Algoritmos y Estructuras de Datos I

Trabajo Práctico N°4	Unidad 4
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo grupal
Metodología de Desarrollo: Det. docente	**Metodología de Corrección:** Via Classroom.
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

MARCO TEÓRICO

Responder el siguiente cuestionario en función de la bibliografía Obligatoria.

Unidad 4

1. ¿Describir el Concepto de Array?

Un array es uno de los tipos de datos estructurados apoyados por C++, es un tipo de datos que nos permite programar operaciones de este tipo con facilidad.

Como estructura de datos, un array difiere de una estructura o una clase en dos formas fundamentales:

1. Un array es una estructura de datos homogénea (todos los componentes son del mismo tipo de datos), mientras que estructuras y clases son tipos heterogéneos (sus componentes pueden ser de diferentes tipos).

2. El acceso a un componente de un array es por medio de su posición en el arreglo, mientras que el acceso a un componente de una estructura o clase es por medio de un identificador (el nombre del miembro).

En la plantilla de sintaxis, DataType describe lo que está almacenado en cada componente del array. Los componentes de arrays pueden ser de casi todos los tipos.

ArrayDeclaration

```
DataType  ArrayName [ ConstIntExpression ] ;
```

Array unidimensional Colección estructurada de componentes, todos del mismo tipo, que recibe un solo nombre. Cada componente (elemento de array) es accesible por medio de un índice que ubica la posición del componente dentro de la colección.

2. Dar un Ejemplo de:

1. **Creación de Arrays (preferentemente con typeof)**

La creación se hace usando el typedef double tTemp [Dias]; Su valor está definido como una constante de Días = 7

2. **Asignación de Arrays**

La asignación se declara con tTemp, de double[Dias]

3. **Recorrido de Arrays.**

El recorrido de la array se hace al ingresar las temperaturas en cada día

La sentencia Typedef como una forma de asignar un nombre adicional a un tipo de datos existente. También podemos usar Typedef para asignar un nombre a un tipo de array.

```
#include <iostream>
using namespace std;
const int Dias = 7;
```

```
 // se usa tTemp, para Días. Su valor está definido como una constante de Dias = 7
//El tTemp quiere decir, double [Dias]
typedef double tTemp[Dias];
double media(const tTemp temp);          // la función calcula la media de temperatura
int main() {
tTemp temp;
// Recorrido del array. Asigna valores al array  mediante un bucle for
for (int i = 0; i < Dias; i++) {
cout << "Temperatura del día " << i + 1 << ": ";
cin >> temp[i];
  }
//impresión usando función media
cout << "Temperatura media: " << media(temp) << endlreturn 0;
}
...
```

3. Describir el Concepto de Listas (sin estructura)

Sin estructura es cuando no se necesita algo específico, como un array o una estructura definida por el usuario de nodos. Los elementos almacenados, que pueden ser de diferentes tipos, esta colección dinámica está dentro de una lista.

La lista maneja los detalles de cómo se almacenan y gestionan los elementos, lo cual facilita el proceso de comparación.

Lista es una colección de longitud variada y lineal de componentes homogéneos. Uno de los usos para un array es el de almacenar una lista de valores. Una lista podrá contener menos valores que el número de lugares reservados en el array.

Tenemos que distinguir entre listas cuyos componentes siempre se deben mantener en orden alfabético o numérico (listas ordenadas) y listas en las cuales los componentes no estén dispuestos en algún orden particular (listas no ordenadas).

1. **Dar un ejemplo de Inserción de un elemento**

insertar el n° 5:

```
#include <iostream>
int main() {
   int* inicio = nullptr;  // Inicializamos la lista vacía
   // Insertamos el número 5 al principio de la lista
   int numero = 5;
int* nuevoElemento = new int;  // Creamos un nuevo espacio en memoria para el n°
   *nuevoElemento = numero;  // Almacenamos el número en ese espacio
   nuevoElemento[1] = aunque; inserción
```

2. **Dar un ejemplo de Borrado de un elemento**

Este ejemplo inserta el número y luego borra el 5 si está presente

```
#include <iostream>
int main() {
   int* lista = nullptr;  // Inicializamos la lista vacía
   // Insertamos algunos números en la lista
   for (int i = 1; i <= 10; ++i) {
 int* nuevoElemento = new int;  // Creamos un nuevo espacio en memoria para el n°
 *nuevoElemento = i;  // Almacenamos el número en ese espacio
  nuevoElemento[1] = 5;  //
```

