Finales de Taller de Programación I [7542-9508] - Veiga

Final 26/2/2019

- Escriba un programa que reciba por línea de comandos un Puerto y una IP. El programa debe aceptar una única conexión y recibir paquetes (cada paquete inicia con `[` y finaliza con `]`). Para cada paquete recibido debe imprimir el checksum (sumatoria de bytes del paquete) del mismo, excepto que el paquete esté vacío (`[]`), en cuyo caso debe finalizar.

Básicamente se pide un servidor. Pasos a seguir:

- Bind
- Listen
- Accept
- Recibir
- Liberar

```
#define CONEX_EN_COLA 1
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    char* port = argv[1];
    char* ip = argv[2];

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;
    struct addrinfo *res = NULL;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(ip, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                ptr->ai_socktype,
                ptr->ai_protocol);
        if (skt != -1) break;
    }

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    int peerskt = accept(skt, NULL, NULL);

    char paquete[MAX_PAQUETE + 1];
    char buffer[MAX_PAQUETE];
    size_t pos_paquete = 0;
```

```
    size_t leido = 0;
    bool salir = false;

    while (! salir) {
        //4) Recibir
        leido = recv(peerskt, buffer, MAX_PAQUETE, 0);
        for (size_t i = 0; i < leido; i++) {
            paquete[pos_paquete] = buffer[i];
            if (buffer[i] == ']') {
                paquete[pos_paquete] = '\0';
                if (strcmp(paquete, "[]") == 0) {
                    salir = true;
                    break;
                } else {
                    int checksum = 0;
                    for (int i = 1; i < strlen(paquete); i++) {
                        checksum += paquete[i];
                    }
                    printf("%d\n", checksum);
                    pos_paquete = 0;
                }
            }
            pos_paquete++;
        }
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    close(peerskt);
    freeaddrinfo(ptr);
    return 0;
}
```

- Explique qué son los métodos virtuales y para qué sirven. De un breve ejemplo donde su uso sea imprescindible.

La distinción entre métodos virtuales y no virtuales sirve para resolver el problema de la ambigüedad cuando una clase y una subclase tienen definida la misma función. Si la función en cuestión es designada "virtual", se llamará a la función de la clase derivada (si existe). Si no es virtual se llamará a la función de la clase base. Sería imprescindible en caso de que se desee llamar al método de la clase derivada en lugar del método de la clase base. Agregar ejemplo

- Describa con exactitud cada uno de los siguientes:

a) `static int A = 7;`

Se define una variable A que es un entero con signo de alcance local con el valor 7. Reside en el data segment no debe liberarse y el tamaño es de 4/2 bytes dependiendo de si se trata de una arquitectura de 32/64 bits o de 16 bits.

b) `extern char* (*B) [7];`

Se declara la variable "B" que apunta a un array de 7 posiciones de punteros a carácteres con signo definido en un archivo externo.

c) `float **C;`

Se declara una variable "C" que es un puntero a un puntero a un número de punto flotante.

- Declare la clase *IPv4* para almacenar una dirección IP. Incluya constructor default, constructor move, constructor de copia y los siguientes operadores: operator<<, operator==, operator= y operator+.

```
class IPv4 {
private:
    std::string ip;
public:
    IPv4(); //constructor default
    IPv4(IPv4&&); //constructor move
    IPv4(const IPv4&); //constructor de copia
    friend std::ostream& operator<<(std::ostream& out, const IPv4&);
    bool operator==(const IPv4&) const;
    IPv4& operator=(const IPv4&);
    IPv4 operator+(const IPv4&) const;
};
```

- ¿Cómo se logra que 2 threads accedan (lectura/escritura) a un mismo recurso compartido sin que se generen problemas de consistencia? Ejemplifique

Si el recurso compartido es inmutable o solamente se accede a tal para operaciones de lectura, no existe la posibilidad de que se generen problemas de consistencia o race conditions. Para evitar la race condition se debe lograr que los hilos se coordinen entre sí para evitar que acedan al objeto compartido a la vez. Existen varias estrategias de sincronización pero la más usual es usar el objeto `std::mutex` que permite forzar la ejecución de un código de forma exclusiva por un hilo a la vez. Una vez accedido al recurso el hilo debe liberar el mutex sino el recurso queda inaccesible para el resto de los hilos.

```
class ProtectedCounter {
    int counter;
    std::mutex m;
public:
    void inc() {
        m.lock();
        ++counter;
        m.unlock();
    }
};
```

De esta manera si se lanzan varios threads que comparten esta variable counter el resultado será determinístico y no dependerá de las decisiones del scheduler del sistema operativo.

- Describa el concepto de loop de eventos (evento loop) utilizado en programación orientada a eventos, y en particular, en entornos de interfaz gráfica (GUIs)

En la programación orientada a eventos el programa está constantemente esperando que se generen eventos, tales como pueden ser un clic, una pulsación del teclado, un paquete de red, un temporizador, etc y luego los despacha a sus respectivos manejadores. El loop de eventos es el ciclo principal del programa y se compone de dos tareas:

1. Decodificar el siguiente evento, donde se determina quién debe ser notificado
2. Despachar el evento, donde se envían las notificaciones

Pseudocódigo del event loop:

```
while se debe continuar:
    evento := obtener el siguiente evento de la cola de eventos
if evento == salir:
    se debe continuar := false
```

```
else if existe manejador para evento:
    ejecutar manejador
```

Los manejadores son secciones de código que saben cómo responder a la aparición de un evento. Como los va a disparar el event loop, se van a ejecutar de manera secuencial así que no van a tener problemas de concurrencia entre ellos y si uno tarda mucho va a retrasar a todos los que vengan después. En aplicaciones con GUI se deben programar handlers cortos y que den feedback al usuario. En muchos frameworks gráficos, el event loop corre en el hilo principal (GTK te abstrae de programarlo mientras que SDL te obliga a hacerlo).

- ¿Qué valor arroja `sizeof(int)`? Justifique

El valor de `sizeof(int)` depende de la arquitectura de la computadora y del compilador, es decir, de la implementación. En el caso de una computadora de 32 bits o 64 bits un int generalmente ocupa 4 bytes y en una de 16 bits 2 bytes.

- ¿Qué significa que una función es bloqueante? ¿Cómo subsanaría esa limitación en términos de mantener el programa 'vivo'?

Significa que el hilo no avanza en su ejecución en tanto y en cuanto la función no retorne un resultado. Según lo que se necesite hacer se pueden utilizar threads para aprovechar mejor los tiempos.

- Escribir un programa ISO C que procese el archivo de texto cuyo nombre es recibido como parámetro. El procesamiento consiste en leer oraciones y suprimir aquellas que tengan más de 3 palabras. Asuma que el archivo no puede cargarse en memoria, pero una oración sí puede.

```
int leer(FILE* f, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    int out = fgetc(f);
    *seek = ftell(f);
    return out;
}

void escribir(FILE* f, char c, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    fputc(c, f);
    *seek = ftell(f);
}

int main(int argc, char* argv[]) {
    FILE *f;
    int c = 0;
    long int c_final = 0;
    long int seek_lectura = 0;
    long int seek_escritura = 0;

    char buffer[1024];
    memset(buffer, 0, 1024);

    bool inicio = true;
    int pos = 0;

    char* file_name = argv[1];
    f = fopen(file_name, "r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;
        if (inicio) {
```

```
            buffer[pos] = c;
            if (c == '.') inicio = false;
            pos++;
            if (inicio) continue;
        }

        int contador_espacios = 0;
        for (int i = 0; i < strlen(buffer); i++) {
            if (buffer[i] == ' ') contador_espacios++;
        }

        bool maximo_palabras = (contador_espacios >= 3);

        if (! maximo_palabras) {
            for (int i = 0; i < strlen(buffer) && buffer[i] != '\0'; i++) {
                escribir(f, buffer[i], &seek_escritura);
            }
        } else {
            c_final -= pos;
        }

        // Reset:
        inicio = true;
        memset(buffer, 0, 1024);
        pos = 0;
    }
    int contador_espacios = 0;
    for (int i = 0; i < strlen(buffer); i++) {
        if (buffer[i] == ' ') contador_espacios++;
    }

    bool maximo_palabras = (contador_espacios == 3);

    if (! maximo_palabras) {
        for (int i = 0; i < strlen(buffer) && buffer[i] != '\0'; i++) {
            escribir(f, buffer[i], &seek_escritura);
        }
    } else {
        c_final -= pos;
    }
    fclose(f);
    truncate(file_name, sizeof(char) * c_final);
    return 0;
}
```

- Implemente una función C++ denominada *Suprimir* que reciba dos listas de elementos y devuelva una nueva lista con los elementos de la primera que no están en la segunda:

```
std::list<T> Suprimir(std::list<T> a, std::list<T> b);
```

```
template<class T>
std::list<T> Suprimir(std::list<T> a, std::list<T> b) {
    std::list<T> resultado;
    for (auto i = a.begin(); i != a.end(); i++) {
        auto item = std::find(b.begin(), b.end(), *i);
```

```
        if (item == b.end()) {
            resultado.push_back(*i);
        }
    }
    return resultado;
}
```

Final 19/2/2019

- Explique qué es y para qué sirve un constructor de copia en C++:

a) Indique cómo se comporta el sistema si éste no es definido por el desarrollador

C y C++ pasan todos los objetos por copia por default. Si un objeto no tiene implementado un constructor por copia se le asigna un constructor por copia default, excepto que se borre explícitamente el constructor por copia, en cuyo caso el objeto dejaría de ser copiable.

b) Explique al menos una estrategia para evitar que una clase sea copiable

Se puede evitar que una clase sea copiable borrando el constructor por copia y el de asignación. Si en algún momento se trata de copiar algún objeto el compilador dará error. Por ejemplo, si se tiene una clase File se puede evitar que sea copiable de esta forma:

```
File(const File& other) = delete;
File& operator=(const File &other) = delete;
```

Otra forma seria declarar pero no definir ni el constructor por copia ni el operador asignación y hacerlos privados. El intento fallido de copia se detecta en tiempo de compilación y linkeo.

c) Indique qué diferencia existe entre un constructor de copia y uno move

A diferencia de una copia, el constructor por movimiento le roba o mueve los atributos del objeto fuente. Para marcar el cambio de ownership es necesario modificar el objeto fuente (other) para que deje de apuntar a los recursos ahora apropiados, de otro modo se tendrían dos objetos apuntando a un mismo recurso. Luego de que el objeto sea movido debe seguir siendo válido de tal manera que se pueda ejecutar sobre other el operador asignación y el destructor. Si se tuviese por ejemplo una clase `Vector` un constructor por movimiento podría ser el que sigue:

```
class Vector:
    int* data;
    size_t size;

    Vector(Vector&& other) {
        this->data = other.data;
        this->size = other.size;
        other.data = nullptr;
        other.size = 0;
    }
};
```

El constructor por copia inicializa sus atributos con la data que proviene del objeto fuente. Como se puede observar la información se duplica, se realiza una copia bit a bit naive y esto resulta en una peor performance ya que se duplica el espacio utilizado por el objeto y además se emplea un cierto tiempo en copiar el objeto.

```
Vector(const Vector& other) {
    this->data = malloc(sizeof(int)*other.size);
    this->size = other.size;
    memcpy(this->data, other.data, this->size);
}
```

- Escriba una función ISO C llamada *Replicar* que reciba una cadena (*S*), dos índices (*I1* e *I2*) y una cantidad (*Q*). La función debe retornar una copia de S salvo los carácteres que se encuentren entre los índices I1 e I2 que serán duplicados Q veces.

Ej: `replicar("Hola", 1, 2, 3)` retorna `"Hololola"`

```
char* replicar(const char* S, size_t I1, size_t I2, size_t Q) {
    if (I1 >= strlen(S) || I2 >= strlen(S) || Q == 0
    || I1 >= I2) {
        fprintf(stderr, "Error en parámetros");
    }
    size_t nuevo_largo = strlen(S) + (I2 - I1 + 1) * (Q - 1) + 1;
    size_t pos = 0;
    char* destiny = (char*) malloc(sizeof(char) * nuevo_largo);
    for (size_t i = 0; i < strlen(S); i++) {
        if (i < I1 || i > I2) {
            destiny[pos] = S[i];
            pos++;
            continue;
        }
        while (Q != 0) {
            for (int j = I1; j <= I2; j++) {
                destiny[pos] = S[j];
                pos++;
            }
            Q--;
        }
    }
    return destiny;
}
```

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `void (*F)(int i)`

Se trata de un puntero a función cuya firma recibe un entero con signo y no devuelve nada.

b) `static void B(float a, float b){};`

Se trata de la definición de una función con alcance local que toma como parámetros dos números de punto flotante.

c) `int *(*C)[5];`

Se declara la variable "C" como un puntero a un arreglo de 5 punteros a enteros con signo.

- Escribir un programa ISO C que reciba por argumento el nombre de un archivo de texto y lo procese sobre sí mismo (sin crear archivos intermedios ni subiendo todo su contenido a memoria). El procesamiento consiste en eliminar todas las líneas de 1 sola palabra.

```
#define MAX_LINEA 1024

bool leer_linea(FILE* f, long int* seek, char* linea) {
    fseek(f, *seek, SEEK_SET);
    char* ret = fgets(linea, MAX_PALABRA, f);
    *seek = ftell(f);
    return ret != NULL;
}
```

```
void escribir_linea(FILE* f, long int* seek, char* linea) {
    fseek(f, *seek, SEEK_SET);
    fwrite(linea, sizeof(char), strlen(linea), f);
    *seek = ftell(f);
}

bool debe_eliminar(char* linea) {
    for (int i = 0; i < strlen(linea); i++) {
        if (linea[i] == ' ') return false;
    }
    return true;
}

int main(int argc, char* argv[]) {
    char* file_name = argv[1];
    FILE* f = fopen(file_name, "r+");
    long int seek_lectura = 0;
    long int seek_escritura = 0;
    long int c_final = 0;
    char linea[MAX_PALABRA];
    while (leer_linea(f, &seek_lectura, linea)) {
        if (debe_eliminar(linea)) continue;
        escribir_linea(f, &seek_escritura, linea);
        c_final += strlen(linea);
    }
    fclose(f);
    truncate(file_name, sizeof(char) * c_final);
    return 0;
}
```

- Declare una clase de elección libre. Incluya todos los campos de datos requeridos con su correcta exposición/publicación, y los operadores ++, -, ==, >> (carga), << (impresión)constructor move y operador float()

```
class Complejo {
private:
    float real;
    float imaginario;
public:
    //Constructores y operadores asignación
    //Constructor por defecto de clase
    Complejo(const float real = 0, const float imaginario = 0);
    //Constructor por copia
    Complejo(const Complejo& copiable);
    //Constructor por movimiento
    Complejo(Complejo&& other);
    //Sobrecarga del operador asignación =
    Complejo& operator=(const Complejo& copiable);
    //Sobrecarga del operador asignación = por movimiento
    Complejo& operator=(Complejo&& copiable);
    //Sobrecarga de operadores
    Complejo operator+(const Complejo& c2) const; //Idem + - *
    bool operator==(const Complejo& c2) const; //Idem == != < > <= >=
    Complejo& operator++(); //prefix ++, ej ++a Idem ++ --
    Complejo operator++(int); //postfix ++, ej a++ Idem ++ --
```

```
    operator bool() const; //Operador de casteo, idem bool int float
    char double etc
    //Prototipos de sobrecarga de operadores globales
    //Opcion 1: indicar con friend la declaracion para darle acceso
    privado a la clase Complejo y evitar getters y setters.
    friend std::ostream& operator<<(std::ostream& output, const Complejo& c);
    friend std::istream& operator>>(std::istream& input, Complejo& c);
};
```

- ¿Qué es una macro de C? Describa las buenas prácticas para su definición y ejemplifique

Una macro es un alias que se puede incluir en el código que será reemplazado por lo que se haya definido cuando el compilador efectúe el paso de preprocesamiento. Las macros son capaces de realizar decisiones lógicas o funciones matemáticas. Se recomienda encerrar los parámetros de una macro entre paréntesis porque si se le pasa una expresión como parámetro, al expandir la macro se podría formar una expresión que no cumpla el orden de las operaciones matemáticas de la forma deseada

Por ejemplo:

```
#define CUBE_WRONG(A) A*A*A /*Macro incorrecta para el cubo*/
#define CUBE(A) (A)*(A)*(A) /*Macro correcta para el cubo*/

CUBE_WRONG(5+1) = 5+1*5+1*5+1 = 16 y debería dar 216
CUBE(5+1) = (5+1)*(5+1)*(5+1) = 216 y debería dar 216
```

- Escriba un programa que reciba por línea de comandos un Puerto y una IP. El programa debe recibir una única conexión e imprimir en stdout la sumatoria de los enteros recibidos en cada paquete. Un paquete está definido como una sucesión de números recibidos como texto, en decimal, separados por comas y terminado con signo igual (ej: `27,12,32=`). Al recibir el texto 'FIN' debe finalizar el programa ordenadamente liberando los recursos.

Básicamente se pide un servidor Pasos a seguir: - Bind - Listen - Accept - Recibir - Liberar

```
#define CONEX_EN_COLA 1
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    char* port = argv[1];
    char* ip = argv[2];

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;
    struct addrinfo *res = NULL;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(ip, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                ptr->ai_socktype,
                ptr->ai_protocol);
        if (skt != -1) break;
```

```
}

//Bind del socket
bind(skt, ptr->ai_addr, ptr->ai_addrlen);

//2) Listen
listen(skt, CONEX_EN_COLA);

//3) Accept
int peerskt = accept(skt, NULL, NULL);

char paquete[MAX_PAQUETE + 1];
char buffer[MAX_PAQUETE];
size_t pos_paquete = 0;
size_t leido = 0;
bool salir = false;

while (! salir) {
    //4) Recibir
    leido = recv(peerskt, buffer, MAX_PAQUETE, 0);
    for (size_t i = 0; i < leido; i++) {
        paquete[pos_paquete] = buffer[i];
        if (buffer[i] == '=') {
            paquete[pos_paquete] = '\0';
            if (strcmp(paquete, "FIN") == 0) {
                salir = true;
                break;
            } else {
                int sumatoria = 0;
                size_t pos = 0;
                char sub_buffer[MAX_PAQUETE];
                for (int i = 0; i < strlen(paquete) + 1; i++) {
                    if (paquete[i] == ',' || paquete[i] == '\0') {
                        sub_buffer[pos] = '\0';
                        sumatoria += atoi(&sub_buffer[0]);
                        pos = 0;
                    } else {
                        sub_buffer[pos] = paquete[i];
                        pos++;
                    }
                }
                printf("%d\n", sumatoria);
                pos_paquete = 0;
            }
        }
        pos_paquete++;
    }
}
//5) Liberar
shutdown(peerskt, SHUT_RDWR);
shutdown(skt, SHUT_RDWR);
close(skt);
close(peerskt);
freeaddrinfo(ptr);
```

```
    return 0;
}
```

- Describa el proceso de transformación de código fuente a un ejecutable. Precise las etapas, las tareas desarrolladas y los tipos de error generados en cada una de ellas.

El proceso de transformación de código a fuente a un ejecutable consta de tres etapas:

1. ***Preprocesado (o precompilación)***

El preprocesador acepta como entrada código fuente (ficheros .c y .h) y se encarga de:

a. Eliminar los comentarios

b. Interpretar y procesar las directivas de preprocesamiento, precedidas siempre por el símbolo #.

2. ***Compilación***

El compilador recibe los ficheros con código fuente preprocesados y analiza la sintaxis y la semántica de los mismos traduciéndolos y generando un fichero que contiene el código objeto (ficheros .o).

Sub etapas de la compilación:

a. Parseado del código fuente para armar un árbol semántico.

b. Traducción de los símbolos parseados en instrucciones de ensamblador.

c. Ensamble del archivo en lenguaje ensamblador obteniendo así un archivo binario, también conocido como el código objeto. (.obj o .o)

3. ***Enlazado***

El enlazador (linker) recibe ficheros con código objeto y resuelve las referencias a objetos externos que se encuentran en un fichero fuente generando un archivo ejecutable. Estas referencias son a objetos que se encuentran en otros módulos compilados, ya sea en forma de ficheros objeto o incorporados en alguna biblioteca.

- ¿Qué ventaja ofrece un lock raii frente al tradicional lock/unlock?

La principal ventaja reside en que el mutex se libera automáticamente cuando la variable de tipo `std::mutex` sale de scope.

Las ventajas de utilizar RAII residen en que al instanciarse objetos RAII en el stack, sus constructores adquieren los recursos automáticamente. Al irse de scope cada objeto se les invoca su destructor automáticamente y liberan sus recursos sin necesidad de hacerlo explícitamente. De esta manera el código C++ se simplifica y se hace más robusto a errores de programación.

- ¿Qué significa que una funciòn es bloqueante? ¿Cómo subsanaría esa limitación en términos de mantener el programa "vivo"?

Ya explicado

Final 12/2/2019

- ¿Qué es un functor? Ejemplifique

Un functor es un objeto que tiene sobrecargado el operador 'llamado' (operator()) y "actúa como una función". El mismo permite desacoplar el momento en que se le pasa los parámetros a una función del momento en que se inicia la ejecución de la misma.

