APÉNDICE 1
EJERCICIOS

1.1. Introducción

1.1.1. Presentación de los enunciados

En este apéndice se proponen ejercicios con diversos niveles de dificultad. Dada la gran cantidad de ejercicios, los mismos están catalogados según su nivel de dificultad y su importancia de resolverlo.

- Dificultad: Básica, Moderada, Intermedia, Alta, Extrema.
- Requerido: Indispensable, Recomendable.

1.1.2. Modo de trabajo

Para cada uno de los siguientes ejercicios:

1. Determinar cuáles son los datos de entrada, contexto y salida.
2. Clasificar según su multiplicidad y tipo de procesamiento (vertical, horizontal o ambos).
3. Diseñar el algoritmo que resuelve el ejercicio, diagramarlo, codificarlo y probar su correcto funcionamiento.

1.2. Lección 1

1.2.1. Ejercicios

1.2.1.1. Operaciones aritméticas

Dificultad: básica, *Requerido*: Indispensable.

Se ingresan por teclado dos valores numéricos enteros, *a* y *b*, se pide calcular e informar por consola el resultado de las siguientes operaciones:

1. La suma: *a*+*b*.
2. La diferencia: *a*-*b*.
3. El producto: *a***b*.
4. El cociente: *a*/*b*, aceptando que *b* es distinto de 0 (cero).

1.2.1.2. Cociente entre dos números

Dificultad: básica, *Requerido*: Indispensable.

Se ingresan por teclado dos valores numéricos enteros: *a* y *b*, se pide calcular e informar el cociente *a*/*b*, validando que *b* sea distinto de cero. En tal caso, mostrar un mensaje de error en la consola.

1.2.1.3. División entera y módulo

Dificultad: básica, *Requerido*: Indispensable.

Se ingresa por teclado un valor numérico entero, informar:

1. La quinta parte de dicho valor.
2. El resto que surge al dividir el valor ingresado en cinco partes iguales.
3. La séptima parte de la quinta parte del valor ingresado.

1.2.1.4. Mayor valor entre dos números

Dificultad: básica, *Requerido*: Indispensable.

Se ingresa por teclado dos valores numéricos enteros diferentes entre sí, informar cuál es el mayor.

1.2.1.5. Mayor y menor valor entre dos números

Dificultad: básica, *Requerido*: Indispensable.

Se ingresa por teclado dos valores numéricos enteros, informar cuál es el mayor y cuál el menor. Si son iguales, entonces, mostrar un mensaje con el siguiente texto: "Los valores ingresados son iguales".

Programación APT (A Prueba de Tontos)

En la jerga de los programadores, existe el concepto de programar APT o No APT (A Prueba de Tontos); referido a si corresponde (o no) que el programador se ocupe de validar que el ingreso de datos coincida con lo esperado.

Si programamos APT aceptamos que el usuario *es un tonto* capaz de ingresar letras donde esperamos números, o valores negativos donde esperamos números positivos. Por eso, preparamos el programa para validar casos de este tipo, y eventualmente mostraremos los mensajes de error que correspondan.

En cambio, si programamos No APT, nos desentendemos de la responsabilidad de validar el ingreso de los datos, porque aceptamos que *el usuario no es ningún tonto*. Si le pedimos que ingrese valores diferentes entre sí, lo hará; si le pedimos un número entero suponemos que no ingresará una cadena de caracteres, etcétera.

En este curso, siempre trabajaremos No APT. No validaremos ningún caso de error, salvo que el enunciado del ejercicio lo requiera explícitamente.

Sin embargo, en la vida real sí debemos programar A Prueba de Tontos.

1.2.1.6. Mayor, medio y menor valor entre tres números

Se ingresan tres valores numéricos enteros, diferentes entre sí, informar cuál es el menor, cuál está en el medio y cuál es el mayor.

1.2.1.7. Tipo de triángulo según sus lados

Dificultad: básica, *Requerido*: Indispensable.

Se ingresan tres valores que representan las longitudes de los lados de un triángulo, informar cuál es el tipo del triángulo ingresado (isósceles, equilátero o escaleno).

1.2.1.8. Separar los atributos de una fecha

Dificultad: básica, *Requerido*: Indispensable.

Dado un número de ocho dígitos que representa una fecha con formato *aaaammdd*, se pide mostrar por separado el día, el mes y el año de la fecha ingresada.

Usar los operadores / (división) y % (módulo)

Ejemplo:

```
int f = 20201230;   // 2020/12/30
int anio = f/10000; // anio vale: 2020
int dia = f%100     // dia vale: 30
```

1.2.1.9. Unificar los atributos de una fecha

Dificultad: básica, *Requerido*: Indispensable.

Dada una terna de números naturales que representan el día, mes y año de una fecha, se pide unificarlos en un único valor numérico entero de ocho dígitos (*aaaammdd*), tal que los primeros cuatro dígitos representen el año, los dos siguientes el mes, y los dos últimos el día.

1.2.1.10. Fecha más próxima

Dificultad: moderada, *Requerido*: Indispensable.

Entre dos fechas indicadas por el usuario, informar cuál es la más cercana a la actual.

Se debe establecer:

1. Cuáles son los datos de entrada que el algoritmo necesitará para resolver el problema planteado.
2. En qué formato el usuario deberá ingresar dichos datos de entrada.

Se debe considerar los años bisiestos, entendiendo que un año es bisiesto si es divisible por 4, o por 400 pero no por 100.

Obtener el valor absoluto de un número

La función `abs`, contenida en la biblioteca `stdlib.h`, retorna el valor absoluto del valor que recibe como parámetro.

```
int a = -1;
int b = abs(a); // b vale: 1
```

1.2.1.11. Cuántos días tiene un mes

Dificultad: básica, *Requerido*: Indispensable.

Se ingresan dos valores numéricos enteros que corresponden a un mes y un año (1 para enero, 2 para febrero, etcétera), se pide informar cuántos días tiene el mes; teniendo en cuenta que el año podría ser bisiesto.

1.2.1.12. Producto mediante sumas sucesivas

Dificultad: básica, *Requerido*: Indispensable.

Dados dos valores numéricos enteros, informar su producto calculándolo mediante sumas sucesivas.

1. Considerando que los valores ingresados serán números positivos o cero.
2. Considerando que los valores ingresados también podrían ser negativos.

1.2.1.13. Factorial de un número

Dificultad: básica, *Requerido*: Indispensable.

Dado un valor numérico entero positivo, informar su factorial.

NOTA: El factorial de un número *n* (se indica *n*!) se calcula así: n * *n*-1 * *n*-2 * … * 3 * 2 * 1. El factorial de 0 es 1. Por ejemplo: 5! es: 120, 4! es: 24 y 0! es: 1.

1.2.1.14. Números primos

Dificultad: básica, *Requerido*: Indispensable.

Dado un valor entero positivo, informar si es un número primo.

NOTA: Un número es primo si sólo es divisible por sí mismo y por 1.

1.2.1.15. Primeros números primos

Dificultad: básica, *Requerido*: Indispensable.

Dado un valor numérico entero positivo *n*, informar los primeros *n* números primos. Por ejemplo: si *n*==6 entonces la salida debe ser: 1, 2, 3, 5, 7, 11.

1.2.1.16. Número de Fibonacci

Dificultad: básica, *Requerido*: Indispensable.

Dado un valor numérico entero positivo *n*, informar el *n*-ésimo término de la sucesión de Fibonacci. Por ejemplo: si *n* es 6, la salida del programa debe ser 8.