4. Describir el Tipo de Dato “String”

String es un tipo definido por el programador que es suministrado por la biblioteca estándar C++, una colección grande de funciones y tipos de datos prescritos que cualquier programador de C++ puede usar. Las operaciones en datos string incluyen comparar los valores de string, buscar una cadena para un carácter particular y unir una cadena a otra.
El archivo iostream contiene declaraciones de medios de entrada o salida, y el archivo string contiene declaraciones acerca del tipo de datos string.
Esta cadena se gestiona de manera aumatica de la memoria, tiene una multitud de funciones en la biblioteca. El tipo string define operadores para concatenar de manera segura.
Su uso requiere un espacio a std, para que inicie la declaración, para copiar con el operador la asignación, para multitud de funciones a concatenar.

1. **Dar ejemplos de 4 Funciones “Incluidas” en el Tipo “String”**

función length

La función length, cuando se aplica a una variable string, devuelve un valor entero sin signo que es igual al número de caracteres actualmente en la cadena.

```
string firstName; string fullName;
 first Name = "Alexandra"; cout << first Name.length() << endl;      // Prints 9 fullName =
firstName + " Jones"; cout << fullName.length() << endl;
```

función size

Algunas personas se refieren a la longitud de una cadena como su tamaño. Para acomodar ambos términos, el tipo string proporciona una función llamada size.
El tipo string define un tipo de datos size_type para que se pueda usar:

```
string firstName; string::size_type len;
first Name = "Alexandra"; len = first Name.length();
```

Función find

La función find busca una cadena para hallar la primera concurrencia de una subcadena particular y devuelve un valor entero sin signo (de tipo string:: size_type) que da el resultado de la búsqueda.
Si str1 y str2 son de tipo string, las siguientes son llamadas de función válidas:
 str1.fi nd("the") str1.fi nd(str2) str1.fi nd(str2 + "abc")

```
string str1;
string str2;
str1 = "Programming and Problem Solving"; str2 = "gram";
```

Función substr

La función substr devuelve una subcadena particular de una cadena. Suponiendo que myString es de tipo string, aquí se da una llamada de función: **myString.substr(5, 20)**
El primer argumento es un entero sin signo que especifica una posición dentro de la cadena. El segundo es un entero sin signo que especifica la longitud de la subcadena deseada. La función devuelve la parte de la cadena que empieza con la posición especificada y continúa para el número de caracteres dados por el segundo argumento. Observe que substr no cambia a myString; devuelve un nuevo valor string temporal, que es copia de una parte de la cadena.

```
#include <iostream>
#include <string>     // Para tipo string
using namespace std;
int main() {
   string fullName;     string name;     string::size_type startPos;
    fullName = "Jonathan Alexander Peterson";     startPos = fullName.fi nd("Peterson");
name = "Mr. " + fullName.substr(startPos, 8);     cout << name << endl;
   return 0;
}
```

Llamada de función (`s` es de tipo `string`)	Tipo(s) de argumento(s)	Tipo de resultado	Resultado (valor obtenido)
`s.length()` `s.size()`	Ninguno	`string::size_type`	Número de caracteres en la cadena
`s.find(arg)`	`string`, cadena literal, o `char`	`string::size_type`	Posición de inicio en `s` donde se encontró `arg`; si no se encontró, el resultado es `string::npos`
`s.substr(pos,len)`	`string::size_type`	`string`	Subcadena de cuando mucho `len` caracteres, comenzando en la posición `pos` de `s`. Si `len` es demasiado grande, significa "hasta el fin" de la cadena `s`. Si `pos` es demasiado grande, se termina la ejecución del programa.*

2. Que sentencia de Entrada usaría para leer un String con espacio x teclado

Para introducir una cadena de caracteres en una variable string, se tienen dos opciones. La primera es usar el operador de extracción (>>). Al leer los caracteres de entrada en una variable string, el operador >> omite cualquier carácter de espacio en blanco inicial. Después lee caracteres sucesivos hacia la variable, deteniéndose en el primer carácter de espacio en blanco posterior.

Aunque el operador >> se emplea ampliamente para introducción de cadena, tiene una posible desventaja: no se puede usar para introducir una cadena con espacios en blanco en su interior.