Un uso común es para lanzar threads, por ejemplo:

```
#include <thread>
class Sumar {
public:
    Sumar(int a, int b): sumando_1(a), sumando_2(b) {}
    void operator()() {
        resultado = sumando_1 + sumando_2;
    }
    int obtener_resultado() const {
        return resultado;
    }
private:
    int sumando_1, sumando_2, resultado;
};

int main() {
    Sumar suma(3, 4); //pasaje de parámetros
    std::thread t1(suma); //ejecución
    t1.join();
    std::cout << suma.obtener_resultado() << std::endl //obtención del resultado
    return 0;
}
```

- Explique qué es cada uno de los siguientes, haciendo referencia a su inicialización, su comportamiento y el área de memoria donde residen:

a) ***Una variable global static***

Una variable global static puede inicializarse con una constante en tiempo de compilación y omitirse su inicialización. En caso de omitirse se inicializará con 0, null o el valor correspondiente al tipo según lo estipulado en el estándar dentro del lenguaje. Su visibilidad es global dentro del archivo en que fue definida ya que al ser static no es exportada al enlazador. Se almacena en el data segment.

b) ***Una variable local***

Una variable local es almacenada en el stack, se crea en el momento de ejecutar la función en la que fue definida y se destruye al finalizar la misma. No son inicializadas (a menos que el programador lo haga explícitamente), con lo cual su valor depende del contenido previo del stack hasta no ser inicializadas explícitamente. Su visibilidad es local a la función que la define, desde el punto en que fueron definidas hasta que termine el scope.

c) ***Un atributo de clase static***

Un atributo de clase static se almacena en el data segment y puede inicializarse con una contante en tiempo de compilación y omitirse su inicialización. En caso de omitirse se inicializará con 0, null o el valor correspondiente al tipo según lo estipulado en el estándar dentro del lenguaje. Su visibilidad está restringida a la clase que la define. El valor del atributo se comparte entre todas las instancias de la clase y en caso de declararse público es accesible desde fuera de la clase, anteponiendo el nombre de la misma (en lugar una instancia es `Clase::Attrib`). Es exportado al enlazador, con lo cual puede accederse desde otros archivos si se declara público.

- Declare una clase a elección considerando:

a) Atributos que son necesarios
b) Accesibilidad a la clase
c) Incluir los operadores +, ++, >> (impresión), << (carga desde consola), int

```
class Fraction {
public:
    Fraction(); //Constructor default
```

```
    Fraction (int denom, int num); //Constructor
    Fraction (const Fraccion& other); //Constructor por copia
    Fraction& operator= (const Fraccion& other);
    Fraction operator+(const Fraccion& other) const;
    Fraction& operator++();
    std::istream& operator>>(std::istream& in, Fraction& a);
    std::ostream& operator<<(std::ostream& out, const Fraction& a);
    operator int() const;
    void simplificate();

 private:
    int numerador, denominador;
};
```

- Escriba un programa que imprima por salida estándar los números entre 1 y 100, en orden ascendente. Se pide que los números sean contabilizados por una variable global única y que los pares sean escritos por un hilo mientras que los impares por otro. Contemple la correcta sincronización entre hilos y la liberación de los recursos utilizados.

```
class MonitorContador {
private:
    std::mutex mx_contador;
    int& m_contador;
    std::condition_variable cv_par, cv_impar;
    bool es_impar = true;

public:
    MonitorContador(int& contador): m_contador(contador) {}
    void imprimir_par() {
        std::unique_lock<std::mutex> lock(mx_contador);
        while (es_impar) {
            cv_par.wait(lock);
        }
        std::cout << m_contador << std::endl;
        m_contador++;
        es_impar = true;
        cv_impar.notify_one();
    }

    void imprimir_impar() {
        std::unique_lock<std::mutex> lock(mx_contador);
        while (! es_impar) {
            cv_impar.wait(lock);
        }
        std::cout << m_contador << std::endl;
        m_contador++;
        es_impar = false;
        cv_par.notify_one();
    }
};

int contador = 1;
int main() {
    MonitorContador monitor(contador);
    std::thread t1([&]() {
```

```
        for (int i = 0; i < 50; i++) {
            monitor.imprimir_par();
        }
    });
    std::thread t2([&]() {
        for (int i = 0; i < 50; i++) {
            monitor.imprimir_impar();
        }
    });
    t1.join();
    t2.join();
    return 0;
}
```

- ¿Qué es la compilación condicional? En qué etapa del proceso de transformación de código se resuelve. Ejemplifique mediante código C, dando un caso de uso útil.

La compilación condicional permite incluir o excluir distintos fragmentos de código según el valor de distintas constantes conocidas al momento de la compilación. La misma se resuelve en la etapa de preprocesado. En general es útil para escribir código portable a distintas plataformas o para incluir código de depuración.

Por ejemplo:

```
#ifdef DEBUG
#define assert(x) if(!(x)) {
    fprintf(stderr, "Assert fail en [%s, %d]", __FILE__, __LINE);
    abort();
}
#else
#define assert(x)
#endif
int main() {
    assert(1 == 0); //Solo agrega código para chequear esto si el debug está definido.
    return 0;
}
```

- Escriba una función ISO C que permita procesar un archivo de texto sobre sí mismo, que contenga un palabra por línea. El procesamiento consiste en ordenar las palabras (líneas) alfabéticamente considerando que el archivo no entra en memoria.

```
#define MIN_ARGS 2
#define ARG_ENTRADA 1
#define MAX_PALABRA 1000

bool leer_palabra(FILE* archivo, int* cursor, char* palabra, int sz_palabra) {
    fseek(archivo, *cursor, SEEK_SET);
    char* ret = fgets(palabra, sz_palabra, archivo);
    *cursor = ftell(archivo);
    return ret != NULL;
}

void escribir_palabra(FILE* archivo, int* cursor, const char* palabra) {
    fseek(archivo, *cursor, SEEK_SET);
    fwrite(palabra, strlen(palabra), 1, archivo);
    *cursor = ftell(archivo);
}
```

```
int main(int argc, char* argv[]) {
    if (argc < MIN_ARGS) return 1;
    FILE* entrada = fopen(argv[ARG_ENTRADA], "r+");
    bool esta_ordenado = false, eof;
    while (! esta_ordenado) {
        int cursor_lectura = 0;
        int cursor_escritura = 0;
        esta_ordenado = true;
        eof = false;
        while (! eof) {
            char palabra_1[MAX_PALABRA], palabra_2[MAX_PALABRA];
            if (! leer_palabra(entrada, &cursor_lectura, palabra_1, MAX_PALABRA)) {
                eof = true;
                continue;
            }
            if (! leer_palabra(entrada, &cursor_lectura, palabra_2, MAX_PALABRA)) {
                eof = true;
                continue;
            }
            if (strcmp(palabra_1, palabra_2) > 0) {
                esta_ordenado = false;
                escribir_palabra(entrada, &cursor_escritura, palabra_2);
                escribir_palabra(entrada, &cursor_escritura, palabra_1);
                cursor_lectura = cursor_escritura = cursor_lectura - strlen(palabra_1);
            }
        }
    }
    fclose(entrada);
    return 0;
}
```

- ¿Por qué las librerías que usan Templates se publican con todo el código fuente y no como un .h y .o/.obj?

Los templates permiten generar código específico según el tipo en el que se especialice el código con templates. Esto requiere conocer en el momento de la compilación el código completo ya que el contenido en el . O generado dependerá de las especializaciones. Por ejemplo, si se tiene `std::list<int>` y `std::list<char>` el compilador generará el código de `std::list` dos veces, uno especializado en int y el otro en char.

- ¿Qué significa la palabra virtual antepuesta a un método de una clase? ¿Qué cambios genera a nivel compilación y al momento de ejecución?

La palabra virtual antepuesta a un método de una clase habilita lo que se conoce como dynamic binding. Esto quiere decir que el método a ejecutar se resuelve en tiempo de ejecución. Esto permite utilizar polimorfismo basado en herencia como se esperaría que funcione en la mayoría de los lenguajes orientados a objetos, es decir, los métodos de las clases derivadas reemplazan a los de la clase base independientemente de cómo se lo acceda (desde un puntero a la base o al tipo derivado). En el código objeto esto genera una diferencia debido a que el dynamic binding requiere tener una tabla de métodos virtuales para cada objeto (la VTable) de modo de permitir determinar en tiempo de ejecución qué método debe ser llamado (resolución dinámica). Esto empeora la performance comparado con static binding ya que requiere un nivel más de indirección (la VTable). Sin utilizar virtual el compilador sabe en el momento de compilación qué método va a llamar (static binding) y puede emitir el llamado directamente. Con el uso del modificador virtual el compilador ya no tiene esta información y utiliza las VTables, lo cual requiere generar código para primero ubicar el método y luego llamarlo, siendo más lento que el caso anterior.

- Escriba un programa (desde la inicialización hasta la liberación de los recursos) que reciba paquetes

de texto delimitados por corchetes angulares (<...>) y los imprima completos por pantalla. Al recibir un paquete vacío (<>) debe cerrarse ordenadamente. No considere errores.

Basicamente se pide un servidor Pasos a seguir:

- Bind
- Listen
- Accept
- Recibir
- Liberar

```
#define CONEX_EN_COLA 1
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    char* port = argv[1];
    char* ip = argv[2];

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;
    struct addrinfo *res = NULL;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(ip, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                ptr->ai_socktype,
                ptr->ai_protocol);
        if (skt != -1) break;
    }

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    int peerskt = accept(skt, NULL, NULL);

    char paquete[MAX_PAQUETE + 1];
    char buffer[MAX_PAQUETE];
    size_t pos_paquete = 0;
    size_t leido = 0;
    bool salir = false;

    while (! salir) {
```

```
        //4) Recibir
        leido = recv(peerskt, buffer, MAX_PAQUETE, 0);
        for (size_t i = 0; i < leido; i++) {
            paquete[pos_paquete] = buffer[i];
            if (buffer[i] == '>') {
                if (strcmp(paquete, "<>") == 0) {
                    salir = true;
                    break;
                } else {
                    paquete[pos_paquete] = '\0';
                    printf("%s", &paquete[1]);
                    pos_paquete = 0;
                }
            }
            pos_paquete++;
        }
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    close(peerskt);
    freeaddrinfo(ptr);
    return 0;
}
```

- Describa el concepto de loop de eventos (event loop) utilizado en programación orientada a eventos, y, en particular, en entornos de interfaz gráfica (GUIs)

Ya explicado

Final 11/12/2018

- ¿Por qué las clases que utilizan templates se declaran y se definen en los .h?

Cuando se instancia un template en C++ el compilador crea una nueva clase. La clase tiene todos los lugares donde se colocaron los argumentos del template reemplazados con los argumentos que se pasaron en la creación. El compilador va a crear una nueva clase por cada tipo de argumento que se pasa. Por lo tanto, el compilador necesita tener acceso a la implementación de la clase y sus métodos antes de encontrar estos argumentos para instanciarlos con el argumento del template. Si las implementaciones de las clases template no estuvieran en el header no compilaría.

- ¿Qué es un functor? Ejemplifique

Ya explicado.

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `extern float (*I)[3];`

Se declara la variable "I" que apunta a un array de 3 posiciones de números de punto flotante definido en un archivo externo.

b) `static int *C[3];`

Se declara y define (intuyo que porque es static) la variable "C" que apunta a un array de 3 posiciones de números enteros signados, solo visible desde el archivo donde es declarada.

c) `static short F(const float *a);`

Se declara la función "F" que recibe como primer argumento un puntero a un número de punto flotante constante. Esto quiere decir que se puede modificar el puntero en sí mismo pero no el valor al que apunta el puntero. En cambio si se hubiese declarado `float* const a`, se declara "a" como un puntero constante a una variable float. Ahora no se puede modificar el puntero en sí mismo pero sí el valor apuntado.

- ¿Qué es un deadlock? Ejemplifique

Un deadlock se produce cuando dos o más procesos que corren en hilos diferentes se bloquean permantemente. Un ejemplo sería que el proceso 1 tome un recurso A y espere que se libere el recurso B para liberar el recurso A y tomar el recurso B. Al mismo tiempo un proceso 2 tiene tomado el recurso B y espera que se libere el recurso A para liberar el recurso B y luego tomar el recurso A. De esta manera ambos procesos se bloquean mutuamente.

Las condiciones para que se presente un Deadlock son:

1. Exclusión mutua: los recursos son de uso exclusivo, es decir, sólo un proceso puede hacer uso de un recurso
2. Retención y espera: el proceso mantiene la posesión de un recurso mientras espera recursos adicionales
3. No apropiación: el proceso no suelta el recurso hasta que termine su uso
4. Espera circular: el proceso tiene uno o más recursos que son requeridos por el siguiente proceso

Un ejemplo:

```
#include <mutex>
#include <thread>
std::mutex m; //mutex global
void callback() {
    m.lock(); //Se bloquea a la espera que se libere el mutex tomado por el lock principal
    m.unlock(); //Nunca llega a liberar el mutex porque se bloqueó esperando para tomarlo
}
int main() {
    std::thread t;
    m.lock(); //Toma el mutex
    t = std::thread(callback); // Lanza un hilo
    t.join(); //Se bloquea a la espera que el hilo termine su ejecución
    m.unlock(); //Nunca llega a liberar el mutex porque se bloqueó en el join
    return 0;
}
```

- Explique qué es y para qué sirve una variable de clase (o atributo estático) en C++. Mediante un ejemplo de uso, indique cómo se define dicha variable, su inicialización y el acceso a su valor para realizar una impresión simple dentro de un main.

Una variable de clase o atributo estático es una variable propia de la clase que la contiene y no de instancias de la misma. Esto quiere decir que cuando se declara un atributo estático en una clase, todos los objetos que sean instancia de esa clase usarán la misma variable estática. Solo ocupará una posición de memoria independientemente del número de instancias que se creen de la clase que la define. De esta manera al realizar un cambio en la variable se verá reflejado en el resto de las instancias.

- ¿Qué significa que una función es bloqueante? ¿Cómo subsanaría esa limitación en término de mantener el programa 'vivo'?

Ya explicado

- Explique qué es y para qué sirve un constructor move en C++. Indique cómo se comporta el sistema si éste no es definido por el desarrollador.

El constructor move en C++ se utiliza principalmente para evitar copiar un objeto que no tiene sentido ser copiado. Por ejemplo, si se tuviese dos punteros a FILE y se llama doblemente a fclose causaría un comportamiento indeterminado en el código. Al crear un constructor por movimiento, el compilador no genera automáticamente el constructor por copia ni el operador de asignación. Además, es más eficiente que el constructor por copia ya que no se generan deep copies al mover recursos. Se "roban" los recursos del objeto fuente y se los asigna al objeto destino, dejando al objeto fuente en un estado default pero válido (debido a que posteriormente llamará a su respectivo destructor). Si no está definido por el desarrollado no se genera un constructor por movimiento default por lo que se llama al constructor por copia y al operador de asignación.

- Escribir un programa ISO C que procese el archivo "valuesword.dat" sobre sí mismo, eliminando los words (2 bytes) múltiplos de 16

```
int leer(FILE* f, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    int out = fgetc(f);
    *seek = ftell(f);
    return out;
}

void escribir(FILE* f, char c, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    fputc(c, f);
    *seek = ftell(f);
}

int main(int argc, char* argv[]) {
    FILE *f;
    int c = 0;
    long int c_final = 0;
    long int seek_lectura = 0;
    long int seek_escritura = 0;

    char buffer[3];
    memset(buffer, 0, 3);

    bool inicio = true;
    int pos = 0;

    f = fopen("valuesword.dat","r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;

        if (inicio) {
            buffer[pos] = c;
            pos++;
            if (pos == 2) inicio = false;
            escribir(f, c, &seek_escritura);
            continue;
        }

        int value = atoi(buffer);
        if (value % 16 == 0) {
            c_final -= 2;
```

```
            seek_escritura = c_final - 1;
        }
        escribir(f, c, &seek_escritura);
        buffer[0] = buffer[1];
        buffer[1] = c;
    }
    int value = atoi(buffer);
    if (value % 16 == 0) {
        c_final -= 2;
        seek_escritura = c_final - 1;
    }
    fclose(f);
    truncate("valuesword.dat", sizeof(char) * c_final);
    return 0;
}
```

- Implemente la función *void ValorHex(char* hex, int* ent)* que interprete la cadena *hex* (de símbolos hexadecimales) y guarde el valor correspondiente en el entero indicado por *ent*.

```
void ValorHex(char* hex, int* ent) {
    *ent = 0;
    int exponent = 0;
    int number = 0;
    for (int i = strlen(hex) - 1; i >= 0; i--) {
        int to_multiply = pow(16, exponent);
        if (hex[i] >= '1' && hex[i] <= '9') {
            number = atoi(&hex[i]);
        } else if (hex[i] >= 'a' && hex[i] <= 'f') {
            number = hex[i] - 'a' + 10;
        } else if (hex[i] >= 'A' && hex[i] <= 'F') {
            number = hex[i] - 'A' + 10;
        }
        *ent += number * to_multiply;
        exponent++;
    }
}
```

- Implemente una función C++ denominada *Intersección* que reciba dos listas de elementos y devuelva una nueva lista con los elementos que se encuentran en ambas listas:

```
std::list<T> Interseccion(std::list <T> a, std::list<T> b);
```

```
template <class T>
std::list<T> Interseccion(std::list<T> a, std::list<T> b) {
    std::list<T> resultado;
    for (auto i = a.begin(); i != a.end(); i++) {
        auto item = std::find(b.begin(), b.end(), *i);
        if (item != b.end()) {
            resultado.push_back(*i);
        }
    }
    return resultado;
}
```

Final 24/7/2018

- Escriba un programa que escriba por salida estándar los números entre 0 y 1000 ordenadamente. Se pide que los números pares sean escritor por un hilo mientras que los impares por otro. Contemple la correcta sincronización entre hilos y la liberación de los recursos utilizados.

Ya explicado

- Defina una rutina en C que se conecte a la IP 10.9.8.7, puerto 7777 y procese la información recibida. El proceso consiste en recibir textos numéricos utilizando el `\n` como caracter delimitador. Para cada entero recibido se debe enviar su valor convertido en 32 bits big-endian en modo binario sin delimitadores. El proceso finaliza al recibir el valor 0.

Interpreto que es siempre el mismo cliente y que cuando le manda el valor 0 terminó la comunicación. Lo del modo binario ni idea.

```
int main(int argc, char* argv[]) {
    const char* port = "7777";
    const char* ip = "10.9.8.7";

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(ip, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                     ptr->ai_socktype,
                     ptr->ai_protocol);
        if (skt != -1) break;
    }

    int val = 1;
    setsockopt(skt, SOL_SOCKET,
                        SO_REUSEADDR, &val, sizeof(val));

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    char buffer[MAX_PAQUETE];
    char num[MAX_PAQUETE];
    size_t leido = 0;
    size_t pos = 0;

    int peerskt = accept(skt, NULL, NULL);
```

```
    while (true) {
        //4) Recibir
        leido = recv(peerskt, &buffer[pos], MAX_PAQUETE, 0);
        pos += leido;
        int first_pos = 0;
        for (int i = 0; i < strlen(buffer); i++) {
            if (buffer[i] == '\n') {
                memcpy(num, buffer, i+2);
                num[i] = '\0';
                uint32_t acumulador = atoi(&buffer[first_pos]);
                acumulador = htonl(acumulador);
                send(peerskt, &acumulador, 4, MSG_NOSIGNAL);
                first_pos += i;
                pos = 0;
                memset(buffer, 0, pos);
            }
        }
        if (atoi(buffer) == 0) break;
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    close(peerskt);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    freeaddrinfo(ptr);
    return 0;
}
```

- ¿Por qué las librerías que usan templates se publican con todo el código fuente y no como un .h/.o?

Ya explicado

- Escriba una función C llamada *Agrandar* que reciba por parámetro 1 cadena (S), dos índices ($I1$ e $I2$) y una cantidad (Q). La función debe retornar una copia de S salvo los caracteres entre los índices I1 e I2 que serán duplicados Q veces en la misma posición.

Ej: `Agrandar("Chau", 1, 2, 3)` retorna `Chahahau`

Ya explicado

- Explique qué es y para qué sirve un constructor move en c++. Indique cómo se comporta el sistema si este no es definido por el desarrollador.

Ya explicado

- Defina la clase URL para permitir almacenar las siguientes propiedades: protocolo(http), host (fi.uba.ar), port(80) y file (index.php o resources/img/logo.png). A su vez se pide que implemente los siguientes operadores: operator<<, operator== y operator=.

```
class URL {
private:
    std::string protocol;
    std::string host;
    int port;
    std::string file;
public:
    URL(std::string protocol, std::string host, int port, std::string file):
        protocol(protocol), host(host), port(port), file(file) {}
```

```
    friend std::ostream& operator<<(std::ostream& out, const URL& url) {
        out << "protocol: " << url.protocol << " host: " << url.host
        << " port: " << url.port << " file: " << url.file << std::endl;
        return out;
    }
    bool operator==(const URL& other) const {
        return this->protocol == other.protocol && this->host == other.host
        && this->port == other.port && this->file == other.file;
    }
    URL& operator=(const URL& other) {
        if (this != &other) {
            this->protocol = other.protocol;
            this->host = other.host;
            this->port = other.port;
            this->file = other.file;
        }
        return *this;
    }
};
```

- ¿Qué es un functor? Ejemplifique

Ya explicado

- Describa el concepto de loop de eventos (evento loop) utilizado en programación orientada a eventos y en particular en entornos de interfaz gráfica (GUIs).

