los bloques 2 y 2 8 <- se combinan los bloques 4 y 4 8 16 <- cae un 16.
2	16	
2	8	
2		
2		
16		
6	3	2 <- cae un 2 2 8 <- cae un 8 2 8 4 <- cae un 4 2 8 4 2 <- cae un 2 2 8 4 2 2 <- cae un 2 2 8 4 4 <- se combinan los bloques 2 y 2 2 8 8 <- se combinan los bloques 4 y 4 2 16 <- se combinan los bloques 8 y 8 2 16 4 <- cae un 4
2	4	
8	16	
4	2	
2		
2		
4		

5. Las bitácoras de la sana distancia (archivo 5.cpp)

Para informar a la población acerca de la distancia social en tiempos de COVID-19, Susana Distancia cuenta con una gran cantidad de telefonistas. El día típico de una telefonista es algo monótono: exactamente cada minuto, un asistente coloca un papelito en el escritorio de la telefonista, el cual tiene anotado un teléfono al que debe llamar. Los papelitos lentamente se van acumulando, ya que el asistente coloca cada papelito encima de los que ya estén ahí. Cuando la telefonista está lista para realizar la siguiente llamada, ella quita el papelito que esté hasta arriba y llama al teléfono anotado, lo cual puede tomarle varios minutos. La telefonista anotó en su bitácora la duración de cada llamada que hizo, pero no anotó a qué teléfono corresponde cada duración. Por su parte, el asistente anotó en su bitácora el orden en el que fue colocando los papelitos en el escritorio.

Lo que Susana Distancia desea saber es ¿en qué orden se llamó a los teléfonos?

Entrada

El primer renglón contiene un entero positivo N . El segundo renglón contiene N enteros D_1, D_2, \dots, D_N separados por espacios, que son las duraciones en minutos de las N llamadas. Los siguientes N renglones contienen los N enteros P_1, P_2, \dots, P_N que son los teléfonos anotados en los papelitos, en el orden en el que fueron colocados en el escritorio.

Salida

Una secuencia de N renglones que contengan los teléfonos en el orden en el que se llamaron.

Entrada	Salida	Descripción
5	300000000	Al minuto 1, colocan el teléfono 300000000 y la telefonista llama a ese número. Esta llamada dura 2 minutos.
2 1 3 1 2	400000000	Al minuto 2, colocan el teléfono 100000000 pero la telefonista sigue ocupada.
300000000	500000000	Al minuto 3, colocan el teléfono 400000000 y la telefonista se desocupa, por lo que ella llama a ese número. Esta llamada dura 1 minuto.
100000000	900000000	Al minuto 4, colocan el teléfono 500000000 y la telefonista se desocupa, por lo que ella llama a ese número. Esta llamada dura 3 minutos.
400000000	100000000	Al minuto 5, colocan el teléfono 900000000 pero la telefonista sigue ocupada.
500000000		A partir de este momento ya no se colocan nuevos teléfonos. El papelito que queda arriba tiene el número 900000000 y el papelito de abajo tiene el número 100000000.
900000000		

6. Los Últimos Serán los Primeros (archivo 6.cpp)

En una carrera en el pueblo de Calientes todos los participantes creen que el que llegue primero será el ganador, pero no, las reglas son diferentes en Calientes, el ganador es el que llega al último. Escribe un programa que dados los nombres de los concursantes como van cruzando la meta, imprima dichos nombres en orden de acuerdo al lugar que consiguieron en la competencia (El que llegó al último obtiene el primer lugar, el penúltimo el segundo lugar, y así sucesivamente).

Entrada

Varios nombres de concursantes como van cruzando la meta. La entrada termina con el string "#".

Salida

Los nombres en orden al lugar que ocuparon en la competencia.

Entrada	Salida
PABLO VANESSA MARIANA DANIELA IVAN #	IVAN DANIELA MARIANA VANESSA PABLO

7. Descarga en el puerto (HINT: implementar en una lista enlazada con nodos que además almacenen el número de folio) (archivo 7.cpp)

A los grandes puertos llegan grandes contenedores. Cuando descargan los contenedores necesitan utilizar grandes grúas. La forma en la que descargan es la siguiente: el primer contenedor es tomado del barco y puesto en un apartado especial del muelle, el siguiente contenedor es tomado del barco y colocado encima del contenedor que ya estaba en el muelle. Es decir, los contenedores que descargan de los barcos son apilados en el muelle. Luego los contenedores son transportados a bodegas.

Entrada

Leerás por filas hasta llegar al fin de archivo. Cada fila puede comenzar con las letras **D**, **P** ó **L**. Si comienza con la letra **D**, significa que se está descargando un contenedor y seguirá su número de folio. Si la letra es **P**, sigue un número de folio y significa que alguien pregunta cuántos contenedores es necesario remover para obtener el contenedor con el folio proporcionado. Si la letra es **T**, significa que el contenedor de arriba es transportado a una bodega.

Salida

Por cada fila que comience con las letras **P**, debes imprimir el número de contenedores necesarios para obtener el contenedor con el folio proporcionado.