NOTA: Los primeros dos términos de la serie de Fibonacci son 1 y 1. Luego, cada término se calcula como la suma de los dos términos anteriores. Así, los primeros términos de la serie de Fibonacci son: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, etcétera.

1.2.1.17. Múltiplos

Dificultad: básica, *Requerido*: Indispensable.

Dado un valor numérico entero positivo *n*, informar los primeros *n* múltiplos de 5 que no sean múltiplos de 3.

Por ejemplo: si *n* fuera 6, la salida deberá ser: 5, 10, 20, 25, 35, 40.

1.2.1.18. Factorial de los primeros números naturales

Dificultad: básica, *Requerido*: Indispensable.

Dado un valor numérico entero positivo *n*, se pide mostrar el factorial de los primeros *n* números naturales.

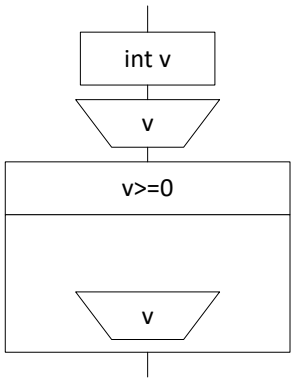
Por ejemplo: si *n*=7, entonces *la salida debe ser:* 1, 2, 6, 24, 120, 720, 5040.

Ejercicios de múltiples registros

Los ejercicios con los que hemos estado trabajando son ejercicios de *registro único*, pues en todos los casos se le solicitó al usuario ingresar un único registro compuesto por una pequeña cantidad de valores.

A partir de aquí, comenzamos a resolver ejercicios de múltiples registros, que requerirán leer y procesar individualmente los datos de cada uno de los múltiples registros de entrada que el usuario ingresará.

Por ejemplo, en un ejercicio cuyo enunciado comienza diciendo: se ingresa un conjunto de valores numéricos enteros positivos, que finaliza cuando se ingresa un valor 0 (cero) o negativo, utilizaremos una estructura como la siguiente para procesar los valores del conjunto que el usuario ingresará:




```
int v;
cin >> v;

while( v>=0 )
{
   // :
   // procesar el valor leido...
   // :

   cin >> v;
}
```

Figura 1.15. Patrón algorítmico que permite procesar múltiples registros

La variable `v` tomará, uno a uno, los valores del conjunto a medida que el usuario los vaya ingresando.

1.2.1.19. Docenas

Dificultad: básica, *Requerido*: Indispensable.

Se ingresan por teclado varios valores enteros positivos, de a uno por vez. Se solicita informar:

1. Cuántos valores *v* fueron ingresados, tal que *v*<=12.
2. Cuántos valores *v* fueron ingresados, tal que12<*v*<=24.
3. Cuántos valores *v* fueron ingresados, tal que 24<v<=36.
4. Cuántos valores *v* fueron ingresados, tal que *v*==0 (cero).

La carga de datos finaliza cuando se ingresa un valor negativo.

1.2.1.20. Cantidades, promedios y porcentajes

Dificultad: básica, *Requerido*: Indispensable.

Se ingresan varios valores numéricos enteros, finalizando la carga de datos al ingresar un 0 (cero). Se pide informar:

1. Cantidad positivos.
2. Cantidad de negativos.
3. Porcentaje de pares.
4. Promedio de los positivos.
5. Porcentaje de negativos.

1.2.1.21. Mayores y menores que

Dificultad: básica, *Requerido*: Indispensable.

Dados 50 valores numéricos enteros, que se ingresan de a uno por vez, se pide informar el promedio de los mayores que 100, y la suma de los menores que -10.

1.2.1.22. Primeros múltiplos flexible

Dificultad: básica, *Requerido*: Indispensable.

Dados tres valores numéricos enteros positivos: *n*, *a* y *b*, informar el *n*-ésimo múltiplo de *a* que no es múltiplo de *b*.

Por ejemplo: si *n*=10, *a*=5, *b*=3 entonces el *n*-ésimo múltiplo de 5 que no es múltiplo de 3 es: 70; y surge de la siguiente lista:

Múltiplos de 5 = { 5, 10, ~~15~~, 20, 25, ~~30~~, 35, 40, ~~45~~, 50, 55, ~~60~~, 65, **70** }

1.2.1.23. Máximos

Dificultad: básica, *Requerido*: Indispensable.

Cada uno de los siguientes ítems debe considerarse como un ejercicio en sí mismo.

1. Dados 100 valores enteros positivos, informar cuál es el mayor.
2. Dados 100 valores enteros, todos mayores que -10, informar cuál es el mayor.
3. Dados 100 valores enteros, informar cuál es el mayor.

1.2.1.24. Mínimos

Dificultad: básica, *Requerido*: Indispensable.

Cada uno de los siguientes ítems debe considerarse como un ejercicio en sí mismo.

1. Dados 100 valores enteros negativos, informar cuál es el menor.
2. Dados 100 valores enteros menores que 10, informar cuál es el menor.
3. Dados 100 valores enteros, informar cuál es el menor.

1.2.1.25. Mayor de los negativos, menor de los positivos

Dificultad: moderada, *Requerido*: Indispensable.

Dado un conjunto de valores numéricos que finaliza con el ingreso de un 0 (cero), informar cuál es el mayor de los negativos, y cuál el menor de los positivos.

1.2.1.26. Mínimo valor dentro de un intervalo

Dificultad: moderada, *Requerido*: Indispensable.

Sea un conjunto de valores numéricos que finaliza al ingresar un 0 (cero), informar cuál es el mínimo valor considerando sólo aquellos que pertenecen al intervalo [-16, 27].

1.2.1.27. Persona más joven, persona más vieja

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa un conjunto de pares (*n*, *f*), donde *n* es el nombre de una persona y *f* su fecha de nacimiento, informar el nombre de la persona más joven y el de la más vieja.

1.2.1.28. Conjuntos y subconjuntos

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresan *n* conjuntos de *m* valores numéricos cada uno. Se pide informar:

1. Para cada uno de los *n* conjuntos:
    a. El valor promedio.
    b. El máximo valor.
    c. Porcentaje de valores positivos.
2. Para todo el lote de datos:
    a. Valor promedio.

b. Porcentaje de valores negativos.
c. Valor mínimo.

1.2.1.29. Conjunto dividido por valores 0 (cero)

Dificultad: moderada, *Requerido*: Indispensable.

Se dispone de un conjunto de valores enteros positivos cuyo ingreso finaliza con la llegada de un número negativo.

El conjunto está dividido en subconjuntos, separados entre sí mediante valores 0 (cero). Se pide informar:

1. Por cada subconjunto:
   a. Promedio de sus valores.
   b. Valor mínimo.
2. Para el conjunto completo:
   a. Cantidad de subconjuntos.
   b. Sumatoria de sus valores.
   c. Número del subconjunto en el que se ingresó el mayor valor (será único), indicando también cuál fue ese valor y en qué posición relativa se encontró.

1.2.1.30. Seguidilla

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa un conjunto de valores numéricos enteros que finaliza con la llegada de un 0 (cero), se pide informar:

1. Cuántas veces se ingresaron valores consecutivos ascendentes. A esto, en adelante, lo llamaremos *seguidilla*.
2. Por cada seguidilla, qué cantidad de elementos la componen.
3. En qué posición relativa aparece la seguidilla más larga.

