Práctico 2

String, Array, Vector & Queue

NOTA: Debe entregar el código completo de cada ejercicio en el archivo `edya2.h`, siguiendo los criterios aconsejados por la cátedra. En la primera línea de cada archivo de código fuente colocar nombre de grupo y miembros.

EJERCICIO 1

Utilizando el tipo de dato abstracto `string` provisto por C++ se pide listar todas las letras del string utilizando:

A. El método at().
B. El operador de indexación.
C. Iteradores.
D. Bucle en un rango (Range-based for loop, C++11).

EJERCICIO 2

Utilizando el tipo de dato abstracto `string` provisto por C++ se pide implementar una función que genere a partir de un string palíndromo un nuevo string con todos los substring palíndromos concatenados en forma de pirámide.

```
string palindromePyramid(const string& str);
```

Para la resolución debe utilizar los constructores y métodos de la clase string por ejemplo append, length, substr, etc. Es importante que lea la documentación de string.
Por ejemplo,

```
palindromePyramid("reconocer");
//"    n    \n   ono   \n  conoc  \n econoce \nreconocer"
```

La salida debe ser igual al ejemplo, si en una línea se agregan por ejemplo 3 espacios al principio también se deben agregar al final.

Salida por consola

```
    n
   ono
  conoc
 econoce
reconocer
```

EJERCICIO 3

Utilizando el tipo de dato abstracto `string` provisto por C++ se pide implementar una función que genere un nuevo string reemplazando todas las ocurrencias del substring `from` por el substring `to` respetando el siguiente prototipo:

```
string replaceAll(const string& str, const string& from, const string& to);
```

EJERCICIO 4

Utilizando el tipo de dato abstracto stack implementar una función que controle si un string es palíndromo.

```
bool palindrome(const string& str);
```

Agregue el correspondiente control al EJERCICIO 2 para solo construir la pirámide en el caso de que la entrada sea palíndroma.

EJERCICIO 5

Utilizando el tipo de dato abstracto `vector` provisto por C++ se pide:

1. Implementar una función que genere en forma aleatoria un vector de N números enteros positivos cuyo valor oscila en rango indicado `[from, to]`.

```
vector<unsigned int> create(unsigned int N, unsigned int from, unsigned int to);
```

2. Codifique una función que imprima los elementos de un vector de enteros en dos formatos. El primer formato es uno por cada línea (**FMT_LINE)** de pantalla usando `endl` y el segundo en notación de lista (**FMT_LIST)** [elemento1, elemento2, …., elementon], esté es el valor por defecto.

```
void print(vector<unsigned int> v, format f=FMT_LIST)
```

3. Implementar una función que ordene ascendentemente el vector de enteros pasado por parámetro.

```
void sort(vector<unsigned int> &v);
```

4. Implementar una función que permita crear un nuevo vector insertando los elementos de un vector dentro de otro en la posición indicada.

```
vector<unsigned int> insert(vector<unsigned int> v1, vector<unsigned int> v2,
unsigned int pos);
// Ejemplo si: v1=[1, 2, 3, 4], v2=[6, 7, 8]
v3 = insert(v1, v2, 1)
// v3 = xxx[1, 6, 7, 8, 2, 3, 4]
```

5. Implementar una función que liste los números contenidos en el vector y la cantidad de ocurrencias.

```
void print_frequency(vector<unsigned int> v);
```

EJERCICIO 6

Describa tres aplicaciones de la vida real donde se usen filas.

EJERCICIO 7

Implementar una función que imprima el contenido de una fila de enteros respetando el siguiente prototipo:

```
void print(std::queue<int> &myqueue);
```

EJERCICIO 8

Implementar una función que invierta el contenido de una fila de enteros respetando el siguiente prototipo:

```
std::queue<int>& reverse(std::queue<int> &myqueue);
```