Ya explicado

- Escriba un programa C que reciba por parámetro el nombre de un archivo de números binarios de 16 bits y lo procese sobre sí mismo. El procesamiento consiste en repetir los números que sean "múltiplos de 5 + 1" (6, 11, 16…) (El archivo se agranda)

No entiendo la consigna

- Describa las siguientes declaraciones/definiciones globales:

a) `int (*A)();`

Se declara un puntero a función "A" que no recibe ningún parámetro y devuelve un int.

b) `char* B(unsigned a, short b) {}`

Se define B como una función que recibe dos parámetros: "a" un unsigned y "b" un short y que devuelve un char*.

c) `static unsigned**C[3];`

Se declara la variable "C" que se trata de un array de alcance local de 3 punteros a punteros a elementos unsigned.

Final 17/7/2018

- Explique qué es cada uno de los siguientes ítems, haciendo referencia a su inicialización, su comportamiento y el área de memoria donde residen:

a) Una variable global static
b) Una variable local static
c) Un atributo de clase static

Ya explicado

- ¿Cómo se logra que 2 threads accedan (lectura/escritura a un mismo recurso sin que se genere problemas de consistencia? Ejemplifique

Ya explicado

- ¿Qué es una macro en C?

Ya explicado

- Explique breve y concretamente qué es `f: char (*f)(float*, unsigned[3])`

Se trata de la declaración de un puntero a función "f" que recibe dos parámetros: un puntero a un número de punto flotante y un arreglo de 3 elementos unsigned y devuelve un char.

- Escriba un programa ISOC que procese el archivo "nros1byte.dat" sobre sí mismo, eliminando los bytes múltiplos de 6.

Ya explicado

- Escriba el .h de una biblioteca de funciones ISO C para cadenas de caracteres. Incluya al menos 4 funciones.

No entiendo si hay que inventar o si hay que usar las existentes. Yo voy a usar las existentes:

```
// Devuelve el largo de una cadena de carácteres
size_t strlen(const char*);
// Copia la cadena scr en dest
char* strcpy(char* dest, const char* src);
// Encuentra la primera ocurrencia de first en second
char* strstr(const char* second, const char* first);
// Compara ambas cadenas. Devuelve 0 si son iguales, un número menor a 0
// si first es menor a second y menor a 0 si first es mayor a second
int strcmp(const char* first, const char* second);
```

- Implemente una función C++ denominada *SinSegunda* que reciba dos listas de elementos y devuelva una nueva lista con los elementos de la primera que no están en la segunda:

```
std:::list<T> SinSegunda(std::list<T> a, std::list<T> b);
```

Ya explicado

- Escriba un programa que reciba por línea de comandos un Puerto y una IP. El programa debe aceptar una única conexión e imprimir en stdout todo lo recibido. Al recibir el texto FIN debe finalizar el programa sin imprimir dicho texto.

Básicamente se pide un servidor Pasos a seguir: - Bind - Listen - Accept - Recibir - Liberar

```
#define CONEX_EN_COLA 1
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    const char* port = "8080";
    const char* ip = "localhost";

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;
    struct addrinfo *res = NULL;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
```

```
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(ip, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                     ptr->ai_socktype,
                     ptr->ai_protocol);
        if (skt != -1) break;
    }

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    int peerskt = accept(skt, NULL, NULL);

    char paquete[MAX_PAQUETE];
    char buffer[MAX_PAQUETE];
    size_t leido = 0;
    bool salir = false;
    size_t pos_paquete = 0;

    while (! salir) {
        //4) Recibir
        leido = recv(peerskt, buffer, MAX_PAQUETE, 0);
        if (leido == 0) {
            salir = true;
        }
        for (size_t i = 0; i < leido; i++) {
            paquete[pos_paquete] = buffer[i];
            if (paquete[pos_paquete] == '\0') { //fin del paquete
                if (strcmp(paquete, "FIN") == 0) {
                    salir = true;
                    break;
                }
                printf("%s", paquete);
                pos_paquete = 0;
            } else {
                pos_paquete++;
            }
        }
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    close(peerskt);
    freeaddrinfo(ptr);
    return 0;
}
```

- Indique la salida del siguiente programa:

```
class A {
    A() {
        cout << "A()" << endl;
    }
    ~A() {
        cout << "~A()" << endl;
    }
};

class B: public A {
    B() {
        cout << "B()" << endl;
    }
    ~B() {
        cout << "~B()" << endl;
    }
}

int main() {
    B b;
    return 0;
}
```

La salida del programa es:

```
A();
B();
~B();
~A();
```

Esto se debe a que primero se llama al constructor de la clase base y luego al de la clase derivada y lo opuesto pasa con el destructor (como si se tratara de una pila).

- Describa el proceso de transformación de código fuente a un ejecutable. Precise las etapas y las tareas desarrolladas en cada una de ellas.

Ya explicado

Final 10/7/2018

- Escriba una rutina que procese un archivo binario indicado por parámetro sobre sí mismo sumarizando los listados de números que posee almacenado. La sumarización consiste en recorrer los valores enteros de 32 bits grabados en formato big-endian y acumular sus valores hasta encontrar el valor 0xffffffff que se considera un separador entre listados. Todos los valores enteros detectados son reemplazados por su sumatoria (en el mismo formato) manteniendo luego el elemento separador. Considere archivos bien formados.

Dejo el ejemplo de cómo crear un archivo binario desde C:

```
bool leer_4bytes(FILE* f, long int* seek_lectura, uint32_t* buffer) {
    fseek(f, *seek_lectura, SEEK_SET);
    int bytes_read = fread(buffer, 1, 4, f);
    *seek_lectura = ftell(f);
    return bytes_read != 0;
```

```
}

void escribir_4bytes(FILE* f, long int* seek_escritura, uint32_t* buffer) {
    fseek(f, *seek_escritura, SEEK_SET);
    fwrite(buffer, 4, 1, f);
    *seek_escritura = ftell(f);
}

int main() {
    FILE* f_bin = fopen("new.bin", "wb");
    uint32_t value = 0x2211;
    fwrite(&value, 1, 4, f_bin);
    uint32_t value_2 = 0x3344;
    fwrite(&value_2, 1, 4, f_bin);
    uint32_t value_3 = 0xffffffff;
    fwrite(&value_3, 1, 4, f_bin);

    uint32_t value_4 = 0x1111;
    fwrite(&value_4, 1, 4, f_bin);
    uint32_t value_5 = 0x2222;
    fwrite(&value_5, 1, 4, f_bin);
    fwrite(&value_3, 1, 4, f_bin);

    fclose(f_bin);

    FILE *f = fopen("new.bin", "rb+");

    uint32_t acumulador = 0;
    uint32_t buffer;
    long int seek_escritura = 0;
    long int seek_lectura = 0;
    long int c_final = 0;
    uint32_t separador = 0xffffffff;

    while (leer_4bytes(f, &seek_lectura, &buffer)) {
        if (buffer != separador) {
            acumulador += buffer;
        } else {
            c_final += 8;
            escribir_4bytes(f, &seek_escritura, &acumulador);
            escribir_4bytes(f, &seek_escritura, &separador);
            acumulador = 0;
        }
        buffer = 0;
    }

    fclose(f);
    truncate("new.bin", sizeof(char) * c_final);
    return 0;
}
```

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `extern char (*I)[3];`

Se declara la variable “I” que apunta a un array de 3 posiciones de tipo char definida en un archivo externo.

b) `static float *C[3];`

Se declara una variable “C” que apunta a un array de 3 posiciones de punteros a número de punto flotante de alcance local.

c) `static int F(const short* a);`

Se declara una función “F” de alcance local que recibe un puntero a un short constante y devuelve un int.

- Explique que es y para que sirve una variable de clase (o atributo estático) en C++. Mediante un ejemplo de uso, indique como se define dicha variable, su inicialización y el acceso a su valor para realizar una impresión simple dentro de un main.

Ya explicado

- Defina una rutina en C que se conecte a la IP 8.8.8.8 al puerto 8192 y reciba un archivo binario. El archivo recibido debe ser descargado a un archivo llamado recibido.bin cuidando el uso de recursos de memoria. El archivo finaliza al detectar la conexión cerrada.

No llego a hacerlo

- Declare la clase *Oraciones* que almacena una acumulación de líneas de texto (strings) en cierto orden e incluye las operaciones de agregación y eliminación de líneas. La clase debe poseer los operadores usuales de copia, asignación, salida a flujo (<<), comparación (==), agregación (+), substracción (-). Los dos últimos operadores deben admitir argumentos del tipo string en formato C (char*) y C++ (std::string).

```
class Oraciones {
private:
    std::list<std::string> lines;
public:
    Oraciones() {}
    Oraciones(const Oraciones& other) {
        this->lines = other.lines;
    }
    Oraciones& operator=(const Oraciones& other);
    friend std::ostream& operator<<(std::ostream& out, const Oraciones& other);
    bool operator==(const Oraciones& other) const;
    void add(std::string added) {
        lines.push_back(added);
    };
    void remove(std::string removed) {
        lines.remove(removed);
    }

    Oraciones operator+(std::string added) const {
        Oraciones oraciones = *this;
        oraciones.add(added);
        return oraciones;
    }
    Oraciones operator-(std::string removed) const {
        Oraciones oraciones = *this;
        oraciones.remove(removed);
        return oraciones;
    }
};
```

- Escriba una clase template llamada *Sumador* que reciba por constructor un vector de elementos genéricos. La clase Sumador debe incluir un metodo llamado sumar que acumule los valores del tipo recibido y retorne un nuevo objeto del tipo genérico. ¿Qué restricciones se le piden al tipo genérico en la definición de Sumador?

```
template<class T>
class Sumador {
private:
    std::vector<T> elementos;
public:
    Sumador(std::vector<T> elementos): elementos(elementos) {}
    T sumar() {
        T result = *elementos.begin();
        for (auto i = ++elementos.begin(); i != elementos.end(); i++) {
            result += *i;
        }
        return result;
    }
};
```

El tipo genérico debe soportar la operación de suma, es decir, tener sobrecargado el operador +.

- Explique la diferencia entre las etapas de compilación y enlazado (linking). Escriba un breve ejemplo de código con errores para cada una de ellas indicándolos de forma clara

La etapa de **compilación** implica analizar la sintaxis y la semántica de los ficheros con código fuente preprocesados y generar un fichero que contiene el código objeto. Por otro lado, la etapa de **enlazado** implica recibir esos ficheros con código objeto y resolver las referencias a objetos externos que se encuentran en un fichero fuente generando como salida un archivo ejecutable.

```
int sumar(int num1, int num2);
int main() {
    int resultado = sumar(1, 2);
    num_t numero;
    return 0;
}
```

Error de compilación: se declara una variable de tipo num_t de la cual se desconoce el tipo. Esto genera un error de compilación ya que el compilador debe reservar memoria en el stack para dicha variable y no sabe cuánto ocupa ya que no está definida.

Error de enlazado: la función `int sumar(int, int)` no se encuentra definida pero sí declarada, motivo por el cual se pudo compilar pero no linkear. En el proceso de linking se reemplaza la tabla de símbolos creada en la etapa de compilación por los valores propiamente dichos y al no estar definida la función no puede generarse ningún valor para esa línea y se produce un error.

- Defina el concepto de mutex y de un ejemplo de uso. Indique en qué casos es necesario

El **mutex** es un objeto con dos estados posibles, tomado y liberado, que puede ser manipulado desde varios hilos simultáneamente. Cuando un hilo solicita el mutex lo recibe de inmediato si está liberado. Cualquier otro hilo que lo solicite posteriormente quedará suspendido a la espera del mutex. Si el primer hilo lo libera, alguno de los hilos en espera lo recibirá a continuación. Pudiendo esto repetirse hasta que no haya más hilos y el mutex quede nuevamente liberado.

Es necesario cuando los hilos tienen un recurso compartido y acceden al mismo para operaciones de escritura. Como el sistema operativo puede interrumpir en cualquier momento la ejecución, el resultado se vuelve no determinístico si no se organiza la adquisición de recursos.

Ejemplo de uso: Cuando el primer hilo (A) intente tomar el mutex lo encontrará liberado. Lo tomará

inmediatamente y procederá a acceder al recurso en la siguiente línea. Si en ese instante el hilo B intenta tomar el mutex, suspenderá su ejecución a la espera de que se libere.

```
#include <mutex>
#include <thread>
std::mutex m; //Mutex global
int recursoCompartido = 0;
void foo(){
    m.lock(); // Se solicita el mutex
    recursoCompartido++; //Accedo al recurso protegido
    m.unlock(); // Se libera el mutex
}

int main() {
    std::thread tA, tB;
    tA = std::thread(foo); //Instancio el hilo A que corre en la funcion foo()
    tB = std::thread(foo); //Instancio el hilo B que corre en la funcion foo()
    tA.join(); //Espera a que el hilo A termine y luego lo une al hilo principal (main)
    tB.join(); //Espera a que el hilo B termine y luego lo une al hilo principal (main)
    return 0;
}
```

- ¿Qué es un thread? ¿Qué funciones se utilizan para manipularlos(lanzarlos, etc)?

Un **thread** es la unidad básica de ejecución del SO. Cualquier programa que se ejecute consta al menos de un thread. Cada thread tiene:

a. Su propio PC

b. Su propio conjunto de registros

c. Su propio stack

d. Espacio de direcciones compartido

e. Acceso a la misma data

En C++ el thread se lanza llamando al constructor: `std::thread(function, args)`. Si la función no recibe parámetros se puede llamar a `std::thread(function, this)`. La función `std::thread::join()` se utiliza para esperar a que el hilo finalice su ejecución y luego unirlo al hilo principal.

- Escriba una función de C llamada *strncat_new* que reciba 3 parámetros: dos punteros a carácter constante (*S1* y *S2*) y un entero sin signo (*L*). La función debe concatenar S1 con S2 y retornar una nueva cadena de caracteres considerando L como tamaño máximo para cualquiera de los elementos (S1, S2 y la nueva cadena). La función debe detectar condiciones de error respecto de la longitud y retornar NULL en cualquier caso.

Ya explicado

Final 3/7/2018

- ¿Qué es la compilación condicional? ¿En qué etapa del proceso de transformación de código se resuelve? Ejemplifique mediante un código C dando un caso de uso útil.

Ya explicado

- Implemente la función *void String_a_Int(char* bin, int* ent)* que interprete la cadena bin (de 32 1s/0s) y guarde el valor correspondiente en el entero indicado por ent.

```
void String_a_Int(char* bin, int* ent) {
    *ent = 0;
    int exponent = 0;
    for (int i = strlen(bin)- 1; i >= 0; i--) {
        if (bin[i] == '1') *ent += pow(2, exponent);
        exponent++;
    }
}
```

- Escribir un programa que procese un archivo binario de enteros sin signo sobre sí mismo. El procesamiento consiste en leer pares de enteros de 1 byte cada uno y reemplazarlos por 3 enteros (el archivo se agranda): su suma, su resta y el OR lógico entre ambos

```
uint8_t leer_reversa(FILE* f, int* seek) {
    int previous_seek = *seek;
    fseek(f, *seek, SEEK_END);
    uint8_t r;
    fread(&r, 1, 1, f);
    *seek = previous_seek - 1;

    return r;
}

void escribir_reversa(FILE* f, int* seek, uint8_t* w) {
    int previous_seek = *seek;
    fseek(f, *seek, SEEK_END);
    fwrite(w, 1, 1, f);
    *seek = previous_seek - 1;
}

int main(int argc, char* argv[]) {

    FILE* f_bin = fopen("arch.bin", "wb");
    uint32_t value = 0x1100;
    fwrite(&value, 1, 2, f_bin);
    uint32_t value_2 = 0x1010;
    fwrite(&value_2, 1, 2, f_bin);

    uint32_t value_4 = 0x1111;
    fwrite(&value_4, 1, 2, f_bin);
    uint32_t value_5 = 0x0101;
    fwrite(&value_5, 1, 2, f_bin);

    fclose(f_bin);

    FILE* f = fopen("arch.bin", "rb+");

    uint16_t buffer;

    long int c_final = 0;
    long int c_extra = 0;

    while (fread(&buffer, 2, 1, f) != 0)  {
        c_extra++;
        c_final += 3;
```

```
    }

    fclose(f);
    truncate("arch.bin", sizeof(uint8_t) * c_final);

    int seek_lectura = -1;
    int seek_escritura = -1;

    f = fopen("arch.bin","rb+");

    seek_lectura -= (c_extra);
    uint8_t r = 0;
    uint8_t r_previo = 0;
    int read = 0;

    while (seek_lectura != (-c_final - 1)) {
        if (read < 2) {
            r = leer_reversa(f, &seek_lectura);
            read++;
            if (read == 1) r_previo = r;
            if (read != 2) continue;
        }

        read = 0;

        uint8_t suma = r + r_previo;
        uint8_t resta = r_previo - r;
        uint8_t or_logico = r | r_previo;

        escribir_reversa(f, &seek_escritura, &or_logico);
        escribir_reversa(f, &seek_escritura, &resta);
        escribir_reversa(f, &seek_escritura, &suma);
    }
    fclose(f);
    return 0;
}
```

- Indicar la salida del siguiente programa:

```
class Base {
public:
    static void f1() {
        cout << "Base.f1" << endl;
    }
    virtual void f2() {
        cout << "Base.f2" << endl; f1();
    }
    virtual void f3() {
        cout << "Base.f3" << endl; f2(); f1();
    }
};

class Derivada : Base {
    static void f1() {
        cout << "Derivada.f1" << endl;
```

```
    }
    void f2() {
        cout << "Derivada.f2" << endl;
        f1();
    }
    void f3() {
        cout << "Derivada.f3" << endl;
        f2();
        f1();
    }
};

void main() {
    Derivada D;
    Base* pB = &D;
    Derivada* pD = &D;
    pD->f1();
    pD->f2();
    pB->f1();
    pB->f2();
}
```

La salida es:

```
Derivada.f1 porque se ejecuta el método f1 de Derivada
Derivada.f2 porque se ejecuta el método f2 de Derivada
Derivada.f1 porque el método f2 de Derivada llama al método f1 de Derivada
Base.f1 porque se ejecuta el método f1 de Base (aunque se supone que debiera ejecutarse
el método de la clase Derivada esto sucede porque al método f1 no se le antepone el prefijo
virtual que hace que se ejecute el método de la clase derivada antes que el de la base)
Derivada.f2 porque se ejecuta el método f2 de Derivada debido a que se antepuso el prefijo
virtual en el método
Derivada.f1 el método f2 de Derivada llama al método f1 de Derivada
```

- Dentro del siguiente código:

```
int main(int argc, char* argv[]) {
    unsigned int* a; //a
    static float b[4]; //b
    static char c; //c
    return 0;
}
```

Defina: a) Un puntero a entero sin signo a alocarse en el stack b) Un arreglo para albergar 4 números de punto flotante que se aloque en el data segment c) Un carácter a alocarse en el data segment

- ¿Cuál es el motivo por el cual las clases que utilizan templates se declaran y definen en los .h?

Ya explicado

- ¿Cuál es el uso de la función listen? ¿Qué parámetros tiene y para qué sirven?

La función listen define hasta cuántas conexiones en espera de ser escuchadas puede guardar el SO. No define un límite de las conexiones totales (es decir, en espera y las que están ya aceptadas).

Firma de la función listen: `int listen(int sockfd, int backlog);`

El parámetro `sockfd` es el socket en cuestión y el parámetro `backlog` es esta cantidad máxima de conexiones en espera de ser escuchadas. Devuelve un int que indica si hubo un error: devuelve 0 en caso de éxito y -1

en caso de error. Los errores pueden ser: algún socket ya está escuchando en ese puerto, el file descriptor del socket es inválido, ese socket no bindeó previamente, el file descriptor del socket no es el de un socket o no permite la operación listen().

- La clase *Oracion* utiliza un `char* a` para almacenar un string terminado en `\0`. Escriba la declaración de esta clase no olvidando: constructor por default, constructor por copia, operadores +, -, =, >> y <<. Implemente el operador = y el operador -. Este último debe eliminar de la primera cadena todas las ocurrencias de la segunda.

```
class Oracion {
private:
    const char* a;
public:
    Oracion();
    explicit Oracion(const char* a) {
        this->a = a;
    }
    Oracion(const Oracion& other); //constructor por copia
    Oracion operator+(const Oracion& other) const;
    Oracion operator-(const Oracion& other) const {
        std::string a_bis = std::string(this->a);
        std::string other_bis = std::string(other.a);
        size_t found;
        while ((found = a_bis.find(other_bis)) != std::string::npos) {
            a_bis.replace(found, other_bis.length(), "");
        }
        Oracion result(a_bis.c_str());
        return result;
    }
    Oracion& operator=(const Oracion& other) {
        if (this != & other) {
            this->a = other.a;
        }
        return *this;
    }
    friend std::ostream& operator<<(std::ostream& out, const Oracion& other);
    friend std::istream& operator>>(std::istream& in, Oracion& other);
};
```

- Escriba un programa que reciba paquetes de 10 bytes por el puerto TCP 815 y los imprima por pantalla. Al recibir el byte 0xff debe cerrarse ordenadamente. No considere errores.

```
#define CONEX_EN_COLA 50
#define MAX_PAQUETE 10

int main(int argc, char* argv[]) {
    const char* port = "815";

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
```

```
    getaddrinfo(NULL, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                     ptr->ai_socktype,
                     ptr->ai_protocol);
        if (skt != -1) break;
    }

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    unsigned short int buffer[MAX_PAQUETE];
    size_t leido = 0;
    int peerskt = accept(skt, NULL, NULL);
    size_t pos = 0;

    while (true) {
        //4) Recibir
        leido = recv(peerskt, &buffer[pos], MAX_PAQUETE, 0);
        pos += leido;
        if (pos != 10) continue;
        printf("%d\n", *buffer);
        if (*buffer == 0xff) break;
        memset(buffer, 0, 10);
        pos = 0;
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    close(peerskt);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    freeaddrinfo(ptr);
    return 0;
}
```

- ¿Qué es un deadlock? Ejemplifique

Ya explicado

Final 1/3/2018

- Indique 2 posibles usos del modificador const. Utilice en cada caso una breve sentencia de código que lo emplee junto con la explicación del significado/consecuencia de su uso.

a) ***Valores constantes***

Especifica que el valor de una variable es constante y le dice al compilador que no deje que el programador cambie su valor.