Por ejemplo, supongamos que los datos que ingresa el usuario son los siguientes:

5 3 1 4 6 8 3 1 4 5 6 7 9 4 6 8 4 2 1 3 5 6 7 2 0

La salida que deberá arrojar el programa será:

1. Cantidad de seguidillas: 4.
2. 4 elementos (seguidilla azul), 6 elementos (seguidilla verde), 3 elementos (seguidilla roja), 5 elementos (seguidilla violeta).
3. La seguidilla más larga se ingresó en la posición relativa: 2.

Tipo de dato char

El tipo `char` permite declarar variables preparadas para contener un carácter. Por ejemplo: `char c='A'`. En este caso, la variable `c` contiene el carácter A.

Veamos el siguiente ejemplo:

```
cout<< "Ingrese la inicial de su nombre:" <<endl;
char c;
cin>> c;  // el usuario ingresa un caracter

cout<< "Su nombre comienza con la letra: " << c <<endl;
```

1.2.1.31. Palabras dentro de una oración

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa una serie de caracteres que conforman las palabras de una oración. Cada palabra está separada de la siguiente por medio de un carácter `' '` (espacio). La oración finaliza cuando se ingresa un `'.'`.

Se pide informar:

1. Cantidad de veces que apareció cada vocal.
2. Cantidad de palabras que contiene la oración.
3. Cantidad de letras que posee la palabra más larga.
4. Además, indicar si la longitud de las palabras que componen la oración es creciente; ejemplo: “Sí que siempre conseguís maravillosas oportunidades”.

NOTA: Los caracteres de la oración deben ingresarse uno por uno a través del teclado. Si se utiliza Eclipse es probable que no pueda ingresar el carácter `' '`

(espacio); entonces, se recomienda asumir que las palabras de la oración están separadas entre sí por medio de carácter `'_'` (guion bajo).

1.2.2. Problemas

Para cada uno de los siguientes problemas:

1. Determinar cuáles son los datos de entrada, contexto y salida.
2. Clasificar en función de su multiplicidad y tipo de procesamiento (vertical, horizontal o ambos).
3. Diseñar el algoritmo que resuelve el ejercicio, diagramarlo, codificarlo y probar su correcto funcionamiento.

1.2.2.1. Cantidad de billetes de cada denominación

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa un valor numérico entero que representa el sueldo de una persona. Informar qué cantidad de billetes de cada una de las denominaciones existentes ($100, $50, $20, $10, $5, $2, $1) se debe utilizar para abonarlo.

1.2.2.2. Cantidad de billetes de cada denominación (versión 2)

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa un conjunto de valores numéricos enteros que representan los sueldos de un conjunto de personas. Informar qué cantidad de billetes de cada una de las denominaciones existentes ($100, $50, $20, $10, $5, $2, $1) se requiere disponer para el día de pago.

1.2.2.3. Carga de contenedores

Dificultad: moderada, *Requerido*: Indispensable.

Un buque de carga traslada 100 contenedores a tres diferentes puertos del país, que se identifican como 1, 2 y 3.

De cada contenedor se registran los siguientes datos:

- Identificación: `idCont`.
- Peso (en kilos): `peso`.
- Puerto de destino: `idPuerto`.

Se pide calcular e informar:

- El peso total que el buque debe trasladar.
- El contenedor de mayor peso.
- Cuántos contenedores se trasladarán a cada puerto.

1.2.2.4. Fábrica de pintura

Dificultad: moderada, *Requerido*: Indispensable.

Luego de una inspección en una fábrica de pintura, se detectaron algunas infracciones. De cada infracción se anotaron los siguientes datos:

- Tipo de Infracción (1, 2, 3, o 4): `tInfr`.
- Motivo de la infracción: `motivo`.
- Valor de la multa: `valor`.
- Gravedad de la infracción ('L','M', 'G'): `gravedad`.

Los datos finalizan cuando se ingrese un `tInfr` igual a cero. Al final del proceso, se requiere informar:

- El total a pagar, discriminado según la gravedad de las infracciones.
- La leyenda "Clausurar Fábrica" si la cantidad de infracciones 3 y 4 con gravedad 'G' es mayor a 3.
- El motivo por el que se aplicó la infracción de menor valor.

1.2.2.5. Encuesta en casas de familia

Dificultad: moderada, *Requerido*: Indispensable.

El Gobierno de la Ciudad de Buenos Aires realiza una encuesta en casas de familia. De cada familia se conoce:

- Domicilio,
- Tipo de vivienda, que puede ser: 'C' (casa) o 'D' (departamento), y
- Cantidad de integrantes.

Además, por cada integrante de la familia se conoce:

- Apellido y nombre,

- Edad,
- Sexo (‘M’ o ‘F’),
- Nivel de estudios: ‘N’ (no posee), ‘P’ (primario), ‘S’ (secundario), ‘T’ (terciario) o ‘U’ (universitario); y un indicador que será ‘I’ (incompleto) o ‘C’ (completo) que hace referencia al ítem anterior.

Los datos concluyen cuando se ingresa una cantidad de integrantes igual a cero. Se pide emitir un listado con los siguientes resultados:

- Los datos de quienes que hayan completado los estudios primarios.
- Porcentaje de analfabetismo en la ciudad (se considera analfabetos a los mayores de 10 años que no posean estudios).
- Domicilio de la familia que, teniendo la mayor cantidad de integrantes, vive en un departamento.
- Edad promedio de cada familia y de la ciudad.
- Por cada nivel de estudio, cuántos encuestados lo tienen incompleto.
- Porcentaje de encuestados de cada sexo.

1.2.2.6. Compañía aérea

Dificultad: moderada, *Requerido*: Indispensable.

Una compañía de aviación desea emitir un listado detallando los movimientos mensuales de sus *m* vuelos al exterior. Para ello, cuenta con la siguiente información:

- De cada vuelo realizado, su número, destino, y cantidad de asientos.
- De cada pasajero, pasaporte e importe abonado por el pasaje.

Se ingresan los datos de un vuelo, y por cada uno se ingresa la lista completa de sus pasajeros. Dicha lista concluye cuando se ingresa un número de pasaporte igual a cero.

Se requiere generar el siguiente listado:

Nro. de vuelo: 9999, *Destino*: xxxxxxxxxxxxxxxxxxxx

Número de pasaporte	*Importe*
9999999999999999	9999,99

```
9999999999999999          9999,99
9999999999999999          9999,99
          :                   :
Total recaudación vuelo: $99999,
Porcentaje de asientos libres: 99%
Porcentaje de asientos ocupados: 99%

:      :      :      :      :      :      :      :
Total recaudación mensual: $9999,99
Mayor cantidad de veces seguidas que se dieron vuelos completos: 9999
Número de vuelo que más recaudó: 999
```

1.2.2.7. Pagos

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa un conjunto de ternas (*d*, *p*, *i*), donde *d* es el DNI de una persona, *p* la fecha de pago e *i* el importe pagado en la fecha *p*. El lote de datos se ingresa ordenado (agrupado) de acuerdo con el número del DNI. De este modo, se garantiza que todos los pagos efectuados por una misma persona se ingresarán juntos (uno detrás del otro).

Se pide:

1. Por cada persona:
    a. Cantidad de pagos efectuados.
    b. Importe promedio de los pagos.
    c. Cantidad total abonada.
2. En general:
    a. Cuántas personas efectuaron al menos un pago.
    b. Promedio de los importes pagados.
    c. DNI de la persona que, en total, abonó la mayor cantidad de dinero.

1.3. Lección 2

1.3.1. Ejercicios

1.3.1.1. Función factorial

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar la función `factorial` que calcula y retorna el factorial de `n`.

```
double factorial(int n);
```

Invocando a `factorial`, desarrollar un programa que muestre el factorial de los primeros *t* números naturales, donde *t* es un valor que se ingresa por consola.