Este hecho conduce a la segunda opción para llevar a cabo la introducción de cadena: la función getline. Una llamada a esta función se parece a esto:

```
getline(cin, myString);
```

La función getline no omite los caracteres de espacio en blanco iniciales y continúa hasta que llega al carácter de nueva línea '\n'. Es decir, getline lee y almacena una línea de entrada completa, espacios en blanco incrustados y todo. Observe que con getline, el carácter de nueva línea se consume (pero no se almacena en la variable de cadena)

Sentencia	¿Omite el espacio en blanco principal?	¿Cuándo se detiene la lectura?
`cin >> inputStr;`	Sí	Cuando se encuentra un carácter de espacio en blanco posterior (que *no* es consumido)
`getline(cin, inputStr);`	No	Cuando se encuentra '\n' (que *es* consumido)

5. Describir la sentencia de repetición Do While

Do-While, hace más fácil programar algunos tipos de ciclos. La sentencia Do-While es una estructura de control de iteración en la que se prueba la condición de
ciclo al final (parte baja) del ciclo. Este formato garantiza que el cuerpo del ciclo se ejecute por lo menos una vez.

Observe que la solución Do-While no requiere que el indicador y los pasos de entrada aparezcan dos veces —una antes del ciclo y otra dentro de él—, sino que prueba el valor de entrada dos veces. Es posible usar también Do-While para poner en práctica un ciclo controlado por conteo si se sabe por adelantado que el cuerpo del ciclo se debe ejecutar siempre por lo menos una vez.

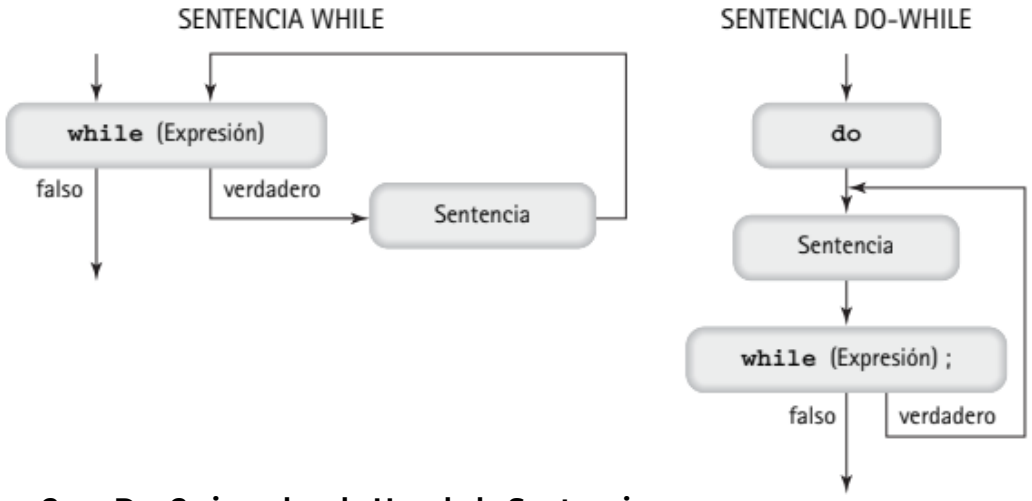
La sentencia

```
do {
Sentencia1;
Sentencia2;
...
SentenciaN;
} while (Expresión);
```

Significa “Ejecute las sentencias entre do y while siempre que la expresión aún tenga el valor true al final del ciclo”.

1. **Qué diferencia tiene con la sentencia While**

While se usa para repetir un curso de acción. La solución While requiere una lectura principal para que inputChar tenga un valor antes de que se introduzca el ciclo. Esto no es necesario para la solución Do-While porque la sentencia de entrada dentro del ciclo se ejecuta antes de que se evalúe la condición.



2. **Dar 2 ejemplos de Uso de la Sentencia.**

do **{** **cout << "Introduzca su edad: ";** **cin >> age;** **if (age <= 0)** **cout << "Su edad debe ser positiva." << endl;** **} while (age <= 0);**	**sum = 0;** **counter = 1; do** **{** **um = sum + counter;** **counter++; }** **while (counter <= n);**	**do** **dataFile >> inputChar; while** **(inputChar != '.');**

MARCO PRÁCTICO

Resolver y Realizar en c++

a. Realizar un programa Que rellene un array con los 100 primeros números enteros y los muestre en pantalla
b. Realizar un Programa Que rellene un array con los números primos comprendidos entre 1 y 100 y los muestre en pantalla
c. Realizar un Programa Que rellene un array con los números impares comprendidos entre 1 y 100 y los muestre en pantalla