Ejemplo

Entrada	Salida
D 001	1
D 007	2
D 200	0
P 007	
P 001	
T	
T	
P 001	

8. Paréntesis Balanceados (archivo 8.cpp)

Dada una secuencia consistente de paréntesis, determina si la expresión está balanceada. Una secuencia de paréntesis está balanceada si cada paréntesis abierto puede ser emparejado únicamente con un paréntesis de cierre. También el intervalo de elementos dentro de él debe estar balanceado. Tu tendrás tres tipos de paréntesis: (, { y [.

- {[0]} - Este SI es un conjunto balanceado.
- {[1]} - Este NO es un conjunto balanceado.

Entrada

La primera línea de la entrada contiene un número **T**, que será el número de casos. Las siguientes **T** líneas contienen una cadena **S** de paréntesis.

Salida

Para cada caso, imprime en una nueva línea "SI" si el conjunto de paréntesis está balanceado, en caso contrario imprime "NO".

Ejemplo

Entrada	Salida
3	SI
{[()]}{[()]}{[[()]]}}	NO
{[()]}{[()]}{[[()]]}}	SI

9. Implementar las siguientes funciones de tratamiento de expresiones aritméticas (archivo 9.1.cpp y 9.2.cpp):

- infija_to_posfija() //Lee una expresión en notación infija, y la traduce a posfija.
//La expresión solo contiene letras como operandos, operadores y ().

- eval_posfija() //Evalúe, usando una pila, una expresión aritmética en posfija.
//La expresión solo contiene [0-9] como operandos y operadores.

https://scantree.com/Data_Structure/prefix-postfix-infix-online-converter

9.1.cpp

Entrada	Salida
(a*(b+(c/d)))	a b c d / + *

9.2.cpp

Entrada	Salida
122*+2^	25

10. Torre de Hanoi (archivo 10.cpp) <https://www.youtube.com/watch?v=k2nLziNzCDE>

Se te ha dado el trabajo de resolver el problema de la Torre de Hanoi con **N** discos de diferentes tamaños.

Las torres se llaman **A**, **B** y **C**. Inicialmente, todos los discos están ubicados en la torre **S** ordenados de mayor a menor desde abajo hasta arriba (es decir, el más pequeño está en la cima de la torre) y quieras llevarlos a otra torre **D**. Solo puedes mover un disco a la vez y no puedes colocar un disco encima de otro que tenga menor tamaño.

Debes describir la secuencia de movimientos más corta posible para completar tu trabajo.

Entrada

La primera línea de entrada contiene un entero **T**, la cantidad de casos de prueba.

Las siguientes **T** líneas contienen un entero **N** y dos caracteres **S** y **D**, la cantidad de discos del **i**-ésimo caso de prueba, la torre en la que están ubicados los discos y la torre a la que quieras llevarlos, respectivamente.

Salida

Para cada caso de prueba, imprime la secuencia de movimientos más corta posible para completar tu trabajo. Cada movimiento debe ser descrito de la forma:

Mueve el disco de **X** a **Y**

Donde **X** y **Y** son los nombres de las torres que intervienen en el movimiento.

Luego de toda la secuencia, imprime **Listo!**

Ejemplo:

Entrada	Salida
2 2 A B 3 C B	Mueve el disco de A a C. Mueve el disco de A a B. Mueve el disco de C a B. Listo! Mueve el disco de C a B. Mueve el disco de C a A. Mueve el disco de B a A. Mueve el disco de C a B. Mueve el disco de A a C. Mueve el disco de A a B. Mueve el disco de C a B. Listo!

```
1: #include<iostream>
2:
3: using namespace std;
4:
5: int precedencia(char x)
6: {
7:     if(x == '^') return 3;
8:     if(x == '*' || x == '/') return 2;
9:     if(x == '+' || x == '-') return 1;
10:    return 0;
11: }
12:
13: struct nodo_s
14: {
15:     char key;
16:     nodo_s *prev;
17: };
18:
19: class stack_list
20: {
21:     public:
22:         nodo_s *top;
23:     public:
24:         stack_list()
25:         {
26:             top = NULL;
27:         }
28:         void push(char x);
29:         char pop();
30:         bool empty();
31:         int precedence_top();
32:         void print();
33: };
34:
35: void stack_list::print()
36: {
37:     nodo_s *x = top;
38:     cout<<"stack"<<endl;
39:     while(x != NULL)
```

```
40:    {
41:        cout<<x->key<<endl;
42:        x = x->prev;
43:    }
44:    cout<<"-----"<<endl;
45: }
46:
47: void stack_list::push(char x)
48: {
49:     nodo_s *p = new nodo_s;
50:     p->key = x;
51:     p->prev = top;
52:     top = p;
53: }
54:
55: char stack_list::pop()
56: {
57:     if(top == NULL)
58:         raise(SIGSEGV);
59:     else
60:     {
61:         char value = top->key;
62:         nodo_s *x = top;
63:         top = top->prev;
64:         delete x;
65:         return value;
66:     }
67: }
68:
69: bool stack_list::empty()
70: {
71:     if(top == NULL)
72:         return 1;
73:     else
74:         return 0;
75: }
76:
77: int stack_list::precedence_top()
78: {
```

```

79:     return precedencia(top->key);
80: }
81:
82: int main()
83: {
84:     stack_list pila;
85:     char infija[100], posfija[100], token, aux;
86:     cin>>infija;
87:     int i=0;
88:     int j=0;
89:     while(infija[i] != '\0')
90:     {
91:         token = infija[i];
92:         if(token >= 'a' && token <= 'z') //es un
operando
93:         {
94:             posfija[j++] = token;
95:         }
96:         else if(token == '(')
97:             pila.push(token);
98:         else if(token == ')')
99:         {
100:             aux = pila.pop();
101:             while(aux != '(')
102:             {
103:                 posfija[j++] = aux;
104:                 aux = pila.pop();
105:             }
106:         }
107:         else //es un operador
108:         {
109:             while(!pila.empty() &&
precedencia(token) <= pila.precedence_top())
110:             {
111:                 if(token == '^' &&
pila.precedence_top() == 3)
112:                     break;
113:                 posfija[j++] = pila.pop();
114:             }