1.3.1.2. Función esPrimo

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar la función `esPrimo` que retorna `true` o `false` según se determine que `n` es un número primo o no:

```
bool esPrimo(int n);
```

Invocando a la función `esPrimo`, desarrollar un programa que muestre por consola los primeros *t* números primos, donde *t* es un valor que ingresa el usuario.

1.3.1.3. Cuántos días tiene el mes

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar las siguientes funciones:

```
bool fechaEsAnioBisiesto(int a);
```

La función debe retornar `true` o `false` según el año `a` que recibe como parámetro sea o no bisiesto.

```
int fechaDiasMes(int m,int a);
```

La función debe retornar la cantidad de días que tiene el mes `m`, donde 1 es enero; 2, febrero, etcétera.

Invocando a las funciones anteriores, desarrollar un programa que, dado un mes y año que el usuario ingresará por teclado, le indicará cuántos días tiene el mes.

1.3.1.4. Unificar los atributos de una fecha

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar la siguiente función:

```
int fechaUnificar(int anio,int mes,int dia);
```

La función debe unificar los valores de `anio`, `mes` y `dia` en un número entero de ocho dígitos, con formato *aaaammdd*. Por ejemplo: si `anio` fuese 2020, `mes` 10 y `dia` 15, entonces, el valor de retorno de `fechaUnificar` será: 20201015.

Invocando a `fechaUnificar`, desarrollar un programa donde el usuario ingresa el día, mes y año de una fecha (como valores separados), y obtiene por consola la fecha unificada con formato *aaaammdd*.

1.3.1.5. Separar los atributos de una fecha

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar la siguiente función:

```
void fechaDividir(int f,int& a,int& m,int& d);
```

La función recibe en `f` una fecha representada como un entero de ocho dígitos con formato *aaaammdd*, y debe dividir sus componentes y asignarlos a los parámetros `a`, `m` y `d` (año, mes y día, respectivamente).

Invocando a `fechaDividir`, desarrollar un programa donde el usuario ingresa una fecha con formato *aaaammdd* y obtiene sus atributos por separado.

Obtener la fecha del sistema

Los lenguajes de programación proveen un modo para obtener la fecha del sistema. En C++ acceder a esta información resulta algo engorroso por lo que, a continuación, desarrollaremos la función `getDate` que devuelve el día, mes y año actual en tres parámetros que recibe por referencia.

No es importante entender el código que expondremos a continuación, únicamente debemos saber que desde ahora, cada vez que sea necesario, podremos obtener la fecha del sistema invocando a la función `getDate`.

```
// asigna la fecha actual a los parametros dia, mes y anio
void getDate(int& dia, int& mes, int& anio)
{
   // fecha actual expresada en segundos
   time_t timestamp;
   time(&timestamp);

   // separo la fecha actual en atributos
   struct tm* fechaActual = localtime(&timestamp);
   dia = fechaActual->tm_mday;
   mes = fechaActual->tm_mon+1;
   anio = fechaActual->tm_year+1900;
}
```

Veamos cómo invocar a `getDate` para obtener la fecha del sistema.

```
// programa principal,invoca a getDate
int main()
{
   int d,m,a;

   // obtengo la fecha actual invocando a getDate
   getDate(d,m,a);

   cout<< "Dia: " << d <<endl;
   cout<< "Mes: " << m <<endl;
   cout<< "Anio: " << a <<endl;

   return 0;
}
```

1.3.1.6. Función today

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar la función `today` que retorna la fecha actual expresada como un entero de ocho dígitos con formato *aaaammdd*.

```
int today();
```

1.3.2. Problemas

1.3.2.1. Plan de telefonía celular

Dificultad: básica, *Requerido*: Indispensable.

Desarrollar la función `toMin` (pasar a minutos) que recibe un número entero de cuatro dígitos con formato de *hhhmm*, que corresponde a una cantidad de tiempo expresado en horas y minutos, y retorna ese mismo valor expresado en minutos.

```
int toMin(int t);
```

Por ejemplo: si `t` fuese: 25048, que indica 250 horas y 48 minutos, la función debe retornar: 15048.

Luego, desarrollar la función `excedente`, que recibe el precio de un abono telefónico, la cantidad de minutos libres incluidos en dicho abono, el cargo por cada minuto excedente y la cantidad de minutos utilizados por el abonado; y retorna la cantidad de minutos excedidos y el importe que debe pagar el abonado.

El prototipo de la función debe ser el siguiente:

```
void excedente(double precio
             , int minLibres
             , double valorMinutoExcedente
             , int minutosUtilizados
             , int &minutosExcedidos
             , double &importeAAbonar);
```

Utilizando lo anterior, junto con los datos y la operatoria que se describe más abajo, emitir el siguiente listado:

Turno: mañana

Abonado	*Dirección*	*Min. Libres*	*Min Exced.*	*Total a abonar*
xxxxxxxxxxx	xxxxxxxxxxxxxx	999	999	9999.99
xxxxxxxxxxx	xxxxxxxxxxxxxx	999	999	9999.99
xxxxxxxxxxx	xxxxxxxxxxxxxx	999	999	9999.99
:	:	:	:	:

Turno: Tarde

Abonado	*Dirección*	*Min. Libres*	*Min Exced.*	*Total a abonar*
xxxxxxxxxxx	xxxxxxxxxxxxxx	999	999	9999.99

```
xxxxxxxxxxx   xxxxxxxxxxxxxx   999   999   9999.99
xxxxxxxxxxx   xxxxxxxxxxxxxx   999   999   9999.99
    :             :            :     :       :
```

Todos los fines de mes, una empresa de telefonía celular emite las facturas por los consumos de sus abonados. Esta tarea se realiza en tres turnos de trabajo: mañana, tarde y noche.

Para ello, por cada número de celular, se ingresa la siguiente información:

1. Número de celular (`int`).
2. Nombre del abonado (`string`).
3. Dirección del abonado (`string`).
4. Tiempo de uso (*hhhmm*, `int`).
5. Tipo de abono (A, B, C, D o E, `char`).

Dependiendo del tipo de abono, el usuario tiene cierta cantidad de minutos libres por los que no paga ningún cargo extra; pero por cada minuto excedente, deberá abonar una suma extra, de acuerdo con los valores de la siguiente tabla:

	PLANES				
	A	B	C	D	E
Precio	1500	1000	700	500	350
Minutos libres	1000	600	400	300	100
Cargo por minuto excedente	1	3	5	7	10

1.4. Lección 3

1.3.1. Ejercicios

1.4.1.1. TAD Fracción

Dificultad: básica, *Requerido*: Indispensable.

Crear un TAD para encapsular la lógica de los números fraccionarios.

1. Analizar qué atributos debe tener su estructura.

2. Analizar qué funcionalidad debe proveer.
3. Desarrollar las funciones y la estructura del TAD.
4. Probar el desarrollo anterior.

1.4.1.2. TAD Fecha

Crear un TAD para encapsular la lógica de una fecha.

1. Analizar qué atributos debe tener su estructura.
2. Analizar qué funcionalidad debe proveer.
3. Desarrollar las funciones y la estructura del TAD.
4. Probar el desarrollo anterior.

1.4.1.3. TAD Hora

Crear un TAD para encapsular la lógica de una hora.

1. Analizar qué atributos debe tener su estructura.
2. Analizar qué funcionalidad debe proveer.
3. Desarrollar las funciones y la estructura del TAD.
4. Probar el desarrollo anterior.