Por ejemplo: `const int i = 5` no puede ser modificado posteriormente.

Así mismo también existen punteros constantes y punteros a valores constantes.

Los *punteros constantes* no permiten modificar el puntero en sí mismo pero sí el contenido al que apuntan:

```
int a = 10;
int* const ptr = &a;
*ptr = 5; //se puede
ptr++; //no se puede
```

Los *punteros a valores constantes* no permiten modificar el contenido al que apuntan los punteros pero sí los punteros:

```
const int a = 10;
const int* ptr = &a;
*ptr = 5; //no se puede
ptr++; //se puede
```

En conclusión: `const int* ptr;` //ptr es un puntero a un int constante

`int* const ptr;` //ptr es un puntero constante a un int

b) ***Métodos constantes***

Indica que la función es de solo lectura y que no modifica el estado interno del objecto que la llamó. No puede llamar ningún método que no sea constante ni modificar atributos no estáticos. Se agrega el modificador `const()` al final de la declaración y debe incluirse además en la definición. Sirve para detectar errores en el código en tiempo de compilación ya que si un método no modifica el estado interno del estado se utiliza esta keyword para que el compilador falle en caso de que sí se modifique y exista un bug a resolver.

Por ejemplo:

```
class Contenedor {
    std::string a;

public:
    Contenedor(std::string& a) : a(a) {}
    std::string obtenerContenido() const {
        return a;
    } // No se debe modificar el contenido de a, solo devolver su valor
}
```

- Escriba una rutina que procese un archivo indicado por parámetro sobre sí mismo eliminando todas las ocurrencias de caracteres repetidos. El procesamiento consiste en detectar toda sucesión de dos o más carácteres idénticos y reemplazarlos por una única ocurrencia.

La idea de este tipo de ejercicios es ir leyendo y escribiendo a la vez, cuando se detecta el fragmento a recortar se mueve el puntero para atras y se escribe todo lo que sigue. Después con la función `truncate` se termina borrando el contenido del archivo que quedó repetido por esta vuelta hacia atrás.

```
int leer(FILE* f, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    int out = fgetc(f);
    *seek = ftell(f);
    return out;
}

void escribir(FILE* f, char c, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    fputc(c, f);
    *seek = ftell(f);
```

```
}

int main(int argc, char* argv[]) {
    FILE *f = fopen(argv[1], "r+");
    int c = 0;
    long int c_final = 0;
    long int seek_lectura = 0;
    long int seek_escritura = 0;
    int cant_ocurrencias = 1;
    int c_previo = 0;

    char* file_name = argv[1];

    f = fopen(file_name, "r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;
        if (c == c_previo) {
            cant_ocurrencias++;
        } else {
            if (cant_ocurrencias >= 2) {
                c_final -= cant_ocurrencias;
                seek_escritura = c_final-1;
                escribir(f, c_previo, &seek_escritura);
                c_final++;

            }
            escribir(f, c, &seek_escritura);
            cant_ocurrencias = 1;
        }
        c_previo = c;
    }
    if (cant_ocurrencias >= 2) c_final -= (cant_ocurrencias - 1);
    fclose(f);
    truncate(file_name, sizeof(char) * c_final);
    return 0;
}
```

- Explique cómo funciona la herencia pública en C++ y en qué se diferencia de la herencia privada

La **herencia** sirve para crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente, evitando el rediseño, la modificación y verificación de la parte ya implementada. Facilita la creación de objetos de otros ya existentes e implica que una subclase obtiene todo el comportamiento y eventualmente los atributos de la super clase.

Existen 3 tipos de herencia: pública, protegida y privada:

1. **Pública**: los atributos y métodos públicos y protected se mantienen con la misma visibilidad que los de su clase padre
2. **Protegida**: los atributos/métodos públicos y protected de la clase padre pasan a ser protegidos en la clase hijo. Entonces el único que sabe que hereda del padre es el hijo y los que hereden de él.
3. **Privada**: los atributos/métodos públicos y protected de la clase padre pasan a ser privados en la propia clase (clase hijo). Entonces el único que sabe que hereda del padre es el hijo. Las clases que heredan de la clase hija no tienen los atributos y métodos de la clase padre.

Además, los métodos también pueden ser públicos, protegidos y privados:

1. Los atributos/métodos públicos pueden ser accedidos desde cualquier clase
2. Los atributos/métodos protected pueden ser accedidos en la clase donde se definen y en sus descendientes
3. Los atributos/métodos privados sólo pueden ser accedidos en la clase donde se definen

- Escriba un pequeño programa de interfaz gráfica que lance una aplicación de ventana simple, con un único botón. El botón debe tener el texto 'Cerrar' que al ser presionado oculta la ventana y cierra el sistema.

```
#include <gtkmm.h>

class BotonCerrar: public Gtk::Button {
    public:
        BotonCerrar(Gtk::Entry* entry) : Gtk::Button("Cerrar") {
            this->entry = entry;
        }
    private:
        Gtk::Entry* entry;
        void on_clicked() {
            Gtk::Main::quit();
        }
};

int main(int argc, char* argv[]) {
    Gtk::Main kit(argc, argv);
    Gtk::Entry entry;
    BotonCerrar boton(&entry);
    Gtk::VBox vbox;
    vbox.add(boton);
    Gtk::Window v;
    v.add(vbox);
    v.show_all();

    Gtk::Main::run(v);
    return 0;
}
```

- Escriba una función de C llamada *obtenerPalabrasCortas* que reciba dos parámetros: un puntero a carácter constante (S) y un entero sin signo (L). La función recibirá una oración en S que debe ser recorrida detectando palabras y copiando aquellas cuya longitud sea menor que L en una lista de resultados. Considere el carácter espacio (`" "`) como delimitador entre palabras. Explique brevemente la estructura de retorno elegida para almacenar los resultados.

Voy a usar una lista de C++ porque fue lo más simple para probar pero supongo que habría usar un TDA lista y explicar que es la mejor alternativa porque no hay que saber de antemano la cantidad de palabras que cumplen con la condición de que su longitud es menor que L.

```
std::list<char*> obtenerPalabrasCortas(const char* S, unsigned int L) {
    std::list<char*> resultado;
    size_t pos = 0;
    char container[L];

    for (int i = 0; i < strlen(S) + 1; i++) {
        container[pos] = S[i];
```

```
        if (S[i] == ' ' || S[i] == '\0') {
            container[pos] = '\0';
            if (pos+1 < L) {
                char* to_add = (char*) malloc(sizeof(char) * L);
                memcpy(to_add, container, pos+1);
                pos--;
                resultado.push_back(to_add);
            }
            memset(container, 0, L);
            pos = 0;
        }
        else pos++;
    }
    return resultado;
}
```

- Defina una macro llamada *LOG* que reciba una cadena como parámetro y la imprima utilizando printf. La macro debe ejecutar el comando printf solamente cuando se detecta que la macro *LOG_ENABLED* se encuentra definida.

```
#define LOG(char* a)
#ifdef LOG_ENABLED
printf("%s", a);
#endif
```

- Defina un operador global == que toma un vector STL de tipo genérico T y un arreglo de elementos T. El operador debe retornar un bool indicando si las colecciones son iguales, es decir, si todos los elementos de T coinciden en ambas colecciones.

```
template<class T>
bool std::operator==(const std::list<T>& ls, T array []) {
    std::list<T>::iterator it;
    int pos = 0;
    for (it = ls.begin(); it != ls.end(); ++it) {
        if (array[pos] != *it) return false;
        pos++;
    }
    return true;
}
```

- Explique el concepto de mutex. Escriba un ejemplo simple de código donde su uso resulte fundamental para garantizar el correcto empleo de recursos en un contexto concurrente.

Ya explicado

- Escriba una función que reciba un parámetro PUERTO, cree un socket y acepte conexiones entrantes indefinidamente hasta detectar un error en la aceptación. A cada cliente conectado se debe transmitir la cadena “Conexión rechazada” y, acto seguido, cerrar su conexión

```
int main(int argc, char* argv[]) {
    char* port = (char*) "8080";
    Socket socket;
    socket.bind(port);
    socket.listen();

    std::vector<Socket> clients;

    while (true) {
```

```
        try {
            std::thread cliente([&]() {
                Socket peerskt = socket.accept();
                clients.push_back(std::move(peerskt));
            });
            cliente.join();
        }
        catch (...) {
            for (auto i = clients.begin(); i != clients.end(); i++) {
                char* rechazo = "Conexion Rechazada";
                i->send(rechazo, strlen(rechazo));
                i->shutdown();
            }
            break;
        }
    }
    return 0;
}

int main(int argc, char* argv[]) {
    char* port = (char*) "8080";

    struct addrinfo hints;
    struct addrinfo* ptr;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;
    getaddrinfo(NULL, port, &hints, &ptr);

    int skt;
    for(; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
        if (skt != -1) break;
    }
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);
    listen(skt, 50);

    std::vector<int> clients;

    bool closed = false;

    while (! closed) {
        std::thread t1([&]() {
            int peerskt = accept(skt, NULL, NULL);
            if (peerskt == -1) {
                for (auto i = clients.begin(); i != clients.end(); i++) {
                    char* rechazo = "Conexion rechazada";
                    send(*i, rechazo, strlen(rechazo), MSG_NOSIGNAL);
                    shutdown(*i, SHUT_RDWR);
                    close(*i);
                }
                closed = true;
            } else {
```

```
                clients.push_back(peerskt);
            }
        });
        t1.join();
    }
    return 0;
}
```

- Escriba:

a) La definición de una función de alcance local que toma como parámetros un puntero a entero con signo y un puntero a carácter con signo; sin retorno.

```
static void function (int* a, char* b) {}
```

b) La declaración de un puntero a función que respete la firma de la función anterior.

```
static void(*function)(int*, char*);
```

c) La declaración de una variable global entera que se encuentra en otra unidad de compilación.

`extern int a;` Wikipedia dice: "external variables are globally accessible"

Final 15/2/2018

- Explique qué usos tiene la palabra reservada static en C++. De un breve ejemplo de uno de ellos.

La palabra **static** en C++ tiene 4 usos:

1. Variable static en funciones
2. Objetos de clase static
3. Miembros de clase static
4. Métodos de clase static

Variables static en funciones

Las variables static cuando son utilizadas dentro de las funciones son inicializadas sólo una vez y guardan su valor a través de las llamadas a funciones. Se guardan en el data segment:

```
void counter() {
    static int count = 0;
    std::cout << count++;
}

int main() {
    for (int i = 0; i < 5; i++) {
        counter();
    }
}
```

Se imprime: `0, 1, 2, 3 y 4`. Si no se hubiese usado el modificador *static* hubiese impreso 0 en todas las llamadas ya que se reinicializa siempre que se llama a *counter()* y se destruye cuando termina la función. Al ser *static* tiene un scope hasta el final de la función *counter()* y lleva su valor mediante las llamadas a función. Sin embargo, su *lifetime* es local al archivo, es decir, que se destruye al finalizar la función main. Si no se hubiese inicializado el valor default es 0, no es basura.

Objetos de clase static

Se guardan en el data segment y tiene como lifetime hasta el final del programa. Usan los constructores de manera normal y no tienen un valor por default en caso de ser una clase definida por el usuario.

```
class Example {
    public:
        Example() {
            std::cout << "constructor" << std::endl;
        }
        ~Example() {
            std::cout << "destructor" << std::endl;
        }
};

int main() {
    int x = 0;
    if (x == 0) {
        static Example obj;
    }
    std::cout << "end" << std::endl;
}
```

Se observa que se imprime `constructor END destructor`. Se verifica que el objeto se destruye cuando sale del scope de main y no del scope de creación, como sucede generalmente con los objetos que no se les incluye el modificador *static*.

Miembros de clase static

Son atributos que son compartidos por todos los objetos. Se almacena solo una vez y no puede copiarse entre objetos. No se inicializan usando un constructor pues no dependen de la inicialización de los objetos. Se debe inicializar explícitamente fuera de la clase, sino se produce un error de linkeo.

```
class X {
    public:
        static int i;
        X() {}
};

int X::i=1;
int main() {
    X obj;
    std::cout << obj.i; //imprime el valor 1
}
```

Una vez que se define un miembro de clase static no se puede reedefinir pero se le pueden aplicar operaciones aritméticas.

Métodos de clase static

Estas funciones no pueden acceder a los atributos y las funciones sino solamente a los miembros de clase estáticos y funciones estáticas. No se puede usar el operador this.

Se puede llamar usando el nombre de la clase y :: o en una instancia en particular y el operador ..

```
class X {
    public:
        static void f() {}
};

int main() {
```

```
    X::f();
    X object;
    object.f();
}
```

- Ejemplifique los distintos tipos de herencia que admite C++. De un breve ejemplo de al menos una de ellas.

Ya explicado

- Escriba una función C que reciba 1 cadena (S) y 1 entero (N) y un caracter (C); y retorne una nueva cadena cuyo contenido sea N veces la cadena S utilizando el caracter C como separador.

```
char* C(char* S, int N, char C) {
    size_t len = N*strlen(S) + N - 1 + 1;
    char* new_S = (char*) malloc(sizeof(char) * len);
    int offset = 0;
    for (size_t i = 0; i < N - 1; i++) {
        memcpy(&new_S[offset], S, strlen(S));
        new_S[offset + strlen(S)] = C;
        offset += strlen(S) + 1;
    }
    memcpy(&new_S[offset], S, strlen(S));
    new_S[offset + strlen(S)] = 0;
    return new_S;
}
```

- Escriba las siguientes definiciones/declaraciones:

a) Una declaración de una función llamada F que tome un arreglo de flotantes y devuelva un puntero a un arreglo de enteros cortos sin signo.

```
unsigned short* F(float B[]);
```

b) Una definición de una variable entero largo con signo, global, visible fuera del fuente, llamada G.

```
extern long G = 5;
```

c) Una definición de un puntero a un arreglo de 10 caracteres sin signo, llamado C.

```
unsigned char (*C)[10] = NULL;
```

- Escriba y ejemplifique el uso de la siguiente instrucción de precompilación: `#ifdef`

El código entre `ifdef/endif` o `ifned/endif` el preprocesador lo saca o lo deja dependiendo de la condición. `Ifdef` significa "si lo que sigue está definido" mientras que ifndef significa "si lo que sigue no está definido".

De esta manera:

```
#define one 0
#ifdef one
printf("one is defined ");
#endif
#ifndef one
printf("one is not defined ");
#endif
```

Es equivalente a `printf("one is defined")` debido a que "one" está definido por lo que la condición ifdef es verdadera e ifndef es falsa.

La macro ifndef se utiliza para no caer en definiciones circulares en los headers. Esto es porque la sintaxis a seguir es:

```
#IFNDEF _ALGO_H_
#DEFINE ALGO_H
```

Por lo que crea el símbolo en caso de que no esté definido. De esta manera, en una primera corrida solo se crea el símbolo la primera vez y el resto de las veces no. En caso de no utilizar esta sintaxis se cae en definiciones circulares ya que estaría definiéndolo en cada #include que se haga del header.

- Defina una rutina que se conecte al puerto 9000 y acepte conexiones de forma secuencial. Al aceptar una conexión, se escuchan los caracteres enviados por el cliente y se cierra la conexión. Si el cliente envia 'Q', se dejan de recibir conexiones y finaliza el programa.

```
#define CONEX_EN_COLA 50
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    const char* port = argv[1];

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(NULL, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                     ptr->ai_socktype,
                     ptr->ai_protocol);
        if (skt != -1) break;
    }

    int val = 1;
    setsockopt(skt, SOL_SOCKET,
                        SO_REUSEADDR, &val, sizeof(val));

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    char buffer[MAX_PAQUETE];
    size_t leido = 0;
    size_t pos = 0;

    while (true) {
        int peerskt = accept(skt, NULL, NULL);

        //4) Recibir
        while (true) {
```

```
            leido = recv(peerskt, &buffer[pos], MAX_PAQUETE, 0);
            pos += leido;
            if (buffer[pos-1] == '\0') break;
        }
        if (strcmp(buffer, "q") == 0) break;
        pos = 0;
        printf("%s\n", &buffer[0]);

        shutdown(peerskt, SHUT_RDWR);
        close(peerskt);
    }
    //5) Liberar
    shutdown(skt, SHUT_RDWR);
    close(skt);
    freeaddrinfo(ptr);
    return 0;
}
```

- Dado un vector de enteros en C++, escriba un procedimiento que reciba el vector, lo recorra y genere un nuevo vector con las potencias de 2 de cada uno de los items del vector original. Para optimizar los recursos de computación, se pide que se utilicen al menos 2 hilos para realizar el procedimiento. ¿Es necesario realizar un control de concurrencia entre dichos hilos?

Es necesario realizar un control de concurrencia entre dichos hilos para que todos los elementos del vector se recorran sólo 1 vez y no se inserten elementos repetidos en la lista.

```
class MonitorVector {
private:
    std::mutex mx_contador;
    std::condition_variable cv_par, cv_impar;
    bool es_impar = true;
    std::list<int>& elementos;
    std::list<int>& resultado;
    std::list<int>::iterator it;

public:
    MonitorVector(std::list<int>& elementos,
            std::list<int>& resultado): elementos(elementos),
            resultado(resultado) {
        it = elementos.begin();
    }

    void potencias_par() {
        std::unique_lock<std::mutex> lock(mx_contador);
        while (es_impar) {
            cv_par.wait(lock);
        }
        int element = *it;
        it++;
        resultado.push_back(pow(2, element));
        es_impar = true;
        cv_impar.notify_one();
    }

    void potencias_impar() {
```

```
        std::unique_lock<std::mutex> lock(mx_contador);
        while (! es_impar) {
            cv_impar.wait(lock);
        }
        int element = *it;
        it++;
        resultado.push_back(pow(2, element));
        es_impar = false;
        cv_par.notify_one();
    }
};

int main() {
    std::list<int> elementos = {1, 2, 3, 4};
    std::list<int> resultado;

    MonitorVector monitor(elementos, resultado);

    std::thread t1([&]() {
        for (int i = 0; i < elementos.size()/2; i++) {
            monitor.potencias_par();
        }
    });
    std::thread t2([&]() {
        for (int i = 0; i < elementos.size()/2; i++) {
            monitor.potencias_impar();
        }
    });
    t1.join();
    t2.join();
    return 0;
}
```

- Escriba una rutina (para ambiente gráfico Windows o Linux) que dibuje la siguiente imagen en su área de dibujo (un circulo azul en un rectangulo negro que lo contiene).

```
class MyCircle : public Gtk::DrawingArea {
protected:
    // Override default signal handler:
    bool on_draw(const Cairo::RefPtr<Cairo::Context>& context) override{
        Gtk::Allocation allocation = this->get_allocation();
        int center_x = allocation.get_width() / 2;
        int center_y = allocation.get_height() / 2;
        int radius = allocation.get_height() / 2;
        int start_angle = 0;
        double end_angle = 2 * M_PI;
        context->arc(center_x, center_y, radius, start_angle, end_angle);

        context->set_source_rgb(0.0, 0.0, 1.0); //azul

        context->fill();
        return true;
    }
};
```

```
int main(int argc, char **argv) {
    auto app = Gtk::Application::create();
    Gtk::Window window;

    MyCircle circle;
    window.add(circle);
    circle.show();
    return app->run(window);
}
```

- Escribir un programa C que reciba por argumento el nombre de un archivo y lo procese sobre sí mismo (sin crear archivos intermedios). El procesamiento consiste en eliminar todas las ocurrencias de la cadena "ABCD". Por ejemplo: la cadena `01 23 AABCDD 45` debe ser reemplazada por `01 23 AD 45`.

```
int leer(FILE* f, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    int out = fgetc(f);
    *seek = ftell(f);
    return out;
}

void escribir(FILE* f, char c, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    fputc(c, f);
    *seek = ftell(f);
}

int main(int argc, char* argv[]) {
    FILE *f;
    char* file_name = argv[1];
    int c = 0;
    long int c_final = 0;
    long int seek_lectura = 0;
    long int seek_escritura = 0;

    char to_compare[] = "ABCD";

    char buffer[5];
    memset(buffer, 0, 5);

    bool inicio = true;
    int pos = 0;

    f = fopen(file_name, "r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;
        if (inicio) {
            buffer[pos] = c;
            pos++;
            if (buffer[3] != '\0') inicio = false;
            escribir(f, c, &seek_escritura);
            continue;
        }
```

```
        for (int i = 1; i <= 4; i++) {
            buffer[i-1] = buffer[i];
        }
        buffer[3] = c;

        if (strcmp(to_compare, buffer) == 0) {
            c_final -= 4; //Saque 4 letras
            seek_escritura = c_final;
        } else {
            escribir(f, c, &seek_escritura);
        }
    }
    fclose(f);
    truncate(file_name, sizeof(char) * c_final);
    return 0;
}
```

- Defina la clase *Documento* que encapsule una cadena de caracteres con el número de documento propiamente dicho y otra cadena con el tipo ('Pasaporte','DNI', etc). La clase debe incluir: constructor default, constructor por copia, operadores ==, <<, =.

```
class Documento {
private:
    std::string number;
    std::string type;
    Documento() {}
    Documento(std::string number, std::string type) {
    this->number = number;
    this->type = type;
    }
    Documento(const Document& other) {
        this->number = other.number;
        this->type = other.type;
    }
    bool operator==(const Documento& other) const {
        return this->number == other.number && this->type == other.type;
    }
    Documento& operator=(const Documento& other) {
        if (this != &other) {
            this->number == other.number;
            this->type == other.type;
        }
        return *this;
    }
    std::ostream& operator<<(std::ostream& out, const Document& a) {
        out << "number: " << a.number << " type: " << a.type << std::endl;
        return out;
    }
};
```

Final 8/2/2018

- Explique el uso de valores por defecto en templates de C++. Escriba un ejemplo.