```

```
115:         pila.push(token);
116:     }
117:     //pila.print();
118:     i++;
119: }
120: while(!pila.empty())
121: {
122:     posfija[j++] = pila.pop();
123: }
124: posfija[j] = '\0';
125:
126: cout<<posfija;
127: }
```

PRÁCTICA/LABORATORIO N° 05
COLAS Y APLICACIONES CON COLAS

Objetivos:

- Utilizar [Dev-C++](#).
- Implementar colas (simples, circulares, dobles) en arreglos (1,2,3; 3.33 puntos c/u).
- Resolver problemas utilizando objetos **cola_circular_array** y **cola_doble_circular_array** (**cuando sea necesario simular una pila** (4,5,6,7,8,9,10,11,12,13; 1 punto c/u).

1. **Implementar, en el archivo de cabecera “cola.h” utilizando un arreglo, la clase “cola_array”, que contiene las siguientes funciones:**
 - enqueue(int k) //inserta k en cola, si no hay espacio escribirá “overflow”.
 - dequeue() // devuelve el elemento de la cola (y lo elimina), considerar “underflow”.
 - queue_empty() //retorna TRUE si la cola está vacía, sino retorna FALSE.
 - queue_full() //retorna TRUE si la cola está llena, sino retorna FALSE.
2. **Implementar, en el archivo de cabecera “cola.h” del ejercicio 1, utilizando un arreglo, la clase “cola_circular_array”, que contiene las siguientes funciones:**
 - enqueue(int k) //inserta k en cola, asume que hay espacio en el array.
 - dequeue() // devuelve el elemento de la cola (y lo elimina), asume que la cola no está vacía.
 - queue_empty() //retorna TRUE si la cola está vacía, sino retorna FALSE.
 - queue_full () //retorna TRUE si la cola está llena, sino retorna FALSE.
3. **Implementar, en el mismo archivo “cola.h” del ejercicio 1, 2 y 3, utilizando un arreglo, la clase “cola_doble_circular_array” que contiene las siguientes funciones:**
 - enqueue_head(int k) //inserta k en la cabeza de la cola, asume que hay espacio en el array.
 - enqueue_tail(int k) //inserta k en la “cola” de la cola, asume que hay espacio en el array.
 - dequeue_head() //devuelve el elemento de la cabeza de la cola (y lo elimina), asume que la cola no está vacía.
 - dequeue_tail () //devuelve el elemento de la “cola” de la cola (y lo elimina), asume que la cola no está vacía.
 - queue_empty() //retorna TRUE si la cola está vacía, sino retorna FALSE.
 - queue_full () //retorna TRUE si la cola está llena, sino retorna FALSE.

4. Fusión de colas (archivo 4.cpp)

Escribir una función toma dos colas con la misma longitud y devuelve una cola, fusionando los elementos de los dos originales uno por uno.

Entrada

La primera línea contiene un entero **N**. Indicando la longitud de las colas, luego se esperan **N** elementos de la primera cola y **N** elementos de la segunda cola.

Salida

Imprimir los elementos de la cola fusionada separados por un espacio.

Ejemplo

Entrada	Salida
3 1 2 3 -1 -2 -3	1 -1 2 -2 3 -3
3 1 2 3 3 2 1	1 3 2 2 3 1

5. Dos estructuras (archivo 5.cpp)

Después de las clases de Estructuras de Datos te interesó comparar las nuevas estructuras de datos que aprendiste. Para ello, quieres ver qué pasaría si haces metes o sacas datos con una pila o una cola. Escribe un programa que utilice una pila y una cola e imprima el resultado de hacer pop/remove a cada una.

Entrada

Un entero **N**: la cantidad de operaciones a ejecutar. Despues, **N** operaciones. Si la operación comienza con "meter datos", vendrá a continuación otro entero que debes hacer push en la pila y enqueue en la cola. Si la línea contiene un "sacar datos", debes imprimir el resultado de hacer pop a la pila y dequeue a la cola.