1.4.1.4. TAD Timer

Dificultad: básica, *Requerido*: Indispensable.

Implementar el TAD `Timer` cuya funcionalidad permite medir el tiempo transcurrido entre dos momentos.

```
// tipo entero muy grande
typedef unsigned long long Long;

struct Timer
{
   // programar aqui...
};

// funcion de inicializacion
Timer timer();

// instante inicial (i)
void timerStart(Timer& t){ // programar aqui }
```

```
// instante final (f)
void timerStop(Timer& t){ // programar aqui }

// retorna el tiempo transcurrido entre f e i,
// expresado en milisegundos
Long timerElapsedTime(Timer t){ // programar aqui }
```

Hora actual incluyendo milisegundos

La siguiente función retorna la fecha actual expresada como la cantidad de millsegundos transcurridos a partir del 1 de enero de 1970.

Requiere: `#include <sys/time.h>`.

```
Long timeInMillis()
{
   struct timeval i;
   gettimeofday(&i, NULL);
   return (i.tv_sec*1000000+i.tv_usec)/1000;
}
```

1.4.1.5. Medición de tiempos

Dificultad: básica, *Requerido*: Indispensable.

Utilizar el TAD `Timer` para medir cuánto demora la computadora en resolver la función `f`, para los siguientes valores de *n*: 100, 1000, 10000, 100000.

```
void f(int n)
{
   for(int i=0;i<n;i++)
      for(int j=0;j<n;j++);
}
```

1.5. Lección 4

1.5.1. Ejercicios

Resolver los siguientes ejercicios usando la API de tratamiento de cadenas de caracteres que desarrollamos durante el capítulo 2.

1.5.1.1. Valor numérico asociado a una cadena

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresa por teclado una cadena de caracteres, sin espacios en blanco y totalmente en mayúscula. Considerando que cada uno de sus caracteres tiene un valor numérico según la siguiente lista: A=1, B=2, C=3, D=4, ...M=13, N=14, Ñ=15, O=16, ... así hasta llegar a Z=27, se debe obtener la suma de los valores asignados a cada uno de los caracteres de la cadena ingresada. Si la suma obtenida tiene más de un dígito, habrá que sumar sus dígitos. Así sucesivamente hasta obtener un valor numérico de un único dígito.

Por ejemplo: Σ OCTAVIANO = 104. Como este valor tiene más de un dígito sumamos sus dígitos: 1 + 0 + 4 = 5. Dado que 5 es un número de un único dígito, llegamos al resultado final. De otro modo, hubiéramos tenido que sumar sus dígitos una y otra vez hasta obtener un valor de un solo dígito.

NOTA: La cadena ingresada no tendrá caracteres especiales como Ñ o tiles.

1.5.1.2. Primeros n números que tienen m dígitos d

Dificultad: moderada, *Requerido*: Indispensable.

Se ingresan por teclado 3 valores enteros: *n*, *m* y *d*. Se pide mostrar por consola los primeros *n* números naturales que tienen, al menos, *m* dígitos *d*. Por ejemplo, si *n*=5, *m*=3 y *d*=4, la salida del programa debería ser: 444, 1444, 2444, 3444, 4044.

1.5.1.3. Anagrama

Dificultad: moderada, *Requerido*: Indispensable.

Desarrollar y probar adecuadamente la función `esAnagrama` (según el siguiente prototipo), que retorna `true` o `false` según resulte que las dos cadenas de recibe sean o no anagramas entre sí.

```
bool esAnagrama(string a,string b);
```

1.5.1.4. Interpretar datos contenidos en una cadena

Dificultad: moderada, *Requerido*: recomendable.

Se ingresa por teclado una cadena que contiene el nombre, la fecha de nacimiento y la nacionalidad de varias personas. Por ejemplo:

"Pedro,2-oct-1970,Argentino|Juan,9-dic-1985,Chileno|Pablo,14-ene-1992,Argentino"

Como vemos, la cadena contiene los datos de Pedro, Juan y Pablo separados mediante el carácter '|' (carácter pipe). A su vez, los datos de cada uno de ellos se separan entre sí mediante el carácter ',' (carácter coma). Por su parte, los atributos de las fechas de nacimiento están separados entre sí por el carácter '-' (guion).

Se pide desarrollar un programa que muestre por pantalla los datos de cada persona. Por ejemplo, según el ejemplo anterior, el programa debería mostrar:

```
Cantidad de personas: 3
---
Nombre: Pedro
Fecha de nacimiento: Dia 2, Mes oct, Anio 1970
Nacionalidad: Argentino
---
Nombre: Juan
   :
```

1.5.1.5. TAD BigInt

Dificultad: intermedia, *Requerido*: recomendable.

Desarrollar y probar el TAD `BigInt` de acuerdo con la estructura y especificaciones de las funciones que se describen a continuación.

Nombre del TAD: `BigInt`.

Descripción: Representa números enteros muy grandes, sin restricciones de tamaño ni cantidad de dígitos .

```
struct BigInt
{
```

```
   string s;
};
```

Prototipo: `BigInt bigInt(string n);`

Descripción: Crea (o instancia) un `BigInt`.

Parámetro: `string n` - Cadena que contiene la representación del número.

Retorna: `BigInt` - Una instancia de `BigInt` lista para trabajar con el número `n`.

Prototipo: `BigInt bigIntSumar(BigInt a,BigInt b);`

Descripción: Retorna una instancia `BigInt` que contiene la suma de `a` y `b`.

Parámetros:

- `BigInt a` - Valor a sumar.
- `BigInt b` - Valor a sumar.

Retorna: `BigInt` - Una instancia de `BigInt` que contiene la suma de `a` y `b`.

Prototipo: `BigInt bigIntRestar(BigInt a,BigInt b);`

Descripción: Retorna una instancia `BigInt` que contiene la resta de `a` y `b`.

Parámetros:

- `BigInt a` - Minuendo.
- `BigInt b` - Sustraendo.

Retorna: `BigInt` - Una instancia de `BigInt` que contiene la resta de `a` menos `b`.

1.5.1.6. TAD Matriz

Dificultad: intermedia, *Requerido*: recomendable.

Desarrollar y probar el TAD `Matriz` de acuerdo con la estructura y especificación de las funciones que se describen a continuación.

Nombre del TAD: `Matriz`.

Descripción: Representa una matriz numérica de *n* filas y *m* columnas.

Restricción: Cada celda de la matriz puede contener un número de un único dígito.

```
struct Matriz
{
   string s;
   int filas;
   int columnas;
}
```

Prototipo: `Matriz matriz(int n,int m);`

Descripción: Retorna una instancia `Matriz`.

Parámetros:

- `int n` - Cantidad de filas de la matriz.
- `int m` - Cantidad de columnas de la matriz.

Retorna: `Matriz` - Una instancia de `Matriz` preparada para contener una matriz de `n` filas y `m` columnas.

Prototipo: `Matriz matriz(String x,int n,int m);`

Descripción: Retorna una instancia `Matriz` inicializada con el contenido de `x`.

Parámetros:

- `String x` - Contenido de la matriz que se está instanciando.
- `int n` - Cantidad de filas.
- `int m` - Cantidad de columnas.

Retorna: `Matriz` - Una instancia de `Matriz` de `n` filas y `m` columnas, inicializada con el contenido de la cadena `x`.