Los parámetros en templates pueden tener argumentos por defecto. El set de los argumentos por defecto se acumula en todas las declaraciones de un template dado. El siguiente ejemplo lo demuestra:

```
template<class T, class U = int> class A;
template<class T = float, class U> class A;
template<class T, class U> class A {
    public:
        T x;
        U y;
};
A<> a;
```

El tipo del atributo `a.x` es float y del atributo `a.y` es int

- Explique cómo se comporta el modificador virtual en C++. De un breve ejemplo dónde su uso sea imprescindible

Ya explicado

- Explique en qué consiste el proceso de transformación de código fuente a código ejecutable. Haga referencia a las etapas y los chequeos que se hacen en cada una de ellas.

Ya explicado

- Escriba una macro llamada *ASSERT*. La macro debe recibir como parámetro una condición de código a validar y un mensaje de error a imprimir por la salida estándar en caso que la condición no se cumpla. Al imprimir el mensaje, se debe incluir el nombre del archivo C. Ejemplo de uso: ASSERT(div > 0, "Imposible dividir por 0")

```
#define ASSERT(condition, error) {\
    if (!condition) {\
        std::cout << error << std::endl;\
        std::cout << "Assertion failed at << __FILE__ << std::endl;\
    }\
}
```

- Implemente un servidor que escuche en el puerto 5000. El servidor debe aceptar conexiones (de 1 a la vez) escribiendo el contenido recibido en un archivo (001.txt, 002.txt, etc) y sin responder nada. Cada conexión debe ser cerrada por el servidor al recibir la palabra "FIN". Luego de una conexión sin contenido (se recibe sólo la palabra "FIN"), el servidor debe cerrarse ordenadamente.

Basicamente se pide un servidor Pasos a seguir:

- Bind
- Listen
- Accept
- Recibir
- Liberar

(Se que no es exactamente lo que piden pero no entiendo bien la consigna)

```
#define CONEX_EN_COLA 50
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    const char* port = "5000";

    //1) Bind
```

```
struct addrinfo hints;
struct addrinfo *ptr;
struct addrinfo *res = NULL;

memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
getaddrinfo(NULL, port, &hints, &ptr);

int skt;
for (; ptr != NULL; ptr = ptr->ai_next) {
    skt = socket(ptr->ai_family,
            ptr->ai_socktype,
            ptr->ai_protocol);
    if (skt != -1) break;
}

//Bind del socket
bind(skt, ptr->ai_addr, ptr->ai_addrlen);

//2) Listen
listen(skt, CONEX_EN_COLA);

//3) Accept
char paquete[MAX_PAQUETE + 1];
char buffer[MAX_PAQUETE];
size_t pos_paquete = 0;
size_t leido = 0;
bool salir = false;

int cant_conexiones = 1;

while (! salir) {
    //4) Recibir
    int peerskt = accept(skt, NULL, NULL);
    leido = recv(peerskt, buffer, MAX_PAQUETE, 0);
    for (size_t i = 0; i < leido; i++) {
        paquete[pos_paquete] = buffer[i];
        pos_paquete++;
    }
    paquete[leido] = '\0';

    if (strcmp(paquete, "FIN") == 0) salir = true;
    char file_name[64];
    snprintf(file_name, 64, "00%d.txt", cant_conexiones);

    FILE* write_file = fopen(file_name, "w");
    fprintf(write_file, "%s", paquete);
    pos_paquete = 0;
    cant_conexiones++;

    shutdown(peerskt, SHUT_RDWR);
    close(peerskt);
```

```
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    close(peerskt);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    freeaddrinfo(ptr);
    return 0;
}
```

- Escribir un programa en C que reciba por argumento el nombre de un archivo de texto y lo procese sobre sí mismo (sin crear archivos intermedios). El procesamiento consiste en reemplazar todas las ocurrencias de '1', '2' y '3' (con un separador antes y uno después) por 'uno', 'dos' y 'tres', respectivamente.

Lo del separador no lo entendí. Yo leo un '1' y lo cambio por 'uno'.

```
int leer_reversa(FILE* f, int* seek) {
    int previous_seek = *seek;
    fseek(f, *seek, SEEK_END);
    int out = fgetc(f);
    *seek = previous_seek - 1;
    return out;
}

void escribir_reversa(FILE* f, char c, int* seek) {
    int previous_seek = *seek;
    fseek(f, *seek, SEEK_END);
    fputc(c, f);
    *seek = previous_seek - 1;
}

int main(int argc, char* argv[]) {
    FILE *f;
    int c = 0;
    long int c_final = 0;
    int seek_lectura = -1;
    int seek_escritura = -1;

    f = fopen(argv[1], "r");

    int cant_extra = 0;

    while ((c = fgetc(f)) != EOF) {
        if (c == '1' || c == '2') {
            c_final += 2;
            cant_extra += 2;
        }
        if (c == '3') {
            c_final += 3;
            cant_extra += 3;
        }
        c_final++;
    }
    fclose(f);
    truncate(argv[1], sizeof(char) * c_final);
```

```
    f = fopen(argv[1], "r+");

    seek_lectura -= (cant_extra);

    while (seek_lectura != (-c_final - 1)) {
        c = leer_reversa(f, &seek_lectura);
        if (c == '1') {
            char* reemplazo = "uno";
            for (int i = strlen(reemplazo) - 1 ; i >= 0; i--) {
                escribir_reversa(f, reemplazo[i], &seek_escritura);
            }
        } else if (c == '2') {
            char* reemplazo = "dos";
            for (int i = strlen(reemplazo) - 1 ; i >= 0; i--) {
                escribir_reversa(f, reemplazo[i], &seek_escritura);
            }
        } else if (c == '3') {
            char* reemplazo = "tres";
            for (int i = strlen(reemplazo) - 1 ; i >= 0; i--) {
                escribir_reversa(f, reemplazo[i], &seek_escritura);
            }
        } else {
            escribir_reversa(f, c, &seek_escritura);
        }
    }
    fclose(f);
    return 0;
}
```

- Escriba una rutina para ambiente gráfico que dibuje un rombo rojo, centrado que ocupe todo el espacio de ventana.

```
class MyRombo : public Gtk::DrawingArea {
protected:
    // Override default signal handler:
    bool on_draw(const Cairo::RefPtr<Cairo::Context>& context) override{
        Gtk::Allocation allocation = this->get_allocation();
        int window_width = allocation.get_width();
        int window_height = allocation.get_height();
        context->move_to(window_width/2, 0);
        context->line_to(0, window_height/2);
        context->line_to(window_width/2, window_height);
        context->line_to(window_width, window_height/2);
        context->set_source_rgb(1.0, 0.0, 0.0); //rojo
        context->fill();
        return true;
    }
};

int main(int argc, char **argv) {
    auto app = Gtk::Application::create();
    Gtk::Window window;
    MyRombo rombo;
    window.add(rombo);
```

```
    return app->run(window);
}
```

- Declare la clase *Pagina* capaz de almacenar una acumulación de líneas de texto (strings) en cierto orden. Incluya las operaciones de agregación y eliminación de líneas. La clase debe poseer los operadores usuales de copia, asignación, salida a flujo (<<), comparación (==), agregación (+), eliminación (-). Implemente este último operador.

Ya explicado

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `int (*I)[3];`

Se declara un puntero a un arreglo "I" de 3 enteros con signo.

b) `static char* C[2];`

Se declara un arreglo "C" de 2 punteros a carácter con signo de alcance local.

c) `extern char**C[2];`

Se declara un arreglo "C" de 2 punteros a punteros a carácter con signo, definido en un archivo externo.

- ¿Qué función se utiliza para lanzar un thread? Ejemplifique mediante código

Ya explicado

Final 14/12/2017

- ¿Qué significa que una función es bloqueante? ¿Cómo subsanaría esa limitación en término de mantener el programa "vivo"?

Ya explicado

- Escribir un programa C que procese el archivo "numeros.txt" sobre sí mismo (sin crear archivos intermedios y sin subir el archivo a memoria). El procesamiento consiste en leer grupos de 5 digitos y reemplazarlos por 4 caracteres hexadecimales que representen el mismo número leído pero en hexadecimal.

Asumo que en el archivo son todos digitos y que pretenden reemplazar los primeros 5 por 4 hexadecimales, luego los segundos 5 por 4 hexadecimales y así.

```
int leer(FILE* f, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    int out = fgetc(f);
    *seek = ftell(f);
    return out;
}

void escribir(FILE* f, char c, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    fputc(c, f);
    *seek = ftell(f);
}

int main(int argc, char* argv[]) {
    FILE *f;
    char* file_name = argv[1];
    int c = 0;
    long int c_final = 0;
```

```
    long int seek_lectura = 0;
    long int seek_escritura = 0;

    char buffer[6];
    memset(buffer, 0, 6);

    char to_hex[5];
    memset(buffer, 0, 5);

    bool inicio = true;
    int pos = 0;

    f = fopen(file_name, "r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;
        if (inicio) {
            buffer[pos] = c;
            if (pos == 4) inicio = false;
            pos++;
            if (inicio) continue;
        }

        int value = atoi(buffer);
        snprintf(to_hex, 5, "%x", value);

        for (int i = 0; i < strlen(to_hex); i++) {
            escribir(f, to_hex[i], &seek_escritura);
        }
        c_final--; //Cambie 5 caracteres por 4

        // Reset:
        inicio = true;
        memset(to_hex, 0, 5);
        memset(buffer, 0, 6);
        pos = 0;
    }

    int value = atoi(buffer);
    snprintf(to_hex, 5, "%x", value);

    for (int i = 0; i < strlen(to_hex); i++) {
        escribir(f, to_hex[i], &seek_escritura);
    }

    fclose(f);
    truncate(file_name, sizeof(char) * c_final);
    return 0;
}
```

- Escriba una rutina (para ambiente gráfico Windows o Linux) que dibuje un círculo verde del tamaño de una ventana.

```
class MyCircle : public Gtk::DrawingArea {
protected:
```

```
    // Override default signal handler:
    bool on_draw(const Cairo::RefPtr<Cairo::Context>& context) override{
        Gtk::Allocation allocation = this->get_allocation();
        int center_x = allocation.get_width() / 2;
        int center_y = allocation.get_height() / 2;
        int radius = allocation.get_height() / 2;
        int start_angle = 0;
        double end_angle = 2 * M_PI;
        context->arc(center_x, center_y, radius, start_angle, end_angle);

        context->set_source_rgb(0.0, 1.0, 0.0); //verde

        context->fill();
        return true;
    }
};

int main(int argc, char **argv) {
    auto app = Gtk::Application::create();
    Gtk::Window window;

    MyCircle circle;
    window.add(circle);
    circle.show();
    return app->run(window);
}
```

- Declare la clase *TELEFONO* para encapsular una cadena numérica correspondiente a un teléfono. Incluya al menos: Constructor(área, numero), constructor default y constructor de copia, operador <<, operador ==, =, long y >>. Implemente el operador >>.

```
class Telefono {
private:
    std::string telefono;
public:
    Telefono(int área, int numero);
    Telefono(); //constructor default
    Telefono(const Telefono& other); //constructor por copia
    bool operator==(const Telefono& other) const;
    Telefono& operator=(const Telefono& other);
    friend std::ostream& operator<<(std::ostream& out, const Telefono& other);
    friend std::istream& operator>>(std::istream& in, Telefono& other) {
        in >> other.telefono;
        return in;
    }
    operator long() const;
```

- Explique qué son los métodos virtuales y para qué sirven. De un breve ejemplo donde su uso sea imprescindible.

Ya explicado

- Explique qué se entiende por "macro". Ejemplifique mediante código

Ya explicado

- ¿Qué función utiliza para esperar la terminación de un thread? Ejemplifique mediante código

Ya explicado

- Escriba un programa C que tome 2 cadenas por línea de comandos: A y B; e imprima la cadena A después de haber suprimido todas las ocurrencias de B.

Ej: `reemp.exe "El final no está aprobado" "no" ---> El final está aprobado`

No llego a hacerlo con el tiempo

- ¿Qué es el polimorfismo? Ejemplifique mediante código

El **polimorfismo** permite definir distintos comportamientos para un método dependiendo de la clase sobre la que se realice la implementación. El caso más común de polimorfismo es el de polimorfismo por herencia. Por ejemplo si se tiene una clase padre de la cual heredan varias clases y esta define un método que será luego redefinido por alguno de esas clases, cada una tendrá su propio comportamiento y se evidenciará el polimorfismo. La principal ventaja del polimorfismo es que permite generalizar las clases a partir de la clase padre y aplicar el método en común a todas, simplificando el código pero manteniendo la individualidad de los métodos en su ejecucion.

Por ejemplo:

```
class Animal:
private:
    int vida;
public:
    Animal() {
        this->vida = 0;
    }
    virtual ~Animal();
    virtual void comer();
};
class Gato: public Animal {
public:
    Gato();
    virtual ~Gato();
    virtual void comer() override {
        this->vida += 5;
    }
};
class Perro: public Animal {
public:
    Perro();
    virtual ~Perro();
    virtual void comer() override {
        this->vida += 10;
    }
};
int main() {
    Animal* perro = new Perro();
    Animal* gato = new Gato();
    std::vector<Animal*> animales;
    animales.push_back(perro);
    animales.push_back(gato);
    for (auto i = animales.begin(); i != animales.end(); i++) {
        *i->comer();
    }
}
```

- Escriba las siguientes definiciones/declaraciones:

a) Declaración de un puntero a puntero a carácter

```
char** a;
```

b) Definición de una función RESTA que tome dos enteros largos con signo y devuelva su resta. Esta función sólo debe ser visible en el módulo donde se la define.

```
static long resta(long a, long b) {
    return a - b;
}
```

c) Definición de un entero corto sin signo solamente visible en el módulo donde se la define.

```
static unsigned short = 1;
```

Final 7/12/2017

- Escribir un programa C que reciba por argumento el nombre de un archivo, y lo procese sobre sí mismo (sin crear archivos intermedios). El procesamiento consiste en eliminar toda secuencia de caracteres consecutivos repetidos, dejando en su lugar un solo cáracter de dicha secuencia.

Ej: la cadena `abcddeeefgggghee` debe ser reemplazada por `abcdefghe`

Ya explicado

- Explique qué es y para qué sirve una variable de clase (o atributo estático) en C++. Mediante un ejemplo de uso, indique cómo se define dicha variable, su inicialización y el acceso a su valor para realizar una impresión simple dentro de un main.

Ya explicado

- Defina una macro llamada *EXEC_IN_POSIX* que tome un puntero a función. La macro debe verificar si existe una versión definida de POSIX durante la compilación y ejecutar el puntero a función en dicho caso. Para detectar si existe una versión de POSIX se debe verificar la existencia de la macro ___POSIX_VERSION_, provista por el compilador.

```
#ifdef _POSIX_VERSION
#define EXEC_IN_POSIX(function)
function()
#endif
```

(No estoy segura para nada de esto)

- Defina el concepto de mutex y de un ejemplo de uso. Indique en qué casos es necesario.

  Ya explicado

- Defina una función C llamada *trim_all* que reciba una lista de strings *L* y el tamaño de dicha lista *C*. La función debe retornar una nueva lista de strings, que posea copias de los valores recibidos pero sin carácteres de espacio (`" "`) en el inicio de cada string. Indique claramente el tipo elegido para las variables L y C.

```
char** trim_all(const char** L, size_t C) {
    char** L_new = (char**) malloc(sizeof(char*) * C);
    for (size_t i = 0; i < C; i++) {
        const char* l = L[i];
        char* l_new = (char*) malloc(sizeof(char) * strlen(l));
        int index = 0;
        int to_copy = strlen(l) + 1;
        if (l[0] == ' ') {
```

```
            index = 1;
            to_copy = strlen(l);
        }
        memcpy(l_new, &l[index], to_copy);
        L_new[i] = l_new;
    }
    return L_new;
}
```

- Defina una rutina en C que se conecte a la IP 10.10.10.10, bajo el puerto 9000 y reciba un archivo binario. El archivo debe ser descargado a un archivo llamado 'datos.bin', cuidando el uso de recursos de memoria. El archivo finaliza al detectar conexión cerrada.

Ya explicado

- Escriba una clase template llamada *Sumador* que reciba por constructor un vector de elementos genéricos. La clase Sumador debe incluir un método llamado 'sumar' que acumule los valores del vector recibido y retorne un nuevo objeto del tipo genérico. ¿Qué restricciones se le piden al tipo genérico en la definición de Sumador?

Ya explicado

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `extern int (*I)[2];`

Se declara un puntero a un arreglo "I" de 2 enteros con signo definidos en un archivo externo

b) `static int *C[3];`

Se declara "C" como un arreglo de 3 punteros a enteros de alcance local.

c) `static float F(const char* a);`

Se declara la función "F" que es de alcance local y recibe un puntero a un carácter con signo constante y devuelve un número de punto flotante.

- Declare la clase *Parrafo* que almacena una acumulación de líneas de texto (strings) en cierto orden e incluye las operaciones de agregación y eliminación de líneas. La clase debe poseer los operadores usuales de copia, asignación, salida a flujo (<<), comparación (==), agregación (+), substracción (-). Los últimos 2 operadores deben admitir argumentos del tipo string en formato C (char*) y C++ (std::string).

Ya explicado

- Escriba una rutina que muestre un botón y un cuadro de texto en una ventana. Al hacer click en el botón debe desaparecer el cuadro de texto, dejando el resto de la ventana intacta.

Ya explicado

Final 10/8/2017

- Defina una rutina en C que se conecte a la IP 10.10.10.10, bajo el puerto 8888 y convierta números enteros recibidos en modo texto. Una vez establecida la conexión se debe recibir la lista de números utilizando el `\n` como carácter delimitador entre cada uno de ellos. Para cada entero recibido en modo texto, se debe enviar su valor convertido en 32 bits big-endian en modo binario sin delimitadores. El proceso finaliza al detectar la conexión cerrada.

Ya explicado

- Explique qué es y para qué sirve un constructor de copia en C++. Indique cómo se comporta el sistema si este no es definido por el desarrollador y explique al menos una estrategia para evitar que una clase particular sea copiable.

Ya explicado

- Escriba una función C llamada *replicate* que reciba 1 cadena (*S*), dos índices (*I1* e *I2*) y una cantidad (*Q*). La función debe retornar una copia de S salvo los caracteres que se encuentran entre los índices I1 e I2 que serán repetidos Q veces en la misma posición.

Ej: `replicate("Hola", 1, 2, 3)` retorna `Hololola`

Ya explicado

- ¿Qué elementos debo exigir a un equipo de desarrollo externo para poder utilizar una función de la librería que ellos están desarrollando? ¿En qué parte del proceso de compilación se resuelven las llamadas a dichas función?

Se necesita la definición de la función y eventualmente un header que incluya la declaración. Se resuelven las llamadas a la función en el paso de enlazado ya que allí se reemplaza la tabla de símbolos creada en la etapa de compilación por los valores propiamente dichos y si no está definida la función no puede generarse ningún valor para esa línea y se produce un error.

- Escriba un procedimiento que escriba por salida estándar los números pares e impares entre 0 y 1000. Se pide que los números pares sean escritos por un hilo mientras los impares sean escritos por otro. Contemple la correcta utilización y limpieza de recursos utilizados.

Ya explicado

- Escribir un programa C que reciba por argumento el nombre de un archivo de texto y lo procese sobre sí mismo (sin crear archivos intermedios). El procesamiento consiste en eliminar todo carácter que no sea letra (a-Z o A-Z) ni número (0-9).

```
int leer(FILE* f, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    int out = fgetc(f);
    *seek = ftell(f);
    return out;
}

void escribir(FILE* f, char c, long int* seek) {
    fseek(f, *seek, SEEK_SET);
    fputc(c, f);
    *seek = ftell(f);
}

int main(int argc, char* argv[]) {
    FILE *f;
    int c = 0;
    long int c_final = 0;
    long int seek_lectura = 0;
    long int seek_escritura = 0;

    char* file_name = argv[1];

    f = fopen(file_name, "r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;
```

```
        bool condicion_minuscula = c >= 'a' && c <= 'z';
        bool condicion_mayuscula = c >= 'A' && c <= 'Z';
        bool condicion_num = c >= '1' && c <= '9';

        if (condicion_minuscula || condicion_mayuscula || condicion_num) {
            escribir(f, c, &seek_escritura);
        } else {
            c_final--;
        }
    }
    fclose(f);
    truncate(file_name, sizeof(char) * c_final);
    return 0;
}
```

- Escriba una clase templates llamada *Delegador* que reciba por constructor un puntero a un objeto de tipo genérico y que a su vez contenga un método llamado 'ejecutar'. Dicho método no recibe argumentos y no realiza retorno alguno, pero invoca el método 'ejecutar' sobre el objeto recibido en el constructor.