Salida

Para cada “sacar datos”, una línea con el resultado de hacer pop a la pila y remove a la cola, separados por un espacio.

Ejemplo

Entrada	Salida
4 meter datos 10 meter datos 11 sacar datos sacar datos	11 10 10 11
6 meter datos 4 meter datos 2 sacar datos meter datos 9 sacar datos sacar datos	2 4 9 2 4 9

6. Cafetería UNJBG (archivo 6.cpp)

En la cafetería de la UAM hay dos colas: la de los alumnos y la de los docentes. Cuando es el turno de atender a la siguiente persona, la política de la cafetería es darle preferencia a la gente formada en la cola de los alumnos ☺; los docentes son atendidos sólo si no hay alumnos formados. Con esta información, escribe un programa que procese una secuencia de eventos de los siguientes tipos.

LLEGA DOCENTE v: Llegó la persona de nombre **v** a formarse a la cola de los docentes.

LLEGA ALUMNO v: Llegó la persona de nombre **v** a formarse a la cola de los alumnos.

ATIENDE: Se debe atender a la siguiente persona, imprimiendo su nombre.

Entrada

Un entero **N** seguido de los **N** eventos. Los nombres consisten de entre 5 y 10 letras minúsculas y que nunca ocurre un evento de atención cuando las dos colas están vacías.

Salida

Los nombres de las personas atendidas en el orden en el que esto ocurrió.

Ejemplo

Entrada	Salida
7 LLEGA DOCENTE luis LLEGA ALUMNO ana LLEGA DOCENTE jorge ATIENDE ATIENDE LLEGA ALUMNO juan ATIENDE	ana luis juan

7. Un banco con clientes no preferentes (archivo 7.cpp)

La sucursal del banco "Dinero++" dispone de dos filas para atender a sus clientes y éstas operan bajo políticas diferentes. La primera fila está destinada para clientes en buen estatus y opera bajo la política de que el primer cliente en llegar es el primero en ser atendido. La segunda fila está destinada para clientes morosos y opera bajo la política de que el último cliente en llegar es el primero en ser atendido. Cuando un cliente llega a la sucursal, el sistema le dice en qué fila debe formarse. Cuenta con una bitácora de eventos que indica en qué momento llegó cada cliente y en qué fila se formó, así como en qué momento se atendió al siguiente cliente y de qué fila provino. Escribe un programa que determine en qué orden fueron atendidos los clientes.

Entrada

Un entero **N** seguido de la secuencia de **N** eventos de la bitácora. Un evento en el que llega un cliente se describe con la letra **E** seguido del número de fila en la que debe formarse y de su nombre. Un evento de atención a cliente se describe con la letra **A** seguido del número de fila que se atiende. Puedes suponer el número de fila es **1** o **2**, que los nombres de todos los clientes tienen una longitud de a lo mucho 10 letras minúsculas y que no existen eventos de atención para filas vacías.

Salida

Para cada evento de atención a cliente, el nombre del cliente atendido.

Ejemplo

Entrada	Salida
8	juan
E 1 juan	pedro
E 2 paco	jorge
E 1 jorge	paco
A 1	
E 2 pedro	
A 2	
A 1	
A 2	

8. Cola bancaria de la suerte (archivo 8.cpp)

Eres cuentahabiente del banco más popular de la región y por lo general la cola del banco es extremadamente larga. Además, cada cuentahabiente tiene una prioridad entera **P** y la política es que los clientes con prioridad más alta sean atendidos primero (aunque hayan llegado a la cola después), lo que puede ocasionar que algunos cuentahabientes con prioridad baja no sean atendidos.

Para solucionar esto y volver menos monótona la espera en la cola, el banco ha decidido realizar sorteos para que algunos clientes avancen más rápido. Durante el sorteo se extrae una letra y un entero **I** de la tómbola: a los clientes que estén formados en ese momento y cuyos nombres inicien con esa letra se les sumarán **I** puntos a su prioridad. En caso de que dos cuentahabientes tengan la misma prioridad, se atenderá al que haya llegado primero. Los eventos que suceden en la cola se representan de la siguiente forma:

C N P: llega un cuentahabiente de nombre **N** con prioridad inicial **P**.

S L I: se realizó un sorteo y se extrajo la letra **L** y el entero **I**.

A: se atendió al cuentahabiente de mayor prioridad de la cola.

Escribe un programa que determine el nombre del cliente que fue atendido en cada evento de atención.

Entrada

Un entero **E** seguido de **E** eventos. Puedes suponer que no se atenderá a nadie mientras la cola esté vacía y que tanto los nombres como las letras del sorteo consisten de letras mayúsculas.

Salida

Los nombres de los cuentahabientes atendidos en el orden en el que se atendieron.

Ejemplo

Entrada	Salida
6 C CARLOS 8 C JORGE 5 S J 10 C JUAN 7 A A	JORGE CARLOS

9. Safari (archivo 9.cpp)

Tu amigo Ángel ha decidido empezar a trabajar en un Safari, él es encargado de ver los nuevos animales que llegan, y como este Safari se preocupa por los animales los deja libres después de cierto tiempo, por ello también administra los que se van y el alimento de los animales. Bien Ángel es algo flojo, por lo que te ha pedido que hagas un programa que te ayude con lo siguiente.