Ejemplo: Si `x="123456789012"`, `n=3` y `m=4`, la matriz representada por este conjunto de parámetros que estos parámetros están representando es la siguiente:

```
1   2   3   4
5   6   7   8
9   0   1   2
```

Prototipo: `int matrizGet(Matriz m,int f,int c);`

Descripción: Retorna el valor (número de un dígito) que la matriz `m` contiene en la celda determinada por la fila `f` y la columna `c`.

Parámetros:

- `Matriz m` - Matriz.
- `int f` - Fila.
- `int c` - Columna.

Retorna: `int` - El valor (número de un dígito) que la matriz `m` contiene en la celda determinada por la fila `f` y la columna `c`.

Prototipo: `void matrizSet(Matriz& m,int f,int c,int v);`

Descripción: Asigna el valor `v` (número de un dígito) en la celda determinada por la fila `f` y la columna `c` de la matriz `m`.

Parámetros:

- `Matriz m` - Matriz.
- `int f` - Fila.
- `int c` - Columna.
- `int v` - Valor numérico (de un solo dígito) que se asignará.

Retorna: `void`.

Prototipo: `Matriz matrizSumar(Matriz a,Matriz b);`

Descripción: Retorna la matriz que resulta de sumar `a+b`.

Parámetros:

- `Matriz a` - Matriz a sumar.
- `Matriz b` - Matriz a sumar.

Retorna: `Matriz` - La matriz resultante de la suma de las matrices `a` y `b`.

Prototipo: `Matriz matrizRestar(Matriz a,Matriz b);`

Descripción: Retorna la matriz que resulta de restar `a-b`.

Parámetros:

- `Matriz a` - Minuendo.
- `Matriz b` - Sustraendo.

Retorna: `Matriz` - La matriz que resulta de restar `a-b`.

NOTA: Esta implementación del TAD `Matriz` tiene la restricción de que cada elemento de la matriz debe ser un valor numérico de un solo dígito, sin signo. Por esto, aceptaremos que las operaciones de suma y resta no arrojarán resultados que excedan estas limitaciones.

Por ejemplo: si sumamos las dos matrices que vemos a continuación, la matriz resultante cumple con las restricciones del TAD.

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

Figura 2.2. Ejemplo de matrices que podrían aplicar para la operación matrizSumar

1.6. Lección 5

1.6.1. Ejercicios

1.6.1.1. Anagramas

Dificultad: intermedia, *Requerido*: indispensable.

Desarrollar un programa que, a partir de un conjunto de palabras que el usuario ingresará por consola, muestre cada una de las palabras, seguida de las otras palabras del conjunto que son anagramas.

Por ejemplo, si el usuario ingresa las siguientes palabras:

mora, roma, operas, espora, separo, amar, rama

La salida debería ser:

```
mora: roma
operas: espora, separo
amar: rama
```

1.6.1.2. Mostrar en orden

Dificultad: intermedia, *Requerido*: indispensable.

El usuario ingresará un conjunto de palabras a través de la consola, todas en mayúsculas, finalizando el ingreso de datos al ingresar la palabra "FIN". Se pide:

- Mostrar todas las palabras ingresadas, en orden alfabético ascendente.
- Mostrar todas las palabras ingresadas, en orden alfabético descendente.
- Mostrar todas las palabras ingresadas, en orden ascendente según sus longitudes.
- Mostrar todas las palabras ingresadas, en orden ascendente según la cantidad de vocales que las componen.

NOTA: considerar cada uno de estos ítems como un ejercicio en sí mismo.

1.7. Lección 7

1.7.1. Ejercicios

1.7.1.1. Palabras repetidas

Dificultad: intermedia, *Requerido*: indispensable.

Se ingresa por teclado un conjunto de palabras (finalizando el ingreso con la palabra "FIN"), se pide mostrar todas las palabras ingresadas, sin repetición.

Por ejemplo: si se ingresa: {Hola, Casa, Árbol, Casa, Hola, Trapo}, la salida del programa debería ser: {Hola, Casa, Árbol, Trapo}.

1.7.1.2. Cantidad de veces que aparece cada palabra

Dificultad: intermedia, *Requerido*: indispensable.

Se ingresa por teclado un conjunto de palabras (finalizando el ingreso con la palabra "FIN"), se pide mostrar todas las palabras ingresadas, sin repetición, indicando por cada una cuántas veces se ingresó.

Por ejemplo: si se ingresa: {Hola, Casa, Arbol, Casa, Hola, Trapo, Casa}, la salida del programa debería ser: [Hola,2], [Casa,3], [Arbol,1], [Trapo,1].

1.7.1.3. Cantidad de veces que aparece cada palabra, ordenado

Ídem problema anterior, pero la salida debe aparecer ordenada decrecientemente según la cantidad de repeticiones de cada palabra.

1.8. Lección 10

1.8.1. Ejercicios

1.8.1.1. Mayor valor, y en qué posiciones aparece

Dificultad: intermedia, *Requerido*: indispensable.

Se dispone de un archivo de caracteres numéricos; cada carácter representa un número positivo de 1 dígito. Se pide determinar cuál es el mayor valor y en qué posiciones aparece. Por ejemplo: si el archivo fuera: 143214324231421, el mayor valor es 4 y aparece en las posiciones: 1, 5, 8 y 12.

1.8.1.2. Mayor valor, y en qué posiciones aparece (versión 2)

Dificultad: intermedia, *Requerido*: indispensable.

Ídem anterior pero los caracteres estarán separados por ‘,’ (coma), por lo cual los números podrán tener más de 1 dígito. Por ejemplo: 23,45,1,4,45,15,7,45,8,12.

1.8.1.3. Nombres

Dificultad: intermedia, *Requerido*: indispensable.

Se tiene un archivo de caracteres que contiene nombres separados por ‘\n’.

Por ejemplo:

```
Pedro
Analia
Pablo
Teresa
Susana
Juan
```

Se pide informar los nombres cuya longitud es máxima. Según el ejemplo anterior la salida será: Analía, Teresa y Susana. Pues, longitud máxima registrada es de 6 caracteres y todos estos nombres tienen dicha longitud.

1.8.1.4. Nombres (versión 2)

Dificultad: intermedia, *Requerido*: indispensable.

Se tiene un archivo de caracteres que contiene nombres separados por '\n', ordenados ascendentemente alfabéticamente. Pero puede haber nombres fuera mal ubicados. Por ejemplo:

```
Alberto
Juan
Pablo
Horacio
Pedro
Sandra
Marcelo
Samuel
```

Según el ejemplo, Horacio y Marcelo están fuera de lugar. Se pide generar, con estos nombres, un nuevo archivo llamado `out.txt`.

1.8.1.5. Unificar archivos

Dificultad: intermedia, *Requerido*: indispensable.

Se dispone de los archivos de caracteres `a.txt` y `b.txt`, ambos contienen nombres de personas separados por '\n' y ordenados alfabéticamente, sin repetición; aunque sí, un mismo nombre, podría aparecer en ambos archivos.

Se pide generar un tercer archivo `c.txt` con los nombres de `a.txt` y `b.txt` intercalados, ordenados alfabéticamente y sin repetición en caso de que un mismo nombre aparezca en ambos archivos.

Imprimir, al final del programa, la lista de nombres que aparecieron en ambos archivos; ordenada de mayor a menor alfabéticamente.

A continuación veremos un ejemplo de los datos que los archivos `a.txt` y `b.txt` que podrían llegar a contener.

a.txt	b.txt
Alberto	Analia
Ana	Armando
Angel	Carlos
Carlos	Maria
Marcelo	Mauro
Rodrigo	Rodrigo
Rogelio	Sebastian
Sebastian	Tamara
	Tatiana

1.8.1.6. Unificar archivos (versión 2)

Dificultad: intermedia, *Requerido*: indispensable.