No entiendo cómo le puede aplicar el método ejecutar al objeto recibido en el constructor si no recibe parámetros. Voy a asumir que el parámetro es el objeto recibido en el constructor:

```
template <class T>
class Delegador {
public:
    void ejecutar(T object) {}
    Delegador(T* object) {
        ejecutar(*object);
    }
};
```

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `void (*A)();`

Se declara un puntero a función "A" que no recibe ningún parámetro y no devuelve nada.

b) `void B(float a, float b){}`

Se define una función "B" que recibe dos números de punto flotante y no devuelve nada.

c) `static int **C[3]`

Se declara un arreglo "C" de alcance local que tiene 3 posiciones de punteros a punteros a números enteros.

- Defina la clase URL para permitir almacenar las siguientes propieades: protocolo (Ej: "http"), host(Ej: "fi.uba.ar"), port (Ej. 80) y file (Ej: "index.php", "resources/img/logo.png"). A su vez, se pide que implemente los siguientes operadores: operator<<,operator==, operator=.

Ya explicado

- Describa el concepto de loop de eventos (event loop) utilizado en programación orientada a eventos y, en particular, en entornos de interfaz gráfica (GUIs)

Ya explicado

Final 3/8/2017

- Explique cómo se comporta el modificador virtual en C++ y qué relación guarda con las VTables (tablas virtuales). De un breve ejemplo donde su uso sea imprescindible.

Ya explicado

- Escriba una macro llamada *ASSERT*. La macro debe permitir escribir una condición de código a validar y un mensaje de error en caso que la condición no se cumpla. Al imprimir el mensaje, se debe incluir el nombre del archivo C para que el programador pueda coregir rápidamente el error. Ejemplo de uso: ASSERT(variable == 2, "Se esperaba un 2 como param.")

Ya explicado

- Defina una rutina que abra el puerto 9000 en modo escucha y acepte hasta 10 conexiones en serie. Por cada conexión aceptada, se debe crear un archivo de texto y escribir toda la información enviada por la conexión hasta su cierre.

```
#define CONEX_EN_COLA 50
#define MAX_PAQUETE 1024

int main(int argc, char* argv[]) {
    const char* port = "5000";

    //1) Bind
    struct addrinfo hints;
    struct addrinfo *ptr;

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;       /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM; /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = AI_PASSIVE;     /* AI_PASSIVE for server           */
    getaddrinfo(NULL, port, &hints, &ptr);

    int skt;
    for (; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                     ptr->ai_socktype,
                     ptr->ai_protocol);
        if (skt != -1) break;
    }

    //Bind del socket
    bind(skt, ptr->ai_addr, ptr->ai_addrlen);

    //2) Listen
    listen(skt, CONEX_EN_COLA);

    //3) Accept
    char paquete[MAX_PAQUETE + 1];
    char buffer[MAX_PAQUETE];
    size_t pos_paquete = 0;
    size_t leido = 0;

    int cant_conexiones = 1;
    int peerskt = accept(skt, NULL, NULL);
    char file_name[64];
    snprintf(file_name, 64, "00%d.txt", cant_conexiones);
    FILE* write_file = fopen(file_name, "w");

    while (cant_conexiones <= 10) {
```

```
        //4) Recibir
        leido = recv(peerskt, buffer, MAX_PAQUETE, 0);
        if (leido == 0) { //Acepto otra conexion
            shutdown(peerskt, SHUT_RDWR);
            close(peerskt);
            peerskt = accept(skt, NULL, NULL);
            cant_conexiones++;

            snprintf(file_name, 64, "00%d.txt", cant_conexiones);
            fclose(write_file);
            write_file = fopen(file_name, "w");
            continue;
        }

        for (size_t i = 0; i < leido; i++) {
            paquete[i] = buffer[i];
        }
        paquete[leido] = '\0';
        if (strcmp(paquete, "FIN") == 0) break;
        fprintf(write_file, "%s", paquete);
    }
    //5) Liberar
    shutdown(peerskt, SHUT_RDWR);
    close(peerskt);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    fclose(write_file);
    freeaddrinfo(ptr);
    return 0;
}
```

- Ejemplifique el uso de valores por defecto en templates de C++. Escriba un ejemplo.

Ya explicado

- Escriba una función C que reciba 1 cadena (T), un arreglo de cadenas (L) y la longitud del arreglo (Q). La función debe buscar ocurrencias en todas las cadenas provistas en L dentro de T y retornar un valor total de coincidencias encontradas. Las coincidencias pueden aparecer en cualquier posición de T y pueden ser múltiples, es decir, pueden ser 0, 1 o más coincidencias.

```
int C(char* T, char** L, int Q) {
    int coincidences = 0;
    for (int L_index = 0; L_index < Q; L_index++) {
        char* l = L[L_index];
        int pos = 0;
        for (int T_index = 0; T_index < strlen(T); T_index++) {
            if(l[pos] == T[T_index]) {
                if (pos == strlen(l) - 1) {
                    coincidences++;
                    pos = 0;
                    continue;
                }
                pos++;
            }
            else pos = 0;
        }
```

```
    }
    return coincidences;
}
```

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `extern int (*I)[2]`

Se declara un puntero a un arreglo "I" de 2 posiciones de números enteros, definido en un archivo externo.

b) `static char *C[3]`

Se declara un puntero a un arreglo "C" de alcance local de 3 posiciones de carácteres con signo.

c) `static float F(float a, float b);`

Se declara la función "F" de alcance local que recibe dos números de punto flotante y devuelve un número de punto flotante.

- Dado un vector de enteros en C++, escriba un procedimiento que retorne el promedio de todos sus números en punto flotante. Debido a la longitud del vector, se pide que el procesamiento sea en paralelo con 2 hilos de ejecución. ¿Es necesario realizar un control de concurrencia entre 2 hilos? ¿Por qué?

Es necesario realizar un control de concurrencia entre 2 hilos para que los elementos de la lista se recorran 1 vez y sólo se computen esa vez.

```
class MonitorPromedio {
private:
    std::mutex m_contador;
    std::condition_variable cv_par, cv_impar;
    float& contador;
    std::list<vector>& elementos;
    std::list<vector>::iterator it;
    bool impar = true;

public:
    MonitorPromedio(float& contador, std::vector<int>& elementos):
            contador(contador), elementos(elementos) {
        it = elementos.begin();
    }

    void procesar_impar() {
        std::unique_lock<std::mutex> lock(m_contador);
        while (! impar) {
            cv_impar.wait(lock);
        }
        contador += *it;
        it++;
        impar = false;
        cv_par.notify_one();
    }

    void procesar_par() {
        std::unique_lock<std::mutex> lock(m_contador);
        while (impar) {
            cv_par.wait(lock);
        }
        contador += *it;
        it++;
```

```
        impar = true;
        cv_impar.notify_one();
    }
};

int main() {
    std::vector<int> elementos = {3, 3, 4, 4};
    float contador = 0;
    MonitorPromedio monitor(contador, elementos);

    size_t limit_par = elementos.size() / 2;
    size_t limit_impar;
    if (elementos.size() % 2 == 0) limit_impar = limit_par;
    else limit_impar = limit_par + 1;

    std::thread t1([&]() {
        for (int i = 0; i < limit_par; i++) {
            monitor.procesar_par();
        }
    });
    std::thread t2([&]() {
        for (int i = 0; i < limit_impar; i++) {
            monitor.procesar_impar();
        }
    });
    t1.join();
    t2.join();

    contador = contador / elementos.size();
    return 0;
}
```

- Escribir un programa C que reciba por argumento el nombre de un archivo de texto y lo procese sobre sí mismo (sin crear archivos intermedios). El procesamiento consiste en reemplazar todas las ocurrencias de 'UNO', 'DOS' y 'TRES' por '1', '2' y '3', respectivamente.

Ya explicado

- Defina la clase *Cadena* para que sea válido el siguiente código:

```
Cadena c2("abc");
Cadena c2("def);
Cadena c4(c2);
Cadena c3 = c1 + c2;
c1 = c2 = "ghi";
std::cout << c4 << std::endl;
```

```
class Cadena {
private:
    std::string cadena;
public:
    Cadena(std::string cadena) {
    this->cadena = cadena;
    }
    Cadena(const Cadena& other) { //Constructor por copia
        this->cadena = other.cadena;
```

```
    }
    Cadena operator+(const Cadena& other) const {
        Cadena result(other.cadena + this->cadena);
        return result;
    }
    Cadena& operator=(std::string other) {
        this->cadena = other;
        return *this;
    }
    friend std::ostream& operator<<(std::ostream& out, const Cadena& other) {
        out << other.cadena << std::endl;
        return out;
    }
};
```

- Escriba una rutina para ambiente que dibuje la siguiente imagen: (circulo azul en rectangulo negro que lo encierra)

Ya explicado

Final 20/7/2017

- ¿Describa brevemente el contenedor map de la librería estándar de C++? Ejemplifique su uso.

El contenedor **map** de C++ guarda elementos formados por una clave y un valor, siguiendo un orden específico. Las *claves* son las que se utilizan para ordenar e identificar unívocamente a los elementos mientras que los *valores* guardan el contenido asociado a cada clave. El orden a seguir está determinado por la función de comparación asociada a la clave. Son implementados como árboles de búsqueda binarios. Se puede acceder a un valor determinado por medio de la clave y el uso de corchetes.

Por ejemplo:

```
std::map<char, int> map;
map.insert({'a', 1}); //Inserta un par clave-valor ('a', 1);
std::cout << map['a'] << std::endl; //Imprime 1
```

- Escriba una macro que calcule el área de un circulo dado su radio. Escriba un pequeño ejemplo de uso.

```
#define AREA(radio) 3.14*radio*radio
int main() {
    int radio = 1;
    float area = AREA(radio);
}
```

- Escribir un programa ISO C que procese el archivo de texto sobre sí mismo. El proceso consiste en detectar las secuencias de caracteres 'ABAB' y reemplazarlas por 'C'.

```
int main(int argc, char* argv[]) {
    FILE *f;
    int c = 0;
    long int c_final = 0;
    long int seek_lectura = 0;
    long int seek_escritura = 0;

    char to_compare[] = "ABAB";

    char buffer[5];
    memset(buffer, 0, 5);
```

```
    bool inicio = true;
    int pos = 0;

    f = fopen("a.txt","r+");

    while ((c = leer(f, &seek_lectura)) != EOF) {
        c_final++;
        if (inicio) {
            buffer[pos] = c;
            pos++;
            if (buffer[3] != '\0') inicio = false;
            escribir(f, c, &seek_escritura);
            if (strcmp(to_compare, buffer) == 0) {
                c_final -= 3; //Saque 3 letras
                seek_escritura = c_final - 1; //Ir una atras para pisar
                escribir(f, 'C', &seek_escritura);
            }
            continue;
        }

        for (int i = 1; i <= 4; i++) {
            buffer[i-1] = buffer[i];
        }
        buffer[3] = c;

        if (strcmp(to_compare, buffer) == 0) {
            c_final -= 3; //Saque 3 letras
            seek_escritura = c_final - 1; //Ir una atras para pisar
            escribir(f, 'C', &seek_escritura);
        } else {
            escribir(f, c, &seek_escritura);
        }
    }
    fclose(f);
    truncate("a.txt", sizeof(char) * c_final);
    return 0;
}
```

- Escriba una función que dado un arreglo de file descriptors de sockets 'list', la cantidad de sockets 'qty' y una cadena de caracteres 'text' finalizada en NULL, envíe todo el contenido de 'text' a cada uno de los sockets. A su vez, se debe esperar una respuesta de 2 bytes de cada socket para retornar true si todas las respuestas fueron 'OK' y false en caso contrario.

```
def send(int list[], int qty, char* text) {
    for (int i = 0; i < qty; i++) {
        int skt = list[i];
        send(skt, text, strlen(text) + 1, MSG_NOSIGNAL);
    }
    bool result = true;
    for (int i = 0; i < qty; i++) {
        char response[3];
        recv(skt, &response[0], 2, 0);
        response[2] = '\0';
        result &= (strcmp(response, "OK") == 0);
```

```
    }
    return result;
}
```

- Escriba las siguientes declaraciones/definiciones considerando un contexto global:

a) La definición de un puntero a número flotante de doble precisión denominado F1

```
double* F1 = NULL;
```

b) La declaración de un puntero a puntero a entero corto denominado F2

```
short* F2;
```

c) La declaración de una función denominada func que tome como parámetro una puntero a función que soporte 2 parámetros enteros con signo y retorne char. La función func, a su vez, debe retornar char

```
char func(char (*F)(int, int));
```

- Escriba una rutina de interfaz gráfica que contenga una ventana que contenga el siguiente dibujo (un triangulo rojo)

```
class MyTriangle : public Gtk::DrawingArea {
protected:
    // Override default signal handler:
    bool on_draw(const Cairo::RefPtr<Cairo::Context>& context) override{
        Gtk::Allocation allocation = this->get_allocation();
        int window_width = allocation.get_width();
        int window_height = allocation.get_height();
        context->move_to(window_width/2, window_height);
        context->line_to(0, window_height);
        context->line_to(window_width/2, 0);
        context->line_to(window_width, window_height); //Seguir los vertices, no da lo mismo cualquier
        context->set_source_rgb(1.0, 0.0, 0.0); //rojo
        context->fill();
        return true;
    }
};

int main(int argc, char **argv) {
    auto app = Gtk::Application::create();
    Gtk::Window window;

    MyTriangle triangle;
    window.add(triangle);
    triangle.show();
    return app->run(window);
}
```

- ¿Qué es un método estático (o de clase) en C++? Muestre su manera de empleo mediante un breve ejemplo

Ya explicado

- Defina una clase template llamada *Concatenator* que posea un método llamado agregar que recibe un elemento que será almacenado internamente y en orden. Dada una instancia llamada concatenador, se debe permitir la impresión de todos los elementos agregados mediante:

```
std::cerr << concatenador << std::endl
```

```
template<class T>
```

```
class Concatenator {
private:
    std::list<T> data;
public:
    void add_element(T element) {
        data.push_back(element);
    }
    friend std::ostream& operator<<(std::ostream& out, const Concatenator& other) {
        for (auto i = other.data.begin(); i != other.data.end(); i++) {
            out << *i << std::endl;
        }
        return out;
    }
};
```

- ¿En qué consiste el proceso de enlazado (linking)?

Ya explicado

- Escriba un programa que reciba un listado de enteros 'list' y un archivo abierto 'out' e imprima en el archivo aquellos números que sean pares. Debido a la longitud del listado se requiere el uso de multithreading para brindar una solución óptima.

```
void escribir_pares(std::ostream& out, std::list<int>& list) {
    MonitorArchivo monitor(out, list);

    std::thread t1([&] () {
        for (size_t i = 0; i < list.size() && !monitor.terminar(); i++) {
            monitor.leer();
        }
    });
    std::thread t2([&] () {
        for (size_t i = 0; i < list.size() && !monitor.terminar(); i++) {
            monitor.escribir();
        }
    });

    t1.join();
    t2.join();
}

class MonitorArchivo {
private:
    std::ostream& archivo;
    std::mutex m;
    std::condition_variable cv_escritor, cv_lector;
    bool esta_leyendo = true;
    std::list<int>& elementos;
    std::list<int>::iterator iterator;
    bool end = false;

public:
    MonitorArchivo(std::ostream& archivo, std::list<int>& elementos):
        archivo(archivo), elementos(elementos) {
        this-> iterator = elementos.begin();
    }
```

```
    void escribir() {
        std::unique_lock<std::mutex> lock(m);
        while (esta_leyendo) {
            cv_escritor.wait(lock);
        }
        archivo << *iterator;
        iterator++;
        if (iterator == elementos.end()) end = true;
        esta_leyendo = true;
        cv_lector.notify_one();
    }
    void leer() {
        std::unique_lock<std::mutex> lock(m);
        while (! esta_leyendo) {
            cv_lector.wait(lock);
        }
        if (iterator == elementos.end()) {
            end = true;
            return;
        }
        if (*iterator % 2 == 0) {
            esta_leyendo = false;
            cv_escritor.notify_one();
        } else {
            iterator++;
        }
    }
    bool terminar() {
        return end;
    }
};
```

Final 13/7/2017

- Explique el concepto de referencias en C++. ¿Qué diferencias posee respecto de los punteros?

Las referencias en C++ deben ser inicializadas al construirse y una vez que referencian a algún objeto no pueden referenciar a otro. Funcionan como un alias y el compilador en algunos casos ni siquiera reservará memoria para una referencia. En cambio los punteors pueden crearse sin inicializar, cambiar de objeto al que apuntan y siempre consumen memoria. Como colorario, las referencias no pueden referenciar a nulls, lo que reduce la posibilidad de crashes.

- Escriba una rutina que procese un archivo binario indicado por parámetro sobre sí mismo, sumarizando los listados de números que posee almacenado. La sumarización consiste en recorrer los valores enteros de 32 bits con signo grabados en formato big-endian y acumular sus valores hasta encontrar el valor 'OXFFFFFFFF' que se considera un separador entre listados. Todos los valores enteros detectados son reemplazados por su sumatoria (en el mismo formato) manteniendo luego el elemento separador. Considere archivos bien formados.

Ya explicado

- Explique el concepto de object slicing (objeto recortado). Escriba un breve ejemplo sobre cómo esto afecta a una función que pretende aplicar polimorfismo sobre uno de sus parámetros.

Ocurre cuando un objeto de la clase derivada es copiado a un objeto de la clase base y dicha copia no posee ningún atributo de la clase derivada. Se evita con el uso de punteros y referencias. Por ejemplo:

```
class Base {
protected:
    int a;
public:
    Base(int a): a(a) {}
    virtual void print() {
        std::cout << "Base, a=" << a << std::endl;
    }
};

class Derivada: public Base {
private:
    int b;
public:
    Derivada(int a, int b): Base(a), b(b) {}
    void print() {
        std::cout << "Derivada, a=" << a << " b=" << b << std::endl;
    }
};

int main() {
    Derivada d(1, 2);
    d.print();
    Base b(3);
    b.print();
    b = d;
    b.print();
    return 0;
}
```

La salida es:

```
Derivada, a=1 b=2
Base, a=3
Base, a=1
```

Como se puede observar se perdió el atributo b en la copia que se realiza del objeto d (de la clase derivada) al objeto b (de la clase base).

Estas son dos soluciones posibles:

```
void print(Base& obj) {
    obj.print();
}

void printPtr(Base* obj) {
    obj->print();
}

int main() {
    Derivada d(1, 2);
    print(d);
    printPtr(&d);
    return 0;
}
```

La salida es en ambos casos:

```
Derivada, a=1, b=2
```

- Escriba una rutina de interfaz gráfica que lance una aplicación de ventana simple, con un cuadro de texto y un botón. Al presionar el botón, se debe borrar el valor ingresado por el usuario en el cuadro de texto.

```
#include <gtkmm.h>

void borrar(Gtk::Entry* entry) {
    Glib::ustring texto = "";
    entry->set_text(texto);
}

class BotonBorrar : public Gtk::Button {
    public:
        BotonBorrar(Gtk::Entry* entry) : Gtk::Button("Borrar") {
            this->entry = entry;
        }
    private:
        Gtk::Entry* entry;
        void on_clicked() {
            borrar(entry);
        }
};

int main(int argc, char* argv[]) {
    Gtk::Main kit(argc, argv);
    Gtk::Entry entry;
    BotonBorrar boton(&entry);
    Gtk::VBox vbox;
    vbox.add(entry);
    vbox.add(boton);
    Gtk::Window v;
    v.add(vbox);
    v.show_all();
    Gtk::Main::run(v);
    return 0;
}
```

- Escriba una función de C llamada *strncat_new* que reciba 3 parámetros: dos punteros a carácter constante (*S1* y *S2*) y un entero sin signo (*L*). La función debe concatenar S1 con S2 y retornar una nueva cadena de caracteres considerando L como tamaño máximo para cualquiera de los elementos (S1, S2 y la nueva cadena). La función debe detectar condiciones de error respecto de la longitud y retornar NULL en cualquier caso.

```
char* strncat_new(const char* S1, const char* S2, unsigned int L) {
    if (L < strlen(S1) || L < strlen(S2) ||
    L < strlen(S1) + strlen(S2)  ) {
        fprintf(stderr, "Error en longitud");
        return NULL;
    }
    int len = strlen(S1) + strlen(S2) + 1;
    char* result = (char*) malloc(sizeof(char) * len);

    strncpy(result, S1, strlen(S1));
    strncat(result, S2, strlen(S2));
```

```
    return result;
}
```

- Explique la diferencia entre las étapas de compilación y enlazado (linking). Escriba un breve ejemplo de código con errores para cada una de ellas indicándolos en forma clara.