Entrada

Recibirás una letra, la cual puede ser N, S, C, A, F. Donde:

- N=Recibirás un nuevo animal por lo que seguido de N Recibirás el nombre del animal.
- S=El animal que tiene más tiempo que llegó se irá, por lo que deberás imprimir que animal es.
- C=Que has recibido alimento, por lo que seguido de C recibirás una variable X que es igual a la cantidad de alimento que recibirás.
- A=Que debes alimentar a todos los animales con la cantidad de alimento que tienes acumulado de forma que todos los animales coman lo mismo, por lo que debes imprimir cuánto le corresponde a cada animal. Presta atención aquí, si en algún momento, no tienes alimento. Deberás liberar a todos los animales empezando por el último animal que entró y terminar tu programa.
- F= Deberás imprimir la cantidad de animales que quedaron en el Safari.
- X=Deberás terminar el programa

Ejemplo

Entrada	Salida
N Leon	2
N Leon	1
F	Leon
C 4	3
N Leon	
N Tigre	
A	
S	
F	
N Leon	3
N Cebra	Tigre
N Tigre	Leon
F	Tigre
N Leon	Cebra
N Tigre	Leon
A	

10. Teclado roto (archivo 10.cpp)

Te encuentras escribiendo un texto largo con un teclado roto. Bueno, no está tan roto. El único problema con el teclado es que a veces la tecla "Inicio" o la tecla "Fin" se presionan automáticamente (de manera interna). No consciente de este problema, porque estás muy concentrado en el texto y no volteas a ver el monitor, terminas de escribir. Después de escribir miras el monitor y te das cuenta de lo sucedido y tienes que comenzar de nuevo, pero ahora tomando las debidas precauciones. Dado el historial de teclas oprimidas, tu tarea consiste en escribir el texto que se despliega en el monitor al momento en el que volteas a verlo.

Entrada

Una sola línea que contiene al menos un carácter. Los caracteres serán únicamente letras minúsculas del alfabeto inglés, espacios y dos caracteres especiales: '[' y ']'.

'[' Significa que la tecla "Inicio" fue presionada internamente, y ']' significa que "Fin" se presionó internamente.

Salida

Lo que tienes que escribir es el texto que se despliega en el monitor al momento en el que volteas a verlo.

Ejemplo:

Entrada	Salida
[][][][][][][]][este es un ejemplo	este es un ejemplo
[]debés[hablar[comoyoda	comoyodahablardebés

11. Fila Bancaria (archivo 11.cpp)

A nadie le agrada esperar en los bancos cuando hay una cola muy grande. Vas a escribir un programa que simule el comportamiento de dos tipos de personas, los que esperan y los que no tanto. Para ello leerás una lista de eventos.

Entrada

Leerás un número **N**. Posteriormente, leerás **N** líneas, en cada fila pueden estar tres letras: **E**, **N**, **F**. Si la letra es **E**, significa que ha llegado al banco una persona que sí va a esperar. Si la letra es **N**, significa que ha llegado una persona muy impaciente y no va a formarse si en la cola hay más de 4 personas. Si la letra que lees es **F**, significa que una persona terminó de ser atendida. Las personas son atendidas en el mismo orden en el que llegan.

Salida

Por cada fila que comienza con la letra **N**, debes decir si la persona poco paciente decidió esperar o no espera.

Ejemplo

Entrada	Salida
9	
E	espera
E	no espera
E	espera
N	
E	
N	
F	
N	
F	

12. Salchipapas (archivo 12.cpp)

Eres el dueño de un puesto de salchipapas. Sin embargo, en estas últimas semanas tus salchipapas se han vuelto realmente famosos dentro de la ciudad gracias a unas cremas/salsas únicas e inigualables. Es por ello que todos los días se forman largas filas de espera. Debido a esto, te gustaría llevar un control del negocio. Para hacer la tarea más fácil, has decidido crear un programa que te ayude en esta tarea.

Entrada

En la primera línea recibirás un entero **N**, el número de operaciones a realizar. En las siguientes **N** líneas recibirás un entero representando la operación a realizar:

- Si el entero es un **1**, significa que un nuevo cliente se forma en la fila, por lo que deberás leer un entero **t** representando el número de salchipapas que desea ordenar.
- Si el entero es un **2**, significa que el cliente que se encuentra al frente de la fila será atendido.
- Si el entero es un **3**, significa que quieres saber el número de clientes que se encuentran formados actualmente en la fila.
- Si el entero es un **4**, significa que quieres saber el número de salchipapas que has vendido hasta ahora.

Salida

Por cada una de las entradas del tipo **3** y **4**, deberás imprimir un entero por línea representando la respuesta a la pregunta correspondiente.