Ídem anterior, aceptando que los nombres pueden estar repetidos. Por ejemplo:

a.txt	b.txt
Alberto	Analia
Ana	Armando
Angel	Carlos
Carlos	Maria
Carlos	Maria
Carlos	Mauro
Marcelo	Rodrigo
Rodrigo	Rodrigo
Rodrigo	Rodrigo
Rogelio	Sebastian
Sebastian	Tamara
	Tatiana

Se pide generar un tercer archivo `c.txt` con los nombres de `a.txt` y `b.txt` intercalados, ordenados alfabéticamente y sin repetición.

Al final del proceso se debe mostrar una lista con los nombres que aparecieron varias veces (en uno o ambos archivos) indicando, por cada uno, cuántas veces se repitió. La lista debe aparecer ordenada descendentemente según dicha cantidad. A igual cantidad de repeticiones, el orden será: alfabéticamente ascendente.

Según el ejemplo, la lista será:

```
Rodrigo, 4
Carlos, 3
Maria, 1
Sebastian, 1
```

1.9. Lección 11

1.9.1. Ejercicios

1.9.1.1. Compactador a medio byte

Dificultad: intermedia, *Requerido*: indispensable.

Construir un compactador a ½ byte para archivos de caracteres que sólo contienen caracteres numéricos. Para esto, se deben desarrollar dos programas: `compactar` y `descompactar`. Ambos programas reciben en línea de comandos los nombres de los archivos de entrada y salida con los cuales van a trabajar.

NOTA: Considerar que para representar un valor numérico comprendido entre 0 (cero) y 9 (nueve) alcanza con 4 bit.

Ejemplo:

```
C:\>compactar arch.txt arch.out
C:\>descompactar arch.out arch.txt
```

1.9.1.2. TAD BitWriter, BitReader

Dificultad: intermedia, *Requerido*: indispensable.

Desarrollar los TAD `BitWriter` y `BitReader` cuyo objetivo es proveer la funcionalidad necesaria grabar y escribir bit archivos, bit por bit. La API de cada uno de estos TAD se describen a continuación:

TAD BitWriter

Estructura:

```
struct BitWriter
{
   // ...
};
```

Prototipo: `BitWriter bitWriter(FILE* f);`

Descripción: Crea e inicializa una variable tipo `BitWriter`.

Parámetros: `FILE* f` – Archivo donde se grabarán los bit.

Retorna: `BitWriter`.

Ejemplo de uso:

```
FILE* f = fopen("arch.bin","w+b");
BitWriter bw = bitWriter(f);
```

Prototipo: `void bitWriterWrite(BitWriter bw,int bit);`

Descripción: Graba un bit en el archivo.

Parámetros:

- `BitWriter br` – Variable del TAD.
- `int bit` – 1 o 0 que se grabará en el archivo.

Retorna: `void`.

Ejemplo de uso:

```
FILE* f = fopen("arch.bin","w+b");
BitWriter bw = bitWriter(f);
bitWriterWrite(bw,0);
bitWriterWrite(bw,1);
bitWriterWrite(bw,0);
bitWriterWrite(bw,0);
bitWriterWrite(bw,0);
bitWriterWrite(bw,0);
bitWriterWrite(bw,0);
bitWriterWrite(bw,1);
fclose(f);
```

Prototipo: `void bitWriterFlush(BitWriter bw);`

Descripción: Indica que ya no se grabarán más bit. En caso de que la cantidad de bit que grabamos no llegue a ser múltiplo de 8, completa con ceros a la derecha y graba.

Parámetros: `BitWriter br` – Variable del TAD.

Retorna: `void`.

Ejemplo de uso:

```
FILE* f = fopen("arch.bin","w+b");
BitWriter bw = bitWriter(f);
bitWriterWrite(bw,0);
bitWriterWrite(bw,1);
bitWriterFlush(bw);
fclose(f);
```

TAD BitReader

Estructura:

```
struct BitReader
{
   // ...
};
```

Prototipo: `BitReader bitReader(FILE* f);`

Descripción: Crea e inicializa una variable tipo `BitReader`.

Parámetro: `FILE* f` – Archivo desde el cual se leerán los bit.

Retorna: `BitReader`.

Ejemplo de uso:

```
FILE* f = fopen("arch.bin","r+b");
BitReader br = bitReader(f);
```

Prototipo: `int bitReaderRead(BitReader br);`

Descripción: Lee un bit desde el archivo.

Parámetro: `BitReader bw` – Variable del TAD.

Retorna: `int` – Bit (1 o 0) que leído desde el archivo.

Ejemplo de uso:

```
FILE* f = fopen("arch.bin","r+b");
BitReader br = bitReader(f);

int bit = bitReaderRead(br);
while( !feof(f) )
{
   cout << bit << endl;
   bit = bitReaderRead(br);
}

fclose(f);
```

1.9.1.3. TAD File

Dificultad: alta, *Requerido*: recomendable.

Implementar el TAD `File` que permite operar con archivos de registros de longitud fija, e incluye la funcionalidad de *eliminar* registros.

NOTA: Se sugiere usar un archivo temporal para guardar información sobre aquellos registros que, mediante la función `fileDelete`, fueron eliminados.

Al existir la posibilidad de que ciertos registros del archivo hayan sido eliminados, todas las funciones deben trabajar en consecuencia. Por ejemplo: `fileRead`, que lee un registro y deja el indicador de posición apuntando al siguiente, debe considerar que dicho registro no haya sido eliminado. En tal situación el indicador de posición deberá quedar apuntando al próximo registro no eliminado. El mismo razona-miento es aplicable a `fileWrite`.

La función `fileSize` debe retornar la cantidad de registros (no eliminados) que contiene el archivo, y todas las otras funciones deben actuar en consecuencia.

```
template<typename T>
struct File
{
   // programar aqui...
};
```

```
template<typename T>
File<T> file(string fileName,string modoApertura)
{
   // programar aqui...
}

// lee y retorna el registro apuntado por el indicador
// de posicion del archivo, avanzando dicho indicador
// hacia el siguiente registro NO eliminado
template<typename T>
T fileRead(File<T>& f)
{
   // programar aqui...
}

// escribe el valor t en el registro apuntado por el
// indicador de posicion del archivo, avanzando dicho
// indicador hacia el siguiente registro NO eliminado
template<typename T>
void fileWrite(File<T>& f,T t)
{
   // programar aqui...
}

// retorna la cantidad de registros NO eliminados
// que tiene el archivo
template<typename T>
int fileSize(File<T> f)
{
   // programar aqui...
}

// hace que el indicador de posicion del archivo
// pase a apuntar al registro numero p.
// Por ejemplo: si p es 3, pero el registro 1
// esta eliminado, entonces fisicamente el tercer
// registro del archivo sera el que ocupa la pocision 4
template<typename T>
void fileSeek(File<T>& f,int p)
{
   // programar aqui...
}
```

```
// retorna el valor del indicador de posicion del archivo.
// Por ejemplo: si el registro actual es 4 pero el registro
// 1 esta eliminado, entonces el registro actual es 3
template<typename T>
int filePos(File<T>& f)
{
   // programar aqui...
}

// borra el registro apuntado por el indicador
// de posicion del archivo
template<typename T>
void fileDelete(File<T>& f)
{
   // programar aqui...
}

// regenera el archivo quitando fisicamente aquellos
// registros que fueron elminados con fileDelete
template<typename T>
void fileRepack(File<T>& f)
{
   // programar aqui...
}

// cierra el archivo
template<typename T>
void fileClose(File<T>& f)
{
   // programar aqui...
}
```

1.9.2. Trabajo práctico - Archivo de registros de longitud variable

Dificultad: alta, *Requerido*: indispensable.