Ya explicado

- Declare la clase Socket y realice un programa que la emplee para realizar conexiones con un servidor externo. Dicho programa recibe una dirección IP por argumento y establece una conexión al puerto 80 de dicha dirección. Luego, debe enviar el mensaje "GET/HTTP/1.1", esperar la respuesta, imprimirla por consola y cerrar la conexión. Se considera como fin de respuesta la detección de `\n\n`.

```
class Socket {
private:
    int skt;

public:
    Socket();
    explicit Socket(int skt);
    Socket(Socket&& origen); // constructor por movimiento
    Socket& operator=(Socket&& origen); // asignador por movimiento

    ~Socket();
    void shutdown();
    void connect(const char* host, const char* port);
    void bind(char* port);
    void listen();
    void send(const void*, size_t);
    void receive(void*, size_t);
    Socket accept();
};

#define MAX_LEN 1024

int main(int argc, char* argv[]) {
    const char* ip = argv[1];
    const char* port = "8080";

    Socket socket;
    socket.connect(ip, port);

    char message[] = "GET/HTTP/1.1";
    socket.send(message, MAX_LEN);

    char response[1024];
    socket.receive(response, MAX_LEN);

    int end = 0;
    for (int i = 0; i < strlen(response); i++) {
        if (response[i] == '\n') end++;
        printf("%c", response[i]);
        if (end == 2) break;
    }
}
```

- Realice una rutina que lance dos hilos. El primero debe realizar la suma de 1 a 1000, el segundo debe realizar la suma de 1001 a 2000. Luego de la finalización de los hilos se debe imprimir la suma de ambos resultados.

```
#include <iostream>
#include <thread>

void primeras_sumas(int* suma) {
    for (int i = 1; i <= 1000; i++) {
        *suma += i;
    }
}

void segundas_sumas(int* suma) {
    for (int i = 1001; i <= 2000; i++) {
        *suma += i;
    }
}

int main() {
    int s1 = 0;
    std::thread t1(primeras_sumas, &s1);
    int s2 = 0;
    std::thread t2(segundas_sumas, &s2);
    t1.join();
    t2.join();
    std::cout << s1 + s2 << std::endl;
    return 0;
}
```

- Describa el concepto de templates en C++. De un breve ejemplo de una función template.

Se utiliza para ganar generalidad en el código y evitar código repetido. La idea es usar el mismo código pero reemplazando el tipo particular por uno genérico T. A diferencia del precompilador, el código template es procesado por el compilador para que sea más seguro y que el chequeo de tipos y errrores estén mejor explicados. Hay clases, struct y funciones template.

Al llamar a una función template se puede especificar sobre qué tipo se trabaja o dejar que el compilador lo deduzca automáticamente basándose en los parámetros.

```
template<class T>
void foo(T i) {
    std::cout << i << std::endl;
}

foo<int>(1); // T = int, indicado de forma explícita
foo(2); // T = int, detectado de forma automática
foo<char>(3); //T = char, indicado de forma explícita
```

- Describa con exactitud las siguientes declaraciones/definiciones globales:

a) `float F(float a, float b);`

Se declara la función F que recibe dos números de punto flotante y devuelve un número de punto flotante.

b) `static int *A;`

Se declara un puntero "A" de alcance local a un entero.

c) `extern unsigned int (*B)[10];`

Se declara un puntero a un arreglo "B" de 10 posiciones de enteros sin signo, definido en un archivo externo.

Final 6/7/2017

- Indique 2 posibles usos del modificador const. Utilice en cada caso una breve sentencia de código que lo emplee junto con la explicación del significado/consecuencia de su uso.

Ya explicado

- Escriba una rutina que procese un archivo indicado por parámetro sobre sí mismo eliminando todas las ocurrencias de caracteres repetidos. El procesamiento consiste en detectar toda sucesión de dos o más carácteres idénticos y reemplazarlos por una única ocurrencia.

Ya explicado

- Explique cómo funciona la herencia pública en C++ y qué diferencias posee con la herencia privada.

Ya explicado

- Escriba un pequeño programa de interfaz gráfica que lance una aplicación de ventana simple, con un único botón. El botón debe tener el texto 'Cerrar' que al ser presionado oculta la ventana y cierra el sistema.

Ya explicado

- Escriba una función de C llamada *obtenerPalabrasCortas* que reciba dos parámetros: un puntero a carácter constante (S) y un entero sin signo (L). La función recibirá una oración en S que debe ser recorrida detectando palabras y copiando aquellas cuya longitud sea menor que L en una lista de resultados. Considere el carácter espacio (") como delimitador entre palabras. Explique brevemente la estructura de retorno elegida para almacenar los resultados.

Ya explicado

- Defina una macro llamada *LOG* que reciba una cadena como parámetro y la imprima utilizando printf. La macro debe ejecutar el comando printf solamente cuando detecta que la macro *LOG_ENABLED* se encuentra definida.

Ya explicado

- Defina un operador global == que toma un vector STL de tipo genérico T y un arreglo de elementos T. El operador debe retornar un bool indicando si las colecciones son iguales, es decir,si todos los elementos T coinciden en ambas colecciones.

Ya explicado

- Explique el concepto de mutex. Escriba un ejemplo simple de código donde su uso resulte fundamental para garantizar el correcto empleo de recursos en un contexto concurrente.

Ya explicado

- Escriba una función que reciba un parámetro *PUERTO*, cree un socket y acepte conexiones entrantes indefinidamente hasta detectar un error en la aceptación. A cada cliente conectado se debe transmitir la cadena "Conexión rechazada" y, acto seguido, cerrar su conexión.

Ya explicado

- Escriba:

a) La definición de una función de alcance local que toma como parámetros un puntero a entero con signo y un puntero a carácter con signo: sin retorno

```
static void F(int* a, char* b){}
```

b) La declaración de un puntero a función que respete la firma de la función anterior

```
static void (*F)(int*, char*);
```

c) La declaración de una variable global entera que se encuentre en otra unidad de compilación

```
extern int a;
```

Final 2/3/2017

- Describa los pasos realizados por la instrucción del preprocesador `#include`. ¿Por qué se encuentra desrecomendado utilizar #include con archivos .c? Justifique

El preprocesador reemplaza la línea `#include nombre` con el archivo de cabecera del sistema con ese nombre. Más concretamente donde se pone la directiva #include se sustituirá por el contenido completo del archivo con ese nombre.

`#include` normalmente obliga a usar protectores de `#include` porque si se incluye más de 1 vez el mismo archivo (dependiendo del contenido) puede causar que se intente declarar varias veces las mismas funciones o tipos de variable, lo que va a generar un error al compilar. Esto se intenta prevenir de la siguiente forma:

```
#ifndef __ARCHIVO_H__
#define __ARCHIVO_H__
/*... declaraciones de funciones, etc. ...*/
#endif
```

Como resultado, al intentar incluirse el archivo por segunda vez, la operación `"ifndef"` va a dar falso porque `__ARCHIVO_H__` ya estaba definido de la primera vez que se incluyó, y a consecuencia se saltea todo el bloque hasta llegar al "endif" que suele estar al final del archivo.

Se encuentra desrecomendado utilizar `#include` con archivos .c porque justamente no utilizan dichos protectores. En definitiva se puede incluir cualquier archivo porque se trata de un copy-paste pero si se incluye un archivo .c y luego se lo compila junto con ese archivo surgirá un error de multiple definición (error de linkeo) ya que no se utiliza la protección detallada previamente.

- Escriba el .H correspondiente a una biblioteca que exporta:

a) La definición de un tipo llamado *alumno_t* correspondiente a una estructura con nombre (cadena de 50 caracteres) y padron (entero sin signo)
b) Una función llamada *alumno_get_padron* que toma un puntero a alumno_t y retorna un entero sin signo
c) Una función llamada *procesar_alumnos* que recibe un puntero a *alumno_t*, un entero sin signo y un puntero a función con parámetro a alumno_t y resultado vacío, 'procesar_alumnos' no debe retornar ningún valor

```
typedef struct {
    char nombre[50];
    unsigned int padron;
} alumno_t;

unsigned int alumno_get_padron(alumno_t* alumno);
void procesar_alumnos(alumno_t* alumno, unsigned int a, void (*F)(alumno_t));
```

- Explique el concepto de referencia en C++. ¿Qué diferencias posee con el uso de punteros? Escriba un breve ejemplo de su uso.

Ya explicado

- Escriba una función ISO C llamada 'split' que reciba una cadena de caracteres constante 'S' y retorne un puntero a una secuencia con todas las palabras que posee 'S'. Considere como separador entre palabras al carácter espacio.

```
char** split(const char* S) {
    int separadores = 0;
    for (int i = 0; i < strlen(S); i++) {
        if(S[i] == ' ') separadores++;
    }
    char** resultado = (char**) malloc(sizeof(char*) * (separadores + 1));
    size_t pos = 0;
    size_t offset = 0;
    size_t pos_array = 0;
    for (int i = 0; i <= strlen(S); i++) {
        if (S[i] == ' ' || S[i] == '\0') {
            char* cadena = (char*) malloc(sizeof(char) * (pos+1));
            memcpy(cadena, &S[offset], pos);
            cadena[pos] = 0;
            resultado[pos_array] = cadena;
            pos_array++;
            offset += pos+1;
            pos = 0;
        }
        else pos++;
    }
    return resultado;
}
```

- Escriba una rutina en C++ que cree un vector STL de enteros, lance 2 hilos y espere su finalización. El primer hilo debe generar los números del 1 al 100 y agregarlos al vector. Luego de esto, finaliza. El segundo hilo debe acceder al vector, eliminar el último número que posee e imprimirlo por salida standard. Al detectar que no hay elementos en el vector, finaliza. Disponga de los elementos necesarios para la sincronización entre los hilos y uso correcto de los recursos.

```
class MonitorImpresor {
private:
    std::mutex m;
    std::condition_variable cv_impresor;
    std::vector<int>& elementos;
    bool fin_escritura = false;

public:
    MonitorImpresor(std::vector<int>& elementos): elementos(elementos) {}
    void escribir_numeros(int i) {
        std::unique_lock<std::mutex> lock(m);
        elementos.push_back(i);
        if (i == 100) {
            fin_escritura = true;
            cv_impresor.notify_one();
        }
    }
    void imprimir_numeros() {
        std::unique_lock<std::mutex> lock(m);
        while (! fin_escritura) {
            cv_impresor.wait(lock);
```

```
        }
        int i = elementos.back();
        std::cout << i << std::endl;
        elementos.pop_back();
    }
};

int main() {
    std::vector<int> elementos;
    MonitorImpresor impresor(elementos);
    std::thread t1([&] () {
        for (int i = 1; i <= 100; i++) {
            impresor.escribir_numeros(i);
        }
    });
    std::thread t2([&] () {
        for (int i = 1; i <= 100; i++) {
            impresor.imprimir_numeros();
        }
    });
    t1.join();
    t2.join();
    return 0;
}
```

- Escribir un programa ISO C que lea el archivo 'a.txt' e invierta sus caracteres sin utilizar archivos intermedios.

```
int main(int argc, char* argv[]) {
    FILE *f = fopen("a.txt", "r+");

    long int seek_leer_adelante = 0;
    long int seek_leer_atras = -1;

    fseek(f, 0L, SEEK_END);
    int len = ftell(f);

    for (int i = 1; i <= len/2; i++) {
        fseek(f, seek_leer_adelante, SEEK_SET);
        int c_adelante = fgetc(f); //Leo primer caracter

        fseek(f, seek_leer_atras, SEEK_END);
        int c_atras = fgetc(f); //Leo ultimo caracter

        fseek(f, seek_leer_atras, SEEK_END);
        fputc(c_adelante, f); //Escribo primer caracter en ultimo lugar

        fseek(f, seek_leer_adelante, SEEK_SET);
        fputc(c_atras, f); //Escribo ultimo caracter en primer lugar

        seek_leer_adelante += 1;
        seek_leer_atras -= 1;
    }
    fclose(f);
    return 0;
```

```
}
```

- ¿Qué son las excepciones en C++? Dé un ejemplo de uso que incluya las claúsulas try/catch

Una excepción es un error que puede ocurrir debido a una mala entrada por parte del usuario, un mal funcionamiento en el hardware, un argumento inválido para un cálculo matemático, etc. Para remediar esto, el programador debe estar atento y escribir los algoritmos necesarios para evitar a toda costa que un error de excepción pueda hacer que el programa se interrumpa de manera inesperada. C++ soporta una forma más directa y fácil de ver tanto para el programador como para los revisores del código en el manejo de excepciones que su símil en el C estándar y esta consiste, tratándose del lenguaje C++, en el mecanismo try, throw y catch.

```
#include <iostream>
#include <string>

int main() {
    std::string s = "Hola";
    try {
        std::cout << s.at(100) << std::endl;
    }
    catch(exception& e) {
        std::cout << e.what() << std::endl;
    }
    std::cin.get();
    return 0;
}
```

En este ejemplo como el string no tiene 101 carácteres, va a saltar una excepción que será capturada por el **catch** y permitirá que se siga ejecutando el programa correctamente. Si no hubiese estado esta claúsula **try** y **catch**, nunca se hubiese ejecutando la línea *std::cin.get()* y se hubiese interrumpido la ejecución.

- ¿Qué propósito tiene la función accept? Indique parámetros que recibe y retorno esperado

La función **accept** se utiliza para aceptar una conexión y es bloqueante.

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Extrae la primera petición de conexión de la cola de conexiones pendientes para el socket **sockfd** y devuelve un nuevo file descriptor refiriendose a ese socket.

- **sockfd** es un socket que se creó con `socket()`, bindeó a una dirección local con `bind()` y está esperando conexiones después de `listen()`.
- **addr**: es un puntero a un `struct sockaddr`, donde se guardan las direcciones del peerskt
- **addrlen**: en un principio contiene el tamaño en bytes de addr y en el retorno el tamaño actual de la dirección del peerskt
- Escriba una rutina para ambientes gráficos que exponga una ventana con un cuadro de texto y un botón. Al realizar click sobre el botón, el sistema debe tomar la información ingresada en el cuadro de texto e imprimirla por consola.

```
#include <gtkmm.h>
#include <iostream>

void imprimir(Gtk::Entry* entry) {
    Glib::ustring texto = entry->get_text();
    std::cout << texto << std::endl;
}
```

```
class BotonImprimir : public Gtk::Button {
    public:
        BotonImprimir(Gtk::Entry* entry) : Gtk::Button("Imprimir") {
            this->entry = entry;
        }
    private:
        Gtk::Entry* entry;
        void on_clicked() {
            imprimir(entry);
        }
};

int main(int argc, char* argv[]) {
    Gtk::Main kit(argc, argv);
    Gtk::Entry entry;
    BotonImprimir boton(&entry);
    Gtk::VBox vbox;
    vbox.add(entry);
    vbox.add(boton);
    Gtk::Window v;
    v.add(vbox);
    v.show_all();
    Gtk::Main::run(v);
    return 0;
}
```

- Utilizando templates defina las clases necesarias para que el siguiente código sea válido:

```
Pila<int> p1(5000), p2(5000);
p1.push(123);
p2.push(456);
std::cout << "p1 == p2. Resultado: " << (p1 == p2) << std::endl;
```

```
template<class T>
class Pila {
private:
    std::vector<T> elements;
    size_t size;

public:
    Pila(size_t size) {
        this->size = size;
    }

    void push(T element) {
        elements.push_back(element);
    }

    bool operator==(const Pila& other) const {
        auto it_1 = this->elements.begin();
        auto it_2 = other.elements.begin();
        while (it_1 != this->elements.end() && it_2 != other.elements.end()) {
            if (*it_1 != *it_2) return false;
            it_1++;
            it_2++;
```

```
        }
        if (it_1 == this->elements.end() && it_2 == other.elements.end()) return true;
        return false;
    }
};
```

Final 23/2/2017

- ¿Qué es una función callback? ¿Por qué es importante en entornos gráficos de programación? De un breve ejemplo

Una **función callback** es una función pasada dentro de otra función como parámetro y que luego es invocada dentro de esa función para completar alguna rutina o acción. Es importante en entornos gráficos de programación porque se puede ejecutar en respuesta a una acción predeterminada por el usuario como cliquear un objeto o cerrar una ventana. Se suele asociar una función callback con una acción específica de usuario al asignar una función callback para dicha acción.

Por ejemplo:

```
void operation(bool (*f)(int)) {
   for (int l = 0; l < 100; l++) {
      if (l % 10 == 0) f(l / 10);
   }
}
```

- Escriba un programa que implemente un cliente TCP. El mismo debe conectarse a un servidor IP y PORT indicados por argumentos y recibir paquetes de datos enviados por el servidor. Los paquetes constan de una cantidad variable de bytes y se encuentran delimitados por `-`. Por cada paquete recibido se debe imprimir: `Bytes recibidos: <cantidad-de-bytes-del-paquete>\n`

```
#define MAX_LEN 1024

int main(int argc, char* argv[]) {
    const char* host = "localhost";
    const char* port = "8080";

    struct addrinfo hints;
    struct addrinfo *result, *ptr;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;          /* IPv4 (or AF_INET6 for IPv6)     */
    hints.ai_socktype = SOCK_STREAM;    /* TCP  (or SOCK_DGRAM for UDP)    */
    hints.ai_flags = 0;                 /* None                            */

    getaddrinfo(host, port, &hints, &result);
    int skt;
    for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {
        skt = socket(ptr->ai_family,
                     ptr->ai_socktype,
                     ptr->ai_protocol);
        if (skt != -1) {
            connect(skt, ptr->ai_addr, ptr->ai_addrlen);
            break;
        }
    }
    char response[MAX_LEN];
```

```
    char package[MAX_LEN];

    int pos = 0;
    int new_pos = 0;

    bool server_closed = false;
    while (! server_closed) {
        int read = recv(skt, &response[pos], MAX_LEN, 0);
        if (read == 0) {
            server_closed = true;
            continue;
        }
        for (int i = pos; i < read && response[i] != '\0'; i++) {
            if (response[i] == '<') continue;
            package[new_pos] = response[i];
            new_pos++;
            if (response[i] == '>') {
                package[new_pos-1] = '\0';
                printf("Bytes recibidos: <%zu>\n", strlen(package));
                new_pos = 0;
                memset(package, 0, MAX_LEN);
            }
        }
        pos += read;
    }
    freeaddrinfo(result);
    shutdown(skt, SHUT_RDWR);
    close(skt);
    return 0;
}
```

- Escribir un programa ISO C que procese el archivo 'a.txt' creando el 'b.txt'. El proceso consiste en reemplazar todas las secuencias de bytes 0x01, 0x02, 0x03, 0x04 por 0x04, 0x03, 0x02 y 0x01, dejando intacto el resto de los datos.

Dejo cómo crear el archivo en binario:

```
int main(int argc, char* argv[]) {

    FILE* f_bin = fopen("a.txt", "wb");

    uint8_t value = 0x01;
    uint8_t value_2 = 0x02;
    uint8_t value_3 = 0x03;
    uint8_t value_4 = 0x04;

    fwrite(&value, 1, 1, f_bin);
    fwrite(&value_2, 1, 1, f_bin);
    fwrite(&value_3, 1, 1, f_bin);
    fwrite(&value_4, 1, 1, f_bin);

    fclose(f_bin);

    FILE* f = fopen("a.txt", "rb");
    FILE* f_write = fopen("b.txt", "wb");
```

```
    uint8_t buffer;

    while (fread(&buffer, 1, 1, f) != 0)  {
        if (buffer == value)  {
            fwrite(&value_4, 1, 1, f_write);
        } else if (buffer == value_2) {
            fwrite(&value_3, 1, 1, f_write);
        } else if (buffer == value_3) {
            fwrite(&value_2, 1, 1, f_write);
        } else if (buffer == value_4) {
            fwrite(&value, 1, 1, f_write);
        } else {
            fwrite(&buffer, 1, 1, f_write);
        }
    }

    fclose(f);
    fclose(f_write);

    return 0;
}
```

- Escribir una función que retorne char* y reciba 3 argumentos: char* string, char car, int qty. La función debe crear una nueva cadena utilizando los caracteres de string pero reemplazando toda ocurrencia del carácter car por qty veces dicho carácter.

Ej:

```
INPUT: "abcdeabc", 'b', 3
OUTPUT: "abbbcdeabbbc"
```

```
char* expand(char* string, char car, int qty) {
    //Encuentro cantidad de ocurrencias del caracter
    int ocurrencias = 0;
    for (int i = 0; i < strlen(string); i++) {
        if (string[i] == car) ocurrencias++;
    }
    size_t len = strlen(string) + ocurrencias * (qty -1);
    char* result = (char*) malloc(sizeof(char) * (len + 1));
    size_t pos = 0;
    for (int i = 0; i < strlen(string); i++) {
        if (string[i] == car) {
            for (int j = 0; j < qty; j++) {
                result[pos] = car;
                pos++;
            }
        }
        else {
            result[pos] = string[i];
            pos++;
        }
    }
    result[len] = '\0';
    return result;
}
```

- ¿Por qué se dice que los sistemas escritos en lenguaje C/C++ son portables? ¿Qué precauciones son necesarias para escribir un programa C/C++ que cumpla con esa cualidad?

Se dice que los sistemas escritos en lenguaje C/C++ son portables porque pueden ejecutarse en varias plataformas de software de base, o, en definitiva, en varios sistemas operativos como Linux o Windows.

Una aplicación escrita en C es portable a nivel de código fuente, es decir, se puede portar el fuente de la aplicación de un sistema a otro, y compilarla para obtener los ejecutables para dicho sistema, sin mayores modificaciones. La precaución a tomar debe ser tener instaladas las bibliotecas de desarrollo utilizadas en el sistema operativo en el que se compilará.

- Describa las siguientes sentencias. Indique si se trata de declaraciones o definiciones. En caso de tratarse de una definición, indique en qué segmento de código vive el elemento definido:

a) `extern int* x;`

Se trata de una declaración de un puntero a un entero "x" declarado en un archivo externo.

b) `static float* x(float* a) { return a; }`

Se trata de una definición de una función "x" de alcance local que recibe un puntero a un número de punto flotante y devuelve un puntero a un número de punto flotante. Las definiciones de funciones viven en el code segment.

c) `float x;`

Se trata de una declaracion de un número de punto flotante "x".