Ejemplo

Entrada	Salida	
9 1 4 1 3 1 10 3 1 5 2 2 1 9 4	3 7	Al principio se forman 3 personas pidiendo 4, 3 y 10 salchipapas respectivamente. Después se pregunta por el número de personas formadas, así que la respuesta es 3. Luego se forma un cliente más pidiendo 5 salchipapas. Posteriormente son atendidos los primeros 2 clientes, por lo que se venden $4+3=7$ salchipapas. Luego se forma un cliente más a la fila pidiendo 9 salchipapas. Finalmente, se pregunta por el número de salchipapas vendidos, así que la respuesta es 7.
6 4 1 3 2 1 7 4 3	0 3 1	

13. Round Robin (archivo 13.cpp)

Cierta computadora tiene un procesador que puede atender un proceso a la vez por **K** unidades de tiempo.

Si el proceso necesita más de **K** unidades para terminar, el procesador lo atiende exactamente **K** unidades y deja pendientes las unidades excedentes. Cuando un proceso es atendido y no finaliza, se forma al final de la lista de procesos pendientes.

Escribe un programa que lea el número de procesos, **K**, los IDs y el tiempo de ejecución de cada uno de los **N** procesos y determine el orden en que los procesos finalizarán.

Entrada

En la primera línea dos enteros, **N** y **K**. En cada una de las siguientes **N** líneas, dos enteros **X**, **Y**, separados por un espacio. Que denotan el ID del proceso y el tiempo de ejecución que necesita para concluir.

Salida

N líneas con los ID de los procesos en el orden que concluyeron. Considera que los procesos de la entrada ya se encuentran en la lista de espera.

Ejemplo

Entrada	Salida
5 6	39
100 30	109
102 50	100
109 13	102
45 88	45
39 7	

```
1: #include<iostream>
2: #define MAX_Q 8
3:
4: using namespace std;
5:
6: class cola_array
7: {
8:     public:
9:         int head,tail;
10:    cola_array()
11:    {
12:        head = -1;
13:        tail = -1;
14:    }
15:    int Q[MAX_Q];
16:    void enqueue(int k);
17:    int *dequeue();
18:    void imprimir();
19: };
20:
21: void cola_array::enqueue(int k)
22: {
23:     if(tail +1 == MAX_Q)
24:     {
25:         cout<<"overflow";
26:     }
27:     else
28:     {
29:         Q[++this->tail] = k;
30:         if(this->head== -1)
31:         {
32:             this->head = this->tail;
33:         }
34:     }
35: }
36:
37: int *cola_array::dequeue()
38: {
39:     if(head == -1)
```

```
40:    {
41:        cout<<"UNDERFLOW";
42:        return NULL;
43:    }
44: else
45: {
46:     int *x = &Q[head];
47:     if(head == tail)
48:     {
49:         head = -1;
50:         tail = -1;
51:     }
52: else
53: {
54:     head++;
55: }
56: return x;
57: }
58: }
59:
60: void cola_array::imprimir()
61: {
62:     if(this->head != -1)
63:     {
64:         for(int i=this->head; i<=this->tail; i++)
65:         {
66:             cout<<this->Q[i]<<"\t";
67:         }
68:     }
69: }
70:
71: int main()
72: {
73:     cola_array cola;
74:     int *x;
75:     cola.enqueue(10);
76:     cola.enqueue(11);
77:     cola.imprimir();
78:     x = cola.dequeue();
```

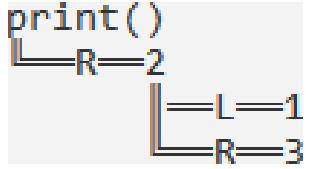
```
79:     x = cola.dequeue();
80:     cout<<"el retorno es "<<*x<<endl;
81:     cola.imprimir();
82: }
```

PRÁCTICA/LABORATORIO N° 06
ÁRBOLES DE BÚSQUEDA BINARIA – BINARY SEARCH TREE

Objetivo: Implementar Árboles de Búsqueda Binaria (Binary Search Tree).

```
1  #include<iostream>
2  #include<string>
3
4  using namespace std;
5
6  struct tree_node{
7      int key;
8      tree_node *parent;
9      tree_node *left;
10     tree_node *right;
11     tree_node(int k){
12         key = k;
13         left = NULL;
14         right = NULL;
15     }
16 };
17
18 class bst{
19 public:
20     tree_node *root;
21     bst(){
22         root = NULL;
23     }
24     void insert(int k);
25     void print();
26     void print(string prefix, tree_node *x, bool isLeft);
27 };
28
29 void bst::print(){
30     cout<<endl;
31     print("", root, false);
32     cout<<endl;
33 }
34
35 void bst::print(string prefix, tree_node *x, bool isLeft){
36     if(x != NULL){
37         cout<<prefix;
38         cout<<(isLeft ? "\xBA\xCD\xCDL\xCD" : "\xC8\xCD\xCDR\xCD");
39         cout<<x->key<<endl;
40         print(prefix + (isLeft ? "\xBA" : " "), x->left, true);
41         print(prefix + (isLeft ? "\xBA" : " "), x->right, false);
42     }
43 }
44
45 void bst::insert(int k){
46     tree_node *x = root;
47     tree_node *y = NULL;
48     tree_node *z = new tree_node(k);
49     while(x != NULL){
50         y = x;
51         if(z->key < x->key)
52             x = x->left;
53         else
54             x = x->right;
55     }
56     z->parent = y;
57     if(y == NULL)
58         root = z;
59     else if(z->key < y->key)
60         y->left = z;
61     else
62         y->right = z;
63 }
64
65 int main(){
66     bst arbol;
67     arbol.insert(2); arbol.insert(1); arbol.insert(3); arbol.print();
68 }
```