1.9.2.1. Introducción

En un archivo de registros de longitud variable cada registro ocupará más o menos espacio en disco dependiendo de los datos que tenga almacenados en sus campos.

Se trata de establecer un mecanismo que permita determinar donde comienza y finaliza cada registro; y dentro de éste, donde comienza y finaliza cada uno de

sus campos o atributos. Este mecanismo es la estructura del archivo; no en términos de `struct` (estructuras estáticas), sino en términos de su composición interna.

1.9.2.2. Consigna del ejercicio

Dado un archivo cuya estructura se describe más abajo, se pide desarrollar un programa para mostrar su contenido por pantalla. Además, se debe actualizar la fecha de último acceso con la fecha del sistema (ver estructura).

Luego, habrá que desarrollar otro programa que interactúe con el usuario para generar un nuevo archivo, con la misma estructura que el anterior, y los datos que ingrese el usuario.

Archivo de registros de longitud variable

Ambos programas recibirán los nombres de archivo por línea de comandos. Por ejemplo, si el archivo fuese `AGENDA.dat`, para ver su contenido haremos lo siguiente:

```
C:\>mostrar AGENDA.dat
```

```
----[CONTENIDO DEL ARCHIVO]--------------------
Nro. de serie: 54321
Full filename: C:/algoritmos/DEMO.dat
Fecha de ultimo acceso: 2020/5/23
Cantidad de campos configurados: 4
Campo [codigo: 1, descripcion: Nombre]
Campo [codigo: 2, descripcion: Telefono]
Campo [codigo: 3, descripcion: Direccion]
Campo [codigo: 4, descripcion: EMail]
Cantidad de Registros (contactos): 5
-----------------------
Nombre: Bill Gates
Telefono: 4532-2411
Direccion: San Martin 126
EMail: bill@microsoft.com
-----------------------
Nombre: Jeff Bezos
Telefono: 5534-2331
EMail: jeff@amazon.com
-----------------------
Nombre: Larry Ellison
EMail: larry@oracle.com
-----------------------
Nombre: Mark Suckerberg
Telefono: 6642-7732
```

```
Direccion: Pje. Mar Del Plata 4645
EMail: mark@facebook.com
----------------------
Nombre: Marcos Galperin
Telefono: 4212-6623
Direccion: Av. Santa Fe 4112
EMail: marcos@mercadolibre.com.ar
----[FIN CONTENIDO DEL ARCHIVO]-----------------
```

La estructura del archivo permite configurar qué campos o atributos de nuestros contactos queremos guardar. Aun así, no estaremos obligados a registrar todos los atributos para todos los contactos.

Si de un contacto no tenemos información sobre alguno de sus atributos, podemos omitirlo; y, en consecuencia, su registro ocupará menos espacio en el disco que otro contacto para el cual sí hayamos completado toda su información.

1.9.2.3. Estructura del archivo

La siguiente tabla describe, de arriba hacia abajo, la estructura del archivo. Los tipos de dato `Integer`, `String`, `Date`, `RegType`, `RegData`, `Character` y `Byte` (estos últimos aparecerán más abajo), son semánticos; y su finalidad es netamente documental y se detallarán a continuación.

Bytes	Tipo de dato	Descripción
2	`Integer`	Número de serie.
n	`String`	Nombre completo del archivo.
2	`Date`	Fecha del último acceso.
2	`Integer`	Cantidad de campos configurados
*m** (1)	`RegType`	Descripción de los campos configurados
2	`Integer`	Cantidad de registros.
*t** (2)	`RegData`	Contenido de los registros.

Tabla 3.5. Estructura del archivo de registros de longitud variable.

(1) Se repite según la cantidad de campos configurados.

(2) Se repite según la cantidad de registros.

1.9.2.4. Tipos de dato

Integer

Representa un valor numérico entero, sin bit de signo, almacenado en 2 bytes.

String

Representa una cadena de caracteres. Su estructura interna (es decir: cómo está almacenada) dependerá de su longitud.

Llamaremos *L* a la longitud de la cadena de caracteres, entonces:

Si 0<=*L*<255:

Bytes	Tipo de dato	Descripción
1	`Byte`	Longitud de la cadena.
*n** (1)	`Character`	Caracteres que componen la cadena.

Tabla 3.6. Estructura de una cadena "corta".

(1) Se repite según la longitud de la cadena.

Si *L*>=255:

Bytes	Tipo de dato	Descripción
1	`Byte`	Valor numérico 255 o 0xFF.
2	`Integer`	Longitud de la cadena.
*n** (1)	`Character`	Caracteres que componen la cadena.

Tabla 3.7. Estructura de una cadena "larga".

(1) Se repite según la longitud de la cadena.

Byte, Character

Representan un valor numérico entero, sin bit de signo, almacenado en 1 byte. En particular, `Character` indica que dicho valor tendrá representación ASCII.

RegType

Con este tipo de datos se representan los campos o atributos de los registros. Su estructura se compone de la siguiente manera.

Bytes	Tipo de dato	Descripción
1	`Byte`	Código de campo.
n	`String`	Descripción.

Tabla 3.8. Estructura del tipo RegType.

RegData

Con este tipo de dato se representan los registros de los contactos almacenados en el archivo. Su estructura se compone del siguiente modo.

Bytes	Tipo de dato	Descripción
1	`Byte`	Cantidad de campos completados.
*(1+n)** (1)	`Byte`	Código de campo.
	`String`	Valor.

Tabla 3.9. Estructura del tipo RegData.

(1) Se repite según la cantidad de campos completados.

Date

Se trata de una fecha representada como un número entero, sin bit de signo y en 2 bytes de longitud. Sus atributos (día, mes y año) se ubican según la estructura que de describe a continuación:

1er. byte								2do. byte							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Año								Mes				Día			
Hasta: 2^7-1 (127)								Hasta: 2^4-1 (15)				Hasta: 2^5-1 (31)			

Tabla 3.10. Estructura del tipo Date.

Los primeros 5 bit, comenzando desde la derecha, representan el día. Los siguientes 4 bit (hacia la izquierda5 el día.

Los primeros 7 bit, contando desde la izquierda, representan el valor del año. Como dicho valor está acotado entre 0 y 127 lo interpretaremos del siguiente modo:

Sea *a* el valor de un año contenido en una fecha:

- Si *a*<100; representa al año 2000+a.
- Si *a*>=100; representa al año 1999-a+100.

Por ejemplo:

Si *a* = 125 entonces representa al año: 1999-a+100 = 1974.

Si *a* = 4 entonces representa al año 2000+a = 2004.

1.10. Lección 13

1.10.1. Ejercicios

1.10.1.1. Anagramas

Dificultad: intermedia, *Requerido*: indispensable.

Desarrollar un programa que, luego de recibir un conjunto de palabras por línea de comandos, indique: para cada una de las palabras ingresadas, cuáles de las otras palabras del conjunto son anagramas.

Por ejemplo, si se ingresan en línea de comandos las siguientes palabras:

mora, roma, operas, espora, separo, amar, rama

La salida debería ser:

```
mora: roma
operas: espora, separo
amar: rama
```

NOTA: Este ejercicio fue planteado en la práctica del capítulo 2. Ahora se propone resolverlo usando las nuevas estructuras de datos y TAD desarrollados durante el presente capítulo.