7) Escriba una pequeña función que reciba por parámetro un std::list genérico, lo recorra y envíe por STDOUT cada uno de los elementos separados por un `\n`. La función debe poder ser ejecutada con cualquier especialización de std::list.

Precondición: la especialización de `std::list` debe tener sobrecargado el `operador<<`.

```
template<class T>
void print(std::list<T> elements) {
    for (auto i = elements.begin(); i != elements.end(); i++) {
        std::cout << *i << std::endl;
    }
}
```

- ¿Por qué se recomienda uilizar los modificadores `const &` en la firma de una función cuando se requiere pasaje de parámetros std::string en C++? Escriba un breve ejemplo con una función que recibe un std::string sin utilizar `const &` junto con su invocación y represente con una ilustración el espacio de stack durante la invocación de la misma.

Se recomienda utilizar los modificadores `const &` en la firma de una función cuando se requiere pasaje de parametros std::string en un primer lugar para evitar una copia innecesaria que empeore tanto la performance como el espacio gastado en memoria y en segundo lugar para garantizar que la misma no se modifique en la ejecución (ya que en caso de hacerlo se detectaría en tiempo de compilación). El diagrama del stack lo debo.

- Explique las diferencias entre un hilo y un proceso. ¿En qué casos es conveniente utilizar dichos elementos para programación paralela?

Un proceso es una instancia (ejecución) de un programa. En un sistema con varios procesadores o al menos un procesador con varios núcleos, cada proceso puede tener varios hilos de ejecución (threads).

Respecto a los recursos compartidos las diferencias son las siguientes:

	Procesos	Hilos
Variables locales no static y automáticas	No	No
Variables globales y locales static	No	Sí

	Procesos	Hilos
Code segment	No	Sí
Heap	No	Sí
File descriptors	No	Sí
Colas I/O	No	No
Registros del procesador	No	No

Consultar: ¿En qué casos…?

- Explique cómo funciona la sobrecarga de operadores en C++

La **sobrecarga de operadores** permite sobrecargar los operadores default de las clases y los struct de C++. De esta manera utiliza el comportamiento del compilador para convertir "funciones del operador". Al programar orientado a objetos permite que el programador pueda predefinir el comportamiento del mismo ante ciertas operaciones consideradas default, como pueden ser la suma, la resta, el operador asignación =, el operador ==, operadores de flujo << y >>, entre muchos otros.

Sin fecha - 3ero de la tanda 2do - 2016

- Escriba un programa ISO C que procese el archivo de texto a.txt sobre sí mismo. El sistema debe recibir dos cadenas por línea de comando: clave y valor. El procesamiento consiste en recorrer el archivo a.txt y reemplazar todas las ocurrencias de clave por el contenido de valor.

Ej de ejecución:

```
./programa CONSTANTE 1
./programa MSG "Hola mundo"
```

Este ejercicio es de agrandar y achicar. FUTURO

- Explique en qué situaciones es recomendable utilizar programación multi-hilo(multithreading) para realizar cierto procesamiento. ¿Existe algún caso donde utilizar multithreading sea perjudicial?

El hecho de usar programación multihilo no implica que siempre se va a tener una mejora de rendimiento y es potestad del programador el decidir cuándo conviene usar este tipo de programación. En líneas generales, si la aplicación desarrollada es muy simple no tiene sentido plantearse este tipo de programación.

Tampoco puede que sea buena idea emplear el multihilo para aplicaciones excesivamente complejas pues provocará excesiva sobrecarga y será más un problema que una ventaja.

Por regla general, se considera que una aplicación es candidata a ser programa en multihilo cuando cumple las siguientes premisas:

1. La aplicación va a realizar diversas tareas diferenciadas con un alto costo computacional que sea mayor al que se genera en un *context switch*.
2. Se preve que pueda haber tareas retenidas o bloqueadas por estar esperando a una lectura de disco, por ejemplo. En estos casos, esta tarea se bloquea y otra entra en acción.

- Utilizando código C++ ejemplifique la redefinición de un operador cuyo retorno es por referencia. En el ejemplo dado, analice qué ocurriría si al utilizar dicho operador, el objeto receptor del mensaje se encontrara declarado como constante.

Por ejemplo:

```
Complex& operator=(const Complex& other) {
    if (this != &other) {
```

```
        this->re = other.re;
        this->im = other.im;
    }
    return *this;
}
```

De esta manera este comportamiento es válido:

```
Complex var1(1, 2);
Complex& var2 = var1;
```

Si se hubiese definido el receptor como constante igual compilaría puesto que eso no genera un cambio de comportamiento en el operador asignación:

```
Complex var1(1, 2);
const Complex& var2 = var1;
```

- Elija 2 contenedores provistos por la STL y explique su modo de empleo con una simple secuencia de código. Indique en qué casos resulta conveniente utilizar dichos contenedores.

1. `std::list`

El contenedor **list** de C++ soporta inserción y borrado de elementos desde cualquier parte del contenedor. Generalmente se implementa como una lista doblemente enlazada. Agregar, borrar y mover elementos entre la lista no invalida los iteradores o las referencias.

Por ejemplo:

```
#include <algorithm>
#include <iostream>
#include <list>

int main()
{
    // Create a list containing integers
    std::list<int> l = { 7, 5, 16, 8 };

    l.push_front(25); //Agrega entero al principio de la lista
    l.push_back(13); //Agrega entero al final de la lista

    auto it = std::find(l.begin(), l.end(), 16); //Agrega entero antes del 16
    if (it != l.end()) l.insert(it, 42);

    for (auto i = l.begin(); l != l.end(); l++) { //Itera e imprime los valores de la lista
        std::cout << *i << std::endl;
    }
}
```

Es conveniente utilizar estos contenedores cuando se necesita guardar elementos en un orden específico.

2. `std::map`

El contenedor **map** de C++ guarda elementos formados por una clave y un valor, siguiendo un orden específico. Las *claves* son las que se utilizan para ordenar e identificar unívocamente a los elementos mientras que los *valores* guardan el contenido asociado a cada clave. El orden a seguir está determinado por la función de comparación asociada a la clave. Son implementados como árboles de búsqueda binarios. Se puede acceder a un valor determinado por medio de la clave y el uso de corchetes.

Por ejemplo:

```
std::map<char, int> map;
```

```
map.insert({'a', 1}); //Inserta un par clave-valor ('a', 1);
std::cout << map['a'] << std::endl; //Imprime 1
```

Es conveniente utilizar estos contenedores cuando se necesita asociar un conjunto de información a otro. Además, si se requiriese una velocidad de cómputo baja debido a que el acceso a un valor de una clave es en O(1).

- Escriba un programa que reciba por línea de comandos un Puerto y una IP. El programa debe aceptar una única conexión e imprimir en stdout todo lo recibido. Al recibir el texto 'SALIR' debe finalizar el programa sin imprimir dicho texto.

Ya explicado

- Escriba el archivo .h correspondiente a una biblioteca que exporta:

a) Una función 'sumar' que toma un puntero a una secuencia de números de punto flotante constantes y retorna un número de punto flotante

b) Una función que retorna entero llamada 'ejecutar' que recibe como argumento un puntero a una función que no posee parámetros y retorna entero

c) Una variable de tipo entero largo sin signo llamada 'ErrorCode'

```
float sumar(const float* a);
int ejecutar(int (*F)(void));
unsigned long ErrorCode;
```

- Describa el proceso de transformación de código fuente a un ejecutable. Precise las etapas y las tareas desarrolladas en cada una de ellas.

Ya explicado

- Implemente la función *char* substract(const char* s, unsigned short inicio, unsigned short fin)* que procese la secuencia s y retorne una nueva cadena de caracteres con la copia de s salvo aquellos caracteres comprendidos entre los índices inicio y fin. Considere las condiciones de borde.

```
char* substract(const char* s, unsigned short inicio, unsigned short fin) {
    if (inicio < 0 || fin >= strlen(s) || inicio > fin) {
        return NULL;
    }
    size_t new_len = strlen(s) - (fin - inicio) - 1;
    char* result = (char*) malloc(sizeof(char) * (new_len + 1)); //\0
    size_t pos = 0;
    for (int i = 0; i < strlen(s); i++) {
        if (i >= inicio && i <= fin) continue;
        result[pos] = s[i];
        pos++;
    }
    result[new_len] = '\0';
    return result;
}
```

- Explicar el efecto de anteponer la palabra `static` a un método de una clase. Ejemplifique con un código simple.

Ya explicado

- Escriba una rutina para ambiente gráfico que pinte un triángulo azul con la base de la ventana y que sea isósceles.

Es lo mismo que el rojo que ya fue explicado

Final 22/12/2016

- Explique qué son los métodos const en C++ y para qué sirven. De un breve ejemplo donde su uso sea imprescindible.

Indica que la función es de solo lectura y que no modifica el estado interno del objecto que la llamó. No puede llamar ningún método que no sea constante ni modificar atributos no estáticos. Se agrega el modificador `const()` al final de la declaración y debe incluirse además en la definición. Sirve para detectar errores en el código en tiempo de compilación ya que si un método no modifica el estado interno del estado se utiliza esta keyword para que el compilador falle en caso de que sí se modifique y exista un bug a resolver.

Es imprescindible por ejemplo en el operador== que no debe modificar ningún atributo de los objetos involucrados.

```
bool operator==(const Objeto& other) const;
```

- Explique qué se entiende por "compilación condicional". Ejemplifique mediante código.

Ya explicado

- Defina una rutina que se conecte al puerto 9000 del equipo '192.168.1.100', envíe el carácter 'A' y cierre la conexión.

- Ejemplifique el uso del modificador virtual de C++.

Ya explicado

- Escriba una función C que reciba 1 cadena (T) y 1 entero (N); y retorne una nueva cadena cuyo contenido sea N veces la cadena T utilizando un espacio como separador.

Ya explicado

- Escriba las siguientes definiciones/declaraciones:

a) Una definición de una variable entero largo sin signo, global, no visible fuera del fuente, llamada X

`static unsigned long X = 2;` Si no le ponía valor también funcionaba porque static te pone un valor por defecto (en este caso 0)

b) Una definición de una función llamada F que tome un arreglo de caracteres y devuelva un puntero a un arreglo de enteros cortos sin signo

```
unsigned short* F(char array[]) {
    return NULL;
}
```

c) Una definición de un puntero a un arreglo de 10 enteros sin signo, llamado A

```
unsigned int (*A)[10];
```

- Dado un vector de enteros en C++, escriba un procedimiento que reciba el vector, lo recorra, imprima su contenido en 2 archivos distintos (a.txt y b.txt) en paralelo y libere los recursos. ¿Es necesario realizar un control de concurrencia entre los 2 hilos? ¿Por qué?

No es necesario realizar un control de concurrencia entre los 2 hilos porque trabajan sobre recursos distintos (archivos distintos), por lo que no se necesita una *condition_variable* para manejar la sincronización de los hilos o un *mutex* para asegurar la exclusión mutua.

```
#include <string>
#include <iostream>
#include <fstream>
```

```
class MonitorEscritor {
    std::ofstream file;

public:
    MonitorEscritor(std::string file_name) {
        this->file.open(file_name,
                        std::ofstream::out | std::ofstream::trunc);
    }
    void escribir(int element) {
        file << element;
    }
};

void write_files(std::vector<int>& elementos, std::string file_1, std::string file_2) {
    MonitorEscritor monitor_1(file_1);
    MonitorEscritor monitor_2(file_2);
    std::thread t1([&] () {
        for (auto i = elementos.begin(); i != elementos.end(); i++) {
            monitor_1.escribir(*i);
        }
    });
    std::thread t2([&] () {
        for (auto i = elementos.begin(); i != elementos.end(); i++) {
            monitor_2.escribir(*i);
        }
    });
    t1.join();
    t2.join();
}


int main() {
    std::string file_1 = "a.txt";
    std::string file_2 = "b.txt";
    std::vector<int> elementos = {1, 3, 2, 4};
    write_files(elementos, file_1, file_2);
    return 0;
}
```

- Escribir un programa C que reciba por argumento el nombre de un archivo, y lo procese sobre sí mismo (sin crear archivos intermedios). El procesamiento consiste en eliminar todos los espacios duplicados dejando solamente 1 espacio en su lugar.

Ya explicado

- Declare la clase *Complejo* para encapsular la información necesaria para representar números complejos de doble precisión. Incluya al menos: constructor default, constructor por copia, operador <<, ==, = y +

```
class Complex {
private:
    float re;
    float im;
public:
    Complex();
    Complex(float re, float im);
```

```
    Complex(const Complex& other);
    std::ostream& operator<<(std::ostream& out, const Complex& other);
    bool operator==(const Complex& other) const;
    Complex& operator=(const Complex& other);
    Complex operator+(const Complex& other) const;
}
```

- Escriba una rutina (para ambiente gráfico Windows o Linux) que dibuje la siguiente imagen en su área de dibujo: circulo azul en rectangulo negro que lo contiene.

Ya explicado

Final 15/12/2016

- Describa y ejemplifique el uso de la siguiente instrucción de precompilaciòn: `#define`

La directiva `#define` se utiliza para definir un símbolo. Cada vez que el precompilador encuentre un match a ese símbolo, lo reemplazará por el valor asociado a dicho símbolo. A un símbolo definido por `define` se le puede asociar tanto sentencias de codigo como valores literales que el precompilador hará un reemplazo literal del mismo.

Por ejemplo si se quiere realizar un programa que calcule el área de un círculo, recibiendo el radio por línea de comandos:

```
#define PI 3.14

int main(int argc, const char* argv[]) {
    int radio = atoi(arg[1]);
    float area = PI * radio * radio;
    printf("%f\n", area);
    return 0;
}
```

El compilador encontrará el símbolo PI, lo buscará en la tabla de símbolos y lo reemplazará por el valor 3.14 para efectuar el cálculo del área.

- Escribir un programa ISO C que procese el archivo palabras.txt sobre sí mismo. El proceso consiste en duplicar las palabras que tengan más de 3 vocales distintas.

```
int leer_reversa(FILE* f, int* seek) {
    int previous_seek = *seek;
    fseek(f, *seek, SEEK_END);
    int out = fgetc(f);
    *seek = previous_seek - 1;
    return out;
}

void escribir_reversa(FILE* f, char c, int* seek) {
    int previous_seek = *seek;
    fseek(f, *seek, SEEK_END);
    fputc(c, f);
    *seek = previous_seek - 1;
}

bool detectar_borrado(char* buffer_palabra) {
    int diferentes = 0;
```

```
    bool a = true;
    bool e = true;
    bool i = true;
    bool o = true;
    bool u = true;
    for (int index = 0; index < strlen(buffer_palabra); index++) {
        if (buffer_palabra[index] == 'a' && a) {
            diferentes++;
            a = false;
        }
        if (buffer_palabra[index] == 'e' && e) {
            diferentes++;
            e = false;
        }
        if (buffer_palabra[index] == 'i' && i) {
            diferentes++;
            i = false;
        }
        if (buffer_palabra[index] == 'o' && o) {
            diferentes++;
            o = false;
        }
        if (buffer_palabra[index] == 'u' && u) {
            diferentes++;
            u = false;
        }
    }
    return diferentes > 3;
}

int main(int argc, char* argv[]) {
    int c = 0;
    int c_final = 0;
    int cant_extra = 0;

    int seek_lectura = -1;
    int seek_escritura = -1;

    FILE* f = fopen("a.txt","r");

    char buffer_palabra[1024];
    size_t pos = 0;

    while ((c = fgetc(f)) != EOF) {
        c_final++;
        if (c == ' ') {
            buffer_palabra[pos] = '\0';
            if (detectar_borrado(buffer_palabra)) {
                cant_extra += strlen(buffer_palabra) + 1; //+1 por el espacio
                c_final += strlen(buffer_palabra) + 1; //+1 por el espacio
            }
            //Reset
            pos = 0;
            continue;
```

```
        }
        buffer_palabra[pos] = c;
        pos++;
    }
    if (detectar_borrado(buffer_palabra)) {
        cant_extra += strlen(buffer_palabra);
        c_final += strlen(buffer_palabra);
    }

    memset(buffer_palabra, 0, 1024);
    pos = 0;

    fclose(f);
    truncate("a.txt", sizeof(char) * c_final);

    f = fopen("a.txt","r+");
    seek_lectura -= (cant_extra);

    while (seek_lectura != (-c_final - 1)) {
        c = leer_reversa(f, &seek_lectura);
        escribir_reversa(f, c, &seek_escritura);
        buffer_palabra[pos] = c;
        pos++;
        if (c == ' ' || seek_lectura == -c_final - 1) {
            buffer_palabra[pos] = '\0';
            if (detectar_borrado(buffer_palabra)) {
                for (int i = 0; i < strlen(buffer_palabra); i++) {
                    escribir_reversa(f, buffer_palabra[i], &seek_escritura);
                }
            }
            //Reset
            pos = 0;
        }
    }
    fclose(f);
    return 0;
}
```

- Explique la función listen haciendo referencia a su prototipo/firma y propósito.

Ya explicado

- ¿Qué es un iterador de la librería estándar de C++? Ejemplifique su uso

Un iterador es un objeto que permite al programador recorrer una colección sin que sea necesario que conozca la implementación de la misma. Usualmente se lo utiliza para búsquedas, consultas de elementos, etc. Los contenedores STL se pueden recorrer utilizando iteradores, por ejemplo:

```
std::vector<int> elementos;
for (auto i = elementos.begin(); i != elementos.end(); i++) {
    std::cout << *i << std::endl;
}
```

- Escriba:

a) La definición de un puntero a entero denominado E.

```
int* E = NULL;
```

b) La definición de una función de alcance local al archivo de definición, denominada suma que tome como parámetros 2 números de punto flotante de doble precisión y devuelva un puntero a caracter con el resultado de la suma formateada como una cadena numérica de 2 decimales

```
static char* suma(double a, double b) {
    static char result[3];
    double suma = a + b;
    if (suma < 10) snprintf(result, 3, "0%d", (int)suma);
    else snprintf(result, 3, "%d", (int)suma);
    return result;
}
```

c) La declaración de un puntero a una función denominado F, que tome como parámetros un entero corto con signo y un puntero a puntero a caracter y devuelva un puntero a un entero

```
int* (*F)(short a, char** b);
```

- Escriba una función que cargue un listbox (combo o dropdown) con 10 strings de la forma "opción xx", siendo xx una cadena de 2 dígitos con la forma 01, 02, ..., 10.

```
#include <gtkmm.h>

int main(int argc, char* argv[]) {
    Gtk::Main kit(argc, argv);
    Gtk::ComboBoxText combo;

    Glib::ustring opcion = "opción xx";
    for (int i = 1; i <= 10; i++) {
        Glib::ustring new_opcion = opcion.substr(0, opcion.length() - 2);
        if (i == 10) {
            new_opcion += "10";
        } else {
            new_opcion += ("0" + std::to_string(i));
        }
        combo.append(new_opcion);
    }

    Gtk::VBox vbox;
    vbox.add(combo);
    Gtk::Window v;
    v.add(vbox);
    v.show_all();
    Gtk::Main::run(v);
    return 0;
}
```

- ¿Qué es un parámetro opcional/default en C++? ¿Cómo se utiliza? ¿Dónde puede usarse? Ejemplifique

Un **parámetro opcional** es un parámetro que posee un valor por defecto con la cual si no pasamos un valor para ese parámetro toma el valor por defecto. Una vez que a uno de los parámetros se le asigna un valor por defecto, los parámetros que le siguen en la lista de argumentos deben tener también un valor por defecto. No se puede dejar un hueco en medio de la lista.

Si la declaración del método se hace en el .h y la implementación en el .cpp, el valor por defecto se coloca en el .h

Ejemplo:

```
//Declaraciones en el .h
float foo(float x,  float y = 0);

//Definiciones en el .cpp
float foo(float x, float y =  0) {
    return x + y;
}  //Esta definición NO compila.
float foo(float x, float y) {
    return x + y;
}  //Esta definición compila
```

- Describa el prototipo de los siguientes operadores para que conserven la semántica esperada:

a) **operator==**

```
bool operator==(const Complex& other) const;
```

b) **operator=**

```
Complex& operator=(const Complex& other);
```

c) **operator float**

```
operator float() const
```

d) **operator-**

```
Complex operator-(const Complex& other) const;
```

e) **operator<<**

```
friend std::ostream& operator<<(std::ostream& out, const Complex& other);
```

- ¿En qué consiste el proceso de pre-compilación?

Ya explicado

- Haciendo uso de un arreglo estático de 1000 posiciones, implemente una pila de enteros. La librería debe contemplar su uso en ambientes multithreading. Por consiguiente se espera que incorpore los recursos necesarios para garantizar la consistencia de la estructura en todo momento.

```
class Pila {
    std::mutex m;
    int arreglo[1000];
    int pos_actual = 0;

public:
    void apilar(int elemento) {
        std::unique_lock<std::mutex> lock(m);
        if (pos_actual == 999) {
            throw std::runtime_error("La pila esta llena");
        }
        arreglo[pos_actual] = elemento;
        pos_actual++;
    }

    int desapilar() {
        std::unique_lock<std::mutex> lock(m);
        if (pos_actual == 0) {
            throw std::runtime_error("La pila está vacía");
        }
```

```
        int elemento = arreglo[pos_actual-1];
        pos_actual--;
        return elemento;
    }

    bool esta_vacia() {
        std::unique_lock<std::mutex> lock(m);
        return pos_actual == 0;
    }

    int ver_tope() {
        std::unique_lock<std::mutex> lock(m);
        return arreglo[pos_actual-1];
    }
};
```