1. Implementar, en el archivo de cabecera “bst.h” usando punteros y memoria dinámica, la clase “bst”, que contiene los siguientes métodos (1.5 puntos c/u, total = 13.5):

- `void insert(int k)` //inserta k en el bst.
- `void print()` //imprime de forma bonita el bst. -----> 
- `void print_inorder()` //imprime el recorrido inorden.
- `void print_postorder()` //imprime el recorrido postorden.
- `tree_node *search(int k)` //busca en el bst el nodo que contiene k y lo devuelve.
- `tree_node *min()` //busca en el bst el nodo que contiene el valor mínimo y lo devuelve.
- `tree_node *max()` //busca en el bst el nodo que contiene el valor máximo y lo devuelve.
- `tree_node successor(tree_node *x)` //encuentra el sucesor del nodo x.
- `tree_node predecessor(tree_node *x)` //encuentra el predecesor del nodo x.
- `void transplant(tree_node *x, tree_node *y)` //
- `void remove(tree_node *z)` //elimina el nodo z del bst

2. Implementar, en el archivo de cabecera “bst.h” los siguientes métodos: (1 punto c/u)

- `int size()` //cuenta el número de nodos.
- `int maxDepth()` //devuelve el número de nodos a lo largo de la raíz hasta el nodo más lejano, un árbol vacío tiene maxDepth()=0.
- `void caminos()` //imprimir el camino desde la raíz hacia c/u de sus hojas.
- `void espejo()` //cambia el árbol para que los roles de los punteros left y right se intercambien en cada nodo. De tal manera que el árbol espejo en recorrido inorden muestre los elementos del mayor al menor.
- `bool iguales(bst *x, bst *y)` //Dos árboles son iguales si están hechos de nodos con los mismos valores y dispuestos de la misma manera.
- `bool esBST()` //Para ser un árbol de búsqueda binaria: para cada nodo, todos los nodos en su sub-árbol izquierdo deben ser < que el nodo y todos los nodos del sub-árbol derecho deben ser > que el nodo.
- `int diametro()` //El diámetro de un árbol es la distancia máxima entre dos hojas.

3. Convertir un árbol binario de búsqueda en una lista simplemente enlazada circular. La lista resultante se debe encontrar en orden creciente, se debe realizar la transformación **SOLO** modificando punteros. Se deberá devolver un puntero que pasará a ser el **head** de la lista simplemente enlazada circular (4.5 puntos)

```
1: #include<iostream>
2: #include<string>
3:
4: using namespace std;
5:
6: struct tree_node
7: {
8:     int key;
9:     tree_node *parent;
10:    tree_node *left;
11:    tree_node *right;
12:    tree_node(int k)
13:    {
14:        key = k;
15:        left = NULL;
16:        right = NULL;
17:    }
18: };
19:
20: class bst
21: {
22: public:
23:     tree_node *root;
24:     bst()
25:     {
26:         root = NULL;
27:     }
28:     void insert(int k);
29:
30:     void print();
31:     void print(string prefix, tree_node *x, bool
isLeft);
32:
33:     void print_inorder();
34:     void print_inorder(tree_node *x);
35:
36:     void print_postorder();
37:     void print_postorder(tree_node *x);
38:
```

```

39:     tree_node *search(int k);
40:
41:     tree_node *min();
42:     tree_node *min(tree_node *x);
43:
44:     tree_node *max();
45:     tree_node *max(tree_node *x);
46:
47:     tree_node *successor(tree_node *x);
48:
49:     tree_node *predecessor(tree_node *x);
50:
51:     void transplant(tree_node *u, tree_node *v);
52:
53:     void remove(tree_node *z);
54:
55: };
56:
57: //_____print_inorder()
58: void bst::print_inorder()
59: {
60:     cout<<endl;
61:     print_inorder(root);
62:     cout<<endl;
63: }
64: void bst::print_inorder(tree_node *x)
65: {
66:     if(x != NULL)
67:     {
68:         print_inorder(x->left);
69:         cout<<x->key<<"\t";
70:         print_inorder(x->right);
71:     }
72: }
73:
74: //_____delete()
75: void bst::remove(tree_node *z)

```

```

76: {
77:     if(z->left == NULL)
78:         transplant(z, z->right);
79:     else if(z->right == NULL)
80:         transplant(z, z->left);
81:     else
82:     {
83:         tree_node *y = min(z->right);
84:         if(y != z->right)
85:         {
86:             transplant(y, y->right);
87:             y->right = z->right;
88:             y->right->parent = y;
89:         }
90:         transplant(z,y);
91:         y->left = z->left;
92:         y->left->parent = y;
93:     }
94: }
95: //_

```

transplant()

```

96: void bst::transplant(tree_node *u, tree_node *v)
97: {
98:     if(u->parent == NULL)
99:         root = v;
100:    else if(u == u->parent->left)
101:        u->parent->left = v;
102:    else
103:        u->parent->right = v;
104:    if(v != NULL)
105:        v->parent = u->parent;
106: }
107:
108: //_

```

predecessor()

```

109: tree_node *bst::predecessor(tree_node *x)
110: {
111:     if(x->left != NULL)
112:     {

```

```

113:         return max(x->left);
114:     }
115: else
116: {
117:     tree_node *y = x->parent;
118:     while(y != NULL and x == y->left)
119:         x = y;
120:         y = y->parent;
121:     return y;
122: }
123: }
124:
125: // _____ successor()
126: tree_node *bst::successor(tree_node *x)
127: {
128:     if(x->right != NULL)
129:     {
130:         return min(x->right);
131:     }
132: else
133: {
134:     tree_node *y = x->parent;
135:     while(y != NULL and x == y->right)
136:         x = y;
137:         y = y->parent;
138:     return y;
139: }
140: }
141:
142: // _____ min()
143: tree_node *bst::min()
144: {
145:     return min(root);
146: }
147:
148: tree_node *bst::min(tree_node *x)
149: {

```

```
150:     while(x->left != NULL)
151:         x = x->left;
152:     return x;
153: }
154:
155: //_____max()
156: tree_node *bst::max()
157: {
158:     return max(root);
159: }
160:
161: tree_node *bst::max(tree_node *x)
162: {
163:     while(x->right != NULL)
164:         x = x->right;
165:     return x;
166: }
167:
168: //_____search()
169: tree_node *bst::search(int k)
170: {
171:     tree_node *x = root;
172:     while(x != NULL and x->key != k)
173:         if(k < x->key)
174:             x = x->left;
175:         else
176:             x = x->right;
177:     return x;
178: }
179:
180: //_____print_postorder()
181: void bst::print_postorder()
182: {
183:     cout<<endl;
184:     print_postorder(root);
185:     cout<<endl;
```

```

186: }
187: void bst::print_postorder(tree_node *x)
188: {
189:     if(x != NULL)
190:     {
191:         print_postorder(x->left);
192:         print_postorder(x->right);
193:         cout<<x->key<<"\t";
194:     }
195: }
196:
197: //_____


---


198: void bst::print()
199: {
200:     cout<<endl;
201:     print("", root, false);
202:     cout<<endl;
203: }
204: void bst::print(string prefix, tree_node *x, bool
205: isLeft)
206: {
207:     if(x != NULL)
208:     {
209:         cout<<prefix;
210:         cout<<(isLeft ? "\xBA\xCD\xCDL\xCD\xCD" :
211: "\xC8\xCD\xCDR\xCD\xCD");
212:         cout<<x->key<<endl;
213:     }
214: }
215:
216: //_____


---


217: void bst::insert(int k) //recursivo
218: {

```

```
219:     tree_node *x = root;
220:     tree_node *y = NULL;
221:     tree_node *z = new tree_node(k);
222:     while(x != NULL)
223:     {
224:         y = x;
225:         if(z->key < x->key)
226:             x = x->left;
227:         else
228:             x = x->right;
229:     }
230:     z->parent = y;
231:     if(y == NULL)
232:         root = z;
233:     else if(z->key < y->key)
234:         y->left = z;
235:     else
236:         y->right = z;
237: }
238:
239: int main()
240: {
241:     bst arbol;
242:     tree_node *result;
243:
244:     arbol.insert(15);
245:     arbol.insert(6);
246:     arbol.insert(3);
247:     arbol.insert(2);
248:     arbol.insert(4);
249:     arbol.insert(7);
250:     arbol.insert(13);
251:     arbol.insert(9);
252:     arbol.insert(18);
253:     arbol.insert(17);
254:     arbol.insert(20);
255:
256:     arbol.print();
257:
```

```
258:     arbol.print_inorder();
259:
260:     arbol.print_postorder();
261:
262:     result = arbol.search(2);
263:     cout<<result->key<<endl;
264:
265:     result = arbol.max();
266:     cout<<result->key<<endl;
267:
268:     result = arbol.min();
269:     cout<<result->key<<endl;
270:
271:     result = arbol.search(13);
272:     result = arbol.successor(result);
273:     cout<<result->key<<endl;
274:
275:     result = arbol.search(9);
276:     result = arbol.predecessor(result);
277:     cout<<result->key<<endl;
278:
279:     result = arbol.search(15);
280:     arbol.remove(result);
281:
282:     arbol.print();
283:
284: }
```

```
285:
286: /*
287: ---R--15
```

```
288:      /--L--6
289:      |
290:      |
291:      |
292:      |
```

|--L--3

|--L--2

+--R--4

+--R--7

293: / *---R---13*

294: / *---L---9*

295: *---R---18*

296: /---L---17

297: *---R---20*

298: */